

# Architecting BPEL Systems for High Availability

*An Oracle White Paper*  
*May 2007*

# Architecting BPEL Systems For High Availability

INTRODUCTION.....	3
The Scope of This Document.....	3
Investing in HA Architecture.....	4
DEPLOYMENT ARCHITECTURE.....	5
HA Architecture Concepts and Dependency Diagrams.....	5
Deployment Architecture Recommendations.....	7
Disaster Recovery.....	9
INSTALLATION AND CONFIGURATION.....	10
Installation.....	10
Configuration.....	11
Configure ONS for FCF.....	11
Configure Data Sources for RAC, FCF.....	12
Configure Database and AQ Adapters for JTA Integration.....	12
Configure BPEL Parameters.....	12
PROCESS DESIGN.....	13
Initiation Methods and Load Balancing.....	13
Message Consumption.....	14
Database Polling.....	14
Filesystem Polling.....	14
Packaged Applications.....	16
Endpoint Virtualization.....	16
Resilience to Endpoint Failures.....	17
Resilience to Dehydration Store Failures.....	19
SYSTEM ADMINISTRATION.....	19
J2EE Container Configuration.....	20
BPEL Domain Management.....	21
BPEL Process Management.....	22
BPEL System Monitoring and Management.....	23
Message Recovery.....	23
Dehydration Store Administration.....	23
TESTING AND TROUBLESHOOTING.....	24
Failure Conditions and Expected Behavior.....	24
RAC Node Failure.....	25
BPEL Node Failure.....	25
Endpoint Failure.....	26
Testing and Validating HA BPEL Systems.....	26
Troubleshooting HA BPEL Systems.....	27
SUMMARY AND FUTURE WORK.....	28
REFERENCES.....	28

# Architecting BPEL Systems for High Availability

**The goal of this paper is to make BPEL HA knowledge more widely available.**

## INTRODUCTION

This paper collects and organizes knowledge on the subject of architecting BPEL systems for high availability - knowledge gained and refined by Oracle and customer personnel while developing and using Oracle BPEL Process Manager. With demand for HA BPEL systems becoming ever more prevalent, the goal of this paper is to make the collected knowledge more widely available, to more efficiently enable consistent customer success with HA BPEL systems.

### The Scope of This Document

High availability is one of several quality attributes that may be important in a given BPEL system. Other quality attributes that may require special attention in a BPEL system include performance, scalability, and transactional correctness. Treatment of those other quality attributes in BPEL systems, however, falls outside the scope of this paper, which focuses on high availability in the software layers of BPEL systems.

**This paper focuses on single-site active-active HA configurations in BPEL 10.1.3.1.**

The quality attribute of scalability deserves special mention vis-à-vis the scope of this paper, because the architectural technique of clustering is used to achieve scalability and is also used to achieve high availability. But all discussion of clustering in this paper is in the context of high availability, even though the additional BPEL system design considerations introduced by clustering must be addressed regardless of why clustering is used. Clustering for high availability is arguably simpler than clustering for scalability, because the goal is merely redundancy, not calibrating resource supply to load patterns.

Likewise, this revision of this paper is targeted to version 10.1.3.1 of Oracle BPEL Process Manager. Future revisions may be targeted to later product versions.

High availability of a system at a given site is typically achieved through architecture configurations termed “active-active” and “active-passive”. In addition, a disaster recovery site may be configured as a backup for a primary site. While Oracle Application Server HA documentation [B28941] describes active-active configurations, active-passive configurations, and disaster recovery for AS-based applications (of which BPEL systems are examples), this revision of this paper focuses primarily on active-active configurations at a single site.

The knowledge collected in this paper concerns the subtopics of deployment architecture, installation and configuration, process design, administration, and testing, for each of which the paper includes a major section. Before delving in to those subtopics, however, it is important to consider the level of investment that an organization should make in HA architecture for a given BPEL system.

### **Investing in HA Architecture**

Organizations contemplating HA architecture for a BPEL system should choose a level of investment commensurate with availability requirements and allocated resources for the system in question. As an example, a departmental system that processes files in batch mode, with a one-day completion window per file, may be able to tolerate occasional downtime and operate with manual recovery procedures. As a contrasting example, a high-volume revenue-critical enterprise system backing interactive user applications may require maximum failure protection, yielding absolutely zero downtime.

The intended benefit of an HA architecture is higher availability of the system having that architecture, and therefore greater continuity of the service provided by that system. Presumably excessive discontinuity of service has business consequences, such as revenue and/or expense impact, whether direct or indirect, immediate or future. Such business consequences may be known, or may not even be easy to estimate.

One key question that needs to be answered, in choosing an HA investment level, is how much discontinuity of service is “excessive”? Can the system have any planned downtime? Within the planned uptime, what is the maximum tolerable duration of an unplanned outage, realistically? How frequently can an unplanned outage of that duration be tolerated, or how much cumulative downtime can be tolerated in a given time period from multiple unplanned outages? From what kinds of failures is protection sought, and how likely are those failures? Is there any risk of critical messages getting lost if the system is down?

These questions are key because they may lead to the conclusion that certain failures can be tolerated via manual recovery procedures, while others can’t be, and require investment in HA architecture. A realistic focus on maximum tolerable outage duration, and maximum tolerable outage frequency or cumulative downtime, is more helpful than an overly simplistic and ambitious statement of availability percentage. For example insisting on “five nines”, just for the sake of it, may not make business sense given the implications. By contrast, 99.9% availability in a 24x7 system allows about 1.5 minutes of downtime daily – enough time to restart a process if necessary. In summary, a decision must be made as to the service level, in availability terms, that will be agreed with clients of the system (and in relation, the service level expected of components in the system).

