Oracle® TimesTen In-Memory Database

Scaleout User's Guide

Release 18.1

E61194-08

July 2020

ORACLE®

Oracle TimesTen In-Memory Database Scaleout User's Guide, Release 18.1

E61194-08

Contents

4 Setting Up a Grid

5 Managing a Database

10 Migrating, Backing Up and Restoring Data

11 Recovering from Failure

A Example for Deploying a Grid and Database

B TimesTen Scaleout Environment

Index

Preface

Oracle TimesTen In-Memory Database (TimesTen) is a relational database that is memory-optimized for fast response and throughput. The database resides entirely in memory at runtime and is persisted to the file system.

- Oracle TimesTen In-Memory Database in classic mode, or TimesTen Classic, refers to single-instance and replicated databases (as in previous releases).
- Oracle TimesTen In-Memory Database in grid mode, or TimesTen Scaleout, refers to a multiple-instance distributed database. TimesTen Scaleout is a grid of interconnected hosts running instances that work together to provide fast access, fault tolerance, and high availability for in-memory data.
- TimesTen alone refers to both classic and grid modes (such as in references to TimesTen utilities, releases, distributions, installations, actions taken by the database, and functionality within the database).
- TimesTen Application-Tier Database Cache, or TimesTen Cache, is an Oracle Database Enterprise Edition option. TimesTen Cache is ideal for caching performance-critical subsets of an Oracle database into cache tables within a TimesTen database for improved response time in the application tier. Cache tables can be read-only or updatable. Applications read and update the cache tables using standard Structured Query Language (SQL) while data synchronization between the TimesTen database and the Oracle database is performed automatically. TimesTen Cache offers all of the functionality and performance of TimesTen Classic, plus the additional functionality for caching Oracle Database tables.
- TimesTen Replication features, available with TimesTen Classic or TimesTen Cache, enable high availability.

TimesTen supports standard application interfaces JDBC, ODBC, and ODP.NET; Oracle interfaces PL/SQL, OCI, and Pro*C/C++; and the TimesTen TTClasses library for C++.

This guide provides:

- Background information to help you understand how TimesTen Scaleout works.
- Step-by-step instructions and examples that show how to perform the most commonly needed tasks to work with TimesTen Scaleout.

Audience

To work with this guide, you should be familiar with TimesTen, SQL (Structured Query Language), and database operations.

Related documents

TimesTen documentation is available at:
`https://docs.oracle.com/database/timesten-18.1`.

Oracle Database documentation is also available on the Oracle documentation website. This may be especially useful for Oracle Database features that TimesTen supports but does not attempt to fully document, such as OCI and Pro*C/C++.

Conventions

TimesTen Classic is supported on multiple platforms. Unless otherwise indicated, the information in this guide applies to all supported platforms. The term Windows refers to all supported Windows platforms. The term UNIX applies to all supported UNIX platforms. The term Linux is used separately.

TimesTen Scaleout is only supported on the Linux platform. The information in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* applies only to the Linux platform.

See the *Oracle TimesTen In-Memory Database Release Notes* (`README.html`) in your installation directory for specific platform versions supported by TimesTen.

> **Note:** In TimesTen documentation, the terms "data store" and "database" are equivalent. Both terms refer to the TimesTen database.

This document uses the following text conventions:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates code, commands, URLs, function names, attribute names, directory names, file names, text that appears on the screen, or text that you enter.
`italic monospace`	Italic monospace type indicates a placeholder or a variable in a code example for which you specify or use a particular value. For example: `LIBS = -Ltimesten_home/install/lib -ltten` Replace *`timesten_home`* with the path to the TimesTen instance home directory.
[ ]	Square brackets indicate that an item in a command line is optional.
{ }	Curly braces indicate that you must choose one of the items separated by a vertical bar (|) in a command line.
|	A vertical bar separates alternative arguments.
. . .	An ellipsis (. . .) after an argument indicates that you may use more than one argument on a single command line.
% or $	The percent sign or dollar sign indicates the UNIX shell prompt, depending on the shell that is used.

Convention	Meaning
#	The number (or pound) sign indicates the prompt for the UNIX root user.

TimesTen documentation uses these variables to identify path, file and user names:

Convention	Meaning
installation_dir	The path that represents the directory where the current release of TimesTen is installed.
timesten_home	The path that represents the home directory of a TimesTen instance.
release or *rr*	The first two parts in a release number, with or without the dot. The first two parts of a release number represent a major TimesTen release. For example, 181 or 18.1 represents TimesTen Release 18.1.
DSN	TimesTen data source name (for the TimesTen database).

Note: TimesTen release numbers are reflected in items such as TimesTen utility output, file names, and directory names, all of which are subject to change with every minor or patch release. The documentation cannot always be up to date. It seeks primarily to show the basic form of output, file names, directory names, and other code that may include release numbers. You can confirm the current release number by looking at *Oracle TimesTen In-Memory Database Release Notes* or executing the `ttVersion` utility.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

What's New

This section summarizes new features and functionality of TimesTen Release 18.1 that are documented in this guide, providing links into the guide for more information.

New features in release 18.1.4.1.0

This release of TimesTen Scaleout adds a new type of backup: staged backups. This type of backup eliminates the overhead of creating local copies of the checkpoint and log files and reduces the network traffic of creating a remote copy in the repository. Staged backups are ideal for when you want to make regular backups on a second site that is independent to your main site. See "Back up a database into a remote repository (WAN-friendly)" on page 10-9 for more information.

To increase the performance of database import operations, the `ttGridAdmin dbImport` command now enables you to use multiple threads to import database objects with the use of the `-numThreads` option. See "Import a database export" on page 10-17 for more information.

If any request to create a channel between elements hangs due to software issues or network failures, then all channel create requests could be blocked. You can set a timeout for how long to wait for a channel create request to a remote element. See "Set a timeout for create channel requests" on page 11-45 for more information.

New features in release 18.1.3.1.0

As always, you can evict a replica set from the distribution map for your grid with the `ttGridAdmin dbDistribute -evict` command. However, now you must make sure that all pending requests for adding or removing elements are applied before requesting the eviction of a replica set. That is, you can no longer combine `ttGridAdmin -add` or `-remove` operations with a request to evict. See "Remove and replace a failed element in a replica set" on page 11-15 for details.

New features in release 18.1.2.1.0

This release of TimesTen Scaleout adds support to importing a client DSN into a TimesTen Client system on Windows. See "Establishing client connections from a TimesTen Client" on page 5-9 for further details.

You can attempt to terminate all user connections to a database with `ttGridAdmin dbDisconnect` command. See "Unloading a database from memory" on page 5-27 for more information.

You can attempt a re-synchronization of your data if the data distribution process is interrupted or fails to complete. Re-synchronization involves executing the `ttGridAdmin dbDistribute -resync` operation. See "Recovering from a data distribution error" on page 11-7 for full details.

New features in release 18.1.1.2.0

You can specify the `TT_DMLCommitOnSuccess` hint to enable or disable a commit operation as part of DML execution. See "Using the TT_CommitDMLOnSuccess hint" on page 7-10 for more information.

1

Overview of TimesTen Scaleout

The following sections describe the features and components of Oracle TimesTen In-Memory Database in grid mode (TimesTen Scaleout).

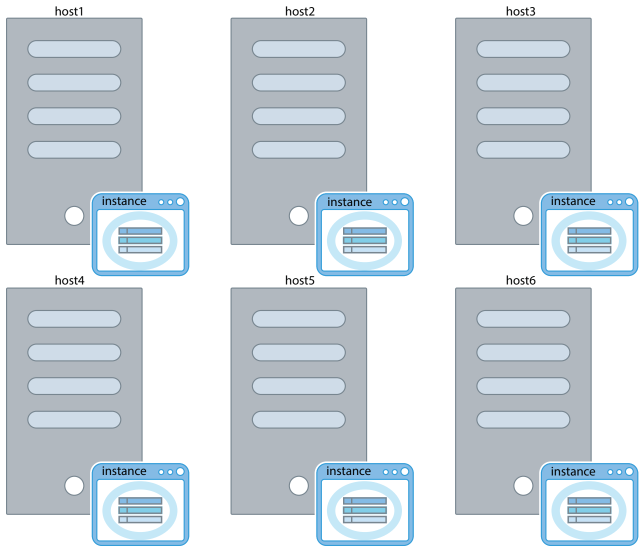
Note: For an overview of Oracle TimesTen In-Memory Database in classic mode (TimesTen Classic), see "Overview for the Oracle TimesTen In-Memory Database" in the *Oracle TimesTen In-Memory Database Introduction*.

Introducing TimesTen Scaleout

TimesTen Scaleout delivers high performance, fault tolerance, and scalability within a highly available in-memory database that provides persistence and recoverability. As shown in Figure 1–1, TimesTen Scaleout delivers these features by distributing the data of a database across a grid of multiple instances running on one or more hosts.

Note: TimesTen Scaleout identifies physical or virtual systems as hosts. Each host represents a different system. You determine the name that TimesTen Scaleout uses as identifier for each host.

Figure 1–1 A grid distributes data across many instances over multiple hosts



TimesTen Scaleout enables you to:

- Create a grid that is a set of interconnected instances installed on one or more hosts.
- Create one or more in-memory, SQL relational, ACID-complaint databases.
- Distribute the data of each database across the instances in the grid in a highly available manner using a shared-nothing architecture.
- Connect applications to your database with full access to all the data, no matter what the distribution of the data is across the database.
- Maintain one or more copies of your data. Your choice to maintain more than one copy protects you from data loss in the event of a single failure.
- Add or remove instances from your grid to:
  - Expand or shrink the storage capacity of your database as necessary.
  - Expand or shrink the computing resources of your database to meet the performance requirements of your applications.

TimesTen Scaleout features

TimesTen Scaleout provides key capabilities, such as:

- In-memory database
- Performance
- Persistence and durability
- SQL and PL/SQL functionality
- Transactions
- Scalability
- Data transparency
- High availability and fault tolerance
- Centralized management

In-memory database

A database in TimesTen is a memory-optimized relational database that empowers applications with the responsiveness and high throughput required by today's enterprises and industries. Databases fit entirely in physical memory (RAM) and provide standard SQL interfaces.

TimesTen is designed with the knowledge that all data resides in memory. As a result, access to data is simpler and more direct resulting in a shorter code path and simpler algorithms and internal data structures. Thus, TimesTen delivers performance by optimizing data residency at run time. By managing data in memory and optimizing data structures and access algorithms accordingly, database operations execute with maximum efficiency, achieving dramatic gains in responsiveness and throughput.

Performance

TimesTen Scaleout achieves high performance by distributing the data of each database across instances in the grid in a shared-nothing architecture. TimesTen Scaleout spreads the work for the database across those instances in parallel, which computes the results of your SQL statements faster.

Persistence and durability

Databases in TimesTen are persistent across power failures and crashes. TimesTen accomplishes this by periodically saving to a file system:

- All data through checkpoint files.
- Changes made by transactions through transaction log files.

In TimesTen Scaleout, the data in your database is distributed into elements. Each element keeps its own checkpoint and transaction log files. As a result, the data stored in each element is independently durable. Each instance in a grid manages one element of a database. In the event of a failure, an instance can automatically recover the data stored in its element from the checkpoint and transaction logs files while the remaining instances continue to service applications.

TimesTen Scaleout also enables you to keep multiple copies of your data to increase durability and fault tolerance.

You can change the durability settings of a database according to your performance and data durability needs. For example, you may choose if data is flushed to the file system with every commit or periodically in batches in order to operate at a higher performance level.

SQL and PL/SQL functionality

Applications use SQL and PL/SQL to access data in a database. Any developer familiar with SQL can be immediately productive developing applications with TimesTen Scaleout.

For more information on SQL, see Chapter 7, "Using SQL in TimesTen Scaleout" in this guide and the *Oracle TimesTen In-Memory Database SQL Reference*. For more information on PL/SQL, see Table 1–9 and the *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide*.

Transactions

TimesTen Scaleout supports transactions that provide atomic, consistent, isolated and durable (ACID) access to data.

For more information, see Chapter 6, "Understanding Distributed Transactions in TimesTen Scaleout" in this guide and "Transaction Management" in the *Oracle TimesTen In-Memory Database Operations Guide*.

Scalability

TimesTen Scaleout enables you to transparently distribute the data of a database across multiple instances, which are located on separate hosts, to dramatically increase availability, performance, storage capacity, processing capacity, and durability. When TimesTen Scaleout distributes the data of your database across multiple instances, it uses the in-memory resources provided by the hosts running those instances.

TimesTen Scaleout enables you to add or remove instances in order to control both performance and the storage capacity of your database. Adding instances expands the memory capacity of your database. It also improves the throughput of your workload by providing the additional computing resources of the hosts running those instances. If your business needs change, then removing instances (and their hosts) enables you to meet your targets with fewer resources.

Data transparency

While TimesTen Scaleout distributes your data across multiple instances, applications do not need to know how data is distributed. When an application connects to any instance in the grid, it has access to all of the data of the database without having to know where specific data is located.

Knowledge about the distribution of data is never required in TimesTen Scaleout, but it can be used to tune the performance of your application. You can use this knowledge to exploit locality where possible. See "Using pseudocolumns" on page 7-9 for more information.

High availability and fault tolerance

TimesTen Scaleout automatically recovers from most transient failures, such as a congested network. TimesTen Scaleout recovers from software failures by recovering from checkpoint and transaction log files. Permanent failures, such as hardware failures, may require intervention by the user.

TimesTen Scaleout provides high availability and fault tolerance when you have multiple copies of data located across separate hosts. TimesTen Scaleout provides a feature called K-safety (`k`) where the value you set for `k` during the creation of the grid defines the number of copies of your data that will exist in the grid. This feature ensures that your database continues to operate in spite of various faults, as long as a single copy of the data is accessible.

- To have only a single copy of the data, set `k` to `1`. This setting is not recommended for production environments.
- To have two copies of the data, set `k` to `2`. A grid can be fault tolerant with this setting. Thus, if one copy fails, another copy of the data exists. Ensure you locate each copy of the data on distinct physical hardware for maximum data safety.

TimesTen Scaleout provides fault tolerance for both software and hardware failures:

- Software failures are often transient. When one copy of the data is unavailable due to a software error, SQL statements are automatically redirected to the other copy of the data (if possible). In the meantime, TimesTen Scaleout synchronizes the data on the failed system with the rest of the database. TimesTen Scaleout does not require any user intervention to recover as long as the instances are still running.
- Hardware failures may eventually require user intervention. In some cases, all that is required is to restart the host.

TimesTen Scaleout provides a membership service to help resolve failures in a consistent manner. The membership service provides a consistent list of instances that are up. This is useful if a network error splits the hosts into two separate groups that cannot communicate with each other.

Centralized management

You do not need to log onto every host within a grid in order to perform management activities. Instead, you conduct all management activity from a single instance using the `ttGridAdmin` utility. The `ttGridAdmin` utility is the main utility you use to define, deploy, and check on the status of each database.

You can also use the `ttGridRollout` utility or the Oracle SQL Developer GUI (both of which use the `ttGridAdmin` utility under the covers to execute all requests) to facilitate creating, deploying, and managing your grid:

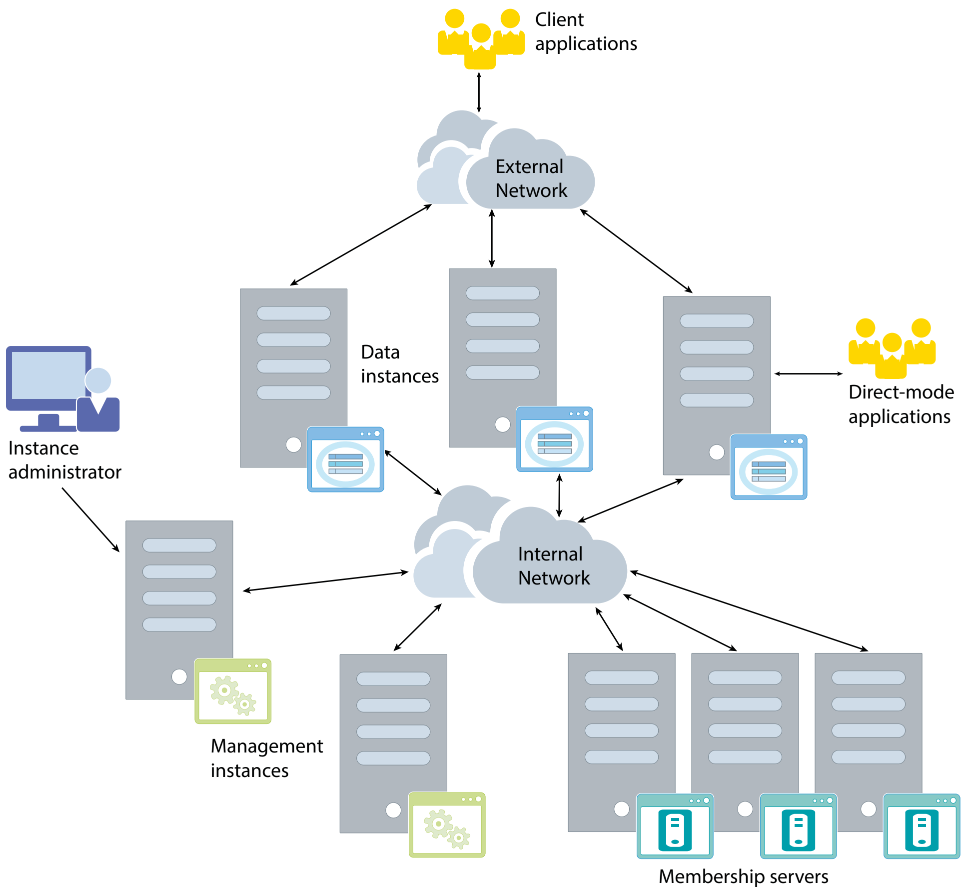
- If you are creating a grid for the first time, you can use the `ttGridRollout` utility to define and deploy your grid. After creation, use either the `ttGridAdmin` utility or Oracle SQL Developer to manage your grid.
- You can create and manage any grid using Oracle SQL Developer, which is a graphical user interface (GUI) tool that gives database developers a convenient way to create, manage, and explore a grid and its components. You can also browse, create, edit, and drop particular database objects; run SQL statements and scripts; manipulate and export data; and view and create reports. See the *Oracle SQL Developer Oracle TimesTen In-Memory Database Support User's Guide* for more information.

TimesTen Scaleout architecture

One OS user creates and manages a grid. This user is called the instance administrator. See "Instance administrator" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide* for details on the instance administrator. TimesTen Scaleout enables the instance administrator to:

- Configure whether the grid creates one or two copies of your data by using K-safety.
- Create one or two management instances through which the grid is managed.
- Create multiple data instances in which data is contained and managed.
- Set up a membership service to track which data instances are operational at any moment. The membership service consists of three or more membership servers.
- Create one or more databases.
- Create one or more repositories to store backups for your databases.

Figure 1–2 Grid structure



A database consists of multiple elements, where each element stores a portion of data from its database. Each data instance contains one element of each database. If you create multiple databases in the grid, then each data instance contains multiple elements (one from each database).

For each database you create, you decide which elements participate in data distribution. Usually, all elements participate, but when you bring online new data instances, you decide when the elements of those new data instances begin to participate in database operations. You need to explicitly add elements into the distribution map of database for them to participate in database operations. Likewise, you need to remove elements from the distribution map (which stops them from participating in database operations) before you can remove their data instances from the grid.

Upon including an element into the distribution map, each element of a database is automatically placed into a replica set. Each replica set contains the same number of elements as the value set for K-safety. Elements in the same replica set hold the same data.

The following sections provide a more detailed description of these components and their responsibilities within a grid:

- Instances
- Installations
- K-safety
- Data distribution

- Backups
- Internal and external networks

Instances

A grid uses instances to manage, contain, and distribute one or more copies of your data. An instance is a running copy of the TimesTen software. When you create an instance on a host, you associate it with a TimesTen installation. An installation can be used by a single instance or shared by multiple instances. Each instance normally resides on its own host to provide maximum data availability and as a safeguard against data loss should one host fail.

Each instance has an associated instance administrator (who created the instance) and an instance home. The instance home is the location for the instance on your host. The same instance administrator manages all instances in the grid.

TimesTen Scaleout supports two types of instances:

- Management instances
- Data instances

Management instances

Management instances control a grid and maintain the model, which is the central configuration of a grid. To ensure that the administrator can easily control a grid, all management activity is executed through a single management instance using the `ttGridAdmin` utility.

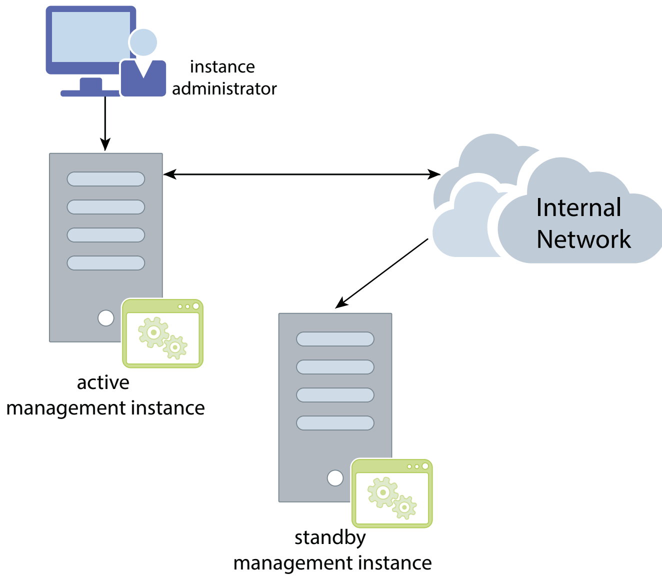
> **Note:** See "Central configuration of the grid" on page 1-16 for more details on the model.

TimesTen Scaleout enables you to create two management instances to provide for high availability and guard against a single management instance failure that could impede grid management. Consider having two management instances a best practice for a production environment. Once created, TimesTen Scaleout configures both management instances in an active standby configuration. You always execute all management operations through the active management instance. The standby management instance exists purely as a safeguard against failure of the active management instance.

If you create two management instances, as shown in Figure 1–3, then all information used by the active management instance is automatically replicated to the standby management instance. Thus, if the active management instance fails, you can promote the standby management instance to become the new active management instance through which you continue to manage the grid.

> **Note:** See "Managing failover for the management instances" on page 11-33 for details on how TimesTen Scaleout replicates information for the management instances.

Figure 1–3 Administrator manages the grid through management instances



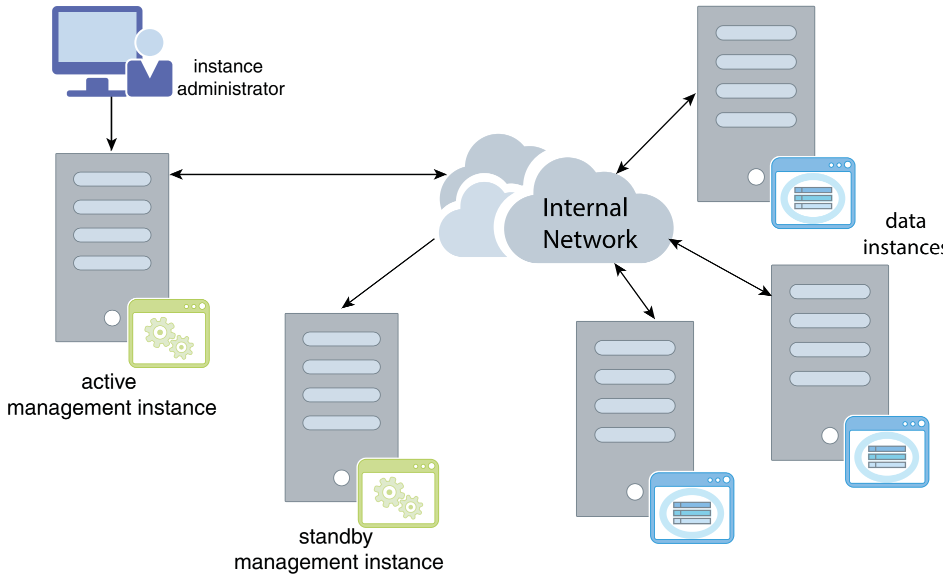
Consider that:

- You can manage a grid through a single management instance without a standby management instance. However, it is not recommended for production environments.
- If both management instances fail, databases in the grid continue to operate. Some management operations are unavailable until you restart at least one of the management instances.

Data instances

Data instances store one element per database in the grid. Data instances execute SQL statements and PL/SQL blocks. A grid distributes the data within each database across data instances. You manage all data instances through the active management instance, as shown in Figure 1–4.

Figure 1–4 Management instances manage a grid of multiple data instances



Installations

Instances need an installation of a TimesTen distribution to operate. An installation is read-only and can be used locally or shared across multiple instances. You create the installation of the initial management instance by extracting a TimesTen distribution on any given location on the system defined as the host of the management instance. TimesTen Scaleout can locally create any subsequent installation on the rest of the hosts in the grid and associate the new installations with the instances run by those hosts. All instances that run on the same host may share the same installation.

As long as an installation can be accessed by multiple hosts that installation can be shared by instances in those hosts. However, sharing an installation on a shared file server, such as NFS, between multiple instances on separate hosts may reduce availability. If the shared network storage or the network connecting all of the hosts to the NFS server fails or has performance issues then all instances sharing that installation are impacted. Thus, while sharing an installation on a shared file server across instances may be a valid option for a development environment, you may want to evaluate whether this is advisable for a production environment.

K-safety

You configure your grid to create either single or multiple copies of the data of each database within your grid. TimesTen Scaleout uses its implementation of K-safety (`k`) to manage one or multiple copies of your data. You specify the number of copies you want of your data by setting `k` to `1` or `2` when you create the grid.

You improve data availability and fault tolerance when you specify that the grid creates two copies of data located across separate hosts.

- If you set `k` to `1`, TimesTen Scaleout stores a single copy of the data (which is not recommended for production environments).

  When `k` is set to `1`, then the following may occur if one or more elements fail:

  – Any data contained in the element is unavailable until the element recovers.

  – Any data contained in the element is lost if the element does not recover.

Even though there is only a single copy of the data, the data is still distributed across separate elements to increase capacity and data accessibility.

- If you set `k` to `2`, then TimesTen Scaleout stores two copies of the data. A grid can tolerate multiple faults when you have two copies of the data.

  If one element fails, the second copy of the data is accessed to provide the requested data. K-safety enables availability to your data as long as one of the copies of the data is available. Where possible, locate each copy of the data on distinct physical hardware for maximum data safety.

The following sections describe how multiple copies are managed and organized.

- Understanding replica sets
- Understanding data spaces
- Assigning hosts to data space groups

Understanding replica sets

Each element of a database is automatically placed into a replica set depending on the value of `k`, where:

- If you set `k` to `1`, then each replica set contains a single element.
- If you set `k` to `2`, then each replica set contains two elements (where each element is an exact copy of the other element in the replica set).

Thus, each replica set contains the same number of elements as the value set for `k`.

When `k` is set to `2`, any change made to the data in one element is also made to the other element in the replica set to keep the data consistent on both elements in the replica set at all times. Because of the transparency capabilities of TimesTen Scaleout, you can initiate transactions on any element, even if the requested data is not contained in that element or if the requested data spans multiple replica sets. If an element fails, then the other element in the replica set is accessed to provide the requested data. All data in the database is available as long as one element in each replica set is functioning.

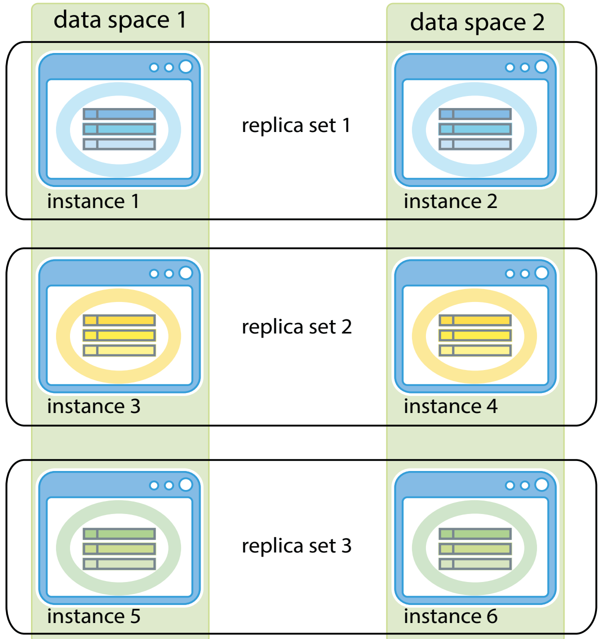
Understanding data spaces

Each database consists of a set of elements, where each element stores a portion of data from its database. The grid organizes the elements for each database into data spaces.

Each database consists of either one or two data spaces. When `k` is set to `2`, the elements within each replica set are assigned to separate data spaces.

Figure 1–5 shows how two copies of the data are organized within two data spaces, where each data space contains the elements that make up a single copy of the data of the database. One copy is contained within data space 1 and the second copy is contained in data space 2. There are three replica sets and the elements of each replica set are assigned to a separate data space. Thus, each element in data space 1 is identical to its partner element in data space 2.

Figure 1–5 Two copies, each in own data space



As your needs grow or diminish, you may add or remove replica sets to a grid. When you add data instances, the grid automatically creates elements for each database. However, the data is not automatically redistributed when you add or remove a data instance. You decide when it is appropriate to assign an element to a replica set and redistribute the data across all the elements in each data space.

Assigning hosts to data space groups

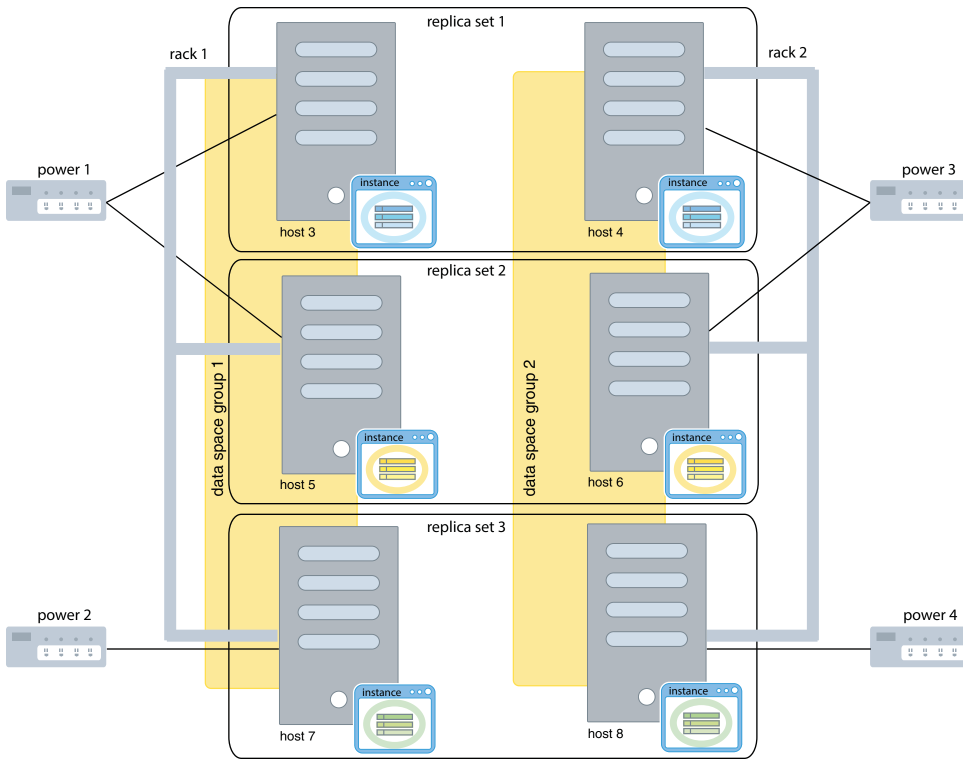
You decide how the data is physically located by assigning hosts into data space groups that represents the physical organization of your grid. As discussed in "Understanding data spaces" on page 1-10, copies of the data are organized logically into data spaces. Each data space should use separate physical resources. Shared physical resources can include similar racks, the same power supply, or the same storage. Be aware that if all elements in a single replica set are stored on hosts that share a physical component, then data stored in that replica set becomes unavailable if that shared physical component fails.

TimesTen Scaleout requires you to assign all hosts that will run data instances into data space groups. When using K-safety, there are *k* copies of the data and the same number of data space groups (which are numbered from 1 to *k*). You should assign hosts that share the same physical resources into the same data space group. The elements in data instances running on hosts that are assigned to the same data space group are in the same data space. Each data space contains a full copy of all data in the database.

If you ensure that the hosts in one data space group do not share physically resources with the hosts in another data space group, then hosts in separate data space groups are less likely to fail simultaneously. This scenario makes it likely that all data in the database is available, even if a single hardware failure takes down multiple hosts. For example, you may ensure that all of the hosts in one data space group are plugged into a power supply that is separate from the power supply for the hosts in another data space group. If that is the case, pulling one plug does not power down both hosts in a single replica set, thus making some data unavailable.

Figure 1–6 shows a grid configured where `k` is set to `2`, so the grid contains two data space groups. There are two racks, each with two power sources and three hosts. Three hosts have been assigned to each data space group. TimesTen Scaleout creates replica sets such that such that one element in each replica set is in each data space group.

Figure 1–6 Hosts organized into data space groups



The process for assigning hosts to a data space group includes deciding how you will physically separate the hosts supporting the data spaces.

- If the physical topology of your grid is simple, you can assign each host yourself to each data space group.
- If the physical topology of your grid is complex, TimesTen Scaleout can recommend an appropriate data space group for each host based on the physical resources shared among those hosts by the use of physical groups.

Describe your physical topology with physical groups If you have a complex configuration where analyzing the physical dependencies make it difficult to assign multiple hosts to separate data space groups, you can ask TimesTen Scaleout to recommend how to assign your hosts to data space groups. In order to do this, TimesTen Scaleout needs to know the physical topology of where your hosts are co-located or those that use the same resources. You can describe the physical topology of your grid where each host is identified with its physical dependencies using physical groups. The physical group informs TimesTen Scaleout what hosts share the same physical resources. Hosts grouped into the same physical group are likely to fail together.

Note: Associating your host within a physical groups is optional.

For example, multiple hosts can be grouped if they:

- Use the same power supply.
- Reside in the same rack consisting of several shelves.
- Reside on the same shelf.

Once all of your hosts are assigned to physical groups, TimesTen Scaleout can suggest which hosts should be assigned to each data space group to minimize your risk. See "Description of the physical topography of the grid" on page 4-20 for more details on assigning hosts to a physical group.

Data distribution

You can create one or more databases within a grid. Each database is independent, with separate users, schemas, tables, persistence, and data distribution. TimesTen Scaleout manages the distribution of the data according to the defined distribution map and the distribution scheme for each table.

- Defining the distribution map for a database
- Defining the distribution scheme for tables

Defining the distribution map for a database

You decide on the number of data instances in a grid, which dictates the maximum number of elements and replica sets for any one database. Each data instance hosts one element of each database in the grid. Thus, the data instances in a grid can manage one or more databases simultaneously. If you create multiple databases in the grid, then each data instance will contain multiple elements (one element from each database).

Each database consists of multiple replica sets, where each replica set stores a portion of data from its database. You define which elements of the available data instances store data of the database with a distribution map. Once the distribution map is defined and applied, TimesTen Scaleout automatically assigns each element to a replica set and distributes the data to its corresponding replica set, where each element communicates with other elements of different replica sets to provide a single database image. The details of how data is distributed may vary for each table of a database based on the distribution scheme of the table.

> **Note:** TimesTen Scaleout stores the composition of the distribution map, or how every data instance associates with each other, in a partition table that is managed by the `ttGridAdmin` utility.

Once you add the elements of the data instances that will manage and contain the data of each database to the distribution map, you can explicitly request that the data be distributed across the resulting replica sets.

As the needs of your business change, you can increase the capacity of a database by increasing the number of replica sets in the grid. To can accomplish this by:

1. Adding new hosts to the grid. The number of hosts you add must be proportional to the number of replica sets you want to add and the value of K-safety. For example, if you want to add another replica set to a database in a grid with `k` set to `2`, you need to add a host for data space group `1` and another for data space group `2`.
2. Creating an installation to support data instances on each new host.
3. Creating a data instance on each new host.

4. Adding the elements of the new data instances to the distribution map of each database you want to increase its capacity. TimesTen Scaleout automatically creates new replica sets as appropriate.
5. Redistributing the data across all replica sets.

   When you add new data instances or remove existing data instances to the grid, the grid does not automatically re-distribute the data stored in the database across the replica sets of those new or remaining data instances. Instead, you decide when is the appropriate time to re-distribute the data across the existing data instances. Redistribution can negatively impact your operational database. You should redistribute in small increments to minimize the impact. The larger the number of data instances that you have, the less of an impact it is to incrementally add or remove a single replica set.

If you need to replace a data instance with a new data instance in the same data space group, this action does not require a redistribution of all data.

To reduce your capacity, remove the data instances that manage a replica set from the distribution map and redistribute the data across the remaining data instances in the grid.

Defining the distribution scheme for tables

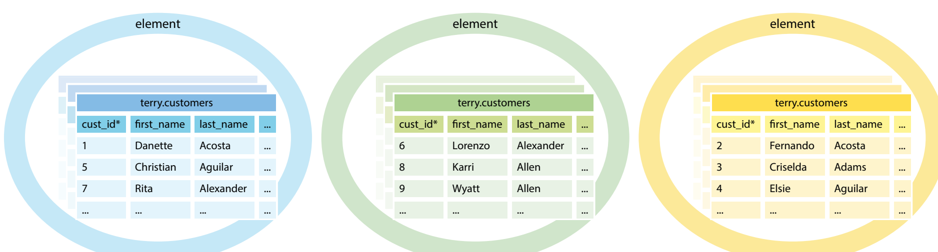
TimesTen Scaleout distributes the data in a database across replica sets. All tables in a database are present in every replica set. You define the distribution scheme for each table in a database in the `CREATE TABLE` statement. The distribution scheme describes how the rows of the table are distributed across the grid.

How the data is distributed is defined by one of the following distribution schemes specified during table creation.

- Hash: The data is distributed based on the hash of the primary key or a composite of multiple columns that are specified by the user. A given row is present in a replica set chosen by the grid. Rows are evenly distributed across the replica sets. This is the default method as it is appropriate for most tables.

  See Figure 1–7 for an example of a table, `terry.customers`, with a hash distribution scheme. Each element belongs to a different replica set.

Figure 1–7 Table with hash distribution



- Reference: Distributes the data of a child table based on the location of the parent table that is identified by the foreign key. That is, a given row of a child table is present in the same replica set as its parent table. This distribution scheme optimizes the performance of joins by distributing related data within a single replica set. Thus, this distribution scheme is appropriate for tables that are logically children of a single parent table as parent and child tables are often referenced together in queries.

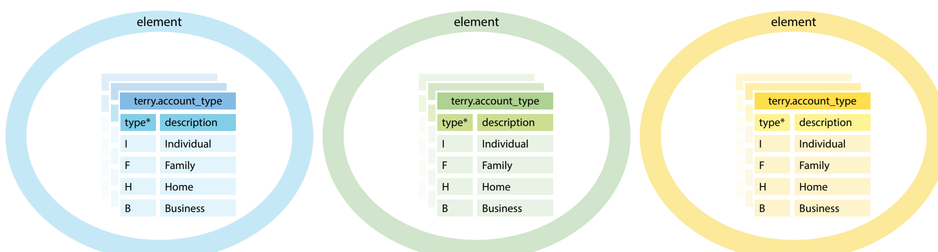
See Figure 1–8 for an example of a child table, `accounts`, with a reference distribution scheme to a parent table, `customers`. Each element belongs to a different replica set.

Figure 1–8 Table with reference distribution

- Duplicate: Distributes full identical copies of data to all the elements of a database. That is, all rows are present in every element. This distribution scheme optimizes the performance of reads by storing identical data in every data instance. This distribution scheme is appropriate for tables that are relatively small, frequently read, and infrequently modified.

  See Figure 1–9 for an example of a table, `account_type`, with a duplicate distribution scheme. Each element belongs to a different replica set.

Figure 1–9 Table with duplicate distribution



Backups

TimesTen Scaleout enables you to create backups of the databases in your grid and restore them to the same grid or another grid with a similar topology. TimesTen Scaleout also enables you to export your databases to a grid with a different topology. You define a repository as a location for your database backups, exports, and collections of log files. Multiple grids may use the same repository.

Internal and external networks

For most production environments, TimesTen Scaleout requires a single private internal network and at least one external network.

- Internal network: Instances in a grid communicate with each other over a single internal network using the TCP protocol. In addition, instances communicate with membership servers through this network. Membership servers use this network to communicate among themselves.

- External networks: Applications use the external network to connect to data instances to access a database. Applications do not need external network access to management instances or membership servers.

See "Network requirements" on page 2-8 for more information.

Central configuration of the grid

TimesTen Scaleout maintains a single central configuration of the grid. This configuration is called the model. The model represents the logical topology of a grid. The model contains a set of objects that represent components of a grid, such as installations, hosts, database definitions, and instances.

You can have several different versions of the model. Each time you apply changes to the model, the grid saves the model as a version. Only one version of the model can be active in the grid at any given time.

- The latest model is the model within which you are making changes, but has not yet been applied. If you are in the process of modifying a model, then this version describes a future desired structure of a grid that only becomes the current model when you apply it.
- The current version of the model (the model that was most recently applied) always describes the current structure of the grid.
- Previous model versions describe what the grid structure used to be.

Perform the following when creating a desired structure for your grid:

1. You design the desired structure of your grid by adding or removing grid components (such as installations, hosts, and instances) to the latest model.
2. Once you complete the desired structure of a model, you apply the model to cause these changes to take effect. This version of the model becomes the current version of the model.
3. After you apply the model, TimesTen Scaleout attempts to implement the current model in the operational grid.

   It is not guaranteed that all components of the current model are running. For example, if your grid has 10 hosts configured, but only 6 of them are running at the moment, the definition of all 10 is still in the model.

Every time you use the `ttGridAdmin` utility to add a grid component, such as an installation, host or instance, you add model objects corresponding to these grid components to the model. Each model object specifies the attributes and relationships of the grid component.

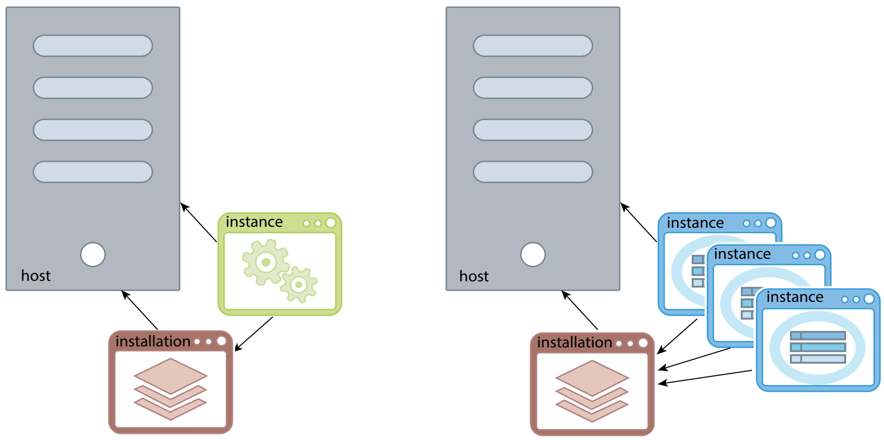
Some model objects have relationships to other model objects. Figure 1–10 shows how the relationship is stored between model objects. That is, the host, installation and instances have a relationship where:

- The installation model object points to the host model object on which it is installed.
- Both the management instance model object and the data instance model object point to an installation model object of the installation that the instance will use and a host model object on which the instance is installed.

Figure 1–10 shows two different types of relationships between the hosts, installation, and instances that is stored within the model.

- You install a single installation on a host with one data instance, where the data instance points to the installation and to the host on which it exists.
- You create multiple data instances on a single host where they all share a single installation. Each data instance points to the same host and the same installation. The installation points to the host on which it is installed. To increase availability, avoid using multiple data instances on a single host.

Figure 1–10 Example of a model



Any time you add or remove model objects from the model, these changes do not immediately impact a grid until you explicitly apply these changes. After you apply the changes, TimesTen Scaleout implements the current model into the operational grid. For example, if you add a new installation model object and data instance model object to the latest version of the model, applying the changes to the model performs all of the necessary operations to create and initialize both the installation and the data instance in that host.

Planning your grid

Before you configure a grid and database in TimesTen Scaleout, gather the information necessary for creating a grid:

- Determine the number of hosts and membership servers
- Define the network parameters of each host and membership server
- Define the locations for the installation directory and instance home of each instance
- Ensure you have all the information you need to deploy a grid

Determine the number of hosts and membership servers

You need to determine how many hosts and membership servers you are going to use based on these considerations:

- *Membership servers*: In a production environment, you need an odd number of membership servers greater than or equal to three to ensure a majority quorum in case one or more membership servers fail. You should ensure that:

- Each membership server uses independent physical resources (such as power, network nodes, and storage) from each other.
- Membership servers do not run on the same system as hosts with data instances.

- *Management instances*: You need two management instances to ensure some measure of availability to the configuration and management capabilities of your grid. Ensure that hosts with management instances use independent physical resources (such as power, network nodes, and storage) from each other.
- *Data instances*: You determine the number of hosts you require for data instances based on the level of K-safety and the number of replica sets. For example, if you set `k` to `2` and you decide to have three replica sets, you need six data instances.

  Also, the level of K-safety determines how many data space groups or independent physical locations you must have for your hosts. Ensure that the hosts with data instances assigned to data space group `1` use independent physical resources than hosts with data instances that are assigned to data space group `2`.

Figure 1–11 shows an example of a setup of three membership servers, one repository, two management instances, and six data instances. The example co-locates a membership server with repository for a total of 11 hosts.

Figure 1–11 Example of a grid

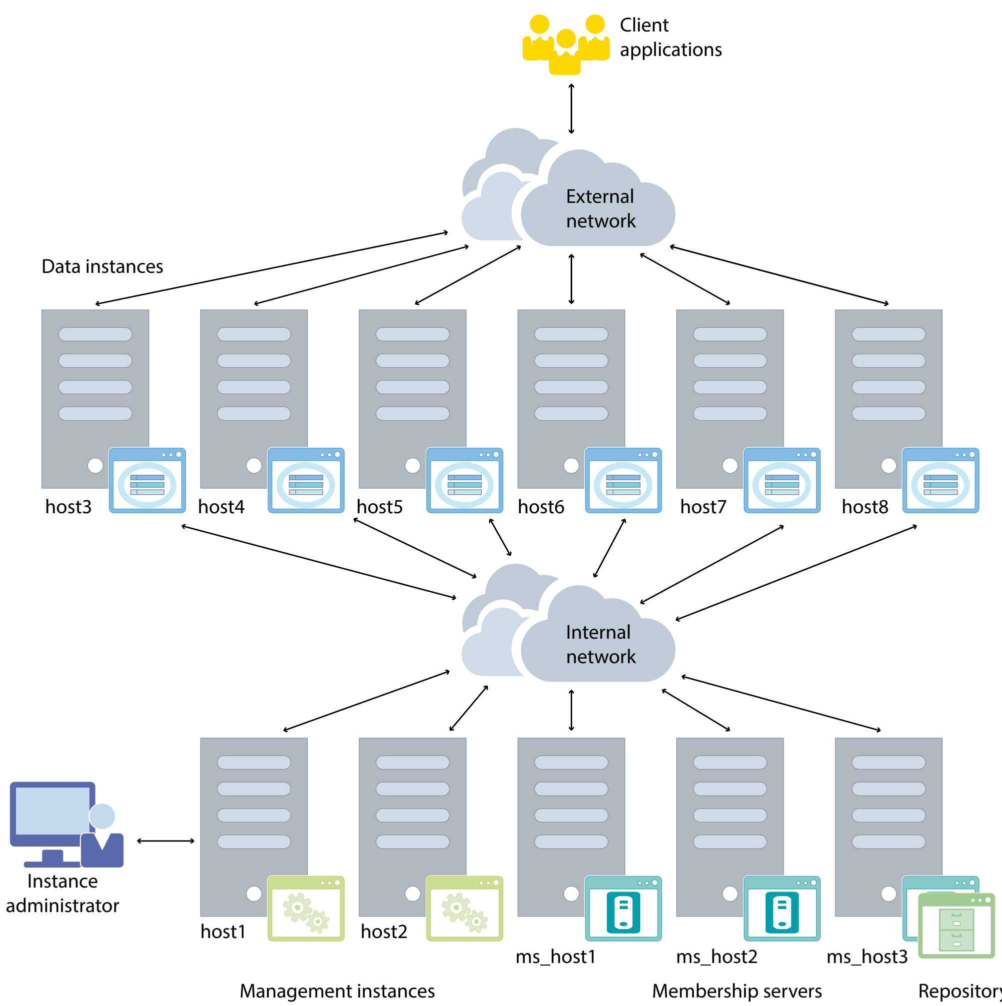
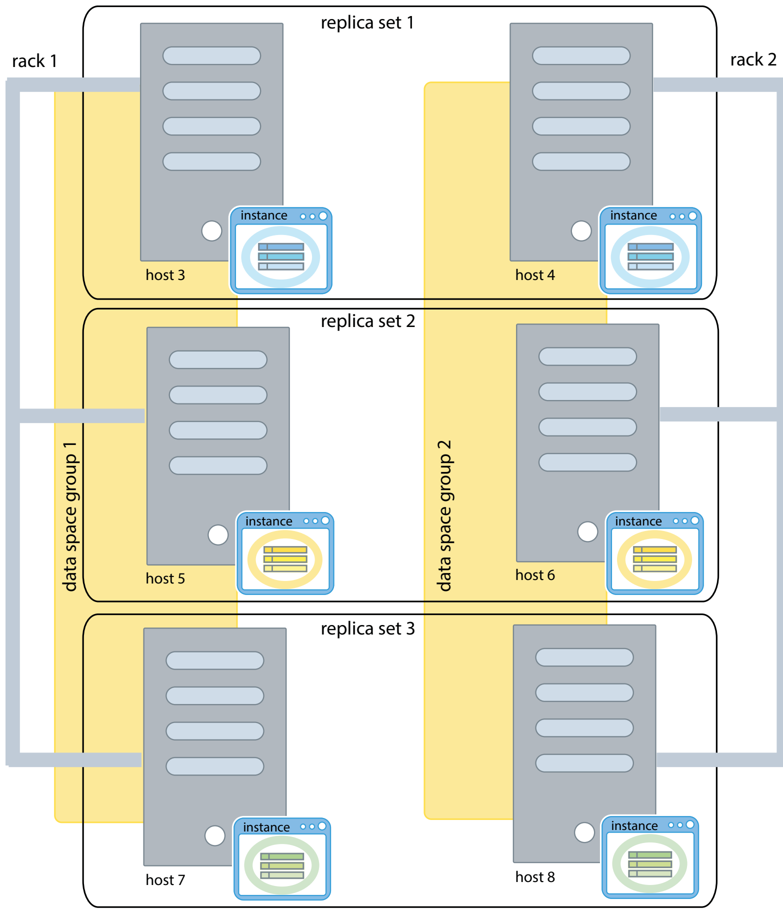


Figure 1–12 shows how the hosts with data instances in this example are organized into two data space groups for a grid with `k` set to `2`. The hosts of each data space group share a rack.

Figure 1–12 Example of hosts organized into data space groups



See Table 1–1 for an example of how you might assign the hosts with data instances into data space groups based on the physical resources they share.

Table 1–1 Systems and their roles

Host name	Membership server	Repository	Management instance	Data instance	Data space group	Physical resources
ms_host1	Yes	Yes	N/A	N/A	N/A	Rack 1
ms_host2	Yes	N/A	N/A	N/A	N/A	Rack 2
ms_host3	Yes	N/A	N/A	N/A	N/A	Rack 2
host1	N/A	N/A	Yes	N/A	N/A	Rack 1
host2	N/A	N/A	Yes	N/A	N/A	Rack 2
host3	N/A	N/A	N/A	Yes	1	Rack 1
host4	N/A	N/A	N/A	Yes	2	Rack 2
host5	N/A	N/A	N/A	Yes	1	Rack 1

Table 1–1 (Cont.) Systems and their roles

Host name	Membership server	Repository	Management instance	Data instance	Data space group	Physical resources
host6	N/A	N/A	N/A	Yes	2	Rack 2
host7	N/A	N/A	N/A	Yes	1	Rack 1
host8	N/A	N/A	N/A	Yes	2	Rack 2

Define the network parameters of each host and membership server

Ensure that you know the network addresses and TCP/IP ports that you expect each host and membership server to use. See "Network requirements" on page 2-8 for details on using internal and external networks in your grid.

See Table 1–2 for an example of the internal and external addresses of the topology described in Table 1–1.

Table 1–2 Internal and external addresses

Host name	Internal address	External address
ms_host1	`ms-host1`	N/A
ms_host2	`ms-host2`	N/A
ms_host3	`ms-host3`	N/A
host1	`int-host1`	N/A
host2	`int-host2`	N/A
host3	`int-host3`	`ext-host3.example.com`
host4	`int-host4`	`ext-host4.example.com`
host5	`int-host5`	`ext-host5.example.com`
host6	`int-host6`	`ext-host6.example.com`
host7	`int-host7`	`ext-host7.example.com`
host8	`int-host8`	`ext-host8.example.com`

Note: All systems must be part of the same private network. It is recommended that you create an external network for applications outside of your private network to connect to your database.

You need to consider which TCP/IP ports each instance will use, especially if your setup is behind a firewall. You must define the TCP/IP ports for the following:

- *Membership servers*: You must define three port numbers (client, peer, and leader) for each membership server. See Table 3–1, " zoo.cfg configuration parameters" for details on these port numbers.
- *Management instances*: There are three port numbers (daemon, server, and management) for each management instance. TimesTen Scaleout sets the default values for the daemon, server, and management ports if you do not specify them.
- *Data instances*: There are two port numbers (daemon and server) for each data instance. TimesTen Scaleout sets the default values for the daemon and server ports if you do not specify them.

If a firewall is in place, you must open all the ports mentioned above plus the local ephemeral ports for the internal network, except the server ports assigned to each instance. The server ports assigned to each instance must be open for the external network.

See Table 1–3 for an example of the TCP/IP ports assigned to each membership server or instance. The example uses the default values for each port.

Table 1–3 TCP/IP ports

Host name	Membership server (client/peer/leader)	Management instance (daemon/server/management)	Data instance (daemon/server)
ms_host1	2181 / 2888 / 3888	N/A	N/A
ms_host2	2181 / 2888 / 3888	N/A	N/A
ms_host3	2181 / 2888 / 3888	N/A	N/A
host1	N/A	6624 / 6625 / 3754	N/A
host2	N/A	6624 / 6625 / 3754	N/A
host3	N/A	N/A	6624 / 6625
host4	N/A	N/A	6624 / 6625
host5	N/A	N/A	6624 / 6625
host6	N/A	N/A	6624 / 6625
host7	N/A	N/A	6624 / 6625
host8	N/A	N/A	6624 / 6625

Define the locations for the installation directory and instance home of each instance

You must define the locations for the installation directory and the instance home that you expect your grid to use. Defining the locations for these grid objects includes defining the name TimesTen Scaleout uses to identify them. Consider these while defining these locations:

- In the case of the instance home, TimesTen Scaleout adds the instance name to the defined location. For example, if you define `/grid` as the location for an instance named `instance1`, the full path for the instance home of that instance becomes `/grid/instance1`.
- A similar behavior applies for installation objects. Instead of adding the installation name, TimesTen Scaleout adds the release version to the defined location. For example, if you define `/grid` as the location of the installation, the full path for the installation becomes `/grid/tt18.1.4.1.0`.

TimesTen Scaleout creates the locations you define for the installation directory and instance home if they do not exist already.

See Table 1–4 for an example of the locations for the membership server installation. You must create these locations on their respective systems prior to installing the membership server.

Table 1–4 installation of the membership servers

Host name	Installation location
ms_host1	`/grid/membership`
ms_host2	`/grid/membership`

Table 1–4 (Cont.) installation of the membership servers

Host name	Installation location
ms_host3	`/grid/membership`

See Table 1–5 for an example of the installation directory and instance home locations for the management instances.

Table 1–5 installation directory and instance home of the management instances

Host name	Installation name	Installation directory	Instance name	Instance home
host1	installation1	`/grid/tt18.1.4.1.0`	instance1	`/grid/instance1`
host2	installation1	`/grid/tt18.1.4.1.0`	instance1	`/grid/instance1`

See Table 1–6 for an example of the installation directory and instance home locations for the data instances.

Table 1–6 installation directory and instance home of the data instances

Host name	Installation name	Installation directory	Instance name	Instance home
host3	installation1	`/grid/tt18.1.4.1.0`	instance1	`/grid/instance1`
host4	installation1	`/grid/tt18.1.4.1.0`	instance1	`/grid/instance1`
host5	installation1	`/grid/tt18.1.4.1.0`	instance1	`/grid/instance1`
host6	installation1	`/grid/tt18.1.4.1.0`	instance1	`/grid/instance1`
host7	installation1	`/grid/tt18.1.4.1.0`	instance1	`/grid/instance1`
host8	installation1	`/grid/tt18.1.4.1.0`	instance1	`/grid/instance1`

See Table 1–7 for an example of the location for the repository.

Table 1–7 Repository

Host name	Repository location
ms_host1	`/grid/repository`

Ensure you have all the information you need to deploy a grid

To verify that you have all the information you need before you start deploying your grid, answer the questionnaire provided in Table 1–8.

Table 1–8 Questionnaire

Question	Source of information
What will your K-safety setting be?	"K-safety" on page 1-9
How many membership servers will you have?	"Determine the number of hosts and membership servers" on page 1-17 and Chapter 3, "Setting Up the Membership Service"
How many management instances will you have?	"Management instances" on page 1-7
How many replica sets will you have?	"Data instances" on page 1-8 and "Understanding replica sets" on page 1-10

Table 1–8 (Cont.) Questionnaire

Question	Source of information
Where will you store your database backups?	"Backups" on page 1-15 and "Define the locations for the installation directory and instance home of each instance" on page 1-21
How many hosts are you going to use for your grid?	"Determine the number of hosts and membership servers" on page 1-17
Which of those hosts are going to run management instances?	"Management instances" on page 1-7
Which of those hosts are going to run data instances?	"Data instances" on page 1-8
What will be the data space group assignments of each host with a data instance?	"Assigning hosts to data space groups" on page 1-11
How will you organize your hosts and membership servers across independent physical resources?	"Assigning hosts to data space groups" on page 1-11
Will you use a single network or separate internal and external networks for your grid?	"Internal and external networks" on page 1-15
What is the DNS name or IP address of each host and membership server?	"Define the network parameters of each host and membership server" on page 1-20
Which TCP/IP ports will you use for each instance?	"Define the network parameters of each host and membership server" on page 1-20
What will be the location for the installation files of each membership server?	"Define the locations for the installation directory and instance home of each instance" on page 1-21
What will be the locations for the installation directory and instance home of each instance?	"Define the locations for the installation directory and instance home of each instance" on page 1-21

Database connections

You can access a database either with a direct connection from a data instance or a client/server connection over an external network.

- *Direct connection*: An application connects directly to a data instance of a database that they specify.

  An application using a direct connection runs on the same system as the database. A direct connection provides extremely fast performance as no inter-process communication (IPC) of any kind is required. However, if the specified data instance is down, the connection is not forwarded to another data instance and an error is returned.

- *Client/server connection*: An application using a client/server connection may run on a data instance or on any host with access to the external network. Client applications are automatically connected to a working data instance.

  All exchanges between client and server are sent over a TCP/IP connection. If the client and server reside on separate hosts in the internal network, they communicate by using sockets and TCP/IP.

  If a data instance fails, TimesTen Scaleout automatically re-connects to another working data instance. You can configure options to control this process, if necessary.

Note: If desired, you can specify that a client/server connection connects to a specific data instance.

If your workload only requests data from the local element, then a direct connection is the best method for your application as this provides faster access than a client/server connection. However, if your workload entails that your application may need to switch between data instances for whichever data instance is readily available and retrieves data from the multiple elements, then a client/server connection may provide better throughput.

Comparison between TimesTen Scaleout and TimesTen Classic

The term TimesTen alone, without TimesTen Scaleout or Classic, typically applies to both single-instance and multiple-instance, such as in references to TimesTen utilities, releases, distributions, installations, actions taken by the database, and functionality within the database.

- TimesTen Scaleout refers to TimesTen In-Memory Database in grid mode. TimesTen Scaleout is a multiple-instance environment that contains distributed databases.
- TimesTen Classic refers to TimesTen In-Memory Database in classic mode. Classic mode is a single-instance environment and databases as in previous releases.
  - The Oracle TimesTen Application-Tier Database Cache (TimesTen Cache) product combines the responsiveness of the TimesTen Classic with the ability to cache subsets of an Oracle database for improved response time in the application tier.

TimesTen Scaleout supports and includes most of the features of TimesTen Classic; it does not support any of the features of the TimesTen Cache. The following list describes what features are not supported in TimesTen Scaleout from both of the previous products:

Note: For more information about TimesTen Classic features, see the *Oracle TimesTen In-Memory Database Operations Guide*.

Table 1–9 TimesTen Classic features that are unsupported in TimesTen Scaleout

TimesTen Classic feature	Supported in TimesTen Scaleout (Y/N)	Description
Cache Connect option for caching data from the Oracle database	N	None of the features documented in the *Oracle TimesTen Application-Tier Database Cache User's Guide* are supported for TimesTen Scaleout. However, TimesTen Scaleout provides facilities for loading data from an Oracle Database.
Replication: both the active standby pair and classic replication schemes	N	Data protection and fault tolerance can be provided through the K-safety feature of TimesTen Scaleout. Thus, none of the features documented in the *Oracle TimesTen In-Memory Database Replication Guide* are supported for TimesTen Scaleout. See "K-safety" on page 1-9 for more details.
Bitmap indexes	N	
LOB support	N	TimesTen Scaleout does not support LOB columns in tables.
Column-based compression	N	Column-based compression within tables

Table 1–9 (Cont.) TimesTen Classic features that are unsupported in TimesTen Scaleout

TimesTen Classic feature	Supported in TimesTen Scaleout (Y/N)	Description
Aging policy for tables	N	
RAM policy	N	TimesTen Scaleout supports the manually loading and unloading of the database through the `ttGridAdmin` utility by system administrators.
X/Open XA standard and the Java Transaction API (JTA)	N	
TimesTen Classic Transaction Log API (XLA) and the JMS/XLA Java API	N	
Oracle Clusterware	N	
Index Advisor	N	
Online upgrade	N	
PL/SQL	Y	While PL/SQL anonymous blocks are fully supported, user-created stored procedures, packages and functions are not supported. You can, however, call procedures and invoke functions from TimesTen-provided packages from your anonymous blocks. TimesTen Scaleout does not support SQL statements that create, alter or drop functions, packages, procedures.
SQL statements	Y	TimesTen Scaleout does not support: ■ `MERGE` ■ Since the Cache Connect feature, active standby pair replication scheme, and classic replication schemes are not supported, neither are the data definition language (DDL) statements that create these objects. TimesTen Scaleout partially supports: ■ `ROWID` data type The semantics of `ROWID` are different in TimesTen Classic than in TimesTen Scaleout. For details, see Chapter 7, "Using SQL in TimesTen Scaleout" in this guide and "ROWID data type" in the *Oracle TimesTen In-Memory Database SQL Reference*. ■ `CREATE [ASYNCHRONOUS] MATERIALIZED VIEW` The `CREATE MATERIALIZED VIEW` statement is supported in a limited capacity. See "CREATE MATERIALIZED VIEW" in the *Oracle TimesTen In-Memory Database SQL Reference* for details. ■ Global temporary tables do not support any form of distribution. When you create a global temporary table, you cannot use any of the `DISTRIBUTE BY` clauses. Global temporary tables are materialized only in the element where the connection is established.

How supported TimesTen features are documented in this book

Throughout the *Oracle TimesTen In-Memory Database Scaleout User's Guide*, the TimesTen Classic features that are included within TimesTen Scaleout are documented as follows:

- If the feature is supported completely as it is within TimesTen Classic, this book provides a small section describing the feature with a cross-reference to the description in other TimesTen books, such as the *Oracle TimesTen In-Memory Database Operations Guide*, *Oracle TimesTen In-Memory Database SQL Reference* and *Oracle TimesTen In-Memory Database Reference*.
- If the feature is used as a base with additional support provided for the unique requirements of TimesTen Scaleout, then the new addition is described and a cross-reference link is provided to the feature in other TimesTen books, such as the *Oracle TimesTen In-Memory Database Operations Guide*, *Oracle TimesTen In-Memory Database SQL Reference* and *Oracle TimesTen In-Memory Database Reference*.
- If the feature is not supported, no cross-reference is provided in this book.

2

Prerequisites and Installation of TimesTen Scaleout

This chapter focuses on the prerequisites needed to successfully deploy TimesTen Scaleout. These sections discuss the requirements for each host used in the grid:

- General prerequisites
- Operating system prerequisites
- Network requirements
- Installing TimesTen Scaleout
- Setting passwordless SSH

General prerequisites

TimesTen Scaleout is only supported on the Linux platform. For the supported Linux platform versions, see the "Platforms and configurations" section in the documentation library. For the most recent information about your particular TimesTen release, see the *Oracle TimesTen In-Memory Database Release Notes* (`README.html`) in your installation directory.

Perform these steps on all hosts that will run the management and data instances and membership servers for the grid:

- Install the same operating system version and release on each host.
- Configure all hosts in the same internal network.

  When you set up your network, you must create a single internal network for all the grid components to communicate with each other. While clients may use the same internal network to connect to instances, you may wish to create an external network for client connections.
- Install and configure NTP (Network Time Protocol). Clocks must be synced.
- Ensure all instances in the grid can communicate with all other instances in the grid over the internal network on any port.
- To avoid problems before and after installation, confirm your file system has sufficient space. (See "Storage provisioning for TimesTen" in *Oracle TimesTen In-Memory Database Operations Guide* for more information.)

Operating system prerequisites

These sections discuss the operating system prerequisites:

- Understanding the TimesTen users group and the operating system user
- Configuring the operating system kernel parameters

Understanding the TimesTen users group and the operating system user

These sections describe and show how to create both the TimesTen users group and the operating system user (which will serve as the instance administrator):

- The TimesTen users group
- The operating system user
- Create the TimesTen users group and the OS user

The TimesTen users group

TimesTen restricts access to the installation and the instances created from that installation to members of a single operating system group. This group, called the *TimesTen users group*, owns the installation and the instances created from the installation. Create this group (for example, `timesten`) and add the desired operating system users prior to installation. Once you create the TimesTen users group, you cannot change the name of the group or the group ID. See "Create the TimesTen users group and the OS user" on page 2-3 for more information.

Note that:

- The instance administrator's primary group must be the TimesTen users group.
- Users who wish to access databases through TimesTen utilities or direct mode applications must be members of the TimesTen users group. This group can be the user's primary or secondary group.
- Users connecting to a database through a client connection do not have to be members of the TimesTen users group.

The operating system user

The instance administrator for all instances in your grid is the operating system user who creates the active management instance. This user then becomes the instance administrator of all other instances in TimesTen Scaleout, including the second management instance and all data instances.

Note that:

- The instance administrator cannot be the root user.
- The instance administrator configures the grid, creates and manages the databases in the grid, starts and stops the databases in the grid, performs all management activities, and performs backup and restore operations.
- You cannot change the instance administrator after that administrator creates the active management instance.
- The instance administrator is a member of the TimesTen users group. See "The TimesTen users group" on page 2-2 for more information.
- The instance administrator's user name and `UID`, and the group name and the group id (`GID`) of the TimesTen users group must be the same on all hosts in the grid, including the hosts on which the management and data instances exist, as well as any of the SCP repository hosts.
- The installation and the instances must have the same owner (the instance administrator).

Create the TimesTen users group and the OS user

In this example, `instanceadmin` is the name of the operating system user and `timesten` is the name of the TimesTen users group.

1. Create the TimesTen users group. Name the group `timesten` with group ID `10000`.

   ```
   # sudo groupadd -g 10000 timesten
   ```

2. Create the `instanceadmin` user with UID `55000` and assign this user to the `timesten` primary group. Then, create a password for the `instanceadmin` user.

   ```
   # sudo useradd -u 55000 -g timesten instanceadmin
   # sudo passwd instanceadmin
   ```

Configuring the operating system kernel parameters

You must configure kernel parameters on the hosts that run instances.

For hosts that run data instances:

- Configure shmmax and shmall
- Configure HugePages
- Modify the memlock settings
- Set the semaphore values

For hosts that run management instances:

- Configure shmmax and shmall
- Modify the memlock settings (optional)
- Set the semaphore values

Configure shmmax and shmall

A database in TimesTen Scaleout consists of elements, where each element stores a portion of data from the database. Each element resides in a shared memory segment. On Linux, shared memory segments consists of pages, where the default page size is normally 4 kB (4,096 bytes). You can verify the default page size by running the `getconf PAGESIZE` command:

```
% getconf PAGESIZE
4096
```

Configure these shared memory kernel parameters to control the size of the shared memory segment:

- `shmmax`: The maximum size of a single shared memory segment expressed in bytes. The value must be large enough to accommodate the size of the total shared memory segment for the element.
- `shmall`: The total size of all shared memory segments system wide. The value is expressed in multiples of the page size (4 kB) and must be greater or equal to the value of `shmmax`. It is recommended that you set the value of `shmall` to less than or equal to the total amount of physical RAM. To display the total amount of physical memory, run the Linux `cat /proc/meminfo` command.

The size of the element is based on the values of the `PermSize`, `TempSize`, `LogBufMB` and `Connections` connection attributes. The element sizing formula is:

```
PermSize + TempSize + LogBufMB + 1 + (0.042 * Connections)
```

The PermSize, TempSize, and LogBufMB values are expressed in MB (megabytes). PermSize, TempSize, LogBufMB, and Connections are connection attributes that you define in your database definition file. If you do not define values for these attributes, TimesTen uses the default values. See "PermSize," "TempSize," and "LogBufMB" in the *Oracle TimesTen In-Memory Database Reference* for details on each connection attribute.