Just as the benefits of HA architecture (i.e. greater continuity of service) may not be easy to estimate, so too are the costs of developing and operating HA BPEL systems perhaps difficult to quantify. At least an initial categorization of those

**Different systems may require different levels of fault protection.**

**How much downtime is too much, realistically?**

**Specify maximum tolerable outage duration or frequency, not “99.999% availability”.**

**Additional labor is entailed, to develop and operate systems employing redundancy.**

costs is straightforward. Most obviously, there is additional non-recurring capital expense for hardware, and software licenses, for redundant cluster nodes (assuming they wouldn't already be needed for scalability) and redundant sites. Not so obvious, or easily quantifiable, is additional recurring labor expense to develop and operate systems that are more complex than when less redundancy is involved. For example, the Process Design section of this paper describes additional BPEL system architecture and design considerations, and the System Administration section describes additional BPEL system administration considerations, that are raised by the presence of redundant nodes and sites in an HA BPEL system architecture.

**Are the endpoints being orchestrated by BPEL themselves highly available?**

Finally there are risks to consider, one of which is greater system complexity. In addition, specific to BPEL systems, there is the matter of whether the endpoints orchestrated by a BPEL process are themselves highly available, or single points of failure. The availability of those endpoints may not be under the control of the operator of a BPEL system in which high availability is desired. An endpoint that is not highly available renders the entire BPEL system not highly available, in the worst case. In the best case, such an endpoint causes additional development cost and system complexity to mask its potential unavailability from clients of the BPEL system. In other words, a system is only as highly available as its least available component for which no failure protection is designed.

For a more thorough discussion of investment in HA architecture, and its returns, see [Schmidt] (chapter 2, in particular). Meanwhile, for purposes of this paper, suffice it to say that organizations contemplating HA BPEL systems should analyze the availability of the endpoints they'll be orchestrating, and consciously choose the appropriate level of investment in HA architecture for their systems, considering uptime requirements and allocated resources. This choice can be facilitated by the HA architecture concepts, and table of protections and risks, presented in the next section of this paper.

## **DEPLOYMENT ARCHITECTURE**

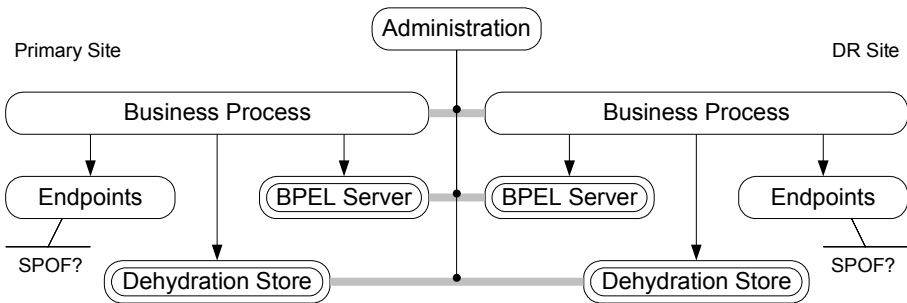
**The right deployment architecture for a given BPEL system depends on context.**

Assuming an organization intends to go ahead with an HA architecture for a BPEL system, a number of deployment architecture recommendations have emerged in collective experience with Oracle BPEL Process Manager. The specific deployment architecture recommended for a given BPEL system will necessarily depend on the context of the system and the forces and complexity therein. To understand the motivation for the deployment architecture recommendations, it is helpful to first review a few key concepts in high availability architecture, and to introduce a diagramming notation specific to high availability.

### **HA Architecture Concepts and Dependency Diagrams**

Figure 1 presents a "dependency diagram" in the notation of [Schmidt], showing the overall approach to achieving high availability in a generic BPEL system. A brief discussion of this diagram serves to highlight the key concepts in high

availability architecture, and their application to BPEL systems (those key concepts are italicized in the following discussion).



**Figure 1: Generic HA BPEL System Dependency Diagram**

Components depend on other components;  
redundancy and retry logic provide failure  
protection.

The oval icons in a dependency diagram represent *components* of a system, and the arrows between them represent *dependencies* between components. Thus in Figure 1 we see that a business process being executed via BPEL with Oracle BPEL Process Manager depends on the endpoints it orchestrates, the BPEL Server to which it is deployed, and the dehydration store used by that BPEL Server. If any of those components on which the business process depends should fail, the execution of the business process will be affected.

To protect dependent components from *failures* in the components on which they depend, high availability architectures employ *redundancy*, indicated in dependency diagrams by double-bordered component icons, or by thick lines between components (with an adornment indicating what is responsible for state replication and fault handling between repetitious component instances). Thus Figure 1 indicates that BPEL Servers and dehydration stores can be made redundant. That is done by clustering BPEL Servers in the middle tier of architecture, and by using RAC in the database tier of architecture (note it's also conceivable to use an active-passive configuration, called Cold Failover Cluster or CFC, in the database tier, although RAC is more common). Figure 1 also indicates that an entire site on which a business depends, with all its executable business processes and supporting components, can be made redundant using repetition at a disaster recovery site, and an administrative approach for state replication and failover.

Using RAC and middle-tier clustering is the  
primary technique for ensuring BPEL HA.

In brief, achieving redundancy by using RAC in the database tier, and clustering in the middle tier, is the primary technique for protecting BPEL processes at a site from failures in the infrastructural components upon which they depend, which helps ensure that they are highly available. That technique is the motivation for the deployment architecture recommendations that follow. It is generally assumed that organizations desiring highly available BPEL systems will implement redundancy for all dependencies, such that no single points of failure exist in the deployment architecture. But for completeness, Table 1 below tabulates the

protections and risks associated with different deployment architecture configurations (readers should note that “node failure” as used in Table 1 and this paper is also intended to include the failure of individual critical processes on that node, etc.).