See "Determining the value of the PermSize attribute" on page 5-19 for information on determining the PermSize value. See "Modifying the connection attributes of a database" on page 5-30 for information on modifying the PermSize or TempSize attribute.

As an example, assume each element has a PermSize value of 32 GB (32,768 MB), a TempSize value of 4 GB (4,096 MB), a LogBufMB value of 1 GB (1,024 MB) and a Connections value of 2,048. Applying the sizing formula, the size of each element is:

```
37975 MB (32768 MB + 4096 MB + 1024 MB + 1 + (0.042 MB * 2048))
```

Notes: For hosts that will run management instances, size shmmax and shmall based on a shared memory segment size of at least 400 MB. You can increase the settings of shmmax and shmall if there are other applications that require them to be greater.

The shmmax and shmall values must be the same on each of the hosts that will run data instances. Similarly, the values must be the same on each host that will run management instances.

In this example, to size shmmax and shmall:

1. As the root user, edit the /etc/sysctl.conf file, modifying kernel.shmmax and kernel.shmall. Assuming the size of the element is 37,975 MB and the shmmax and shmall values must be greater than the size of the element, for this example, set shmmax to 48 GB (51,539,607,552 bytes) and shmall to 56 GB (60,129,542,144 bytes = 58,720,256 kB /4 kB page size = 14,680,064 kB pages).

   ```
   # sudo vi /etc/sysctl.conf
   ...
   kernel.shmmax=51539607552
   kernel.shmall=14680064
   ```

2. To reload the settings from the modified /etc/sysctl.conf file:

   ```
   # sudo /sbin/sysctl -p
   ```

3. Run the Linux ipcs lm command to display the current shmmax and shmall settings. The max seg size (kbytes) is the shmmax value and the max total shared memory (kbytes) is the shmall value. The shmmax value expressed in kB is 50,331,658 (51,539,607,552 bytes) and the shmall value expressed in kB is 58,720,256 (60,129,542,144 bytes).

   ```
   % ipcs -lm

   ------ Shared Memory Limits --------
   max number of segments = 4096
   max seg size (kbytes) = 50331648
   max total shared memory (kbytes) = 58720256
   min seg size (bytes) = 1
   ```

Note: The settings for `shmmax` and `shmall` in these examples can be increased if there are other applications that require them to be greater.

Configure HugePages

You can configure `HugePages` for more efficient memory management. For hosts that will run management instances, there is no requirement to configure `HugePages`. For hosts that will run data instances, if the element's shared memory segment is greater than 256 GB, you must configure `HugePages`. Once configured, the memory allocated for `HugePages` is taken from the total RAM on the Linux system and is not available for any other use. In addition, the `HugePages` memory segment is automatically locked and cannot be swapped to the file system.

To configure `HugePages`, you need to know:

- The maximum size of the shared memory segment for the element
- The `HugePages` page size on your Linux system
- The group ID of the instance administrator

Using the examples in the "Configure shmmax and shmall" on page 2-3 section, where the size of the element is 37,975 MB and the `shmmax` value is 48 GB and the "Create the TimesTen users group and the OS user" on page 2-3 section, where the group ID of the `instanceadmin` user is 10000:

- The size of the total shared memory segment is 48 GB.
- The `HugePages` page size is 2,048 kB. (This value is fixed for each platform and is not configurable.)

  To determine the HugePages page size, run the Linux `cat /proc/meminfo|grep Hugepagesize` command:

  ```
  % cat /proc/meminfo | grep Hugepagesize
  Hugepagesize:       2048 kB
  ```

- The group ID is 10,000.

  To determine the group ID of the instance administrator, log in as the `instanceadmin` user, and run the Linux `id` command:

  ```
  % id
  uid=55000(instanceadmin) gid=10000(g10000)groups=10000(g10000)
  ```

To configure `HugePages`:

1. Determine the number of `HugePages` by dividing the size of the total shared memory segment (expressed in MB) by the value of `Hugepagesize` (expressed in MB). In this example, the total shared memory segment for the element is 48 GB (49,152 MB) and the `Hugepagesize` value is 2,048 kB (2 MB):

   ```
   49152 MB/ 2 MB = 24576
   ```

2. As the root user, edit the `/etc/sysctl.conf` file, and set `vm.nr_hugepages` to the number of `HugePages` (24,576 in the example) and set `vm.hugetlb_shm_group` to the group ID of the instance administrator (10,000 in the example). The latter setting restricts access to `HugePages` to members of the group.

   ```
   # sudo vi /etc/sysctl.conf
   ...
   ```

```
...
vm.nr_hugepages=24576
vm.hugetlb_shm_group=10000
```

3. Reload the settings from the modified `/etc/sysctl.conf` file:

```
# sudo /sbin/sysctl -p
```

4. To verify that you have configured `HugePages` correctly, run the Linux `cat/proc/meminfo|grep HugePages` command and verify the value for `HugePages_Total` is 24,576 and the value for `HugePages_Free` is 24,576.

```
% cat /proc/meminfo|grep HugePages
HugePages_Total:   24576
HugePages_Free:    24576
...
```

Notes:

- For hosts that will run data instances, `HugePages` for these hosts must be the same.
- Because `HugePages` must be allocated in contiguous available memory space, the requested allocation may not be granted, or may be only partially granted, until after the system is restarted. Check the `HugePages_Total` and `HugePages_Free` values from `/proc/meminfo`. Restarting grants the full allocation, assuming enough memory is available in the system.
- If a database less than or equal to 256 GB does not fit into the available `HugePages` space, regular pages are used. If a database greater than 256 GB does not fit into the `HugePages` space, the database cannot be loaded into memory.
- The TimesTen PL/SQL shared memory segment consumes some of the configured HugePages allocation, determined by the value of the `PLSQL_MEMORY_SIZE` connection attribute. See "PLSQL_MEMORY_SIZE" in the *Oracle TimesTen In-Memory Database Reference* for more information.
- On Linux, the `HugePages` segment is automatically locked such that the memory segment is not a candidate to be swapped to the file system. Therefore, if you configure `HugePages`, you do not need to set the `MemoryLock` connection attribute.

Modify the memlock settings

The `memlock` entries in the `/etc/security/limits.conf` file control the amount of memory a user can lock. These entries are set at the system level and are different than the `MemoryLock` connection attribute setting. For hosts that will run management instances, setting the `memlock` settings is optional. For hosts that will run data instances, set the `hard memlock` and `soft memlock` entries (expressed in kB) to the size of the shared memory segment for each element. If `HugePages` are configured, the `memlock` values must be large enough to accommodate the size of the shared memory segment or the element will not be loaded into memory.

For example, for the `instanceadmin` user, assuming a total shared memory segment size of 48 GB (49,152 MB), set the `memlock` entries to 50,331,648 kB (49,152 * 1,024):

1. As the `root` user, edit the `/etc/security/limits.conf` file, and set the `memlock` entries to 50,331,648 kB for the `instanceadmin` user. This value indicates the total amount of memory the `instanceadmin` user can lock.

   ```
   # sudo vi /etc/security/limits.conf
   ...
   ...
   instanceadmin soft   memlock 50331648
   instanceadmin hard   memlock 50331648
   ```

2. As the `instanceadmin` user, log out and log in again for the changes to take effect.

   Notes: For hosts that will run data instances, the `memlock` settings for these hosts must be the same. Similarly, if you set `memlock` settings for hosts that will run management instances, the `memlock` settings for these hosts must be the same.

Set the semaphore values

TimesTen has an upper bound on the maximum number of connections to the database. The database connections consist of:

- User connections: established by user applications
- System connections: established internally by TimesTen (set at 48 connections)

The number of user connections is the sum of all user connections across all elements of the grid, not just the user connections to the local grid element. For example, if the grid will support 10,000 concurrent applications, each host running the data instance must be configured to support the 10,000 connections (plus the system connections).

Each user and system connection (a database connection) is assigned one semaphore, such that the total semaphores for a database are:

```
Total semaphores = user connections (N) + system connections (48) +
                   other required connections (107)

Total semaphores = N + 155
```

The semaphore settings are located in the `kernel.sem` configuration directive in `/etc/sysctl.conf`:

```
kernel.sem = SEMMSL SEMMNS SEMOPM SEMMNI
```

where:

- `SEMMSL` is the maximum number of semaphores per array. Configure this value to 155 plus the number of connections.
- `SEMMNS` is the maximum number of semaphores system wide. Use the formula `SEMMNS` = (`SEMMNI * SEMMSL`) as a guideline.
- `SEMOPM` is the maximum number of operations for each `semop` call.
- `SEMMNI` is the maximum number of arrays.

Follow these steps to configure the `SEMMSL` and `SEMMNS` settings (Ensure that the user is `root`):

1. View the existing kernel parameter settings:

   ```
   # /sbin/sysctl -a | grep kernel.sem
   kernel.sem = 2500 320000 1000 1280
   ```

2. Edit the `/etc/sysctl.conf` file, changing `semmsl` (the first value in `kernel.sem`) to 155 plus the number of connections. For hosts that will run management instances, the number of connections is 400. For hosts that will run data instances, the number of connections is not fixed. In this example, to support up to 3,845 connections, set the `semmsl` value to 4,000 (155 + 3,845). Change `semmns` (the second value in `kernel.sem`) to 400,000.

```
# sudo vi /etc/sysctl.conf
...
...
kernel.sem = 4000 400000 2000 2560
```

3. Reload the settings from the modified `/etc/sysctl.conf` file:

```
# sudo /sbin/sysctl -p
```

> **Note:** For hosts that will run data instances, the semaphore values for these hosts must be the same. Similarly, for hosts that will run management instances, the semaphore values for these hosts must be the same.

Network requirements

For most production environments, TimesTen Scaleout requires a single private internal network and at least one external network. This section describes the requirements for those networks.

- Internal network
- External network

Internal network

Instances in a grid communicate with each other over a single internal network using the TCP protocol. TimesTen Scaleout uses this network to perform all SQL, backup, and management operations required by the grid and its databases. In addition, instances communicate with membership servers through this network. Membership servers use this network to communicate among themselves.

Ensure that your internal network has these characteristics:

- High bandwidth. The faster the network the better, in terms of throughput (gigabits per second). For production environments, ensure at minimum a 10 Gigabit Ethernet network or equivalent.
- Low latency. To reduce network latency (time to transmit a message from one host to another) to a minimum, the hosts and membership servers attached to your internal network should either:
  - Span a single data center within a small number of racks.
  - Span multiple data centers within a small geographic region (city or suburb) connected by a metropolitan area network (MAN). Only recommended with a 10 GbE network or better.
  - Not span multiple data regions (states or provinces) connected by a wide area network (WAN).
- IPv4 or IPv6 addresses.
- No network address translation (NAT).

- No TCP packet filtering.

For an on-premises environment, ensure your internal network meets these requirements:

- If your internal network consists of a single network segment, all hosts are connected to a single Ethernet switch or equivalent.
- If your internal network consists of multiple network segments, those segments are connected through bridges instead of IP routers.
- If your internal network uses a MAN, ensure that the MAN can provide the required bandwidth and latency for your workload.

Syntax for internal addresses

When you define a host for your grid, you must specify a single value for the internal address of that host. Optionally, you specify a value for that the external address of that host. The value you specify for the internal address of a host can be either an IPv4 address, an IPv6 address or a name that resolves into one or more IPv4 or IPv6 addresses. For example:

- A dot-decimal IPv4 address such as `192.168.1.1`
- A colon-hexadecimal IPv6 address such as `2606:fe80::f816:3eff:fe15:44b3`
- A name specified in the `/etc/hosts` file such as `host1`
- A name defined in a private Domain Name Server (DNS) such as `int-host1.example.com`

If you use a name to define the internal address of a host:

- If the name resolves to multiple IP addresses, those addresses must be on the same network segment.
- Every host in the grid must be able to resolve a name to the same addresses. For example, if you use the hosts file to define a name, then the hosts file on each host in the grid must contain identical entries for that name.

External network

A grid may optionally use one or more public external networks. These networks enable applications running on machines that are not part of the grid to create client/server connections to databases in the grid. You cannot perform any grid or database management operations through an external network.

While the performance of an external network is important, it is less important than the performance of the internal network. If the internal network performs poorly or unreliably, the grid and its databases may perform poorly or unreliably for all users. Conversely, if an external network performs poorly or unreliably, it may only affect the applications connected to the databases in the grid through that network. As a result, there are fewer requirements for an external network than for the internal network.

Your external networks should have these characteristics:

- Bandwidth based on the requirements of your client/server applications.
- Latency based on the requirements of your client/server applications.
- IPv4 or IPv6 addresses.
- TCP connectivity to the server port of each data instance.
- Any combination of network technologies (VPN, routers, LAN, WAN, etcetera).

If your grid uses a single external network, then the value you specify for the external address of a host can be in any of the forms described in the "Syntax for internal addresses" on page 2-9 section. If your grid uses multiple external networks, then you must use a name to define the external address of a host. The name must resolve to at least one IP address for each external network you use.

Installing TimesTen Scaleout

When you unpack the TimesTen distribution on a host, you create an installation (that is read only). Do not add, alter, or remove files or directories within the installation, unless you are deleting the installation.

The installation may be a full installation or a client-only installation. A client-only installation supports the client use of TimesTen:

Type	Description
Full installation	Use the TimesTen full distribution for this type of installation (for example, `timesten181410.server.linux8664.zip`).
Client-only installation	You can connect and access databases in TimesTen Classic through a client. Use the full installation (for example, `timesten181410.server.linux8664.zip`) to unpack the distribution and then specify `ttInstanceCreate -clientonly`. See "Database connections" on page 1-23 for details on the types of clients and how they connect to a database.

The operating system user that you designated as the instance administrator creates the installation by:

1. Downloading the TimesTen distribution on the host that will contain the active management instance. The distribution is a ZIP file where the ZIP file name indicates the platform, release number, and the type of distribution. For example, `timesten181410.server.linux8664.zip`.
2. Unpacking the ZIP file to create a TimesTen installation. The installation includes the binaries and the support files from which you can create a grid (and all of its components), membership servers, and clients

Only the first installation is created manually by the instance administrator on the host containing the active management instance. Additional installations used by additional instances are created by TimesTen Scaleout utilities. See "Configure your grid" on page 4-1 for information on when to create additional installations for additional instances.

After you download the distribution, follow these steps:

1. Log in as the instance administrator to the host that will contain the initial management instance. In this example, `instanceadmin` is the name of the instance administrator. You can verify the instance administrator with the Linux `id` command.
2. Create the desired directory for the installation such as `/grid/installation1`.

   ```
   % mkdir -p /grid/installation1
   ```

3. Extract and unpack the distribution file into the directory. This example unpacks the installation using the `unzip` command:

   ```
   % unzip /timesten181410.server.linux8664.zip -d /grid/installation1
   [...UNZIP OUTPUT...]
   ```

The top level directory of the installation is the TimesTen release. For example, the directory created under `/grid/installation1` is:

```
dr-xr-x--- 19 instanceadmin timesten 4096 Mar  2 22:07 tt18.1.4.1.0
```

Verifying the installation

These sections provide details on how to verify your installation:

- Run the ttInstallationCheck utility
- Review the installation directory and subdirectories

Run the ttInstallationCheck utility

The `ttInstallationCheck` utility, located in the *`installation_dir`*`/tt18.1.4.1.0/bin` directory, verifies the success or failure of the installation. This utility generates an error if the checksum value for the installation differs from the original checksum value. Checksum values are different if there are any of these changes to the installation directory or files:

- Contents of a file
- Name of a file
- Addition of a file to a directory
- Removal of file from a directory
- Changes to the permissions of a file or directory

In this example, the installation is verified:

```
% ttInstallationCheck
This installation has been verified.
```

In this example, permissions on a file were changed, and `ttInstallationCheck` generates an error:

```
% ttInstallationCheck
Cannot validate the installation in /grid/installation1/tt18.1.4.1.0.
```

See "ttInstallationCheck" in the *Oracle TimesTen In-Memory Database Reference* for detailed information on the `ttInstallationCheck` utility.

Review the installation directory and subdirectories

A TimesTen full installation includes these subdirectories located under the top-level *`installation_dir`*`/tt18.1.4.1.0` directory.

- `3rdparty`: Includes resources for:
  - Apache ZooKeeper
  - Java Message Service (JMS)
- `bin`: TimesTen utilities and executables
- `grid`: Files and resources for TimesTen Scaleout
- `include`: TimesTen include files, among them `timesten.h` (for TimesTen ODBC features) and `tt_errCode.h` (for information about TimesTen error codes)
- `lib`: TimesTen libraries

- `plsql`: Files and resources for TimesTen PL/SQL
- `ttoracle_home`: Oracle Database Instant Client files and resources, for OCI, Pro*C/C++, and ODP.NET

Note: A client-only installation does not include the `3rdparty` or the `grid` directories.

Setting passwordless SSH

The instance administrator must be able to use SSH to log without a password to all hosts within a grid for the management instances and `ttGridAdmin` utility to be able set up and manage the grid and all its members.

Specifically, all hosts with management instances need passwordless SSH access for the instance administrator to all hosts with instances and repositories. Also, hosts with data instances need passwordless SSH access for the instance administrator to all hosts with repositories.

The `ttGridAdmin gridSshConfig` command is able to set for the current user the required passwordless SSH access. Ensure that you execute the command with the user you intend for instance administrator.

Before setting up a grid, you can run the `ttGridAdmin gridSshConfig` command while providing the addresses or DNS names that you will later use to host management instances, data instance, and repositories. When prompted, enter the OS password of the user executing the command. The user and password must already be set on all systems and be identical. See "Understanding the TimesTen users group and the operating system user" on page 2-2 for more information on the instance administrator.

```
% grid/installation1/tt18.1.4.1.0/bin/ttGridAdmin gridSshConfig
 -mgmtAddress int-host1 int-host2
 -dataAddress int-host3 int-host4 int-host5 int-host6 int-host7 int-host8
Enter password:
Setup ssh configuration on local system...............................OK
Setup ssh configuration on int-host1..................................OK
Setup ssh configuration on int-host2..................................OK
Setup ssh configuration on int-host3..................................OK
Setup ssh configuration on int-host4..................................OK
Setup ssh configuration on int-host5..................................OK
Setup ssh configuration on int-host6..................................OK
Setup ssh configuration on int-host7..................................OK
Setup ssh configuration on int-host8..................................OK
Setup passwordless ssh from local system to int-host1.................OK
Setup passwordless ssh from local system to int-host2.................OK
Setup passwordless ssh from local system to int-host3.................OK
Setup passwordless ssh from local system to int-host4.................OK
Setup passwordless ssh from local system to int-host5.................OK
Setup passwordless ssh from local system to int-host6.................OK
Setup passwordless ssh from local system to int-host7.................OK
Setup passwordless ssh from local system to int-host8.................OK
Setup passwordless ssh from int-host1 to int-host1....................OK
Setup passwordless ssh from int-host1 to int-host2....................OK
Setup passwordless ssh from int-host1 to int-host3....................OK
Setup passwordless ssh from int-host1 to int-host4....................OK
Setup passwordless ssh from int-host1 to int-host5....................OK
Setup passwordless ssh from int-host1 to int-host6....................OK
Setup passwordless ssh from int-host1 to int-host7....................OK
```

```
Setup passwordless ssh from int-host1 to int-host8....................OK
Setup passwordless ssh from int-host2 to int-host1....................OK
Setup passwordless ssh from int-host2 to int-host2....................OK
Setup passwordless ssh from int-host2 to int-host3....................OK
Setup passwordless ssh from int-host2 to int-host4....................OK
Setup passwordless ssh from int-host2 to int-host5....................OK
Setup passwordless ssh from int-host2 to int-host6....................OK
Setup passwordless ssh from int-host2 to int-host7....................OK
Setup passwordless ssh from int-host2 to int-host8....................OK

Passwordless ssh working between hosts:

From\To   int-host1 int-host2 int-host3 int-host4 int-host5 ... int-host8
--------- --------- --------- --------- --------- --------- ... ---------
*us*      Yes       Yes       Yes       Yes       Yes       ... Yes
int-host1 Yes       Yes       Yes       Yes       Yes       ... Yes
int-host2 Yes       Yes       Yes       Yes       Yes       ... Yes
int-host3 N/A       N/A       N/A       N/A       N/A       ... N/A
int-host4 N/A       N/A       N/A       N/A       N/A       ... N/A
int-host5 N/A       N/A       N/A       N/A       N/A       ... N/A
int-host6 N/A       N/A       N/A       N/A       N/A       ... N/A
int-host7 N/A       N/A       N/A       N/A       N/A       ... N/A
int-host8 N/A       N/A       N/A       N/A       N/A       ... N/A
```

For a grid where the latest version of the model has yet to be applied and new hosts and instances were added to the model, run the `ttGridAdmin gridSshConfig` command on the active management instance. The `ttGridAdmin` utility then will query the latest version of the model and set up the appropriate SSH connectivity amongst the hosts described in the model.

```
% ttGridAdmin gridSshConfig
Enter password:
Setup ssh configuration on local system..............................OK
Setup ssh configuration on int-host1.................................OK
Setup ssh configuration on int-host2.................................OK
Setup ssh configuration on int-host3.................................OK
Setup ssh configuration on int-host4.................................OK
Setup ssh configuration on int-host5.................................OK
Setup ssh configuration on int-host6.................................OK
Setup ssh configuration on int-host7.................................OK
Setup ssh configuration on int-host8.................................OK
Setup passwordless ssh from local system to int-host1.................OK
Setup passwordless ssh from local system to int-host2.................OK
Setup passwordless ssh from local system to int-host3.................OK
Setup passwordless ssh from local system to int-host4.................OK
Setup passwordless ssh from local system to int-host5.................OK
Setup passwordless ssh from local system to int-host6.................OK
Setup passwordless ssh from local system to int-host7.................OK
Setup passwordless ssh from local system to int-host8.................OK
Setup passwordless ssh from int-host1 to int-host1....................OK
Setup passwordless ssh from int-host1 to int-host2....................OK
Setup passwordless ssh from int-host1 to int-host3....................OK
Setup passwordless ssh from int-host1 to int-host4....................OK
Setup passwordless ssh from int-host1 to int-host5....................OK
Setup passwordless ssh from int-host1 to int-host6....................OK
Setup passwordless ssh from int-host1 to int-host7....................OK
Setup passwordless ssh from int-host1 to int-host8....................OK
Setup passwordless ssh from int-host2 to int-host1....................OK
Setup passwordless ssh from int-host2 to int-host2....................OK
Setup passwordless ssh from int-host2 to int-host3....................OK
```

```
Setup passwordless ssh from int-host2 to int-host4...................OK
Setup passwordless ssh from int-host2 to int-host5...................OK
Setup passwordless ssh from int-host2 to int-host6...................OK
Setup passwordless ssh from int-host2 to int-host7...................OK
Setup passwordless ssh from int-host2 to int-host8...................OK

Passwordless ssh working between hosts:

From\To   int-host1 int-host2 int-host3 int-host4 int-host5 ... int-host8
--------- --------- --------- --------- --------- --------- ... ---------
*us*      Yes       Yes       Yes       Yes       Yes       ... Yes
int-host1 Yes       Yes       Yes       Yes       Yes       ... Yes
int-host2 Yes       Yes       Yes       Yes       Yes       ... Yes
int-host3 N/A       N/A       N/A       N/A       N/A       ... N/A
int-host4 N/A       N/A       N/A       N/A       N/A       ... N/A
int-host5 N/A       N/A       N/A       N/A       N/A       ... N/A
int-host6 N/A       N/A       N/A       N/A       N/A       ... N/A
int-host7 N/A       N/A       N/A       N/A       N/A       ... N/A
int-host8 N/A       N/A       N/A       N/A       N/A       ... N/A
```

For more information on the `ttGridAdmin gridSshConfig` command, see "Configure SSH (gridSshConfig)" in the *Oracle TimesTen In-Memory Database Reference*.

3
Setting Up the Membership Service

This chapter discusses how to set up your membership service.

- Overview of the TimesTen Scaleout membership service
- Using Apache ZooKeeper as the membership service
- Installing Apache ZooKeeper
- Configuring Apache ZooKeeper as the membership service
- Starting the membership servers
- Configure a grid as a membership service client

Overview of the TimesTen Scaleout membership service

The TimesTen Scaleout membership service enables a grid to operate in a consistent manner, even if it encounters a network failure between instances that interrupts communication and cooperation between the instances.

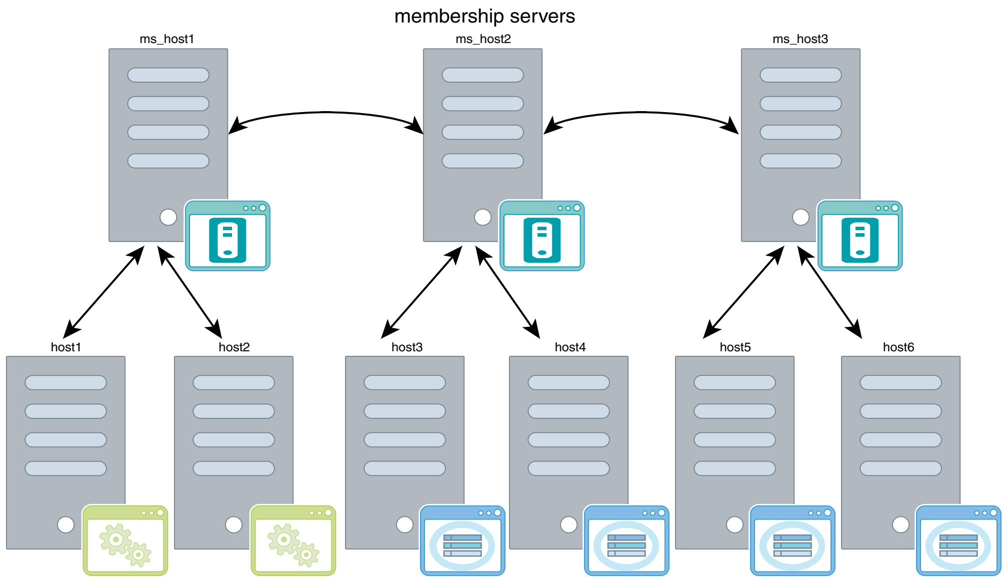
The TimesTen Scaleout membership service performs the following:

- Tracking the instance status. This helps instances maintain communication between each other.
- Recovering from a network partition error, once the communications fault is fixed.

Tracking the instance status

A grid is a collection of instances that reside on multiple hosts that communicate over a single private network. The membership service knows which instances are active. When each instance starts, it connects to a membership server within the membership service to register itself, as shown in Figure 3–1. If one of the membership servers fails, the instances that were connected to the failed membership server transparently reconnect to one of the available membership servers.

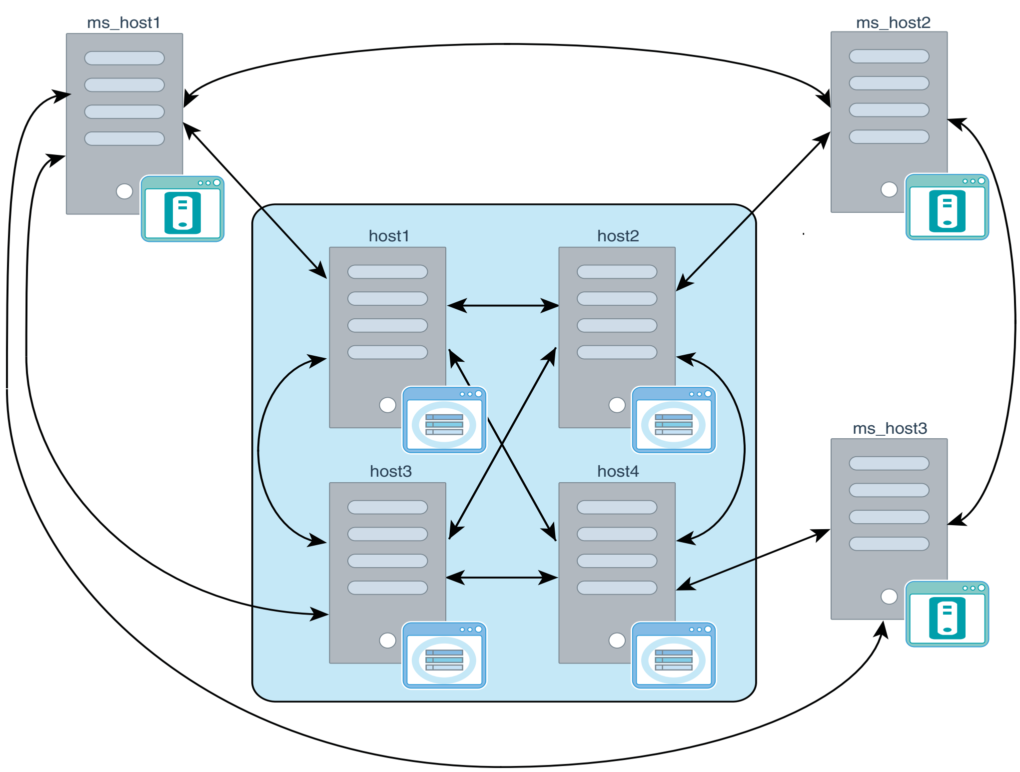
Figure 3–1 Instances register with the membership servers



Each instance maintains a persistent connection to one of the membership servers, so that it can query the active instance list. If the network between the membership servers and the instances is down, the instances refuse to perform until the network is fixed and communication is restored with the membership servers.

Figure 3–2 demonstrates how data instances in a grid connect to each other, where each data instance connects to every other data instance in a grid. It also shows how each data instance in this example maintains a persistent connection with one of the membership servers.

Figure 3–2 Data instances communicating with each other

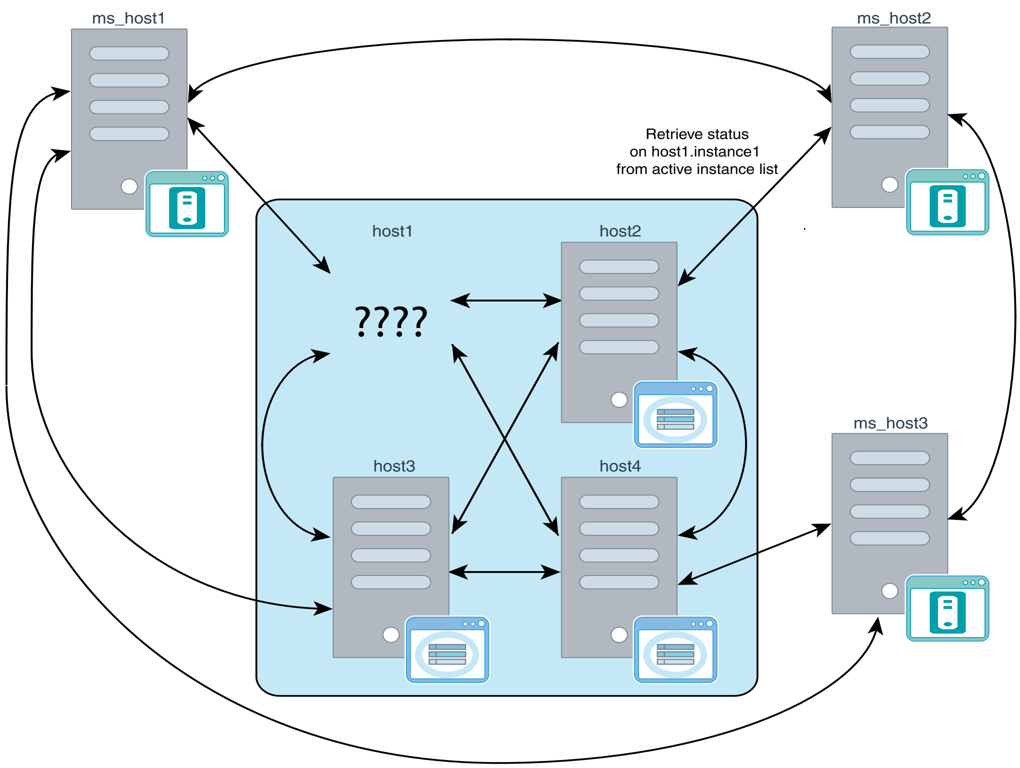


If a data instance loses a connection to another instance, it queries the active instance list on its membership server to verify if the "lost" instance is up. If the "lost" instance is up, then the data instance makes an effort to re-establish a connection with that instance. Otherwise, to avoid unnecessary delays, no further attempts are made to establish communication to the "lost" instance.

When a "lost" instance restarts, it registers itself with the membership service and proactively informs all other instances in a grid that it is up. When it is properly synchronized with the rest of a grid, the recovered instance is once again used to process transactions from applications.

In Figure 3–3, the `host1` data instance is not up. If the `host2` data instance tries to communicate with the `host1` data instance, it discovers a broken connection. The `host2` data instance queries the active instance list on its membership server, which informs it that the `host1` data instance is not on the active instance list. If the `host1` data instance comes back up, it registers itself again with the membership service, which then includes it in the list of active instances in this grid.

Figure 3–3 Instance reacts to a dead connection

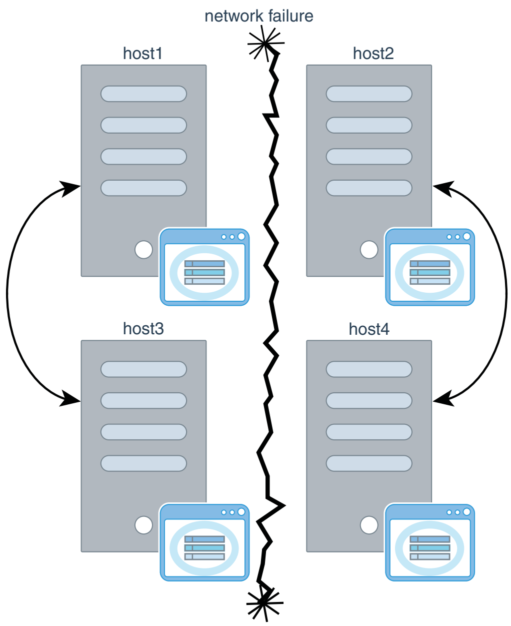


Recovering from a network partition error

A network partition error splits the instances involved in a single grid into two subsets. With a network partition error, each subset of instances is unable to communicate with the other subset of instances.

Figure 3–4 shows a network partition that would return inconsistent results to application queries without the membership service, since the application could access one subset of instances without being able to contact the disconnected subset of instances. Any updates made to one subset of instances would not be reflected in the other subset. If an application connects to the `host1` data instance, then the query returns results from the `host1` and `host3` data instances; but any data that resides on the host2 and host4 data instances is not available because there is no connection between the two subsets.

Figure 3–4 Network partition failure



If you encounter a network partition, the membership service provides a resolution. Figure 3–5 shows a grid with four instances and three membership servers. A network communications error has split a grid into two subsets where host1 and host3 no longer know about or communicate with host2 and host4. In addition, the ms_host1 membership server is not in communication with the other two membership servers.

For the membership service to work properly to manage the status of a grid, there must be a majority of active membership servers of the total servers created that can communicate with each other in order to work properly. If a membership server fails, the others continue to serve requests as long as a majority is available.

For example:

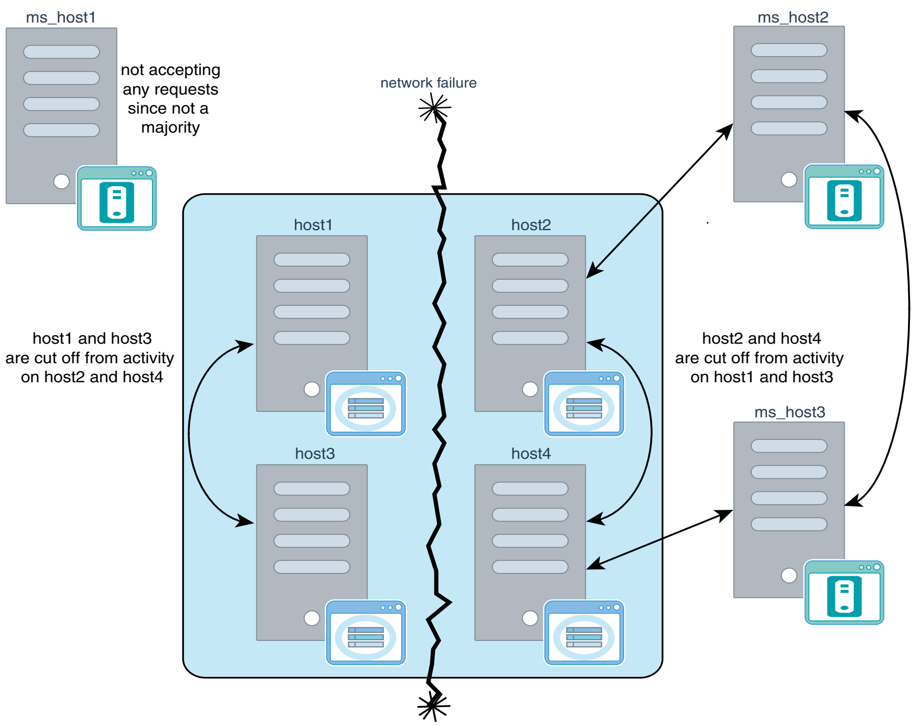
- A membership service that consists of three membership servers can handle one membership server failure.
- A membership service of five membership servers can handle two membership server failures.
- A membership service of six membership servers can handle only two failures since three membership servers are not a majority.

Note: When you configure the number of membership servers, you should always create an odd number of membership servers to serve as the membership service. If you have an even number of membership servers and a network partition error occurs, then each subset of a grid might have the same number of membership servers where neither side would have a majority. Thus, both sides of the network partitioned grid would stop working.

If the number of remaining membership servers falls below the number needed for a majority, the remaining membership servers refuse all requests until at least a majority of membership servers are running. In addition, data instances that cannot communicate with the membership service cannot execute any transactions. You must research the failure issue and restart any failed membership servers.

Because of the communications failure, the `ms_host1` membership server does not know about the other two membership servers. Since there are not enough membership servers that it does know to constitute a majority, the `ms_host1` membership server can no longer accept incoming requests from the `host1` and `host3` data instances. And the `host1` and `host3` data instances cannot execute any transactions until the failed membership server is restarted.

Figure 3–5 Network partition with membership service



To discover if there may be a network partition, you will see errors in the daemon log about elements losing contact with their membership server.

Once you resolve the connection error that caused your grid to split into two, all of the membership servers reconnect and synchronize the membership information. In our example in Figure 3–5, the `ms_host1` membership server rejoins the membership service. After which, the `host1` and `host3` data instances also rejoin this grid as active instances.

Using Apache ZooKeeper as the membership service

Apache ZooKeeper is a third-party, open-source centralized service that maintains information for distributed systems and coordinates services for multiple hosts. TimesTen Scaleout uses Apache ZooKeeper to provide its membership service, which tracks the status of all instances and provides a consistent view of the instances that are active within a grid.

TimesTen Scaleout requires that you install and configure Apache ZooKeeper to work as the membership service for a grid. Each membership server in a grid is an Apache ZooKeeper server.

Note: Since membership servers are ZooKeeper servers, see the Apache ZooKeeper documentation on how to use and manage ZooKeeper servers at `http://zookeeper.apache.org`.

If you create a second grid, you can use the same ZooKeeper servers to act as the membership service for the second grid. However, all ZooKeeper servers should act only as a membership service for TimesTen Scaleout.

For ZooKeeper servers in a production environment, it is advisable to:

- Configure an odd number of replicated ZooKeeper servers on separate hosts. Use a minimum of three ZooKeeper servers for your membership service. If you have *n* Zookeeper servers, you should have (*n*`/2+1`) ZooKeeper servers alive as a majority. A larger number of ZooKeeper servers increases reliability.
- It is recommended (but not required) that you use hosts for your membership servers that are separate from any hosts used for instances. If you do locate your ZooKeeper servers and instances on separate hosts, then this guarantees that if the host fails, you do not lose both the instance and one of the membership servers.
- Avoid having ZooKeeper servers be subject to any single point of failure. For example, use independent physical racks, power sources, and network locations.
- Your Zookeeper servers could share the same physical infrastructure as your data instances. For example, if your data instances are spread across two physical racks, you could host your Zookeeper servers in these same two racks.

  For example, you configure your grid with an active and standby management instance, two data space groups (each with three data instances), and three ZooKeeper servers configured in your grid. If you have two data racks, the best way to organize your hosts is to:

  - Locate one of the management instances on rack 1 and the other management instance on rack 2.
  - Locate two of the ZooKeeper servers on rack 1 and the third on rack 2.
  - Locate the hosts for data instances for data space group 1 on rack 1 and the hosts for the data instances for data space group 2 on rack 2.

  Thus, if rack 2 loses power or its ethernet connection, this grid continues to work since rack 1 has the majority of ZooKeeper servers. If rack 1 fails, you lose the majority of the ZooKeeper servers and need to recover your ZooKeeper servers. A grid does not work without at least a majority of the configured ZooKeeper servers active.

Note: For more directions for best practices for your ZooKeeper servers, go to: `http://zookeeper.apache.org`.

Installing Apache ZooKeeper

On each host on which you intend to provide a membership server, install the TimesTen-specific Apache ZooKeeper distribution, which is a ZooKeeper TAR file located in the *installation_dir*/tt18.1.4.1.0/3rdparty directory of the TimesTen installation.

Important:

- Using Apache ZooKeeper as a membership service for TimesTen Scaleout requires Java release 1.8 (JDK 8) or greater on each ZooKeeper server.
- All hosts that contain data instances, management instances and membership servers must be connected to the same private network.

---

1. Create a directory for the ZooKeeper installation on each host that you intend to act as one of the membership servers. You may install the ZooKeeper distribution file into any directory with any name you wish.
2. From a host where you have already installed TimesTen Scaleout, copy the ZooKeeper apache-zookeeper-3.5.8-bin.tar.gz file from *installation_dir*/tt18.1.4.1.0/3rdparty to the desired directory on each host.
3. Unpack the provided Apache ZooKeeper distribution using the standard operating system tar command into the desired location on each host intended to be a membership server.

The following example on Linux unpacks an Apache ZooKeeper installation into the zkdir directory (a subdirectory of the current directory). A TimesTen Scaleout installation on host1 is located in /swdir/TimesTen/tt18.1.4.1.0.

On the ms1_host membership server, create the zkdir directory.

```
% mkdir -p zkdir
```

Copy the apache-zookeeper-3.5.8-bin.tar.gz file from the *installation_dir*/tt18.1.4.1.0/3rdparty directory on host1 to the zkdir directory you created on ms1_host.

```
% tar -C zkdir -xzvf
/swdir/TimesTen/tt18.1.4.1.0/3rdparty/apache-zookeeper-3.5.8-bin.tar.gz
[...TAR OUTPUT...]
```

Note: The version of the ZooKeeper distribution that TimesTen Scaleout provides is shown in the name of the TAR file provided in the *installation_dir*/18.1.4.1.0/3rdparty directory. For example, the apache-zookeeper-3.5.8-bin.tar.gz file in this example shows that the provided Apache ZooKeeper distributed version is 3.5.8.

Configuring Apache ZooKeeper as the membership service

To configure each Apache ZooKeeper server to act as a membership server for your grid, you need to configure the following configuration files on each host that hosts a membership server:

- zoo.cfg configuration file: In replicated mode, each membership server has a zoo.cfg configuration file. The zoo.cfg configuration file identifies all of the membership servers involved in the membership service, where each membership server is identified by its DNS (or IP address) and port number.

All configuration parameters in the `zoo.cfg` on each membership server must be exactly the same, except for the client port. The client port can be different (but is not required to be different) for each membership server. The client port can be the same if each membership server runs on a different host.

Place the `zoo.cfg` file in the Apache ZooKeeper installation `/conf` directory. For example, if you unpacked the `apache-zookeeper-3.5.8-bin.tar.gz` file into the `/swdir/zkdir` directory on each membership server, then you would place the `zoo.cfg` file into the following directory:

```
/swdir/zkdir/apache-zookeeper-3.5.8-bin/conf/zoo.cfg
```

- `myid` configuration file: Provides the number that identifies this particular membership server. Each membership server is identified by a unique number. For example, if you have 5 servers, they must be identified with unique integers of `1`, `2`, `3`, `4` and `5`.

  This number corresponds to the definition of the host in the `zoo.cfg` file by the *x* in the `server.`*x* parameter. All `zoo.cfg` files must have a listing for all membership servers. For example, if you have 5 membership servers, they are configured as `server.1`, `server.2`, and so on in the `zoo.cfg` file.

  The `myid` configuration file on each host contains a single line with the integer number of that server. For example, the 2nd membership server is identified in `zoo.cfg` as `server.2` and in its `myid` configuration file is a single line with a `2`.

  The `myid` configuration file is a text file located in the Apache ZooKeeper data directory of the membership server. The location of the data directory is configured with the `dataDir` parameter in the `zoo.cfg` file. For example, if you configure the data directory to be `/swdir/zkdir/3.5.8/data`, then you would place the `myid` text configuration file as follows:

  ```
  /swdir/zkdir/apache-zookeeper-3.5.8-bin/data/myid
  ```

Table 3–1 shows the commonly used configuration parameters for the `zoo.cfg` file.

Table 3–1 zoo.cfg configuration parameters

Parameter	Description
`tickTime`	The unit of time (in milliseconds) used for each tick for both `initLimit` and `syncLimit` parameters. For the best performance, you should set this to the recommended setting of `250` milliseconds. This parameter is required to run the membership server in replicated mode.
`initLimit`	The timeout (in ticks) for how long the membership servers have to connect to the leader. For the best performance, you should set this to the recommended setting of `40` ticks. This parameter is required to run the membership server in replicated mode.
`syncLimit`	The limit of how out of date a membership server can be from a leader. This limit (in ticks) specifies how long is allowed between sending a request and receiving an acknowledgement. For best performance, you should set this recommended setting to `12` ticks. This parameter is required to run the membership server in replicated mode.

Table 3–1 (Cont.) zoo.cfg configuration parameters

Parameter	Description
`dataDir`	You decide on and create the data directory location to store the ZooKeeper data, snapshots and its transaction logs. When creating the directory where the transaction logs are written, it is important to your performance that the transaction logs are written to non-volatile storage. A dedicated device for your transaction logs is key to consistent good performance. Logging your transactions to a busy device adversely effects performance.
`clientPort`	The port on which to listen for client connections. The default is port `2181`.
`autopurge.snapRetainCount`	Defines the number of most recent snapshots and corresponding Apache ZooKeeper transaction logs to keep in the `dataDir` and `dataLogDir` respectively. Defaults to 3.
`autopurge.purgeInterval`	The time interval in hours for when to trigger the purge of older snapshots and corresponding Apache ZooKeeper transaction logs. Set to a positive integer (1 and above) to enable the auto purge. Defaults to 0. We recommend that you set this to 1.
`minSessionTimeout`	The minimum session timeout in milliseconds that the server will allow the client to negotiate. Defaults to 2 times the `tickTime`.
`maxSessionTimeout`	The maximum session timeout in milliseconds that the server will allow the client to negotiate. Defaults to 20 times the `tickTime`.

Table 3–1 (Cont.) zoo.cfg configuration parameters

Parameter	Description
server.*x*=[systemName]:nnnnn:nnnnn	The configuration for each membership server is identified by the server.*x* parameter. The list of hosts defined by this parameter designate all of the membership servers used by the membership service. This list must correlate to the same list of membership servers in each zoo.cfg file on each membership server in the membership service. This parameter is required to run the membership server in replicated mode. The *x* is the identifying integer number for the membership server, which is also configured in the myid configuration file on the membership server. The systemName parameter specifies the DNS (or IP address) of the host on which the membership server is installed and will run. If no systemName is provided for the server, the default is localhost. Define two port numbers after each server name. ■ First port number: Used by peers to connect to and communicate with other peers. This port connects followers to the leader. ■ Second port number: Used for leader election among the membership servers. If necessary, this port is used to elect a new leader in case of failure. For a production environment, each of the membership servers should be configured on different hosts. In this case, the convention is to assign the same port numbers, such as: server.1=system1:2888:3888 server.2=system2:2888:3888 server.3=system3:2888:3888 However, for a testing environment, you may want to place all membership servers on the same host. In this case, you need to configure all membership servers with different ports.
4lw.commands.whitelist	Enables the specified Zookeeper four-letter-words commands. TimesTen Scaleout utilities like ttGridRollout require some of these commands to operate properly.

All membership servers that are installed should be run in replicated mode. To run your membership servers in replicated mode, you need to include the tickTime, initLimit, and syncLimit parameters and provide the host name with two port numbers for each membership server.

Note: For more details on replicated mode, go to: http://zookeeper.apache.org.

Then, refer to the Getting Started > Running Replicated ZooKeeper section of the documentation.

The following example demonstrates the zoo.cfg membership server configuration file, where there are three membership servers installed on hosts whose DNS names are ms_host1, ms_host2 and ms_host3. All three membership servers are configured to run in replicated mode.

```
# The number of milliseconds of each tick
tickTime=250
```

```
# The number of ticks that the initial synchronization phase can take
initLimit=40
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=12
# The directory where you want the ZooKeeper data stored.
dataDir=/swdir/zkdir/apache-zookeeper-3.5.8-bin/data
# The port at which the clients will connect
clientPort=2181
# Every hour, keep the latest three Apache ZooKeeper snapshots and
# transaction logs and purge the rest
autopurge.snapRetainCount=3
autopurge.purgeInterval=1
# The minimum and maximum allowable timeouts for Apache ZooKeeper sessions.
# Actual timeout is negotiated at connect time.
minSessionTimeout=2000
maxSessionTimeout=10000
# The membership servers
server.1=ms_host1:2888:3888
server.2=ms_host2:2888:3888
server.3=ms_host3:2888:3888
# Enabled Zookeeper four-letter-words commands
4lw.commands.whitelist=stat, ruok, conf, isro
```

Note: There is a sample file that explains some of the parameters for your `zoo.cfg` file in the Apache ZooKeeper installation `/conf` directory called `zoo_sample.cfg`. However, it does not have all of the recommended parameters or settings for TimesTen Scaleout. Use `zoo_sample.cfg` for reference only.

This example creates a `myid` text file on three hosts, where each is a membership server. Each `myid` text file contains a single-line with the server id (an integer) corresponding to one of the membership servers configured in the `zoo.cfg` file. The server id is the number *x* in the `server.`*x*= entry of the configuration file. The `myid` text file must be located within the data directory on each membership server. The data directory location is `/swdir/zkdir/apache-zookeeper-3.5.8-bin/data`.

- Create a `myid` text file in the `/swdir/zkdir/apache-zookeeper-3.5.8-bin/data` directory on `ms_host1` for its membership server. The `myid` text file contains the value `1`.
- Create a `myid` text file in the `/swdir/zkdir/apache-zookeeper-3.5.8-bin/data` directory on `ms_host2` for its membership server. The `myid` text file contains the value `2`.
- Create a `myid` text file in the `/swdir/zkdir/apache-zookeeper-3.5.8-bin/data` directory on `ms_host3` for its membership server. The `myid` text file contains the value `3`.

When the membership server starts up, it identifies which server it is in by the integer configured in the `myid` file in the ZooKeeper data directory.

Note: For full details of the configuration parameters that can exist in the Apache ZooKeeper `zoo.cfg` configuration file, see `http://zookeeper.apache.org`.

Starting the membership servers

Before you can start the membership server with the zkServer.sh shell script, you need to set the maximum Java heap size, which determines if ZooKeeper swaps to the file system. The Java maximum heap size should not be larger than the amount of available real memory. Edit the zkEnv.sh shell script to add a new line with the JVMFLAGS environment variable setting the maximum Java heap size to 4 GB. Upon startup, the zkServer.sh shell script sources the zkEnv.sh shell script to include this new environment variable.

The ZooKeeper shell scripts are located in the ZooKeeper server /bin directory. For example, if you unpacked the apache-zookeeper-3.5.8-bin.tar.gz file into the /swdir/zkdir directory on each membership server, then the zkEnv.sh and zkServer.sh shell scripts are located in the /swdir/zkdir/bin directory.

The following example edits the zkEnv.sh shell script and adds the JVMFLAGS=Xmx4g configuration within the zkEnv.sh script after the line for ZOOKEEPER_PREFIX.

```
ZOOBINDIR="${ZOOBINDIR:-/usr/bin}"
ZOOKEEPER_PREFIX="${ZOOBINDIR}/.."
JVMFLAGS=-Xmx4g
```

Start each membership server by running the zkServer.sh start shell script on each server.

```
% setenv ZOOCFGDIR /swdir/zkdir/apache-zookeeper-3.5.8-bin/conf
% /swdir/zkdir/apache-zookeeper-3.5.8-bin/bin/zkServer.sh start
```

You can verify the status for each membership server by executing the zkServer.sh status command on each membership server:

```
% /swdir/zkdir/apache-zookeeper-3.5.8-bin/bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /swdir/zkdir/apache-zookeeper-3.5.8-bin/conf/zoo.cfg
Mode: { leader | follower }
```

If the membership server is not running, is not in replicated mode, or there is not a majority executing, these errors are displayed:

```
ZooKeeper JMX enabled by default
Using config: /swdir/zkdir/apache-zookeeper-3.5.8-bin/conf/zoo.cfg
Error contacting service. It is probably not running.
```

Additionally, you can verify if a membership sever is running in a non-error state with the ruok ZooKeeper command. The command returns imok if the server is running. There is no response otherwise. From a machine within the network, run:

```
% echo ruok | nc ms_host1 2181
imok
```

For statistics about performance and connected clients, use the stat ZooKeeper command. From a machine within the network, run:

```
% echo stat | nc ms_host1 2181
```

Once the membership servers are started, you can create your grid. See "Configure a grid as a membership service client" on page 3-13 for details on how to create a grid that knows about your membership servers.

Configure a grid as a membership service client

A grid must know how to connect to each of the membership servers. Thus, you must provide a ZooKeeper client configuration file to the `ttGridAdmin` utility when you create a grid that details all of the membership servers. You can name the ZooKeeper client configuration file with any prefix as long as the suffix is `.conf`.

The ZooKeeper client configuration file specifies all membership servers that coordinate with each other to provide a membership service. Within the client configuration file is a single line with the `Servers` parameter that provides the DNS (or IP address) and client port numbers for each membership server. The configuration information for these hosts must:

- Use the same DNS (or IP address) as what you specified in the `server.x` parameters in each of the individual `zoo.cfg` files on each membership server.
- Provide the same client port number as what is specified in the `clientPort` parameter specified in each of the individual `zoo.cfg` files on each membership server.

In our example, we use the `membership.conf` file as the ZooKeeper client configuration file. For this example, there are three hosts that support three membership servers, where `ms_host1` listens on client port `2181`, `ms_host2` listens on client port `2181`, and `ms_host3` listens on client port `2181`.

```
Servers ms_host1!2181,ms_host2!2181,ms_host3!2181
```

A grid knows how to reach these membership servers because the ZooKeeper client configuration file is provided as an input parameter when you create your grid. See "Creating a grid" on page 4-4 for more details.

Once you provide the ZooKeeper client configuration file to the `ttGridAdmin` command when a grid is created, the ZooKeeper client configuration file is no longer needed and can be discarded.

Note: You can modify the list of provided membership servers for a grid by importing a new list of membership servers. See "Reconfiguring membership servers" on page 8-6 for details.

4

Setting Up a Grid

This chapter discusses how to create and configure a grid in TimesTen Scaleout.

- Configure your grid
- Description of the physical topography of the grid

Configure your grid

A grid is a set of associated instances that contain the distributed data of one or more databases. There are two types of instances in a grid:

- **Management instances** control a grid and maintain the model, which is the central configuration of a grid. You can configure up to two management instances to provide availability for the management of the grid.
- **Data instances** store the data of every database managed by the grid.

These sections describe the tasks to set up a grid:

- Creating the initial management instance
- Creating a grid
- Adding the standby management instance
- Calculating the number of hosts and data instances for the grid
- Assigning hosts to data space groups
- Adding data instances
- Applying the changes made to the model
- Setting instances to automatically start at system startup

Note: While this chapter describes the tasks necessary to completely configure a grid by using the command line and the `ttGridAdmin` utility, it is also possible to configure a grid by using Oracle SQL Developer. See "Working with TimesTen Scaleout" in the *Oracle SQL Developer Oracle TimesTen In-Memory Database Support User's Guide* for more information.

Additionally, TimesTen Scaleout provides the `ttGridRollout` to quickly set up a simple grid with a single database for development and testing purposes. See "ttGridRollout" in the *Oracle TimesTen In-Memory Database Reference* and "Deploy a grid and database" on page A-5 in this document for more information on the utility and an example of its use, respectively.

Creating the initial management instance

TimesTen Scaleout uses management instances to configure and manage a grid. A management instance stores and maintains the model, a comprehensive list of the objects that give shape to a grid.

Important:

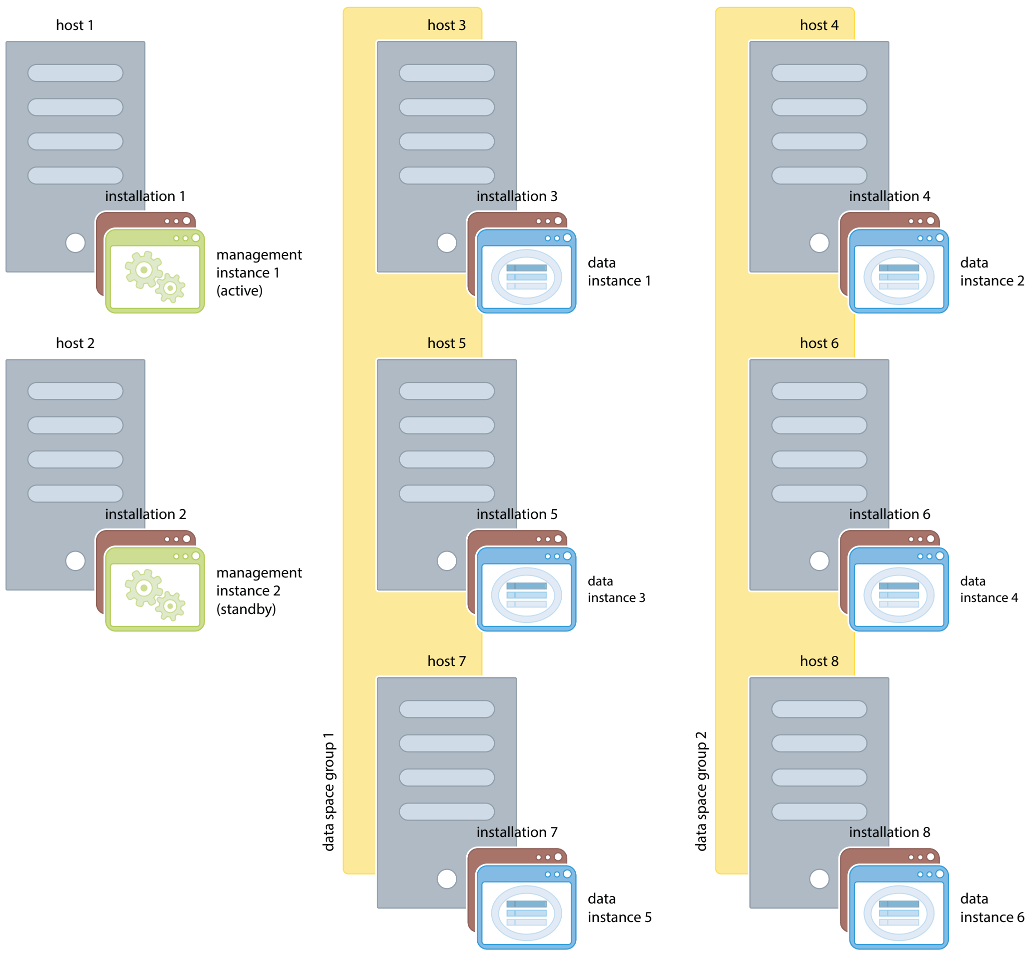
- TimesTen Scaleout stores multiple versions of the model that may describe a previous, present, or desired structure of a grid. See "Model versioning" on page 4-18 for more information on how these versions of the model are created or managed.
- Most model objects have a user-defined name. TimesTen Scaleout uses those names to define relationships between model objects. In general, each type of model object has its own namespace. See "Grid objects and object naming" in the *Oracle TimesTen In-Memory Database Reference* for further details.
- See "Central configuration of the grid" on page 1-16 for a complete list of the types of model objects and their descriptions.

To ensure high availability for the management of the grid, TimesTen Scaleout enables you to create a standby management instance in an active standby configuration. It is highly recommended that you configure a standby management instance, which would be available in the case of a failure of the active management instance. If you only have a single management instance and it fails, the databases remain operational, but most management operations are unavailable until the management instance is restored. The steps to set up a standby management instance are discussed later in this chapter.

The `ttInstanceCreate` utility creates new instances. You create the initial management instance with the `ttInstanceCreate` utility by including the `-grid` option to enable the instance for TimesTen Scaleout management. Once you create a grid from this instance, all subsequent instances associated with the grid are created through the `ttGridAdmin` utility. All instances in the grid share the same OS username as instance administrator.

Note: The tasks described in this and the next several sections use a scenario of a grid with a K-safety (`k`) set to `2` and that consists of eight hosts: two hosts with a TimesTen installation and a management instance, and two data space groups with three hosts each, each host with a TimesTen installation and a data instance. Figure 4–1 shows a graphical representation of this scenario.

Figure 4–1 Grid scenario



On a host with a TimesTen 18.1 installation, create a management instance in a location of your choice, for example, the `/grid` directory.

Note: See Chapter 2, "Prerequisites and Installation of TimesTen Scaleout" for information on how to install TimesTen and its prerequisites for TimesTen Scaleout.

```
% /grid/tt18.1.4.1.0/bin/ttInstanceCreate -name instance1 -location
 /grid -grid
Creating instance in /grid/instance1 ...
INFO: Mapping files from the installation to /grid/instance1/install

NOTE: The TimesTen daemon startup/shutdown scripts have not been installed.
```

```
The startup script is located here :
        '/grid/instance1/startup/tt_instance1'

Run the 'setuproot' script :
        /grid/instance1/bin/setuproot -install
This will move the TimesTen startup script into its appropriate location.

The 18.1.4.1 Release Notes are located here :
  '/grid/tt18.1.4.1.0/README.html'
```

Note:

- TimesTen Scaleout sets `instance1` as the default instance name of new instances when you create them with the `ttGridAdmin` utility. Subsequent instances that you create on the same host require that you provide a different name for the instances. The example uses `instance1` to stay in line with the default value. You may use the name of your choice.
- TimesTen Scaleout creates a subdirectory with the instance name in the specified location. TimesTen Scaleout creates all instance files in this subdirectory. For example, the instance files of the `instance1` management instance are allocated in the `/grid/instance1` directory of the local system.
- TimesTen Scaleout sets the default values for the TCP/IP port numbers of the instance daemon and server (`6624` and `6625`, respectively) if you do not specify a value for the port numbers. Use the `-daemonPort` or `-csPort` options of the `ttInstanceCreate` utility to set different values for the port numbers.

Ensure that you set the environment variables for the `instance1` management instance with the `ttenv` script (`ttenv.csh` or `ttenv.sh`) appropriate for your shell.

For a Bourne-type shell, such as `sh`, `bash`, `zsh`, or `ksh`:

```
$ . /grid/instance1/bin/ttenv.sh
```

For a `csh` or `tcsh` shell:

```
% source /grid/instance1/bin/ttenv.csh
```

For more information on the `ttInstanceCreate` utility, see "ttInstanceCreate" in the *Oracle TimesTen In-Memory Database Reference*.

See "Environment variables" on page B-1 for more information on the environment variables.

Creating a grid

You can manipulate the state and configuration of a grid with the `ttGridAdmin` utility. All operations that require the use of the `ttGridAdmin` utility must be performed by the instance administrator and from the active management instance, unless stated otherwise. Use this utility to perform all the operations related to the configuration and maintenance of a grid, which include:

- Creating a new grid
- Creating and removing model objects such as hosts and instances

- Creating and destroying databases
- Defining and modifying how the user data is distributed across the available data instances
- Modifying the attributes of model objects such as the connection attributes of the databases
- Querying the status of the grid and its databases
- Maintaining the different versions of the model
- Applying the changes made to the latest version to the model to the operational grid.

Note: For more information on the `ttGridAdmin` utility, see "ttGridAdmin" in the *Oracle TimesTen In-Memory Database Reference*.

The `ttGridAdmin gridCreate` command performs the next operations:

- Starts the active management instance.
- Creates a grid with a user-defined name.
- Creates the required number of data space groups as indicated by the value of K-safety.
- Defines the client configuration of the membership service. See Chapter 3, "Setting Up the Membership Service" for details on the membership service and the required ZooKeeper client configuration file.
- Adds the management instance and its associated host and installation as model objects to the latest version of the model.

Create a grid with `k` set to `2`. Specify a name for the grid, the internal address or the internal and external address of the local system, and provide the ZooKeeper client configuration file.

```
% ttGridAdmin gridCreate grid1 -k 2 -internalAddress int-host1
 -externalAddress ext-host1.example.com -membershipConfig
 /tmp/membership.conf
Grid grid1 created
```

Note:

- If you do not specify the `-host` option of the `ttGridAdmin gridCreate` command, TimesTen Scaleout sets the hostname of the local system as the name of the host in the model.
- TimesTen Scaleout automatically identifies the local TimesTen installation as the `installation1` installation.
- TimesTen Scaleout sets the default value for the TCP/IP port number of the replication agent of the active management instance (`3754`) if you do not specify a value for the port number. Use the `-mgmtPort` option in the `ttGridAdmin gridCreate` command to specify a different value for the port number.

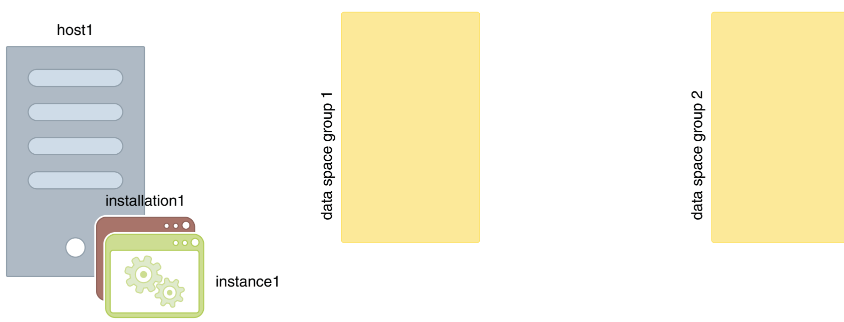
To create the `grid1` grid, TimesTen Scaleout starts the `instance1` management instance. Then, the `instance1` management instance creates the `grid1` grid and its model. Finally, the `instance1` management instance performs these operations in the model of the `grid1` grid:

- Creates a host object, `host1`, in the model to represent the local system.
- Creates an installation object, `installation1`, in the model to represent the local TimesTen installation.
- Creates an instance object, `instance1`, in the model.
- Associates the `installation1` installation with both the `host1` host and the `instance1` instance.
- Creates two data space groups (since `k` is set to `2`).

> **Important:** From this point forward, the described tasks only add and modify model objects to the latest version of the model and do not make any changes on the systems associated with such model objects until the changes made to the latest version of the model are applied. See "Applying the changes made to the model" on page 4-18 for full details.

Figure 4–2 shows a graphical representation of the model after the creation of the `grid1` grid.

Figure 4–2 The model after creating a grid



For more information on the `ttGridAdmin gridCreate` command, see "Create a grid (gridCreate)" in the *Oracle TimesTen In-Memory Database Reference*.

Adding the standby management instance

TimesTen Scaleout enables you to create a second management instance for a grid. When two management instances exist in a grid, the configuration of the grid is replicated from the active management instance to the standby management instance using an active standby configuration. Replication between the active and standby management instances is asynchronous. See "Managing failover for the management instances" on page 11-33 for more information on how TimesTen Scaleout uses an active standby configuration for the management instances.

TimesTen Scaleout automatically configures the second management instance as the standby. All operations that use and manipulate the configuration of the grid must be performed from the active management instance with the `ttGridAdmin` utility.

It is highly recommended that every management instance that you configure in a grid is located on a different host. Those hosts should be in different failure domains (with independent power, storage, and other resources). You must manually add every host to the model by providing the communication parameters (fully qualified domain or IP address) of the system they are associated with.

The `ttGridAdmin hostCreate` command defines a host object in the model. This command enables you to create an instance (management or data) and copy the attributes, such as the data space group, of an existing host by using the `-like` option. In addition, you have the option to copy the associated installations and instances by using the `-cascade` option along with the `-like` option.

Create a standby management instance and its associated installation by duplicating the host associated with the active management instance, `host1.instance1`. Ensure that you identify the fully qualified domain name or IP address of the new host.

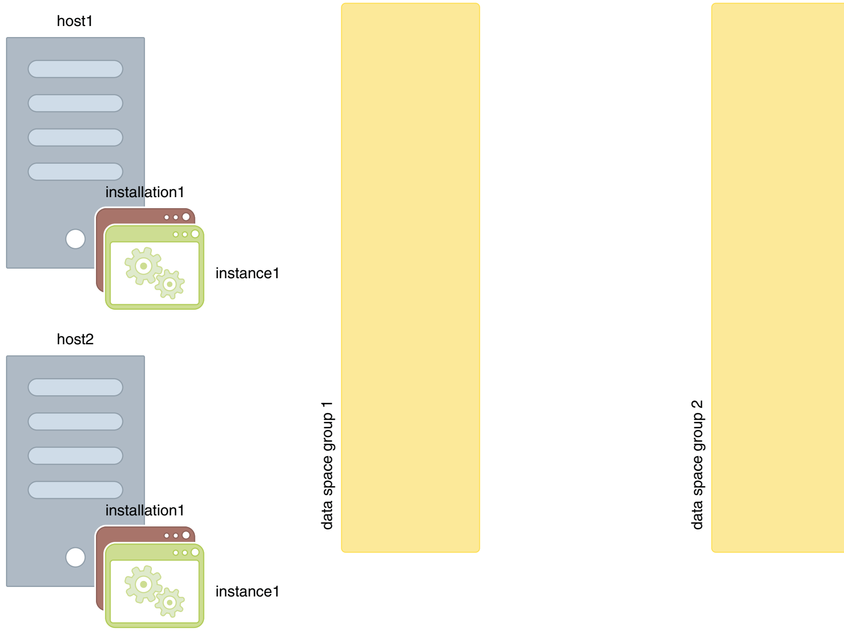
```
% ttGridAdmin hostCreate -internalAddress int-host2 -externalAddress
 ext-host2.example.com -like host1 -cascade
Host host2 created in Model
Installation installation1 created in Model
Instance instance1 created in Model
```

Note:

- If you do not specify a name for the host, TimesTen Scaleout sets the OS hostname of the remote system as the name of the new host.
- Any additional options you define in the `ttGridAdmin hostCreate` command will overwrite the attributes inherited from the existing host in the new host. In this example, TimesTen Scaleout uses the same values for the daemon, server, and management ports (`6624`, `6625`, and `3754`, respectively) as the values set for the `host1.instance1` management instance.
- This example uses the `-like` and `-cascade` options of the `ttGridAdmin hostCreate` command to create the standby management instance and its associated host and installation. Alternatively, you can create them separately. See "Adding data instances" on page 4-11 for more details.

Figure 4–3 shows a graphical representation of the model of the `grid1` grid after creating the `host2` host, `host2.installation1` installation and `host2.instance1` management instance.

Figure 4–3 The model after creating the standby management instance



Notice that the names assigned to the installation and management instance created for the `host2` host are identical to the names assigned to the `host1` host, a result of the cascade operation. This does not generate a conflict, since the fully qualified names are different. See "Grid objects and object naming" in the *Oracle TimesTen In-Memory Database Reference* for more information.

For more information on the `ttGridAdmin hostCreate` command, see "Create a host (hostCreate)" in the *Oracle TimesTen In-Memory Database Reference*.

Calculating the number of hosts and data instances for the grid

A database is distributed across multiple data instances that collectively provide a single database image. Data instances reside on hosts. You create each host and data instance that is to be included in the grid. Thus, you need to calculate how many hosts and data instances to create when you are designing your grid.

The number of copies of the data that you define for the value of K-Safety (`k`) is a factor for how many data instances and hosts that you need to create for your grid. If you define a duplicate copy of the data by setting `k` set to `2`, then you need twice as many data instances and hosts as when a single copy of the data is requested with `k` set to `1`.

Note: `2` is the maximum number that you can assign as the value for `k`.

- Calculate the number of data instances to create
- Calculate the number of hosts you need to support your data instances

Calculate the number of data instances to create

The number of data instances that you create depends on two factors:

- The value of `k`: If you set `k` to 1, the number of data instances you create equals the number of elements you desire for each database. If you set `k` to `2`, then you need to create twice as many data instances, one set of data instances to manage each copy of the database contained within the replica sets.
- The number of replica sets across which you want the data distributed: The number of data instances you create is dictated by the number of elements in all replica sets, since each data instance manages one element of each database.

  All elements that make up a single copy of the database are assigned within a data space. If you set `k` to 2 for two copies of the database, then each replica set contains two elements, where each element is an exact copy of the other element in the replica set. Each data space contains one of the replica elements of each replica set.

  > **Note:** Each data space logically contains a full copy of the data for the database. Since there are *k* copies of the data, there are *k* data spaces.
  >
  > Data instances are assigned to data spaces based on how hosts are assigned to data space groups.

  To calculate the number of replica sets across which you want the data distributed, determine the maximum of the two values below:

  – Database size versus host memory size. The size of the database and the amount of memory you have on each host determines the number of replica sets you want. For example, if you have a two Terabyte database and hosts with 512 Gigabytes of memory each, then you need at least four replica sets to hold all of the data. More likely that you will need five hosts, since you cannot use all of the memory on each host for the data.

  – Throughput. Even if your database is small enough to fit in the memory of a single host, you need to spread your data over multiple hosts if a single host cannot handle the number of transactions per second that your applications require.

Once you decide on the number of replica sets, you can calculate the number of data instances.

For the equation to find the number of data instances required, *r* represents the number of replica sets (where each replica set contains 1 or 2 elements) and *k* represents the K-safety value which denotes the number of copies of the data and subsequently, the number of elements in each replica set. To create enough data instances, you need to create *k* * *r* data instances.

```
number of data instances = k * r
```

For example, if you set `k` to `2` for two copies of the database and each copy of the database is to be distributed across three replica sets, then you need to create 6 data instances where three data instances contain replica elements for data space `1` and three data instances contain the replica elements for data space `2`.

See "Understanding data spaces" on page 1-10 for more details on data spaces. See "Understanding replica sets" on page 1-10 for more details on replica sets.

Calculate the number of hosts you need to support your data instances

To calculate the number of physical or virtual systems for a production deployment of your grid involves considering:

- Hosts for the membership servers. See "Using Apache ZooKeeper as the membership service" on page 3-5 for details on the number of hosts needed for the membership servers.
- Hosts for the management instances. See "Adding the standby management instance" on page 4-6 for details on the number of hosts needed for the management instances.
- Hosts for the data instances. This section describes how many hosts you need to support the number of data instances in your grid.

The number of hosts that you need depends on the how many data instances you install on each host. The following is described in "Data instances" on page 1-8.

Each data instance normally resides on a separate host to provide maximum data availability and as a guard against data loss should one of the hosts fail. However, you might want to run multiple data instances on a single host if:

- The hosts in the grid contain a large amount of computing resources.
- For experimentation of a larger grid before deployment, you might want to test a larger grid configuration on a smaller number of hosts.

Thus, to decide on the number of hosts:

- If you install a single data instance on each host, then the number of hosts required is the same number of data instances in the grid. For example, if you have six data instances, then you would create six hosts.
- If you install more than one data instance on each host, then the number of hosts required depends on how many data instances are on each host. For example, if you have eight data instances and you want to install two data instances on each host, then you only need four hosts.

Once you create the hosts for data instances, you assign them to a data space group. See "Assigning hosts to data space groups" on page 4-10 for details.

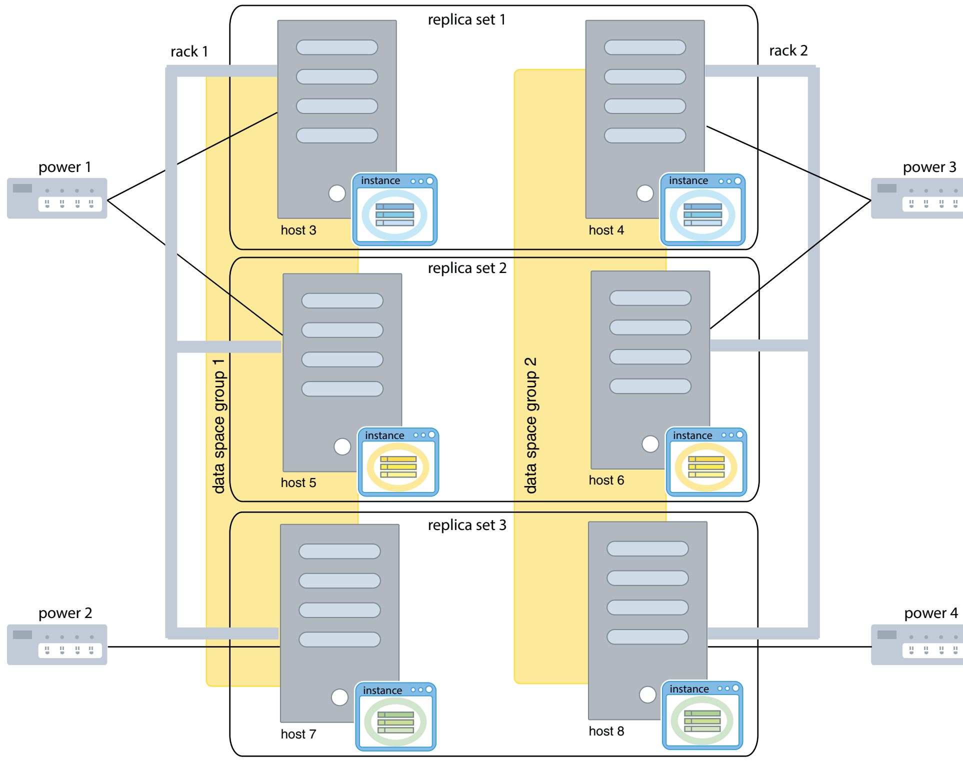
Assigning hosts to data space groups

Data instances will not be created on hosts that are not part of a data space group. The number of data space groups depends on the value set for `k`. If `k` is set to `2`, then you will have two data space groups.

As described in "Assigning hosts to data space groups" on page 1-11, adding hosts to data space groups specifies the physical location of your data. The hosts in one data space group should be physically separate from the group of hosts in another data space group to protect each full copy of the database from hardware failures.

Figure 4–4 is an example of a grid with two data space groups, each containing a single copy of the data. Six hosts contain data instances with the three replica sets that support two copies of the data in a K-safety environment where `k` is set to `2`. Three hosts with data instances that contain a single copy of the data are assigned to data space group 1; the other three hosts with data instances that contain the second copy of the data are assigned to data space group 2.

Figure 4–4 DataSpaceGroup example



You assign the hosts to data space groups so that there is an equal number of hosts in each data space group. However, you can assign the hosts to each data space group later or ask TimesTen Scaleout to recommend host assignments for you.

Note: TimesTen Scaleout cannot create data instances on a host unless the host has been assigned to a data space group.

- You can assign the host to the data space group as part of the host creation with the `-dataspacegroup` option of the `ttGridAdmin hostCreate` command. "Adding data instances" on page 4-11 shows examples of this option.
- You can create the host and assign it to a data space group later with the `ttGridAdmin hostModify -dataspacegroup` command.
- If you have a complicated physical topology, you can use the `ttGridAdmin dataSpaceGroupSuggest` command to receive data space group assignment recommendations from TimesTen Scaleout for multiple hosts with the `ttGridAdmin dataSpaceGroupSuggest` command. See "Description of the physical topography of the grid" on page 4-20 for more details on this method.

Adding data instances

A data instance contains an element for every single database defined in the grid. An element stores a portion of the data of a single database. The data may be distributed among a number of elements equal to the number of data instances defined in the grid.

Perform the following tasks to create data instances in a grid:

- Create a host for a data instance

- Create the installation for the data instance
- Create the data instance
- Create data instances by duplicating the configuration of an existing host

---

Note: Remember that the operations described in the following sections only modify the latest version of the model and do not become part of the operational grid until those changes are applied. See "Applying the changes made to the model" on page 4-18 for full details.

Create a host for a data instance

As with a host associated with a management instance, for every system you intend to use to store a portion of the data of a database, you must manually add the system as a host model object. Likewise, you must provide the communication parameters (fully qualified domain or IP address) of the system. Although, each host can have more than one data instance, it is recommended that you only configure one data instance per host.

To create a data instance, you need to associate the host with a data space group. All data space groups must be associated with the same number of data instances. If you follow the recommendation of one data instance per host, all data space groups must be associated with the same number of hosts.

As mentioned in "Adding the standby management instance" on page 4-6, the `ttGridAdmin hostCreate` command creates a host in the grid. You can associate the host with a data space group or physical group at host creation or later.

Note: If you do not initially associate a host with a data space group, you have the option to let TimesTen Scaleout analyze the model after you have created all the hosts and suggest which hosts to associate with each data space group, based on the physical groups associated with each host, with the use of the `ttGridAdmin dataSpaceGroupSuggest` command. See "Describe your physical topology with physical groups" on page 1-12 and "Assigning hosts to data space groups" on page 4-10 for more information.

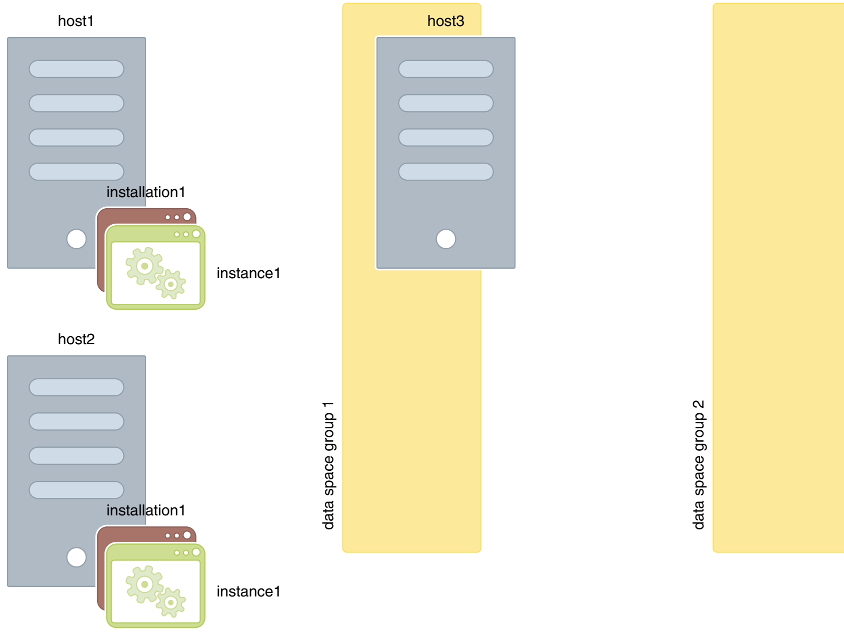
Create a host for a data instance and associate it with data space group `1`. Ensure that you identify the fully qualified domain name or IP address of the host.

```
% ttGridAdmin hostCreate -internalAddress int-host3 -externalAddress
 ext-host3.example.com -dataSpaceGroup 1
Host host3 created in Model
```

Note: If you do not specify a name for the host, TimesTen Scaleout sets the OS hostname of the remote system as the name of the new host.

Figure 4–5 shows a graphical representation of the model of the `grid1` grid after creating the `host3` host.

Figure 4–5 The model after adding a host for a data instance



For more information on the `ttGridAdmin hostCreate` command, see "Create a host (hostCreate)" in the *Oracle TimesTen In-Memory Database Reference*.

Create the installation for the data instance

Every host must have an installation associated with it. A host can either have its own copy of the installation files or share an installation with one or more hosts through network-attached storage. For shared installations, an installation model object with the location of the shared installation files must be associated with the host.

For more information on how to share a TimesTen installation, see "Copying an installation on Linux/UNIX" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide.*

The `ttGridAdmin installationCreate` command creates an installation in the grid and associates it with a host.

Create an installation in a directory of your choice in the `host3` host, for example, the `/grid` directory.

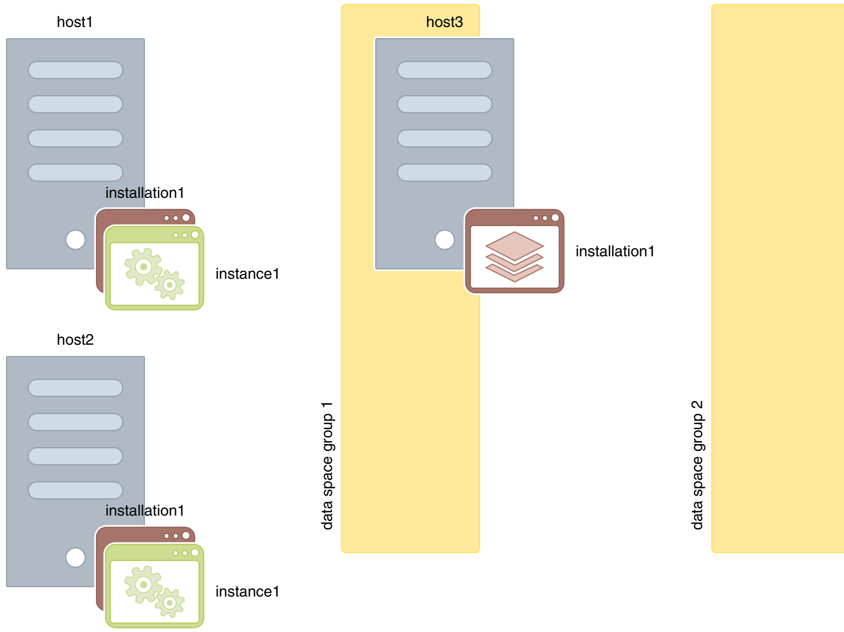
```
% ttGridAdmin installationCreate host3 -location /grid
Installation installation1 on Host host3 created in Model
```

Note:

- If you do not specify a name for the installation, TimesTen Scaleout sets `installation1` as the name of the installation. Any subsequent installation associated with the same host requires that you provide a name for it.
- If the management instance running the command has only one installation associated with it and the source for the installation files is not specified in the `-source` option of the `ttGridAdmin installationCreate` command, TimesTen Scaleout copies the installation files from the installation associated with the management instance running the command.

Figure 4–6 shows a graphical representation of the model of the `grid1` grid after creating the `installation1` installation in the `host3` host.

Figure 4–6 The model after adding an installation for the data instance



For more information on the `ttGridAdmin installationCreate` command, see "Create an installation (installationCreate)" in the *Oracle TimesTen In-Memory Database Reference*.

Create the data instance

The `ttGridAdmin instanceCreate` command creates an instance in the grid and associates it with a host and installation.

Create a data instance in the location of your choice in the `host3` host, for example, the `/grid` directory.

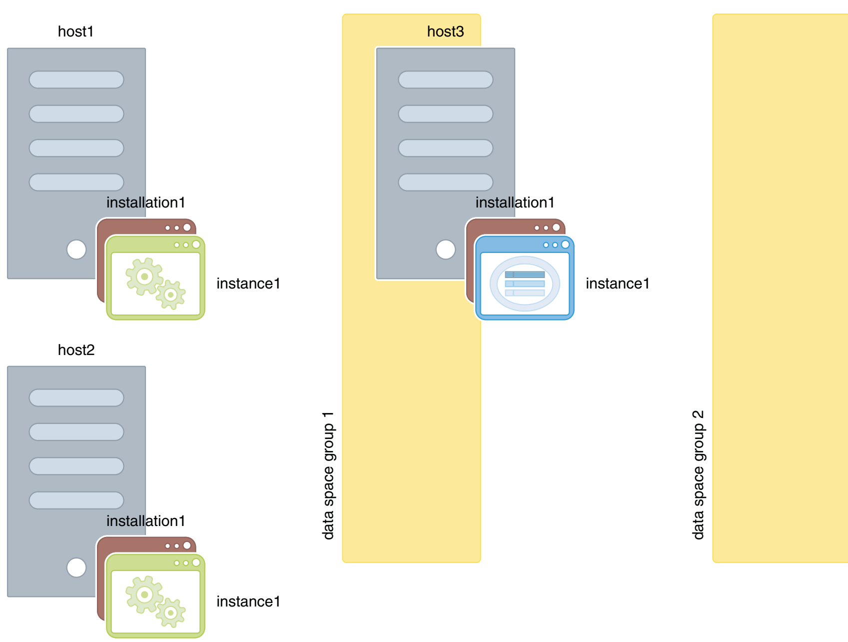
```
% ttGridAdmin instanceCreate host3 -location /grid
Instance instance1 on Host host3 created in Model
```

Note:

- If you do not specify a name for the instance, TimesTen Scaleout sets `instance1` as the name of the instance. Any subsequent instances associated with the same host requires that you provide a name for it.
- Because the `host3` host only has the `installation1` installation associated with it, the `installation1` installation is associated with the `instance1` data instance by default and there is no need to specify the `-installation` option.
- TimesTen Scaleout defines an instance as a data instance by default. Use the `-type` management option of the `ttGridAdmin instanceCreate` command to create a management instance.
- TimesTen Scaleout creates a subdirectory with the instance name in the specified location. TimesTen Scaleout allocates all instance files in this subdirectory. For example, the instance files of the `instance1` data instance are allocated in the `/grid/instance1` directory of the `host3` host.
- TimesTen Scaleout sets the default values for the TCP/IP port numbers of the instance daemon and server (`6624` and `6625`, respectively) if you do not specify a value for the port numbers. Use the `-daemonPort` or `-csPort` options of the `ttGridAdmin instanceCreate` utility to set different values for the port numbers.

Figure 4–7 shows a graphical representation of the model of the `grid1` grid after creating a data instance.

Figure 4–7 The model after adding a data instance



For more information on the `ttGridAdmin instanceCreate` command, see "Create an instance (instanceCreate)" in the *Oracle TimesTen In-Memory Database Reference*.

Create data instances by duplicating the configuration of an existing host

As mentioned in "Adding the standby management instance" on page 4-6, you can create an instance (management or data) by duplicating the configuration of an existing host, including its associated installations and instances with the `-like` and `-cascade` options of the `ttGridAdmin hostCreate` command.

- The `-like` option identifies the host to be duplicated and associates the new host with the same physical groups and data space group. You can override the physical groups and data space group associated with the new host by providing different parameters in the `-physicalGroup` and `-dataSpaceGroup` options, respectively.
- The `-cascade` option duplicates the configuration of the installations and data instances associated with the specified host.

Create five data instances based on the same attributes defined for the `host3` host and its associated installation and data instance. Also, associate three hosts with data space group `2`, instead of data space group `1`. Ensure that you identify the fully qualified domain name of the new hosts.

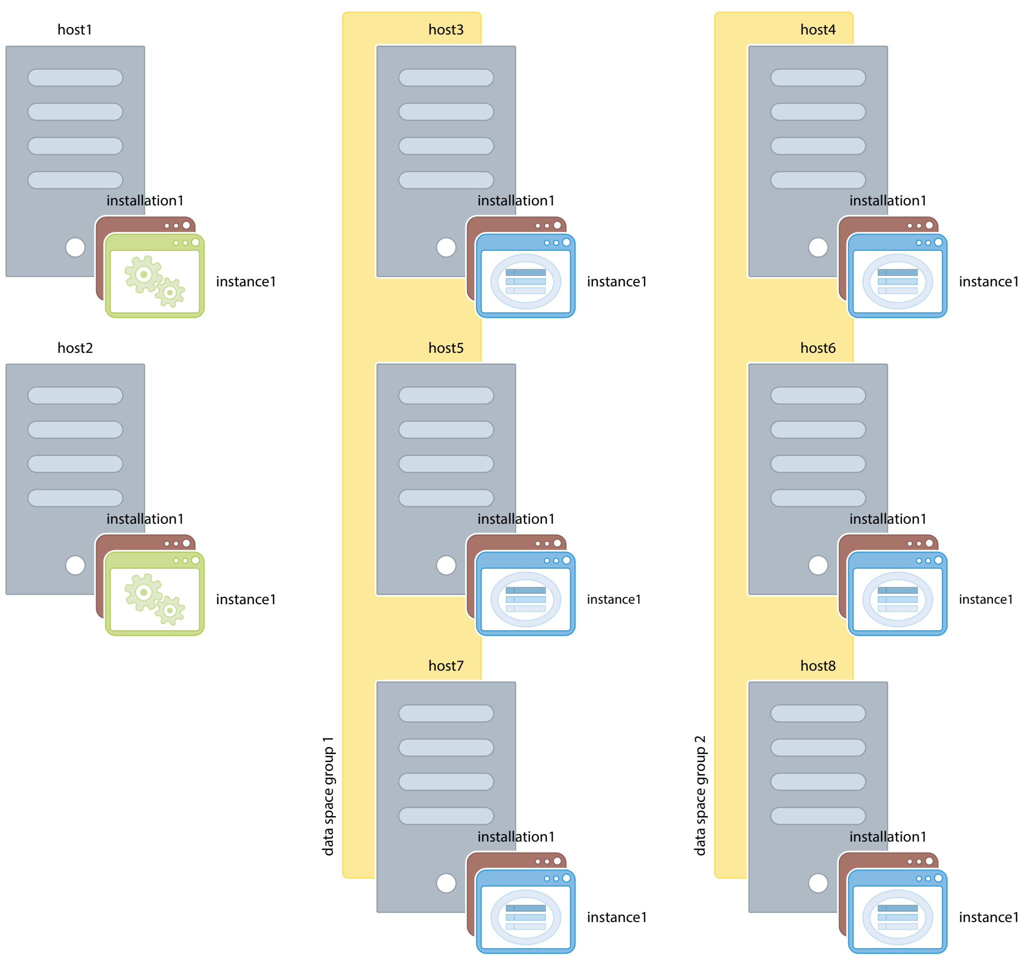
```
% ttGridAdmin hostCreate -internalAddress int-host4 -externalAddress
 ext-host4.example.com -like host3 -cascade -dataSpaceGroup 2
Host host4 created in Model
Installation installation1 created in Model
Instance instance1 created in Model

% ttGridAdmin hostCreate -internalAddress int-host5 -externalAddress
 ext-host5.example.com -like host3 -cascade
Host host5 created in Model
Installation installation1 created in Model
Instance instance1 created in Model

% ttGridAdmin hostCreate -internalAddress int-host6 -externalAddress
 ext-host6.example.com -like host4 -cascade
Host host6 created in Model
Installation installation1 created in Model
Instance instance1 created in Model

% ttGridAdmin hostCreate -internalAddress int-host7 -externalAddress
 ext-host7.example.com -like host3 -cascade
Host host7 created in Model
Installation installation1 created in Model
Instance instance1 created in Model

% ttGridAdmin hostCreate -internalAddress int-host8 -externalAddress
 ext-host8.example.com -like host4 -cascade
Host host8 created in Model
Installation installation1 created in Model
Instance instance1 created in Model
```

Note:

- If you do not specify a name for the host, TimesTen Scaleout sets the OS hostname of the remote system as the name of the new host.
- Any additional option you define in the `ttGridAdmin hostCreate` command will overwrite the attributes inherited from the existing host in the new host, as shown with the addition of the `-dataSpaceGroup 2` parameter in the command that creates the `host6` host.
- Notice that the `ttGridAdmin hostCreate` commands that create the `host6` and `host8` hosts use the `host4` host as reference in the `-like` option.

Figure 4–8 shows a graphical representation of the model of the `grid1` grid after duplicating the `host3` host as the `host4`, `host5`, `host6`, `host7`, and `host8` hosts and their associated installations and instances.

Figure 4–8 The model after duplicating an existing host



For more information on the `ttGridAdmin hostCreate` command, see "Create a host (hostCreate)" in the *Oracle TimesTen In-Memory Database Reference*.

Applying the changes made to the model

The latest version of the model describes the desired structure of a grid, not its current structure. Any changes made to the latest version of the model are not immediately reflected in the operational configuration of a grid. Changes made to the latest version of the model need to be explicitly applied to the grid.

Model versioning

Management instances store multiple versions of the model. Only one version of the model can be active in the grid at any given time.

TimesTen Scaleout classifies model versions as follows:

- **Current version:** The current version of the model describes the operational configuration of the grid. This version, and all previous versions, is read-only.
- **Latest version:** The latest version of the model can be modified and has yet to be applied to the grid. This version is read/write.

When you create a grid, the `version 1` model is initially populated with the configuration of the first host, installation, and management instance, and the `version 1` model is recognized as the latest version of the model. Any subsequent changes that you make to the model are added to the latest version of the model (`version 1`). When you implement these changes with the `ttGridAmin modelApply` command, a new latest version of the model (`version 2`) is created for future changes and the previous latest version of the model (`version 1`) becomes the current version of the model.

Every time you run the `ttGridAdmin modelApply` command, TimesTen Scaleout:

1. Makes the latest version of the model (`version` *n*) read-only.
2. Creates a writable copy (`version` *n+1*) of the latest version of the model.
3. Attempts to apply the changes previously made to the `version` *n* model to the operational grid.
4. Identifies the `version` *n* model as the current version of the model.
5. Identifies the `version` *n+1* model as the latest version of the model.

The `ttGridAdmin` utility enables the user to perform several operations regarding the model, like:

- Applying the changes made to the latest version of the model
- Comparing two versions of the model
- Exporting a version of the model into a flat file in JSON format
- Importing a flat file in JSON format as the latest version of the model
- Listing all the available versions of the model

Previous versions of the model are automatically stored. With the `ttGridAdmin gridModify` command, you can specify the retention period for old versions of the model either in terms of days, in terms of the number of stored versions, or both. TimesTen Scaleout by default retains the last 10 versions for a period of 30 days.

For more information on model operations or the `ttGridAdmin gridModify` utility, see "Model operations" or "Modify grid settings (gridModify)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Apply the latest version of the model

The `ttGridAdmin modelApply` command attempts to apply the changes made to the latest version of the model into the operational grid. If, for example, you add a new data instance to the latest version of the model, running this command performs all of the necessary operations to create and initialize the instance in the specified host. Some of the operations that the `ttGridAdmin modelApply` command performs include these:

- Identify and delete any object removed from the latest version of the model.
- Create new installations.
- Create new instances, data and management.
- Overwrite the configuration files of all instances. The new versions of these files include any new entries found in the latest version of the model.
- Verify the SSH connectivity between hosts.

> **Note:** If you recently added new instances to the model or have yet to set the required passwordless SSH access to the hosts managing instances, either manually set the required passwordless SSH access for the instance administrator or use the `ttGridAdmin gridSshConfig` command before applying the latest version of the model. See "Setting passwordless SSH" on page 2-12 for more information.

- Start new instances.

Apply all the changes made to the latest version of the model of the `grid1` grid.

```
% ttGridAdmin modelApply
Creating new model version............................................OK
Exporting current model (version 1)...................................OK
Identifying any changed management instances..........................OK
Identifying any deleted objects.......................................OK
Verifying installations...............................................OK
Verifying instances...................................................OK
Creating new instances................................................OK
Updating grid state...................................................OK
Configuring instance authentication...................................OK
Pushing new configuration files to each instance.....................OK
Making model version 1 current, version 2 writable...................OK
Checking ssh connectivity of new instances...........................OK
Starting new management instance.....................................OK
Configuring standby management instance..............................OK
Starting new data instances..........................................OK
ttGridAdmin modelApply complete
```

Given all the tasks you performed in the previous sections, the `ttGridAdmin modelApply` command performs the following operations:

1. Creates a copy of the installation files on every configured host:
2. Creates the instance home directory and files for the standby management instance and data instances on their associated hosts:
3. Makes the latest version of the model read-only and a creates a new writable model.
4. Verifies SSH connectivity to every configured host.
5. Starts the daemons of the standby management instance and data instances.

6. Configures the active and standby management instances.

For more information on the `ttGridAdmin modelApply` command, see "Apply the latest version of the model (modelApply)" in the *Oracle TimesTen In-Memory Database Reference*.

Setting instances to automatically start at system startup

Optionally, you can configure data instances to automatically start or shut down every time their systems boot or shut down, respectively. Each instance needs to be configured independently and only after its creation has been applied to the current version of the model.

To accomplish this, the `root` user must run the `setuproot` script with the `-install` option. You can find this script in the *`timesten_home`*`/bin` directory of every instance of the grid. For example:

From the host of the `host3.instance1` data instance:

```
% /grid/instance1/bin/setuproot -install
Would you like to install the TimesTen daemon startup scripts into /etc/init.d?
 [ yes ]
```

For more information on how to use the `setuproot` script, see "Start an instance automatically at system startup" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

Description of the physical topography of the grid

As described in "Assigning hosts to data space groups" on page 1-11, TimesTen Scaleout requires you to assign the hosts that will contain a data instance into a data space group.

Note: The hosts in one data space group should not share physical resources with the hosts in another data space group to protect each full copy of the database from hardware failures.

The example provided in "Adding data instances" on page 4-11, demonstrates how to assign hosts to a data space group during host creation with the `ttGridAdmin hostCreate -dataSpaceGroup` option:

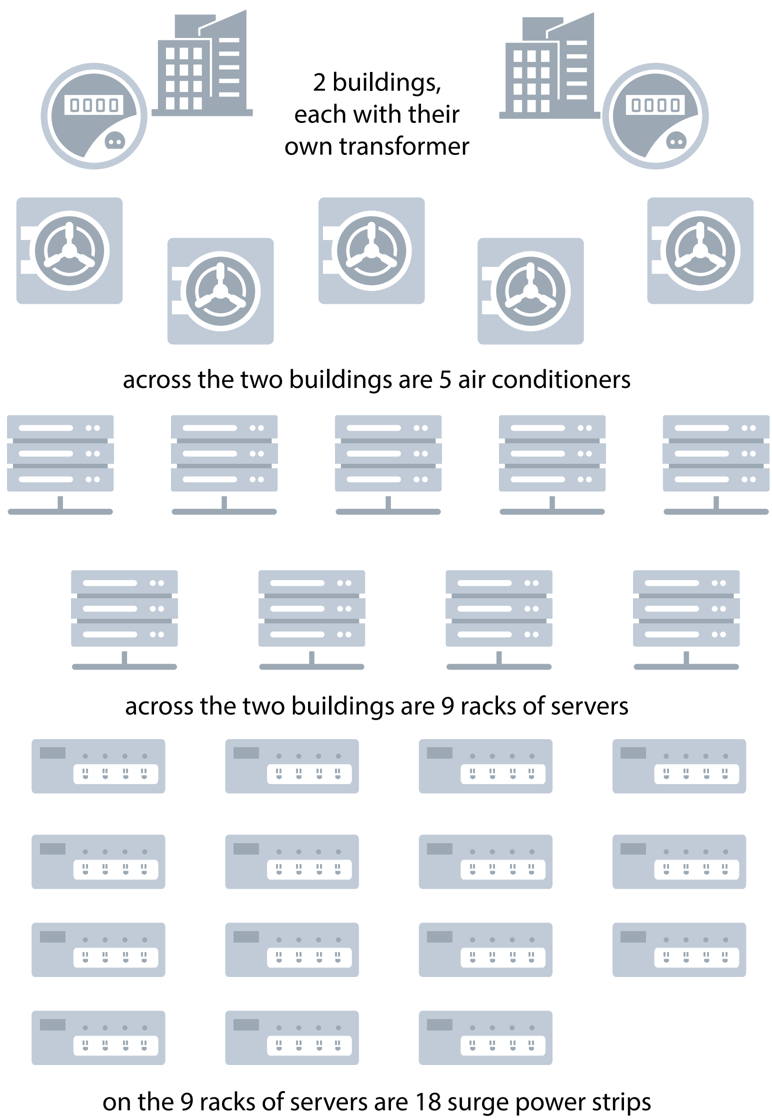
```
% ttGridAdmin hostCreate -internalAddress int-host3 -externalAddress
 ext-host3.example.com -dataSpaceGroup 1
Host host3 created in Model
```

However, if your physical topography is complicated so that it is difficult to decide on the best assignments of hosts to data space groups, you can ask TimesTen Scaleout to recommend assignments of hosts to data space groups using the `ttGridAdmin dataSpaceGroupSuggest` command. In order to accomplish this, you need to inform TimesTen Scaleout of the physical topology of where your hosts are co-located or hosts that use the same resources. You can inform TimesTen Scaleout of all physical hardware for each host with the physical group.

Important: If you are going to use physical groups to have TimesTen Scaleout recommend assignments, you should not assign hosts to data space groups upon host creation. Hosts can only be assigned to data space groups once.

Your physical topology may include buildings, transformers, air conditioners, racks, fans, top of rack switches, hypervisors, storage filers and power sources. For example, the following production environment includes:

- 2 buildings, each with their own transformer.
- 5 air conditioners in rooms within the 2 buildings.
- 9 racks of servers in these rooms, each with several hosts.
- 18 power surge protectors in which the hosts are plugged.



Instead of determining what would be the best data space group assignment for the hosts that share these resources, you can assign hosts to physical groups to identify where the hosts are located and what physical resources they share. This informs

TimesTen Scaleout of similar potential points of failure. Thus, when TimesTen Scaleout recommends assignment of hosts to data space groups using the `ttGridAdmin dataSpaceGroupSuggest` command, it physically separates the hosts that contain the separate copies of the data.

Note: See "Assigning hosts to data space groups" on page 4-10 for more details on data space groups. See "Get recommendations for data space group assignments (dataSpaceGroupSuggest)" in the *Oracle TimesTen In-Memory Database Reference* for more details on the `ttGridAdmin dataSpaceGroupSuggest` command.

Create and manage the physical topography of your grid by:

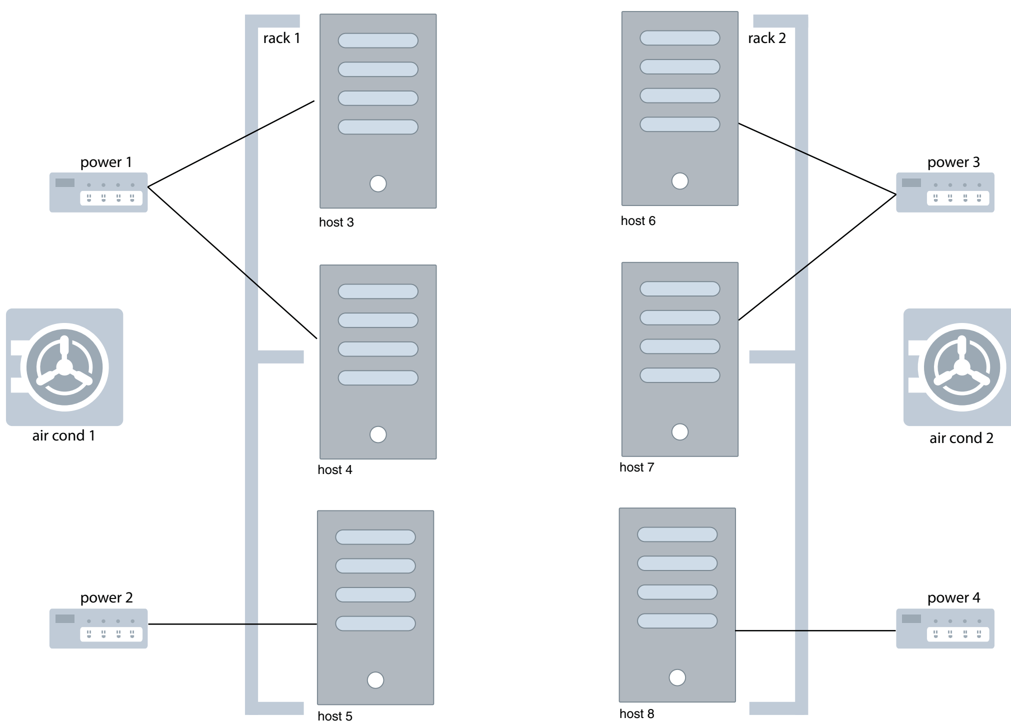
- Assigning hosts to physical groups
- Propose data space group assignments

Assigning hosts to physical groups

Assigning hosts to physical groups to denote the physical topology is optional and only necessary when you want TimesTen Scaleout to assign hosts to data space groups.

The example in this section assigns the hosts to physical groups using the same host configuration of the example used in this chapter. In production, you would not use physical groups for this small example as you can easily identify which hosts should be in which data space group.

Figure 4–9 Hosts attached to physical resources



Let us assume that:

- Hosts 3, 4, and 5 share an air conditioner.

- Hosts 3, 4 and 5 are on the same rack.
- Hosts 3 and 4 share a power source.
- Host 5 has its on power source.
- Hosts 6, 7 and 8 share an air conditioner.
- Hosts 6, 7 and 8 are on the same rack.
- Hosts 6 and 7 share a power source.
- Host 8 has its own power source.

To identify which hosts are connected to which physical resources:

1. Define a physical group, which is simply a container object that groups hosts together to show that there is a shared physical resource. This enables TimesTen Scaleout to separate different copies of the data to hosts that do not share physical aspects.
2. Assign hosts to physical groups either when creating the host with the `ttGridAdmin hostCreate` command or when modifying the host with the `ttGridAdmin hostModify` command.

   You can assign a host to one or more physical groups, as appropriate. If they are assigned to a similar rack and share a power source, you can assign the host to both of these physical groups.

See "Create a host (hostCreate)" and "Modify a host (hostModify)" in the *Oracle TimesTen In-Memory Database Reference* for more details on these `ttGridAdmin` commands.

Example 4–1 Example of hosts that are physically located on two separate racks

This example shows how you would create physical groups to model the shared resources and assign hosts to these physical groups.

1. Define the physical group: Use the `ttGridAdmin physicalCreate` command to create:
   - `rack1` and `rack2` to model the physical racks in the lab.
   - `aircond1` and `aircond2` to model the air conditioners.
   - `power1`, `power2`, `power3` and `power4` to model the power surge protectors.
2. Assign hosts to the physical group: Use the `ttGridAdmin hostModify` to assign the hosts to each shared physical resource using the `-physicalgroup` option:
   - `host3`, `host4` and `host5` are on the `rack1`.
   - `host3`, `host4` and `host5` share `aircond1`.
   - `host3` and `host4` share `power1`.
   - `host5` has its own `power2`.
   - `host6`, `host7` and `host8` are on `rack2`.
   - `host6`, `host7` and `host8` share an `aircond2`.
   - `host6` and `host7` share `power3`.
   - `host8` has its own `power4`.

Note: If a host was already associated with a physical group before you execute the `-physicalgroup` option of the `ttGridAdmin hostModify` command, these physical groups are removed and the only physical groups associated with the host are those that are specified on the current command line.

However, you can add physical groups in addition to those physical groups that are already assigned to the existing host by using the `-addPhysicalGroup` option on the `ttGridAdmin hostModify` command.

```
% ttGridAdmin physicalCreate rack1
 PhysicalGroup rack1 created.
% ttGridAdmin physicalCreate rack2
 PhysicalGroup rack2 created.
% ttGridAdmin physicalCreate aircond1
 PhysicalGroup aircond1 created.
% ttGridAdmin physicalCreate aircond2
 PhysicalGroup aircond2 created.
% ttGridAdmin physicalCreate power1
 PhysicalGroup power1 created.
% ttGridAdmin physicalCreate power2
 PhysicalGroup power2 created.
% ttGridAdmin physicalCreate power3
 PhysicalGroup power3 created.
% ttGridAdmin physicalCreate power4
 PhysicalGroup power4 created.
% ttGridAdmin hostModify host3 -physicalgroup rack1 aircond1 power1
 Host host3 modified in Model
% ttGridAdmin hostModify host4 -physicalgroup rack1 aircond1 power1
 Host host4 modified in Model
% ttGridAdmin hostModify host5 -physicalgroup rack1 aircond1 power2
 Host host5 modified in Model
% ttGridAdmin hostModify host6 -physicalgroup rack2 aircond2 power3
 Host host6 modified in Model
% ttGridAdmin hostModify host7 -physicalgroup rack2 aircond2 power3
 Host host7 modified in Model
% ttGridAdmin hostModify host8 -physicalgroup rack2 aircond2 power4
 Host host8 modified in Model
```

Note: You can use the `ttGridAdmin modelExport` command to see which hosts are assigned to a physical group.

Removing the physical layout of the hosts

If you want to remove a physical description of how the hosts are organized physically, use the `-nophysicalgroup` option with the `ttGridAdmin hostModify` command. The `host3` host was originally associated with the physical groups `rack1`, `aircond1` and `power1`. By executing the following command, `host3` is not assigned to any physical groups.

```
% ttGridAdmin hostModify host3 -nophysicalgroup
 Host host3 modified in Model
```

Deleting physical groups

You can delete a physical group only if no hosts are assigned to it. The following deletes the `rack1` physical group after removing the `host1` and `host2` hosts assigned to it:

```
% ttGridAdmin hostModify host3 -nophysicalgroup
 Host host3 modified in Model
% ttGridAdmin hostModify host4 -nophysicalgroup
 Host host4 modified in Model
% ttGridAdmin hostModify host5 -nophysicalgroup
 Host host5 modified in Model
% ttGridAdmin physicalDelete rack1
 PhysicalGroup rack1 deleted.
```

Propose data space group assignments

You can use the `ttGridAdmin dataSpaceGroupSuggest` command to propose an assignment of hosts to different data space groups based on how the hosts are currently assigned to physical groups. You can either accept the proposed assignments or specify your own assignments.

The `ttGridAdmin dataSpaceGroupSuggest` command makes recommendations on how to assign hosts so that the hosts that contain one copy of the data do not share resources with the hosts that contain the copy of the data. Once the `ttGridAdmin` command assigns the hosts to their data space groups, the assignment cannot be changed.

See "Get recommendations for data space group assignments (dataSpaceGroupSuggest)" in the *Oracle TimesTen In-Memory Database Reference* for more details on the `ttGridAdmin dataSpaceGroupSuggest` command.

Example 4–2 Requesting TimesTen Scaleout assign hosts to data spaces

This example shows `ttGridAdmin dataSpaceGroupSuggest` command, which writes its recommendations into the `recommendations.sh` file. You can execute this file as it contains the `ttGridAdmin hostModify` commands necessary for assigning hosts to the recommended data space groups.

```
% ttGridAdmin dataSpaceGroupSuggest /tmp/recommendations.sh
% more /tmp/recommendations.sh
#!/bin/sh
# Recommendations generated by ttGridAdmin -dataSpaceGroupSuggest

TIMESTEN_HOME=/grid/instance1
export TIMESTEN_HOME
. $TIMESTEN_HOME/bin/ttenv.sh > /dev/null 2>/dev/null

# Number of possibilities evaluated:     126
#
# Number of usable possibilities found:  10
# (A 'usable' possibility is one that is compatible with pre-existing
# assignments of Hosts to DataSpaceGroups)
#
# Number of 'ideal' possibilities found: 1
# (An 'ideal' possibility is one where no PhysicalGroups span multiple
# DataSpaceGroups)
#
# Possibilities evaluated (best 10 displayed):
# ...
#
```

```
# This script, if executed, would implement the only 'ideal' configuration found.
# Even though this recommendation was 'ideal', you should carefully evaluate it
# prior to running this script.
# Host host1 is already in DataSpaceGroup 1
ttGridAdmin hostModify host3 -dataSpaceGroup 2
ttGridAdmin hostModify host4 -dataSpaceGroup 2
ttGridAdmin hostModify host5 -dataSpaceGroup 2
ttGridAdmin hostModify host6 -dataSpaceGroup 1
ttGridAdmin hostModify host7 -dataSpaceGroup 1
ttGridAdmin hostModify host8 -dataSpaceGroup 1
```

If you decide to accept these recommendations, execute the provided shell script, which in our example is `recommendations.sh`. Once executed, all hosts are assigned to the designated data space groups.

```
% sh /tmp/recommendations.sh
Host host3 modified in Model
Host host4 modified in Model
Host host5 modified in Model
Host host6 modified in Model
Host host7 modified in Model
Host host8 modified in Model
```

5

Managing a Database

This chapter discusses how to create and configure a database in TimesTen Scaleout.

Note:

- These tasks assume that you have already created and configured a grid. See "Configure your grid" on page 4-1 for more information on how to set up a grid and the grid scenario on which the examples in this chapter are based.
- Run the commands provided in the examples from the active management instance, unless stated otherwise. For more information on how to set the environment variables for the active management instance, see "Creating the initial management instance" on page 4-2 or "Environment variables" on page B-1.

Creating a database

The process of creating a database involves these tasks:

- Create a database definition
- Create a database based on the database definition
- Define the distribution map of the database
- Open the database for user connections

Create a database definition

A database definition contains the description of a database. It defines the database name, as well as the attributes of the database. Once a database definition is added to the current version of the model, it can be used to create a database. Each database has one or more connectables associated with it. Connectables specify how applications connect to the database. Connectables are discussed in "Connecting to a database" on page 5-7.

Creating a database definition file

To create a database definition, you need a database definition file. The database definition file must use `.dbdef` as the file name suffix. The name of the database definition derives from the name of the database definition file. For example, a database definition file named `database1.dbdef` creates a database definition named `database1`.

Note: Database definition names have the same restrictions as Data Source Names. See "Specifying Data Source Names to identify TimesTen databases" in the *Oracle TimesTen In-Memory Database Operations Guide* for more information.

In the database definition file, you specify the connection attributes for the database. The types of connection attributes that a database definition supports are:

- **Data store attributes** are associated with a database when it is created. They can only be modified by recreating the database.

  The most commonly used data store attributes are:

  – `DataStore`: Defines the full path and file name prefix of the checkpoint files for every element of the database. *Required*.

  – `LogDir`: Defines the file system directory of the transaction log files for every element of the database.

  – `DatabaseCharacterSet`: Defines the character set to be used by the database. *Required*.

  – `Durability:` Defines the degree of durability for transactions.

  Note: Ensure that you set the appropriate durability setting based on your business needs and data loss tolerance. See "Durability settings" on page 6-3 for more information.

- **First connection attributes** are associated with a database when it is loaded into memory. They can only be modified when the database is unloaded from memory and reloaded with different values for the first connection attributes.

  The most commonly used first connection attributes are:

  – `PermSize`: Defines the allocated size of the permanent memory region of each element of the database. The permanent memory region contains persistent database objects, such as tables. TimesTen Scaleout only writes the contents of the permanent memory region to the file system.

– `TempSize`: Defines the allocated size of the temporary memory region of each element of the database. The temporary memory region contains the transient data generated when executing a statement.

> **Note:** Each host must have sufficient main memory to accommodate as many elements of the database as data instances associated with the host. For more details on setting region sizes, see "Determining the value of the PermSize attribute" on page 5-19 in this document and "Specifying the memory region sizes of a database" and "Storage provisioning for TimesTen" in the *Oracle TimesTen In-Memory Database Operations Guide*.

- **PL/SQL first connection attributes** define the behavior of a database regarding PL/SQL operations and are associated with the database when it is loaded into memory. They can only be modified when the database is unloaded from memory and reloaded with different values for the PL/SQL first connection attributes.
- **Server connection attributes** define the behavior of the database regarding connections and are associated with the database when it is loaded into memory. They can only be modified when the database is unloaded from memory and reloaded with different values for the server connection attributes.

> **Note:** TimesTen Scaleout adds any connection attribute in the database definition file that is not a data store, first connection, PL/SQL first connection, or server connection attribute to a connectable that TimesTen Scaleout creates by default. See "Create a connectable" on page 5-7 for more information.

Create a database definition file as shown in Example 5–1.

Example 5–1 Database definition file

The following example creates a database definition file named `database1.dbdef` that defines:

- The full path for the checkpoint files as `/disk1/databases/database1`
- The directory for the log files as `/disk2/logs`
- The database character set as `AL32UTF8`
- The durability setting as `0`.
- 32 GB for the permanent memory region of every element
- 4 GB for the temporary memory region of every element
- 1 GB for the internal transaction log buffer of every element
- An upper limit of 2048 user-specified concurrent connections to the database

```
vi /mydir/database1.dbdef

DataStore=/disk1/databases/database1
LogDir=/disk2/logs
DatabaseCharacterSet=AL32UTF8
Durability=0
PermSize=32768
TempSize=4096
```

```
LogBufMB=1024
Connections=2048
```

For a complete description of all the connection attributes, see "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

Adding a database definition to the model

The `ttGridAdmin dbdefCreate` command creates a database definition based on a database definition file. TimesTen Scaleout uses the name of the database definition file to name the database definition.

Create the `database1` database definition based on the `database1.dbdef` file.

```
% ttGridAdmin dbdefCreate /mydir/database1.dbdef
Database Definition database1 created.
```

The `ttGridAdmin dbdefCreate` command also creates a connectable of the same name, which includes any general connection attribute found in the database definition file. Considering that the `database1.dbdef` file in Example 5–1 includes no general connection attribute, the `database1` connectable contains no attributes. This connectable is always set for direct connections only.

Add the `database1` database definition to the current version of the model.

```
% ttGridAdmin modelApply
...
Updating grid state.................................................OK
Pushing new configuration files to each instance.....................OK
...
ttGridAdmin modelApply complete
```

TimesTen Scaleout adds a `database1` connectable to the configuration files of every data instance based on the attributes defined in the `database1` database definition.

> **Note:** TimesTen Scaleout overwrites the configuration files every time you apply the changes made to the latest version of the model to the operational grid. For this reason, you must refrain from modifying these files without the assistance of the `ttGridAdmin` utility.

For more information on the `ttGridAdmin dbdefCreate` or `ttGridAdmin modelApply` command, see "Create a database definition (dbdefCreate)" in the *Oracle TimesTen In-Memory Database Reference* or "Applying the changes made to the model" on page 4-18 in this document, respectively.

Create a database based on the database definition

In TimesTen Scaleout, user data is distributed to a set of elements that form a database. Each data instance in the current version of the model contains one element of every user database in the grid.

You can create a database based on the attributes stored in a database definition. On database creation, every data instance creates an element of the database and loads it into memory.

The process of creating an element of the database on every data instance is asynchronous. The daemon of each data instance performs the operations necessary to create and load the element into memory independently, as soon as it realizes that there is a new database flagged for creation.

The `ttGridAdmin dbCreate` command creates a database based on a database definition.

Create the `database1` database based on the `database1` database definition.

```
% ttGridAdmin dbCreate database1
Database database1 creation started
```

Wait until all data instances report that they have loaded their element of the database into memory before proceeding with the definition of the distribution map. You can verify the status of the database creation process with the `ttGridAdmin dbStatus` command as shown in Example 5–2.

Example 5–2 Verifying the status of the database creation process

The example shows a status summary for the `database1` database. Notice that the report shows all elements of the database as loaded.

```
% ttGridAdmin dbStatus database1 -element
Database database1 element level status as of Wed Jan 10 14:34:08 PST 2018

Host  Instance  Elem Status Date/Time of Event  Message
----- --------- ---- ------ ------------------- -------
host3 instance1    1 loaded 2018-01-10 14:33:23
host4 instance1    2 loaded 2018-01-10 14:33:21
host5 instance1    3 loaded 2018-01-10 14:33:23
host6 instance1    4 loaded 2018-01-10 14:33:23
host7 instance1    5 loaded 2018-01-10 14:33:23
host8 instance1    6 loaded 2018-01-10 14:33:23
```

For more information on the `ttGridAdmin dbCreate` or `ttGridAdmin dbStatus` command, see "Create a database (dbCreate)" or "Monitor the status of a database (dbStatus)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Define the distribution map of the database

TimesTen Scaleout allows for elastic scalability. You can increase or reduce the number of elements in your database according to your business needs. When you add new data instances to a grid, TimesTen Scaleout does not automatically re-distribute the data stored in the database across the elements of the new or remaining instances. The way the data is distributed in TimesTen Scaleout is defined by the data space group associated to each host in the grid and the elements of the data instances defined in the distribution map of the database.

> **Note:** TimesTen Scaleout blocks DDL and DML statements during operations that change the distribution map of the database. Ensure that you make changes to the distribution map while there are no open transactions, such as during a maintenance period or scheduled outage.

The `ttGridAdmin dbDistribute` command with the `-add` option adds the element of a data instance to the distribution map of a database. Using `all` as the parameter for the `-add` option adds the elements of all the available data instances in the grid. The `all` parameter is typically used for the initial definition of the distribution map of a new database.

Add all the elements of the available data instances in the `grid1` grid to the distribution map of the `database1` database.

```
% ttGridAdmin dbDistribute database1 -add all -apply
Distribution map updated
```

For more information on the `ttGridAdmin dbDistribute` command, see "Set or modify the distribution scheme of a database (dbDistribute)" in the *Oracle TimesTen In-Memory Database Reference*.

Open the database for user connections

For an application to be able to connect to a database, the database needs to be open for user connections. As with the database creation process, the process of opening elements is asynchronous. The daemon of every data instance performs the operations necessary to open its element as soon as it realizes that the database is flagged for opening.

Note:

- The instance administrator can connect to the database without it being open for user connections.
- Before you open the database to user connections, you may want to create your database users. See "Creating or identifying a database user" in the *Oracle TimesTen In-Memory Database Security Guide*.

  Also, you may want to have the SQL schema defined which includes the distribution scheme of each table, as shown in "Defining table distribution schemes" on page 5-13.

The `ttGridAdmin dbOpen` command opens a database for user connections.

Open the `database1` database for user connections.

```
% ttGridAdmin dbOpen database1
Database database1 open started
```

You can verify the status of the database opening process with the `ttGridAdmin dbStatus` command as shown in Example 5–3.

Example 5–3 Verifying the status of the database opening process

The example shows a status summary for the `database1` database. Notice that the report shows all elements of the database as open.

```
% ttGridAdmin dbStatus database1 -element
Database database1 element level status as of Wed Jan 10 14:34:43 PST 2018

Host  Instance  Elem Status Date/Time of Event  Message
----- --------- ---- ------ ------------------- -------
host3 instance1    1 opened 2018-01-10 14:34:43
host4 instance1    2 opened 2018-01-10 14:34:43
host5 instance1    3 opened 2018-01-10 14:34:42
host6 instance1    4 opened 2018-01-10 14:34:42
host7 instance1    5 opened 2018-01-10 14:34:42
host8 instance1    6 opened 2018-01-10 14:34:42
```

For more information on the `ttGridAdmin dbOpen` or `ttGridAdmin dbStatus` command, see "Open a database (dbOpen)" or "Monitor the status of a database (dbStatus)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Connecting to a database

To be able to connect to a database, every element of the database needs to be created, loaded into memory, added to the distribution map, and opened for user connections. All these operations are covered in the previous section, "Creating a database" on page 5-1.

A connectable defines a name that applications can use to connect to a database. The connectable may have the same name as the database or may have a different name. There are two types of connectables:

- **Direct connectable:** Defines a name by which applications may connect to a database through direct communication.
- **Client/server connectable:** Defines a name by which applications may connect to a database through client/server communication.

TimesTen Scaleout enables you to create multiple connectables with different sets of connection attributes defined for a single database.

Connectables support these types of connection attributes:

- **General connection attributes** are set by each connection and persist for the duration of that connection.
- **NLS general connection attributes** define the connection-specific behavior of the database regarding globalization.
- **PL/SQL general connection attributes** define the connection-wise behavior of the database regarding PL/SQL operations.
- **TimesTen Client connection attributes** define the connection parameters for client/server connections.

For a complete description of all the connection attributes, see "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

Create a connectable

TimesTen Scaleout creates a direct connectable by default for every database definition added to the grid, this connectable enables applications to create direct connections to the database from any data instance in the distribution map of the database. TimesTen Scaleout uses the name assigned to the database definition to name the connectable. You need to create a client/server connectable to establish client/server connections to a database.

The tasks to create a connectable are:

- Creating a connectable file
- Creating a connectable based on the connectable file

Creating a connectable file

A connectable file specifies the attributes to use to connect to a database. The connectable file must use `.connect` as file name suffix. The file name prefix of the connectable file sets the name of the connectable. For example, a connectable file named `database1CS.connect` creates a connectable named `database1cs`.

Note: Connectable names have the same restrictions as Data Source Names. See "Specifying Data Source Names to identify TimesTen databases" in the *Oracle TimesTen In-Memory Database Operations Guide* for more information.

Create a client/server connectable file with the connection attributes for the `database1` database as the one shown in Example 5–4.

Example 5–4 Connectable file

The example shows the contents of a connectable file named `database1CS.connect` that sets `AL32UTF8` as the connection character set and `terry` as the user ID for the connection.

```
ConnectionCharacterSet=AL32UTF8
UID=terry
```

Note: If you do not provide a user ID, TimesTen utilizes the OS user ID of the user that sends the connection request as the `UID`. In this case, connection requests coming from systems other than the localhost fail since the OS user ID cannot be authenticated. See "UID and PWD" in the *Oracle TimesTen In-Memory Database Reference* and "Authentication in TimesTen" in the *Oracle TimesTen In-Memory Database Security Guide* for more information.

Creating a connectable based on the connectable file

The `ttGridAdmin connectableCreate` command creates a connectable based on a connectable file.

Create the `database1CS` connectable based on the `database1CS.connect` connectable file.

```
% ttGridAdmin connectableCreate -dbdef database1 -cs /mydir/database1CS.connect
Connectable database1CS created.
```

Note:

- The `-cs` option enables the connectable for client connections instead of direct connections.
- Use the `-only` option to establish client connections only to the element of the specified data instance.

Apply the creation of the `database1CS` connectable to the current version of the model to make the connectable available for use.

```
% ttGridAdmin modelApply
...
Updating grid state..................................................OK
Pushing new configuration files to each instance.....................OK
...
ttGridAdmin modelApply complete
```

For more information on the `ttGridAdmin connectableCreate` or `ttGridAdmin modelApply` command, see "Create a connectable (connectableCreate)" in the *Oracle*

TimesTen In-Memory Database Reference or "Applying the changes made to the model" on page 4-18 in this document, respectively.

Connect to a database using ODBC and JDBC drivers

Applications can use the ODBC direct driver, the ODBC client driver, or an ODBC driver manager to connect to a database. See "Connecting to TimesTen with ODBC and JDBC drivers" in the *Oracle TimesTen In-Memory Database Operations Guide* for more information.

The following topics discuss how to use those DSNs to establish direct and client connections to a database:

- Establishing direct connections from a data instance
- Establishing client connections from a TimesTen Client
- Redirecting client connections

Establishing direct connections from a data instance

TimesTen Scaleout automatically creates a direct connectable that includes any general connection attribute included in the database definition file. TimesTen Scaleout uses the name of the database definition to name the connectable. When the connectable is applied to the current version of the model, TimesTen Scaleout defines a DSN in every data instance with the same name as the connectable. This allows ODBC and JDBC applications to connect to the database associated with the connectable.

You may use the `ttIsql` utility from a data instance to establish direct connections to a database.

From the `host3.instance1` data instance, connect to the `database1` database using the `database1` connectable.

```
% ttIsql -connStr "DSN=database1"

Copyright (c) 1996, 2016, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=database1";
Connection successful: DSN=database1;UID=pat;DataStore=/disk1/databases/database1;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;LogDir=/disk2/logs;
PermSize=32768;TempSize=4096;TypeMode=0;
(Default setting AutoCommit=1)
Command>
```

Note: The example connects to the database as the instance administrator, which is defined for all instances (data and management) of the `grid1` grid. For more information on database users, see "Overview of TimesTen users" in the *Oracle TimesTen In-Memory Database Security Guide*.

For more information on the `ttIsql` utility, see "Using the ttIsql Utility" in the *Oracle TimesTen In-Memory Database Operations Guide*.

Establishing client connections from a TimesTen Client

A client/server connectable enables all data instances to accept connections from a TimesTen Client instance or applications using the TimesTen Client driver. However, to establish a client connection from a TimesTen Client that is not part of the grid, you have to create a client DSN in the system or user `odbc.ini` file of the TimesTen Client.

The `ttGridAdmin gridClientExport` command exports every client/server connectable available for the grid into a file that is formatted to replace the system or user `odbc.ini` file used by the TimesTen Client.

Export the client/server connectables of the `grid1` grid into a file.

```
% ttGridAdmin gridClientExport /mydir/sys.odbc.ini
```

Example 5–5 shows the content of the resulting file.

Example 5–5 Exported odbc.ini file

```
[ODBC Data Sources]
database1CS=TimesTen 18.1.4 Client Driver

[database1CS]
TTC_SERVER_DSN=DATABASE1
# External address/port info for host3.instance1
TTC_SERVER1=host3.example.com/6625
# External address/port info for host4.instance1
TTC_SERVER2=host4.example.com/6625
# External address/port info for host5.instance1
TTC_SERVER3=host5.example.com/6625
# External address/port info for host6.instance1
TTC_SERVER4=host6.example.com/6625
# External address/port info for host7.instance1
TTC_SERVER5=host7.example.com/6625
# External address/port info for host8.instance1
TTC_SERVER6=host8.example.com/6625
ConnectionCharacterSet=AL32UTF8
UID=terry
```

For more information on the `ttGridAdmin gridClientExport` command, see "Export sys.odbc.ini for client/server connections outside grid (gridClientExport)" in the *Oracle TimesTen In-Memory Database Reference*.

Adding a client DSN to a TimesTen Client on Linux or UNIX To add a client DSN to a TimesTen Client on Linux or UNIX, either replace the system or user `odbc.ini` file of the TimesTen Client with the file you just created, or copy the contents of the file into the system or user `odbc.ini` file. Then, from the TimesTen Client, connect to the `database1` database using the `database1CS` DSN with the `ttIsqlCS` utility.

```
% ttIsqlCS -connStr "DSN=database1CS"

Copyright (c) 1996, 2016, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=database1CS;UID=terry;
Enter password for 'terry':
Connection successful: DSN=database1CS;TTC_SERVER=host3.example.com;
TTC_SERVER_DSN=DATABASE1;UID=terry;DATASTORE=/disk1/databases/database1;
DATABASECHARACTERSET=AL32UTF8;CONNECTIONCHARACTERSET=AL32UTF8;
PERMSIZE=32768;TEMPSIZE=4096;TYPEMODE=0;
(Default setting AutoCommit=1)
Command>
```

Note: The example connects to the database with the `terry` user, which has at least `CREATE SESSION` privileges on the database. For more information on database users and how to create them, see "Creating or identifying a database user" in the *Oracle TimesTen In-Memory Database Security Guide*.

For more information on the `ttIsqlCS` utility and the TimesTen Client, see "Working with the TimesTen Client and Server" in the *Oracle TimesTen In-Memory Database Operations Guide*.

Adding a client DSN to a TimesTen Client on Windows You can add a client DSN to a TimesTen Client on Windows by using the `ttInstallDSN` utility included in the TimesTen Client Release 18.1. The `ttInstallDSN` utility creates a system DSN based on the contents of the output file of the `ttGridAdmin gridClientExport` command. You will need to make the file or its contents available to the Windows system where the TimesTen Client is installed.

```
C:\>ttInstallDSN -f C:\Users\terry\Downloads\sys.odbc.ini

Found the following DSNs in available 'C:\Users\terry\Downloads\sys.odbc.ini'.
0 : database1CS
[ Please select the DSN to be imported: ]
0
Adding DSN 'database1CS'.
```

Note: You must run the `ttInstallDSN` utility as administrator of Windows with the environment variables for the TimesTen Client set. See "Setting environment variables for TimesTen" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide* for further details.

From the TimesTen Client system, you can now connect to the `database1` database using the `database1CS` DSN with the `ttIsql` utility.

```
C:\>ttIsql -connStr "DSN=database1CS"

Copyright (c) 1996, 2016, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=database1CS;UID=terry;
Enter password for 'terry':
Connection successful: DSN=database1CS;TTC_SERVER=host3.example.com;
TTC_SERVER_DSN=DATABASE1;UID=terry;DATASTORE=/disk1/databases/database1;
DATABASECHARACTERSET=AL32UTF8;CONNECTIONCHARACTERSET=AL32UTF8;
PERMSIZE=256;TEMPSIZE=128;TYPEMODE=0;
(Default setting AutoCommit=1)
Command>
```

Note: The example connects to the database with the `terry` user, which has at least `CREATE SESSION` privileges on the database. For more information on database users and how to create them, see "Creating or identifying a database user" in the *Oracle TimesTen In-Memory Database Security Guide*.

For more information on the `ttInstallDSN` utility, see "ttInstallDSN" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttIsql` utility and the TimesTen Client, see "Working with the TimesTen Client and Server" in the *Oracle TimesTen In-Memory Database Operations Guide*.

Using a connection string to establish a client connection Alternatively, you can connect to a specific element by defining in the connection string the address of the host associated with that element, the database name, and a database user with at least `CREATE SESSION` privileges. (The client TCP/IP port is only necessary if the instance is not running with the default port.)

```
% ttIsqlCS -connStr "TTC_SERVER=host3.example.com;TTC_SERVER_DSN=database1;
 TCP_Port=6625;UID=terry"
```

Redirecting client connections

When an application connects to a client/server connectable a TCP/IP connection is established to one of the data instances in the grid. However, if the instance is busy then the instance can automatically redirect the client connection to another instance in the grid.

By default, a client connection can be automatically redirected to any available instance within the grid. However, you can limit or change this behavior with:

- The `TTC_Redirect` connection attribute, which defines how a client is redirected.
  - Automatic redirection: By default, this connection attribute is set to `1` so that a client connection is automatically redirected to any available instance within the grid if the current instance is busy or unavailable. The connection is redirected to the instance with the fewest number of client connections.
  - Elements within a single replica set: If you want the client to connect to instances with elements within a single replica set (because the data you are interested in is contained within this replica set), then set the `TTC_Redirect` attribute to `0`. Then, the client connects only to the instances with elements in the same replica set. If the connection is rejected, then a connection error is returned.
- The `TTC_Redirect_Limit` connection attribute, which limits how many times the client is redirected. The number of instances in your grid may be of a size that you want to limit the number of redirected client connection attempts for performance reasons. You can set the `TTC_Redirect_Limit` attribute to the number of connection redirection attempts. For example, setting `TTC_Redirect_Limit` limits the number of client connection redirection attempts to other instances to 10 attempts. If the client does not connect within this number of attempts, a connection error is returned.

If the client connection cannot be redirected to a suitable instance, then the client connection fails. See "Client connection failover" on page 11-30 for more information on the client failover process.

For more information on the `TTC_Redirect` or `TTC_Redirect_Limit` connection attributes, see "TTC_REDIRECT" or "TTC_Redirect_Limit", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

See "Modify the connection attributes in a connectable" on page 5-32 for information on how to modify TimesTen Client connection attributes.

Verify if your database is a distributed database

If you want to verify that the database you are connected to is indeed a distributed database (TimesTen Scaleout) and not a single-instance database (TimesTen Classic), call for the value of the `ttGridEnable` attribute with the `ttConfiguration` built-in procedure. The built-in procedure returns `ttGridEnable=1` for databases in a grid.

```
Command> CALL ttConfiguration('ttGridEnable');
< TTGridEnable, 1 >
1 row found.
Command>
```

For more information on the `ttConfiguration` built-in procedure, see "ttConfiguration" in the *Oracle TimesTen In-Memory Database Reference*.

Defining table distribution schemes

In TimesTen Scaleout, data is distributed across the elements of the grid. How the data is distributed is defined by the distribution scheme specified in the `DISTRIBUTE BY` clause of the `CREATE TABLE` statement. Regardless of how the data is distributed or on which element specific data is located, applications can access all the data in the database while connecting to a single element. However, there are some considerations you should take into account when defining the distribution scheme of a table.

Important: Before you start creating database objects, see "Authentication in TimesTen" in the *Oracle TimesTen In-Memory Database Security Guide*.