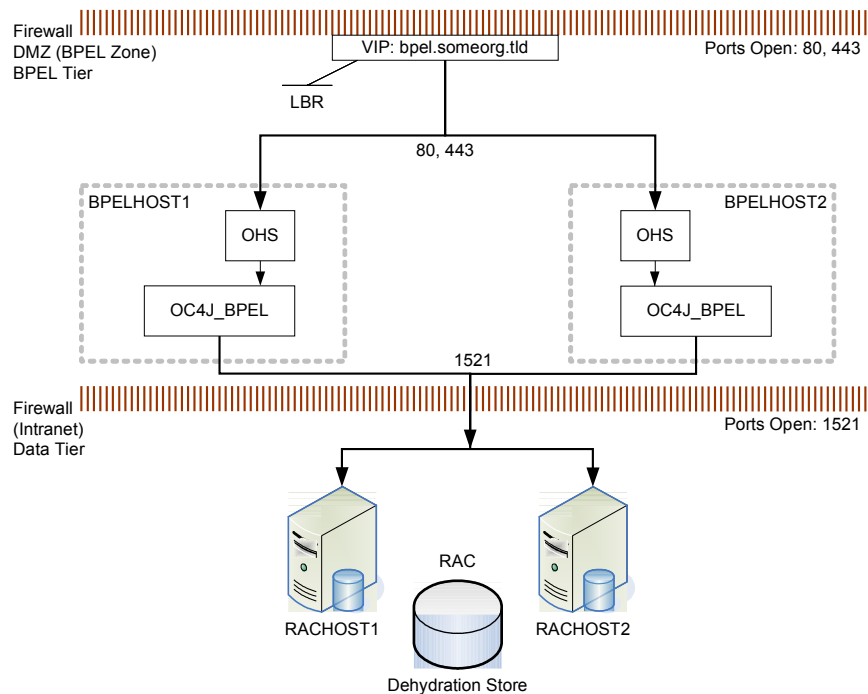
**Table 1: Protections and Risks of Deployment Architecture Configurations**

Configuration	Protections	Risks
Clustered BPEL tier only	<ul style="list-style-type: none"> <li>• Service continuity on BPEL node failure</li> </ul>	<ul style="list-style-type: none"> <li>• <b>Service continuity on DB failure</b></li> </ul>
Clustered DB tier only	<ul style="list-style-type: none"> <li>• Service continuity on DB node failure</li> </ul>	<ul style="list-style-type: none"> <li>• <b>Service continuity on BPEL node failure</b></li> </ul>
Both tiers clustered	<ul style="list-style-type: none"> <li>• Service continuity on BPEL node failure</li> <li>• Service continuity on DB node failure</li> </ul>	
Single site	<ul style="list-style-type: none"> <li>• Service continuity on component failure</li> </ul>	<ul style="list-style-type: none"> <li>• <b>Service continuity on site failure</b></li> </ul>
<b>Disaster recovery site</b>	<ul style="list-style-type: none"> <li>• <b>Service continuity on any failure</b></li> </ul>	

## Deployment Architecture Recommendations

The most basic deployment architecture recommendation is depicted in Figure 2. The BPEL tier of architecture is fronted by a load balancer and consists of two clustered BPEL nodes, depending on a two-node RAC cluster in the database tier (readers should note that the definition of “clustered BPEL nodes” need not imply clustering of the underlying Oracle Application Server instances [B28980]). This configuration affords protection against both BPEL node failure and DB node failure, subject to the expected behavior on failure conditions documented in the Testing and Troubleshooting section of this paper. Readers should be aware that a load balancer is included in the depicted architecture for completeness, but is only relevant if BPEL processes deployed to the cluster are initiated by HTTP invocation. A great many BPEL customers deploy BPEL processes initiated (only) by other means, such as file arrival, MOM message arrival, database table insert, etc.

The basic HA deployment architecture uses two nodes per tier, and a load balancer in front.

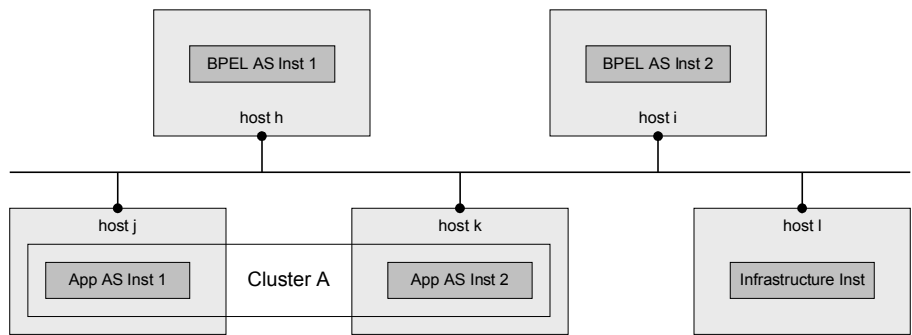


**Figure 2: Basic BPEL HA Deployment Architecture**

**Separate the BPEL and non-BPEL parts of an overall system, and allocate different hardware resources to each subsystem.**

Beyond the basic BPEL HA deployment architecture, a significant fraction of BPEL customers develop systems that include other, custom, non-BPEL J2EE applications that may interface with their BPEL systems somehow, but may be first-class systems in their own right, with considerable load, resource requirements, codebases, and complexity of their own. For these types of systems it is recommended that customers separate the BPEL and non-BPEL parts of the overall system into two different subsystems, and plan different groups of hosts in the deployment architecture for each subsystem. Such a deployment architecture is diagrammed in Figure 3. In such a deployment architecture, the set of all AS mid-tier instances in the environment is partitioned into a subset for the BPEL subsystem and a subset for the non-BPEL subsystem, each AS instance has its own host, and AS clustering may be used in the non-BPEL partition. The rationale for this partitioning and deployment architecture is that each partition may be configured, tuned, and scaled independently, according as the needs of the software it contains.





**Figure 3: Deployment Architecture for Mixed BPEL / Non-BPEL Systems**

Finally there are a couple of important specific variations on these general deployment architecture themes. First, for systems with which clients interact via HTTP, it is common to extract the Oracle HTTP Server components of AS installations to a separate tier of nodes located in its own DMZ for security reasons, although that is not explicitly shown on any of the above deployment architecture diagrams.

Second, for BPEL systems in which processes are initiated by File Adapter or FTP Adapter, an interesting dilemma arises in high availability architecture. Namely, such processes cannot be deployed to two active BPEL nodes concurrently, as would normally be done for redundancy in an active-active configuration, else they will enter a race condition upon arrival of a triggering file, and both try to process it. But if such processes are deployed to only one node, that node becomes a single point of failure. The solution to the dilemma is to adopt an active-passive configuration for the BPEL Servers to which those processes are deployed, so that the passive Server becomes active only upon failure of the active Server. With BPEL Process Manager version 10.1.3.1 such a configuration is supported directly within the Adapter framework itself (see Section 6.18 of [B31005]), so operating system –level support for active-passive failover clustering is not required to implement this deployment architecture variation. However BPEL system architects may wish to consider setting up a separate tier of BPEL servers to which to deploy only such File/FTP Adapter –initiated processes, and designing those processes to be mere front-end Dispatchers [Hohpe] to downstream processes deployed to a “normal” active-active cluster of BPEL nodes.

## Disaster Recovery

The widespread adoption and rapid maturation of Oracle BPEL Process Manager has produced an increasing number of inquiries about configuring BPEL systems for disaster recovery, which is a very important and large sub-topic of high availability, protecting against failures of entire sites. Fortunately Oracle supports disaster recovery for BPEL systems deployed on Oracle Application Server and using Oracle Database, by leveraging Oracle Application Server’s AS Guard

**For file-initiated BPEL processes, consider an additional Dispatcher tier, in active-passive configuration.**

**Disaster recovery is supported and leverages Oracle Application Server’s AS Guard feature.**

feature. In broad outline, adding a disaster recovery site to a BPEL system's deployment architecture entails provisioning the DR site and performing the same software installations as at the primary site, then using AS Guard to synchronize the BPEL dehydration store database, and ORACLE\_HOME configuration, between the primary and disaster recovery sites. For more information on creating such a configuration, refer to Section 14.1.2 of the Oracle Application Server Release Notes 10g Release 3 (10.1.3.1.0) [B31014], and the Oracle Application Server High Availability Guide 10g Release 3 (10.1.3.1.0) [B28941].

## INSTALLATION AND CONFIGURATION

Having selected an HA deployment architecture, BPEL system architects or their delegates of course must sometime install and configure Oracle products as planned in the selected deployment architecture. These products are likely to include at least Oracle Database RAC, Oracle Application Server, and Oracle BPEL Process Manager. Specifically in the case of installing Oracle BPEL Process Manager version 10.1.3.1 in a cluster, and configuring it to work with a RAC-based dehydration store, collective experience has shown there to be a proven path to achieving the desired installation and configuration. The following subsections elaborate the details of the required installation and configuration steps.

### Installation

For the installation of the RAC database in the database tier, the reader is referred to the Oracle Database RAC installation documentation [B14203]. Following are the steps to install a BPEL 10.1.3.1 cluster. The official documentation references are:

- Chapter 3 of the Oracle Application Server Enterprise Deployment Guide [B28939]
- Chapter 5 of the Oracle Application Server High Availability Guide [B28941]
- Chapter 4 of the Oracle BPEL Process Manager Installation Guide [B28980]

The steps below will depart from this documentation in a couple of places, based on collective experience to date with clustering in 10.1.3.1 BPEL. Specifically, practitioners have found the safest route at present is to not formally cluster either the AS instances or the BPEL instances, in contrast to the instructions in Chapter 4 of the BPEL Process Manager Installation Guide. Instead it is sufficient to point the BPEL instances at the same dehydration store, and put a load balancer in front of them, as was done in BPEL Process Manager version 10.1.2.0.2. Note that approach does not depend on any application server –level clustering capabilities and should therefore work with other supported application servers besides Oracle Application Server, which is assumed herein. The steps for installing with Oracle Application Server are:

1. On each mid-tier node, first install the Oracle Application Server J2EE Server and Web Server installation type, or just the J2EE

**Collective experience has shown a proven path to installing BPEL 10.1.3.1 clusters.**

Server installation type. To do this you'll have to select "Advanced Install" from the first screen of the Oracle Universal Installer.

2. Then on each mid-tier install BPEL Process Manager for AS Mid-Tier into the Oracle home from the above step. For that you need the BPEL Process Manager CD-ROM from the 10.1.3.1 distribution.
3. Go into the BPEL Admin Console on each node and set the soapServerUrl and soapCallbackUrl to point to your load balancer (if in step 1 you installed the J2EE Server and Web Server installation type, then you can configure one of the OHS instances as the load balancer).

## Configuration

**After installation, several configuration steps need to be performed for HA.**

After installation of the Oracle products in the environment, there are a number of configuration steps that need to be performed to facilitate a correctly functioning BPEL HA environment. Several of these steps have to do with configuring the BPEL Servers in the environment to interact with a RAC database correctly. The following paragraphs present the details of the necessary configuration steps.

### Configure ONS for FCF

For a BPEL Server to tolerate RAC node failure, it is necessary and crucial to configure Fast Connection Failover for the JDBC connections connecting the BPEL Server to the RAC database. Fast Connection Failover is a feature of Oracle JDBC in which the JDBC driver receives notifications from the database of instance down events, and reacts by cleaning its pools of connections to the dead instance, and raising exceptions in threads using connections to the dead instance. In FCF the driver also receives notifications of instance up events, and reacts by re-balancing its pool (see Chapter 27 of [B14355]).