The available data distribution schemes for a table in TimesTen Scaleout are:

- Hash
- Reference
- Duplicate
- Materialized views as a secondary form of distribution

Hash

The hash distribution scheme distributes data based on the hash of the primary key or a set of user-specified columns. The hash key determines in which replica set a row should be stored. Any given row in the table is stored in only one replica set. If the table does not have a primary key or a user-specified distribution column, TimesTen Scaleout distributes the data based on the hash of a hidden column that TimesTen Scaleout adds for this purpose. This distribution scheme is adaptive to topology changes and uses consistent hashing. In other words, a row with an specific value in the hash key columns will always be allocated on the same replica set, provided that the topology does not change. If the topology changes, the location of the row may change when the data is re-distributed.

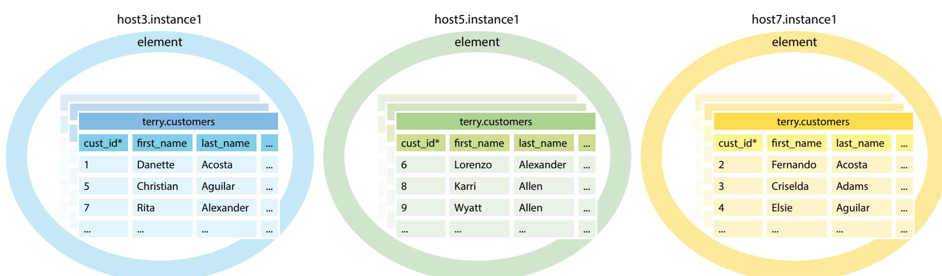
Note: If you create a table without specifying a `DISTRIBUTE BY` clause, TimesTen Scaleout defines a hash distribution scheme on the table. In addition, if a column is not specified in the `DISTRIBUTE BY HASH` clause, TimesTen Scaleout selects the primary key columns as the key columns of the distribution scheme. If a primary key is not defined, TimesTen Scaleout creates a hidden column as the hash key.

Create the `customers` table that uses a `DISTRIBUTE BY HASH` clause, which distributes data based on the hash of the `cust_id` primary key column.

```
CREATE TABLE customers (
    cust_id             NUMBER(10,0) NOT NULL PRIMARY KEY,
    first_name          VARCHAR2(30) NOT NULL,
    last_name           VARCHAR2(30) NOT NULL,
    addr1               VARCHAR2(64),
    addr2               VARCHAR2(64),
    zipcode             VARCHAR2(5),
    member_since        DATE NOT NULL
) DISTRIBUTE BY HASH;
```

Figure 5–1 shows the data distribution for the `customers` table in the `database1` database, as configured in "Creating a database" on page 5-1. TimesTen Scaleout distributes the data to each element based on the hash of the `cust_id` column.

Figure 5–1 Table distributed by hash



For more information on the hash distribution scheme, see "CREATE TABLE" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Reference

The reference distribution scheme distributes the data of a child table based on the location of the corresponding parent row of a foreign key constraint. This distribution scheme optimizes the performance of joins by distributing related data on a single element. When you join the parent and child tables, TimesTen Scaleout does not need to access different elements because all of the data is stored on the same element. The parent table can be distributed by hash or reference, which allows for a multi-tiered reference distribution.

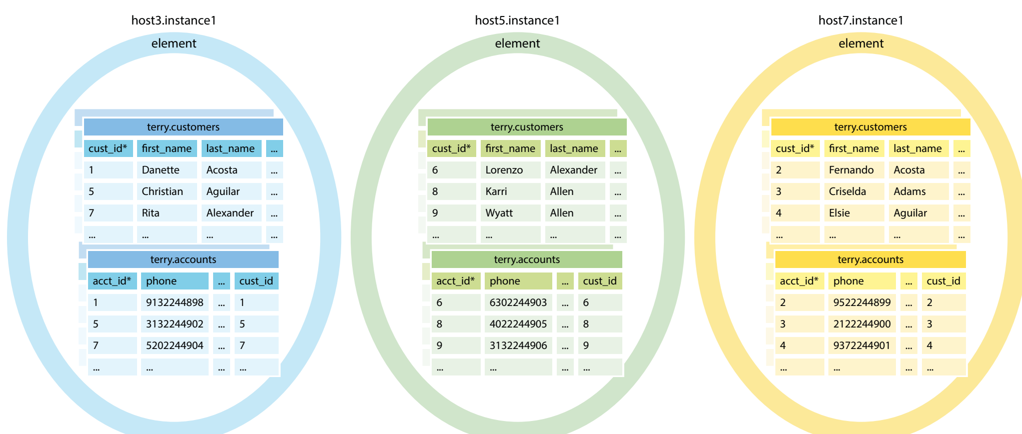
Note: Ensure you declare the child key columns of a foreign key constraint as `NOT NULL` when you use the `DISTRIBUTE BY REFERENCE` clause.

Create the `customers` parent table that uses a `DISTRIBUTE BY HASH` clause that distributes data based on the hash of the `cust_id` primary key column. Then, create the `accounts` child table that uses a `DISTRIBUTE BY REFERENCE` clause that distributes the data in the `accounts` table based on the location of the corresponding value of the referenced column, `customers(cust_id)`, in the `fk_customer` foreign key.

```
CREATE TABLE customers (
    cust_id             NUMBER(10,0) NOT NULL PRIMARY KEY,
    first_name          VARCHAR2(30) NOT NULL,
    last_name           VARCHAR2(30) NOT NULL,
    addr1               VARCHAR2(64),
    addr2               VARCHAR2(64),
    zipcode             VARCHAR2(5),
    member_since        DATE NOT NULL
) DISTRIBUTE BY HASH;

CREATE TABLE accounts (
    account_id          NUMBER(10,0) NOT NULL PRIMARY KEY,
    phone               VARCHAR2(15) NOT NULL,
    account_type        CHAR(1) NOT NULL,
    status              NUMBER(2) NOT NULL,
    current_balance     NUMBER(10,2) NOT NULL,
    prev_balance        NUMBER(10,2) NOT NULL,
    date_created        DATE NOT NULL,
    cust_id             NUMBER(10,0) NOT NULL,
    CONSTRAINT fk_customer
        FOREIGN KEY (cust_id)
            REFERENCES customers(cust_id)
) DISTRIBUTE BY REFERENCE (fk_customer);
```

Figure 5–2 shows the data distribution for the `customers` table in the `database1` database, as configured in "Creating a database" on page 5-1. TimesTen Scaleout distributes the data in the `customers` table to each replica set based on the hash of the `cust_id` primary key column. The figure also shows the data distribution for the `accounts` table, which is based on the location of the corresponding value of the referenced column, `cutomers(cust_id)`, in the `fk_customer` foreign key.

Figure 5–2 Table distributed by reference



For more information on the reference distribution scheme, see "CREATE TABLE" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Duplicate

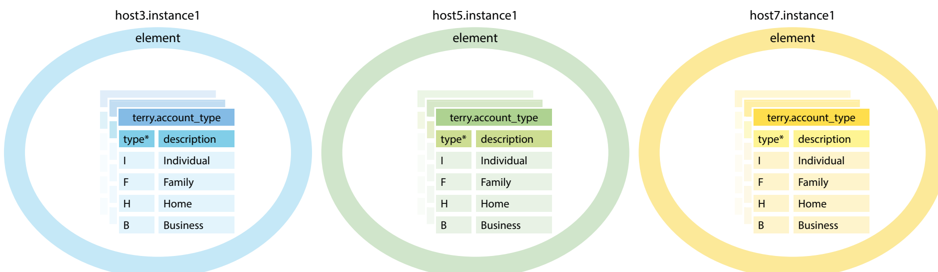
The duplicate distribution scheme distributes identical copies of the data of a table to all the elements of a database. This distribution scheme optimizes the performance of reads and joins against the table by ensuring that all data access is local. However, inserts and updates are more resource intensive than other distribution schemes.

Create the `account_type` table that uses a `DUPLICATE` clause that distributes the data to all the elements of a database.

```
CREATE TABLE account_type (
    type            CHAR(1) NOT NULL PRIMARY KEY,
    description     VARCHAR2(100) NOT NULL
) DUPLICATE;
```

Figure 5–3 shows the data distribution for the `account_type` table in the `database1` database, as configured in "Creating a database" on page 5-1. TimesTen Scaleout creates a copy of the data on all the elements of the database.

Figure 5–3 Table distributed by duplicate



For more information on the duplicate distribution scheme, see "CREATE TABLE" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Materialized views as a secondary form of distribution

Materialized views enable you to create a secondary form of distribution for a table and can be useful in scenarios such as:

- If you have a table with a primary key and a unique column and you distribute the table by hash based on the primary key column, TimesTen Scaleout would need to connect to every element of the database to verify the uniqueness of the values inserted or updated in the unique column. In this case, the hash distribution scheme cannot guarantee that duplicate values are located in the same element. If you additionally create a materialized view of the table that is distributed by hash based on the unique column, TimesTen Scaleout would be able to verify the uniqueness of the values in the unique column more efficiently since the location of the row in the materialized view would be determined by the value in the unique column. It is also recommended to create an index on the materialized view for the unique column to get optimum query performance. See Example 5–6 for an example of this scenario.
- If you have a table with two independent groups of columns that are commonly joined in queries, consider distributing the table by hash based on one of the groups of columns. This will optimize the queries against this group of columns. Then, to optimize queries against the second groups of columns, create a materialized view of the table that is distributed by hash based on the second group of columns. See Example 5–7 for an example of this scenario.

Consider the following when using materialized views in TimesTen Scaleout:

- Distribution is limited to the hash distribution scheme only.
- The hash key column must be explicitly specified, even if you intend to use the primary key as the hash key.
- The SQL optimizer may re-write a query against a base table to use an available materialized view instead if it detects that it may improve the execution time of the query.

Example 5–6 Materialized view with an unique column as hash key

This example creates the `customers` parent table that uses a `DISTRIBUTE BY HASH` clause, which distributes data based on the hash of the `cust_id` primary key column. The example also creates the `accounts` child table that uses a `DISTRIBUTE BY REFERENCE` clause. TimesTen Scaleout distributes the data in the `accounts` table based on the location to the corresponding value of the `cust_id` foreign key column. Finally, the example creates the `phone_mv` materialized view that uses a `DISTRIBUTE BY HASH` clause, which distributes data based on the hash of the `phone` unique column. This example enables you to validate the uniqueness of the values of the `phone` column in the `accounts` table while only reviewing the data contained in single data instance for a specific `phone` value.

```
CREATE TABLE customers (
    cust_id           NUMBER(10,0) NOT NULL PRIMARY KEY,
    first_name        VARCHAR2(30) NOT NULL,
    last_name         VARCHAR2(30) NOT NULL,
    addr1             VARCHAR2(64),
    addr2             VARCHAR2(64),
    zipcode           VARCHAR2(5),
    member_since      DATE NOT NULL
) DISTRIBUTE BY HASH;

CREATE TABLE accounts (
    account_id        NUMBER(10,0) NOT NULL PRIMARY KEY,
    phone             VARCHAR2(16) NOT NULL UNIQUE,
    account_type      CHAR(1) NOT NULL,
    status            NUMBER(2,0) NOT NULL,
    current_balance   NUMBER(10,2) NOT NULL,
    prev_balance      NUMBER(10,2) NOT NULL,
    date_created      DATE NOT NULL,
    cust_id           NUMBER(10,0) NOT NULL,
    CONSTRAINT fk_customer
        FOREIGN KEY (cust_id)
            REFERENCES customers(cust_id)
) DISTRIBUTE BY REFERENCE (fk_customer);

CREATE MATERIALIZED VIEW phone_mv
    DISTRIBUTE BY HASH (phone)
    AS SELECT phone FROM accounts;
```

Example 5–7 Materialized views for independent column groups

This example creates the `customers` table that uses a `DISTRIBUTE BY HASH` clause, which distributes data based on the hash of the `cust_id` primary key column. The example also creates the `accounts` table that uses a `DISTRIBUTE BY HASH` clause, which distributes data based on the hash of the `account_id` primary key column. Then, creates the `call_records` table that uses a `DISTRIBUTE BY HASH` clause, which distributes data based on the hash of the `call_id` primary key column.

```
CREATE TABLE customers (
    cust_id           NUMBER(10,0) NOT NULL PRIMARY KEY,
    first_name        VARCHAR2(30) NOT NULL,
    last_name         VARCHAR2(30) NOT NULL,
    addr1             VARCHAR2(64),
    addr2             VARCHAR2(64),
    zipcode           VARCHAR2(5),
    account_id        NUMBER(10,0),
    member_since      DATE NOT NULL
) DISTRIBUTE BY HASH;

CREATE TABLE accounts (
    account_id        NUMBER(10,0) NOT NULL PRIMARY KEY,
    phone             VARCHAR2(15) NOT NULL,
    account_type      CHAR(1) NOT NULL,
    status            NUMBER(2,0) NOT NULL,
    current_balance   NUMBER(10,2) NOT NULL,
    prev_balance      NUMBER(10,2) NOT NULL,
    date_created      DATE NOT NULL,
    cust_id           NUMBER(10,0) NOT NULL UNIQUE
) DISTRIBUTE BY HASH;

CREATE TABLE call_records (
    call_id           NUMBER(10,0) NOT NULL PRIMARY KEY,
    caller            NUMBER(10,0) NOT NULL,
    receiver          NUMBER(10,0) NOT NULL,
    call_time         TIMESTAMP NOT NULL,
    code              INT NOT NULL
) DISTRIBUTE BY HASH;
```

Consider that you need a report on the accounts and customers that made a call with an specific code, like shown in the following query.

```
SELECT accounts.account_id, customers.cust_id, call_records.code
    FROM accounts, customers, call_records
    WHERE customers.cust_id = call_records.caller
        AND call_records.code = ?
        AND customers.account_id = accounts.account_id;
```

To optimize the join between the `customers` and `call_records` tables, the example creates the `customers_calls_mv` materialized view on the `call_records` table based on the `caller` column.

```
CREATE MATERIALIZED VIEW customers_calls_mv
    DISTRIBUTE BY HASH (caller)
    AS SELECT caller, code FROM call_records;
```

Also, to optimize the join between the `customers` and `accounts` tables, the example creates the `customes_account_mv` materialized view on the customers table based on the `account_id` column.

```
CREATE MATERIALIZED VIEW customers_account_mv
    DISTRIBUTE BY HASH (account_id)
    AS SELECT account_id FROM customers;
```

For more information on materialized views, see "CREATE MATERIALIZED VIEW" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Determining the value of the PermSize attribute

You must have enough memory available in both the permanent and temporary memory regions of every element for the database to operate successfully. You can monitor the amount of memory allocated, in-use, and in-use high-water for this two regions for the local element or all elements of the database by querying the SYS.V$MONITOR and SYS.GV$MONITOR system views, respectively, as shown in Example 5–8.

Example 5–8 Monitoring the memory regions of an element

```
Command> SELECT elementid, perm_allocated_size, perm_in_use_size,
 perm_in_use_high_water, temp_allocated_size, temp_in_use_size,
 temp_in_use_high_water FROM sys.v$monitor;

  ELEMENTID:                1
  PERM_ALLOCATED_SIZE:      262144
  PERM_IN_USE_SIZE:         30338
  PERM_IN_USE_HIGH_WATER:   30338
  TEMP_ALLOCATED_SIZE:      131072
  TEMP_IN_USE_SIZE:         21073
  TEMP_IN_USE_HIGH_WATER:   24600

1 row found.
Command> SELECT elementid, perm_allocated_size, perm_in_use_size,
 perm_in_use_high_water, temp_allocated_size, temp_in_use_size,
 temp_in_use_high_water FROM sys.gv$monitor;

  ELEMENTID:                1
  PERM_ALLOCATED_SIZE:      262144
  PERM_IN_USE_SIZE:         30338
  PERM_IN_USE_HIGH_WATER:   30338
  TEMP_ALLOCATED_SIZE:      131072
  TEMP_IN_USE_SIZE:         21073
  TEMP_IN_USE_HIGH_WATER:   24600


  ELEMENTID:                3
  PERM_ALLOCATED_SIZE:      262144
  PERM_IN_USE_SIZE:         30289
  PERM_IN_USE_HIGH_WATER:   30322
  TEMP_ALLOCATED_SIZE:      131072
  TEMP_IN_USE_SIZE:         21070
  TEMP_IN_USE_HIGH_WATER:   24470


  ELEMENTID:                5
  PERM_ALLOCATED_SIZE:      262144
  PERM_IN_USE_SIZE:         30289
  PERM_IN_USE_HIGH_WATER:   30322
  TEMP_ALLOCATED_SIZE:      131072
  TEMP_IN_USE_SIZE:         20943
  TEMP_IN_USE_HIGH_WATER:   24407


  ELEMENTID:                2
  PERM_ALLOCATED_SIZE:      262144
  PERM_IN_USE_SIZE:         30338
  PERM_IN_USE_HIGH_WATER:   30338
  TEMP_ALLOCATED_SIZE:      131072
```

```
  TEMP_IN_USE_SIZE:         20943
  TEMP_IN_USE_HIGH_WATER:   24470


  ELEMENTID:                4
  PERM_ALLOCATED_SIZE:      262144
  PERM_IN_USE_SIZE:         30289
  PERM_IN_USE_HIGH_WATER:   30322
  TEMP_ALLOCATED_SIZE:      131072
  TEMP_IN_USE_SIZE:         21006
  TEMP_IN_USE_HIGH_WATER:   24407


  ELEMENTID:                6
  PERM_ALLOCATED_SIZE:      262144
  PERM_IN_USE_SIZE:         30289
  PERM_IN_USE_HIGH_WATER:   30322
  TEMP_ALLOCATED_SIZE:      131072
  TEMP_IN_USE_SIZE:         21006
  TEMP_IN_USE_HIGH_WATER:   24470
1 row found.
```

If necessary, increase the amount of memory allocated for either region by increasing the value of the `PermSize` or `TempSize` attribute. See "Modify the connection attributes in a database definition" on page 5-30 for information on how to modify the `PermSize` or `TempSize` attribute of a database.

You can estimate the value of the `PermSize` attribute based on the SQL schema and the expected number of rows for each table of the database with the `ttSize` utility. For example, if you eventually expect to insert 1,000,000 rows into the `customers` table, the table will need about 287 MB (300,448,527 bytes = 286.53 MB) available, as shown in Example 5–9.

Example 5–9 Estimating the size of a table

```
% ttSize -tbl terry.customers -rows 1000000 database1

 Rows = 1000000

Total in-line row bytes = 300442597

Indexes:
  Range index TERRY.CUSTOMERS adds 5930 bytes
  Total index bytes = 5930

Total = 300448527
```

However, the `ttSize` utility is optimized for databases in TimesTen Classic. A database in TimesTen Scaleout uses 8 to 16 bytes more per row than a similar database in TimesTen Classic. Consider adding to the value calculated by the `ttSize` utility from 8 to 16 bytes per row for a more accurate estimate. In the case of the `customers` table, if you add 16 bytes per row to the value calculated by the `ttSize` utility, you will need about 302 MB (316,448,527 bytes = 301.79 MB) available.

If you repeat this estimate for every table of the database, you can get a rough idea of the size of the permanent memory region a database requires across all hosts by adding the estimated size of every table. However, the `PermSize` attribute defines the amount of memory allocated for an element, not the whole database. To determine how much of the size estimated for each table you must assign to each element you have to take into consideration the distribution scheme of the table:

- For a table using a hash or reference distribution scheme, divide the number of rows by the number of replica sets before doing the estimation with the `ttSize` utility.

  Note: Consider that tables with a reference distribution scheme may reference key values unevenly. If your data uses one or more key values as reference more often then any other key value available, it is possible that dividing the number of rows by the number of replica sets would be an inaccurate calculation. You should take special considerations based on the composition of your data.

- For a table using a duplicate distribution scheme, use the total number of rows for the estimation. After all, you find every row of a table using a duplicate distribution on every element of the database.

Considering that the `customers` table uses a hash distribution scheme and that the `database1` database consists of three replica sets, each element should be able to store 333,334 rows, which represents 101 MB (100,209,711 + 16 * 333,334 bytes = 100.65 MB) in the permanent memory region (defined by the `PermSize` attribute) for just the `customers` table, as shown in Example 5–10.

Example 5–10 Estimating the size of a table in a single element

```
% ttSize -tbl terry.customers -rows 333334 database1

Rows = 333334

Total in-line row bytes = 100203781

Indexes:
  Range index TERRY.CUSTOMERS adds 5930 bytes
  Total index bytes = 5930

Total = 100209711
```

For more information on the `ttSize` utility, see "ttSize" in the *Oracle TimesTen In-Memory Database Reference*.

Bulk loading data into a database

TimesTen Scaleout enables you to load data into a database from various sources. You can load data into a specific table either from a file by using the `ttBulkCp` utility or an Oracle database table by using the `ttLoadFromOracle` built-in procedure.

Both the `ttBulkCp` utility and `ttLoadFromOracle` built-in procedure support in TimesTen Scaleout a `localOnly` filter option that enables you to load only the rows that are hashed to the local element and its replicas. If you use the `localOnly` filter option, the `ttBulkCp` utility and `ttLoadFromOracle` built-in procedure ignore rows that are hashed to remote elements that are not a replica of the local element. Regardless of the options you specify, the `ttBulkCp` utility and `ttLoadFromOracle` built-in procedure do not copy duplicate rows into a table.

With the `localOnly` filter option enabled and depending of the distribution scheme of the table, the `ttBulkCp` utility and `ttLoadFromOracle` built-in procedure behave as follows:

- **Hash:** Retain and insert rows that have hash key values that are hashed to the elements of the local data instance and its replicas. They ignore rows that are hashed to the remaining elements.
- **Reference:** Retain and insert rows whose reference key value references to a hash or reference key value that is hashed to the local element and its replicas. They ignore rows that are hashed to the remaining elements.
- **Duplicate:** Ignore the `localOnly` option. They insert rows into the elements of all data instances.

The advantages of using the `localOnly` filter option are:

- It requires less network bandwidth to distribute the data during the bulk loading operation.
- It allows a failed bulk loading operation to be retried independent of other elements.

The disadvantages of using the `localOnly` filter option are:

- The source file must be available to all hosts, or at least to one host for each replica set of the grid. This only applies for bulk loading operations with the `ttBulkCp` utility.
- You must run a bulk loading operation on an element of every replica set.
- Every bulk loading operation must process the entire data set, even though it ignores any rows hashed to a different replica set.

These sections describe how to load data into a table in TimesTen Scaleout.

- Populating a table with the ttBulkCp utility
- Populating a table with the ttLoadFromOracle built-in procedure

---

Note: The following examples consider the same grid scenario as the one described in "Define the distribution map of the database" on page 5-5.

Populating a table with the ttBulkCp utility

The `ttBulkCp` utility with the `-i` option enables you to load data from a file. This option uses standard `INSERT` SQL statements to load data into a specific table of a database. The `ttBulkCp` utility inserts each row into its corresponding element based on the distribution scheme of the table.

Note:

- Unlike operations performed with the `ttGridAdmin` utility, the `ttBulkCp` utility (and the `ttBulkCpCS` utility) must be run on a data instance instead of the active management instance, and its use is not limited to the instance administrator.
- Ensure that the user running the command or the one specified in the connection string has `INSERT` privileges on the specified table.

These sections describe the options for loading data into a database while using the `ttBulkCp` utility.

- Populate a table from a single location
- Populate a table from several locations

Populate a table from a single location

If the source file is only available to a single data instance, run the `ttBulkCp` utility with the `-i` option to insert the rows of the specified database into their corresponding element based on the distribution scheme of the specified database.

From the data instance with access to the source file, insert all rows in the file into the `customers` table of the `database1` database.

```
% ttBulkCp -i -connStr "DSN=database1;UID=terry" customers
 /mydir/customers_data.dmp
Enter password for 'terry':

/mydir/customers_data.dmp:
    1000 rows inserted
    1000 rows total
```

For more information on using the `ttBulkCp` utility, see "Bulk copy data using the ttBulkCp utility" in the *Oracle TimesTen In-Memory Database Operations Guide* and "ttBulkCp" in the *Oracle TimesTen In-Memory Database Reference*.

Populate a table from several locations

If the source file is available to any given host in the grid, run the `ttBulkCp` utility with the `-i` and `-localOnly` options on one data instance of each replica set of the database to insert the rows hashed to the local element and its replicas from a file into a table.

Use the `ttGridAdmin dbStatus -replicaset` command from the active management instance (in this example the active management instance is `host1.instance1`) to help you determine the data instances associated with each replica set.

```
% ttGridAdmin dbStatus database1 -replicaset
Database database1 Replica Set status as of Thu Jan 11 13:17:29 PST 2018

RS DS Elem Host  Instance  Status Date/Time of Event  Message
-- -- ---- ----- --------- ------ ------------------- -------
 1  1    1 host3 instance1 opened 2018-01-10 14:34:43
    2    2 host4 instance1 opened 2018-01-10 14:34:43
 2  1    3 host5 instance1 opened 2018-01-10 14:34:42
    2    4 host6 instance1 opened 2018-01-10 14:34:42
 3  1    5 host7 instance1 opened 2018-01-10 14:34:42
    2    6 host8 instance1 opened 2018-01-10 14:34:42
```

Insert the rows hashed to the local element and its replica from the source file into the `customers` table of the `database1` database. Ensure you run the `ttBulkCp` utility on one data instance of each replica set available, the `host3.instance1`, `host5.instance1`, and `host7.instance1` data instances for example.

On the `host3.instance1` data instance:

```
% ttBulkCp -i -localOnly -connStr "DSN=database1;UID=terry" customers
 /mydir/customers_data.dmp
Enter password for 'terry':

/mydir/customers_data.dmp:
```

```
    339 rows inserted
    661 rows not inserted (ignored)
    1000 rows total
```

Note: For this example where the element of the `host4.instance1` data instance is defined as the replica of the element of the `host3.instance1` data instance, the same rows inserted into the `customers` table in the element of the `host3.instance1` data instance are inserted into the `customers` table in the element of the `host4.instance1` data instance.

On the `host5.instance1` data instance:

```
% ttBulkCp -i -localOnly -connStr "DSN=database1;UID=terry" customers
 /mydir/customers_data.dmp
Enter password for 'terry':

/mydir/customers_data.dmp:
    327 rows inserted
    673 rows not inserted (ignored)
    1000 rows total
```

Note: For this example where the element of the `host6.instance1` data instance is defined as the replica of the element of the `host5.instance1` data instance, the same rows inserted into the `customers` table in the element of the `host5.instance1` data instance are inserted into the `customers` table in the element of the `host6.instance1` data instance.

On the `host7.instance1` data instance:

```
% ttBulkCp -i -localOnly -connStr "DSN=database1;UID=terry" customers
 /mydir/customers_data.dmp
Enter password for 'terry':

/mydir/customers_data.dmp:
    334 rows inserted
    666 rows not inserted (ignored)
    1000 rows total
```

Note: For this example where the element of the `host8.instance1` data instance is defined as the replica of the element of the `host7.instance1` data instance, the same rows inserted into the `customers` table in the element of the `host7.instance1` data instance are inserted into the `customers` table in the element of the `host8.instance1` data instance.

For more information on the `ttGridAdmin dbStatus` command, see "Monitor the status of a database (dbStatus)" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on using the `ttBulkCp` utility, see "Bulk copy data using the ttBulkCp utility" in the *Oracle TimesTen In-Memory Database Operations Guide* and "ttBulkCp" in the *Oracle TimesTen In-Memory Database Reference*.

Populating a table with the ttLoadFromOracle built-in procedure

The `ttLoadFromOracle` built-in procedure enables you to load data from an Oracle database.

These sections describe how to load data from a Oracle database into a database while using the `ttLoadFromOracle` built-in procedure.

- Enable communication to an Oracle database
- Populate a table from several locations

Enable communication to an Oracle database

For the `ttLoadFromOracle` built-in procedure to be able to import data from an Oracle database table into a database table, TimesTen Scaleout must be able to recognize and communicate with the Oracle database. For this to happen, you need to:

- Import the contents of the sqlnet.ora file
- Import the contents of the tnsnames.ora file
- Apply the changes made to the latest version of the model

> **Note:** Importing the contents of both the `sqlnet.ora` and `tnsnames.ora` files is also relevant for applications that use OCI, Pro*C/C++, or ODP.NET to communicate with an Oracle Database. See "Oracle Database operations" in the *Oracle TimesTen In-Memory Database Reference* for further details.

Import the contents of the sqlnet.ora file The `ttGridAdmin SQLNetImport` command imports the contents of a `sqlnet.ora` file into the latest version of the model.

Import the contents of the `sqlnet.ora` file.

```
% ttGridAdmin SQLNetImport /mydir/sqlnet.ora
SQLNet configuration file /mydir/sqlnet.ora imported
```

Import the contents of the tnsnames.ora file The `ttGridAdmin TNSNamesImport` command imports the contents of a `tnsnames.ora` file into the latest version of the model.

Import the contents of the `tnsnames.ora` file.

```
% ttGridAdmin TNSNamesImport /mydir/tnsnames.ora
TNSNames configuration file /mydir/tnsnames.ora imported
```

Apply the changes made to the latest version of the model The `ttGridAdmin modelApply` command applies the changes made to the latest version of the model into the operational grid.

```
% ttGridAdmin modelApply
...
Updating grid state.................................................OK
Pushing new configuration files to each Instance.....................OK
...
ttGridAdmin modelApply complete
```

See "Applying the changes made to the model" on page 4-18 for more information on the `ttGridAdmin modelApply` command.

Populate a table from a single location

The following example connects with the `ttIsql` utility to the `database1` database to copy the rows from the `terry.customers` table of an Oracle database into the `terry.customers` table of the `database1` database.

Note: Ensure that the database user has the `INSERT` privilege on the table the built-in procedure copies data into.

From a connection to the element of any data instance:

```
Command> call ttLoadFromOracle('terry', 'customers', 'SELECT * FROM
 terry.customers');
< 1000 >
1 row found.
```

For more information on the `ttLoadFromOracle` built-in procedure, see "ttLoadFromOracle" in the *Oracle TimesTen In-Memory Database Reference*.

Populate a table from several locations

Call the `ttLoadFromOracle` built-in procedure with the `localOnly=Y` parameter to copy the rows hashed to a local element and its replicas from an Oracle database table into a TimesTen Scaleout database table. If you use the `localOnly=Y` parameter, the `ttLoadFromOracle` built-in procedure ignores rows that are hashed to remote elements that are not a replicas of the local element.

The following example connects with the `ttIsql` utility to the `database1` database to copy the rows hashed to the local element and its replicas from the `terry.customers` table of an Oracle database into the `terry.customers` table of the `database1` database. If necessary, use the `ttGridAdmin dbStatus -replicaset` command from the active management instance (in this example the active management instance is `host1.instance1`) to help you determine the data instances associated with each replica set.

```
% ttGridAdmin dbStatus database1 -replicaset
Database database1 Replica Set status as of Thu Jan 11 13:17:29 PST 2018

RS DS Elem Host  Instance  Status Date/Time of Event  Message
-- -- ---- ----- --------- ------ ------------------- -------
 1  1    1 host3 instance1 opened 2018-01-10 14:34:43
    2    2 host4 instance1 opened 2018-01-10 14:34:43
 2  1    3 host5 instance1 opened 2018-01-10 14:34:42
    2    4 host6 instance1 opened 2018-01-10 14:34:42
 3  1    5 host7 instance1 opened 2018-01-10 14:34:42
    2    6 host8 instance1 opened 2018-01-10 14:34:42
```

Ensure you call the `ttLoadFromOracle` built-in procedure on one replica of each replica set available, the `host3.instance1`, `host5.instance1`, and `host7.instance1` data instances for example.

Note: Ensure that the database user has the `INSERT` privilege on the table the built-in procedure copies data into.

From a connection to the element of the `host3.instance1` data instance:

```
Command> call ttLoadFromOracle('terry', 'customers', 'SELECT * FROM
```

```
 terry.customers', 4, 'localOnly=Y');
< 339 >
1 row found.
```

Note: For this example where the element of the `host4.instance1` data instance is defined as the replica of the element of the `host3.instance1` data instance, the same rows inserted into the `customers` table in the element of the `host3.instance1` data instance are inserted into the `customers` table in the element of the `host4.instance1` data instance.

From a connection to the element of the `host5.instance1` data instance:

```
Command> call ttLoadFromOracle('terry', 'customers', 'SELECT * FROM
 terry.customers', 4, 'localOnly=Y');
< 327 >
1 row found.
```

Note: For this example where the element of the `host6.instance1` data instance is defined as the replica of the element of the `host5.instance1` data instance, the same rows inserted into the `customers` table in the element of the `host5.instance1` data instance are inserted into the `customers` table in the element of the `host6.instance1` data instance.

From a connection to the element of the `host7.instance1` data instance:

```
Command> call ttLoadFromOracle('terry', 'customers', 'SELECT * FROM
 terry.customers', 4, 'localOnly=Y');
< 334 >
1 row found.
```

Note: For this example where the element of the `host8.instance1` data instance is defined as the replica of the element of the `host7.instance1` data instance, the same rows inserted into the `customers` table in the element of the `host7.instance1` data instance are inserted into the `customers` table in the element of the `host8.instance1` data instance.

For more information on the `ttGridAdmin dbStatus` command or `ttLoadFromOracle` built-in procedure, see "Monitor the status of a database (dbStatus)" or "ttLoadFromOracle", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Unloading a database from memory

In TimesTen Scaleout, a database is automatically loaded into memory upon creation. Once loaded into memory, a database remains in memory until the database is explicitly unloaded. Closing all connections to the database will not automatically unload the database from memory.

One of the reasons you may need to unload a database is to modify the value of a first connection attribute, like increasing the value of the `PermSize` attribute.

To unload a database from memory, perform these tasks:

- *Close the database to user connections*. The `ttGridAdmin dbClose` command disables new user connections to a database.
- *Disconnect all applications from the database.* The `ttGridAdmin dbDisconnect` command terminates all user connections to a database.
- *Unload the database from memory*. The `ttGridAdmin dbUnload` command unloads every element of the database from the memory of their respective hosts.

Close the `database1` database from user connections.

```
% ttGridAdmin dbClose database1
Database database1 close started
```

> **Note:** The `ttGridAdmin dbClose` command does not close existing connections to the database, but instead disallows the creation of new user connections. You must terminate all open connections independently. Closing a database is an asynchronous operation that is performed independently to each element by its data instance.
>
> Also, the instance administrator can always create new connections to a database regardless of the database being closed or not.

Verify that all the elements of the `database1` database are closed to user connections.

```
% ttGridAdmin dbStatus database1 -elements
Database database1 element level status as of Tue Nov 27 13:35:45 PST 2018

Host  Instance  Elem Status Date/Time of Event  Message
----- --------- ---- ------ ------------------- -------
host3 instance1    1 loaded 2018-11-27 13:35:43
host4 instance1    2 loaded 2018-11-27 13:35:43
host5 instance1    3 loaded 2018-11-27 13:35:43
host6 instance1    4 loaded 2018-11-27 13:35:43
host7 instance1    5 loaded 2018-11-27 13:35:43
host8 instance1    6 loaded 2018-11-27 13:35:43
```

> **Note:** The `ttGridAdmin dbStatus` utility displays the status of an element as `loaded` instead of `opened` when the element is closed to user connections.

Disconnect all applications from the `database1` database. You must stop the workload and gracefully disconnect every application from the database. If you are unable to individually disconnect every application from the database, use the `ttGridAdmin dbDisconnect` command to disconnect all user connections from the database as shown in Example 5–11.

Example 5–11 Disconnecting applications from a database

The example disconnects all user connections from the `database1` database once all open transactions commit or roll back.

```
% ttGridAdmin dbDisconnect database1 -transactional
Database database1 dbDisconnect started
```

Use the `ttGridAdmin dbDisconnectStatus` command to check the status of the disconnection process.

```
% ttGridAdmin dbDisconnectStatus database1
```

```
Database  Host  Instance  Elem State        Started
--------- ----- --------- ---- ------------ ------------------------
database1                      Complete     2018-11-27T13:38:43.000Z
          host3 instance1    1 Disconnected
          host4 instance1    2 Disconnected
          host5 instance1    3 Disconnected
          host6 instance1    4 Disconnected
          host7 instance1    5 Disconnected
          host8 instance1    6 Disconnected
```

Then, verify that there are no connections to the database with the `-connections` option of `ttGridAdmin dbStatus` command.

```
% ttGridAdmin dbStatus database1 -connections
Host Instance ConnId Name Pid Type CHost CAddr CPid
---- -------- ------ ---- --- ---- ----- ----- ----
```

> **Note:** If the `-transactional` option fails or takes too long, use the `-immediate` option of the `ttGridAdmin dbDisconnect` command to force a rollback on all open transactions and disconnect the applications.
>
> Furthermore, if the `-immediate` option fails to close all connections, you can use the `-abort` option. This option ungracefully disconnects all applications and may cause loss of data.

Unload the `database1` database.

```
% ttGridAdmin dbUnload database1
Database database1 unload started
```

The unloading of the database is an asynchronous operation that is performed independently by each data instance. This operation returns an error if there is an open user connection to the database.

> **Note:** If you used the `ttGridAdmin dbDisconnect -abort` command, some elements may be invalidated and the `ttGridAdmin dbUnload` command may fail. Use the `-force` option of the `ttGridAdmin dbUnload` command to allow the unload to proceed anyway. This option may cause loss of data.

You can verify the status of the database unloading process with the `ttGridAdmin dbStatus` command as shown in Example 5–12.

Example 5–12 Verifying the status of the unloading process of a database

The example shows a status summary for the `database1` database. Notice that the report shows all elements of the database as closed and unloaded.

```
% ttGridAdmin dbStatus database1
Database database1 summary status as of Tue Nov 27 13:41:18 PST 2018

created,unloaded,closed
Completely created elements: 6 (of 6)
Completely loaded elements: 0 (of 6)
Completely created replica sets: 3 (of 3)
Completely loaded replica sets: 0 (of 3)
```

```
Open elements: 0 (of 6)
```

For more information on the `ttGridAdmin dbClose`, `ttGridAdmin dbDisconnect`, `ttGridAdmin dbDisconnectStatus`, `ttGridAdmin dbUnload`, or `ttGridAdmin dbStatus` command, see "Close a database (dbClose)", "Force all connections to disconnect (dbDisconnect)", "Check status of forced disconnection (dbDisconnectStatus)", "Unload a database (dbUnload)", or "Monitor the status of a database (dbStatus)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Reloading a database into memory

To reload a database into memory, perform these the tasks:

- *Load the database into memory*. The `ttGridAdmin dbLoad` command loads every element of the database into memory of their respective hosts.
- *Open the database for user connections*. The `ttGridAdmin dbOpen` command enables the database for user connections.

Load all the elements of the `database1` database into memory.

```
% ttGridAdmin dbLoad database1
Database database1 load started
```

Open the `database1` database for user connections.

```
% ttGridAdmin dbOpen database1
Database database1 open started
```

For more information on the `ttGridAdmin dbLoad` or `ttGridAdmin dbOpen` command, see "Load a database into memory (dbLoad)" or "Open a database (dbOpen)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Modifying the connection attributes of a database

There are three types of connection attributes based on their persistence:

- Attributes that are set on database creation and that cannot be modified. You store the value assigned for these attributes in the database definition.
- Attributes that are set when the database is loaded into memory and that can only be modified upon unloading and reloading the database into memory. You store the value assigned for these attributes in the database definition.
- Attributes that are set by each connection to the database and persist for the duration of that connection. You store the value assigned for these attributes in a connectable.

These sections describe how to modify the connection attributes of a database depending on where they are stored:

- Modify the connection attributes in a database definition
- Modify the connection attributes in a connectable

Modify the connection attributes in a database definition

To modify a database definition is to modify the assigned value of the connection attributes that a database definition supports. The types of connection attributes that a database definition supports and that can be modified after database creation are:

- First connection attributes
- PL/SQL first connection attributes
- TimesTen Server connection attributes

> **Note:** You cannot modify data store attributes after database creation. To use a different value for a data store attribute, you need to destroy and re-create the database. See "Destroying a database" on page 5-33 and "Creating a database" on page 5-1 for details on how to destroy and re-create a database.

TimesTen Scaleout assigns the default value to any supported attribute not explicitly specified in the database definition. Attributes with the default value assigned can be modified by including the attribute in the database definition. Once you add or modify the attributes defined in a database definition and apply the changes to current version of the model, TimesTen Scaleout overwrites the configuration files of every data instance with the new attributes in the DSN associated with the database definition.

To modify the values assigned to the attributes supported by a database definition, perform these tasks:

1. If you don't have access to the file that you used to create (or modify) the database definition, export the contents of the `database1` database definition to a file.

   ```
   % ttGridAdmin dbdefExport database1 /mydir/database1.dbdef
   ```

   Example 5–13 shows the contents of the exported file.

Example 5–13 Exported database definition file

```
# DbDef GUID ED157D81-D915-490B-AC80-353234E8516E Exported 2016-06-14 14:52:32
[database1]
Connections=2048
DatabaseCharacterSet=AL32UTF8
DataStore=/disk1/databases/database1
Durability=0
LogBufMB=1024
LogDir=/disk2/logs
PermSize=32768
TempSize=4096
```

2. Modify the value of the `PermSize` attribute from `32768` and `49152` in the exported database definition file, as shown in Example 5–14.

Example 5–14 Modified database definition file

```
# DbDef GUID ED157D81-D915-490B-AC80-353234E8516E Exported 2016-06-14 14:52:32
[database1]
Connections=2048
DatabaseCharacterSet=AL32UTF8
DataStore=/disk1/databases/database1
Durability=0
LogBufMB=1024
LogDir=/disk2/logs
PermSize=49152
TempSize=4096
```

3. Import the contents of the modified database definition file into the `database1` database definition.

```
% ttGridAdmin dbdefModify /mydir/database1.dbdef
Database Definition DATABASE1 modified.
```

4. Apply the changes to the `database1` database definition to the current version of the model.

```
% ttGridAdmin modelApply
...
Updating grid state.................................................OK
Pushing new configuration files to each Instance.....................OK
...
ttGridAdmin modelApply complete
```

5. Unload the `database1` database as shown in "Unloading a database from memory" on page 5-27.

6. Restart the `database1` database as shown in "Reloading a database into memory" on page 5-30 to bring the changes you made to the `database1` database definition into effect.

For a complete description of all the connection attributes, see "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin dbdefExport`, `ttGridAdmin dbdefModify`, or `ttGridAdmin modelApply` command, see "Export a database definition (dbdefExport)", "Modify a database definition (dbdefModify)", or "Apply the latest version of the model (modelApply)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Modify the connection attributes in a connectable

To modify a connectable is to modify the assigned value of the connection attributes that a connectable supports. The types of connection attributes that a connectable supports are:

- General connection attributes
- NLS general connection attributes
- PL/SQL connection attributes
- TimesTen Client connection attributes

TimesTen Scaleout assigns the default value to any supported attribute not explicitly specified in the connectable. Attributes with the default value assigned can be modified by including the attribute in the connectable. Once you add or modify the attributes defined in a connectable and apply the changes to current version of the model, TimesTen Scaleout overwrites the configuration files of every data instance with the new attributes in the DSN associated with the connectable.

To modify the values assigned to the attributes supported by a connectable, perform these tasks:

1. If you don't have access to the file that you used to create (or modify) the connectable, export the contents of the `database1CS` connectable to a file.

```
% ttGridAdmin connectableExport database1CS /mydir/database1CS.connect
```

Example 5–15 shows the contents of the exported file.

Example 5–15 Exported connectable file

```
# Connectable GUID E3175374-EC83-4826-A78C-8E3D21A0EFF6 Exported 2016-06-07
 12:40:47
[database1CS]
ConnectionCharacterSet=AL32UTF8
UID=terry
```

2. Modify the value of the `SQLQueryTimeout` connection attribute to `300` in the exported connectable file, as shown in Example 5–16.

Example 5–16 Modified connectable file

```
# Connectable GUID E3175374-EC83-4826-A78C-8E3D21A0EFF6 Exported 2016-06-07
 12:40:47
[database1CS]
ConnectionCharacterSet=AL32UTF8
UID=terry
SQLQueryTimeout=300
```

3. Import the contents of the modified connectable file into the `database1CS` connectable.

   ```
   % ttGridAdmin connectableModify /mydir/database1CS.connect
   Connectable DATABASE1CS modified.
   ```

4. Apply the changes to the `database1CS` connectable to the current version of the model.

   ```
   % ttGridAdmin modelApply
   ...
   Updating grid state.................................................OK
   Pushing new configuration files to each Instance.....................OK
   ...
   ttGridAdmin modelApply complete
   ```

For a complete description of all the connection attributes, see "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin connectableExport`, `ttGridAdmin connectableModify`, or `ttGridAdmin modelApply` command, see "Export a connectable (connectableExport)", "Modify a connectable (connectableModify)", or "Apply the latest version of the model (modelApply)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Destroying a database

Before you attempt to destroy a database, ensure you backup all your data, since it will be discarded in the destruction process. See "Backing up and restoring a database" on page 10-7 for more information on how to backup your data in TimesTen Scaleout.

The `ttGridAdmin dbDestroy` command performs these operations in order to destroy a database:

- Delete the checkpoint and log files of the database stored on every data instance.
- Delete the entries in the management instance that keep track of the status of the database, including the entry that recorded the creation of the database.

However, before you can destroy a database, you must unload the database. See "Unloading a database from memory" on page 5-27 for details.

Destroy the `database1` database.

```
% ttGridAdmin dbDestroy database1
Database DATABASE1 destroy started
```

You may also want to delete the database definition associated with the database. The `ttGridAdmin dbdefDelete` command deletes a database definition in the latest version of the model. This command also deletes any connectable associated with the database definition.

Delete the `database1` database definition and its associated connectables from the latest version of the model.

```
% ttGridAdmin dbdefDelete database1
Database Definition database1 deleted
```

Apply the deletion of the `database1` database definition to the current version of the model.

```
% ttGridAdmin modelApply
...
Pushing new configuration files to each Instance......................OK
...
ttGridAdmin modelApply complete
```

TimesTen Scaleout removes the database definition and its connectables from the grid.

For more information on the `ttGridAdmin dbDestroy`, `ttGridAdmin dbdefDelete`, or `ttGridAdmin modelApply` command, see "Destroy a database (dbDestroy)", "Delete a database definition (dbdefDelete)", or "Apply the latest version of the model (modelApply)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

6 Understanding Distributed Transactions in TimesTen Scaleout

In TimesTen Scaleout, distributed transactions are processed by a two-phase commit protocol. This chapter discusses how TimesTen Scaleout maintains ACID-compliant databases through distributed transactions.

The following terminology is related to understanding the distributed transaction processing algorithms that TimesTen Scaleout employs:

- **Participant:** An element that executes one or more SQL statements from a distributed transaction. Not all elements in a database participate in every transaction. An element only becomes a participant of a transaction if one or more operations of that transaction requires access to the data stored in the element.

  > **Note:** Each element maintains its own independent set of checkpoint and transaction log files. They behave in the same manner as the checkpoint and transaction log files of a database in TimesTen Classic. See "Checkpoint operations" and "Transaction logging" in the *Oracle TimesTen In-Memory Database Operations Guide* for more information.

- **Transaction manager:** The thread of the application (or of the TimesTen Server, for a client/server application) that is connected to the database and initiates the transaction. The transaction manager coordinates the transaction operations with all participants.
- **Prepare-to-commit log record**: A type of log record written to the transaction log of the database during the prepare phase of the two-phase commit protocol. It contains the commit decision for the transaction.
- **Durable log record:** Participants write a prepare-to-commit or commit log record synchronously to the transaction log. Nondurable log records are asynchronously written by the participants.
- **Distributed transaction:** A transaction with two or more participants.
- **Single-element transaction:** A transaction with only one participant. Single-element transactions do not use the two-phase commit protocol. Single-element transactions are only possible in a grid with K-safety set to `1`.
- **In-doubt transaction:** A transaction where a participant wrote a prepare-to-commit log record, but the commit log record is not present in the transaction log. If the transaction manager wrote the prepare-to-commit log record to the transaction log, which means there is a known commit decision, then the transaction is not in-doubt.

- **Remote connection:** A connection from the transaction manager to a participant of the transaction.

This chapter includes the following topics:

- Transaction manager
- Durability settings
- Two-phase commit protocol
- Troubleshooting distributed transactions

Transaction manager

Applications connect to a database in TimesTen Scaleout by connecting to one element of the database. Each transaction executed by a connection requires a transaction manager. For client/server applications the transaction manager is the thread in the TimesTen Scaleout server that is acting as a proxy for the application. For direct mode applications the transaction manager is the thread in the application that connects to TimesTen Scaleout. The transaction manager coordinates the execution of statements on elements (participants), or more specifically:

- If the application issues a commit or rollback, the transaction manager ensures that all participants have consistent data based on the commit or rollback decision from the two-phase commit protocol.
- If a participant returns an error, such as a constraint violation, the transaction manager coordinates the response. The transaction manager ensures that TimesTen Scaleout returns the appropriate error message to the user and that all participants release the allotted resources.
- If a participant fails, the transaction manager creates a state that the failed participant uses during its recovery to restore to a consistent state.
- If the participant where the transaction manager resides fails, participants classify the transaction as in-doubt if they completed the prepare phase but did not receive the commit decision and are no longer able to reach the transaction manager.

Status of the participants

When a participant completes the execution of a statement, it sends a message to the transaction manager. The message includes information about the number of rows affected. If the message specifies that:

- The participant modified the affected rows, such as with a `INSERT`, `UPDATE`, or `DELETE` operation, the transaction manager flags the participant as a write participant.
- The participant did not modify any rows, then the transaction manager flags the participant as a read participant.

The read or write status of a participant affects the way the transaction manager processes a commit operation:

- If all participants are read participants, then the transaction manager handles the commit without going through the prepare phase. In other words, read participants perform the commit operation without needing a consensus from the other participants.
- If there are one or more write participants, then the transaction manager handles the commit as a two-phase operation.

Durability settings

You control how durable your transactions are with the `Durability` attribute. This attribute defines if transactions create durable prepare-to-commit log records. Regardless of the setting of this attribute, transactions that include DDL statements create durable prepare-to-commit and commit log records. The `Durability` attribute supports two different values:

- Durability set to 1
- Durability set to 0

Durability set to 1

If you set the `Durability` attribute to `1`, participants write durable prepare-to-commit log records and nondurable commit log records for distributed transactions. Having the `Durability` attribute set `1` ensures that committed transactions are recoverable in the case of a failure. This is the default setting of the `Durability` attribute when K-safety is set to `1`.

For more information on the `Durability` attribute, see "Durability" in the *Oracle TimesTen In-Memory Database Reference*.

Durability set to 0

If you set the `Durability` attribute to `0`, participants write nondurable prepare-to-commit and commit log records for distributed transactions. To ensure a measure of durability, TimesTen Scaleout provides the following new features that are generally exclusive to databases with the `Durability` attribute set to `0`:

- Epoch transactions
- EpochInterval attribute
- CreateEpochAtCommit attribute

Epoch transactions

An epoch transaction is a distributed transaction that creates a durable commit log record that marks a globally consistent point in time across all elements of a database. Epoch transactions are durably committed on every element of the database. An epoch transaction ensures that the database is consistent up to the timestamp of the epoch transaction. In other words, an epoch transaction ensures that any transaction already in the commit phase is recoverable.

> **Note:** TimesTen Scaleout uses Lamport timestamps to provide partial ordering for transactions that commit on different elements of a database. Each element has a Lamport timestamp that is updated by, among others, prepare and commit operations. The transaction manager logs the Lamport timestamp of every committed transaction.

Transactions in a grid with K-safety set to `2` and a database with the `Durability` attribute set to `0` are durable under normal conditions, since TimesTen Scaleout writes durable prepare-to-commit log records of transactions that involve a replica set with a failed element until the failed element recovers. Only if both elements of the replica set fail simultaneously, a transaction may become nondurable. However, TimesTen Scaleout enables you to promote transactions to epoch transactions. An epoch transactions and all transactions committed before it are more resilient to catastrophic failures, since you can recover a database to the consistent point marked by the epoch commit log record of the epoch transaction.

Note:

- See "Recovering a replica set after an element goes down" on page 11-14 for more information on how to recover failed element in a replica set.
- See "Recovering from a down replica set" on page 11-16 for more information on how to recover a failed replica set.

Before promoting a transaction, consider that a commit for an epoch transaction is more expensive than a commit for a regular transaction, because it creates durable log records for both the prepare-to-commit and commit phase and involves every element of the database, including those that were not participants before the promotion of the transaction to an epoch transaction.

Use these built-in procedures and system view to promote and manage epoch transactions:

- The `ttEpochCreate` built-in procedure promotes a transaction to an epoch transaction, including read-only transactions.
- The `ttDurableCommit` built-in procedure promotes a write transaction to an epoch transaction.
- The `SYS.V$EPOCH_SESSION` system view stores the Lamport timestamp of the latest epoch transaction that the connection created since the second-to-last checkpoint operation.

Example 6–1 Promoting a transaction to an epoch transaction

The example shows and verifies the promotion of a write transaction to an epoch transaction.

```
Command> autocommit OFF;
Command> INSERT INTO transactions VALUES (txn_seq.NEXTVAL, 189, SYSDATE, NULL,
 'A', 5.49);
Command> SELECT epoch FROM sys.v$epoch_session;
< 1023.1 >
1 row found.
Command> CALL ttEpochCreate();
Command> COMMIT;
Command> SELECT epoch FROM sys.v$epoch_session;
< 1024.1 >
1 row found.
```

For more information on the `ttEpochCreate` or `ttDurableCommit` built-in procedure, see "ttEpochCreate" or "ttDurableCommit", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `SYS.V$EPOCH_SESSION` system view, see "SYS.V$EPOCH_SESSION" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

EpochInterval attribute

Each epoch commit log record is associated to a specific checkpoint file on every element. In the case of an unexpected failure of an element, the recovery process must use the checkpoint file on each element that is associated with the latest epoch commit log record, which is not necessarily the latest checkpoint available on the element.

You can configure a database to generate periodic epoch transactions at an specified interval with the `EpochInterval` first connection attribute. The value set for the `EpochInterval` attribute must be less than one half of the value set for the `CkptFrequency` first connection attribute, so that there is at least one epoch transaction for every checkpoint operation. If you set the `CkptFrequency` attribute to a value greater than zero and the `EpochInterval` attribute to a value greater than one half of the value set for the `CkptFrequency` attribute, TimesTen Scaleout will re-adjust the `EpochInterval` attribute to one half of value set for the `CkptFrequency` attribute.

For more information on the `EpochInterval` or `CkptFrequency` attribute, see "EpochInterval" or "CkptFrequency", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

CreateEpochAtCommit attribute

You can configure a connection to promote every write transaction committed by that connection to an epoch transaction with the `CreateEpochAtCommit` general connection attribute. If you set the `CreateEpochAtCommit` attribute to `1`, you ensure that every transaction you commit during the connection is recoverable in the case of failure. However, as with any epoch transaction, commits operations are more expensive than with regular transactions, so it is recommended that you limit `CreateEpochAtCommit=1` for critical operations only.

> **Note:** Even though the `DurableCommits` attribute is intended for databases in TimesTen Classic, the attribute emulates the behavior of the `CreateEpochAtCommit` attribute when set to `1` for a database in TimesTen Scaleout. See "DurableCommits" in the *Oracle TimesTen In-Memory Database Reference*.

When the `Durability` attribute is set to `0`, the transaction manager and the participants behave differently depending of the settings of the `CreateEpochAtCommit` attribute, as shown on Table 6–1.

Table 6–1 Participants behavior on commit based on CreateEpochAtCommit setting

CreateEpochAtCommit	Commit behavior
`0`	Participants write nondurable prepare-to-commit and commit log records for every distributed transaction to commit.
`1`	Promotes every transaction to an epoch transaction.

Setting both the `Durability` and `CreateEpochAtCommit` attributes to `0` provides the best performance. In this case, call the `ttEpochCreate` or `ttDurableCommit` built-in procedures to ensure that you have durable records of important transactions.

For more information on the `Durability` or `CreateEpochAtCommit` attribute, see "Durability" or "CreateEpochAtCommit", respectively, in the *Oracle TimesTen In-Memory Database Reference*. For more information on the `ttEpochCreate` or `ttDurableCommit` built-in procedure, see "ttEpochCreate" or "ttDurableCommit", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Two-phase commit protocol

Ensure that you understand the concepts covered in "Transaction manager" on page 6-2 and "Durability settings" on page 6-3 before reading this section.

As previously mentioned, distributed transactions follow a two-phase commit protocol. TimesTen Scaleout implements the two-phase commit protocol as follows:

Phase 0: Transaction

1. An application establishes a connection to a database. Every connection is associated with a specific element of the database, which becomes the transaction manager for all distributed transactions initiated from that connection.
2. The application executes one or more SQL statements. The transaction manager sends the statements to all the participants for execution. Based on the returned results of the execution of the SQL statement, the transaction manager identifies and updates the status of the participants.
3. The application issues a commit.

Phase 1: Prepare phase

1. The transaction manager sends a prepare message to all participants. The message includes the identity of the transaction manager and all the participants.
2. Each participant, once it receives the prepare message, performs either of these actions:
   - If the participant is a write participant, it writes a prepare-to-commit log record that stores information to subsequently either commit or rollback the transaction. The participant also locks the modified rows to prevent read operations.
   - If the participant is a read participant, it identifies the transaction as read-only.
3. The participant sends a prepare response to the transaction manager with its vote for the commit decision:
   - A write participant only votes 'Yes' if it was able to write the prepare-to-commit log record.

     Note: If `Durability` is set to `1`, the participant writes a durable prepare-to-commit log record.

   - A read participant always votes 'Yes' and commits the transaction without waiting for the commit decision. In this case, the commit operation consists on releasing all locks and temporary resources related to the transaction.

Phase 2: Commit phase

1. Once the transaction manager receives the prepare response from at least one element in every replica set participating in the transaction, it writes a prepare-to-commit log record that includes the commit decision. The transaction manager bases the commit decision on the scenarios described in Table 6–2.

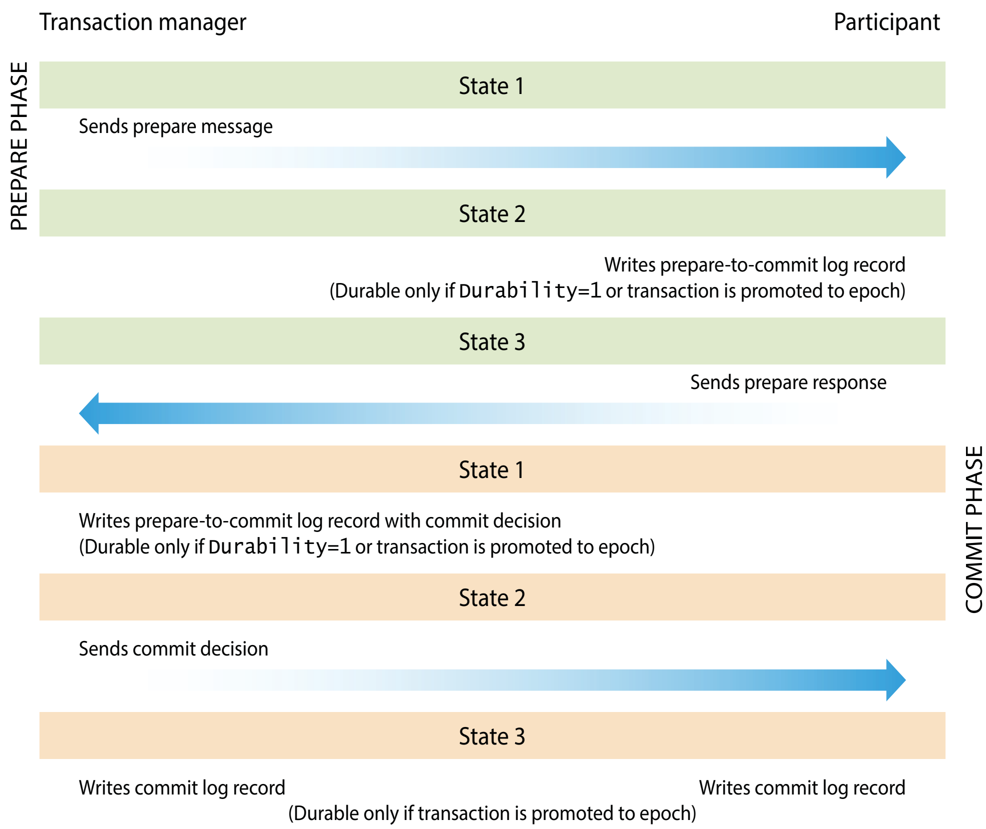
Table 6–2 Scenarios for commit decision

Scenarios	Decision
All write participants send a 'Yes' vote in their prepare response and within them there is at least one element for each participating replica set. (Failed participants do not affect the commit decision once they are identified as failed as long as its replica sends a response.)	Commit
Any write participant sends a 'No' vote in their prepare response.	Roll back

2. The transaction manager sends a message to all write participants with the commit decision.
3. All write participants, including the transaction manager, commit or rollback the transaction based on the commit decision.

Figure 6–1 shows the two-phase commit protocol as implemented for distributed transactions in TimesTen Scaleout.

Figure 6–1 Two-phase commit protocol



Two-phase commit failure analysis

There are several types of potential failures that may affect the operation of a database for outstanding distributed transactions. Table 6–3 summarizes these failure types and describes how TimesTen Scaleout responds to them.

Table 6–3 Failure types in a distributed transaction

Failure	Action
Transaction manager fails.	If the transaction manager fails (for example, the application terminates), the main daemon for that instance catches the failure and informs the subdaemon. The subdaemon sends a commit or rollback message to all participants depending on the state of the transaction.
The host of the transaction manager fails.	If the host of the transaction manager fails, the daemon and all subdaemons fail. Each participant will recognize this failure when their TCP connection to the transaction manager closes or times out. Once a participant recognizes the failure, the participant rolls back any transaction that has not reached the prepare phase. If the participant already sent its prepare response, it will ask other participants for the commit decision and perform one of the following actions: ■ If at least one of the other participants received the commit decision, then the asking participant will fullfil the commit decision. ■ If none of the other participants received the commit decision, then the asking participant waits for the transaction manager to recover.
All elements from a participating replica set fail before writing a prepare-to-commit log record.	The transaction manager decides to rollback the transaction.
Participant fails after writing a prepare-to-commit log record.	The participant, once it recovers, requests the commit decision from one of the other participants.
Participant is busy.	The transaction manager waits until it receives a prepare response from the participant.

Troubleshooting distributed transactions

In TimesTen Classic, a transaction may need to wait for a resource held by another transaction. If that resource is protected by a lock, the transaction waits for the lock to be released. It is possible that the other transaction is waiting on an external event that is not represented as database lock, so the deadlock detector does not resolve the problem. The following are possible resources that can cause a transaction to wait:

- a semaphore wait
- a latch wait
- an I/O event
- an unattended open transaction
- a long running operation

In TimesTen Scaleout, these cases still apply, and there is an additional possible case. When an element fails, all the transactions initiated from that element have lost their transaction manager. If the remote participants did not receive the commit decision for a transaction after sending their prepare response, then the participants must wait to commit or rollback the now in-doubt transaction. Also, if a participant fails after sending its prepare response but before receiving the commit decision, the transaction becomes an in-doubt transaction for the failed participant.

Global transaction id

The global transaction id uniquely identifies a transaction across all the elements of a database. The global transaction id is composed of these parameters:

- The element id of the transaction manager
- The connection id of the transaction manager or local transaction id
- A counter for the transactions issued from the connection

See Example 6–2 for details on how to retrieve the global transaction id of an outstanding transaction in TimesTen Scaleout.

Example 6–2 Retrieving the global transaction id

This example shows how to retrieve the global transaction id from within the connection issuing the transaction. The `SYS.V$XACT_ID` system view stores all the parameters necessary to construct the global transaction id of a transaction.

```
Command> autocommit 0;
Command> INSERT INTO transactions VALUES (txn_seq.NEXTVAL, 342, SYSDATE, NULL,
 'A', 8.33);
1 row inserted.
Command> SELECT elementId, xactId, counter FROM sys.v$xact_id;
< 3, 1, 148 >
1 row found.
```

For more information on the `SYS.V$XACT_ID` system view, see "SYS.V$XACT_ID" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

Managing in-doubt transactions

TimesTen Scaleout resolves in-doubt transactions automatically during element recovery. The prepare-to-commit log record of the transaction contains the information about other participants. To resolve the in-doubt transaction, the recovering element requests the commit decision from one of the participants listed in the prepare-to-commit log record.

In the case of a transaction manager failure, TimesTen Scaleout should be able to resolve an in-doubt transaction as long as one participant from each write replica set is available. However, if none of the participants have the commit decision and not all write replica sets are available, TimesTen Scaleout cannot resolve the in-doubt transaction. If TimesTen Scaleout failed to resolve an in-doubt transaction, use the `ttXactAdmin` utility to force the commit or rollback of the transaction.

> **Important:** For most cases, you should always roll back an unresolved in-doubt transaction. However, if you decide to externally commit the transaction, you first will need to evict any unreachable participating replica set to ensure a consistent database. Evicting a replica set implies losing all the data stored in that replica set.
>
> See "Unavailability of data when a full replica set is down or fails" on page 11-15 for more information on evicting replica sets.

You may use the `ttXactAdmin` utility to verify the `state` of every outstanding transaction, as shown in Example 6–3. If the transaction `state` is `in-doubt`, you can externally commit or rollback the transaction with the same utility, as shown in Example 6–4 or Example 6–5, respectively.

Example 6–3 Verifying the state of every outstanding transaction

This example shows how to retrieve the status of every outstanding transaction that the element of the data instance running the command is a participant. The `ttXactAdmin` utility only retrieves information related to the element of the data instance executing the command.

```
% ttXactAdmin -connStr "DSN=database1"
2016-12-14 11:00:36.995
/disk1/databases/database1
TimesTen Release 18.1.4.1.0
ElementID 3

Program File Name: _ttIsql
XactID          PID     Context           State      Loghold       Last ID
3.1.148         26247   0x13b3ff0         Active     -1.-1         [-1:2]

  Resource  ResourceID          Mode  SqlCmdID            Name
  Database  0x01312d0001312d00  IX    0
  HashedKey ffffffffe5a341d5    SF    284478280           PAT.ACCOUNTS
  Table     2367304             IRC   284478280           PAT.ACCOUNTS
  EndScan   AAAVVUAAAA9AAAAGjO  En    284478280           PAT.TRANSACTIONS
  Table     2367320             IRC   284478280           PAT.TRANSACTIONS

  Begin Time: 10:59:21.695
```

Example 6–4 Committing an in-doubt transaction

The example uses the `ttXactAdmin` utility to commit transaction 3.1.148. This command can only be successfully run if the transaction manager is down and its replica set is evicted from the database. See "Recovering from a down replica set" on page 11-16 for more information on when and how to evict a failed replica set.

```
% ttXactAdmin -connStr "DSN=database1" -xactIdCommit 3.1.148
```

Example 6–5 Rolling back an in-doubt transaction

The example uses the `ttXactAdmin` utility to roll back transaction 3.1.148.

```
% ttXactAdmin -connStr "DSN=database1" -xactIdRollback 3.1.148
```

For more information on the `ttXactAdmin` utility, see "ttXactAdmin" in the *Oracle TimesTen In-Memory Database Reference*.

7

Using SQL in TimesTen Scaleout

This chapter describes how to use SQL to work with databases in TimesTen Scaleout. Topics include:

Overview of SQL

A database consists of elements. Each element stores a portion of your data. You manipulate and query the data in the database through SQL operations from any element. For example, you can use the `CREATE USER` statement to create a user in your database from any element. After TimesTen Scaleout creates the user, this user is available in all elements of the database. You can issue DDL and DML statements from any element which TimesTen Scaleout then applies to all elements in your database. You can issue a `SELECT` statement to execute a query that is prepared from one element and executed on other elements in the query with the result returned to the originating element.

Notes:

- The syntax and semantics for SQL statements, functions, and the like are detailed in the *Oracle TimesTen In-Memory Database SQL Reference*.
- See "Summary of SQL statements supported in TimesTen" in the *Oracle TimesTen In-Memory Database SQL Reference* for information on the SQL statements supported in TimesTen Scaleout.

Overview of PL/SQL

Applications can use PL/SQL to access and manipulate data. Anonymous blocks are fully supported. PL/SQL is executed on the element to which the application is connected. SQL statements that are invoked from PL/SQL are executed across the grid as with any other SQL.

See the *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide* for detailed information on PL/SQL and the "Examples Using TimesTen SQL in PL/SQL" chapter in the *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide* for examples. For unsupported PL/SQL features, see Table 1–9, " TimesTen Classic features that are unsupported in TimesTen Scaleout" for information.

Working with tables

Tables are the objects used to define how to distribute data in your database. Each user-defined table has a defined distribution scheme. TimesTen Scaleout manages the distribution of data according to this defined distribution scheme. The distribution scheme defines how the rows of data in the table are distributed across the grid. The `CREATE TABLE` statement allows you to specify a distribution clause to define the distribution scheme for the table. When you create the table, it exists on every element of the database. Rows of data in the table exist on different elements of the database.

For detailed information on the syntax and semantics for creating, altering, and dropping tables, see "CREATE TABLE" in the *Oracle TimesTen In-Memory Database SQL Reference*. See "Data distribution" on page 1-13 or "Defining table distribution schemes" on page 5-13 for more information on defining distribution schemes.

Altering tables

You can alter tables in TimesTen Scaleout to change defaults or add and drop columns and constraints. However, you cannot change the distribution scheme unless the table is empty. In addition, you cannot drop a constraint that is named in the `DISTRIBUTE BY REFERENCE` clause. See "ALTER TABLE" in the *Oracle TimesTen In-Memory Database SQL Reference* for more information.

Table 7–1, " ALTER TABLE rules for distribution schemes" shows the rules associated with altering tables. Supporting examples follow.

Table 7–1 ALTER TABLE rules for distribution schemes

ALTER statement	Comment
`CREATE TABLE t1 (c1 NUMBER, c2 VARCHAR2 (10));` `ALTER TABLE t1 DISTRIBUTE BY HASH (c1);`	The operation succeeds if the table is empty. If the table is not empty, the operation fails because the distribution key cannot be changed on tables that are not empty.
`CREATE TABLE t1...CONSTRAINT fk1... DISTRIBUTE BY REFERENCE(fk1);` `ALTER TABLE t1 DROP CONSTRAINT(fk1);`	The operation fails. The foreign key is used to distribute the table.

Examples include:

- Example 7–1, "Use ALTER TABLE to add a primary key constraint"
- Example 7–2, "Use ALTER TABLE to change the distribution key"

Example 7–1 Use ALTER TABLE to add a primary key constraint

This example creates the `mytable` table without a primary key or distribution clause. The table is distributed by hash on a hidden column. Then the `ALTER TABLE` statement is used to add a primary key constraint. The operation succeeds but the distribution key is not changed.

```
Command> CREATE TABLE mytable (col1 NUMBER NOT NULL, col2 VARCHAR2 (32));
Command> describe mytable;

Table SAMPLEUSER.MYTABLE:
  Columns:
    COL1                            NUMBER NOT NULL
    COL2                            VARCHAR2 (32) INLINE
  DISTRIBUTE BY HASH

1 table found.
(primary key columns are indicated with *)
```

Now alter the table to add the primary key. The operation succeeds. The distribution scheme and distribution key do not change.

```
Command> ALTER TABLE mytable ADD CONSTRAINT c1 PRIMARY KEY (col1);
Command> describe mytable;

Table SAMPLEUSER.MYTABLE:
  Columns:
   *COL1                            NUMBER NOT NULL
    COL2                            VARCHAR2 (32) INLINE
  DISTRIBUTE BY HASH

1 table found.
(primary key columns are indicated with *)
```

Example 7–2 Use ALTER TABLE to change the distribution key

This example shows that you can use the `ALTER TABLE` statement to change the distribution key, but only if the table is empty.

```
Command> CREATE TABLE mytable2 (col1 NUMBER NOT NULL, col2 VARCHAR2 (32))
DISTRIBUTE BY HASH (col1,col2);
```

```
Command> describe mytable2;

Table SAMPLEUSER.MYTABLE2:
  Columns:
    COL1                            NUMBER NOT NULL
    COL2                            VARCHAR2 (32) INLINE
  DISTRIBUTE BY HASH (COL1, COL2)

1 table found.
(primary key columns are indicated with *)
```

Use the `ALTER TABLE` statement to change the distribution key to `col1`. The operation succeeds because the table is empty.

```
Command> ALTER TABLE mytable2 DISTRIBUTE BY HASH (col1);
Command> describe mytable2;

Table SAMPLEUSER.MYTABLE2:
  Columns:
    COL1                            NUMBER NOT NULL
    COL2                            VARCHAR2 (32) INLINE
  DISTRIBUTE BY HASH (COL1)

1 table found.
(primary key columns are indicated with *)
```

Understanding materialized views

Materialized views provide a second mechanism for distributing rows of data. (The first mechanism is tables.) See "Materialized views as a secondary form of distribution" on page 5-16 for more information.

Materialized views are useful as global indexes to reduce or eliminate multi-element access (broadcasts). See "Understanding indexes" on page 7-5 for more information.

Additional considerations:

- Specify the `DISTRIBUTE BY HASH` distribution scheme. `DISTRIBUTE BY REFERENCE` and `DUPLICATE` clauses are not supported.
- You must specify the `DISTRIBUTE BY HASH` clause and specify a set of columns in the `DISTRIBUTE BY` clause. Even if you distribute the materialized view by primary key, you must specify the primary key in the distribution clause.
- Create a unique index for a materialized view if there is a unique column on the detail table, but only create it if the unique column is used as the distribution key for the materialized view. This will increase performance if the columns are frequently used in DML operations and queries.
- The unique index columns and primary key columns must be a super set of the distribution key columns.
- Consider creating a materialized view on a table along with any indexes after the table is populated to decrease the time it takes to populate the table.

See "CREATE MATERIALIZED VIEW" in the *Oracle TimesTen In-Memory Database SQL Reference* for more information on materialized views.

Understanding indexes

When you create an index, TimesTen Scaleout creates the index on all elements that are in the distribution map. TimesTen Scaleout populates each element's index with the rows that are stored on that element. When you drop an index, TimesTen Scaleout drops the index on all elements.

Indexed access that does not include all columns of the distribution key requires multi-element access (a broadcast across all elements). To avoid broadcasts, thereby optimizing index access, consider creating a materialized view and an index on the materialized view. Example 7–3, "Use materialized view as a global index" illustrates how to create a materialized view to optimize index access.

> **Note:** There are storage and overhead costs incurred for DML operations against the columns that are defined in the materialized view. This is a key tuning trade-off in TimesTen Scaleout.

Example 7–3 Use materialized view as a global index

The `accounts` table is distributed by reference based on `cust_id`. To increase the performance of an `UPDATE` statement in which the distribution key is not used, consider creating a materialized view and distribute the materialized view on the column(s) not included in the distribution key (`account_id`, in this example). Then, create an index on the `account_id` column. Use the `ttIsql set timing` command to illustrate the difference in execution times.

Use the `ttIsql describe` command to describe the `accounts` table.

```
Command> SELECT status FROM accounts WHERE account_id = 500;
< 10 >
1 row found.

Command> autocommit off;
Command> set timing on;

Command> UPDATE accounts SET status=20 WHERE account_id=500;
1 row updated.
Execution time (SQLExecute) = 1.018369 seconds.

Command> set timing off;
Command> SELECT status FROM accounts WHERE account_id = 500;
< 20 >
1 row found.

Command> rollback;
Command> SELECT status FROM accounts WHERE account_id = 500;
< 10 >
1 row found.

Command> CREATE MATERIALIZED VIEW account_id_mv
         DISTRIBUTE BY HASH (account_id) AS SELECT * FROM accounts;
1010000 rows materialized.
Command> CREATE UNIQUE HASH INDEX account_index_mv ON account_id_mv (account_id);

Command> autocommit off;
Command> set timing on;
Command> UPDATE accounts SET status=20 WHERE account_id=500;
1 row updated.
Execution time (SQLExecute) = 0.002601 seconds.
```

```
Command> set timing off;
Command> rollback;
Command> SELECT status FROM accounts WHERE account_id = 500;
< 10 >
1 row found.
```

Using sequences

The `CREATE SEQUENCE` statement creates a new sequence number generator that can subsequently be used by multiple users to generate unique `BIGINT` data types. As with materialized views and tables, once you create the sequence object, sequence values can be retrieved from any element of the database.

The values are retrieved from the sequence in blocks and cached in order to reduce the overhead of performing a globally coordinated update on the sequence object every time a value is retrieved. While the values returned from a sequence in TimesTen Scaleout are guaranteed to be unique, they are not guaranteed to be sequential.

The `BATCH` clause is specific to TimesTen Scaleout. The batch value configures the range of unique sequence values stored in the element. Each element has its own batch. An element will get a new batch when its local batch is consumed. There is one element that owns the sequence and is responsible for allocating batch sequence blocks to other elements.

Sequence values are unique, but across elements the values might not be returned in monotonic order. Within a single element, sequence values are in monotonic order. But over time, across elements, sequence values are not returned monotonically. However, the monotonic property is guaranteed within an element.

If your application records events and tags each event with a sequence value, the application cannot assume that event 100, for example, happened after event 80. If your application needs to make this assumption, then set `BATCH` to 1. However, there is substantial communication overhead if you set `BATCH` to 1.

In summary, unless the `BATCH` value is set to 1, the order of sequence values is not guaranteed to be maintained across all elements. However, no matter what the batch value is, the uniqueness of the sequence value is guaranteed to be maintained across all elements. In addition, the order of sequence values is guaranteed to be maintained within an element.

You can change the default batch value of an existing sequence by issuing the `ALTER SEQUENCE` statement. The batch value is the only alterable clause. See "CREATE SEQUENCE" and "ALTER SEQUENCE" in the *Oracle TimesTen In-Memory Database SQL Reference* for more information. Use the `DROP SEQUENCE` statement to drop a sequence. See "DROP SEQUENCE" in the *Oracle TimesTen In-Memory Database SQL Reference* for information on dropping a sequence.

Understanding batch allocation

Deciding what to set for the batch value depends on these considerations:

- If you set the value to 1, sequence values are issued in monotonic order, no matter how many elements exist. However, there is substantial communication overhead with a value of 1, which results in a detrimental impact on performance. Unless absolutely necessary, do not set the value to 1 as it will directly impact the performance of your system.

- If you set the value greater than 1, unique sequence values are not issued in strict order across all elements. If your connection retrieves multiple values from a sequence, there is no guarantee that the values will be consecutive or contiguous. If multiple connections retrieve values from a sequence, there may be gaps in the range of values retrieved.
- You should consider setting batch to a high value to avoid excessive communication among elements (unless it is necessary to set the batch value to 1 for the proper functioning of your application).
- The unique sequence value within the batch boundary cannot be greater than `MAXVALUE`. For example, if a sequence increments by 1, has a batch value of 3, and a maximum value of 5, the first batch includes 1, 2, and 3. The second batch includes 4 and 5 only.
- The batch value must be greater or equal to the cache value.
- If you do not specify a batch value, the default is 10 million. Each element starts with its own set of 10 million values. If the 10 million values are used up, the element gets 10 million more. The minimum and maximum values and the number of unique values are determined by the `MINVALUE`, `MAXVALUE`, and `INCREMENT BY` values.
- Each element in a replica set has different batches.

Examples of batch assignment:

- Example 7–4, "Illustrate batch assignment for three elements"
- Example 7–5, "Illustrate a second batch assignment for three elements"

Example 7–4 Illustrate batch assignment for three elements

This example creates the `myseq` sequence with a batch value of 100. Then, from the connection that is connected to element 1, the example issues a `SELECT...NEXTVAL` query. The example then issues a second and third `SELECT...NEXTVAL` query from the connection that is connected to element 2 and the connection that is connected to element 3 respectively. The example illustrates the allocation of batch assignment for each element. In this example:

- Element 1 receives a batch of 1-100.
- Element 2 receives a batch of 101-200.
- Element 3 receives a batch of 201-300.

From the connection that is connected to element 1 (demonstrated by `SELECT elementId# FROM dual`), create the `myseq` sequence specifying a batch value of 100. Then, issue a `SELECT...NEXTVAL` query. Observe the value `1` is returned.

```
Command> SELECT elementId# FROM dual;
< 1 >
1 row found.
Command> CREATE SEQUENCE myseq BATCH 100;
Command> SELECT myseq.NEXTVAL FROM dual;
< 1 >
1 row found.
```

From the connection that is connected to element 2, first verify the connection to element 2, then issue a `SELECT...NEXTVAL` query. Observe the value `101` is returned.

```
Command> SELECT elementId# FROM dual;
< 2 >
1 row found.
```

```
Command> SELECT myseq.NEXTVAL FROM dual;
< 101 >
1 row found.
```

From the connection that is connected to element 3, first verify the connection to element 3, then issue a `SELECT...NEXTVAL` query. Observe the value `201` is returned.

```
Command> SELECT elementId# FROM dual;
< 3 >
1 row found.
Command> SELECT myseq.NEXTVAL FROM dual;
< 201 >
1 row found.
```

Example 7–5 Illustrate a second batch assignment for three elements

This example creates the `myseq2` sequence with a batch value of 100. Then, from the connection that is connected to element 1, the example issues a `SELECT...NEXTVAL` query. The example then issues a second and third `SELECT...NEXTVAL` query from the connection that is connected to element 3 and the connection that is connected to element 2 respectively. The example illustrates the allocation of batch assignment for each element. In this example:

- Element 1 receives a batch of 1-100.
- Element 3 receives a batch of 101-200.
- Element 2 receives a batch of 201-300.

From the connection that is connected to element 1 (demonstrated by `SELECT elementId# FROM dual`), create the `myseq2` sequence specifying a batch value of 100. Then, issue a `SELECT...NEXTVAL` query. Observe the value `1` is returned.

```
Command> SELECT elementId# FROM dual;
< 1 >
1 row found.
Command> CREATE SEQUENCE myseq2 BATCH 100;
Command> SELECT myseq2.NEXTVAL FROM dual;
< 1 >
1 row found.
```

From the connection that is connected to element 3, first verify the connection to element 3, then issue a `SELECT...NEXTVAL` query. Observe the value `101` is returned.

```
Command> SELECT elementId# FROM dual;
< 3 >
1 row found.
Command> SELECT myseq2.NEXTVAL FROM dual;
< 101 >
1 row found.
```

From the connection that is connected to element 2, first verify the connection to element 2, then issue a `SELECT...NEXTVAL` query. Observe the value `201` is returned.

```
Command> SELECT elementId# FROM dual;
< 2 >
1 row found.
Command> SELECT myseq2.NEXTVAL FROM dual;
< 201 >
1 row found.
```

Performing DML operations

TimesTen Scaleout supports the `INSERT`, `DELETE`, and `UPDATE`, and `SELECT` DML operations. The `MERGE` operation is not supported.

All data in all elements is accessible from everywhere. You can query or modify data in any or all elements. Transactions obey ACID rules. TimesTen Scaleout provides read committed semantics for isolation level. Readers do not block writers and writers do not block readers.

Using pseudocolumns

A pseudocolumn is an assigned value used in the same context as a column, but is not stored. Pseudocolumns are not actual columns in a table but behave like columns. You can perform select operations, but you cannot perform insert, update, or delete operations on a pseudocolumn.

Use the `replicaSetId#` pseudocolumn to determine the replica set in which the row is stored. This pseudocolumn returns a `NOT NULL TT_INTEGER` data type.

See "Pseudocolumns in TimesTen Scaleout" in the *Oracle TimesTen In-Memory Database SQL Reference* for information on the additional pseudocolumns supported in TimesTen Scaleout.

Examples include:

- Example 7–6, "Use replicaSetId# to locate data"
- Example 7–7, "Use replicaSetId# with a table that has a duplicate distribution scheme"

Example 7–6 Use replicaSetId# to locate data

This example issues a query on the `customers` table, returning the replica set in which the data is stored (as determined by `replicaSetId#`).

```
Command> SELECT replicasetid#, cust_id,last_name,first_name
         FROM customers WHERE cust_id BETWEEN 910 AND 920
         ORDER BY cust_id, last_name, first_name;
< 2, 910, Riley, Tessa >
< 1, 911, Riley, Rashad >
< 1, 912, Riley, Emma >
< 1, 913, Rivera, Erin >
< 1, 914, Roberts, Ava >
< 1, 915, Roberts, Lee >
< 2, 916, Roberts, Clint >
< 3, 917, Robertson, Faith >
< 2, 918, Robinson, Miguel >
< 2, 919, Robinson, Mozell >
< 3, 920, Rodgers, Darryl >
11 rows found.
```

Example 7–7 Use replicaSetId# with a table that has a duplicate distribution scheme

This example first uses the `ttIsql describe` command on the `account_status` table to validate the table has a duplicate distribution scheme. The example then issues a query to return the `replicasetId#`. The example then repeats the same query from a different connection. The example shows that the data returned is located on the replica set to which the application is connected and thus is present in every element in the database (duplicate distribution scheme).

```
Command> describe account_status;

Table SAMPLEUSER.ACCOUNT_STATUS:
  Columns:
   *STATUS                          NUMBER (2) NOT NULL
    DESCRIPTION                     VARCHAR2 (100) INLINE NOT NULL
 DUPLICATE

1 table found.
(primary key columns are indicated with *)
```

Query the `dual` table to return the replica set to which the application is connected. In this example, the replica set is 1.

```
Command> SELECT replicaSetId# FROM dual;
< 1 >
1 row found.

Command> SELECT replicaSetId#,* FROM account_status;
< 1, 10, Active - Account is in good standing >
< 1, 20, Pending - Payment is being processed >
< 1, 30, Grace - Automatic payment did not process successfully >
< 1, 40, Suspend - Account is in process of being disconnected >
< 1, 50, Disconnected - You can no longer make calls or receive calls >
5 rows found.
```

Issue a second query from a different `ttIsql` session running on a different data instance:

```
Command> SELECT elementid# from dual;
< 6>
1 row found.

Command> SELECT replicaSetId#, * FROM account_status;
< 3, 10, Active - Account is in good standing >
< 3, 20, Pending - Payment is being processed >
< 3, 30, Grace - Automatic payment did not process successfully >
< 3, 40, Suspend - Account is in process of being disconnected >
< 3, 50, Disconnected - You can no longer make calls or receive calls >
5 rows found.
```

Using the TT_CommitDMLOnSuccess hint

The `TT_CommitDMLOnSuccess` hint is used to enable or disable a commit operation as part of DML execution. You can specify the hint at the connection level or at the statement level.

While using this hint (`TT_CommitDMLOnSuccess` set to `1`):

- At statement level, if a statement encounters an error while executing, the transaction remains active and the database consistent.
- For transactions that impact a single replica set, the commit operation uses a one-phase commit instead of a two-phase commit protocol.

There is no difference in performance if you set `autocommit` to `1` or if you set the `TT_CommitDMLOnSuccess` hint to `1`.

See "TT_CommitDMLOnSuccess optimizer hint" in the *Oracle TimesTen In-Memory Database SQL Reference* for detailed information.

Using optimizer hints

The TimesTen query optimizer is a cost-based optimizer that determines the most efficient way to execute a given query by considering possible query plans. A query plan in TimesTen Scaleout is affected by the distribution scheme and the distribution keys of a hash distribution scheme as well as the column and table statistics, the presence or absence of indexes, the volume of data, the number of unique values, and the selectivity of predicates. You can manually examine a query plan by running the `ttIsql explain` command. See "The TimesTen Query Optimizer" in the *Oracle TimesTen In-Memory Database Operations Guide* for more information.

You can use optimizer hints to influence the execution plan generated by the optimizer. There are two optimizer hints that are specific to TimesTen Scaleout. These hints are valid at the statement and the connection levels. At the statement level, the hints are valid for `SELECT` statements only:

- TT_GridQueryExec
- TT_PartialResult

See "Optimizer hints supported in TimesTen Scaleout only" in the *Oracle TimesTen In-Memory Database SQL Reference* for information on the optimizer hints specific to TimesTen Scaleout. See "Use optimizer hints to modify the execution plan" in the *Oracle TimesTen In-Memory Database Operations Guide* for more information on all optimizer hints.

TT_GridQueryExec

The `TT_GridQueryExec` optimizer hint enables you to specify whether the query should return data from the local element or from all elements, including the elements in a replica set when K-safety is set to 2.

If you do not specify this hint, the query is executed in one logical data space. It is neither local nor global. Exactly one full copy of the data is used to compute the query.

Valid options for this hint are `LOCAL` and `GLOBAL`:

- `LOCAL`: TimesTen Scaleout executes the queries in the local element only. Data is retrieved locally from the element to which you are connected. If the local element does not have a full copy of the data, TimesTen Scaleout returns partial results.
- `GLOBAL`: TimesTen Scaleout retrieves data from all elements, including copies of the rows from all tables from all replica sets to generate the results. This results in duplicate data returned if K-safety is set to 2 or if tables have a duplicate distribution scheme.

As with all queries, the element that you are directly connected to and issue the SQL query from prepares the query and sends it to all other elements in the grid. The request is executed on elements that are up and the results are reported locally on the connected element.

See "TT_GridQueryExec optimizer hint" in the *Oracle TimesTen In-Memory Database SQL Reference* for information on the syntax and semantics for this hint.

The distribution scheme is a determining factor in the number of rows returned. For example, Table 7–2 shows the number of rows used in query for the three distribution schemes. *k* represents the number of copies (`k`=2 in our example), *e* represents one element from each replica set (*e*=3 in our example), and *r* represents the number of rows in the table.

Table 7–2 TT_GridQueryExec optimizer hint

Option	Table Type	Number of rows used in query
LOCAL	Duplicate distribution scheme table	*r*
	Distributed by hash table	*r*/*e* (Assumes uniform distribution)
	Distributed by reference table	*r*/*e* (Assumes uniform distribution)
GLOBAL	Duplicate distribution scheme table	*e***k***r*
	Distributed by hash table	*k***r*
	Distributed by reference table	*k***r*

Examples include:

- Example 7–8, "Use TT_GridQueryExec on a hash distribution scheme table"
- Example 7–9, "Use TT_GridQueryExec on a duplicate distribution scheme table"
- Example 7–10, "Use TT_GridQueryExec on a reference distribution scheme table"

Note: Reads do not get a distributed lock and return committed data. For the examples that use the `TT_GridQueryExec(GLOBAL)` optimizer hint, if a write to a replica set happens between the reads to its replicas, it is possible that the count will not match for all replicas. This is expected behavior because each replica is afforded read committed isolation.

Example 7–8 Use TT_GridQueryExec on a hash distribution scheme table

This example uses the `ttIsql describe` command on the `customers` table to illustrate the table is distributed by hash. The example executes a `SELECT COUNT (*)` query on the `customers` table to return the number of rows in the table (1000). From the connection that is connected to element 4, the example uses the `TT_GridQueryExec (Local)` and `(Global)` optimizer hints to return the number of rows. The rows returned differ based on whether `Local` or `Global` was specified in the `TT_GridQueryExec` hint.

```
Command> describe customers;

Table SAMPLEUSER.CUSTOMERS:
  Columns:
   *CUST_ID                         NUMBER (10) NOT NULL
    FIRST_NAME                      VARCHAR2 (30) INLINE NOT NULL
    LAST_NAME                       VARCHAR2 (30) INLINE NOT NULL
    ADDR1                           VARCHAR2 (64) INLINE
    ADDR2                           VARCHAR2 (64) INLINE
    ZIPCODE                         VARCHAR2 (5) INLINE
    MEMBER_SINCE                    DATE NOT NULL
  DISTRIBUTE BY HASH (CUST_ID)

1 table found.
(primary key columns are indicated with *)

Command> SELECT COUNT (*) FROM customers;
< 1000 >
1 row found.
```

Issue a `SELECT elementId# FROM dual` query to determine the local element connection (4).

```
Command> SELECT elementId# FROM dual;
< 4 >
1 row found.
```

From this connection, issue a `SELECT` query supplying the `TT_GridQueryExec(LOCAL)` optimizer hint. Expect approximately 333 rows to be returned (1000/3).

```
Command> SELECT /*+TT_GridQueryExec(LOCAL)*/ COUNT (*), elementId#
         FROM customers GROUP BY elementId#;
< 326, 4 >
1 row found.
```

Now issue a `SELECT` query supplying the `TT_GridQueryExec(GLOBAL)` optimizer hint. Expect 2000 rows returned (`k`=2 * `r`=1000 = 2000). Validate the results by using the `SUM` function to calculate the total rows returned for all 6 elements.

```
Command> SELECT /*+TT_GridQueryExec(GLOBAL)*/ COUNT (*), elementId#
         FROM customers GROUP BY elementId#
         ORDER BY elementId#;
< 338, 1 >
< 338, 2 >
< 326, 3 >
< 326, 4 >
< 336, 5 >
< 336, 6 >
6 rows found.

Command> SELECT SUM (338+338+326+326+336+336) FROM dual;
< 2000 >
1 row found.
```

Validate the total count using the `TT_GridQueryExec(GLOBAL)` hint.

```
Command> SELECT/*+TT_GridQueryExec(GLOBAL)*/ COUNT(*) FROM customers;
< 2000 >
1 row found.
```

Example 7–9   Use TT_GridQueryExec on a duplicate distribution scheme table

This example uses the `ttIsql describe` command on the `account_status` table to illustrate the table is a duplicate distribution scheme. The example executes a `SELECT COUNT (*)` query on the `account_status` table to return the number of rows in the table (5). From the connection that is connected to element 2, the example uses the `TT_GridQueryExec (Local)` and `(Global)` optimizer hints to return the number of rows. The rows return differ based on whether `Local` or `Global` was specified in the `TT_GridQueryExec` hint.

```
Command> describe account_status;
Table SAMPLEUSER.ACCOUNT_STATUS:
  Columns:
   *STATUS                          NUMBER (2) NOT NULL
    DESCRIPTION                     VARCHAR2 (100) INLINE NOT NULL
 DUPLICATE

1 table found.
(primary key columns are indicated with *)

Command> SELECT count (*) FROM account_status;
```

```
< 5 >
1 row found.

Command> SELECT elementId# FROM dual;
< 2 >
1 row found.
```

Issue a `SELECT` query supplying the `TT_GridQueryExec(LOCAL)` optimizer hint. Expect approximately 5 rows to be returned (`r` = 5).

```
Command> SELECT /*+TT_GridQueryExec(LOCAL)*/ COUNT (*),elementId#
         FROM account_status GROUP BY elementId#;
< 5, 2 >
1 row found.
```

Now issue a `SELECT` query supplying the `TT_GridQueryExec(GLOBAL)` optimizer hint. Expect 30 rows returned (`e`=3 *`k`=2 * `r`=5= 30).

```
Command> SELECT /*+TT_GridQueryExec(GLOBAL)*/ COUNT (*),elementId#
         FROM account_status GROUP BY elementId#
         ORDER BY elementId#;
< 5, 1 >
< 5, 2 >
< 5, 3 >
< 5, 4 >
< 5, 5 >
< 5, 6 >
6 rows found.
```

Validate the total count using the `TT_GridQueryExec(GLOBAL)` hint.

```
Command> SELECT /*+TT_GridQueryExec(GLOBAL)*/ COUNT (*) FROM account_status;
< 30 >
1 row found.
```

Example 7–10 Use TT_GridQueryExec on a reference distribution scheme table

This example uses the `ttIsql describe` command on the `accounts` table to illustrate the table is distributed by reference. The example executes a `SELECT COUNT (*)` query on the `accounts` table to return the number of rows in the table (1010). From the connection that is connected to element 1, the example uses the `TT_GridQueryExec (Local)` and `(Global)` optimizer hint to return the number of rows. The rows returned differ based on whether `Local` or `Global` was specified in the `TT_GridQueryExec` hint.

```
Command> describe accounts;
Table SAMPLEUSER.ACCOUNTS:
  Columns:
   *ACCOUNT_ID                      NUMBER (10) NOT NULL
    PHONE                           VARCHAR2 (15) INLINE NOT NULL
    ACCOUNT_TYPE                    CHAR (1) NOT NULL
    STATUS                          NUMBER (2) NOT NULL
    CURRENT_BALANCE                 NUMBER (10,2) NOT NULL
    PREV_BALANCE                    NUMBER (10,2) NOT NULL
    DATE_CREATED                    DATE NOT NULL
    CUST_ID                         NUMBER (10) NOT NULL
  DISTRIBUTE BY REFERENCE (FK_CUSTOMER)
1 table found.
(primary key columns are indicated with *)

Command> SELECT COUNT (*) FROM accounts;
< 1010 >
```

```
1 row found.

Command> SELECT elementId# FROM dual;
< 1 >
1 row found.
```

Issue a `SELECT` query supplying the `TT_GridQueryExec(LOCAL)` optimizer hint. Expect approximately 336 rows to be returned (1010/3).

```
Command> SELECT /*+TT_GridQueryExec(LOCAL)*/ COUNT (*), elementId#
         FROM accounts GROUP BY elementId#;
< 339, 1>
1 row found.
```

Now issue a `SELECT` query supplying the `TT_GridQueryExec(GLOBAL)` optimizer hint. Expect 2020 rows returned (`k`=2 * `r`=1010 = 2020). Validate the results by using the `SUM` function to calculate the total rows returned for all 6 elements.

```
Command> SELECT /*+TT_GridQueryExec(GLOBAL)*/ COUNT (*), elementId#
         FROM accounts GROUP BY elementId#
         ORDER BY elementId#;
< 339, 1 >
< 339, 2 >
< 332, 3 >
< 332, 4 >
< 339, 5 >
< 339, 6 >
6 rows found.

Command> SELECT SUM (339+339+332+332+339+339) FROM dual;
< 2020 >
1 row found.
```

Validate the total count using the `TT_GridQueryExec(GLOBAL)` hint.

```
Command> SELECT/*+TT_GridQueryExec(GLOBAL)*/ COUNT(*) FROM accounts;
< 2020 >
1 row found.
```

TT_PartialResult

The `TT_PartialResult` optimizer hint enables you to specify whether the query should return partial results or error if data is not available.

Use `TT_PartialResult(1)` to direct the query to return partial results if all elements in a replica set are not available.

Use `TT_PartialResult(0)` to direct the query to return an error if the required data is not available in the case where all elements in a replica set are not available. If at least one element from each replica set is available or the data required by the query is available, the optimizer returns the query result correctly without error.

The default is `TT_PartialResult(0)`.

See "TT_PartialResult optimizer hint" in the *Oracle TimesTen In-Memory Database SQL Reference* for information on the syntax and semantics for this hint.

Example 7–11 Examine results using TT_PartialResult

In this example, select the `elementId#`, `replicaSetId#`, and `dataspaceId#` pseudocolumns to locate the row of data involved in the query. Force elements 3 and 4 to be unavailable. Set `TT_PartialResult` to 0 to return an error if the replica set is unavailable. Then, set `TT_PartialResult` to 1 to return partial results from the elements that are available.

```
Command> SELECT elementId#,replicasetId#,dataspaceId#, last_name,first_name
         FROM customers WHERE last_name LIKE ('%Wh%') ORDER BY last_name;
< 6, 3, 2, Whitaker, Armand >
< 4, 2, 2, Whitaker, Ariel >
< 6, 3, 2, White, Carlene >
< 6, 3, 2, White, Marcelo >
< 4, 2, 2, White, Dona >
< 4, 2, 2, White, Ellyn >
< 4, 2, 2, White, Nora >
< 4, 2, 2, White, Phylis >
8 rows found.

Command> SELECT /*+TT_PartialResult(0)*/ elementId#,replicasetId#,dataspaceId#,
           last_name,first_name FROM customers
         WHERE last_name like ('%Wh%') ORDER BY last_name;
< 6, 3, 2, Whitaker, Armand >
< 4, 2, 2, Whitaker, Ariel >
< 6, 3, 2, White, Carlene >
< 6, 3, 2, White, Marcelo >
< 4, 2, 2, White, Dona >
< 4, 2, 2, White, Ellyn >
< 4, 2, 2, White, Nora >
< 4, 2, 2, White, Phylis >
8 rows found.

Command> SELECT /*+TT_PartialResult(1)*/ elementId#,replicasetId#,dataspaceId#,
           last_name,first_name FROM customers
         WHERE last_name LIKE ('%Wh%') ORDER BY last_name;
< 6, 3, 2, Whitaker, Armand >
< 4, 2, 2, Whitaker, Ariel >
< 6, 3, 2, White, Carlene >
< 6, 3, 2, White, Marcelo >
< 4, 2, 2, White, Dona >
< 4, 2, 2, White, Ellyn >
< 4, 2, 2, White, Nora >
< 4, 2, 2, White, Phylis >
8 rows found.
```

Element 4 is no longer available. Expect same results. Element 3 is available.

```
Command> SELECT /*+TT_PartialResult(1)*/ elementId#,replicasetId#,dataspaceId#,
           last_name,first_name FROM customers
          WHERE last_name LIKE ('%Wh%') ORDER BY last_name;
< 6, 3, 2, Whitaker, Armand >
< 3, 2, 1, Whitaker, Ariel >
< 6, 3, 2, White, Carlene >
< 6, 3, 2, White, Marcelo >
< 3, 2, 1, White, Dona >
< 3, 2, 1, White, Ellyn >
< 3, 2, 1, White, Nora >
< 3, 2, 1, White, Phylis >
8 rows found.

Command> SELECT /*+TT_PartialResult(0)*/ elementId#,replicasetId#,dataspaceId#,
           last_name,first_name FROM customers
         WHERE last_name LIKE ('%Wh%') ORDER BY last_name;
< 6, 3, 2, Whitaker, Armand >
< 3, 2, 1, Whitaker, Ariel >
```

```
< 6, 3, 2, White, Carlene >
< 6, 3, 2, White, Marcelo >
< 3, 2, 1, White, Dona >
< 3, 2, 1, White, Ellyn >
< 3, 2, 1, White, Nora >
< 3, 2, 1, White, Phylis >
8 rows found.
```

Now element 3 becomes unavailable. Replica set 2 is unavailable. Expect `TT_PartialResult` set to 1 to return partial results. Expect `TT_PartialResult` set to 0 to return an error.

```
Command> SELECT /*+TT_PartialResult(1)*/ elementId#,replicasetId#,dataspaceId#,
           last_name,first_name FROM customers
         WHERE last_name LIKE ('%Wh%') ORDER BY last_name;
< 6, 3, 2, Whitaker, Armand >
< 6, 3, 2, White, Carlene >
< 6, 3, 2, White, Marcelo >
3 rows found.

Command> SELECT /*+TT_PartialResult(0)*/ elementId#,replicasetId#,dataspaceId#,
           last_name,first_name FROM customers
         WHERE last_name LIKE ('%Wh%') ORDER BY last_name;
 3723: Replica set 2 down
The command failed.
```

Understanding ROWID in data distribution

TimesTen Scaleout requires a unique id for row distribution. It uses `ROWID` to ensure uniqueness across all elements.

For tables with a duplicate distribution scheme where K-safety is set to 1 and for all tables (no matter what the distribution scheme is) where K-safety is set to 2, the physical location of each copy of a row is different, so each copy of the row has different `ROWID` values. In this case, when using `ROWID` based access, TimesTen Scaleout returns the value of the `ROWID` in the first data space. If the row in the first data space is not available, TimesTen Scaleout returns the `ROWID` in the next (second) data space.

Since `ROWID` is the identifier of a specific copy of a row, if that copy is not available, you cannot access the row by `ROWID`. In this case, you should access the row by primary key.

See "ROWID pseudocolumn" in the *Oracle TimesTen In-Memory Database SQL Reference* for more information.

Note: Applications should not store `ROWID` values in the database and try to use these values later. Applications can fetch the `ROWID` in a transaction and then use the `ROWID` later in the same transaction.

Understanding system views

There are several local (`V$`) global (`GV$`) system views you can query to retrieve metadata information about your database.

- The `V$` views contain data for the element to which your application is connected.
- The `GV$` views contain the contents of the `V$` view for every element of the database.

In addition, there are several views that you can query that are based on TimesTen built-in procedures. See "System Tables and Views" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference* for more information.

8

Maintaining and Upgrading a Grid

This chapter discusses how to maintain and modify a grid in TimesTen Scaleout.

- Maintaining the model of a grid
- Modifying a grid
- Redistributing data in a database
- Stopping a grid
- Restarting a grid
- Upgrading a grid
- Destroying a grid

Note:

- The following sections consider the grid and database generated by the examples found in "Configure your grid" on page 4-1 and "Creating a database" on page 5-1 as the grid and database configuration on which the commands are run.
- All the tasks described in the next sections require that you run the `ttGridAdmin` utility from the active management instance as the instance administrator, unless stated otherwise.

Maintaining the model of a grid

The model is a comprehensive list of the objects that give shape to a grid. Depending of the version of the model, the model may either describe a previous, present, or desired structure of a grid.

The `ttGridAdmin` utility has several commands that enable you to review any stored version of the model:

- Compare different versions of the model
- Export a version of the model
- Import a model as the latest version of the model
- List the available versions of the model

For more information on the different versions of the model or model operations, see "Model versioning" on page 4-18 in this document or "Model operations" in the *Oracle TimesTen In-Memory Database Reference*, respectively.

Modifying a grid

TimesTen Scaleout defines several different types of objects in the model to give shape to a grid:

- Data space groups
- Hosts
- Installations
- Instances
- Physical groups

Additionally, there are two types of model objects that describe the databases that the grid manages and that, in conjunction, define the names by which you connect to these databases. These types of objects are:

- Database definitions
- Connectables

Note: See "Central configuration of the grid" on page 1-16 for a complete list of the types of model objects and their descriptions.

You can create, modify, or delete objects in the model. Consider that changes you make to the model only take effect after you apply them to the current version of the model.

Note:

- See "Configure your grid" on page 4-1 for details on how to create the objects that give shape to a grid.
- See "Creating a database" on page 5-1, "Modifying the connection attributes of a database" on page 5-30, or "Destroying a database" on page 5-33 for details on how to create, modify, or delete the objects that define a database, respectively.
- See "Applying the changes made to the model" on page 4-18 for details on the versions of the model and applying changes to the current version of the model.

The following sections describe how to modify or delete the objects that give shape to a grid:

- Modifying objects in a grid
- Deleting objects from a grid
- Reconfiguring membership servers

Modifying objects in a grid

Of the objects in the model that give shape to a grid, only hosts and instances can be modified. Physical groups or installations can only be deleted. These sections describe how to modify the attributes of the hosts and instances in a grid:

- Modify a host
- Modify an instance

Modify a host

You can modify certain attributes of a host with `ttGridAdmin hostModify` command. The name and communication parameters (internal or external DNS names or IP addresses) of a host cannot be modified. Once you assign a host to a data space group and apply that assignment to the current version of the model, you cannot change it. You can modify the assignments to physical groups at any time. However, once a host is assigned to a data space group, its physical group assignments are no longer relevant.

See "Assigning hosts to physical groups" on page 4-22 for an example where several hosts are modified.

For more information on the `ttGridAdmin hostModify` command, see "Modify a host (hostModify)" in the *Oracle TimesTen In-Memory Database Reference*.

Modify an instance

You can modify the installation associated with an instance with `ttGridAdmin instanceModify` command. Also, this command enables you to modify the TCP/IP port number of the replication agent of a management instance, but only if there is not a second management instance available. In other words, you can only modify the TCP/IP port number of the replication agent of a management instance if the port is not in use. See "Upgrade a grid to a patch-compatible release" on page 8-19 for an example where several instances are modified.

For more information on the `ttGridAdmin instanceModify` command, see "Modify an instance (instanceModify)" in the *Oracle TimesTen In-Memory Database Reference*.

Deleting objects from a grid

This section describes how to delete objects from a grid:

- Delete an instance
- Delete an installation
- Delete a host
- Delete a physical group

Delete an instance

Depending on the type of instance you want to delete, follow either of the next procedures:

- Delete a data instance
- Delete a management instance

Delete a data instance Before you can delete a data instance from a grid, you need to remove the element of the data instance from the distribution map of every database, as shown next:

1. Remove the element of the data instance from the distribution map of every database as shown in "Removing elements from the distribution map" on page 8-12.
2. Delete the data instance from the latest version of the model.

```
% ttGridAdmin instanceDelete host7.instance1
Instance instance1 on Host host7 deleted from Model
```

3. Apply the changes made to the latest version of the model.

```
% ttGridAdmin modelApply
...
Identifying any deleted objects.....................................OK
Stopping deleted instances..........................................OK
Deleting instances..................................................OK
...
ttGridAdmin modelApply complete
```

For more information on the `ttGridAdmin instanceDelete` command, see "Delete an instance (instanceDelete)" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin modelApply` command, see "Applying the changes made to the model" on page 4-18 in this document and "Model operations" in the *Oracle TimesTen In-Memory Database Reference*.

Delete a management instance Only the standby management instance can be deleted from a grid. If you intend to delete the active management instance in a grid with two management instances, first switch the standby management instance to active with the `ttGridAdmin mgmtActiveSwitch` command, then proceed.

> **Important:** For availability, we highly recommend that you always have an active and a standby management instance in your grid. Only delete the standby management instance if you intend to replace it with another one as soon as possible.

If you intend to delete the active management instance in a grid with only one management instance, consider destroying the grid in a graceful manner. See "Destroying a grid" on page 8-26 for details on how to gracefully destroy a grid.

To delete the standby management instance from a grid, perform these tasks:

1. Confirm that the instance you want to delete is the standby management instance.

```
% ttGridAdmin mgmtStatus
Host  Instance  Reachable RepRole(Self) Role(Active) Role(Self) Seq RepAgent
RepActive Message
----- --------- --------- ------------- ------------ ---------- --- --------
--------- -------
host1 instance1 Yes       Active        Unknown      Active     338 Up
Yes
host2 instance1 Yes       Standby       Unknown      Standby    338 Up       No
```

2. Delete the standby management instance from the latest version of the model.

```
% ttGridAdmin instanceDelete host2.instance1
Instance instance1 on Host host2 deleted from Model
```

3. Apply the changes made to the latest version of the model.

```
% ttGridAdmin modelApply
...
Unconfiguring standby management instance...........................OK
Identifying any deleted objects.....................................OK
Stopping deleted instances..........................................OK
Deleting instances..................................................OK
...
ttGridAdmin modelApply complete
```

For more information on the `ttGridAdmin mgmtActiveSwitch` command, see "Starting, stopping and switching management instances" on page 11-35 in this document and "Switch the active management instance (mgmtActiveSwitch)" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin instanceDelete` command, see "Delete an instance (instanceDelete)" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin modelApply` command, see "Applying the changes made to the model" on page 4-18 in this document and "Model operations" in the *Oracle TimesTen In-Memory Database Reference*.

Delete an installation

You may want to delete an installation if you just performed an upgrade operation to a new release of TimesTen Scaleout. Deleting an installation does not remove the installation files, since the files may be still in use if the location of the files is shared by other installations in this or any other grid. See "Upgrade a grid to a patch-compatible release" on page 8-19 for more information on upgrade and cleanup operations, which includes deleting the previous release installation model object and files.

However, if you are deleting an installation because you are removing its associated host from the topology of the grid, see "Delete a host" on page 8-5 for details on how to delete a host and its associated objects, which includes the installation model object and files.

Delete a host

Before you can delete a host from a grid, you must ensure that other model objects associated with the host are not in use, as shown next:

1. Remove the element of every data instance associated with the host from the distribution map of every database, as shown in "Removing elements from the distribution map" on page 8-12.
2. Delete every instance and installation associated with the host, and then, delete the host from the latest version of the model. You can either delete each object separately, as shown in Example 8–1, or use the `-cascade` of option of the `ttGridAdmin hostDelete` command to delete the host and every instance and installation associated with it, as shown in Example 8–2.

Example 8–1 Delete a host and all its associated objects separately.

```
% ttGridAdmin instanceDelete host7.instance1
Instance instance1 on Host host7 deleted from Model

% ttGridAdmin installationDelete host7.installation1
Installation installation1 on Host host7 deleted from Model

% ttGridAdmin hostDelete host7
Host host7 deleted from Model
```

Example 8–2 Delete a host and all its associated objects

```
% ttGridAdmin hostDelete host8 -cascade
Instance instance1 on Host host8 deleted from Model
Installation installation1 on Host host8 deleted from Model
Host host8 deleted from Model
```

3. Apply the changes made to the latest version of the model.

```
% ttGridAdmin modelApply
...
Identifying any deleted objects.....................................OK
Stopping deleted instances..........................................OK
Deleting instances..................................................OK
Deleting installations from model...................................OK
Deleting any hosts that are no longer in use........................OK
...
ttGridAdmin modelApply complete
```

4. If the installation files associated with the installation model objects you just deleted are not in use by any other installation object in this or any other grid, then delete the files. Ensure that you change the permissions of the directory so that you can delete all files.

```
% cd /grid
% chmod -R 750 tt18.1.4.1.0/
% rm -rf tt18.1.4.1.0/
```

For more information on the `ttGridAdmin instanceDelete`, `ttGridAdmin installationDelete`, or `ttGridAdmin hostDelete` command, see "Delete an instance (instanceDelete)", "Delete an installation (installationDelete)", or "Delete a host (hostDelete)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin modelApply` command, see "Applying the changes made to the model" on page 4-18 in this document and "Model operations" in the *Oracle TimesTen In-Memory Database Reference*.

Delete a physical group

Before you can delete a physical group from a grid, you must ensure that it is not in use, as shown next:

1. Remove the physical group from every host associated with it.

```
% ttGridAdmin hostModify host3 -removePhysicalGroup rack1
Host host3 modified in Model
```

2. Delete the physical group form the latest version of the model.

```
% ttGridAdmin physicalDelete rack1
PhyscialGroup RACK1 deleted.
```

3. Apply the changes made to latest version of the model.

```
% ttGridAdmin modelApply
```

For more information on the `ttGridAdmin hostModify` or `ttGridAdmin physicalDelete` command, see "Modify a host (hostModify)" or "Delete a physical group (physicalDelete)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin modelApply` command, see "Applying the changes made to the model" on page 4-18 in this document and "Model operations" in the *Oracle TimesTen In-Memory Database Reference*.

Reconfiguring membership servers

These sections describe how to view and modify your current membership configuration:

- View the current membership configuration

- Add membership servers
- Enable the new membership configuration

For more information on membership servers, see the Apache ZooKeeper documentation at `http://zookeeper.apache.org`.

View the current membership configuration

To view your current membership configuration, run the `ttGridAdmin membershipConfigExport` command. This lists the membership servers and the ports used.

```
% ttGridAdmin membershipConfigExport
Servers ms_host1!2181,ms_host2!2181,ms_host3!2181
```

For more information on the `ttGridAdmin membershipConfigExport` command, see "Export the membership configuration file (membershipConfigExport)" in the *Oracle TimesTen In-Memory Database Reference*.

Add membership servers

You can add a new server to the list of membership servers to reflect your desired membership configuration. To add the `ms_host4` server and its client port `2181`:

1. Create a new server configuration file, for example, `membership2.conf`. For more information on the ZooKeeper client configuration file, see "Configuring Apache ZooKeeper as the membership service" on page 3-7.
2. Append the new membership server and port to the current list of membership servers.

   ```
   Servers ms_host1!2181,ms_host2!2181,ms_host3!2181,ms_host4!2181
   ```

Enable the new membership configuration

To enable your new membership configuration, perform these tasks:

1. Replace the ZooKeeper client configuration file in the latest version of the model with the newly created file.

   ```
   % ttGridAdmin membershipConfigImport membership2.conf
   Membership configuration file membership2.conf imported
   ```

2. Run the `ttGridAdmin modelApply` command to apply the changes to the latest version of the model.

   ```
   % ttGridAdmin modelApply
   Creating new model version...........................................OK
   Exporting current model (version 3).................................OK
   Identifying any changed management instances.........................OK
   Identifying any deleted objects......................................OK
   Verifying installations..............................................OK
   Verifying instances..................................................OK
   Updating grid state..................................................OK
   Pushing new configuration files to each instance.....................OK
   Making model version 3 current, version 4 writable...................OK
   ttGridAdmin modelApply complete
   ```

3. Stop and restart every instance in the grid. For more information on stopping and restarting a grid, see "Stopping a grid" on page 8-17 and "Restarting a grid" on page 8-17 respectively.

For more information on the `ttGridAdmin membershipConfigImport` command, see "Import the membership configuration file (membershipConfigImport)" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin modelApply` command, see "Applying the changes made to the model" on page 4-18 in this document and "Model operations" in the *Oracle TimesTen In-Memory Database Reference*.

Redistributing data in a database

You can increase or decrease the number of elements in which your data is distributed. However, this requires more than just adding or removing data instances from the current version of the model; you must also add or remove the elements of the data instances from the distribution map of the database.

The different tasks for maintaining the distribution map of a database are:

- *Add a replica set to the distribution map*. When you add a replica set to the distribution map (and the distribution map is applied), TimesTen Scaleout re-distributes a portion of the data in the elements of each replica set to the elements of the newly added replica set.
- *Remove a replica set without a replacement from the distribution map*. If the removed replica set is not replaced with another replica set, when the distribution map is applied, the data stored in the elements of the removed replica set is evenly re-redistributed into the elements of the remaining replica sets.
- *Remove a data instance and replace it with another data instance that is not already defined in the distribution map*. In this case, when the distribution map is applied, the data is copied from the element of the removed data instance to the element of the new data instance; the data stored in the elements of the other replica sets is not re-distributed.
- *Evict a replica set from the distribution map*. If all elements in a replica set have unrecoverable failures, evict the replica set from the distribution map. Evicting a replica set results in data loss. When you evict a replica set from the distribution map, you can either:
  - *Evict the replica set without a replacement*. If the evicted replica set is not replaced with another replica set, when the distribution map is applied, the data in the evicted replica set is lost and the data stored in the elements of the other replica sets is not re-distributed.
  - *Evict and replace the replica set with another replica set that is not already defined in the distribution map*. When the distribution map is applied, since the data in the elements of the evicted replica set is lost, the element of the new replica set is empty and the data stored in the elements of the other replica sets is not re-distributed.

    See "Recovering when the replica set has a permanently failed element" on page 11-20 for information on how to evict failed replica sets from the distribution map.

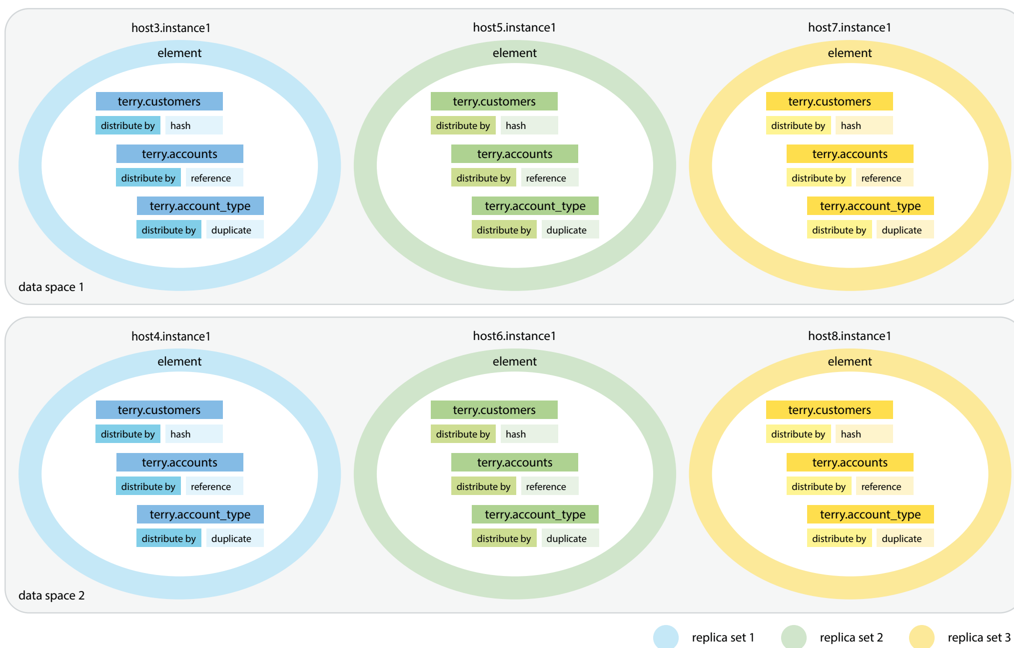
The `ttGridAdmin dbDistribute` command can add and remove elements and evict replica sets from the distribution map of a database, then redistribute existing data across the resulting replica sets. Your existing data is redistributed once you apply the change to distribution map with the `ttGridAdmin dbDistribute -apply` command.

Note: Data distribution cannot execute concurrently with DDL or DML statements. As a result, the `ttGridAdmin dbDistribute -apply` command terminates with an error if you are currently executing any DDL or DML statements that insert, update, or delete data. Any DML statements that insert, update or delete while data distribution is in process are blocked until data distribution completes. However, you can execute any read-only statements while data distribution is in process.

- Adding elements to the distribution map
- Removing elements from the distribution map

Figure 8–1 shows the database schema and topology of the elements of the `database1` database that the examples in the following sections use.

Figure 8–1 Data spaces and replica sets



Adding elements to the distribution map

To increase the number of elements in which your data is distributed, you need to first increase the number of data instances associated with the grid. Also, you must ensure that you have the same number of data instances to each data space group. For example, in a grid with `k` set to `2`, you must add an equal number of data instances to data space group `1` and data space group `2`.

If you are adding elements to the distribution map of the database with the intention of increasing the amount of memory available in the permanent memory region, consider increasing the size of the permanent memory region instead. You can accomplish this by modifying the value of the `PermSize` attribute.

Note:

- Every host with a data instance must have enough physical memory available to support the value of the `PermSize` attribute. See "Determining the value of the PermSize attribute" on page 5-19 and "Modify the connection attributes in a database definition" on page 5-30 for more information on how to calculate and modify the value of the `PermSize` attribute.
- Consider that even when rows are re-distributed to the elements of the new data instances, the memory previously used by these rows in their original elements is still in use by a table page and can only be used by new rows of the same table.

Add a data instance for each data space group available to the current version of the model.

```
% ttGridAdmin hostCreate -internalAddress int-host9.example.com -externalAddress
 ext-host9.example.com -like host3 -cascade -dataSpaceGroup 1
Host host9 created in Model
Installation installation1 created in Model
Instance instance1 created in Model

% ttGridAdmin hostCreate -internalAddress int-host10.example.com -externalAddress
 ext-host10.example.com -like host3 -cascade -dataSpaceGroup 2
Host host10 created in Model
Installation installation1 created in Model
Instance instance1 created in Model

% ttGridAdmin modelApply
...
Verifying installations...............................................OK
Creating new installations............................................OK
Verifying instances...................................................OK
Creating new instances................................................OK
...
Checking ssh connectivity of new instances............................OK
Starting new data instances...........................................OK
ttGridAdmin modelApply complete
```

Note: See "Adding data instances" on page 4-11 and "Applying the changes made to the model" on page 4-18 for more information on how to add data instances to a grid.

Figure 8–2 shows an example of the hash distribution of the `customers` table in the `database1` database. Notice that the element of the `host9.instance1` data instance is empty. Even though the `host9` host is assigned to data space group `1`, element `7` is not considered part of data space `1` until the `host9.instance1` data instance is added to the distribution map of the `database1` database.

Figure 8–2 Data distribution of a table

Add the element of the `host9.instance1` data instance to the distribution map of the `database1` database.

```
% ttGridAdmin dbDistribute database1 -add host9.instance1
Element host9.instance1 has been marked to be added
Distribution map change enqueued
```

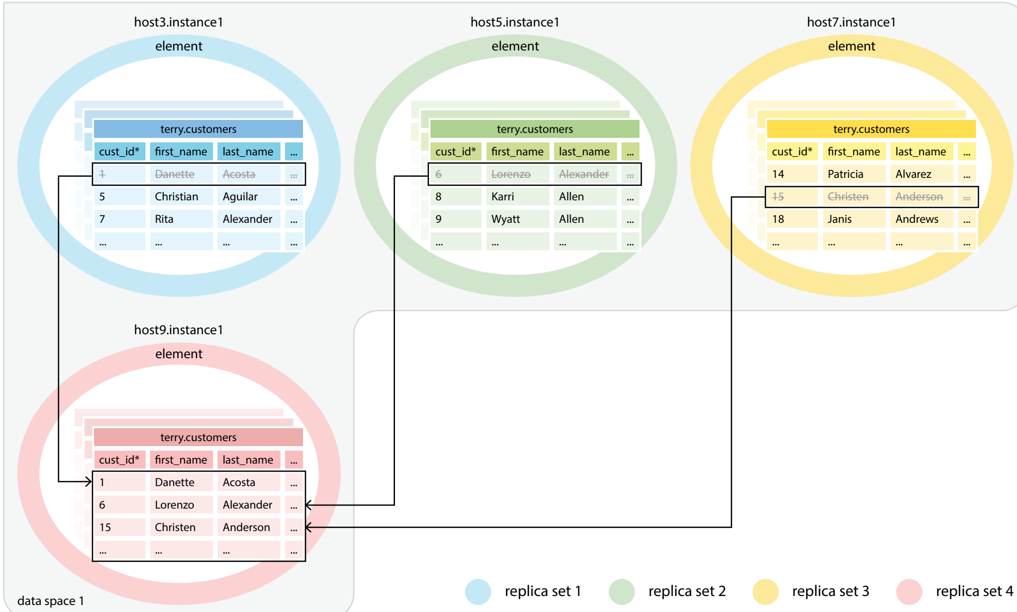
To ensure that the distribution map of the database remains balanced, add the element of the data instance that will hold the replica of the element of the `host9.instance1` data instance, `host10.instance1`, to the distribution map of the `database1` database.

```
% ttGridAdmin dbDistribute database1 -add host10.instance1 -apply
Element host10.instance1 has been marked to be added
Distribution map updated
```

> **Note:** Ensure that you only use the `-apply` option when you are done adding all new elements to the distribution map of the database to avoid TimesTen Scaleout returning an error.

Figure 8–3 shows how some of the data stored in the elements inside data space 1 in Figure 8–2 is re-distributed into the element of the new data instance, `host9.instance1`.

Figure 8–3 Data distribution after adding an element (and its replica)



You can verify the progress of the redistribution operation from any element of the database with the `ttDistributionProgress` built-in procedure.

```
Command> call ttDistributionProgress();
< 2018-12-04 14:49:48.872975, 1, 2, 1, Data Checkpoint, <NULL>, <NULL>, <NULL>,
 <NULL>, <NULL>, 1910, 0, 176, 1910, 8, 8 >
1 row found.
```

For more information on the `ttGridAdmin hostCreate` or `ttGridAdmin dbDistribute` command, see "Create a host (hostCreate)" or "Set or modify the distribution scheme of a database (dbDistribute)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin modelApply` command, see "Applying the changes made to the model" on page 4-18 in this document and "Model operations" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttDistributionProgress` built-in procedure, see "ttDistributionProgress" in the *Oracle TimesTen In-Memory Database Reference*.

Removing elements from the distribution map

You can remove and replace elements from the distribution map with the following in mind:

- Remove and replace a single element:
  - If you have a grid where `k` is set to `1`, you can remove and replace the element only if both the element and data instance are operational.
  - If you have a grid where `k` is set to `2`, you can remove and replace a single element within a replica set by removing the element and replacing it with another element as long as the other element in the replica set is operational.

    See "Replace an element with another element" on page 8-13 and "Remove a replica set" on page 8-15 for more information on how to use the `ttGridAdmin dbDistribute` command with the `-remove` option.

> **Note:** "Remove and replace a failed element in a replica set" on page 11-15 has more information on how to resolve failure issues of a single element within a replica set.

- Evict an entire replica set:
  - If all the elements of a replica set have failed, then the data stored in the replica set is unavailable. "Recovering when the replica set has a permanently failed element" on page 11-20 describes what happens when a replica set fails, how TimesTen Scaleout recovers the replica set, or how you can evict the entire replica set if the elements in the replica set cannot be automatically recovered.

The `ttGridAdmin dbDistribute` command with the `-remove` option removes an element from the distribution map of a database. When you remove a an element from the distribution map of a database, you have these options:

- Replace an element with another element
- Remove a replica set

Replace an element with another element

If the removed element is replaced with the element of a new data instance and you apply this change to the distribution map of the database, the data in the replica set is copied to the element of the new data instance. The data stored in the other replica sets is not re-distributed. Consider doing this when you want to replace a host with another one or a host must be shut down, but you do not want to modify the way your data is being distributed.

Add a data instance to the current version of the model.

```
% ttGridAdmin hostCreate -internalAddress int-host9.example.com -externalAddress
 ext-host9.example.com -like host3 -cascade -dataSpaceGroup 1
Host host9 created in Model
Installation installation1 created in Model
Instance instance1 created in Model

% ttGridAdmin modelApply
...
Verifying installations...............................................OK
Creating new installations............................................OK
Verifying instances...................................................OK
Creating new instances................................................OK
...
Checking ssh connectivity of new instances............................OK
Starting new data instances...........................................OK
ttGridAdmin modelApply complete
```

> **Note:** See "Adding data instances" on page 4-11 and "Applying the changes made to the model" on page 4-18 for more information on how to add data instances to a grid.

Figure 8–4 shows an example of the hash distribution of the `customers` table in the `database1` database. Notice that the element of the `host9.instance1` data instance is empty. Even though the `host9` host is assigned to data space group `1`, its element is not part of a replica set until it is added to the distribution map of the `database1` database.

Figure 8–4 Data distribution of a table

Remove the element of the `host7.instance1` data instance and replace it with the element of the `host9.instance1` data instance in the distribution map of the `database1` database.

```
% ttGridAdmin dbDistribute database1 -remove host7.instance1
 -replaceWith host9.instance1 -apply
Element host7.instance1 has been marked to be removed and replaced by element
host9.instance1
Distribution map updated
```

Figure 8–5 shows how the data previously stored in the element of the `host7.instance1` data instance is copied to its replacement.

Figure 8–5 Data distribution after replacing an element

To destroy the checkpoints and transaction logs of the removed element, use the `ttGridAdmin dbDestroy -instance` command.

```
% ttGridAdmin dbDestroy database1 -instance host7.instance1
Database database1 instance host7 destroy started
```

For more information on the `ttGridAdmin dbDistribute` or `ttGridAdmin dbDestroy` command, see "Set or modify the distribution scheme of a database (dbDistribute)" or "Destroy a database (dbDestroy)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Remove a replica set

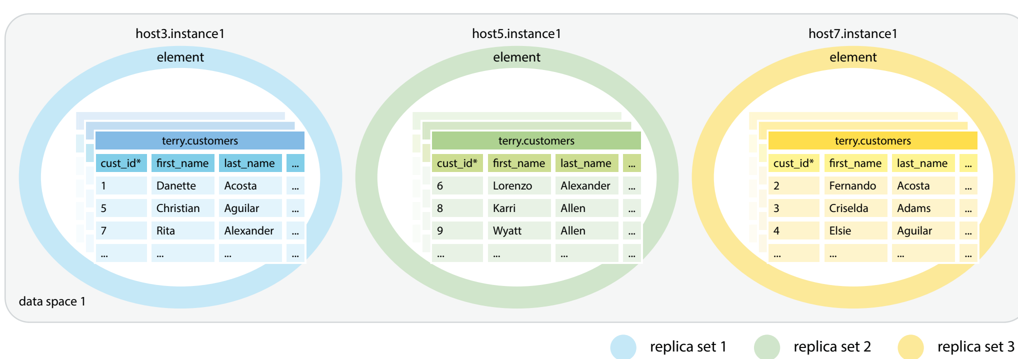
If you remove the element of a data instance without a replacement from the distribution map of a database, you must also remove its replica. In other words, you must remove the replica set in its entirety. When you remove a replica set, TimesTen Scaleout re-distributes the data stored in the replica set to the remaining replica sets. Consider doing this when you want to scale down the number of hosts in which your data is stored.

Important: Consider that the database size is defined by the value of the `PermSize` attribute times the number of replica sets available. Removing one replica set from the distribution map of the database will remove as many MB from the database size as MB set in the `PermSize` attribute. See "Determining the value of the PermSize attribute" on page 5-19 for more information on how to determine the database size of a database.

Before removing a replica set, ensure that the remaining replica sets will have enough space to store a portion of the data stored in the replica set you are about to remove. If necessary, increase the database size by increasing the value of the `PermSize` attribute. See "Modify the connection attributes in a database definition" on page 5-30 for more information on how to increase the value of the `PermSize` attribute.

Figure 8–6 shows an example of the hash distribution of the `customers` table in the `database1` database.

Figure 8–6 Data distribution of a table



Remove the element of the `host7.instance1` data instance from the distribution map of the `database1` database.

```
% ttGridAdmin dbDistribute database1 -remove host7.instance1
Element host7.instance1 has been marked to be removed
```

```
Distribution map change enqueued
```

To ensure that the distribution map of the database remains balanced, remove the element of the data instance holding the replica of the element of the `host7.instance1` data instance from the distribution map of the `database1` database.

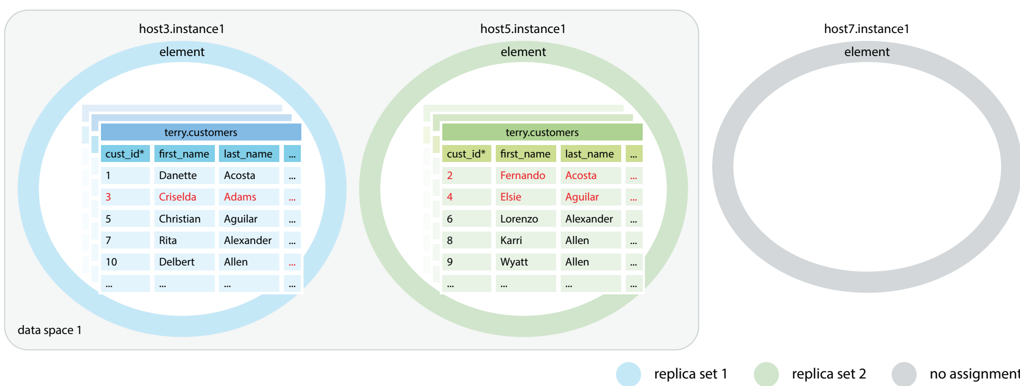
```
% ttGridAdmin dbDistribute database1 -remove host8.instance1 -apply
Element host8.instance1 has been marked to be removed
Distribution map updated
```

Note:

- To find out which data instance holds the replica of the element of another data instance, use the `ttGridAdmin dbStatus` command while specifying the `-replicaset` option.
- Ensure that you only use the `-apply` option when you are done removing all the necessary data instances from the distribution map of the database to avoid TimesTen Scaleout returning an error.

Figure 8–7 shows how removing a replica set from the distribution map of a database removes its elements from their previously assigned data spaces. The figure also shows how the data previously stored in the removed replica set is re-distributed to the replica sets still within each data space.

Figure 8–7 Data distribution after removing a replica set



To destroy the checkpoints and transaction logs of the removed replica set, use the `ttGridAdmin dbDestroy -instance` command.

```
% ttGridAdmin dbDestroy database1 -instance host7.instance1
Database database1 instance host7 destroy started
% ttGridAdmin dbDestroy database1 -instance host8.instance1
Database database1 instance host8 destroy started
```

For more information on the `ttGridAdmin dbDistribute` or `ttGridAdmin dbDestroy` command, see "Set or modify the distribution scheme of a database (dbDistribute)" or "Destroy a database (dbDestroy)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Stopping a grid

Gracefully stopping a grid can only occur if the grid has no loaded databases. Once you ensure that all databases are unloaded, you can proceed to stop the grid, as shown next:

1. Unload all databases. See "Unloading a database from memory" on page 5-27 for details.
2. Stop all data instances.

```
% ttGridAdmin instanceExec -type data ttDaemonAdmin -stop
Overall return code: 0
Commands executed on:
  host3.instance1 rc 0
  host4.instance1 rc 0
  host5.instance1 rc 0
  host6.instance1 rc 0
  host7.instance1 rc 0
  host8.instance1 rc 0
Return code from host3.instance1: 0
Output from host3.instance1:
TimesTen Daemon (PID: 4498, port: 6624) stopped.
Return code from host4.instance1: 0
Output from host4.instance1:
TimesTen Daemon (PID: 4536, port: 6624) stopped.
Return code from host5.instance1: 0
Output from host5.instance1:
TimesTen Daemon (PID: 4492, port: 6624) stopped.
Return code from host6.instance1: 0
Output from host6.instance1:
TimesTen Daemon (PID: 4510, port: 6624) stopped.
Return code from host7.instance1: 0
Output from host7.instance1:
TimesTen Daemon (PID: 4539, port: 6624) stopped.
Return code from host8.instance1: 0
Output from host8.instance1:
TimesTen Daemon (PID: 4533, port: 6624) stopped.
```

3. If there is an standby management instance, stop it.

```
% ttGridAdmin mgmtStandbyStop
Standby management instance host2.instance1 stopped
```

4. Stop the active management instance.

```
% ttGridAdmin mgmtActiveStop
Active management instance stopped
```

For more information on the `ttGridAdmin instanceExec` command, see "Execute a command or script on grid instances (instanceExec)" in the *Oracle TimesTen In-Memory Database Reference*.

Restarting a grid

To restart a grid, you must first restart all instances before attempting to reload any database, as shown next:

1. Follow the step that matches the configuration of your grid:

- If your grid has a single management instance configuration, start the management instance.

```
% ttGridAdmin mgmtActiveStart
This management instance is now the active
```

- If your grid has an active standby configuration, follow the instructions described in "Bring back both management instances" on page 11-42 to determine the best candidate for the active role and restart both the active and standby management instances.

2. Start all data instances.

```
% ttGridAdmin instanceExec -type data ttDaemonAdmin -start
Overall return code: 0
Commands executed on:
  host3.instance1 rc 0
  host4.instance1 rc 0
  host5.instance1 rc 0
  host6.instance1 rc 0
  host7.instance1 rc 0
  host8.instance1 rc 0
Return code from host3.instance1: 0
Output from host3.instance1:
TimesTen Daemon (PID: 19072, port: 6624) startup OK.
Return code from host4.instance1: 0
Output from host4.instance1:
TimesTen Daemon (PID: 19144, port: 6624) startup OK.
Return code from host5.instance1: 0
Output from host5.instance1:
TimesTen Daemon (PID: 19210, port: 6624) startup OK.
Return code from host6.instance1: 0
Output from host6.instance1:
TimesTen Daemon (PID: 19247, port: 6624) startup OK.
Return code from host7.instance1: 0
Output from host7.instance1:
TimesTen Daemon (PID: 19284, port: 6624) startup OK.
Return code from host8.instance1: 0
Output from host8.instance1:
TimesTen Daemon (PID: 19315, port: 6624) startup OK.
```

3. Reload all databases as shown in "Reloading a database into memory" on page 5-30.

For more information on the `ttGridAdmin mgmtActiveStart` and `ttGridAdmin mgmtStandbyStart` commands, see "Management instance operations" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin instanceExec` command, see "Execute a command or script on grid instances (instanceExec)" in the *Oracle TimesTen In-Memory Database Reference*.

Upgrading a grid

This section discusses how to upgrade your grid to either a patch-compatible release or a not patch-compatible release.

Note: To check if your version of TimesTen is patch compatible with the target TimesTen release for upgrade, see the README.html file in the target TimesTen distribution.

- Upgrade a grid to a patch-compatible release
- Upgrade a grid to a not patch-compatible release

Upgrade a grid to a patch-compatible release

Upgrading a grid to a patch-compatible release consists of ensuring that every instance uses for its operations the installation files provided by a newer patchset (or patch) release of TimesTen, for example, upgrading the installation from a 18.1.*x* to a 18.1.*y* release.

To upgrade a grid to a newer patch-compatible release, perform these tasks:

1. Create an installation from a new release on every host
2. Upgrade management instances
3. Upgrade data instances
4. Optional: Delete the installation of the previous release on every host

Create an installation from a new release on every host

You may use the ttGridAdmin installationList command to determine the hosts that need to be upgraded and the location of the current installations, as shown in Example 8–3.