**Configure ONS by editing  
\$OH/opmn/conf/opmn.xml.**

To configure FCF, it is necessary to configure Oracle Notification Service (ONS) in the AS installations into which BPEL Process Manager was installed, and to configure the BPEL Server's data-sources.xml as described in the next section. To configure ONS, all that is necessary is to edit \$ORACLE\_HOME/opmn/conf/opmn.xml to add RAC node host and port information to the <node> element's list attribute in the <notification-server> section of the file. Complete instructions are available in [Castro].

After editing opmn.xml as just described, you can verify that the ONS configuration is working by starting the AS instance (e.g. via opmnctl startall), then issuing "\$ORACLE\_HOME/opmn/bin/opmnctl debug" and looking for connections with the RAC nodes in that command's output.

Configure FCF by editing  
\$OH/j2ee/OC4J\_BPEL/config/opmn.xml.

### Configure Data Sources for RAC, FCF

In addition to configuring ONS for FCF, you also have to configure each BPEL Server's DataSources for FCF. This step involves editing the file \$ORACLE\_HOME/j2ee/OC4J\_BPEL/config/data-sources.xml to ensure that in every data-source used by the BPEL Server to connect to a RAC database, (i) the JDBC URL is correct for a RAC database; and (ii) FCF-related configuration properties are set, as in the following example. For more information on configuring DataSources in OC4J, refer to the OC4J Services Guide Chapter 5 [B28958].

```
<connection-pool name="BPELPM_CONNECTION_POOL">
  <connection-factory
    factory-class="oracle.jdbc.OracleDriver"
    url="jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=on)
      (ADDRESS=(PROTOCOL=tcp) (HOST=racnode1.oracle.com) (PORT=1521))
      (ADDRESS=(PROTOCOL=tcp) (HOST=racnode2.oracle.com) (PORT=1521)))
      (CONNECT_DATA=(SERVICE_NAME=some_db)))">
    <property name="connectionCachingEnabled" value="true"/>
    <property name="fastConnectionFailoverEnabled" value="true" />
  </connection-factory>
</connection-pool>
```

Configure JTA integration by editing the  
oc4j-ra.xml files for the AQ and DB Adapter.

### Configure Database and AQ Adapters for JTA Integration

If BPEL processes in the system use AQAdapters or DbAdapters that connect to RAC databases, then to complete the configuration of each BPEL Server for interacting with the RAC databases correctly, you need to configure those adapters for JTA integration. That amounts to editing the <connector-factory> elements in \$ORACLE\_HOME/j2ee/OC4J\_BPEL/application-deployments/default/[Db|Aq]Adapter/oc4j-ra.xml to ensure there exists a <config-property> element with name xADataSourceName, whose value is the JNDI location of a managed-data-source declared in data-sources.xml with a tx-level attribute value of "global" – for example:

```
<config-property name="xADataSourceName"
value="jdbc/BPELServerDataSource" />
```

This ensures that JDBC connections used by the adapters are managed by an underlying OC4J DataSource and participate in JTA transactions associated with the invoking BPEL process's execution. For more information on the DbAdapter oc4j-ra.xml configuration property and its relationship to data-source configuration in data-sources.xml, see Section 4.5.5.3 of [B28994].

### Configure BPEL Parameters

Aside from configuring ONS, DataSources, and adapters to interact with RAC correctly in an HA BPEL architecture, a couple of other configuration steps remain, having to do with parameters of BPEL Process Manager itself.

Set configuration parameters in Admin  
Console for cluster URLs, and DB retries.

First, as mentioned above in the installation instructions, if you're using an HTTP load balancer in front of your BPEL tier, you need to set the soapServerUrl and

soapCallbackUrl parameters in each BPEL node's Admin Console configuration tab to be the load balancer's URL.

Second, in the same UI, you might wish to consider increasing the value of the nonFatalConnectionMaxRetry parameter from its default of two to, say, ten. This parameter controls the number of times the BPEL engine automatically retries execution of a process instance upon detection of a "RAC node down" condition. But user beware: execution is retried from last message arrival for that process instance, which means there could be multiple invocations of non-idempotent endpoints if the process in question happens to invoke any.

## PROCESS DESIGN

**Clustering adds a dimension of complexity, which impacts BPEL process design.**

Astute BPEL system architects will appreciate that a clustered deployment architecture (whether for high availability or scalability) contributes an additional dimension of complexity to the behavioral characteristics of a BPEL system. The impact of clustering must be taken into account in the design of the BPEL processes deployed to the cluster, particularly with respect to how load is balanced over the cluster given the myriad ways in which BPEL processes can be initiated.

The load balancing approach per process initiation method is not the only impact of high availability on process design. Other design considerations involve virtualizing endpoints and designing resilience to their failures; and understanding the consequences of resilience to dehydration store failures. The following subsections examine these process design impacts in greater detail.

### Initiation Methods and Load Balancing

**Different BPEL process initiation methods need different load balancing approaches.**

**In practice, SOAP over HTTP is rarely the only process initiation method used.**

When all BPEL process invocation in a system occurs via SOAP invocation over HTTP, everything is nice and neat from a load-balancing perspective: some kind of load balancer (whether hardware or software) sits in front of the cluster of BPEL Servers and distributes the incoming HTTP requests over the cluster using some configured algorithm. However the BPEL systems that customers build are rarely so idyllic or simplistic. Rather they tend to contain processes initiated in any or all of the following ways:

- Through enterprise messaging adapters, on detection of messages in queues;
- Through database adapters, on detection of rows in tables;
- Through file/FTP adapters, on detection of files in filesystems;
- Through packaged applications, by a variety of means.