Example 8–3 List of hosts and installations

The example uses the ttGridAdmin installationList command to display the hosts and their associated installations of the grid1 grid.

```
% ttGridAdmin installationList
Host  Install       Location                     Comment
----- ------------- ---------------------------- -------
host1 installation1 /grid/tt18.1.4.1.0
host2 installation1 /grid/tt18.1.4.1.0
host3 installation1 /grid/tt18.1.4.1.0
host4 installation1 /grid/tt18.1.4.1.0
host5 installation1 /grid/tt18.1.4.1.0
host6 installation1 /grid/tt18.1.4.1.0
host7 installation1 /grid/tt18.1.4.1.0
host8 installation1 /grid/tt18.1.4.1.0
```

Create an installation from the new TimesTen release on every host defined in the model.

Note: If the default name for installations, installation1, is already in use, you need to provide a name for the new installation. The example uses installation2 as the name for the new installation on every host of the grid.

```
% ttGridAdmin installationCreate host1.installation2 -location
 /grid -source host1:/mydir/timesten181420.server.linux8664.zip
```

```
Installation installation2 on Host host1 created in Model

% ttGridAdmin installationCreate host2.installation2 -location
 /grid -source host1:/mydir/timesten181420.server.linux8664.zip
Installation installation2 on Host host2 created in Model

% ttGridAdmin installationCreate host3.installation2 -location
 /grid -source host1:/mydir/timesten181420.server.linux8664.zip
Installation installation2 on Host host3 created in Model

% ttGridAdmin installationCreate host4.installation2 -location
 /grid -source host1:/mydir/timesten181420.server.linux8664.zip
Installation installation2 on Host host4 created in Model

% ttGridAdmin installationCreate host5.installation2 -location
 /grid -source host1:/mydir/timesten181420.server.linux8664.zip
Installation installation2 on Host host5 created in Model

% ttGridAdmin installationCreate host6.installation2 -location
 /grid -source host1:/mydir/timesten181420.server.linux8664.zip
Installation installation2 on Host host6 created in Model

% ttGridAdmin installationCreate host7.installation2 -location
 /grid -source host1:/mydir/timesten181420.server.linux8664.zip
Installation installation2 on Host host7 created in Model

% ttGridAdmin installationCreate host8.installation2 -location
 /grid -source host1:/mydir/timesten181420.server.linux8664.zip
Installation installation2 on Host host8 created in Model
```

Apply the changes made to the latest version of the model. TimesTen copies the installation files to the location specified for each host.

```
% ttGridAdmin modelApply
```

For more information on the `ttGridAdmin installationList` or `ttGridAdmin installationCreate` command, see "List installations (installationList)" or "Create an installation (installationCreate)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin modelApply` command, see "Applying the changes made to the model" on page 4-18 in this document and "Model operations" in the *Oracle TimesTen In-Memory Database Reference*.

Upgrade management instances

How you upgrade your management instances depends on whether you have one or two management instances configured in your grid. Follow the procedure that better applies to your configuration:

- Upgrading management instances in an active standby configuration
- Upgrading a single management instance

Upgrading management instances in an active standby configuration When you have an active standby configuration for your management instances, you can upgrade each management instance separately without any interruption of service by ensuring that an active management instance is always up.

1. Stop the standby management instance.

```
% ttGridAdmin mgmtStandbyStop
Standby management instance host2.instance1 stopped
```

2. Modify the standby management instance to use the new installation.

```
% ttGridAdmin instanceModify host2.instance1 -installation installation2
Instance instance1 on Host host2 modified in Model
```

3. Apply the changes made to the latest version of the model.

```
% ttGridAdmin modelApply
```

4. Start the standby management instance by running the `ttGridAdmin mgmtStandbyStart` command on the standby management instance.

```
% ttGridAdmin mgmtStandbyStart
Standby management instance started
```

5. Verify that the standby management instance is operational and synchronized with the active management instance with the `ttGridAdmin mgmtStatus` command.

```
% ttGridAdmin mgmtStatus
Host Instance Reachable RepRole(Self) Role(Active) Role(Self) Seq RepAgent
                                                                     RepActive
---------------------------------------------------------------------
host1 instance1 Yes     Active        Unknown      Active     445 Up  Yes
host2 instance1 Yes     Standby       Unknown      Standby    445 Up  No
```

Note: Ensure that the sequence number matches in both instances to ensure that both instances are communicating properly and synchronized. If the sequence number does not match, run the `ttGridAdmin mgmtExamine` command for instructions on how to proceed. See "Examine management instances (mgmtExamine)" in the *Oracle TimesTen In-Memory Database Reference* for more information on the `ttGridAdmin mgmtExamine` command.

6. On the standby management instance, switch the active and standby management instances.

```
% ttGridAdmin mgmtActiveSwitch
This is now the active management instance
```

   TimesTen Scaleout stops the active management instance and promotes the standby management instance to active.

7. On the new active management instance, modify the installation of the old active management instance.

```
% ttGridAdmin instanceModify host1.instance1 -installation installation2
Instance instance1 on Host host1 modified in Model
```

8. Apply the changes made to the latest version of the model.

```
% ttGridAdmin modelApply
```

9. On the old active management instance, start the instance as a standby management instance.

```
% ttGridAdmin mgmtStandbyStart
Standby management instance started
```

10. Verify that the standby management instance is operational and synchronized with the active management instance.

```
% ttGridAdmin mgmtStatus
Host Instance Reachable RepRole(Self) Role(Active) Role(Self) Seq RepAgent
                                                                   RepActive
---------------------------------------------------------------------------
host1 instance1 Yes     Standby       Unknown      Standby    451 Up No
host2 instance1 Yes     Active        Unknown      Active     451 Up Yes
```

> **Note:** Ensure that the sequence number matches in both instances to ensure that both instances are communicating properly and synchronized. If the sequence number does not match, run the `ttGridAdmin mgmtExamine` command for instructions on how to proceed. See "Examine management instances (mgmtExamine)" in the *Oracle TimesTen In-Memory Database Reference* for more information on the `ttGridAdmin mgmtExamine` command.

For more information on the `ttGridAdmin mgmtStandbyStop`, `ttGridAdmin mgmtStandbyStart`, and `ttGridAdmin mgmtStatus` commands, see "Management instance operations" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin instanceModify` command, see "Modify an instance (instanceModify)" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin mgmtActiveSwitch` command, see "Starting, stopping and switching management instances" on page 11-35 in this document and "Switch the active management instance (mgmtActiveSwitch)" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin modelApply` command, see "Applying the changes made to the model" on page 4-18 in this document and "Model operations" in the *Oracle TimesTen In-Memory Database Reference*.

Upgrading a single management instance In a single management instance configuration, you need to restart the active management instance for the new installation to take effect, as shown next:

1. Modify the active management instance to use the new installation.

```
% ttGridAdmin instanceModify host1.instance1 -installation installation2
Instance instance1 on Host host1 modified in Model
```

2. Apply the changes made to the latest version of the model.

```
% ttGridAdmin modelApply
```

3. Stop the active management instance.

```
% ttGridAdmin mgmtActiveStop
Active management instance stopped
```

4. Restart the active management instance.

```
% ttGridAdmin mgmtActiveStart
This management instance is now the active
```

For more information on the `ttGridAdmin instanceModify` command, see "Modify an instance (instanceModify)" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin modelApply` command, see "Applying the changes made to the model" on page 4-18 in this document and "Model operations" in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin mgmtActiveStop` and `ttGridAdmin mgmtActiveStart`, see "Management instance operations" in the *Oracle TimesTen In-Memory Database Reference*.

Upgrade data instances

Before you can restart a data instance so that the new installation takes effect, you need to unload all databases:

1. Unload all databases as shown in "Unloading a database from memory" on page 5-27.
2. Stop all data instances.

```
% ttGridAdmin instanceExec -type data ttDaemonAdmin -stop
Overall return code: 0
Commands executed on:
  host3.instance1 rc 0
  host4.instance1 rc 0
  host5.instance1 rc 0
  host6.instance1 rc 0
  host7.instance1 rc 0
  host8.instance1 rc 0
Return code from host3.instance1: 0
Output from host3.instance1:
TimesTen Daemon (PID: 4498, port: 6624) stopped.
Return code from host4.instance1: 0
Output from host4.instance1:
TimesTen Daemon (PID: 4536, port: 6624) stopped.
Return code from host5.instance1: 0
Output from host5.instance1:
TimesTen Daemon (PID: 4492, port: 6624) stopped.
Return code from host6.instance1: 0
Output from host6.instance1:
TimesTen Daemon (PID: 4510, port: 6624) stopped.
Return code from host7.instance1: 0
Output from host7.instance1:
TimesTen Daemon (PID: 4539, port: 6624) stopped.
Return code from host8.instance1: 0
Output from host8.instance1:
TimesTen Daemon (PID: 4533, port: 6624) stopped.
```

3. Modify all the data instances to use the new installations.

> **Note:** You can perform this step before unloading the databases or stopping the data instances if you want to reduce the down time the databases incur during the upgrading operation.

```
% ttGridAdmin instanceModify host3.instance1 -installation installation2
Instance instance1 on Host host3 modified in Model

% ttGridAdmin instanceModify host4.instance1 -installation installation2
Instance instance1 on Host host3 modified in Model

% ttGridAdmin instanceModify host5.instance1 -installation installation2
```

```
Instance instance1 on Host host3 modified in Model

% ttGridAdmin instanceModify host6.instance1 -installation installation2
Instance instance1 on Host host3 modified in Model

% ttGridAdmin instanceModify host7.instance1 -installation installation2
Instance instance1 on Host host3 modified in Model

% ttGridAdmin instanceModify host8.instance1 -installation installation2
Instance instance1 on Host host3 modified in Model
```

4. Apply the changes made to the latest version of the model.

```
% ttGridAdmin modelApply
```

5. Restart all data instances.

```
% ttGridAdmin instanceExec -type data ttDaemonAdmin -start
Overall return code: 0
Commands executed on:
  host3.instance1 rc 0
  host4.instance1 rc 0
  host5.instance1 rc 0
  host6.instance1 rc 0
  host7.instance1 rc 0
  host8.instance1 rc 0
Return code from host3.instance1: 0
Output from host3.instance1:
TimesTen Daemon (PID: 19072, port: 6624) startup OK.
Return code from host4.instance1: 0
Output from host4.instance1:
TimesTen Daemon (PID: 19144, port: 6624) startup OK.
Return code from host5.instance1: 0
Output from host5.instance1:
TimesTen Daemon (PID: 19210, port: 6624) startup OK.
Return code from host6.instance1: 0
Output from host6.instance1:
TimesTen Daemon (PID: 19247, port: 6624) startup OK.
Return code from host7.instance1: 0
Output from host7.instance1:
TimesTen Daemon (PID: 19284, port: 6624) startup OK.
Return code from host8.instance1: 0
Output from host8.instance1:
TimesTen Daemon (PID: 19315, port: 6624) startup OK.
```

6. Restart all databases as shown in "Reloading a database into memory" on page 5-30.

For more information on the `ttGridAdmin instanceExec` or `ttGridAdmin instanceModify` command, see "Execute a command or script on grid instances (instanceExec)" or "Modify an instance (instanceModify)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin modelApply` command, see "Applying the changes made to the model" on page 4-18 in this document and "Model operations" in the *Oracle TimesTen In-Memory Database Reference*.

Optional: Delete the installation of the previous release on every host

To avoid assigning the wrong installation to new instances, it is recommended that you delete the installation model objects of the previous release for each host in the model, as shown in Example 8–4.

Example 8–4 Deleting the installations of a previous release

The example uses the `ttGridAdmin installationDelete` command to delete the installation model objects that are no longer in use from the model of the `grid1` grid.

```
% ttGridAdmin installationDelete host1.installation1
Installation installation1 on Host host1 deleted from Model

% ttGridAdmin installationDelete host2.installation1
Installation installation1 on Host host2 deleted from Model

% ttGridAdmin installationDelete host3.installation1
Installation installation1 on Host host3 deleted from Model

% ttGridAdmin installationDelete host4.installation1
Installation installation1 on Host host4 deleted from Model

% ttGridAdmin installationDelete host5.installation1
Installation installation1 on Host host5 deleted from Model

% ttGridAdmin installationDelete host6.installation1
Installation installation1 on Host host6 deleted from Model

% ttGridAdmin installationDelete host7.installation1
Installation installation1 on Host host7 deleted from Model

% ttGridAdmin installationDelete host8.installation1
Installation installation1 on Host host8 deleted from Model
```

Then, the example uses the `ttGridAdmin modelApply` to apply the changes made to the latest version of the model.

```
% ttGridAdmin modelApply
```

Furthermore, if the installation files associated with the installation model objects you just deleted are not in use by any other installation object in this or any other grid, then delete the files on every host. Ensure that you change the permissions of the directory so that you can delete all files, as shown in Example 8–5.

Example 8–5 Deleting the installation files of a previous release

```
% cd /grid
% chmod -R 750 tt18.1.4.1.0/
% rm -rf tt18.1.4.1.0/
```

For more information on the `ttGridAdmin installationDelete` command, see "Delete an installation (installationDelete)" in the *Oracle TimesTen In-Memory Database Reference*.

Upgrade a grid to a not patch-compatible release

Generally, all patchset (and patch) releases of the same major release of TimesTen are patch compatible. For exceptions or major release upgrades, you need to migrate the data from your current databases to databases in a different grid, one based on the target upgrade release.

To upgrade a grid to a newer release that is not patch compatible with your current release, perform these tasks:

1. Install the newer TimesTen distribution, as shown in "Installing TimesTen Scaleout" on page 2-10.
2. Using your new TimesTen installation, set up a new grid and databases to import the data of your current databases, as shown in "Configure your grid" on page 4-1 and "Creating a database" on page 5-1.

> **Note:** The new grid does not need to match the topography (K-safety value and number of replica sets) of your current grid.

3. Export the data from your current databases and import it into the databases of your new grid, as shown in "Exporting and importing a database" on page 10-15.
4. Optional: Destroy your previous grid, as shown in "Destroying a grid" on page 8-26.

Destroying a grid

Gracefully destroying a grid consists in destroying all databases and deleting every object of the model, as shown next:

1. Unload all databases as shown in "Unloading a database from memory" on page 5-27.
2. Destroy all databases as shown in "Destroying a database" on page 5-33.
3. Delete all hosts, installations, and instances from the latest version of model, except for the active management instance and its associated host and installation.

```
% ttGridAdmin hostDelete host2 -cascade
Instance instance1 on Host host2 deleted from Model
Installation installation1 on Host host2 deleted from Model
Host host2 deleted from Model

% ttGridAdmin hostDelete host3 -cascade
Instance instance1 on Host host3 deleted from Model
Installation installation1 on Host host3 deleted from Model
Host host3 deleted from Model

% ttGridAdmin hostDelete host4 -cascade
Instance instance1 on Host host4 deleted from Model
Installation installation1 on Host host4 deleted from Model
Host host4 deleted from Model

% ttGridAdmin hostDelete host5 -cascade
Instance instance1 on Host host5 deleted from Model
Installation installation1 on Host host5 deleted from Model
Host host5 deleted from Model

% ttGridAdmin hostDelete host6 -cascade
Instance instance1 on Host host6 deleted from Model
Installation installation1 on Host host6 deleted from Model
Host host6 deleted from Model

% ttGridAdmin hostDelete host7 -cascade
Instance instance1 on Host host7 deleted from Model
Installation installation1 on Host host7 deleted from Model
Host host7 deleted from Model
```

```
% ttGridAdmin hostDelete host8 -cascade
Instance instance1 on Host host8 deleted from Model
Installation installation1 on Host host8 deleted from Model
Host host8 deleted from Model
```

4. Apply the changes made to the latest version of the model.

```
% ttGridAdmin modelApply
...
Unconfiguring standby management instance............................OK
Identifying any deleted objects......................................OK
Stopping deleted instances...........................................OK
Deleting instances...................................................OK
Deleting installations from model....................................OK
Deleting any hosts that are no longer in use.........................OK
...
ttGridAdmin modelApply complete
```

5. Stop the active management instance.

```
% ttGridAdmin mgmtActiveStop
Active management instance stopped
```

6. Destroy the active management instance.

```
% /grid/tt18.1.4.1.0/bin/ttInstanceDestroy

** WARNING **

   The uninstallation has been executed by a non-root user.
   If the TimesTen daemon startup scripts were installed,
   you must run $TIMESTEN_HOME/bin/setuproot -uninstall
   to remove them. If you proceed with this uninstallation, you
   will have to remove the startup scripts manually.

** WARNING **

   All the files in the directory :

   /grid/instance1

   will be removed, including any files that you or other users
   may have created.

   Are you sure you want to completely remove this instance? [ yes ]

NOTE: /grid/instance1/info contains information related to the data
      storesthat have been created with this release. If you remove
      /grid/instance1/info you will no longer be able to access your
      data stores, nor would you be able to restore nor migrate your data.

   Would you also like to remove all files in
 /grid/instance1/info? [ no ] yes

NOTE: /grid/instance1/conf contains information related to the
      instance configuration.

   Would you also like to remove all files in
 /grid/instance1/conf? [ no ] yes
/grid/instance1 Removed
The TimesTen instance instance1 has been destroyed.
```

7. Delete the installation files on each system with a TimesTen installation. Ensure that you change the permissions of the directory so that you can delete all files.

```
% cd /grid
% chmod -R 750 tt18.1.4.1.0/
% rm -rf tt18.1.4.1.0/
```

For more information on the `ttGridAdmin hostDelete`, `ttGridAdmin mgmtActiveStop`, or `ttInstanceDestroy` command, see "Delete a host (hostDelete)", "Stop the active management instance (mgmtActiveStop)", or "ttInstanceDestroy", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

For more information on the `ttGridAdmin modelApply` command, see "Applying the changes made to the model" on page 4-18 in this document and "Model operations" in the *Oracle TimesTen In-Memory Database Reference*.

9

Monitoring TimesTen Scaleout

The following sections discusses how to monitor a grid and database in TimesTen Scaleout.

There are several ways to monitor a grid and database:

- The `ttStats` utility - This utility enables you to monitor database metrics (statistics, states, and other information) or take and compare snapshots of metrics. See "Using the ttStats utility" on page 9-1 for more information.
- SQL Developer - SQL Developer enables you to create, manage, and explore a grid and its components. Additionally, you can also browse, create, edit and drop particular database objects; run SQL statements and scripts; manipulate and export data; view and create reports; and view database metrics. For more information, see "Using SQL Developer to work with TimesTen Scaleout" on page 9-7.

Using the ttStats utility

The `ttStats` utility enables you to monitor database metrics (statistics, states, and other information), automatically captures system snapshots, and take and compare snapshots of metrics. The `ttStats` utility can perform the following functions.

- Monitor and display database performance metrics in real-time, calculating rates of change during each preceding interval.

Monitoring and analyzing reports of the database helps you determine the overall performance of your grid. By knowing the overall performance of your database, you can take preventive measures that ensure that your database is running with optimal conditions.

There are several differences in how `ttStats` works in TimesTen Classic and TimesTen Scaleout. For more information, see "ttStats" in the *Oracle TimesTen In-Memory Database Reference*. For details on the `TT_STATS` PL/SQL package, see "TT_STATS" in the *Oracle TimesTen In-Memory Database PL/SQL Packages Reference*.

The following sections describe how to use the `ttStats` utility:

- Configure the ttStats utility
- Monitor a database with the ttStats utility
- Create a snapshot with the ttStats utility
- Create a report between two snapshots with the ttStats utility

View the configuration of the ttStats utility

The `ttStatsConfigGet` built-in procedure enables you to view the configuration settings of the `ttStats` utility. This built-in shows the values of the `pollSec`, `retentionDays`, and `retainMinutes` parameters which set the collection settings of the `ttStats` utility.

This following example shows the collection settings of the `ttStats` utility:

```
Command> call ttStatsConfigGet();

< POLLSEC, 30 >
< RETENTIONDAYS, 62 >
< RETAINMINUTES, 120 >
3 rows found.
```

The `pollSec`, `retentionDays`, and `retainMinutes` parameters, which are only supported in TimesTen Scaleout, enable you to set the polling interval, purging time for aggregated data, and purging time for raw data for TimesTen Scaleout statistics, respectively. The polling interval parameter, `pollsec`, determines the interval, in seconds, at which the `ttStats` daemon collects metrics of the database.

The value of the polling interval does not affect the performance of the database. However, a polling interval of 10 seconds tends to use six times less space than a polling interval of 60 seconds. Most metrics get aggregated and use around 6 MB (even up to 10 years worth of metrics) of `PermSize` space on each element. However, some metrics such as log holds, top SQL commands, and checkpoint history cannot be aggregated. You can use the `ttStats -snapshotInfo` utility to determine how much `PermSize` is being used for your metrics.

For more information of the `ttStatsConfigGet` built-in procedure and the `ttStats` utility, see "ttStatsConfigGet" and "ttStats", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Configure the ttStats utility

The `ttStatsConfig` built-in procedure controls the configuration settings of the `ttStats` utility and when `ttStats` automatically takes system snapshots. Call the `ttStatsConfig` built-in procedure to modify statistics collection parameters that affect the `ttStats` utility. For more information on the parameters of the `ttStatsConfig` built-in procedure and the `SYS.V$STATS_CONFIG` system view, see "ttStatsConfig" in the *Oracle TimesTen In-Memory Database Reference* and "SYS.V$STATS_CONFIG" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*, respectively.

The polling interval parameter, `pollsec`, determines the interval, in seconds, at which the `ttStats` daemon collects metrics of the database.

The following example returns the current value of the polling interval for TimesTen Scaleout statistics:

```
SQL> SELECT VALUE FROM SYS.V$STATS_CONFIG WHERE PARAM='POLLSEC';
< 30 >
1 row found.
```

The following example sets the polling interval of TimesTen Scaleout statistics to 45 seconds:

```
Command> call ttStatsConfig('pollsec', 45);
< POLLSEC, 45 >
1 row found.
```

The retention time interval, `retentionDays`, determines the interval, in days, at which the `ttStats` daemon drops metrics of the database. For example, if the retention time interval is 62 days, the `ttStats` daemon drops the 1st day's snapshot on the 63rd day. Ensure that you have sufficient `PermSize` to support the desired retention time interval.

The following example returns the current value of the retention time interval for TimesTen Scaleout statistics:

```
SQL> SELECT VALUE FROM SYS.V$STATS_CONFIG WHERE PARAM='RETENTIONDAYS';
< 62 >
1 row found.
```

The following example sets the retention time interval for TimesTen Scaleout statistics to 30 days:

```
Command> call ttStatsConfig('retentionDays', 30);
< RETENTIONDAYS, 30 >
1 row found.
```

The purging time interval, `retainMinutes`, determines the interval, in minutes, in which the `ttStats` daemon purges raw metrics of the database. For example, if the retention time interval is 120 minutes, the `ttStats` daemon purges the raw metrics every 120 minutes.

The following example returns the current value of the purging time interval for TimesTen Scaleout statistics:

```
SELECT VALUE FROM SYS.V$STATS_CONFIG WHERE PARAM='RETAINMINUTES';
< 120 >
1 row found.
```

The following example sets the purging time interval for TimesTen Scaleout statistics to 60 minutes:

```
Command> call ttStatsConfig('retainMinutes', 60);
< RETAINMINUTES, 60 >
1 row found.
```

Monitor a database with the ttStats utility

Use the `ttStats -monitor` utility to monitor your database workload on a local instance in real-time. You can specify the `-duration` or `-iterations` option to set the length of time that the `ttStats` utility monitors the TimesTen Scaleout. Monitoring continues until the limit of the `-duration` or `-iterations` options is reached or when you use `Ctrl-C`. You can also specify an interval time, `-interval`, which sets the time interval between sets of metrics that are displayed, in seconds. These options can be specified together. You can specify the following options:

- `-duration`: This option sets the duration of how long the `ttStats` utility runs, in seconds. After this duration, the utility exits.

  The following example monitors a database for 60 seconds:

```
% ttStats -monitor -duration 60 database1
Connected to TimesTen Version 18.01.0002.0001 Oracle TimesTen IMDB version
18.1.2.1.0.

Waiting for 10 seconds for the next snapshot
Description                          Current  Rate/Sec  Notes
date.2017-Feb-22 11:33:41         1456169621         1  sample #, not rate
cmdcache.id:278352904.preps           142072         1  COMMIT
cmdcache.id:283596680.execs           135242         1  SELECT COUNT(*) FROM
SYS.TTSTATS
cmdcache.id:283613080.execs           340200         3  SELECT COUNT(*) FROM
SYS.TTSTATS
cmdcache.id:283619720.execs           135242         1  INSERT INTO
SYS.TTSTATS_SQL_COMM
connections.count                         15
db.joins.nested_loop                   22874         1
db.table.full_scans                   136618         2
lock.locks_granted.immediate        24138575       291
log.buffer.bytes_inserted         4887634664     52988
log.buffer.insertions               41123321       447
log.file.writes                      247855        2
log.forces                           183285        1
log.log_bytes_per_transaction             0
loghold.bookmark.log_force_lsn  88/46899200
loghold.bookmark.log_write_lsn  88/46899464
loghold.checkpoint_hold_lsn     88/41543680           database1.ds0
loghold.checkpoint_hold_lsn     88/33990656           database1.ds1
plsql.GetHitRatio                     0.714     0.000
plsql.GetHits                       380.000     0.200
plsql.Gets                          532.000     0.200
plsql.PinHitRatio                     0.989     0.000
plsql.PinHits                     34556.000     0.500
plsql.Pins                        34933.000     0.500
stmt.executes.count                 1103839       12
stmt.executes.inserts                280246        2
stmt.executes.selects                777408        9
stmt.prepares.count                  173038        1
txn.commits.count                    233082        2
txn.commits.durable                  182275        1
...
```

- `-iterations`: This option sets the number of iterations that the `ttStats` utility performs when gathering and displaying metrics. After these iterations, the utility exits.

  The following example sets the number of iterations to 3:

```
% ttStats -monitor -iterations 3 database1
Connected to TimesTen Version 18.01.0002.0001 Oracle TimesTen IMDB version
18.1.2.1.0.

Waiting for 10 seconds for the next snapshot
Description                           Current  Rate/Sec  Notes
date.2017-Feb-22 11:54:34          1456170874         1  sample #, not rate
connections.count                          15
lock.locks_granted.immediate         24195281         1
log.log_bytes_per_transaction               0
loghold.bookmark.log_force_lsn    88/61253632
loghold.bookmark.log_write_lsn    88/61253896
loghold.checkpoint_hold_lsn       88/55470080           database1.ds0
loghold.checkpoint_hold_lsn       88/48414720           database1.ds1
```

```
plsql.GetHitRatio                        0.730     0.000
plsql.GetHits                          410.000     0.200
plsql.Gets                             562.000     0.200
plsql.PinHitRatio                        0.989     0.000
plsql.PinHits                        34667.000     0.200
plsql.Pins                           35044.000     0.200
stmt.executes.count                    1106494         1
stmt.executes.selects                   779348         1
...
```

- `-interval`: This option sets the time interval between sets of metrics that are displayed, in seconds.

  The following example sets the interval time to 30 seconds:

```
% ttStats -interval 30 -monitor database1
Connected to TimesTen Version 18.01.0002.0001 Oracle TimesTen IMDB version
18.1.2.1.0.

Waiting for 30 seconds for the next snapshot
Description                         Current  Rate/Sec  Notes
date.2017-Feb-19 15:18:38        1455923918         1  sample #, not rate
connections.count                        15
lock.locks_granted.immediate       12536701         1
log.log_bytes_per_transaction             0
loghold.bookmark.log_force_lsn   45/13309952
loghold.bookmark.log_write_lsn   45/13310216
loghold.checkpoint_hold_lsn       45/4683776           database1.ds0
loghold.checkpoint_hold_lsn      45/11804672           database1.ds1
plsql.GetHitRatio                     0.700     0.000
plsql.GetHits                       355.000     0.067
plsql.Gets                          507.000     0.067
plsql.PinHitRatio                     0.980     0.000
plsql.PinHits                     18201.000     0.067
plsql.Pins                        18578.000     0.067
...
```

Create a snapshot with the ttStats utility

Use the `ttStats -snapshot` utility to associate a snapshot ID with the latest system generated snapshot of your database. Snapshots are used to create reports that show you database metrics. When a system generated snapshot gets automatically purged, the associated user snapshots will also be purged.

The following example uses the `ttStats -snapshot` utility to create a snapshot. The `-description` command is required when you use the `-snapshot` command. The `-description` command lets you provide any description or notes for the snapshot, for example to distinguish it from other snapshots.

```
% ttStats -snapshot -description 1 database1
Connected to TimesTen Version 18.01.0002.0001 Oracle TimesTen IMDB version
18.1.2.1.0.
Snapshot ID was 88412
```

You can reference the snapshot that was created from the example with a snapshot ID of `1`.

Create a report between two snapshots with the ttStats utility

Use the `ttStats -report` utility to create a report between two snapshots of your TimesTen Scaleout database. `ttStats` reports show the change in statistics between two snapshots of your database. The `-outputFile` option enables you to specify a file path and name where the report is to be written. Use one of the following set of options to define the start and end points of the report:

- The `-snap1` and `-snap2` options to specify snapshot IDs. The report period must span at least four existing snapshot ID values. Therefore, you must have at least three snapshots between `-snap1` and `-snap2`.
- The `-timestamp1` and `-timestamp2` options to specify timestamps.

You can use the `ttStats -snapshotInfo` command to view available snapshots for your database.

The following example uses the `ttStats -snapshotInfo` utility to return the IDs and timestamps of available snapshots. This command also returns information about aggregated snapshots as well as the values of the `ttStatsConfig` built-in procedure.

```
% ttStats -snapshotInfo database1
Connected to TimesTen Version 18.01.0002.0001 Oracle TimesTen IMDB version
18.1.2.1.0.
There are 2 user snapshots:
Snapshot ID  User comment     When snapshot occurred
============ ================ ======================
   88412   1                   2018-02-09 13:28:50
   88412   2                   2018-02-10 11:13:55
   88412   3                   2018-02-10 18:39:50
   88412   4                   2018-02-11 08:10:12
   88412   5                   2018-02-12 17:23:46
There are 151 AGGREGATED snapshots:
 Oldest snapshot         2880, 2018-01-04 15:37:29
 Newest snapshot        88412, 2018-02-03 10:00:26
There are 240 NON AGGREGATED snapshots:
  Oldest snapshot        88173, 2018-02-03 08:00:42
  Newest snapshot        88412, 2018-02-03 10:00:26

There are about 16.3 MB of metrics stored in ttStats SYS tables

  The PollSec was 30
  The RetentionDays was 62
  The RetainMinutes was 120
```

The following example creates a report, `snapreport.txt`, between the snapshots with ID `1` and ID `5`:

```
% ttStats -report -snap1 1 -snap2 5 -outputFile snapreport.txt database1
Connected to TimesTen Version 18.01.0002.0001 Oracle TimesTen IMDB version
18.1.2.1.0.
Report snapreport.txt was created.
```

The following example creates a report, `timereport.txt`, between two timestamps:

```
% ttStats -report -timestamp1 2018-02-22 12:50:31 -timestamp2 2018-02-23 09:15:23
      -outputFile snapreport.txt database1
Connected to TimesTen Version 18.01.0002.0001 Oracle TimesTen IMDB version
18.1.2.1.0.
Report timereport.txt was created.
```

For more information about the tables of metrics that a `ttStats` report generates and the `ttStats` utility, see "Report examples" and "ttStats", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

Using SQL Developer to work with TimesTen Scaleout

Oracle SQL Developer is a graphical user interface (GUI) tool that gives database developers a convenient way to create, manage, and explore a grid and its components. You can also browse, create, edit and drop particular database objects; run SQL statements and scripts; manipulate and export data; view and create reports; and view database metrics.

For more information, see *Oracle SQL Developer Oracle TimesTen In-Memory Database Support User's Guide*.

Monitoring the management instances

Management instances store metadata used to manage the grid. It is recommended that you use an active and a standby instance to have high availability for this metadata. If you use a single management instance and that management instance is down, the grid continues to operate but you cannot perform certain management operations for your grid.

A management instance can get full because it stores information about your grid, previous grid model versions, and logs of your grid. It is important for the management instances to have enough free space to function properly. If your management instance begins to get full, any command that you run with the `ttGridAdmin` utility outputs a warning.

You can perform these tasks to maintain the management instances:

- Monitor the free space of the management instance
- Modify retention values of previous grid models and warning threshold of the management instance
- Resize the management instance

Monitor the free space of the management instance

When you create a grid, the grid sets a used-space warning threshold for the management instance. If the size of your management instance reaches this threshold, commands that you run with the `ttGridAdmin` utility output warnings that the management instance is getting full.

This example shows the output of a `ttGridAdmin instanceCreate` command for a grid where the management instance is almost full.

```
% ttGridAdmin instanceCreate host5 -location /grid

Instance instance1 on Host host5 created in Model
Warning: the TTGRIDADMIN database is 91% full; Temp space: 57%
```

Note: When you use a `ttGridAdmin` command and you see a warning that your management instance is getting full, TimesTen Scaleout deletes old grid model versions and logs based on the retention days and retention versions parameters of your grid.

Use the gridDisplay command of the ttGridAdmin utility on your management instance to see the current used-space warning threshold for the management instance, and retention days and quantity of previous versions of the grid model that TimesTen Scaleout stores.

This example shows the output of the ttGridAdmin gridDisplay command.

```
% ttGridAdmin gridDisplay

Grid name:                  grid1
Grid GUID:                  864C0CB2-AF40-4047-A711-7A9F9F0E7D6C
Created:                    2018-12-12 12:20:32.000000
Major Release:              18.1.4
Created Release:            18.1.4.1.0
K:                          2
Admin Userid:               instanceadmin
Admin UID:                  4133
Admin Group:                admins
Admin GID:                  900
Retain Days:                30
Retain Versions:            10
Warn Threshold:             90
Perm In Use Pct:            91
Temp In Use Pct:            57
```

For more information about the ttGridAdmin gridDisplay command and the default values for the retention of previous grid models and warning threshold of the management instance, see "Display information about the grid (gridDisplay)" and "Modify grid settings (gridModify)", respectively, in the *Oracle TimesTen In-Memory Database Reference*.

See "Resize the management instance" on page 9-9 for more information on resizing the grid administration database.

Modify retention values of previous grid models and warning threshold of the management instance

In some cases, you may want to increase or decrease the retention values of previous grid models or the current used-space warning threshold for the management instance.

This example sets the current used-space warning threshold for the management instance to 80%.

```
% ttGridAdmin gridModify -warnThresh 80

Grid Definition modified.
```

This example sets the retention days value to 60 and the retention versions value to 15. These values ensure that TimesTen Scaleout only deletes previous grid models that are older than 60 days and are at least 16 grid model versions old.

Note: If you specify either the -retainDays or the -retainVersions parameter as 0, then only the other parameter is used. If you set both parameters values to 0, TimesTen Scaleout never automatically deletes previous grid model versions.

```
% ttGridAdmin gridModify -retainDays 60 -retainVersions 15
```

```
Grid Definition modified.
```

For more information about the `ttGridAdmin gridModify` command, see "Modify grid settings (gridModify)" in the *Oracle TimesTen In-Memory Database Reference*.

Resize the management instance

In some cases, you may want to resize the management instance because it is getting full.

Depending on if your grid has a single management instance or an active and a standby management instances, follow one of these procedures:

- Grid with a single management instance
- Grid with active and standby management instances

Grid with a single management instance

To resize the management instance of a grid with one management instance, ensure that you are connected to the management instance.

1. Export the database definition.

   ```
   % ttGridAdmin dbdefExport TTGRIDADMIN /tmp/ttgridadmin.dbdef
   ```

   Example 9–1 shows the contents of the exported file.

Example 9–1 Exported database definition file

```
# DbDef GUID CF85D379-E776-41C6-A271-ACB6A2033BBB Exported 2018-03-24 14:31:52
[TTGRIDADMIN]
AutoCreate=0
Connections=100
DBUUID=C12C4FAE-5732-4307-A08F-5F7FBF9BF1C0
DataStore=!!TIMESTEN_HOME!!/grid/admin/database/!!TTGRIDADMIN!!
DatabaseCharacterSet=AL32UTF8
DurableCommits=1
LockWait=120
Overwrite=0
PLSQL=1
PLSQL_TIMEOUT=0
PermSize=200
TempSize=100
```

2. With a text editor, modify the value of the `PermSize` connection attribute to a larger value.

   Example 9–2 shows the contents of the modified database definition file. In this example, the new value of the `PermSize` connection attribute is `400`.

Example 9–2 Modified database definition file

```
# DbDef GUID CF85D379-E776-41C6-A271-ACB6A2033BBB Exported 2018-03-24 14:31:52
[TTGRIDADMIN]
AutoCreate=0
Connections=100
DBUUID=C12C4FAE-5732-4307-A08F-5F7FBF9BF1C0
DataStore=!!TIMESTEN_HOME!!/grid/admin/database/!!TTGRIDADMIN!!
DatabaseCharacterSet=AL32UTF8
DurableCommits=1
```

```
LockWait=120
Overwrite=0
PLSQL=1
PLSQL_TIMEOUT=0
PermSize=400
TempSize=100
```

3. Import the contents of the modified database definition file into the TTGRIDADMIN database definition.

```
% ttGridAdmin dbdefModify /tmp/ttgridadmin.dbdef
Database Definition TTGRIDADMIN modified.
```

4. Apply the changes of the TTGRIDADMIN database definition file to the current version of the model.

```
% ttGridAdmin modelApply
...
Pushing new configuration files to each Instance......................OK
...
ttGridAdmin modelApply complete
```

5. Stop the management instance.

> **Note:** Stopping the management instance does not impact existing databases. However, you are unable to perform management operations until you start the management instance.

```
% ttGridAdmin mgmtActiveStop

Active management instance stopped
```

6. Start the management instance.

```
% ttGridAdmin mgmtActiveStart

This management instance is now the active
```

You have successfully resized your management instance.

Grid with active and standby management instances

To resize the management instances of a grid with active and standby management instances, ensure that you are connected to the active management instance.

1. Export the database definition of the grid administration database.

```
% ttGridAdmin dbdefExport TTGRIDADMIN /tmp/ttgridadmin.dbdef
```

Example 9–3 shows the contents of the exported file.

Example 9–3 Exported database definition file

```
# DbDef GUID CF85D379-E776-41C6-A271-ACB6A2033BBB Exported 2018-03-24 14:31:52
[TTGRIDADMIN]
AutoCreate=0
Connections=100
DBUUID=C12C4FAE-5732-4307-A08F-5F7FBF9BF1C0
DataStore=!!TIMESTEN_HOME!!/grid/admin/database/!!TTGRIDADMIN!!
DatabaseCharacterSet=AL32UTF8
```

```
DurableCommits=1
LockWait=120
Overwrite=0
PLSQL=1
PLSQL_TIMEOUT=0
PermSize=200
TempSize=100
```

2. With a text editor, modify the value of the `PermSize` connection attribute to a larger value.

   Example 9–4 shows the contents of the modified database definition file. In this example, the new value of the `PermSize` connection attribute is `400`.

Example 9–4 Modified database definition file

```
# DbDef GUID CF85D379-E776-41C6-A271-ACB6A2033BBB Exported 2018-03-24 14:31:52
[TTGRIDADMIN]
AutoCreate=0
Connections=100
DBUUID=C12C4FAE-5732-4307-A08F-5F7FBF9BF1C0
DataStore=!!TIMESTEN_HOME!!/grid/admin/database/!!TTGRIDADMIN!!
DatabaseCharacterSet=AL32UTF8
DurableCommits=1
LockWait=120
Overwrite=0
PLSQL=1
PLSQL_TIMEOUT=0
PermSize=400
TempSize=100
```

3. Import the contents of the modified database definition file into the `TTGRIDADMIN` database definition.

   ```
   % ttGridAdmin dbdefModify /tmp/ttgridadmin.dbdef
   Database Definition TTGRIDADMIN modified.
   ```

4. Apply the changes of the `TTGRIDADMIN` database definition file to the current version of the model.

   ```
   % ttGridAdmin modelApply
   ...
   Pushing new configuration files to each Instance......................OK
   ...
   ttGridAdmin modelApply complete
   ```

5. From the standby management instance, stop the management instance.

   Note: This procedure does not impact existing databases or affect operations of the grid.

   ```
   % ttGridAdmin mgmtStandbyStop

   Standby management instance host2.instance1 stopped
   ```

6. From the standby management instance, start the management instance.

   ```
   % ttGridAdmin mgmtStandbyStart

   Standby management instance started
   ```

7. From the standby management instance, promote the standby management instance to be the new active management instance and shut down the original active management instance.

```
% ttGridAdmin mgmtActiveSwitch

This is now the active management instance
```

8. From the original active management instance, start the new standby management instance.

```
% ttGridAdmin mgmtStandbyStart

Standby management instance started
```

You have successfully resized your management instances. Additionally, your original active management instance is now the standby management instance and the original standby management instance is now the active management instance. This does not affect operations of your grid.

Collecting grid logs

TimesTen Scaleout enables you to collect various logs from every host that is part of your grid. These logs are useful for troubleshooting errors that you may encounter while using your grid or database. You can collect these logs with the `ttGridAdmin gridLogCollect` command:

- `ttGridAdmin.log`

  Shows support messages about the grid.

- `tterrors.log`

  Shows any error or warning messages that the TimesTen daemon encountered.

- `ttmesg.log`

  Shows support messages about the TimesTen daemon.

- Configuration files

  Configuration files that are stored in the *`timesten_home`*`/conf/` directory of each instance.

> **Note:** The logs are stored in the *`timesten_home`*`/diag/` directory. This directory contains multiple `tterrors.log` and `ttmesg.log` files that are appended with numbers. The logs without the appended number are the most recent log files.

> **Note:** Before collecting logs for your grid, ensure that you have configured a repository. See "Working with repositories" on page 10-4 for more information on repositories.

This example collects the logs for a grid and stores these logs in the repository `repo1`. By default, TimesTen Scaleout names your collection of logs with the current date and time, `L`*`yyyymmddhhss`*. The prefix of the backup name, `L`, stands for logs.

Note: You can add the `-name` parameter to specify a collection name. For example, `ttGridAdmin gridLogCollect -repository repo1 -name mylogs` creates a collection of logs named `mylogs`.

```
% ttGridAdmin gridLogCollect -repository repo1

Logs copied to collection L20170331143740 in repository repo1
```

The `ttGridAdmin gridLogCollect` command creates a collection directory in the repository. The collection directory contains a directory for every host of your grid where each host directory contains logs for that specific host.

Retrieving diagnostic information

TimesTen Scaleout enables you to retrieve diagnostic information for a whole grid. This diagnostic information can be useful for the Oracle Support team to be able to diagnose any issue that might come up with your grid.

The following example retrieves diagnostic information for your whole grid by using the `ttGridAdmin` utility from the management instance. You can then provide this file to the Oracle Support team.

```
% ttGridAdmin gridDump /tmp/grid.status.txt
```

Verifying clock synchronization across all instances

It is important to ensure that the system clocks of every host in your grid are roughly synchronized. Synchronized system clocks ensure that timestamps of transactions and logs are accurate on all hosts.

This example outputs the system date and time of every host in the grid.

```
% ttGridAdmin hostExec date

Commands executed on:

  host1 rc 0
  host2 rc 0
  host3 rc 0
  host4 rc 0
  host5 rc 0
  host6 rc 0
Return code from host1: 0
Output from host1:
Fri Mar 31 18:16:51 PDT 2018
Return code from host2: 0
Output from host2:
Fri Mar 31 18:16:49 PDT 2018
Return code from host3: 0
Output from host3:
Fri Mar 31 18:16:51 PDT 2018
Return code from host4: 0
Output from host4:
Fri Mar 31 18:16:51 PDT 2018
Return code from host5: 0
Output from host5:
Fri Mar 31 18:16:50 PDT 2018
```

```
Return code from host6: 0
Output from host6:
Fri Mar 31 18:16:52 PDT 2018
```

In case that the system clock of a host is not synchronized with the other hosts, adjust the system clock on that specific host. You can use the Network Time Protocol (NTP) to ensure that the system clock of your hosts are synchronized.

10

Migrating, Backing Up and Restoring Data

The following chapter discusses how to migrate data from a TimesTen Classic database, work with repositories, and how to back up and restore data in a TimesTen Scaleout database.

- Migrating a database from TimesTen Classic to TimesTen Scaleout
- Working with repositories
- Backing up and restoring a database
- Exporting and importing a database
- Determining the size of a backup or export

Migrating a database from TimesTen Classic to TimesTen Scaleout

TimesTen Scaleout enables you to migrate a database from TimesTen Classic to TimesTen Scaleout. TimesTen Scaleout supports and includes most of the features of TimesTen Classic; it does not support any of the features of the TimesTen Cache or TimesTen Replication. See "Comparison between TimesTen Scaleout and TimesTen Classic" on page 1-24 for more information on what features are supported in TimesTen Scaleout. These procedures are for TimesTen Classic databases. You cannot migrate the following objects:

- Tables containing a LOB column.
- Tables that contain `ROWID` columns.
- Tables with in-memory columnar compression.
- Tables with aging policies.
- Cache groups.
- Replication schemes.

Prerequisites before migrating a database from TimesTen Classic to TimesTen Scaleout:

- Create a grid with management and data instances. See Chapter 4, "Setting Up a Grid" for more information.
- Create a backup of your TimesTen Classic database. See "ttBackup" and "ttRestore" in *Oracle TimesTen In-Memory Database Reference* for more information.

  See "Backing up and restoring a database" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.
- After you have created a backup of your TimesTen Classic database, consider removing LOB columns from your tables. TimesTen Scaleout cannot import a table

with LOB columns and the import process displays an error message if a table contains LOB columns. Use the `ALTER TABLE` statement with the `DROP` keyword to drop these columns. See "ALTER TABLE" in the *Oracle TimesTen In-Memory Database SQL Reference* for more information.

- In case that you have tables with `ROWID` columns consider not using `ROWID` based access in your applications. The semantics of `ROWID` columns are different in TimesTen Classic than in TimesTen Scaleout. See "Understanding ROWID in data distribution" on page 7-17 for more information.
- Understand the performance trade-off between table distribution schemes. See "Defining table distribution schemes" on page 5-13 for more information.

The procedures in this section explain how to remove the objects that cannot be migrated from your TimesTen Classic database.

To migrate a database from TimesTen Classic to a TimesTen Scaleout database, export your database schema, and migrate supported objects out of the TimesTen Classic database. Then restore these into a new TimesTen Scaleout database.

1. Disconnect all applications from your TimesTen Classic database.
2. On the TimesTen Classic instance, export the database schema with the `-list` option of the `ttSchema` utility. The `-list` option only specifies objects that are supported in TimesTen Scaleout. Ensure that you replace `database1` with the name of your database:

   ```
   % ttSchema -list tables,views,sequences,synonyms database1 >
   /tmp/database1.schema
   ```

   For more information about the `ttSchema` utility, see "ttSchema" in the *Oracle TimesTen In-Memory Database Reference*.
3. On the TimesTen Classic instance, save a copy of your database with the `ttMigrate` utility.

   ```
   % ttMigrate -c database1 /tmp/database1.data

   Saving user PUBLIC
   User successfully saved.
   ...
   Sequence successfully saved.
   ```

   For more information about the `ttMigrate` utility, see "ttMigrate" in the *Oracle TimesTen In-Memory Database Reference*.
4. Copy the database schema and the migrate object files to a file system that is accessible by one of your data instances. You can choose any data instance and you need to complete all further procedures from this same data instance unless stated otherwise.
5. On your selected data instance, use a text editor to edit the database schema file to remove SQL statements and clauses that are not supported in TimesTen Scaleout and add distribution scheme clauses for your tables. This is the database schema file that you created in step 3.

   Remove the following SQL statements:

   - `CREATE CACHE GROUP`
   - `CREATE REPLICATION`
   - `CREATE ACTIVE STANDBY PAIR`

- `CREATE INDEX` (Before removing these statements review the note below)

> **Note:** `CREATE INDEX` statements are supported in TimesTen Scaleout, but it is more efficient to create indexes once your data has been distributed. However, for child tables which you want to distribute with the `DISTRIBUTE BY REFERENCE` distribution scheme, you should not remove the `FOREIGN KEY` clause of the child table, nor the `CREATE INDEX` statement of the referenced parent table. Step 9 restores your indexes once your data has been inserted into your TimesTen Scaleout database.

Remove the following `CREATE TABLE` clauses:

- `COMPRESS BY`
- `FOREIGN KEY` (Before removing these statements review the note above)
- `AGING`

Add `CREATE USER` statements to create the schema owners referenced by the objects in `database1.schema`. For example, `hr.employees`, would require a `CREATE USER hr IDENTIFIED BY` *`password`* statement. You also may need to add privileges to these users if you want to log in as the users.

Add distribution scheme clauses for all of your table definitions. If you do not specify a distribution scheme for a `CREATE TABLE` statement, TimesTen Scaleout distributes the data of that table with the `DISTRIBUTE BY HASH` distribution scheme.

> **Note:** When you use the `DISTRIBUTE BY REFERENCE` distribution scheme, ensure that you declare the child key columns of a foreign key constraint as `NOT NULL`.

Before adding distribution schemes to your table definitions, ensure that you understand the performance trade-off between the distribution schemes. For more information on distribution schemes in a TimesTen Scaleout database, see "Defining table distribution schemes" on page 5-13 for more information.

6. From a TimesTen Scaleout management instance, create a TimesTen Scaleout database. See "Creating a database" on page 5-1 for more information.

7. On your selected data instance, log in as the instance administrator to create the database schema from the database schema file. Ensure that you replace `new_database1` with the name of your new TimesTen Scaleout database:

```
% ttIsql -connStr "DSN=new_database1" -f /tmp/database1.schema

Copyright (c) 1996, 2020, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=new_database1";
Connection successful:
...
exit;
Disconnecting...
Done.
```

Note: It can be useful to redirect the output of the ttIsql command to an output file. You can then review this output to ensure that the command ran successfully. To redirect output to a file, add `> myoutput.txt` after the `ttIsql -connStr "DSN=new_database1" -f /tmp/database1.schema` command.

---

8. On your selected data instance, use the following `ttMigrate` command to restore rows for all user tables:

```
% ttMigrate -r -gridRestoreRows new_database1 /tmp/database1.data

Restoring table HR.EMPLOYEES
...
 10/10 rows restored.
Table successfully restored.
```

9. On your selected data instance, use the following `ttMigrate` command to restore indexes and foreign keys:

```
% ttMigrate -r -gridRestoreFinale new_database1 /tmp/database1.data

Restoring table HR.EMPLOYEES
...
 10/10 rows restored.
Table successfully restored.
```

Note: If you did not remove `FOREIGN KEY` clauses in step 5 because you are using a `DISTRIBUTE BY REFERENCE` distribution scheme, you may see error messages that TimesTen Scaleout is unable to create some foreign keys. If you already created these foreign keys in step 5, you can ignore these messages.

Once the database is operational on TimesTen Scaleout, create a backup of the TimesTen Scaleout database to have a valid restoration point for your database. See "Backing up and restoring a database" on page 10-7 for more information. Once you have created a backup of your database, you may remove the database schema file (in this example, `/tmp/database1.schema`) and the `ttMigrate` copy of your database (in this example, `/tmp/database1.data`).

Working with repositories

In a grid, a repository is used to store backups of databases, database exports, and collections of log files and configuration files. TimesTen Scaleout enables you to define a repository as a directory path mounted using NFS on each host or as a directory path that is not directly mounted on each host. Multiple grids can use a single repository.

A repository contains a number of collections. A collection can be a backup of a database, a database export, or a set of saved daemon logs and configuration files. Collections are essentially subdirectories that use the name of the collection and are stored inside of a repository. Each collection can contain a combination of files and sub-collections.

Ensure that you create your repository where there is enough file system space to store your database backups, database exports, and collections of log and configuration files.

You must create a repository for your grid before attempting to backup a database, export a database, or create a daemon log collection.

TimesTen Scaleout enables you to perform the following procedures with repositories:

- Create a repository
- Attach a repository
- Detach a repository
- List repositories and collections

Create a repository

Before you back up a database, export a database, or create a daemon log collection, you need to configure a repository for your grid. Depending on the value of the `-method` parameter, the `ttGridAdmin repositoryCreate` command creates a repository as a directory path mounted using NFS on each host or as a directory path that is accessible on each host with SSH or SCP.

> **Note:** For more information on valid names for repositories, see "Grid objects and object naming" in the *Oracle TimesTen In-Memory Database Reference*.

The mount (NFS) method can only be used if all instances are on the same network and all instances must use the same NFS. The SCP method can be used on any system but may be slower for larger grids.

Example 10–1 Create a repository as a directory path mounted using NFS on each host

This example creates a repository as a directory path mounted using NFS on each host of your grid. Ensure that the directory specified by the `-path` parameter exists and is accessible by the instance administrator on each element. This directory must have the same identical mount path on every element. For example, if the directory path is mounted at `/repositories` on one element, it must be mounted at `/repositories` on all elements.

```
% ttGridAdmin repositoryCreate repo1 -path /repositories -method mount
Repository repo1 created
```

Example 10–2 Create a repository as a directory path that is accessible on each host with SSH or SCP

This example creates a repository as a directory path that is not directly mounted on each host of your grid. Ensure that the path value specified by the `-path` parameter exists on the host that you specify with the `-address` parameter. The address parameter is the fully qualified domain name of the host on which the repository exists. Also, ensure that each host can use the `scp` command to access files in the path value specified by the `-path` parameter. You can use the `ttGridAdmin gridSshConfig` command to verify that your hosts can communicate through SSH with each other. For more information, see "Configure SSH (gridSshConfig)" in the *Oracle TimesTen In-Memory Database Reference*.

```
% ttGridAdmin repositoryCreate repo2 -path /repositories -method scp -address
```

```
host1.example.com
Repository repo2 created
```

For more information about the `ttGridAdmin repositoryCreate` command, see "Create a repository (repositoryCreate)" in the *Oracle TimesTen In-Memory Database Reference*.

Attach a repository

Multiple grids can use a single repository as long as each grid is associated with that repository. If you have an existing repository, you can attach it to another grid as long as each host from your grid has access to the path of the repository. Depending on the value of the `-method` parameter, you can attach a repository as a directory path mounted using NFS on each host or as a directory path that is accessible on each host with SSH or SCP. However, you can only attach a repository with the same `-method` as which was used to create it. For example, if you created a repository with `-method mount`, you can only attach it to another grid with `-method mount`.

Example 10–3 Attach a repository as a directory path mounted using NFS on each host

This example attaches a repository as a directory path mounted using NFS on each host of your grid. Ensure that the path value specified by the `-path` parameter exists and is accessible by the instance administrator on each host of your grid.

The name of the repository needs to be the same on each grid to which you attach your repository.

```
% ttGridAdmin repositoryAttach repo1 -path /repositories -method mount
Repository repo1 attached
```

Example 10–4 Attach a repository as a directory path that is accessible on each host with SSH or SCP

This example attaches a repository as a directory path that is not directly mounted on each host of your grid. Ensure that each host can use the `scp` command to access files in the path value specified by the `-path` parameter. The address parameter is the fully qualified domain name of the host on which the repository exists.

The name of the repository needs to be the same on each grid to which you attach your repository.

```
% ttGridAdmin repositoryAttach repo2 -path /repositories -method scp -address
host1.example.com
Repository repo2 attached
```

For more information about the `ttGridAdmin repositoryAttach` command, see "Attach a repository (repositoryAttach)" in the *Oracle TimesTen In-Memory Database Reference*.

Detach a repository

TimesTen Scaleout enables you to detach, but not destroy, a repository from a grid when you no longer need to use that repository with your grid.

To detach a repository from a grid, specify the name of the repository to detach from your grid:

```
% ttGridAdmin repositoryDetach repo1
Repository repo1 detached
```

Detaching a repository from a grid does not delete the directory or the contents of that repository.

For more information about the `ttGridAdmin repositoryDetach` command, see "Detach a repository (repositoryDetach)" in the *Oracle TimesTen In-Memory Database Reference*.

List repositories and collections

TimesTen Scaleout enables you to view a list of all repositories that are attached to a grid and all collections within the repository.

To view a list of all repositories that are attached to a grid:

```
% ttGridAdmin repositoryList
Repository Method Location           Address
---------- ------ ------------------- --------
repo1      mount  /repositories/repo1
```

To view a list of all collections that are part of every repository that are attached to a grid:

```
% ttGridAdmin repositoryList -contents
Repository Collection      Type          Date                     Details
---------- -------------   ------------- ------------------------ ------------------
repo1      B20170222145544 Backup        2017-02-22T14:55:48.000Z Database database1
repo1      B20170615142115 Backup        2017-06-15T14:21:20.000Z Database database1
repo2      L20170615143145 gridLogCollect 2017-06-15T14:31:48.000Z
repo2      L20170616102242 gridLogCollect 2017-06-16T10:22:50.000Z
```

Note: You can add the name of a repository to only view the collections that are part of a specific repository. For example, `ttGridAdmin repositoryList repo1 -contents` shows all collections of the `repo1` repository.

For more information about the `ttGridAdmin repositoryList` command, see "List repositories (repositoryList)" in the *Oracle TimesTen In-Memory Database Reference*.

Backing up and restoring a database

The TimesTen Scaleout backup and restore functionality is essential in order to protect your data. It is recommended to perform regular backups in order to minimize the risks of potential data loss. When you perform a backup of a database, TimesTen Scaleout performs the backup asynchronously on each replica set and creates a sub-collection for each replica set that is backed up.

When you are considering backing up and restoring a TimesTen Scaleout database, keep in mind that:

- The current grid topology must be the same size or larger than the topology from the grid of the database backup. If your current grid topology is not large enough for *n* replica sets, TimesTen Scaleout displays an error message. That is, if you backup a database with three replica sets and you want to restore into a database that has only two replica sets, this operation will fail. However, you can use the export and import feature of TimesTen Scaleout to import a database from a grid topology with more replica sets into a database of a grid topology with less replica sets. See "Exporting and importing a database" on page 10-15 for more information.

- You can restore a backup into a grid of the same grid topology, even the same grid from which the backup was created. That is, if you create a backup of a database where there are three replica sets, then you can restore into the same grid or a new grid where there are three replica sets.
- You can restore a backup into a grid that has a larger topology than the grid where the backup was created. If you back up a database that has *n* replica sets, the restore operation creates a database with exactly *n* replica sets. However, if your current grid topology is larger than the original grid topology, TimesTen Scaleout creates the additional elements, but TimesTen Scaleout does not add these elements to the distribution map of the database and no data is stored on these elements. Instead, the restore only populates the same number of replica sets as the original grid topology. That is, if you create a backup of a grid where there are three replica sets, you can restore a backup into a new grid where there are four replica sets. However, the restore only populates three of those four replica sets. Thus, in order to populate all replica sets, you must redistribute the data across all replica sets after the restore using the `ttGridAdmin dbDistribute` command. See "Redistributing data in a database" on page 8-8 for more information.
- There are two type of backups: normal or staged.
  - Normal backups can be performed either on a repository mounted using NFS on each host of your grid or on a repository where each host of your grid uses SSH/SCP to connect to it. The time it takes to create a normal backup varies based on the size of your database, but you should expect every backup to take roughly the same time to complete.
  - Staged backups can only be performed on a repository where each host of your grid uses SSH/SCP to connect to it. Even though the first staged backup may take a similar time to complete as a normal backup (or even longer based on the performance of your network), all subsequent staged backups should take only a small fraction of that time to complete. Staged backups are ideal when you want to make regular backups on a second site that is independent to your main site.

  See "Determining the size of a backup or export" on page 10-18 for information on the file system space each backup operation requires.

> **Note:** If the database where the data would be restored is from a version of TimesTen Scaleout that is not patch-compatible, such as for a major upgrade, then you cannot backup and restore a database. Instead, you must export and import that database. See "Exporting and importing a database" on page 10-15 for more information.

TimesTen Scaleout enables you to perform the following procedures with backups:

- Back up a database
- Back up a database into a remote repository (WAN-friendly)
- Check the status of a backup
- Delete a backup
- Restore a database
- Check the status of a restore

Back up a database

Regular backups minimize the risks for potential data loss. Before attempting to back up your database, ensure that you have configured a repository for your grid. See "Working with repositories" on page 10-4 for more information.

This example creates a backup of the database `database1` and stores that backup in the repository `repo1`. By default, TimesTen Scaleout names your backup with the current date and time, `B`*yyyymmddhhss*. The prefix of the backup name, `B`, stands for backup. Ensure that you run the `ttGridAdmin dbBackup` command on a management instance.

> **Note:** You can add the `-name` parameter to specify a backup name. For example, `ttGridAdmin dbBackup database1 -repository repo1 -name mybackup` creates a backup named `mybackup`.

```
% ttGridAdmin dbBackup database1 -repository repo1
dbBackup B20170222145544 started
```

Depending on the size of your database, the number of replica sets that your database uses, the performance of your secondary storage device, and the performance of your network the backup time varies. The `ttGridAdmin dbBackup` command only starts the backup process and the output does not indicate that the backup is complete. Use the `ttGridAdmin dbBackupStatus` command to see the status of your backup. See "Check the status of a backup" on page 10-12 for more information.

For more information about the `ttGridAdmin dbBackup` command, see "Back up a database (dbBackup)" in the *Oracle TimesTen In-Memory Database Reference*.

Back up a database into a remote repository (WAN-friendly)

Normal backups to repositories using the SCP method require two copies of the most recent checkpoint and transaction logs files for each replica set. One copy consists of the checkpoint and log files of one element for each replica set, which are temporarily copied to a directory in the instance home. The second copy consists of the same checkpoint and log files per replica set after they are sent and stored at the repository, which construct the backup itself.

TimesTen Scaleout enables you to create staged backups to SCP repositories. This type of backup eliminates the overhead of creating local copies of the checkpoint and log files and reduces the network traffic required to create a remote copy in the repository. To accomplish this, staged backups use symbolic links instead of temporary local files (with the exception of the latest log file) and maintain a staging directory on the repository with the checkpoint and log files per replica set used for the latest backup. The next staged backup will copy the latest log files from each replica set and synchronize the rest of the files in the staging directory over the network. Finally, the repository uses the resulting files in the staging directory to create the backup, which removes the load of that task from the data instances and network.

Note:

- The system hosting the repository makes use of the Linux `cp` and `rsync` commands and the TimesTen `ttTransferAgent` utility to perform staged backup operations. The `ttTransferAgent` utility is copied to the staging directory at the beginning of a staged backup if it is not already available from a previous staged backup.
- See "Working with repositories" on page 10-4 for more information on SCP repositories.

The next sections describe the recommended settings for staged backups and how to create a staged backup:

- Recommendations for staged backups
- Create a staged backup

Recommendations for staged backups

Consider these before starting staged backups for your database:

- Prerequisites
- SSH config file
- BackupFailThreshold attribute
- File system space
- WAN throughput

Prerequisites Staged backups have these prerequisites:

- Passwordless SSH access: Staged backups require that all hosts with instances (data and management) have passwordless SSH access for the instance administrator to the system hosting the repository. See "Setting passwordless SSH" on page 2-12 for more information.
- The `rsync` command: Staged backups require that the `rsync` command is available on hosts with data instances and on the system allocating the repository.

SSH config file Staged backups depend on SSH for data transport and control. On every host with a data instance, consider updating the SSH configuration file for the instance administrator (`/home/`*`instance_administrator`*`/.ssh/config`) or the global SSH configuration file (`/etc/ssh/ssh_config`) to improve the reliability of staged backups. These options may prove useful:

- `HostName`: You can use this option to specify multiple aliases for the repository. SSH tries them in order. Provide a list of multiple aliases in a different order to every host.
- `Port`: SSH uses by default port 22. You may need to use a different port number if SSH has to pass through a NAT gateway.
- `BindAddress` or `BindInterface`: You can use these options to control which Ethernet interface SSH will use to contact the repository.
- `ConnectionAttempts`: By default SSH only makes one connection attempt. You can use this option to set how many connection attempts SSH will make before aborting and returning a failure notification.

- `ConnectTimeout`: By default SSH uses the system TCP timeout. You can use this option to set the timeout (in seconds) to establish a SSH connection. Consider increasing this connection timeout on high-latency WAN links.
- `ProxyJump`: You can use this option to set bastion hosts to serve as proxies to connect to the repository. The hosts with a data instance may be able to access the bastion hosts but not other hosts, like the repository. Likewise, the bastion hosts may be able to access the remote repository. You can configure multiple bastion hosts for high availability.
- `ServerAliveCountMax`: You can use this option to set the maximum number of keepalive messages sent through the encrypted channel by a host without receiving any message back from the repository. The connection is terminated after reaching this threshold. You must use this option in conjunction with the `ServerAliveInterval` option.
- `ServerAliveInterval`: You can use this option to set the time (in seconds) between receiving no data from the repository and the host sending a keepalive message. This serves to detect if the repository has crashed or the network has gone down.

On the system hosting the repository, consider setting this option in the global SSH daemon configuration file (`/etc/ssh/sshd_config`):

- `MaxStartups`: You can use this option the set the maximum number of concurrent unauthenticated connections to the SSH daemon. Consider setting the `start` parameter to a value larger than the number of replica sets and the `full` parameter to ten times the value of the `start` parameter. For example, if you have ten replica sets, set this option as:

  ```
  MaxStartups 15:30:150
  ```

BackupFailThreshold attribute The `BackupFailThreshold` first connection attribute determines the number of transactions log files that can accumulate in the `LogDir` directory since the start of a backup before TimesTen is forced to release the hold on checkpoint operations. If a checkpoint is initiated before the completion of a backup, the backup is invalidated.

Set the `BackupFailThreshold` attribute to a value that is high enough to ensure the safe completion of your backup. For example, if a backup typically takes *`n`* seconds to complete and your database creates *`m`* transaction log files per second, then set `BackupFailthreshold` to a value greater than `n * m`. The number of log files generated by your database per any given unit of time is directly proportional to your write workload and inversely proportional to the value set for the `LogFileSize` attribute.

See "Modifying the connection attributes of a database" on page 5-30 for information on how to modify the value of a first connection attribute.

For more information on the `BackupFailThreshold`, `LogDir`, and `LogFileSize` connection attributes, see "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

File system space To avoid out-of-space failures due to staged backups, ensure that:

- The file system used by each data instance has enough space for one transaction log file (`LogFileSize`) in the instance home plus enough space to store in `LogDir` all the transaction log files that may be generated while the backup operation is in progress (`BackupFailThreshold * LogFileSize`).
- The file system used by the repository has enough space to store as many backups you wish to retain plus enough space in the staging directory for 1.25 backups for all staged backups of the same database.

See "Determining the size of a backup or export" on page 10-18 for information on the file system space each backup operation requires.

For more information on the LogFileSize, LogDir, and BackupFailThreshold connection attributes, see "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

WAN throughput The minimum WAN throughput required by a subsequent staged backup depends on the aggregate size of the database, the desired time for the backup to take, and the speed-up factor provided by the staging repository and the defined transfer compression. You will need to perform a series of staged backups to test how much the performance of your network and overall setup (plus the inherent advantages subsequent staged backups over normal backups provide) reduces the backup time for regular staged backups under typical workload conditions. Consider this formula:.

```
Minimum WAN throughput = file size of a backup/(desired backup time * (first
staged backup time/average subsequent staged backups time))
```

See "Determining the size of a backup or export" on page 10-18 for information on the file system space each backup operation requires.

Create a staged backup

This example creates a staged backup named stgbackup1 of the database1 database and stores that backup in the scprepo1 repository. The staged backup is set to use an aggregate network traffic of 62 MB per second and a compression level of 9 for that network traffic.

```
% ttGridAdmin dbBackup database1 -repository scprepo1 -name stgbackup1 -backupType
staged -bwlimit 62 -compression 9
dbBackup stgbackup1 started
```

Note: Ensure that you run the ttGridAdmin dbBackup command as the instance administrator on the active management instance.

Depending on the size of your database, the number of replica sets that your database uses, the performance of your secondary storage device, and the performance of your network the backup time varies. The ttGridAdmin dbBackup command only starts the backup process and the output does not indicate that the backup is complete. Use the ttGridAdmin dbBackupStatus command to see the status of your backup. See "Check the status of a backup" on page 10-12 for more information.

For more information about the ttGridAdmin dbBackup command, see "Back up a database (dbBackup)" in the *Oracle TimesTen In-Memory Database Reference*.

Check the status of a backup

The ttGridAdmin dbBackupStatus command enables you to view the progress of all backup processes for a specific database.

This example displays the status of all backup processes for the database database1.

```
% ttGridAdmin dbBackupStatus database1
Database  Backup          Repository Host  Instance  Elem State     Started                Finished
--------- --------------- ---------  ----- --------- ---- --------- ---------------------- --------
database1 B20170222145544 repo1                           Completed 2017-02-22T14:55:44.000Z Y
                                     host4 instance1    2 Complete
```

```
host5 instance1    3 Complete
```

Ensure that the `ttGridAdmin dbBackupStatus` output shows that the overall state of the backup process is marked as `Completed`. In case that you see a state value of `Failed`, perform these tasks:

- Use the `ttGridAdmin dbStatus` *`database1`* `-details` command to ensure that the host and instance of that element are up and running. If at least one host from each replica set is up, the `ttGridAdmin dbBackup` command can create a full backup of your database. For more information, see "Monitor the status of a database (dbStatus)" in the *Oracle TimesTen In-Memory Database Reference*.
- Ensure that the repository where you are attempting to create the backup has enough free file system space to create a backup of your database.

If the backup failed, you may attempt to perform another backup using a different backup name. If a new backup name does not perform a successful backup, diagnose the issue and perform any necessary fixes. After you have resolved the problem, use the `ttGridAdmin dbBackupDelete` to delete any failed backups. TimesTen Scaleout does not automatically delete a failed backup. Then, use the `ttGridAdmin dbBackup` command to start a new backup. Depending on your available file system space, you can use these commands in any order. See "Delete a backup" on page 10-13 and "Back up a database" on page 10-9 for more information.