For these methods of process initiation, an HTTP load balancer is irrelevant, and different approaches are taken to balancing load over the BPEL nodes in a cluster. While Table 2 summarizes load balancing approach by process initiation method, the following subsections discuss the details.

**Table 2: Load Balancing Approach by Process Initiation Method**

Initiation Method	Load Balancing Approach
SOAP over HTTP	Fronting HW/SW load balancer
MOM message consumption	Competing Consumers pattern [Hohpe]
Polling DB Adapter	Distributed polling strategy [B28994]
Polling File/FTP Adapter	Extra Active-passive BPEL tier for Dispatcher [Hohpe]
Applications Adapter	Some of the above

### Message Consumption

For BPEL processes initiated by message receipt from inbound AQ, JMS, or MQ adapters using single-consumer queues, a straightforward and clean solution exists for the problem of how to balance the load over the cluster. That solution is called the Competing Consumers pattern, and is described by Gregor Hohpe and Bobby Woolf in their book Enterprise Integration Patterns [Hohpe]. Basically Competing Consumers is what happens if you just let each node in the cluster attempt to dequeue and process the same message from the same queue – only one of them will succeed transactionally, and the others will just wait for another message. This approach requires that all consuming nodes connect to, and attempt to dequeue from, the same shared message destination or the illusion thereof, but that requirement can be met for all of the enterprise messaging systems supported by Oracle’s enterprise messaging adapters.

However if the AQ, JMS, or MQ adapters are using multi-consumer queues (i.e., JMS Topics, a publish-subscribe model), then clustering introduces added complexity in that multiple redundant process instances could be initiated on the cluster nodes as a result of the publication of a single message. That is probably not desired behavior, so readers beware.

### Database Polling

When a BPEL process is initiated by a polling database adapter, balancing load across a clustered deployment of that process becomes more complicated than for the message consumption case. In this case one must resort to the “distributed polling strategy” documented in the Adapters User’s Guide [B28994] Section 4.5.6.10. For BPEL Process Manager version 10.1.3.1 there have been significant enhancements in the implementation of the distributed polling strategy which not only dramatically improve performance, but at the same time reduce the probability of lock contention or deadlock at the database level.

### Filesystem Polling

Balancing load across a clustered deployment of a BPEL process initiated by a polling file or FTP adapter is incrementally more impactful than the preceding two cases. For starters, it’s not really possible to achieve an active-active clustered deployment of such a process, because attempting that would lead to race conditions as mentioned in the Deployment Architecture section of this paper.

**For processes initiated by DB Adapter, clustering requires a distributed polling strategy.**

The best that can be achieved (from an HA perspective) is an active-passive “failover” cluster.

So, what to do if the processing downstream of the initiating file or FTP adapter can’t be a single point of failure, or can’t scale to the load with only one active BPEL Server? The answer is to partition the processing: the process that reads the file should do little more than to “hand off” records from the file to a second BPEL process that does the “real” work. The hand-off could utilize an enterprise messaging system, a database schema, even SOAP over HTTP.

**For processes initiated by File/FTP Adapter, deploy a separate tier for Dispatcher processes.**

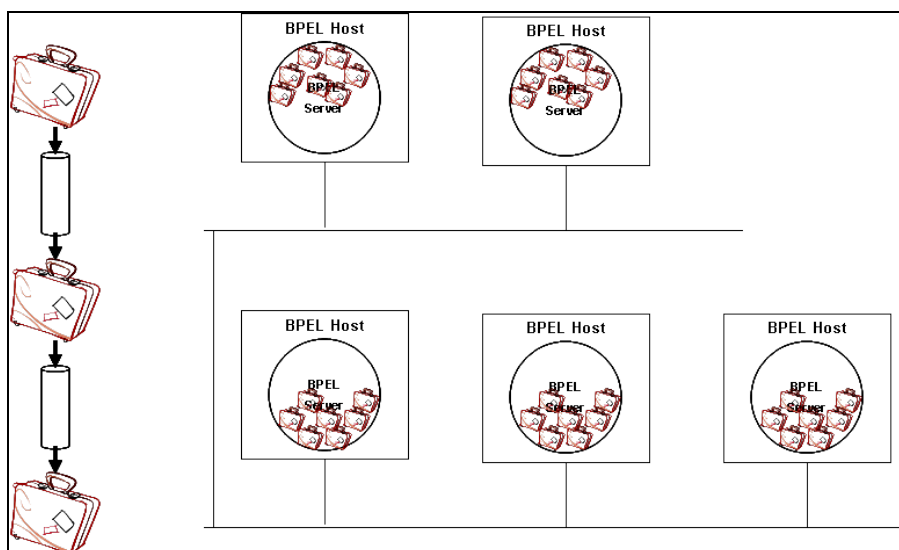
The advantage to such partitioning is that the “real” downstream process can truly be deployed across an active-active cluster, thereby increasing availability and scalability. The front-end file-reading process becomes a Dispatcher [Hohpe], and of course the system architect could choose a hand-off technology for which load balancing is straightforward (e.g. JMS or HTTP).

Finally, with such a partitioning, the Dispatcher process could be deployed to its own tier of BPEL nodes configured as an active-passive failover cluster of minimal size for the job, while the “real” processing could be deployed to a cluster configured and tuned appropriately for its requirements. Figure 4, following, illustrates that concept.

**These architecture considerations are best not grafted onto a system at a late stage.**

**They need to be thought about up front, in early architecture planning stages.**

Obviously these sorts of system architecture considerations are best not grafted onto a system at a late stage: these are the kinds of things that need to be thought about up front, in early architecture planning stages. The file / FTP adapter case in particular underscores the point that quality attributes (availability, scalability) strongly influence deployment architecture which in turn has considerable impact on BPEL process design. Parsing and transforming and passing XML documents through a business process flow that orchestrates service endpoints, is only one dimension of the problem. Furthermore it’s clear that shaping a deployment architecture could benefit from some ability in capacity planning for BPEL systems. But discussing capacity and scalability of BPEL systems in general is made difficult by their context-dependent uniqueness, and remains a topic for another paper.



*Figure 4: Separate Tiers for Different Stages of a "Process Pipeline"*

#### Packaged Applications

For processes initiated by packaged apps, the HTTP, messaging, or file load-balancing approaches may suffice.

Finally we come to the matter of load balancing across a clustered deployment of a BPEL process initiated by a packaged application adapter. This represents a frontier of knowledge on HA BPEL architecture. In the case of the Oracle Applications Adapter, since two of the interfaces it supports (the Business Event System and the XML Gateway) are based on AQ internally, it's possible that a clustered deployment of an Oracle Applications Adapter –initiated process using those interfaces would exhibit Competing Consumers behavior. In the case of the SAP Adapter, the SAP Gateway Server implements load balancing over different connecting SAP Adapter instances [ATN009]. In the case of the Seibel Adapter, Seibel can publish “integration objects” through MQSeries, HTTP, and files, so the load balancing approaches for those process initiation methods apply [ATN011]. In the case of adapters for other packaged applications, more knowledge needs to be collected and captured in a future revision of this document.

#### Endpoint Virtualization

Endpoint virtualization is an important principle in design of HA BPEL systems.

As discussed in the introduction, there is some probability that a service endpoint invoked by a BPEL process may not be highly available itself. But, in the best case, a BPEL process can invoke web services that are redundant in their implementation, and virtual in their endpoint URL. This virtualization is an important principle in the design of highly available BPEL systems and BPEL processes: BPEL process implementors and administrators should strive to use virtualized locations for the services on which their processes depend, and should symmetrically publish only virtualized locations for the services or processes they're implementing. For in the words of Klaus Schmidt:



We need virtualization to implement redundancy. ... When we introduce redundant components into an architecture, we do it to cope with potential failures of one component. This means that our component usage must not be bound to a specific instance, otherwise outage of that instance would render our complete system unusable. Instead, a virtual component must be created to conceal the real thing. [Schmidt p.78]

**Other Oracle SOA Suite components can be used to virtualize third-party endpoints from BPEL's perspective.**

This raises the question of how endpoints can be virtualized. If the endpoint in question is a BPEL process initiated by SOAP over HTTP, the answer is simple: in an HA deployment architecture fronted by a load balancer, the endpoint URL is virtualized by using the load balancer's URL. At the other end of the spectrum, if the endpoint in question is a third-party service whose identifier reflects a particular instance, the implementor of a BPEL process depending on that service may not have any influence over the identifier and binding of the service. With Oracle Application Server 10.1.3.1, however, there is an emerging pattern of using other SOA Suite components (e.g. ESB, OWSM, and Service Registry) in conjunction with BPEL for exactly this purpose: virtualizing endpoints and thereby protecting BPEL processes from variation in the binding of services on which they depend (see "Protected Variations" in [Cockburn]).

### **Resilience to Endpoint Failures**

**Requirements should specify how BPEL systems will respond to endpoint failure.**

**The standard HA strategy is to retry the operation against a redundant component.**

Even if an endpoint is virtualized and redundantly implemented, there is a possibility that an instance of it could fail while doing work on behalf of a client (e.g. a BPEL process) that is awaiting a response. The failure might cause some sort of fault to be returned to the client, or it might cause the client to time out while waiting for a response. In either case, the analyst and architect of the client (BPEL) system have to decide how the system should respond to the failure. Especially in cases where it is known that the component on which the system depends is redundantly implemented, the standard strategy is to retry the operation that failed, against a redundant instance of the component. In fact this is the exact strategy implemented by the Oracle JDBC driver's Fast Connection Failover feature – on detection of RAC node failure, applications are expected to retry transactions using newly-acquired connections [B14355, p.27-7], whose JDBC URLs virtualize the RAC database.

Implementing that strategy makes a system resilient to failures in things it depends on – which may or may not be appropriate for a given system; there is longstanding debate in HA circles on the appropriate location (farther upstream or farther downstream) for "retry logic". The professional literature offers some guidance – see, for example, Chapter 8 ("Designing Reliable Collaborations") of Rebecca Wirfs-Brock's excellent Object Design, and the works of Alistair Cockburn cited from there [Wirfs]. Ultimately it is a matter of requirements specification how a system should respond to a failure in a component upon which it depends – by implementing resilience, or by propagating the failure to its client.

Strictly speaking, making a system more resilient to failure in components on which it depends makes the system more *reliable*, not necessarily more *available*. But it's related closely enough to availability that I'm discussing it here, especially since the failure in a supporting component might have been a sudden unavailability of that component. So, assuming the analyst and architect of an HA BPEL system want to implement resilience to failure of an endpoint a process depends on, the question becomes what mechanisms are available to do so?

**BPEL offers many mechanisms for implementing resilience to endpoint failure.**

It turns out that BPEL Process Manager offers several mechanisms for implementing resilience to failures in endpoints on which processes depend. The first such mechanism is a number of partnerLinkBinding configuration properties in the BPEL process deployment descriptor. Two of these properties are named `retryMaxCount` and `retryInterval`. These properties control the number of retries BPEL Server attempts on a partner link invocation if an invoke fails because of "network problems", and the delay between retries. Oracle consulting personnel have found these properties to behave as expected with DbAdapter invocations against a RAC database in the event of RAC node failure, when the DbAdapter in question is not configured to participate in the current JTA transaction. A third such property is named "timeout", and specifies the number of seconds in which a SOAP invocation times out if there is no response from the invoked service. If this happens, a remote fault is thrown, which can be handled by a fault handler in the invoking BPEL process.