For more information about the `ttGridAdmin dbBackupStatus` command, see "Display the status of a database backup (dbBackupStatus)" in the *Oracle TimesTen In-Memory Database Reference*.

Delete a backup

TimesTen Scaleout does not automatically delete backups. In some cases, you may want to delete backups that have failed or old backups to free up file system space.

Use the `ttGridAdmin repositoryList -contents` command to view all of your available backups and their respective repositories. See "List repositories and collections" on page 10-7 for more information.

This example deletes the backup named `B20170222145544` from repository `repo1`.

```
% ttGridAdmin dbBackupDelete -repository repo1 -name B20170222145544
Backup B20170222145544 deleted
```

TimesTen Scaleout deletes the collection and all of the sub-collections that are part of the backup.

For more information about the `ttGridAdmin dbBackupDelete` command, see "Delete a database backup (dbBackupDelete)" in the *Oracle TimesTen In-Memory Database Reference*.

Restore a database

Before attempting to restore a database, consider the following:

- The database definition name must not be in use by other databases when you attempt to perform a database restore. For example, you cannot name the restored database `database1` if another database is using the `database1` name.
- The database definition of the backed up database does not need to match the database name of the database that you are restoring. For example, you can restore a backup of the `payroll` database to the `new_payroll` database definition.

- The K-safety value of the database that you backed up does not need to match the K-safety value of the restore database.
- The database definition must have at least as many connections as the database definition of the backed up database.

This example restores the database `res_db1` from the backup `B20170222145544` from repository `repo1`. Ensure that you run the `ttGridAdmin dbRestore` command on the management instance.

```
% ttGridAdmin dbRestore res_db1 -repository repo1 -backup B20170222145544
dbRestore B20170222145544 started
```

Note: Ensure that the `res_db1` database definition exists before attempting to perform a restore. You do not need to create a database from this database definition. See "Create a database definition" on page 5-2 for more information.

Depending on the size of your backup, the number of replica sets that your database uses, the performance of your secondary storage device, and the performance of your network the restore time varies. The `ttGridAdmin dbRestore` command only starts the restore process and the output does not indicate that the restore is complete. The restore process is performed asynchronously on every element. Use the `ttGridAdmin dbRestoreStatus` command to see the status of your restore. See "Check the status of a restore" on page 10-14 for more information.

For more information about the `ttGridAdmin dbRestore` command, see "Restore a database (dbRestore)" in the *Oracle TimesTen In-Memory Database Reference*.

Check the status of a restore

The `ttGridAdmin dbRestoreStatus` command enables you to view the progress of the restore process for a specific database.

This example displays the status of all restore processes for the database `res_db1`.

```
% ttGridAdmin dbRestoreStatus res_db1
Database Restore Repository Host  Instance  Elem State                   Started          Finished
-------- ------- ---------- ----- --------- ---- ------------------------------------------------ -
res_db1  mybkup  repo1                           Restore_Finale_Complete 2017-03-03T13:19:39.000Z Y
                            host3 instance1      Restore_Instance_Complete
                            host4 instance1      Restore_Instance_Complete
                            host5 instance1      Restore_Instance_Complete
                            host6 instance1      Restore_Finale_Comnplete
```

Ensure that the `ttGridAdmin dbRestoreStatus` output shows that the restore has been completed for every element of your grid. The restore operation is fully completed when the `State` column of the row with the database name is marked as `Completed`.

Ensure that the `ttGridAdmin dbRestoreStatus` output shows that the overall state of the restore process is marked as `Completed`. In case that you see a state value of `Failed` or `Restore_Instance_Failed` for an element or an overall state of `Restore_Finale_Failed` or `Restore_Init_Failed`, stop the database with `ttGridAdmin dbClose` and `ttGridAdmin dbUnload` commands. Once you have stopped the database, use the `ttGridAdmin dbDestroy` command to delete the database that did not restore successfully. Then, attempt the restore operation again. See "Unloading a database from memory" on page 5-27 and "Destroying a database" on page 5-33 for more information.

For more information about the `ttGridAdmin dbRestoreStatus` command, see "Display the status of a database restore (dbRestoreStatus)" in the *Oracle TimesTen In-Memory Database Reference*.

Exporting and importing a database

The TimesTen Scaleout export and import functionality enables you to migrate data between two grid databases.

In these circumstances you must export a database:

- The source database is from a version of TimesTen Scaleout that is not patch-compatible such as for a major upgrade. See "Upgrading a grid" on page 8-18 for more information on both types of upgrades (patch-compatible or otherwise).
- The destination database is in a grid topology that has fewer replica sets than the grid topology where the database is exported.

When you export a database, TimesTen Scaleout performs the export asynchronously of each replica set and creates a sub-collection for each replica set that is exported.

See "Determining the size of a backup or export" on page 10-18 for information on the file system space each export operation requires.

TimesTen Scaleout enables you to perform the following procedures with database exports:

- Export a database
- Check the status of a database export
- Delete a database export
- Import a database export
- Check the status of a database import

Export a database

Before attempting to export a database, ensure that you have configured a repository for your grid. See "Working with repositories" on page 10-4 for more information.

Ensure that you disconnect all application connections to the database before performing a database export to ensure that no applications are modifying data during the database export operation. Also, ensure that you close the database to prevent any new connections to the database. Any transaction committed during an export operation may result in an inconsistent database.

This example creates a database export of the database `database1` and stores that export in the repository `repo1`. By default, TimesTen Scaleout names your database export with the current date and time, `Myyyymmddhhss`. Ensure that you run the `ttGridAdmin dbExport` command on a management instance.

Note: You can add the `-name` parameter to specify a database export name. For example, `ttGridAdmin dbExport database1 -repository repo1 -name myexport` creates a database export named `myexport`.

```
% ttGridAdmin dbExport database1 -repository repo1
dbExport M20170302144218 started
```

Depending on the size of your database, the number of replica sets that your database uses, the performance of your secondary storage device, and the performance of your network the database export time varies. Use the `ttGridAdmin dbExportStatus` command to see the status of your database export. See "Check the status of a database export" on page 10-16 for more information.

For more information about the `ttGridAdmin dbExport` command, see "Export a database (dbExport)" in the *Oracle TimesTen In-Memory Database Reference*.

Check the status of a database export

The `ttGridAdmin dbExportStatus` command enables you to view the progress of all database export processes for a specific database.

This example displays the status of all database export processes for the database `database1`.

```
% ttGridAdmin dbExportStatus database1
Database   Export          Repository Host Instance Elem State      Started
---------  --------------- ---------- ---- -------- ---- ---------- ------------------------
database1  M20170321073022 repo1                         Completed  2017-03-21T07:30:27.000Z
                                      host3 instance1    Complete
                                      host5 instance1    Complete
```

Ensure that the `ttGridAdmin dbExportStatus` output shows that a database export has been completed for every replica set of your grid. In case that you see a state value of `Failed` for an element, perform these tasks:

- Use the `ttGridAdmin dbStatus` *`database1`* `-details` command to ensure that the host and instance of that element are up and running. For more information, see "Monitor the status of a database (dbStatus)" in the *Oracle TimesTen In-Memory Database Reference*.
- Ensure that the repository where you are attempting to create the backup has enough free file system space to create a backup of your database.

After you have resolved the issues that caused the export to fail, use the `ttGridAdmin dbExportDelete` to delete the failed database export. TimesTen Scaleout does not automatically delete a failed database export. Then, use the `ttGridAdmin dbExport` command to start a new database export. See "Delete a database export" on page 10-16 and "Export a database" on page 10-15 for more information.

For more information about the `ttGridAdmin dbExportStatus` command, see "Display the status of a database export (dbExportStatus)" in the *Oracle TimesTen In-Memory Database Reference*.

Delete a database export

TimesTen Scaleout does not automatically delete database exports. In some cases, you may want to delete database exports that have failed or old database exports to free up file system space.

Use the `ttGridAdmin repositoryList -contents` command to view all of your available database exports and their respective repositories. See "List repositories and collections" on page 10-7 for more information.

This example deletes the database export named `M20170321073022` from repository `repo1`.

```
% ttGridAdmin dbExportDelete -repository repo1 -name M20170321073022
```

```
Export M20170321073022 deleted
```

TimesTen Scaleout deletes all of the sub-collections that are part of the database export.

For more information about the `ttGridAdmin dbExportDelete` command, see "Delete a database export (dbExportDelete)" in the *Oracle TimesTen In-Memory Database Reference*.

Import a database export

Before attempting to import a database export, consider the following:

- The database to which you import must exist when you attempt to perform a database import. The database can either contain data or be empty. It is not necessary to create the users or tables of the original database.
- The database name of the database that you exported does not need to match the database name of the database where you are importing the database export. For example, you can import a database export of the `payroll` database in the `new_payroll` database.
- The K-safety value of the database that you exported does not need to match the K-safety value of the grid where you are importing the database export.
- Ensure that you disconnect all application connections to the database before performing a database import to ensure that no applications are modifying data during the database import operation. Also, ensure that you close the database to prevent any new connections to the database. Any transaction committed during an import operation may result in an inconsistent database. See "Close a database (dbClose)" in the *Oracle TimesTen In-Memory Database Reference* for more information.

This example imports the database `import_db` from the database export `M20170321073022` from repository `repo1`. Ensure that you run the `ttGridAdmin dbImport` command on a management instance.

```
% ttGridAdmin dbImport import_db -repository repo1 -name M20170321073022
-numThreads 8
dbImport M20170321073022 started
```

> **Note:** Ensure that the `import_db` database exists before attempting to perform a restore. See "Create a database definition" on page 5-2 for more information.

Depending on the size of your database export, the number of replica sets that your database uses, the performance of your secondary storage device, and the performance of your network the import time varies. To increase the performance of the import operation, use the `-numThreads` option to specify the number threads that concurrently read rows from the export database and insert them into the import database.

For more information about the `ttGridAdmin dbImport` command, see "Import a database (dbImport)" in the *Oracle TimesTen In-Memory Database Reference*.

Check the status of a database import

The `ttGridAdmin dbImportStatus` command enables you to view the progress of the import process for a specific database.

This example displays the status of all import processes for the database `import_db`.

```
% ttGridAdmin dbImportStatus import_db
Database  Import         Repository Host Instance Elem State      Started
--------- ---------------------------- -------- ---- ---------- ----------------------------------
database1 M20170321073022 repo1                      Import_Finale_Complete2017-03-21T10:30:27.000Z
                                   host3 instance1 1 Import_Rows_Complete
                                   host5 instance1 1 Import_Rows_Complete
```

Ensure that the `ttGridAdmin dbImportStatus` output shows that the import operation has been completed for every element of your grid. The import operation is fully completed when the `State` column of the row with the database name is marked as `Completed`.

In case that you see a state value of `Failed` for an element, use the `ttGridAdmin dbDestroy` command to delete the database that did not import successfully. Then, recreate the database and attempt the import operation again. See "Destroying a database" on page 5-33 for more information.

For more information about the `ttGridAdmin dbImportStatus` command, see "Display the status of a database import (dbImportStatus)" in the *Oracle TimesTen In-Memory Database Reference*.

Determining the size of a backup or export

Every database backup and export stored in a repository requires file system space (in megabytes) that is equivalent to the value assigned to the `PermSize` attribute plus the sum of file sizes of the transaction log files created after the latest checkpoint, per replica set.

The file size of transaction log files and how many are typically written between background checkpoints is dependent of the configuration of your database. Your typical workload and the settings of attributes like `CkptFrequency`, `CkptLogVolume`, and `LogFileSize` have direct impact in determining how many transaction log files would need to be considered for a backup or export operation. See "Storage provisioning for transaction log files" in the Oracle TimesTen In-Memory Database Operations Guide for more information.

Additionally, each data instance requires available temporary file system space (`/`*`instance_home`*`/grid/admin/temp/`) that is equivalent to the size of a database backup or database export divided by the number of replica sets for every normal backup, export, restore, or import operation. Staged backups only require temporary file system space equivalent to one transaction log file (`LogFileSize`).

11

Recovering from Failure

Error conditions and failure situations can impact availability. If the error condition can be recovered automatically, then normal operations resume. However, there may be situations where you need to intervene to recover from failure.

TimesTen Scaleout has included error and failure detection with automatic recovery for many error and failure situations in order to maintain a continuous operation for all applications using TimesTen Scaleout. Errors and failure situations can include:

- Software errors.
- Network outage or other communication channel failures. A communication channel is a TCP connection.
- One or more machines hosting a data instance unexpectedly reboots or crashes.
- The main TimesTen daemon for an instance or any of its sub-daemons fail.
- An element becomes slow or unresponsive either from a hang situation or as a result of a heavy load.
- A machine or rack of machines hosting data instances are unexpectedly brought down for unknown reasons.

The response necessary for error conditions and failure situations are as follows:

- Transient errors: A transient error is due to a temporary condition that TimesTen Scaleout is usually able to quickly resolve. You can immediately retry the failed transaction, which normally succeeds.
- Element failure: When an element fails, TimesTen Scaleout can automatically recover the element most of the time. However, there are certain element failure situations where you may be required to fix the problem. The application response to an element failure may differ depending on the configuration of the grid and the database. After the problem is fixed, either TimesTen Scaleout recovers the element and operations continue or you supply a new element to take the place of the failed element.
- Replica set failure: If all of the elements in a replica set fail, there is a method for TimesTen Scaleout to automatically recover the elements (once the original failure issue has been fixed). The element with the latest changes, known as the seed element, is recovered first. Then, all subsequent elements are recovered from the seed element.
- Database failure: If all replica sets fail, the database is considered failed. You need to reload the database for recovery. How a database recovers when the database reloads depends on the value for the `Durability` attribute.

- Data distribution failure: You can attempt a re-synchronization of your data if the data distribution process is interrupted or fails to complete. Re-synchronization involves executing the `ttGridAdmin dbDistribute -resync` operation.

The following sections describe the error or failure situations and recovery:

Displaying the database, replica set and element status

The element status shows:

- If the element is loaded (`opened`).
- If the element is in process of a change, such as being opened (`opening`), loaded (`creating`, `loading`), unloaded (`unloading`), destroyed (`destroying`) or closed (`closing`).
- If the element or its data instance has failed and is waiting on the seed element to recover, then the status displayed is `waiting for seed`. The element that failed with the latest changes, known as the seed element, is recovered first to the latest transaction in the checkpoint and transaction log files. The other element in the replica set is copied from the seed element of the replica set.
- If the element is not up (`evicted` or `down`).

The following examples show how to display the status of the database, data space groups, replica sets and elements. See "Troubleshooting based on element status" on page 11-10 for details on how to respond to each status.

Example 11–1 Displaying the status of the database and all elements

You can use the `ttGridAdmin dbStatus -all` command to list the current status for the database, all elements, replica sets and data space groups in your database.

The first section describes the status of the overall database. In this example, the database has been created, loaded, and open. The status also shows the total number of created, loaded and open elements.

The database status shows the progression of the database being first created, then loaded and finally opened. In bringing down the database, the reverse order is performed, where the database is first closed, then unloaded and finally destroyed.

```
% ttGridAdmin dbStatus database1 -all
Database database1 summary status as of Thu Feb 22 07:37:28 PST 2018
```

```
created,loaded-complete,open
Completely created elements: 6 (of 6)
Completely loaded elements: 6 (of 6)
Completely created replica sets: 3 (of 3)
Completely loaded replica sets: 3 (of 3)

Open elements: 6 (of 6)
```

However, if the database status shows that the database is created, loaded and closed, then the database has not yet been opened. The following example shows that the database is not open yet, but that the distribution map has been updated, showing the created and loaded replica sets. Note that none of the elements are opened until the database is opened.

```
% ttGridAdmin dbStatus database1 -all
Database database1 summary status as of Thu Feb 22 07:37:01 PST 2018

created,loaded-complete,closed
Completely created elements: 6 (of 6)
Completely loaded elements: 6 (of 6)
Completely created replica sets: 3 (of 3)
Completely loaded replica sets: 3 (of 3)

Open elements: 0 (of 6)
```

The second section provides information about the elements: the host and instance name in which each element exists, the number assigned to the element, and the status of the element.

```
Database database1 element level status as of Thu Feb 22 07:37:28 PST 2018

Host  Instance  Elem Status Date/Time of Event  Message
----- --------- ---- ------ ------------------- -------
host3 instance1    1 opened 2018-02-22 07:37:25
host4 instance1    2 opened 2018-02-22 07:37:25
host5 instance1    3 opened 2018-02-22 07:37:25
host6 instance1    4 opened 2018-02-22 07:37:25
host7 instance1    5 opened 2018-02-22 07:37:25
host8 instance1    6 opened 2018-02-22 07:37:25
```

The third section provides information about the replica sets. In this example, there are three replica sets. In addition to information about the elements, it also provides the number of the replica set in which each element exists, identified by the `RS` column. The data space group in which each element exists (within its data instance within its host) is identified with the `DS` column. Notice that each replica set has one element in each data space group.

```
Database database1 Replica Set status as of Thu Feb 22 07:37:28 PST 2018

RS DS Elem Host  Instance  Status Date/Time of Event  Message
-- -- ---- ----- --------- ------ ------------------- -------
 1  1    1 host3 instance1 opened 2018-02-22 07:37:25
    2    2 host4 instance1 opened 2018-02-22 07:37:25
 2  1    3 host5 instance1 opened 2018-02-22 07:37:25
    2    4 host6 instance1 opened 2018-02-22 07:37:25
 3  1    5 host7 instance1 opened 2018-02-22 07:37:25
    2    6 host8 instance1 opened 2018-02-22 07:37:25
```

The final section organizes the information about the elements to show which elements are located in each data space group, shown under the `DS` column. In this example, there are two data space groups. The elements are organized either under data space group 1 or 2.

```
Database database1 Data Space Group status as of Thu Feb 22 07:37:28 PST 2018

DS RS Elem Host  Instance  Status Date/Time of Event  Message
-- -- ---- ----- --------- ------ ------------------- -------
 1  1    1 host3 instance1 opened 2018-02-22 07:37:25
    2    3 host5 instance1 opened 2018-02-22 07:37:25
    3    5 host7 instance1 opened 2018-02-22 07:37:25
 2  1    2 host4 instance1 opened 2018-02-22 07:37:25
    2    4 host6 instance1 opened 2018-02-22 07:37:25
    3    6 host8 instance1 opened 2018-02-22 07:37:25
```

The following shows the status if you evicted one of your replica sets without replacement. While the database is loaded and opened, it shows that there are six created elements, but only four of those are loaded. There is one less replica set in all displayed sections and the evicted elements are shown as evicted with their status.

```
% ttGridAdmin dbStatus database1 -all
Database database1 summary status as of Thu Feb 22 07:52:08 PST 2018

created,loaded-complete,open
Completely created elements: 6 (of 6)
Completely loaded elements: 4 (of 6)
Completely created replica sets: 2 (of 2)
Completely loaded replica sets: 2 (of 2)

Open elements: 4 (of 6)

Database database1 element level status as of Thu Feb 22 07:52:08 PST 2018

Host  Instance  Elem Status  Date/Time of Event  Message
----- --------- ---- ------- ------------------- -------
host3 instance1    1 evicted 2018-02-22 07:52:06
host4 instance1    2 evicted 2018-02-22 07:52:06
host5 instance1    3 opened  2018-02-22 07:37:25
host6 instance1    4 opened  2018-02-22 07:37:25
host7 instance1    5 opened  2018-02-22 07:37:25
host8 instance1    6 opened  2018-02-22 07:37:25

Database database1 Replica Set status as of Thu Feb 22 07:52:08 PST 2018

RS DS Elem Host  Instance  Status Date/Time of Event  Message
-- -- ---- ----- --------- ------ ------------------- -------
 1  1    3 host5 instance1 opened 2018-02-22 07:37:25
    2    4 host6 instance1 opened 2018-02-22 07:37:25
 2  1    5 host7 instance1 opened 2018-02-22 07:37:25
    2    6 host8 instance1 opened 2018-02-22 07:37:25

Database database1 Data Space Group status as of Thu Feb 22 07:52:08 PST 2018

DS RS Elem Host  Instance  Status Date/Time of Event  Message
-- -- ---- ----- --------- ------ ------------------- -------
 1  1    3 host5 instance1 opened 2018-02-22 07:37:25
    2    5 host7 instance1 opened 2018-02-22 07:37:25
 2  1    4 host6 instance1 opened 2018-02-22 07:37:25
    2    6 host8 instance1 opened 2018-02-22 07:37:25
```

See "Troubleshooting based on element status" on page 11-10 in this guide and "Database management operations" and "Monitor the status of a database (dbStatus)" in the *Oracle TimesTen In-Memory Database Reference* for full details on the different status options.

Recovering from transient errors

Because a grid spans multiple hosts, there is an opportunity for multiple types of failure, many of which can be transient errors. For the most part, TimesTen Scaleout can detect transient errors and adapt to them quickly. Most errors in the grid are transient with error codes designated as `Transient`, which may cause a specific API, SQL statement or transaction to fail. Most of the time, the application can retry the exact same operation with success.

The potential impacts of a transient error are:

- The execution of a particular statement failed. Your application should re-execute the statement.
- The execution of a particular transaction failed. Your application should roll back the transaction and perform the operations of the transaction again.
- The connection to the data instance fails. If you are using a client/server connection, then the TimesTen Scaleout routes the connection to another active data instance. See "Client connection failover" on page 11-30 for full details.

The following sections describe how TimesTen Scaleout recovers the element from the more common transient errors:

- Retry transient errors
- Communications error
- Software error
- Host or data instance failure
- Heavy load or temporary communication failure

Retry transient errors

While TimesTen Scaleout automatically handles the source of most transient errors, your application may retry the entire transaction when receiving the error described in Table 11–1.

Table 11–1 SQLSTATE and ORA errors for retrying after transient failure

SQLSTATE	ORA errors	PL/SQL exceptions	Error message
`TT005`	`ORA-57005`	Exception `-57005`	Transient transaction failure due to unavailability of a grid resource. Roll back the transaction and then retry the transaction.

Your applications can check for the transient error as follows:

- ODBC or JDBC applications check for the `SQLSTATE TT005` error to determine if the application should retry the transaction. See "Retrying after transient errors (ODBC)" in the *Oracle TimesTen In-Memory Database C Developer's Guide* and "Retrying after transient errors (JDBC)" in the *Oracle TimesTen In-Memory Database Java Developer's Guide* for more details.

- OCI and Pro*C applications check for the `ORA-57005` error to determine if the application should retry a SQL statement or transaction. See "Transient errors (OCI)" in the *Oracle TimesTen In-Memory Database C Developer's Guide* for more details.
- PL/SQL applications check for the `-57005` PL/SQL exception to determine if the application should retry the transaction. See "Retrying after transient errors (PL/SQL)" in the *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide* for more details.

Communications error

The following describes the type of communications that might fail:

- Communication between elements: Used to execute SQL statements within transactions and stream data between elements, as required. If there is a communications error while the application is executing a transaction, then you must roll back the transaction. When you retry the transaction, communications are recreated and work continues.
- Communication between data instances: The data instances communicate with each other for creating communication as well as sending or receiving recovery messages. If there is a break in the communication between the data instances, then communications are automatically recovered when you retry the operation.
- Communication between data instances and the ZooKeeper membership servers: Each data instance communicates with the ZooKeeper membership service through one of the defined ZooKeeper servers. If communications fail between a data instance and the ZooKeeper server with which it has been communicating, then the data instance attempts to connect to another ZooKeeper server. If the data instance cannot connect to any ZooKeeper server, then the data instance considers itself to be down.

  See "Recovering when a data instance is down" on page 11-29 for details on what to do when a data instance is down.

Software error

If a software error causes an element to be unloaded, then an error is returned to the active application. After rolling back the transaction, the application can continue executing transactions as long as one element from each replica set is open.

TimesTen Scaleout attempts to reload the element. Once opened, the element can accept transactions again.

> **Note:** You can manually initiate the reload of an element by reloading the database with the `ttGridAdmin dbload` command. If element status is `load failed`, fix what caused the element load to fail and then reload the element with the `ttGridAdmin dbload` command. See "Load a database into memory (dbLoad)" in the *Oracle TimesTen In-Memory Database Reference* for details.

Host or data instance failure

If the host that contains a data instance crashes or if the data instance crashes, then an error is returned to the active application. Since the data instance is down, the element status is displayed as `down`. If the data instance restarts (whether from automatic recovery or manual intervention), the element within the data instance most likely recovers. Monitor the status of the element with the `ttGridAdmin dbStatus` command to verify if it did recover.

> **Note:** See "Troubleshooting based on element status" on page 11-10 for information on how to respond to the element status. See "Recovering when a data instance is down" on page 11-29 on how to manually recover a data instance.

Heavy load or temporary communication failure

A transient failure may occur if an element becomes slow or unresponsive due to heavy load. During a database operation, a transient failure can occur for many reasons.

- A query timeout may occur if one or more hosts of the TimesTen Scaleout are overloaded and are slow to respond.
- A transient failure occurs with a temporary suspension of communication, such as unplugging from the network to reset communications.

Recovering from a data distribution error

Your existing data is redistributed once you apply the change to the distribution map with the `ttGridAdmin dbDistribute -apply` command. (See "Redistributing data in a database" on page 8-8 for full details.) You receive an error if you request a data distribution or a reset while a data distribution is in progress.

TimesTen spawns multiple processes to perform data distribution. In addition, the active management instance communicates with the data instances to facilitate data distribution. The active management instance stores metadata to track the progress of each data distribution. Thus, the data distribution could fail if a critical process fails, an instance fails, or communication fails between the active management instance and the data instances.

The following error message displays if the `dbDistribute -apply` command fails during data distribution:

```
% ttGridAdmin dbDistribute database1 -apply
Error : Distribution failed, error message lost due to process failure
```

There are a few failure cases where the active management instance may not know about the success or failure of a data distribution operation and the metadata may be left in an intermediate state. This could occur if the process in which the `dbDistribute -apply` was executed dies or is killed.

Do not re-initiate another `dbDistribute -apply` command if the data distribution fails or does not complete. Instead, execute the `dbDistribute -resync` command. The `dbDistribute -resync` command examines the metadata in the active management instance to determine if a `dbDistribute -apply` operation was in progress but did not complete (neither committing nor rolling back the changes). If so, the `dbDistribute -resync` command re-synchronizes the metadata in the database with the metadata in the active management instance (if they do not have matching states).

- If the `dbDistribute -resync` command succeeds, the re-synchronization may result in committing or rolling back the metadata changes of the previous `dbDistribute -apply` operation.

- If the `dbDistribute -resync` command fails, you can either:
  - Execute the `dbDistribute -apply` command to attempt the same distribution.
  - Execute the `dbDistribute -reset` command to discard all distribution settings that have not yet been applied, then attempt a new data distribution with the `dbDistribute -apply` command.

The following example shows the output when the `dbDistribute -resync` command successfully completes the data distribution operation:

```
% ttGridAdmin dbDistribute -resync
Distribution map updated
```

The following example shows the output when the `dbDistribute -resync` command rolls back the data distribution operation:

```
% ttGridAdmin dbDistribute database1 -resync
Distribution map Rolled Back
```

The following example shows the output when the `dbDistribute -resync` command discovers that there is no data distribution in progress.

```
% ttGridAdmin dbDistribute database1 -resync
No DbDistribute is currently in progress
```

The following example shows the output when the `dbDistribute -resync` command discovers that the data distribution is still in progress.

```
% ttGridAdmin dbDistribute database1 -resync
Distribute is still in progress. Wait for dbDistribute to complete, then call
resync
```

An error displays if the re-synchronization fails. For example, you might attempt to re-synchronize a data distribution when there are no active data instances. In this case, the following error displays:

```
% ttGridAdmin dbDistribute database1 -resync
Error : Could not connect to data instance to retrieve partition table version
```

See "Set or modify the distribution scheme of a database (dbDistribute)" in the *Oracle TimesTen In-Memory Database Reference* for more details.

Tracking the automatic recovery for an element

If an element becomes unloaded, TimesTen Scaleout attempts to reload the element if the database is supposed to be loaded. During this time, the element status changes to `loading` as the element is being automatically recovered by TimesTen Scaleout.

You can monitor the element status with the `ttGridAdmin dbStatus -element` command. This example shows that the element on the `host3.instance1` data instance is in the process of recovering by showing a status of `loading`.

```
% ttGridAdmin dbStatus database1 -element
Database database1 element level status as of Wed Jan 10 14:34:08 PST 2018

Host  Instance  Elem Status  Date/Time of Event  Message
----- --------- ---- ------  ------------------- -------
host3 instance1    1 loading 2018-01-10 14:33:23
host4 instance1    2 opened  2018-01-10 14:33:21
host5 instance1    3 opened  2018-01-10 14:33:23
host6 instance1    4 opened  2018-01-10 14:33:23
```

```
host7 instance1    5 opened  2018-01-10 14:33:23
host8 instance1    6 opened  2018-01-10 14:33:23
```

See "Availability despite the failure of one element in a replica set" on page 11-9 and "Unavailability of data when a full replica set is down or fails" on page 11-15 for more details on what happens when an element or a full replica set goes down.

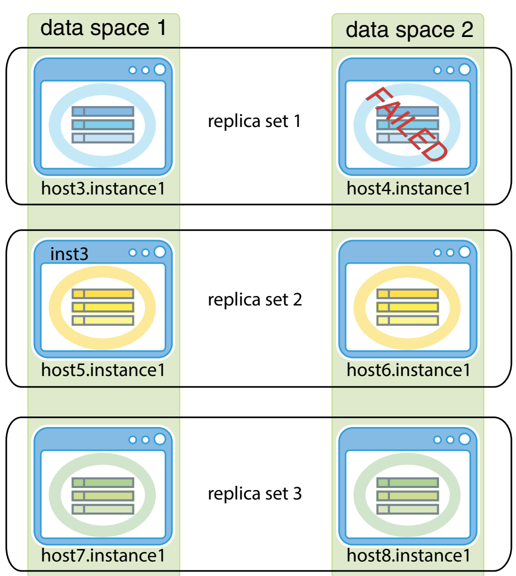
Availability despite the failure of one element in a replica set

A main goal for TimesTen Scaleout is to provide access to the data even if there are failures. When *k* = `2`, the data contained within a replica set is available as long as at least one element in the replica set is up. If an element in the replica set goes down and then recovers, then the element is automatically re-synchronized with the other element in its replica set.

Note: If *k* = `1`, any element failure results in the replica set being down because the replica set contains only a single element. See "Unavailability of data when a full replica set is down or fails" on page 11-15 for details on recovery when an element permanently fails when *k* = `1`.

The following example shows a grid where *k* = `2`. Three replica sets are created, each with two elements in the replica set. The element on the `host4.instance1` data instance fails. TimesTen Scaleout automatically re-connects to the element within the `host3.instance1` data instance to continue executing the transaction. While the element on the `host4.instance1` data instance is unavailable or in the middle of recovering, the element on the `host3.instance1` data instance handles all transactions for the replica set. Once the element on the `host4.instance1` data instance recovers, both elements in the replica set can handle transactions.

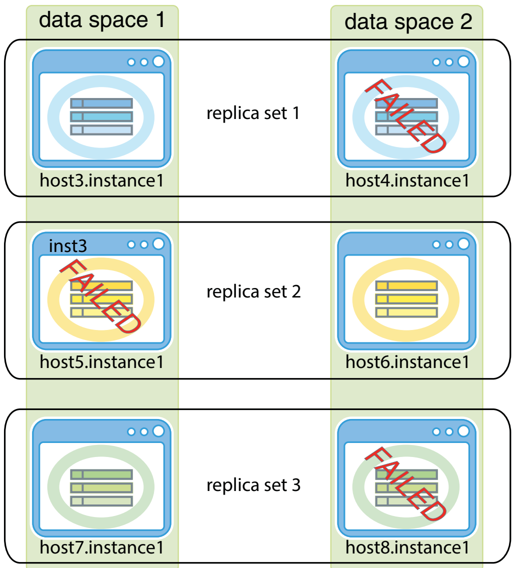
Figure 11–1 K-safety reacts to one data instance failure



Multiple failures in different replica sets do not result in loss of functionality, as long as there is one element up in each replica set. You may lose data if an entire replica set fails.

The following example shows a grid where k = 2 with three replica sets. In this example, the elements in the `host4.instance1`, `host5.instance1`, and `host8.instance1` data instances fail. However, your transactions continue to execute since there is at least one element available in each replica set.

Figure 11–2 K-safety reacts to multiple data instance failures



Recovering when a single element fails in a replica set

See the following sections on how to respond when a single element fails within a replica set when k=2:

- Troubleshooting based on element status
- Recovering a replica set after an element goes down
- Remove and replace a failed element in a replica set

Troubleshooting based on element status

For some of the element states, you may be required to intervene. When you display the element status, you can respond to each of these element states. Table 11–2 shows details on each element status and a recommendation of how to respond to changes in the element status.

Table 11–2 Element status

Status	Meaning	Notes and recommendations
`close failed`	The attempt to close the element failed.	Refer to the `ttGridAdmin dbStatus` command output for information about the failure. You can try `ttGridAdmin dbClose` again.
`closing`	The element is in the process of closing.	Wait, and run the `ttGridAdmin dbStatus` command again to see when the element is closed. You can unload the database when some elements are still closing, but you would have to use the `ttGridAdmin dbUnload -force` command.

Table 11–2 (Cont.) Element status

Status	Meaning	Notes and recommendations
`create failed`	The attempt to create the element failed.	Refer to the `ttGridAdmin dbStatus` output for information about the failure. A common issue is that there are not enough semaphores to create the element or there is something wrong with the directory (incorrect permissions) for the checkpoint files. See "Set the semaphore values" on page 2-7 for details on how to set enough semaphores. You can use the `ttGridAdmin dbCreate` command with the `-instance` *`hostname`*`[.`*`instancename`*`]` option to retry the creation of the element on that data instance. See Example 11–2, "Retrying element creation" for details.
`creating`	The element is being created.	Wait, and run the `ttGridAdmin dbStatus` command again to see when the element is created.
`destroy failed`	The attempt to destroy the element failed.	Refer to the `ttGridAdmin dbStatus` command output for information about the failure. If the element status is `destroy failed`, you can retry the destroy of the element on the data instance with the `ttGridAdmin dbDestroy` command with the `-instance` *`hostname`*`[.`*`instancename`*`]` option. See Example 11–4, "Destroying an evicted element or an element where a destroy failed" for an example.
`destroyed`	The element has been destroyed.	Element no longer exists. **Note:** When the last element of a database is destroyed, no record of the database, including element status, will exist.
`destroying`	The element is being destroyed.	Wait, and run the `ttGridAdmin dbStatus` command again to see when the element is destroyed.
`down`	The data instance where this element is located is not running.	If the data instance is down, the status of an element is down. Try to restart the data instance by using the `instanceExec` command to execute `ttDaemonAdmin -start`, using the `instanceExec` option `-only` *`hostname`*`[.`*`instancename`*`]`. See Example 11–3, "Restart a data instance that is down" and "Recovering when a data instance is down" on page 11-29 for more details on how to manually restart a data instance.
`evicted`	The element was evicted or removed through `ttGridAdmin dbDistribute` and has been removed from the distribution map.	When the element status is `evicted`, destroy the element of the data instance with the `ttGridAdmin dbDestroy` command with the `-instance` *`hostname`*`[.`*`instancename`*`]` option. See Example 11–4, "Destroying an evicted element or an element where a destroy failed" for more information.

Table 11–2 (Cont.) Element status

Status	Meaning	Notes and recommendations
`evicted (loaded)`	The element was evicted or removed through `ttGridAdmin dbDistribute` but removal from the distribution map has not yet begun.	Wait, and run `ttGridAdmin dbStatus` command again to see when the element is unloaded. When the element status is `evicted`, destroy the element with the `ttGridAdmin dbDestroy` command with the `-instance` *hostname*`[.`*instancename*`]` option. See Example 11–4, "Destroying an evicted element or an element where a destroy failed" for more information.
`evicted (unloading)`	The element was evicted or removed through `ttGridAdmin dbDistribute` and is being removed from the distribution map.	Wait, and run `ttGridAdmin dbStatus` command again to see when the element is unloaded. When the element status is `evicted`, destroy the element of the data instance with the `ttGridAdmin dbDestroy` command with the `-instance` *hostname*`[.`*instancename*`]` option. See Example 11–4, "Destroying an evicted element or an element where a destroy failed" for more information.
`load failed`	The attempt to load the element failed.	Refer to the `ttGridAdmin dbStatus` command output for information about the failure. You can try again to load the element with the `ttGridAdmin dbLoad` command with the `-instance` *hostname*`[.`*instancename*`]` option.
`loaded`	The element is loaded.	Element is loaded and can now be opened. You can confirm if the element is in the distribution map with the `ttGridAdmin dbStatus -replicaset` command.
`loading`	The element is being loaded.	Wait, and run the `ttGridAdmin dbStatus` command again to see when the element is loaded.
`opened`	The element is open.	Normal status for a functioning element. Database connections are possible through the element.
`open failed`	The attempt to open the element failed.	Refer to the `ttGridAdmin dbStatus` command output for information about the failure. You can try `ttGridAdmin dbOpen` again.
`opening`	The element is in the process of opening.	Wait, and run `ttGridAdmin dbStatus` command again to see when the element is open.
`uncreated`	The element should be created, but creation has not yet started.	Wait, and run the `ttGridAdmin dbStatus` command again to see when creation begins (status `creating`).
`unloaded`	The element has been unloaded.	Database is ready to be loaded again (`ttGridAdmin dbLoad`) or destroyed (`ttGridAdmin dbDestroy`). You can run the `ttGridAdmin dbLoad` command to reload the database.

Table 11–2 (Cont.) Element status

Status	Meaning	Notes and recommendations
`unloading`	The element is being unloaded.	Wait, and run the `ttGridAdmin dbStatus` command again to see when the element is unloaded.
`waiting for seed`	The element will be loaded, but not until after the other element in its replica set is loaded.	Note the status of the other element in the replica set. ■ If the status of the other element is `loading`, then this element will load as soon as the status of the other element is `loaded`. ■ If the status of the other element is `load failed`, then address that problem. See the entry for `load failed` above. ■ If the status of the other element is `down`, then the element cannot recover. Restart the data instance as indicated within the element `down` status information in this table. ■ If both elements in the replica set are in the `waiting for seed` state, then the only way to recover the replica set is to either: - Reload the database with the `ttGridAdmin dbLoad` command. See "Database recovery" on page 11-30 for details. - If a reload of the database does not recover the elements and if your `Durability=0`, then you may need to evict the replica set, unload and reload the database with the `ttGridAdmin dbDistribute -evict`, `unLoad` and `dbLoad` commands. See "Recovering a failed replica set when Durability=0" on page 11-18 for details.

Note: The notes and recommendations column often refers to `ttGridAdmin` commands. For more information on these commands within the *Oracle TimesTen In-Memory Database Reference*, see "Monitor the status of a database (dbStatus)" for `ttGridAdmin dbStatus`, "Create a database (dbCreate)" for `ttGridAdmin dbCreate`, "Open a database (dbOpen)" for `ttGridAdmin dbOpen`, "Load a database into memory (dbLoad)" for `ttGridAdmin dbLoad`, "Unload a database (dbUnload)" for `ttGridAdmin dbUnload`, "Close a database (dbClose)" for `ttGridAdmin dbClose`, "Destroy a database (dbDestroy)" for `ttGridAdmin dbDestroy`, and "Execute a command or script on grid instances (instanceExec)" for `ttGridAdmin instanceExec`.

The following sections demonstrate how to respond with different scenarios where a single element in the replica set has failed:

- Retrying element creation
- Restart a data instance that is down

- Destroying an evicted element or an element where a destroy failed

Example 11–2 Retrying element creation

If the creation of the element failed, then retry the creation of the element with the `ttGridAdmin dbCreate -instance` command on the same data instance where the element should exist.

```
% ttGridAdmin dbCreate database1 -instance host3
Database database1 creation started
```

Example 11–3 Restart a data instance that is down

When a data instance is down, the element within the data instance is down. You restart the daemon of the data instance by using the `ttGridAdmin instanceExec -only` command to execute the `ttDaemonAdmin -start` command. See "Recovering when a data instance is down" on page 11-29 for more details.

```
% ttGridAdmin instanceExec -only host4.instance1 ttDaemonAdmin -start
Overall return code: 0
Commands executed on:
  host4.instance1 rc 0
Return code from host4.instance1: 0
Output from host4.instance1:
TimesTen Daemon (PID: 15491, port: 14000) startup OK.
```

Example 11–4 Destroying an evicted element or an element where a destroy failed

If you evict an element, you still need to destroy the element to free up the file system space used by the element. After which, you may decide to create a new element. See "Unavailability of data when a full replica set is down or fails" on page 11-15 for more details on eviction.

When the element status is `destroy failed` or `evicted`, destroy the element of the data instance with the `ttGridAdmin dbDestroy -instance` command.

```
% ttGridAdmin dbDestroy database1 -instance host3
Database database1 destroy started
```

Recovering a replica set after an element goes down

When $k = 2$, all active elements in the same replica set are transactionally synchronized. Any DML or DDL statements applied to one element in a replica set are also applied to all other elements in the replica set. When one element in the replica set is not up, the other element can continue to execute DML or DDL statements.

- If the failed element recovers, it was unavailable for a time and fell behind transactionally. Before this element can resume its part in the replica set in the grid, it must synchronize its data with the active element of its replica set.
- If the element permanently fails, such as a file system failure, you need to remove that element from the replica set and replace it with another element with the `ttGridAdmin dbDistribute -remove -replaceWith` command. See "Replace an element with another element" on page 8-13 for details.

TimesTen Scaleout automatically re-synchronizes and restores the data on the restored or new element in the replica set with the following methods:

- Log-based catch up: This process transfers the transaction logs from the active element in the replica set and applies transaction records that are missing on the recovering element. This operation applies the DML or DDL statements that occurred while the element was not participating in the replica set.

Transactions that are started while one of the elements of the replica set is down must be replayed when recovering the down element. The log-based catch up process waits for any open transactions to commit or roll back before replaying them from the transaction log. If the down element is in the recovery process for an extended period of time, then there may be an open transaction (on the active element) preventing the completion of the log-based catch up process for the recovering element. Use the `ttXactAdmin` utility to check for open transactions. Resolve any open transactions by either committing or rolling them back.

- Duplicate: TimesTen Scaleout duplicates the active element either to a recovering element or to a new element that replaces a failed element. The duplication operation copies all checkpoint and log files of the active element to the recovering element.

  However, since the active element continues to accept transactions during the duplicate operation, there may be additional transaction log records that are not a part of the copied transaction log files. After completing the duplicate operation, TimesTen Scaleout contacts the active element and performs a log-based catch up operation to bring the new element completely up to date.

Remove and replace a failed element in a replica set

When *k* = 2, if an element cannot be recovered automatically, then you have to investigate what caused the failure. You may discover a problem that can be fixed, such as a drive that needs to be remounted. However, you may discover a problem that cannot be fixed, such as a drive that is completely destroyed. Most permanent, unrecoverable failures are normally related to hardware failures.

- If you can, fix the problem with the host or the data instance and then perform one of the following:
  - Restart the data instance. See "Recovering when a data instance is down" on page 11-29 for directions on how to restart the data instance.
  - Reload the TimesTen database with the `ttGridAdmin dbload` command, which attempts to reload the element.
- If you cannot fix the problem with the host or data instance, then the data on the element may be in a state where it cannot be retrieved. In this case, you must remove the element and replace it with another element. Once replaced, the active element updates the new element with the data for this replica set.

If one of your hosts is encountering multiple errors (even though it has been able to automatically recover), you may decide to replace it with another host that is more reliable.

To replace an element without data loss, execute the `ttGridAdmin dbDistribute -remove -replaceWith` command, which takes the data that exists on the element you want to replace and redistributes to a new element. See "Replace an element with another element" on page 8-13 for more details.

Unavailability of data when a full replica set is down or fails

If all elements in a single replica set are down or failed, the data stored in the down replica set is unavailable. In order to guard against full replica set failure, distribute your elements in a way that reduces the chances of full replica set failure. See

"Assigning hosts to data space groups" on page 1-11 for details on installing data instances on hosts that are physically separated from each other.

The following sections describe the transaction behavior when a replica set is down, how TimesTen Scaleout may recover the replica set, and what you can do if the replica set needs intervention to fully recover.

- Recovering from a down replica set
- Recovering when the replica set has a permanently failed element

Recovering from a down replica set

As described in Table 11–3, if you have a down or failed replica set, the outcome of preserving your data successfully may depend on how you set the `Durability` connection attribute. See "Durability settings" on page 6-3 for more details on `Durability` connection attribute settings.

Table 11–3 Potential for transaction recovery based on Durability value

Durability value	Affect on transactions when a replica set fails
1	Participants synchronously write a prepare-to-commit or commit log record to the transaction log for distributed transactions. This ensures that committed transactions have the best possible chance of being preserved. If a replica set goes down, all transaction log records have been durably committed to the file system and can be recovered by TimesTen Scaleout.
0	Participants asynchronously write prepare-to-commit and commit log records for distributed transactions. If an entire replica set goes down, transaction log records are not guaranteed to be durably committed to the file system. There is a chance for data loss, depending on how the elements within the replica set fail or go down.

The following sections describe what happens with new transactions after a replica set goes down or how the replica set recovers depends on the `Durability` connection attribute value.

- Transaction behavior with a down replica set
- Durably recovering a failed replica set when Durability=1
- Recovering a failed replica set when Durability=0

Transaction behavior with a down replica set

The following list describes what occurs for your transaction when there is a down replica set.

- Transactions with queries that access rows only within active replica sets (and no rows within a down replica set) succeed. Queries that try to access data within a down replica set fail. Your application should retry the transaction when the replica set has recovered.

  A global read with a partial results hint that does not require data from the down replica set succeeds.

  For example, if both elements in replica set 1 failed and the queries within the transaction require data from replica set 1, then the transaction fails. Your application should perform the transaction again.

- Transactions with any DDL statement fail when there is a down replica set as DDL statements require all replica sets to be available. Your application should roll back the transaction.
- Transactions with any DML statements fail if the transaction tries to update at least one row on elements in a down replica set. Your application should roll back the transaction. When `Durability=0`, this scenario may encounter data loss. See "Recovering a failed replica set when Durability=0" on page 11-18 for full details.
- When `Durability=1`, transactions with DML that do not require data from the down replica set succeeds. For example, if both elements in replica set 1 failed, then the transaction succeeds only if any `SELECT`, `INSERT`, `INSERT...SELECT`, `UPDATE` or `DELETE` statements do not depend on data that was stored in replica set 1.

Durably recovering a failed replica set when Durability=1

The following sections describe the process for recovery of a failed replica set when `Durability=1`.

If all elements in the replica set go down, even temporarily, TimesTen Scaleout might be able to automatically recover the full replica set (if the initial issue is resolved) by:

1. Determining and recovering the seed element. The element that failed with the latest changes, known as the seed element, is recovered first. The seed element is recovered to the latest transaction in the checkpoint and transaction log files.
2. After recovery of the element is complete, TimesTen Scaleout checks for in-doubt transactions.

   When an element is loaded from the file system (from checkpoint and transaction log files) to recover after a transient failure or unexpected termination, any two-phase commit transactions that were prepared, but not committed, are left pending. This is referred to as an *in-doubt transaction*. When a transaction has been interrupted, there may be a doubt of whether the entire transaction was committed with the two-phase commit protocol.

   – If there are no in-doubt transactions, operation proceeds as normal.

   – If there are in-doubt transactions, normal processing that includes this replica set does not continue until all in-doubt transactions are resolved. If there are any in-doubt transactions, TimesTen Scaleout checks the transaction log to determine whether the transaction committed or was prepared to commit on any of the participants. The transaction log records contain information about other participants in the transaction. See Table 11–4 for how TimesTen Scaleout resolves in-doubt transactions.

     If an element fails during this process and then comes back up after the transaction commits or rolls back, the element recovers itself by requesting the result of the other participating elements.
3. After the seed element is recovered, the other element in the replica set is recovered from the seed element using the duplicate and log-based catch up methods. See "Recovering a replica set after an element goes down" on page 11-14 for details on the duplicate and log-based catch up methods.

Table 11–4 How TimesTen Scaleout resolves an in-doubt transaction

Failure	Action
At least one participant received the commit log record; all other participants at least receive the prepare-to-commit log record.	The transaction commits on all participants
All participants in the transaction received the prepare-to-commit log record.	The transaction commits on all participants.
At least one participant did not receive the prepare-to-commit log record.	The transaction manager notifies all participants to undo the prepare-to-commit, which is a prelude to a roll back of the transaction. ■ If the transaction was executed with `autocommit 1`, then the transaction manager rolls back the transaction. ■ If the transaction was executed with `autocommit 0`, then the transaction manager throws an error informing the application that it must roll back the transaction.

However, if you cannot recover the elements in a down replica set, then you may need to either remove and replace one of the elements or evict the entire replica set. See "Recovering when the replica set has a permanently failed element" on page 11-20 for details.

Recovering a failed replica set when Durability=0

The following describes the process for recovery of a failed replica set when `Durability=0`.

If you set `Durability=0`, you are acknowledging that there is a chance of data loss when a replica set fails. However, TimesTen Scaleout attempts to avoid data loss if the elements fail at separate times.

- If only a single element of the replica set fails, then TimesTen Scaleout attempts to switch the remaining element in the replica set (when $k = 2$) into durable mode. That is, in order to limit data loss (which would occur if the remaining element fails when `Durability=0`), TimesTen Scaleout changes the durability behavior of the element as if it was configured with `Durability`=1.

  If TimesTen Scaleout can switch the remaining element in the replica set into durable mode, then the participating element synchronously writes prepare-to-commit log records to the file system for distributed transactions. Then, if this element also fails so that the entire replica set is down, TimesTen Scaleout recovers the replica set from the transaction log records. Thus, no transaction is lost in this scenario and TimesTen Scaleout automatically recovers the replica set as when you have set `Durability=1`. See "Durably recovering a failed replica set when Durability=1" on page 11-17 for details on recovering after the single element is recovered.

- If TimesTen Scaleout cannot switch the replica set into durable mode before the final surviving element fails, then you may encounter data loss depending on whether the replica set encounters a temporary or permanent failure.

  – Temporary replica set failure when elements are non-durable: Since neither element in the replica set synchronously wrote prepare-to-commit log records for distributed transactions that the replica set was involved in before going down, then any transactions that committed after the last successful epoch transaction are lost.

If both elements show the `waiting for seed` status, then there was no switch into durable mode before the replica set went down. If this is the case, epoch recovery is necessary and any transactions committed after latest successful epoch transaction are lost. When the elements in this replica set recover, they may remain in the `waiting for seed` status, since neither element is able to recover with the transaction logs. Instead, you must perform epoch recovery by either recovering or evicting the replica set, followed by unloading and reloading the database. See "Process when replica set fails when in a non-durable state" on page 11-19 for details.

– Permanent replica set failure: If you cannot recover either element in the replica set, you may have to evict these elements. This results in a loss of the data on that replica set. See "Recovering when the replica set has a permanently failed element" on page 11-20 for details.

Process when replica set fails when in a non-durable state

When a replica set goes down and the state is non-durable, transactions may continue to commit into the database until TimesTen Scaleout realizes that the replica set is down. Once TimesTen Scaleout realizes that a replica set is down (after a failed epoch transaction execution), then the database is switched to read-only to minimize the number of lost transactions. During epoch recovery, the database is reloaded to the last successful epoch transaction, effectively losing any transactions that committed after that last successful epoch transaction. In this scenario, the value of the `EpochInterval` connection attribute not only determines the amount of time between the epoch transactions, but also determines the approximate amount of time during which you can lose committed transactions.

Note: The database is set to read-only when the epoch transaction fails due to a down replica set; TimesTen Scaleout does not set the database to read-only if the epoch transaction fails for other reasons.

Figure 11–3 shows the actions across a time span of eight intervals.

Figure 11–3 Durability=0 and a replica set fails

time span	operation during each interval
1	Last common epoch before failure.
2	——
3	Replica set 1 goes down.
4	——
5	Epoch transaction fails. Database becomes read-only.
6	——
7	Database reloads to the last common epoch.
8	Epoch transaction is run.

1. An epoch transaction commits successfully.
2. Transactions may continue after the successful epoch transaction. Any committed transactions after the last successful epoch transaction are lost after epoch recovery as neither element in the down replica set was able to durably flush the transaction logs.

3. Replica set 1 goes down without either element switching to durable mode.

> **Note:** Sequences may be incremented while the replica set is down.

4. Transactions may continue after the replica set goes down if the database has not yet been set to read-only. Any transactions that commit after the last successful epoch transaction are lost after epoch recovery as neither element in the down replica set was able to durably flush the transaction logs.

> **Note:** The behavior of transactions after a replica set goes down depends on the type of statements within the transactions, as described in "Transaction behavior with a down replica set" on page 11-16.

5. The next epoch transaction fails since not all replica sets are up. TimesTen Scaleout informs all data instances that the database is now read-only. All applications will fail when executing a DML, DDL, or commit statements within open transactions. You must roll back each transaction.

> **Note:** The `ttGridAdmin dbStatus` command shows the state of the database, including if it is in read-only or read-write mode.

6. The replica set must be recovered or evicted.
   - Recover the down replica set. If multiple replica sets are down, the database cannot enter read-write mode until all replica sets are recovered or replaced.
   - If you cannot recover either element in the replica set, you may have to evict the replica set, which results in a loss of the data on that replica set. See "Recovering when the replica set has a permanently failed element" on page 11-20 for details.
7. You perform an epoch recovery by unloading and reloading the database to the last successful epoch transaction to recover the database consistently with only a partial data loss. Any transactions that commit after the last successful epoch are lost when the database is unloaded and reloaded to the last successful epoch transaction. See "Load a database into memory (dbLoad)" for information on the `ttGridAdmin dbLoad` command and "Unload a database (dbUnload)" for information on the `ttGridAdmin dbUnload` command.
8. A new epoch transaction is successful. Database is set to read-write. Normal transaction behavior resumes.

> **Note:** If you want to ensure that the data for a transaction is always recovered, you can promote a transaction to be an epoch transaction. See "Epoch transactions" on page 6-3 for more details.

Recovering when the replica set has a permanently failed element

If an element in the replica set or a full replica set is unrecoverable because there has been a permanent failure, then you need to remove the failed element or evict the failed replica set. Permanent failure can occur when a host permanently fails or if all elements in the replica set fail.

- If all elements within a replica set permanently fail, you must evict the entire replica set, which results in the permanent loss of the data on the elements within that replica set.

  When $k = 1$, then the permanent failure of one element is a replica set failure. When $k = 2$, both elements in a replica set must fail in order for the replica set to be considered failed. If $k = 2$ and the replica set permanently fails, you need to evict both elements of the replica set simultaneously.

  Evicting the replica set removes it from the distribution for the grid. However, you cannot evict the replica set if the failed replica set is the only replica set in the database. In this case, save any checkpoint files, transaction log files or daemon log files (if possible) and then destroy and recreate the database.

  When a replica set goes down:

  – If `Durability`=0, the database goes into read-only mode.

  – If `Durability`=1, then all transactions that include the failed replica set are blocked until you evict the failed replica set. However, all transactions that do not involve the failed replica set continue to work as if nothing was wrong.

- If $k = 2$ and only one element of a replica set fails, the active element takes over all of the requests for data until the failed element can be replaced with a new element. Thus, no data is lost with the failure. The active element in the replica set processes the incoming transactions. You can simply remove and replace the failed element with a new element that is duplicated from the active element in the replica set. The active element provides the base for a duplicate for the new element. See "Replace an element with another element" on page 8-13 for details on how to remove and replace a failed element.

> **Note:** If you know about problems that TimesTen Scaleout is not aware of and that a replica set needs to be evicted, you can evict and replace a replica set as needed.

You can evict the replica set from the distribution map for your grid with the `ttGridAdmin dbDistribute -evict` command. Make sure that all pending requests for adding or removing elements are applied before requesting the eviction of a replica set.

You have the following options when you evict a replica set:

- Evict the replica set without replacing it immediately.

  If the data instances and hosts for this replica set have not failed, then you can recreate the replica set using the same data instances. This is a preferred option if there are other databases on the grid and the hosts are fine.

  In this case, you must:

  1. Evict the elements of the failed replica set, while the data instances and hosts are still up.

     When you evict the replica set, the data is lost within this replica set, but the other replica sets in the database continue to function. There is now one fewer replica set in your grid.

  2. Eliminate all checkpoint and transaction logs for the elements within the evicted replica set if you want to add new elements to the distribution map on the same data instances which previously held the evicted elements.

3. Destroy the elements of the evicted replica set, while the data instances and hosts are still up.
4. Optionally, you can replace the evicted replica set with a new replica set either on the same data instances and hosts if they are still viable or on new data instances and hosts. Add the new elements to the distribution map. This restores the grid to its expected configuration.

- Evict the replica set and immediately replace it with a new replica set to restore the grid to its expected configuration.
  1. Create new data instances and hosts to replace the data instances and hosts of the failed replica set.
  2. Evict the elements of the failed replica set, while replacing it with a new replica set. When you evict the replica set, the data is lost within this replica set, but the other replica sets in the database continue to function.

     Use the `ttGridAdmin dbDistribute -evict -replaceWith` command to evict and replace the replica set with a new replica set, where each new element is created on a new data instance and host. The elements of the new replica set are added to the distribution map. However, the remaining data from the other replica sets are not redistributed to include the new replica. Thus, the new replica set remains empty until you insert data.
  3. Destroy the elements of the evicted replica set.

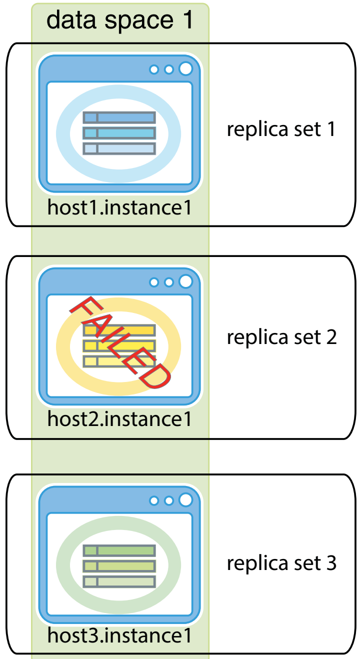
The following sections demonstrate how to evict a failed replica set when you have one or two elements in the replica set:

- Evicting the element in the permanently failed replica set when k = 1
- Evicting all elements in a permanently failed replica set when k = 2
- Maintaining database consistency after an eviction

Evicting the element in the permanently failed replica set when k = 1

The example shown in Figure 11–4 shows a TimesTen database that has been configured with *k* set to 1 with three data instances: `host1.instance1`, `host2.instance1` and `host3.instance1`. The element on the `host2.instance1` data instance fails because of a permanent hardware failure.

Figure 11–4 Grid database where k = 1



The following examples demonstrate the eviction options:

- Example 11–5, "Evict the element to potentially replace at another time"
- Example 11–6, "Evict and replace the data instance without re-distribution"

Example 11–5 Evict the element to potentially replace at another time

If you cannot recover a failed element, you evict the replica set.

The following example:

1. Evicts the replica set for the element on the `host2.instance1` data instance with the `ttGridAdmin dbDistribute -evict` command.
2. Destroys the checkpoint and transaction logs for only this element within the evicted replica set with the `ttGridAdmin dbDestroy -instance` command.

Note: Alternatively, see the instructions in "Remove and replace a failed element in a replica set" on page 11-15 if the data instance or host on which the element exists is not reliable.

```
% ttGridAdmin dbDistribute database1 -evict host2.instance1 -apply
Element host2.instance1 evicted
Distribution map updated

% ttGridAdmin dbDestroy database1 -instance host2.instance1
Database database1 instance host2 destroy started

% ttGridAdmin dbStatus database1 -all
Database database1 summary status as of Thu Feb 22 16:44:15 PST 2018

created,loaded-complete,open
Completely created elements: 2 (of 3)
Completely loaded elements: 2 (of 3)
```

```
Open elements: 2 (of 3)

Database database1 element level status as of Thu Feb 22 16:44:15 PST 2018

Host  Instance  Elem Status    Date/Time of Event  Message
----- --------- ---- --------- ------------------- -------
host1 instance1    1 opened    2018-02-22 16:42:14
host2 instance1    2 destroyed 2018-02-22 16:44:01
host3 instance1    3 opened    2018-02-22 16:42:14

Database database1 Replica Set status as of Thu Feb 22 16:44:15 PST 2018

RS DS Elem Host  Instance  Status Date/Time of Event  Message
-- -- ---- ----- --------- ------ ------------------- -------
 1  1    1 host1 instance1 opened 2018-02-22 16:42:14
 2  1    3 host3 instance1 opened 2018-02-22 16:42:14

Database database1 Data Space Group status as of Thu Feb 22 16:44:15 PST 2018

DS RS Elem Host  Instance  Status Date/Time of Event  Message
-- -- ---- ----- --------- ------ ------------------- -------
 1  1    1 host1 instance1 opened 2018-02-22 16:42:14
    2    3 host3 instance1 opened 2018-02-22 16:42:14
```

This example creates a new element for the replica set as the data instance and host are still viable. Then, adds the new elements to the distribution map.

1. Creates a new element with the `ttGridAdmin dbCreate -instance` command on the same data instance where the previous element existed before its replica set was evicted.
2. Adds the new element into the distribution map with the `ttGridAdmin dbDistribute -add` command.

```
% ttGridAdmin dbCreate database1 -instance host2
Database database1 creation started
% ttGridAdmin dbDistribute database1 -add host2 -apply
Element host2 is added
Distribution map updated
% ttGridAdmin dbStatus database1 -all
Database database1 summary status as of Thu Feb 22 16:53:17 PST 2018

created,loaded-complete,open
Completely created elements: 3 (of 3)
Completely loaded elements: 3 (of 3)

Open elements: 3 (of 3)

Database database1 element level status as of Thu Feb 22 16:53:17 PST 2018

Host  Instance  Elem Status Date/Time of Event  Message
----- --------- ---- ------ ------------------- -------
host1 instance1    1 opened 2018-02-22 16:42:14
host3 instance1    3 opened 2018-02-22 16:42:14
host2 instance1    4 opened 2018-02-22 16:53:14

Database database1 Replica Set status as of Thu Feb 22 16:53:17 PST 2018

RS DS Elem Host  Instance  Status Date/Time of Event  Message
-- -- ---- ----- --------- ------ ------------------- -------
```

```
 1  1    1 host1 instance1 opened 2018-02-22 16:42:14
 2  1    3 host3 instance1 opened 2018-02-22 16:42:14
 3  1    4 host2 instance1 opened 2018-02-22 16:53:14

Database database1 Data Space Group status as of Thu Feb 22 16:53:17 PST 2018

DS RS Elem Host  Instance  Status Date/Time of Event  Message
-- -- ---- ----- --------- ------ ------------------- -------
 1  1    1 host1 instance1 opened 2018-02-22 16:42:14
    2    3 host3 instance1 opened 2018-02-22 16:42:14
    3    4 host2 instance1 opened 2018-02-22 16:53:14
```

Example 11–6 Evict and replace the data instance without re-distribution

To recover the initial capacity with the same number of replica sets as you started with for the database, evict and replace the evicted element using the `ttGridAdmin dbDistribute -evict -replaceWith` command.

The following example:

1. Creates a new host (identified as `host4`), installation, data instance and element.
2. Evicts the replica set that contains the failed element on the `host2.instance1` data instance and replaces the evicted element with the element on the `host4.instance1` data instance using the `ttGridAdmin dbDistribute -evict -replaceWith` command.

   The data that exists on the elements on the `host1.instance1` and `host3.instance1` data instances is not redistributed to the new element on the `host4.instance1` data instance. The element on the `host4.instance1` data instance is empty.
3. Destroys the element on the `host2.instance1` data instance with the `ttGridAdmin dbDestroy -instance` command.

```
% ttGridAdmin hostCreate host4 -address myhost.example.com -dataspacegroup 1
Host host4 created in Model
% ttGridAdmin installationCreate -host host4 -location
/timesten/host4/installation1
Installation installation1 on Host host4 created in Model
% ttGridAdmin instanceCreate -host host4 -location /timesten/host4
Instance instance1 on Host host4 created in Model
% ttGridAdmin modelApply
Copying Model.......................................................OK
Exporting Model Version 2...........................................OK
Marking objects 'Pending Deletion'..................................OK
Deleting any Hosts that are no longer in use........................OK
Verifying Installations.............................................OK
Creating any missing Installations..................................OK
Creating any missing Instances......................................OK
Adding new Objects to Grid State....................................OK
Configuring grid authentication.....................................OK
Pushing new configuration files to each Instance....................OK
Making Model Version 2 current......................................OK
Making Model Version 3 writable.....................................OK
Checking ssh connectivity of new Instances..........................OK
Starting new data instances.........................................OK
ttGridAdmin modelApply complete
% ttGridAdmin dbDistribute database1 -evict host2.instance1
 -replaceWith host4.instance1 -apply
Element host2.instance1 evicted
Distribution map updated
```

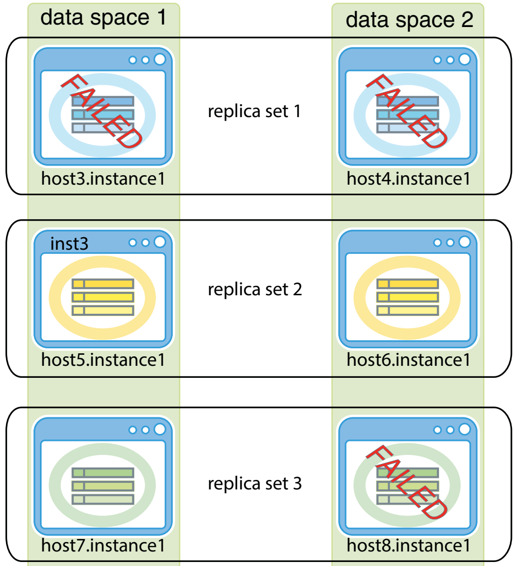
```
% ttGridAdmin dbDestroy database1 -instance host2
Database database1 instance host2 destroy started
% ttGridAdmin dbStatus database1 -all
Database database1 summary status as of Thu Feb 22 17:04:21 PST 2018

created,loaded-complete,open
Completely created elements: 3 (of 4)
Completely loaded elements: 3 (of 4)

Open elements: 3 (of 4)

Database database1 element level status as of Thu Feb 22 17:04:21 PST 2018

Host  Instance  Elem Status    Date/Time of Event  Message
----- --------- ---- --------- ------------------- -------
host1 instance1    1 opened    2018-02-22 16:42:14
host3 instance1    3 opened    2018-02-22 16:42:14
host2 instance1    4 destroyed 2018-02-22 17:04:11
host4 instance1    5 opened    2018-02-22 17:03:18

Database database1 Replica Set status as of Thu Feb 22 17:04:21 PST 2018

RS DS Elem Host  Instance  Status Date/Time of Event  Message
-- -- ---- ----- --------- ------ ------------------- -------
 1  1    1 host1 instance1 opened 2018-02-22 16:42:14
 2  1    3 host3 instance1 opened 2018-02-22 16:42:14
 3  1    5 host4 instance1 opened 2018-02-22 17:03:18

Database database1 Data Space Group status as of Thu Feb 22 17:04:21 PST 2018

DS RS Elem Host  Instance  Status Date/Time of Event  Message
-- -- ---- ----- --------- ------ ------------------- -------
 1  1    1 host1 instance1 opened 2018-02-22 16:42:14
    2    3 host3 instance1 opened 2018-02-22 16:42:14
    3    5 host4 instance1 opened 2018-02-22 17:03:18
```

Evicting all elements in a permanently failed replica set when k = 2

If *k* = 2 and the replica set permanently fails, then you need to evict both elements of the replica set simultaneously.

Figure 11–5 shows where replica set 1 fails.

Figure 11–5 Failed replica set



For the example shown in Figure 11–5, replica set 1 contains elements that exist on both the `host3.instance1` and `host4.instance1` data instances. The replica set fails in an unrepairable way. When you execute the `ttGridAdmin dbDistribute` command to evict the replica set, specify the data instances of both elements in the replica set that are being evicted.

```
% ttGridAdmin dbDistribute database1 -evict host3.instance1
 -evict host4.instance1 -apply
Element host3.instance1 evicted
Element host4.instance1 evicted
Distribution map updated
```

Replacing the replica set with new elements with no data redistribution If you cannot recover either element in the replica set, you evict both elements in the replica set simultaneously. To recover the initial capacity with the same number of replica sets as you started with for the database, evict and replace the evicted elements in the failed replica set using the `ttGridAdmin dbDistribute -evict -replaceWith` command.

The following example:

1. Creates new elements in the `host9.instance1` and `host10.instance1` data instances.
2. Evicts the replica set with the failed elements on the `host3.instance1` and `host4.instance1` data instances, replacing them with new elements in the `host9.instance1` and `host10.instance1` data instances.

   The data that exists on the elements in the active replica sets is not redistributed to include the new elements on the `host9.instance1` and `host10.instance1` data instances. The elements on the `host9.instance1` and `host10.instance1` data instances are empty.
3. Destroys the elements on the `host3.instance1` and `host4.instance1` data instances with the `ttGridAdmin dbDestroy -instance` command.

   The new replica set is now listed as replica set 1 with the elements from the replaced elements located in the `host9.instance1` and `host10.instance1` data instances.

```
% ttGridAdmin hostCreate host9 -internalAddress int-host9 -externalAddress
 ext-host9.example.com -like host3 -cascade
Host host9 created in Model
Installation installation1 created in Model
Instance instance1 created in Model
% ttGridAdmin hostCreate host10 -internalAddress int-host10 -externalAddress
 ext-host10.example.com -like host4 -cascade
Host host10 created in Model
Installation installation1 created in Model
Instance instance1 created in Model
% ttGridAdmin dbDistribute database1 -evict host3.instance1
 -replaceWith host9.instance1 -evict host4.instance1
 -replaceWith host10.instance1 -apply
Element host3.instance1 evicted
Element host4.instance1 evicted
Distribution map updated
% ttGridAdmin dbStatus database1 -all
Database database1 summary status as of Fri Feb 23 10:22:57 PST 2018

created,loaded-complete,open
Completely created elements: 8 (of 8)
Completely loaded elements: 6 (of 8)
Completely created replica sets: 3 (of 3)
Completely loaded replica sets: 3 (of 3)

Open elements: 6 (of 8)

Database database1 element level status as of Fri Feb 23 10:22:57 PST 2018

Host   Instance  Elem Status  Date/Time of Event  Message
------ --------- ---- ------- ------------------- -------
 host3 instance1    1 evicted 2018-02-23 10:22:28
 host4 instance1    2 evicted 2018-02-23 10:22:28
 host5 instance1    3 opened  2018-02-23 07:28:23
 host6 instance1    4 opened  2018-02-23 07:28:23
 host7 instance1    5 opened  2018-02-23 07:28:23
 host8 instance1    6 opened  2018-02-23 07:28:23
host10 instance1    7 opened  2018-02-23 10:22:27
 host9 instance1    8 opened  2018-02-23 10:22:27

Database database1 Replica Set status as of Fri Feb 23 10:22:57 PST 2018

RS DS Elem Host   Instance  Status Date/Time of Event  Message
-- -- ---- ------ --------- ------ ------------------- -------
 1  1    8 host9  instance1 opened 2018-02-23 10:22:27
    2    7 host10 instance1 opened 2018-02-23 10:22:27
 2  1    3 host5  instance1 opened 2018-02-23 07:28:23
    2    4 host6  instance1 opened 2018-02-23 07:28:23
 3  1    5 host7  instance1 opened 2018-02-23 07:28:23
    2    6 host8  instance1 opened 2018-02-23 07:28:23

Database database1 Data Space Group status as of Fri Feb 23 10:22:57 PST 2018

DS RS Elem Host   Instance  Status Date/Time of Event  Message
-- -- ---- ------ --------- ------ ------------------- -------
 1  1    8 host9  instance1 opened 2018-02-23 10:22:27
    2    3 host5  instance1 opened 2018-02-23 07:28:23
    3    5 host7  instance1 opened 2018-02-23 07:28:23
 2  1    7 host10 instance1 opened 2018-02-23 10:22:27
    2    4 host6  instance1 opened 2018-02-23 07:28:23
```

```
    3    6 host8  instance1 opened 2018-02-23 07:28:23

% ttGridAdmin dbDestroy database1 -instance host3
Database database1 instance host3 destroy started
% ttGridAdmin dbDestroy database1 -instance host4
Database database1 instance host4 destroy started
```

Maintaining database consistency after an eviction

Eviction of an entire replica set results in data loss, which can leave the database in an inconsistent state. For example, if the parent records were stored in an evicted replica set, then any child rows on other elements in a different replica set are in a table without a corresponding foreign key parent.

To ensure that you maintain database consistency after an eviction, fix all foreign key references by performing one of the following steps:

- Delete any child row that does not have a corresponding parent.
- Drop the foreign key constraint for any child row that does not have a corresponding parent.

Recovering when a data instance is down

If the error is a hardware error involving the host, then fix the problem with the host and reload the data instance with the `ttGridAdmin dbLoad` command. During reload, TimesTen Scaleout attempts to recover the element within that data instance.

If a data instance is down, you should restart it. If a data instance is not running, then all of the elements that the data instance manages are down.

The `ttGridAdmin dbStatus -element` command shows if a data instance (and thus its element) is considered down.

```
% ttGridAdmin dbStatus database1 -element

Database database1 element level status as of Wed Mar 8 14:07:11 PST 2017

Host  Instance  Elem Status Date/Time of Event  Message
----- --------- ---- ------ ------------------- -------
host3 instance1    1 opened 2017-03-08 13:58:06
host4 instance1    2 down
host5 instance1    3 opened 2017-03-08 13:58:06
host6 instance1    4 opened 2017-03-08 13:58:09
host7 instance1    5 opened 2017-03-08 13:58:09
host8 instance1    6 opened 2017-03-08 13:58:09
```

When a data instance is down (due to a hardware or software failure), all communication channels to its managed elements are shut down and no new connections are allowed to access these elements until all the data instance is restored and the element that it manages is recovered.

If the data instance is down, you restart it by restarting its TimesTen daemon. Once restarted, the data instance connects to a ZooKeeper server. If it does not immediately connect, it continues to try to connect to a ZooKeeper server. After connection, the data instance loads its element.

> **Note:** If the data instance fails to connect to any ZooKeeper server, it may be in an unending loop as it continues to try to connect.

You can manually restart the daemon for that data instance by using the `instanceExec` command to execute the TimesTen `ttDaemonAdmin -start` command, using the `instanceExec` command options of `-only` *`hostname`*`[.`*`instancename`*`]`.

```
% ttGridAdmin instanceExec -only host4.instance1 ttDaemonAdmin -start
Overall return code: 0
Commands executed on:
  host4.instance1 rc 0
Return code from host4.instance1: 0
Output from host4.instance1:
TimesTen Daemon (PID: 15491, port: 14000) startup OK.
```

For more information, see "Execute a command or script on grid instances (instanceExec)" in the *Oracle TimesTen In-Memory Database Reference* or "ttDaemonAdmin" in the *Oracle TimesTen In-Memory Database Reference*.

If you know what caused the error that caused the data instance to fail, then reload the database with the `ttGridAdmin dbLoad` command after you fix the problem.

```
% ttGridAdmin dbLoad database1
```

You can verify the results with the `ttGridAdmin dbStatus` command.

Database recovery

You reload the database to initiate database recovery when either all of the data instances are down or both elements in a replica set show the `waiting for seed` state.

To reload the database:

1. Run the `ttGridAdmin dbStatus` command to see the status of all elements within their respective replica sets.
2. Resolve any issues with the elements of the database, as denoted by each element status, as described in Table 11–2, " Element status".
3. Execute the `ttGridAdmin dbload` command to reload your database, as described in "Reloading a database into memory" on page 5-30.

> **Note:** If an element of a replica set shows the `waiting for seed` status, but the seed element does not recover, then evaluate the host and data instance for that element to see if you need to intervene on either a hardware or software error.
>
> If the seed element still does not recover after reloading the database, then evict the down replica set. See "Recovering when the replica set has a permanently failed element" on page 11-20 for details. If `Durability=0`, then evict the replica set and then unload and reload the database to perform epoch recovery. See "Recovering a failed replica set when Durability=0" on page 11-18 for details.

Client connection failover

When constructing a highly available system, you want to ensure that:

- Client application connections are automatically routed to an active data instance for that database.
- If an existing client connection to a data instance fails, the client is automatically reconnected to another active data instance in the database.

- If the data instance to which a client is connected fails, then that client is automatically reconnected to another active data instance in the database.

> **Note:** See "Connecting to a database" on page 5-7 for details on how a client connects to a data instance in a grid.

By default, if a connection fails, then the client automatically attempts to reconnect to another data instance (if possible). Consider the following details on how to prepare for and respond to a connection failure:

- The `TTC_REDIRECT` client connection attribute defines how a client is redirected. By default, `TTC_REDIRECT` is set to 1 for automatic redirection. If set to 0 and the initial connection attempt to the desired data instance fails, then an error is returned and there are no further connection attempts. See "TTC_REDIRECT" in the *Oracle TimesTen In-Memory Database Reference* for more details.
- The `TTC_NoReconnectOnFailover` client connection attribute defines whether TimesTen should reconnect after a failover. The default is 0, which indicates that TimesTen should attempt to reconnect. Setting this to 1 specifies that TimesTen performs typical client failover, but without reconnecting. This is useful where an application does its own connection pooling or attempts to reconnect to the database on its own after failover. See "TTC_NoReconnectOnFailover" in the *Oracle TimesTen In-Memory Database Reference* for more details.
- Most connection failures tend to be software failures. Reconnecting to another data instance takes some time during which the connection is not available until the client failover process is completed. Any attempt to use the connection during the client failover processing time generates a native error. See "JDBC support for automatic client failover" in the *Oracle TimesTen In-Memory Database Java Developer's Guide* or "Using automatic client failover in your application" in the *Oracle TimesTen In-Memory Database C Developer's Guide* for the native errors that can be received.
- If you receive a native error in response to an operation within your application, your application should place all recovery actions within a loop with a short delay before each subsequent attempt, where the total number of attempts is limited. If you do not limit the number of attempts, then the application may appear to hang if the client failover process does not complete successfully. See "Application action in the event of failover" in the *Oracle TimesTen In-Memory Database Java Developer's Guide* or "Application action in the event of failover" in the *Oracle TimesTen In-Memory Database C Developer's Guide* for an example on how to write a retry block within your application for automatic client failover.

Configuring TCP keep-alive parameters

One of the ways that a client connection can fail is with a network failure, such as disconnecting a cable or a host that is hanging or crashing. When the client connection is lost, then client connection failover is initiated. However, when a TCP connection is started, you can configure the TCP keep-alive parameters for the connection to ensure reliable and rapid detection of connection failures.

Note: You can also detect that there is a problem with the connection by setting the `TTC_Timeout` attribute, which sets a maximum time limit for a network operation that is completed by using the TimesTen client and server. The `TTC_Timeout` attribute also determines the maximum number of seconds a TimesTen client application waits for the result from the corresponding TimesTen Server process before timing out.

TimesTen Scaleout recommends configuring the TCP keep-alive parameters for determining a failed TCP connection in addition to the `TTC_TIMEOUT` attribute, as some database operations may unexpectedly take longer than the value set for the `TTC_TIMEOUT` attribute.

Refer to "TTC_Timeout" in *Oracle TimesTen In-Memory Database Reference* for more information about that attribute.

You can control the per connection keep-alive settings with the following parameters:

- `TTC_TCP_KEEPALIVE_TIME_MS`: The duration time (in milliseconds) between the last data packet sent and the first probe. The default is 10000 milliseconds.

Note: The Linux client platform converts this value to seconds by truncating the last three digits off of the value of `TTC_TCP_KEEPALIVE_TIME_MS`. Thus, a setting of 2500 milliseconds becomes 2 seconds, instead of 2.5 seconds.

- `TTC_TCP_KEEPALIVE_INTVL_MS`: The time interval (in milliseconds) between subsequential probes. The default is 10000 milliseconds.
- `TTC_TCP_KEEPALIVE_PROBES`: The number of unacknowledged probes to send before considering the connection as failed and notifying the client. The default is set to 2 unacknowledged probes.

If you keep the default settings, then TimesTen Scaleout sends the first probe after 10 seconds (the `TTC_TCP_KEEPALIVE_TIME_MS` setting).

- If there is a response, then the connection is alive and the `TTC_TCP_KEEPALIVE_TIME_MS` timer is reset.
- If there is no response, then TimesTen Scaleout sends another probe after this initial probe at 10 second intervals (the `TTC_TCP_KEEPALIVE_INTVL_MS` setting). If no response is received after 2 successive probes, then this connection is aborted and TimesTen Scaleout redirects the connection to another data instance.

For example, you could modify the TCP keep alive settings in the client/server connectable to have a shorter wait time for the initial probe of 50000 milliseconds, and to check for a connection every 20000 milliseconds for a maximum number of 3 times as follows:

```
TTC_TCP_KEEPALIVE_TIME_MS=50000
TTC_TCP_KEEPALIVE_INTVL_MS=20000
TTC_TCP_KEEPALIVE_PROBES=3
```

See "TTC_TCP_KEEPALIVE_TIME_MS", "TTC_TCP_KEEPALIVE_INTVL_MS", and "TTC_TCP_KEEPALIVE_PROBES" in the *Oracle TimesTen In-Memory Database Reference* for more information on these connection attributes.

Managing failover for the management instances

You conduct all management activity from a single management instance, called the active management instance. However, it is highly recommended that you configure two management instances, where the standby management instance is available in case the active management instance goes down or fails.

- If you only have a single management instance and it goes down, the databases remain operational. However, most management operations are unavailable until the management instance is restored.
- If you configure both the active and standby management instances in your grid and only the active management instance is alive, then you can configure and manage the entire grid from this one management instance.

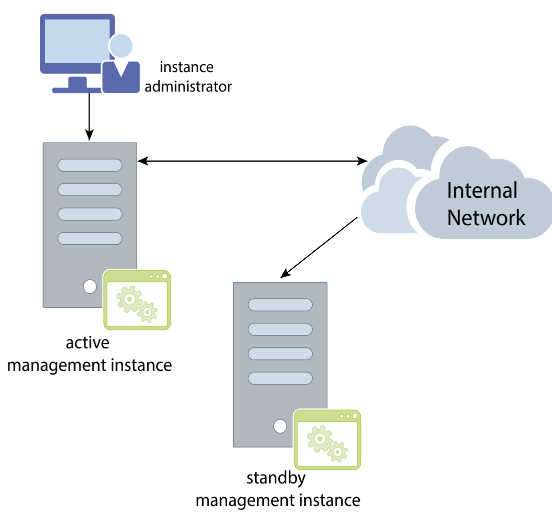
If both management instances are down, then:

- You can still access all databases in the grid. However, since all management actions are requested through the active management instance, you cannot manage your grid until the active management instance is restored.
- If data instances or their elements in the grid go down or fail, they cannot recover, restart or rejoin the grid until the active management instance is restored.

Note: You cannot add a third management instance.

As shown in Figure 11–6, all management information used by the active management instance is automatically replicated to the standby management instance. Thus, if the active management instance goes down or fails, you can promote the standby management instance to become the new active management instance through which you continue to manage the grid.

Figure 11–6 Active standby configuration for management instances



The following sections describes how you can manage the management instances:

- Status for management instances
- Starting, stopping and switching management instances
- Active management instance failure

- Standby management instance failure
- Both management instances fail

Status for management instances

You use the `ttGridAdmin mgmtExamine` command for both the status for the management instances and to see if there are any issues that need to be resolved. This command recommends any corrective actions you can execute to fix any open issues, if necessary.

The following example shows both management instances working:

```
% ttGridAdmin mgmtExamine
Both active and standby management instances are up. No action required.

Host  Instance  Reachable RepRole(Self) Role(Self) Seq RepAgent RepActive
------------------------------------------------------------------------
host1 instance1 Yes       Active        Active     598 Up       Yes
host2 instance1 Yes       Standby       Standby    598 Up       No
```

If one of the management instances goes down or fails, the output shows that the management instance role is `Unknown` and a message states that its replication agent is down. The output provides recommended commands to restart the management instance.

```
% ttGridAdmin mgmtExamine
Active management instance is up, but standby is down

Host  Instance  Reachable RepRole(Self) Role(Self) Seq RepAgent RepActive  Message
----- --------- --------- ------------- ---------- --- -------- --------- --------
host1 instance1 Yes       Active        Active     600 Up       No
host2 instance1 No        Unknown       Unknown        Down     No        Management
 database is not available

Recommended commands:
ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x host2.example.com
 /timesten/host2/instance1/bin/ttenv ttGridAdmin mgmtStandbyStart
```

For each management instance displayed:

- **Host** and **Instance** show the name of the management instance and the name of the host where it is located.
- **Reachable** indicates whether the command was successful in reaching the management instance to determine its state.
- **RepRole(Self)** indicates the recorded role, if any, known by the replication agents for replicating data between management instances. While **Role(Self)** indicates the recorded role known within the database for the management instances. Both of these should show the same role. If the roles are different, the `ttGridAdmin mgmtExamine` command will try to determine the commands that would rectify the error.
- **Seq** is the sequence number of the most recent change on the management instance. If the `Seq` values are the same, then the two management instances are synchronized; otherwise, the one with the larger `Seq` value has the more recent data.
- **RepAgent** indicates whether a replication agent is running on each management instance.

- **RepActive** indicates whether changes by the `ttGridAdmin mgmtStatus` command, which is invoked internally by the `ttGridAdmin mgmtExamine` command, to management data on the management instance were successful.
- **Message** provides any further information about the management instance.

See "Examine management instances (mgmtExamine)" in the *Oracle TimesTen In-Memory Database Reference* for more details.

Starting, stopping and switching management instances

Most `ttGridAdmin` commands are executed through the active management instance. However, when you manage recovery for an active management instance, you may be required to execute `ttGridAdmin` commands on the standby management instance.

When starting, stopping, or promoting a standby management instance:

- You can execute the `ttGridAdmin mgmtStandbyStop` command on either management instance. The grid knows where the standby management instance is and stops it.
- You must execute the `ttGridAdmin mgmtStandbyStart` command on the management instance that you wish to become the standby management instance. The `ttGridAdmin mgmtStandbyStart` command assumes that you want the current instance to become the standby management instance.
- If the active management instance is down, you must execute the `ttGridAdmin mgmtActiveSwitch` command on the standby management instance to promote it to be the active management instance.

For those commands that require you to execute commands on the standby management instance, remember to set the environment with the `ttenv` script (as described in "Creating the initial management instance" on page 4-2) after you log onto the host and before you execute the `ttGridAdmin` utility.

Active management instance failure

You should re-activate an active management instance after a failure as soon as possible to make sure that everything continues to run as expected.

- Single management instance fails
- Active management instance fails

Single management instance fails

While it is not recommended, you can manage the grid with a single active management instance with no standby management instance. If the single active management instance fails and recovers, re-activate the active management instance as follows:

1. Verify that there is only one management instance acting as the active management instance and that it has failed with the `ttGridAdmin mgmtExamine` command:

```
% ttGridAdmin mgmtExamine
The only defined management instance is down. Start it.
Recommendation: define a second management instance

Host Instance Reachable RepRole(Self) Role(Self) Seq RepAgent RepActive
-------------------------------------------------------------------------
host1 instance1 No      Unknown       Unknown    Down     No
```

```
Recommended commands:
ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host1.example.com /timesten/host1/instance1/bin/ttenv ttDaemonAdmin -start
```

2. After determining the reason for the failure and resolving that issue, execute the `ttGridAdmin mgmtActiveStart` command to re-activate the active management instance.

```
% ttGridAdmin mgmtActiveStart
This management instance is now the active
```

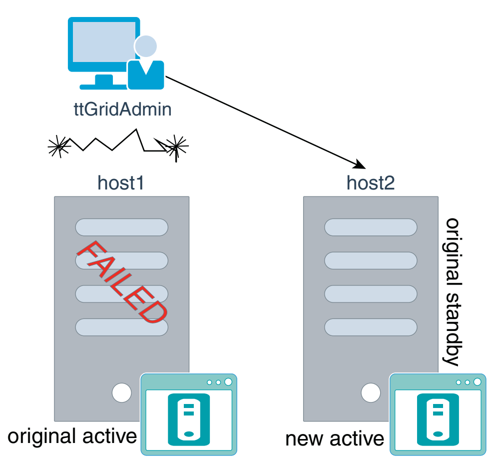
3. Re-execute the `ttGridAdmin mgmtExamine` command to verify that the active management instance is up. Follow any commands it displays if the management instance is not up.

Active management instance fails

If the active management instance fails, then you can no longer execute `ttGridAdmin` commands on it.

- Promote the standby management instance on the `host2` host to be the new active management instance.
- Create a new standby management instance by either:
  - Recovering the failed management instance on `host1` up as the new standby management instance. This causes the new active management instance to replicate all management information to the new standby management instance.
  - Deleting the failed active management instance if the failed management instance has permanently failed, then creating a new standby management instance.

Figure 11–7 Switch from a failed active



For example, your environment has two management instances where the active management instance is on `host1` and the standby management instance is on `host2`. Then, if the active management instance on `host1` fails, then you can no longer execute `ttGridAdmin` commands on it. As shown in Figure 11–7, you must promote the standby management instance on `host2` to become the new active management instance.

1. Log in to the host2 host on which the standby management instance exists and set the environment with the ttenv script (as described in "Creating the initial management instance" on page 4-2) on the host with the standby management instance.

2. Execute the ttGridAdmin mgmtActiveSwitch command on the standby management instance. TimesTen promotes the standby management instance into the new active management instance. You can now continue to manage your grid with the new active management instance.

```
% ttGridAdmin mgmtActiveSwitch
This is now the active management instance
```

3. Verify that the old standby management instance is now the new active management instance with the ttGridAdmin mgmtExamine command:

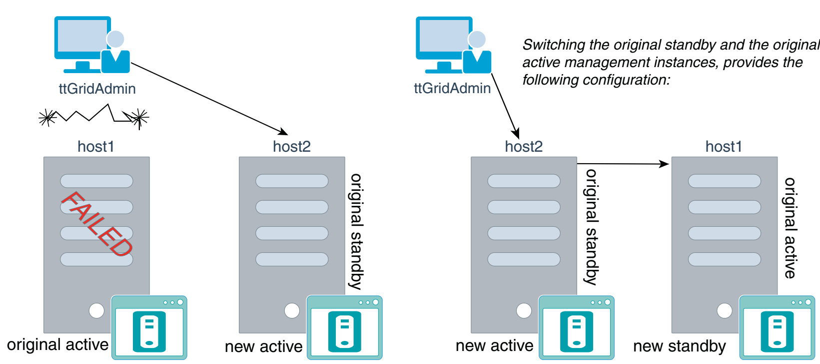
```
% ttGridAdmin mgmtExamine
Active management instance is up, but standby is down

Host Instance Reachable RepRole(Self) Role(Self) Seq RepAgent RepActive
-------------------------------------------------------------------------
host2 instance1 Yes     Active        Active     622 Up       Yes
host1 instance1 No      Unknown       Unknown        Down     No
Management database is not available

Recommended commands:
ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host1.example.com /timesten/host1/instance1/bin/ttenv ttGridAdmin
mgmtStandbyStart
```

Once the new active management instance is processing requests, ensure that a new standby management instance is created by one of the following methods:

- Failed management instance can be recovered
- Failed management instance encounters a permanent failure

Failed management instance can be recovered If the failed active management instance can be recovered, you need to perform the following tasks:

Figure 11–8 The failed management instance can be recovered



1. If you can recover the failed management instance, as shown in Figure 11–8, then bring back up the failed host on which the old active management instance existed. Then, execute the ttGridAdmin mgmtStandbyStart command on this host, which re-initiates the management instance as the new standby management instance. It also re-creates the active standby configuration between the new active

and standby management instances and replicates all management information on the active management instance to the standby management instance.

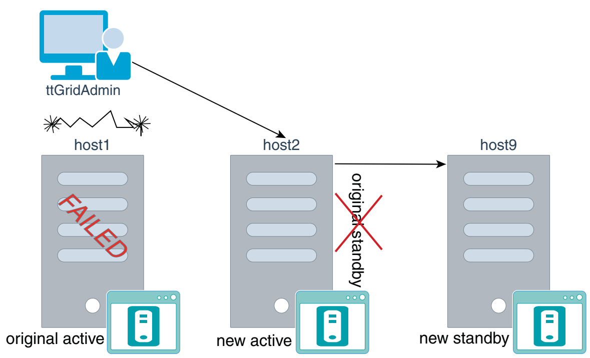
```
% ttGridAdmin mgmtStandbyStart
Standby management instance started
```

2. Verify that the active and standby management instances are as expected in their new roles with the `ttGridAdmin mgmtExamine` command:

```
% ttGridAdmin mgmtExamine
Both active and standby management instances are up. No action required.

Host Instance Reachable RepRole(Self) Role(Self) Seq RepAgent RepActive
-------------------------------------------------------------------------
host2 instance1 Yes     Active        Active     603 Up       Yes
host1 instance1 Yes     Standby       Standby    603 Up       No
```

Failed management instance encounters a permanent failure If the failed active management instance has failed permanently, you need to perform the following tasks:

Figure 11–9 The active management instance fails permanently



1. Remove the permanently failed active management instance from the model with the `ttGridAdmin instanceDelete` command.

```
% ttGridAdmin instanceDelete host1.instance1
Instance instance1 on Host host1 deleted from Model
```

> **Note:** If there are no other instances on the host where the failed active management instance existed, you may want to delete the host and the installation.

2. Add a new standby management instance with its supporting host and installation to the model.

```
% ttGridAdmin hostCreate host9 -address host9.example.com
Host host9 created in Model
% ttGridAdmin installationCreate -host host9 -location
 /timesten/host9/installation1
Installation installation1 on Host host9 created in Model
% ttGridAdmin instanceCreate -host host9 -location /timesten/host9
 -type management
Instance instance1 on Host host9 created in Model
```

3. Apply the configuration changes to remove the failed active management instance and add in a new standby management instance to the grid by executing the `ttGridAdmin modelApply` command.

```
% ttGridAdmin modelApply
Copying Model.........................................................OK
Exporting Model Version 2.............................................OK
Unconfiguring standby management instance.............................OK
Marking objects 'Pending Deletion'....................................OK
Stop any Instances that are 'Pending Deletion'........................OK
Deleting any Instances that are 'Pending Deletion'....................OK
Deleting any Hosts that are no longer in use..........................OK
Verifying Installations...............................................OK
Creating any missing Installations....................................OK
Creating any missing Instances........................................OK
Adding new Objects to Grid State......................................OK
Configuring grid authentication.......................................OK
Pushing new configuration files to each Instance......................OK
Making Model Version 2 current........................................OK
Making Model Version 3 writable.......................................OK
Checking ssh connectivity of new Instances............................OK
Starting new management instance......................................OK
Configuring standby management instance...............................OK
Starting new data instances...........................................OK
ttGridAdmin modelApply complete
```

The `ttGridAdmin modelApply` command initiates the active standby configuration between the active and standby management instances and replicates the management information on the active management instance to the standby management instance.

4. Verify that the active and standby management instances are as expected in their new roles with the `ttGridAdmin mgmtExamine` command:

```
% ttGridAdmin mgmtExamine
Both active and standby management instances are up. No action required.

Host Instance Reachable RepRole(Self) Role(Self) Seq RepAgent RepActive
-------------------------------------------------------------------------
host2 instance1 Yes     Active        Active     603 Up       Yes
host9 instance1 Yes     Standby       Standby    603 Up       No
```

Standby management instance failure

How you re-activate the standby management instance depends on the type of failure as described in the following sections:

- Standby management instance recovers
- Standby management instance experiences permanent failure

Standby management instance recovers

If the standby management instance recovers, then:

1. Check the status with the `ttGridAdmin mgmtExamine` command:

```
% ttGridAdmin mgmtExamine
Active management instance is up, but standby is down

Host Instance Reachable RepRole(Self) Role(Self) Seq RepAgent RepActive Message
-----------------------------------------------------------------------------
```

```
host1 instance1 Yes     Active        Active     605 Up       No
host2 instance1 No      Unknown       Unknown        Down     No
Management database is not available

Recommended commands:
ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host2.example.com /timesten/host2/instance1/bin/ttenv ttGridAdmin
mgmtStandbyStart
```

2. Log into the host with the standby management instance. If you have not done so already, set the environment with the `ttenv` script (as described in "Creating the initial management instance" on page 4-2).

3. Once you bring the failed management instance back up, then execute the `ttGridAdmin mgmtStandbyStart` command on the host with the standby management instance.

   ```
   % ttGridAdmin mgmtStandbyStart
   Standby management instance started
   ```

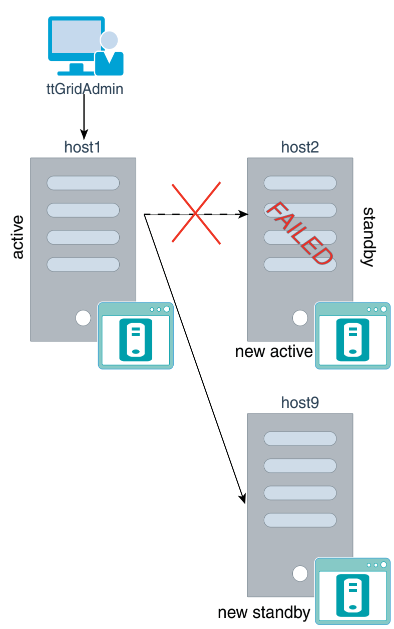
   This command re-integrates the standby management instance in your grid, initiates the active standby configuration between the active and standby management instances and replicates all management information on the active management instance to the standby management instance.

Standby management instance experiences permanent failure

If the standby management instance has permanently failed, perform the following commands:

- Delete the failed standby management instance on the `host2` host.
- Create a new standby management instance on the `host9` host to take over the duties of the failed standby management instance. Then, the active management instance replicates the management information to the new standby management instance.

Figure 11–10 The standby management instance fails permanently



1. Remove the permanently failed standby management instance from the model with the `ttGridAdmin instanceDelete` command.

```
% ttGridAdmin instanceDelete host2.instance1
Instance instance1 on Host host2 deleted from Model
```

Note: If there are no other instances on the host where the failed management instance existed, you may want to delete the host and the installation.

---

2. Add a new standby management instance with its supporting host and installation to the model.

```
% ttGridAdmin hostCreate host9 -address host9.example.com
Host host9 created in Model
% ttGridAdmin installationCreate -host host9 -location
/timesten/host9/installation1
Installation installation1 on Host host9 created in Model
% ttGridAdmin instanceCreate -host host9 -location /timesten/host9
-type management
Instance instance1 on Host host9 created in Model
```

3. Apply the configuration changes to remove the failed standby management instance and add in a new standby management instance to the grid by executing the `ttGridAdmin modelApply` command, as shown in "Applying the changes made to the model" on page 4-18.

```
% ttGridAdmin modelApply
Copying Model.......................................................OK
Exporting Model Version 9...........................................OK
Unconfiguring standby management instance...........................OK
Marking objects 'Pending Deletion'..................................OK
Stop any Instances that are 'Pending Deletion'......................OK
Deleting any Instances that are 'Pending Deletion'..................OK
Deleting any Hosts that are no longer in use........................OK
Verifying Installations.............................................OK
Creating any missing Instances......................................OK
Adding new Objects to Grid State....................................OK
Configuring grid authentication.....................................OK
Pushing new configuration files to each Instance....................OK
Making Model Version 9 current......................................OK
Making Model Version 10 writable....................................OK
Checking ssh connectivity of new Instances..........................OK
Starting new management instance....................................OK
Configuring standby management instance.............................OK
Starting new data instances.........................................OK
ttGridAdmin modelApply complete
```

The `ttGridAdmin modelApply` command initiates the active standby configuration between the active and standby management instances and replicates the management information on the active management instance to the standby management instance.

Both management instances fail

You must restart the management instances to return the grid to its full functionality and to be able to manage the grid through the active management instance.

If both of the management instances are down, you need to discover which management instance has the latest changes on it to decide which management instance is to become the new active management instance.

> **Note:** If both management instances fail permanently, call Oracle Support.

The following describes the methods to perform when both management instances are down:

- Bring back both management instances
- Bring back one of the management instances

Bring back both management instances

If you can bring back both management instances:

> **Note:** If you have not done so already, set the environment with the `ttenv` script (as described in "Creating the initial management instance" on page 4-2).

1. Execute the `ttGridAdmin mgmtExamine` command on one of the management instances to discover which is the appropriate one to become the active management instance. The `ttGridAdmin mgmtExamine` command evaluates both management instances and prints out the highest sequence number for the management instance that has more management data. It is this management instance that should be re-activated as the active management instance.

```
% ttGridAdmin mgmtExamine
One or more management instance is down.
Start them and run mgmtExamine again.

Host Instance Reachable RepRole(Self) Role(Self) Seq RepAgent RepActive Message
-------------------------------------------------------------------------------
host1 instance1 No      Unknown       Unknown        Down     No
Management database is not available
host2 instance1 No      Unknown       Unknown        Down     No
Management database is not available

Recommended commands:
ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host1.example.com /timesten/host1/instance1/bin/ttenv ttDaemonAdmin -start
-force
ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host2.example.com /timesten/host2/instance1/bin/ttenv ttDaemonAdmin -start
-force
sleep 30
/timesten/host1/instance1/bin/ttenv ttGridAdmin mgmtExamine
```

2. Execute the recommended commands listed by the `ttGridAdmin mgmtExamine` command. The commands for this example result in restarting the daemons for each management instance:

```
% ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host1.example.com /timesten/host1/instance1/bin/ttenv ttDaemonAdmin -start
-force
```

```
TimesTen Daemon (PID: 3858, port: 11000) startup OK.
% ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host2.example.com /timesten/host2/instance1/bin/ttenv ttDaemonAdmin -start
-force

TimesTen Daemon (PID: 4052, port: 12000) startup OK.
```

3. Re-execute the `ttGridAdmin mgmtExamine` command to verify that both management instances are up. If either of the management instances are not up, then the `ttGridAdmin mgmtExamine` command may suggest another set of commands to run.

   In this example, the second invocation of the `ttGridAdmin mgmtExamine` command shows that the management instances are not up. Thus, this example shows that the command next requests that you:

   a. Stop the main daemon of the data instance for both management instances.

   b. Execute the `ttGridAdmin mgmtActiveStart` command on the management instance with the higher sequence number provided by the `ttGridAdmin mgmtExamine` command. This re-activates the active management instance.

   c. Execute the `ttGridAdmin mgmtStandbyStart` command on the management instance that you want to act as the standby management instance. This command assigns the other management instance as the standby management instance in TimesTen Scaleout, initiates the active standby configuration between the active and standby management instances and synchronizes the management information on the active management instance to the standby management instance.

```
% ttGridAdmin mgmtExamine
Host Instance Reachable RepRole(Self) Role(Self) Seq RepAgent RepActive Message
------------------------------------------------------------------------
host1 instance1 Yes     Active        Active     581 Down     No
host2 instance1 Yes     Standby       Standby    567 Down     No

Recommended commands:
ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host1.example.com /timesten/host1/instance1/bin/ttenv ttDaemonAdmin -stop
ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host2.example.com /timesten/host2/instance1/bin/ttenv ttDaemonAdmin -stop
sleep 30
ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host1.example.com /timesten/host1/instance1/bin/ttenv ttGridAdmin
mgmtActiveStart
sleep 30
ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host2.example.com /timesten/host2/instance1/bin/ttenv ttGridAdmin
mgmtStandbyStart
```

Executing these commands restarts both the active and standby management instances:

```
% ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host1.example.com /timesten/host1/instance1/bin/ttenv ttDaemonAdmin -stop
TimesTen Daemon (PID: 3858, port: 11000) stopped.

% ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host2.example.com /timesten/host2/instance1/bin/ttenv ttDaemonAdmin -stop
TimesTen Daemon (PID: 3859, port: 12000) stopped.
```

```
% ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host1.example.com /timesten/host1/instance1/bin/ttenv ttGridAdmin
mgmtActiveStart
This management instance is now the active

% ssh -o StrictHostKeyChecking=yes -o PasswordAuthentication=no -x
host2.example.com /timesten/host2/instance1/bin/ttenv ttGridAdmin
mgmtStandbyStart
Standby management instance started
```

Continue to re-execute the `ttGridAdmin mgmtExamine` command until you receive the message that both management instances are up.

```
% ttGridAdmin mgmtExamine
Both active and standby management instances are up. No action required.

Host Instance Reachable RepRole(Self) Role(Self) Seq RepAgent RepActive Message
---------------------------------------------------------------------
host1 instance1 Yes     Active        Active     567 Up       Yes
host2 instance1 Yes     Standby       Standby    567 Up       No
```

Bring back one of the management instances

As soon as you notice that your standby management instance is down, it is important that you recreate it as soon as possible. If not, then your grid topology may be dramatically different than it was before if your active management instance also goes down. That is, if the active management instance goes down or fails in such a way that the best option is to bring back up the standby management instance that has been down for a while, then this may result in an incorrect grid topology as follows:

- If you had recently added instances to your grid, they may be gone.
- If you had recently deleted instances from your grid, they may be back.
- If you had recently created databases, they may have been deleted.
- If you had recently destroyed databases, they might be recreated.

If you can bring back only one of the management instances, re-activate this instance as the active management instance. The following example assumes that the management instance on the `host2` host is down and the management instance on the `host1` host was able to be brought back.

1. Execute the `ttGridAdmin mgmtActiveStart` command on the management instance on `host1`. This re-activates as the active management instance.

   ```
   % ttGridAdmin mgmtActiveStart
   This management instance is now the active
   ```

2. Remove the permanently failed standby management instance from the model with the `ttGridAdmin instanceDelete` command.

   ```
   % ttGridAdmin instanceDelete host2.instance1
   Instance instance1 on Host host2 deleted from Model
   ```

   Note: If there are no other instances on the host where the down management instance existed, you may want to delete the host and the installation.

3. Add a new standby management instance with its supporting host and installation to the model.

```
% ttGridAdmin hostCreate host9 -address host9.example.com
Host host9 created in Model
% ttGridAdmin installationCreate -host host9 -location
/timesten/host9/installation1
Installation installation1 on Host host9 created in Model
% ttGridAdmin instanceCreate -host host9 -location /timesten/host9
-type management
Instance instance1 on Host host9 created in Model
```

4. Apply the configuration changes to remove the failed standby management instance and add in a new standby management instance to the grid by executing the `ttGridAdmin modelApply` command.

```
% ttGridAdmin modelApply
Copying Model.........................................................OK
Exporting Model Version 9.............................................OK
Unconfiguring standby management instance.............................OK
Marking objects 'Pending Deletion'....................................OK
Stop any Instances that are 'Pending Deletion'........................OK
Deleting any Instances that are 'Pending Deletion'....................OK
Deleting any Hosts that are no longer in use..........................OK
Verifying Installations...............................................OK
Creating any missing Instances........................................OK
Adding new Objects to Grid State......................................OK
Configuring grid authentication.......................................OK
Pushing new configuration files to each Instance......................OK
Making Model Version 9 current........................................OK
Making Model Version 10 writable......................................OK
Checking ssh connectivity of new Instances............................OK
Starting new management instance......................................OK
Configuring standby management instance...............................OK
Starting new data instances...........................................OK
ttGridAdmin modelApply complete
```

The `ttGridAdmin modelApply` command initiates the active standby configuration between the active and standby management instances and replicates the management information on the active management instance to the standby management instance.

Performance recommendations

Enhance your performance by setting a timeout for the channel create.

Set a timeout for create channel requests

Each element communicates over channels to all other elements. However, if any request to create a channel between elements hangs due to software issues or network failures, then all channel create requests could be blocked. Since open channels are required for element communication, we need to detect any hangs within the channel creation process.

You can set a timeout (in milliseconds) to wait for a response to a channel create request to a remote element with the `ChannelCreateTimeout` general connection attribute. See "ChannelCreateTimeout" in the *Oracle TimesTen In-Memory Database Reference* for full details.

A

Example for Deploying a Grid and Database

This appendix provides an example for how to install, create, and deploy a simple grid using the `ttGridRollout` utility.

Note:

- See Chapter 1, "Overview of TimesTen Scaleout" to get familiarized with the concepts discussed in this appendix.
- See Chapter 2, "Prerequisites and Installation of TimesTen Scaleout" for a more comprehensive description of TimesTen Scaleout prerequisites and its installation process.

Note: While this appendix describes how to quickly set up a grid with a single database for development and testing purposes by using the `ttGridRollout` utility, it is also possible to configure a grid by using:

- The `ttGridAdmin` utility: Uses the command line to set up a grid with one or more databases. It provides access to the full range of configuration, management, and monitoring capabilities of TimesTen Scaleout. See "Configure your grid" on page 4-1 for more information.
- Oracle SQL Developer: Uses a GUI that provides the some of the same functionality as the ttGridAdmin utility. See "Working with TimesTen Scaleout" in the *Oracle SQL Developer Oracle TimesTen In-Memory Database Support User's Guide* for more information.

The following sections show a simple example that installs TimesTen Scaleout, sets up three membership servers, and configures a database in a grid with `k` set to `2`. The grid configuration consists of two management instances and six data instances.

Note: The parameters defined for every system in the topology of this example is based on the scenario described in "Planning your grid" on page 1-17.

TimesTen Scaleout prerequisites

Before you install TimesTen Scaleout and configure your grid, ensure that your hosts fulfill certain prerequisites.

- Ensure that TimesTen Scaleout supports the OS installed on each host
- Create a TimesTen user group and OS user
- Set the Linux system kernel parameters
- Set the memlock settings for the instance administrator

Ensure that TimesTen Scaleout supports the OS installed on each host

Once you know which systems you are going to use as hosts in your grid, ensure that TimesTen Scaleout supports the platform and operating system installed on each host. All hosts must run the same platform and OS version and release.

For a list of the operating systems that TimesTen Scaleout supports, see the *Oracle TimesTen In-Memory Database Release Notes* that are located in the installation directory.

Configure all hosts in the same internal network

Create a single internal network for all hosts to communicate with each other. Client connections to the database may be handled through a external network, if available.

See "Network requirements" on page 2-8 for more information.

Create a TimesTen user group and OS user

Create the GID for the TimesTen users group and the username and UID for the role of instance administrator. Ensure that they exist and are the same on all hosts.

```
% sudo groupadd -g 10000 timesten
% sudo useradd -u 55000 -g timesten instanceadmin
% sudo passwd instanceadmin
```

See "Understanding the TimesTen users group and the operating system user" on page 2-2 for more information.

Set the Linux system kernel parameters

Configure the following parameters of the system kernel on all hosts with a data instance. These values are based on your database requirements:

```
% sudo vi /etc/sysctl.conf

...
kernel.shmmax=51539607552
kernel.shmall=14680064
vm.nr_hugepages=24576
vm.hugetlb_shm_group=10000
...
```

Enable these settings without restarting on all modified hosts. Consider that the HugePages parameters may require a system reboot to take full effect.

```
sudo /sbin/sysctl -p
```

See "Configure shmmax and shmall" on page 2-3 and "Configure HugePages" on page 2-5 for more information on how you calculate the values for these parameters.

Set the memlock settings for the instance administrator

Set the recommended `memlock` settings for the instance administrator based on the shared memory segment of each host.

```
% sudo vi /etc/security/limits.conf

...
instanceadmin soft   memlock 50331648
instanceadmin hard   memlock 50331648
...
```

See "Modify the memlock settings" on page 2-6 for more information on how you calculate the values for these parameters.

Set the semaphore values

Configure the semaphore values of the system kernel on all hosts based on your database requirements:

```
% sudo vi /etc/sysctl.conf

...
kernel.sem = 4000 400000 2000 2560
...
```

Enable this setting without restarting the system on all modified hosts.

```
sudo /sbin/sysctl -p
```

See "Set the semaphore values" on page 2-7 for more information on how you calculate the values for this parameter.

Install TimesTen Scaleout

Unpack a TimesTen Scaleout distribution in the location you defined for the host of your active management instance. For this example, the location is the `/grid` directory on the `host1` host. TimesTen Scaleout automatically sets `/grid/tt18.1.4.1.0` as the location for the installation of the management instance when the grid is created.

Note: Unless stated otherwise and up to the end of this appendix, you should run all commands on the system that you defined for the host of the active management instance.

```
% mkdir -p /grid
% unzip /mydir/timesten181410.server.linux8664.zip -d /grid
...
```

See "Installing TimesTen Scaleout" on page 2-10 for more information on how to install TimesTen Scaleout.

Set passwordless SSH between all hosts

Use the `ttGridAdmin gridSshConfig` command to set up the required passwordless SSH access between the internal network addresses of all hosts for the instance administrator.

```
% /grid/tt18.1.4.1.0/bin/ttGridAdmin gridSshConfig
 -mgmtAddress int-host1 int-host2
 -dataAddress int-host3 int-host4 int-host5 int-host6 int-host7 int-host8
```

See "Setting passwordless SSH" on page 2-12 for more information.

Set up the membership service

TimesTen Scaleout includes Apache ZooKeeper as a third party membership service. You can find the ZooKeeper installation files in the *`installation_dir`*`/tt18.1.4.1.0/3rdparty/apache-zookeeper-3.5.8-bin.tar.gz` file of a TimesTen Scaleout installation.

To configure and initialize the membership service as required for TimesTen Scaleout, complete the next steps:

1. Install ZooKeeper
2. Configure the ZooKeeper servers
3. Start the ZooKeeper servers
4. Create the client configuration file

Note: See "Overview of the TimesTen Scaleout membership service" on page 3-1 for a more comprehensive description of the membership service in TimesTen Scaleout, including the configuration of Apache ZooKeeper.

Install ZooKeeper

Unpack Apache ZooKeeper on each system that you defined for the role of a membership server.

```
% mkdir -p /grid/membership
% tar -zvxf apache-zookeeper-3.5.8-bin.tar.gz -C /grid/membership
```

Configure the ZooKeeper servers

Once the installation files are available on all the systems defined as membership servers, create the `zoo.cfg` and `myid` configuration files on those systems.

```
% vi /grid/membership/apache-zookeeper-3.5.8-bin/conf/zoo.cfg

tickTime=250
initLimit=40
syncLimit=12
dataDir=grid/membership/apache-zookeeper-3.5.8-bin/data
clientPort=2181
server.1=ms-host1:2888:3888
server.2=ms-host2:2888:3888
server.3=ms-host3:2888:3888
autopurge.snapRetainCount=3
autopurge.purgeInterval=1
```

```
4lw.commands.whitelist=stat, ruok, conf, isro
```

Ensure that in the `myid` file you assign the same *n* value as in the `server.n` parameter of the `zoo.cfg` file. For example, since the `ms-host1` system is identified as `server.1` in the `zoo.cfg` file, then the `myid` file of that system must contain a single line with a `1`.

```
% vi /grid/membership/apache-zookeeper-3.5.8-bin/conf/myid

1
```

Also, create the location specified for the `dataDir` parameter.

```
% mkdir -p /grid/membership/apache-zookeeper-3.5.8-bin/data
```

See "Configuring Apache ZooKeeper as the membership service" on page 3-7 for more information on the parameters included in the `zoo.cfg` and `myid` configuration files.

Start the ZooKeeper servers

Start the ZooKeeper server on all the systems that you defined for the role of a membership server.

```
% /grid/membership/apache-zookeeper-3.5.8-bin/bin/zkServer.sh start
```

If you want to verify that ZooKeeper is running properly, use:

```
% /grid/membership/apache-zookeeper-3.5.8-bin/bin/zkCli.sh -server ms-host1:2181
```

Create the client configuration file

The client configuration file identifies the host names and client TCP/IP ports of all membership servers.

Create a client configuration file in a directory on the system defined as the host of the active management instance, as shown in Example A–1.

Example A–1 Sample client configuration file: membership.conf

```
% vi /mydir/membership.conf

Servers ms-host1!2181,ms-host2!2181,ms-host3!2181
```

Deploy a grid and database

TimesTen Scaleout provides several options for you to successfully configure and deploy a grid. One of those options is the `ttGridRollout` utility. The `ttGridRollout` utility uses the parameters you define in a configuration file to deploy a grid and database from start to finish without needing further input from you. This utility uses `ttGridAdmin` commands to perform the operations related to the initial configuration and deployment of a grid and database. You can find the `ttGridRollout` utility in the `bin` directory of a TimesTen Scaleout installation.

- Create a database definition file
- Create a connectable file
- Create a SQL script file for your database
- Create a configuration file for the ttGridRollout utility
- Create a grid and database

- Connect to the database

For more information on the `ttGridRollout` utility, see "ttGridRollout" in the *Oracle TimesTen In-Memory Database Reference*.

Create a database definition file

The database definition file (suffix of `.dbdef`) contains the data store and first connection attributes of a database. You must name the file as *`database_name`*`.dbdef`. For example, for a database named `database1`, the database definition file would be `database1.dbdef`.

Create a database definition file in a directory on the system defined as the host of the active management instance, as shown in Example A–2.

Example A–2 Database definition file

```
% vi /mydir/database1.dbdef

[database1]
DataStore=/disk1/databases/database1
LogDir=/disk2/logs
DatabaseCharacterSet=AL32UTF8
Durability=0
PermSize=32768
TempSize=4096
LogBufMB=1024
Connections=2048
```

See "Creating a database definition file" on page 5-2 for more information on the database definition file.

Create a connectable file

The connectable file (suffix of `.connect`) contains the general connection attributes for a connection to a database. TimesTen Scaleout supports connectables that can be either for direct or client/server connections to the database.

Create a connectable file in a directory on the system defined as the host of the active management instance, as shown in Example A–3.

Example A–3 Connectable file

```
% vi /mydir/database1CS.connect

ConnectionCharacterSet=AL32UTF8
```

See "Creating a connectable file" on page 5-7 for more information on the connectable file.

Create a SQL script file for your database

The SQL script file contains the SQL statements to create SQL objects for your database.

Create a SQL script file in a directory on the system defined as the host of the active management instance, as shown in Example A–4.

Note: See "Defining table distribution schemes" on page 5-13 for details on the `CREATE TABLE` statements and their distribution schemes included in the `database1.sql` file.

Example A–4 SQL script file

```
% vi /mydir/database1.sql

CREATE USER terry IDENTIFIED BY password;

GRANT CREATE SESSION TO terry;

CREATE TABLE terry.account_type
(
    type            CHAR(1) NOT NULL PRIMARY KEY,
    description     VARCHAR2(100) NOT NULL
)
DUPLICATE;

CREATE TABLE terry.account_status
(
    status          NUMBER(2,0) NOT NULL PRIMARY KEY,
    description     VARCHAR2(100) NOT NULL
)
DUPLICATE;

CREATE TABLE terry.customers
(
    cust_id            NUMBER(10,0) NOT NULL PRIMARY KEY,
    first_name         VARCHAR2(30) NOT NULL,
    last_name          VARCHAR2(30) NOT NULL,
    addr1              VARCHAR2(64),
    addr2              VARCHAR2(64),
    zipcode            VARCHAR2(5),
    member_since       DATE NOT NULL
)
DISTRIBUTE BY HASH;

CREATE TABLE terry.accounts
(
    account_id         NUMBER(10,0) NOT NULL PRIMARY KEY,
    phone              VARCHAR2(16) NOT NULL,
    account_type       CHAR(1) NOT NULL,
    status             NUMBER(2,0) NOT NULL,
    current_balance    NUMBER(10,2) NOT NULL,
    prev_balance       NUMBER(10,2) NOT NULL,
    date_created       DATE NOT NULL,
    cust_id            NUMBER(10,0) NOT NULL,
    CONSTRAINT fk_customer
        FOREIGN KEY (cust_id)
            REFERENCES terry.customers(cust_id),
    CONSTRAINT fk_acct_type
        FOREIGN KEY (account_type)
            REFERENCES terry.account_type(type),
    CONSTRAINT fk_acct_status
        FOREIGN KEY (status)
            REFERENCES terry.account_status(status)
)
DISTRIBUTE BY REFERENCE (fk_customer);
```

```
CREATE TABLE terry.transactions
(
    transaction_id      NUMBER(10,0) NOT NULL,
    account_id          NUMBER(10,0) NOT NULL ,
    transaction_ts      TIMESTAMP NOT NULL,
    description         VARCHAR2(60),
    optype              CHAR(1) NOT NULL,
    amount              NUMBER(6,2) NOT NULL,
    PRIMARY KEY (account_id, transaction_id, transaction_ts),
    CONSTRAINT fk_accounts
        FOREIGN KEY (account_id)
            REFERENCES terry.accounts(account_id)
)
DISTRIBUTE BY REFERENCE (fk_accounts);

CREATE SEQUENCE terry.txn_seq CACHE 100 BATCH 1000000;
```

Create a configuration file for the ttGridRollout utility

The configuration file for the `ttGridRollout` utility defines all the necessary parameters to successfully create and deploy a grid and database in TimesTen Scaleout.

Create a configuration file for the `ttGridRollout` utility, as shown in Example A–5. The configuration file in Example A–5:

- Names the grid as `grid1`.
- Defines the membership servers provided by the `membership.conf` file.
- Defines the location for the installation files for every installation object as `/grid/tt18.1.4.1.0` on their respective host.
- Defines the location for the instance files of every instance object as `/grid` on their respective hosts.
- Creates the database definition provided by the `database1.dbdef` file.
- Creates the client/server connectable provided by the `database1CS.connect` file.
- Adds the SQL schema provided by the `database1.sql` file to the `database1` database.
- Creates two management instances, including their respective hosts and installations.
- Creates six data instances, including their respective hosts and installations, evenly assigned to two data space groups. The `ttGridRollout` utility sets K-safety to `2` at grid creation to satisfy the need of two data space groups.

Note: See "Define the network parameters of each host and membership server" on page 1-20 for details on the attributes used for every instance in this example.

Example A–5 ttGridRollout configuration file

```
% vi /mydir/grid1.conf

grid_name = grid1
zoo_conf = /mydir/membership.conf
instance_location = /grid
```

```
installation_location = /grid
dbdef_file = /mydir/database1.dbdef
cs_connect_files = /mydir/database1CS.connect
init_script = /mydir/database1.sql
mgmt_instances = [
        { "host":"host1", "address":"int-host1", "instance":"instance1",
 "daemonport":6624, "csport":6625, "mgmtport":3754},
        { "host":"host2", "address":"int-host2", "instance":"instance1",
 "daemonport":6624, "csport":6625, "mgmtport":3754}
]
data_instances = [
        { "host":"host3", "internalAddress":"int-host3",
 "externalAddress":"ext-host3.example.com", "dataspacegroup":1,
  "instance":"instance1", "daemonport":6624, "csport":6625},
        { "host":"host4", "internalAddress":"int-host4",
 "externalAddress":"ext-host4.example.com", "dataspacegroup":2,
 "instance":"instance1", "daemonport":6624, "csport":6625},
        { "host":"host5", "internalAddress":"int-host5",
 "externalAddress":"ext-host5.example.com", "dataspacegroup":1,
 "instance":"instance1", "daemonport":6624, "csport":6625},
        { "host":"host6", "internalAddress":"int-host6",
 "externalAddress":"ext-host6.example.com", "dataspacegroup":2,
 "instance":"instance1", "daemonport":6624, "csport":6625},
        { "host":"host7", "internalAddress":"int-host7",
 "externalAddress":"ext-host7.example.com", "dataspacegroup":1,
 "instance":"instance1", "daemonport":6624, "csport":6625},
        { "host":"host8", "internalAddress":"int-host8",
 "externalAddress":"ext-host8.example.com", "dataspacegroup":2,
 "instance":"instance1", "daemonport":6624, "csport":6625}
]
```

Create a grid and database

Use the `ttGridRollout` utility to create a grid and database based on the configuration file you provide.

```
% /grid/tt18.1.4.1.0/bin/ttGridRollout /mydir/grid1.conf
INFO: Checking Zookeeper on ms-host1!2181 -- OK
INFO: Checking Zookeeper on ms-host2!2181 -- OK
INFO: Checking Zookeeper on ms-host3!2181 -- OK
INFO: Checking the address for the management database -- OK
INFO: Checking connectivity to int-host1 -- OK

================================================================================

/grid/tt18.1.4.1.0/bin/ttInstanceCreate -grid -location /grid -name
 instance1 -daemonport 6624 -csport 6625
Creating instance in /grid/instance1 ...

NOTE: The TimesTen daemon startup/shutdown scripts have not been installed.

The startup script is located here :
'/grid/instance1/startup/tt_instance1'

Run the 'setuproot' script :
/grid/instance1/bin/setuproot -install
This will move the TimesTen startup script into its appropriate location.

The 18.1.4.1 Release Notes are located here :
  '/grid/tt18.1.4.1.0/README.html'
```

```
/grid/instance1/bin/ttenv ttGridAdmin gridCreate grid1 -k 2 -host host1 -address
 int-host1 -membership zookeeper -membershipConfig /mydir/membership.conf
 -mgmtport 3754
/grid/instance1/bin/ttenv ttGridAdmin hostCreate host2 -address int-host2
/grid/instance1/bin/ttenv ttGridAdmin installationCreate host2 -location /grid
/grid/instance1/bin/ttenv ttGridAdmin instanceCreate host2.instance1 -location
 /grid -type management -daemonport 6624 -csport 6625 -mgmtport 3754
/grid/instance1/bin/ttenv ttGridAdmin modelApply
/grid/instance1/bin/ttenv ttGridAdmin hostCreate host3 -externaladdress int-host3
 -internaladdress ext-host3.example.com -dataspacegroup 1
/grid/instance1/bin/ttenv ttGridAdmin installationCreate host3 -location /grid
/grid/instance1/bin/ttenv ttGridAdmin hostCreate host4 -externaladdress int-host4
 -internaladdress ext-host4.example.com -dataspacegroup 2
/grid/instance1/bin/ttenv ttGridAdmin installationCreate host4 -location /grid
/grid/instance1/bin/ttenv ttGridAdmin hostCreate host5 -externaladdress int-host5
 -internaladdress ext-host5.example.com -dataspacegroup 1
/grid/instance1/bin/ttenv ttGridAdmin installationCreate host5 -location /grid
/grid/instance1/bin/ttenv ttGridAdmin hostCreate host6 -externaladdress int-host6
 -internaladdress ext-host6.example.com -dataspacegroup 2
/grid/instance1/bin/ttenv ttGridAdmin installationCreate host6 -location /grid
/grid/instance1/bin/ttenv ttGridAdmin hostCreate host7 -externaladdress int-host7
 -internaladdress ext-host7.example.com -dataspacegroup 1
/grid/instance1/bin/ttenv ttGridAdmin installationCreate host7 -location /grid
/grid/instance1/bin/ttenv ttGridAdmin hostCreate host8 -externaladdress int-host8
 -internaladdress ext-host8.example.com -dataspacegroup 2
/grid/instance1/bin/ttenv ttGridAdmin installationCreate host8 -location
 /grid
/grid/instance1/bin/ttenv ttGridAdmin instanceCreate host3.instance1 -location
 /grid -daemonport 6624 -csport 6625
/grid/instance1/bin/ttenv ttGridAdmin instanceCreate host4.instance1 -location
 /grid -daemonport 6624 -csport 6625
/grid/instance1/bin/ttenv ttGridAdmin instanceCreate host5.instance1 -location
 /grid -daemonport 6624 -csport 6625
/grid/instance1/bin/ttenv ttGridAdmin instanceCreate host6.instance1 -location
 /grid -daemonport 6624 -csport 6625
/grid/instance1/bin/ttenv ttGridAdmin instanceCreate host7.instance1 -location
 /grid -daemonport 6624 -csport 6625
/grid/instance1/bin/ttenv ttGridAdmin instanceCreate host8.instance1 -location
 /grid -daemonport 6624 -csport 6625
/grid/instance1/bin/ttenv ttGridAdmin dbdefCreate /mydir/database1.dbdef
/grid/instance1/bin/ttenv ttGridAdmin modelApply
/grid/instance1/bin/ttenv ttGridAdmin dbCreate -wait 180 database1
/grid/instance1/bin/ttenv ttGridAdmin dbDistribute database1 -add all -apply
/grid/instance1/bin/ttenv ttGridAdmin dbOpen -wait 180 database1
/grid/instance1/bin/ttenv ttGridAdmin connectableCreate -dbdef database1 -cs
 /mydir/database1CS.connect
/grid/instance1/bin/ttenv ttGridAdmin modelApply
/grid/instance1/bin/ttenv ttGridAdmin instanceExec -only host3.instance1 "ttIsql
 database1 <<EOF
CREATE USER terry IDENTIFIED BY password;

GRANT CREATE SESSION TO terry;

CREATE TABLE terry.account_type
(
    type            CHAR(1) NOT NULL PRIMARY KEY,
    description     VARCHAR2(100) NOT NULL
)
DUPLICATE;
```

```
CREATE TABLE terry.account_status
(
    status          NUMBER(2,0) NOT NULL PRIMARY KEY,
    description     VARCHAR2(100) NOT NULL
)
DUPLICATE;

CREATE TABLE terry.customers
(
    cust_id            NUMBER(10,0) NOT NULL PRIMARY KEY,
    first_name         VARCHAR2(30) NOT NULL,
    last_name          VARCHAR2(30) NOT NULL,
    addr1              VARCHAR2(64),
    addr2              VARCHAR2(64),
    zipcode            VARCHAR2(5),
    member_since       DATE NOT NULL
)
DISTRIBUTE BY HASH;

CREATE TABLE terry.accounts
(
    account_id         NUMBER(10,0) NOT NULL PRIMARY KEY,
    phone              VARCHAR2(16) NOT NULL,
    account_type       CHAR(1) NOT NULL,
    status             NUMBER(2,0) NOT NULL,
    current_balance    NUMBER(10,2) NOT NULL,
    prev_balance       NUMBER(10,2) NOT NULL,
    date_created       DATE NOT NULL,
    cust_id            NUMBER(10,0) NOT NULL,
    CONSTRAINT fk_customer
        FOREIGN KEY (cust_id)
            REFERENCES terry.customers(cust_id),
    CONSTRAINT fk_acct_type
        FOREIGN KEY (account_type)
            REFERENCES terry.account_type(type),
    CONSTRAINT fk_acct_status
        FOREIGN KEY (status)
            REFERENCES terry.account_status(status)
)
DISTRIBUTE BY REFERENCE (fk_customer);

CREATE TABLE terry.transactions
(
    transaction_id      NUMBER(10,0) NOT NULL,
    account_id          NUMBER(10,0) NOT NULL ,
    transaction_ts      TIMESTAMP NOT NULL,
    description         VARCHAR2(60),
    optype              CHAR(1) NOT NULL,
    amount              NUMBER(6,2) NOT NULL,
    PRIMARY KEY (account_id, transaction_id, transaction_ts),
    CONSTRAINT fk_accounts
        FOREIGN KEY (account_id)
            REFERENCES terry.accounts(account_id)
)
DISTRIBUTE BY REFERENCE (fk_accounts);

CREATE SEQUENCE terry.txn_seq CACHE 100 BATCH 1000000;

EOF"
```

```
================================================================================
6-instance (3x2) grid successfully created.


Management Instance Locations
-----------------------------
- int-host1:/grid/instance1
- int-host2:/grid/instance1

Please source ttenv script under Management Instances for grid management via
 "ttGridAdmin" commands.

  For example, to use the first management instance, on int-host1:
  sh:  . /grid/instance1/bin/ttenv.sh
  csh: source /grid/instance1/bin/ttenv.csh


Data Instance Locations
-----------------------
- host3.instance1 ==> int-host3:/grid/instance1
- host4.instance1 ==> int-host4:/grid/instance1
- host5.instance1 ==> int-host5:/grid/instance1
- host6.instance1 ==> int-host6:/grid/instance1
- host7.instance1 ==> int-host7:/grid/instance1
- host8.instance1 ==> int-host8:/grid/instance1

Please source ttenv script under Data Instances for database operations.

  For example, to use instance1, on int-host3:
  sh:  . /grid/instance1/bin/ttenv.sh
  csh: source /grid/instance1/bin/ttenv.csh
```

Connect to the database

Connect to your database through a direct or client connection. For a direct connection, set your environment to one of the data instances, `host3.instance1` for example, and use the `database1` connectable to connect to the database.

```
% source /grid/instance1/bin/ttenv.csh
...
% ttIsql -connStr "DSN=database1;UID=terry"
```

See "Connecting to a database" on page 5-7 for more information on how to connect to a database in TimesTen Scaleout.

B

TimesTen Scaleout Environment

This appendix provides reference material on:

- Environment variables
- Instance home directory and subdirectories
- Managing a development or test environment

Environment variables

These sections discuss environment variables:

- Setting environment variables
- Environment variable descriptions

Setting environment variables

You set environment variables for a terminal window, which enables the window to run commands for a particular instance. Here is a list of situations where you should set your environment variables:

- After you create the active management instance
- Before using `ttGridAdmin` or any TimesTen utility
- Before executing a direct mode application on a host running a data instance
- Before executing a client server application on a host running a client (or data) instance

You set the environment variables by sourcing the `ttenv` shell script (`ttenv.sh` or `ttenv.csh`). TimesTen creates the scripts after you create an instance. These scripts are located in the *`grid`*/*`instance1`*/`bin` directory (where *`grid`*/*`instance1`* is the full path of the instance). By sourcing these scripts, the environment variables required to use a TimesTen Scaleout instance are set.

The environment variables include `TIMESTEN_HOME`, `PATH`, `LD_LIBRARY_PATH` (or equivalent) and `TNS_ADMIN`.

For example:

For a Bourne-type shell, such as `sh`, `bash`, `zsh`, or `ksh`:

```
% ./ttenv.sh
```

For a `csh` or `tcsh` shell:

```
% source ttenv.csh
```

Environment variable descriptions

These sections provide more details on the environment variables:

- TIMESTEN_HOME environment variable
- NLS_LANG environment variable
- Shared library path environment variable
- PATH environment variable
- Temporary directory environment variable
- TNS_ADMIN environment variable
- Java environment variables

TIMESTEN_HOME environment variable

The `TIMESTEN_HOME` environment variable specifies the home directory of the TimesTen Scaleout instance. You explicitly set this variable when sourcing the `ttenv` script.

NLS_LANG environment variable

The character set specified in the database definition file is used by default for the connection, if not overridden by `NLS_LANG` or if not in the connectable. While setting the character set explicitly is recommended, the default is normally `AMERICAN_AMERICA.US7ASCII`. To use the environment variable to set the character set, do the following:

```
NLS_LANG=.WE8ISO8859P1
```

For more information, see:

- "Character sets" in *Oracle TimesTen In-Memory Database C Developer's Guide* for more information.
- "Supported character sets" in *Oracle TimesTen In-Memory Database Reference* for more information.

Shared library path environment variable

The shared library path environment variable is set when sourcing `ttenv`. This environment variable specifies the path for shared libraries. The `ttenv` script adds *$TIMESTEN_HOME*`/install/lib` to `LD_LIBRARY_PATH`.

PATH environment variable

TimesTen provides utilities for managing and debugging your applications. For these utilities to be available, the path for executables in *$TIMESTEN_HOME*`/bin` and *$TIMESTEN_HOME*`/install/bin` must be designated in the `PATH` setting. The path is updated to include these directories when you source `ttenv`.

In addition, to compile programs, be sure the location of the compiler for your programming language is in the `PATH` setting.

Temporary directory environment variable

`TMPDIR` specifies the location of the temporary directory, which TimesTen uses during recovery and other operations.

TNS_ADMIN environment variable

The `TNS_ADMIN` environment variable specifies the full path to the directory where the `tnsnames.ora` file is located.

- For TimesTen OCI, Pro*C/C++, or ODP.NET, set the `TNS_ADMIN` environment variable to indicate the full path to the directory where the `tnsnames.ora` file is located.
- TimesTen Scaleout automatically populates the `tnsnames.ora` file on all instances with entries for all the connectables. Do not manually configure these entries, as the related configuration files are owned by TimesTen Scaleout.
- The `tnsnames` and related information for additional entries, such as for Oracle database connections (as applicable), are brought in and distributed through the `ttGridAdmin` `TNSNamesImport` and `SQLNetImport` commands. See "Import TNS names (TNSNamesImport)" and "Import a sqlnet file (SQLNetImport)" in *Oracle TimesTen In-Memory Database Reference* for more information.

Java environment variables

For Java applications, there are additional environment variables of interest. These sections provide information about additional environment variables or considerations that affect Java applications:

- CLASSPATH environment variable
- PATH environment variable settings for Java

CLASSPATH environment variable Java classes and class libraries are found on the class path, as specified by the `CLASSPATH` environment variable. Before executing a Java program that loads any of the TimesTen JDBC drivers, the `CLASSPATH` setting must include the class library file and path:

```
$TIMESTEN_HOME/install/lib/ttjdbcjdk_ver.jar
```

where *`jdk_ver`* indicates the JDK version. For JDK8, *`jdk_ver`* is 8 and the file name is `ttjdbc8.jar`.

> **Note:** If multiple JAR files are listed in the `CLASSPATH`, ensure that the TimesTen JAR file is listed first.

`CLASSPATH` elements are separated by colons. For example (`sh` type shell):

```
CLASSPATH=.:$TIMESTEN_HOME/install/lib/ttjdbc7.jar
export CLASSPATH
```

Or (`csh` type shell):

```
setenv CLASSPATH .:$TIMESTEN_HOME/install/lib/ttjdbc7.jar
```

To check the JDK version:

```
% java -version
```

PATH environment variable settings for Java For Java applications, ensure that the locations of the `java` and `javac` executables are in the `PATH` setting.

Instance home directory and subdirectories

When you create an instance, each instance includes these subdirectories within *$TIMESTEN_HOME*:

- `bin`: TimesTen utilities and executables tailored and specific to the instance

  This includes `ttenv`, which sets environment variables appropriately for the TimesTen environment for your session, and `setuproot.sh`, which can be run as root to cause data instances to be automatically started whenever the operating system reboots.

  Note that `ttenv` also puts the `bin` directory in your path.
- `conf`: Contains the `timesten.conf` file, which is the TimesTen instance configuration file
- `diag`: Diagnostic output, including the daemon log and error log
- `grid`: Files and resources for TimesTen Scaleout
- `info`: Working directory of the TimesTen daemon, containing persistent state about the TimesTen instance
- `install`: Symbolic link referencing the installation associated with this instance.
- `plsql`: Contains this subdirectory:
  - `utl_file_dir`: The only directory that can be read from or written to by PL/SQL blocks using the `UTL_FILE` package
- `startup`: Contains a script that can be added to `/etc/init.d` to cause the instance to be automatically started at system startup and stopped at system shutdown.

Notes:

- TimesTen Scaleout updates configuration files as needed. Do not update them manually.
- Client-only instances do not include the `grid` or the `startup` directories.

Managing a development or test environment

If you have a test or a development environment where you are creating, using and then destroying multiple grids, you may need to purge the membership server meta data for any grid that is destroyed and will not be used again.

TimesTen Scaleout creates membership server meta data to represent each independent grid on each instance in a grid. If you know that a particular grid has been destroyed and is never going to be used again, then you can perform the following on one instance:

1. Locate the `timesten.conf` configuration file in the `/conf` directory under the instance home directory.
2. Identify the membership server entries in the `timesten.conf` configuration file with the `grid_guid` and `grid_name` parameters, such as:

```
grid_guid=4012FC64-8B9X-45D1-A16C-ED52C3098CAD
grid_name=mygrid
```

The membership server entry has the naming structure of *`grid_name.grid_guid`* and exists within the `/oracle/timesten/grid/membership` directory.

Note: It is imperative that you identify the correct grid to avoid deleting a membership server of an active grid.

---

3. Execute the `zkCli.sh` command to connect to the membership servers.

   ```
   ./zkCli.sh -server ms_host1:2181
   ```

4. Using the `zkCli.sh rmr` command, delete the membership server entries.

   ```
   rmr
   /oracle/timesten/grid/membership/mygrid.4012FC64-8B9B-45D1-A16C-ED52C3098CAD
   ```

Index

D

E

F

G

H

I

J

K

Q

R

S

T

U

Z