BPEL Process Manager also implements a failover feature in which a partner link can be configured with a list of endpoint locations in the process's deployment descriptor; if an attempt to invoke the first implementation in the list results in a retrievable exception, the next implementation in the list is automatically tried. The details of these partner link features are described in Chapter 8 of the BPEL Process Manager Developer's Guide [B28981].

The final mechanism offered by BPEL Process Manager is its implementation of the fault handling provisions of the BPEL specification. BPEL process activities can be enclosed in scopes which can provide fault handlers for specific faults or all faults, and the fault handlers can sequence arbitrary BPEL process activities, potentially including retry logic. However experience has shown that fault handlers provided by scopes enclosing DBAdapters don't catch faults associated with database unavailability if the DbAdapter is configured to participate in the current JTA transaction.

**Some failure-resilience mechanisms only work when JTA integration isn't configured.**

So it appears that with the current product version, a BPEL system architect must strike a balance between implementing resilience to failures in supporting components, on the one hand, and implementing guaranteed atomicity in distributed transactions associated with BPEL process execution, on the other hand. Like scalability of BPEL systems, and performance of BPEL systems, transaction design within BPEL systems is another serious topic that merits another paper such as this one. Whether transactional correctness is more important than resilience to failure probably varies from system to system, so

analysts and architects should understand the relative priorities of these quality attributes for the system in question, and balance accordingly.

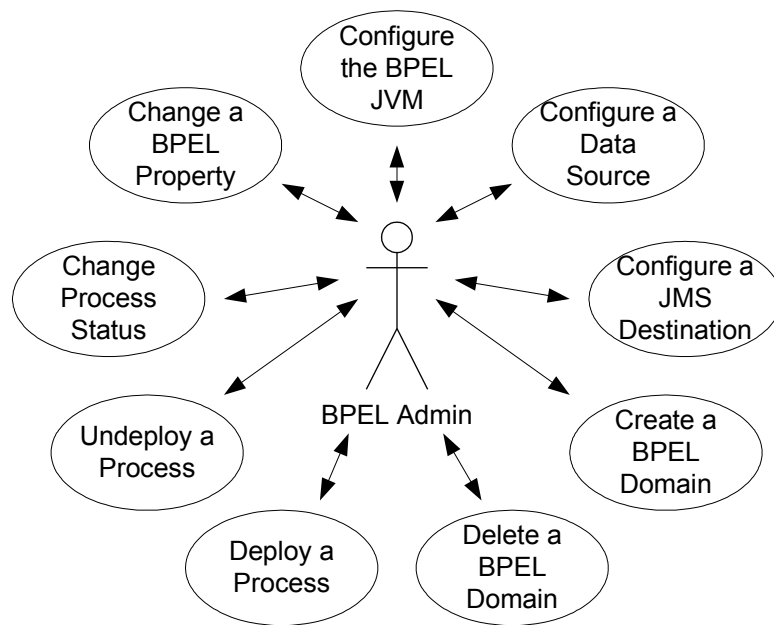
### **Resilience to Dehydration Store Failures**

**BPEL's implementation of RAC node failure tolerance may cause multiple invocations of non-idempotent endpoints.**

A final impact of high availability on BPEL process design, also somewhat in the vein of transaction design, is the consequence of BPEL Server's implementation of resilience to failures in dehydration store RAC nodes. Specifically, BPEL Server provides a configuration parameter named `nonFatalConnectionMaxRetry`, which specifies the maximum number of times BPEL Server will attempt to execute a process instance due to the arrival of a message for it, when the previous attempt encountered a non-fatal database connection error. This scenario happens when, for example, a message arrives for a process instance, execution proceeds successfully, then at the end of execution, dehydration and audit trail insertion fails due to RAC node failure. BPEL Server implements configurable retry behavior to provide resilience in this scenario, but the scope of BPEL Server's retry is large: it re-executes the process instance's flow again, from the activity first triggered by the message's arrival. That results in endpoints being invoked multiple times for the same process instance. If those endpoints are non-idempotent, that may cause inconsistent states amongst the systems being orchestrated by the BPEL process in question. Unless logic can be designed at the BPEL process level to preclude such eventualities, then with current product versions, BPEL system architects may also have to balance RAC node failure tolerance against at-most-once invocation semantics for non-idempotent endpoints.

### **SYSTEM ADMINISTRATION**

Enough about architects – let's turn to administrators and operators, and the impact of HA deployment architectures on the use cases they might typically perform against instances of BPEL systems in the environments they administer. After all, administrators are people too, and important ones, at that.



**Figure 5: Some Use Cases of BPEL Administration**

HA deployment architecture significantly impacts BPEL system administration.

The following subsections categorize and enumerate administration use cases, and describe the impact to those use cases of deployment architectures featuring clustering and other HA-motivated characteristics.

### J2EE Container Configuration

In the category of J2EE container configuration, there are a number of typical use cases that are likely to be performed by a BPEL system administrator. The impact of HA deployment architecture on these use cases is that now every cluster node is affected, and must be kept in sync – whereas in a single-node deployment architecture, the use cases are much simpler to perform. Earlier versions of Oracle Application Server automatically synchronized cluster nodes for some of these configuration changes, but in version 10.1.3.1 changes must be made separately on every node (except changes to data-sources.xml and jms.xml when OC4J groups are used). Table 3, below, tabulates the typical likely J2EE container configuration use cases, and the configuration files they affect. Note that the file names in Table 3 are relative to \$ORACLE\_HOME. Note also the content of these files is modifiable using JMX MBeans, for example through the Enterprise Manager MBean browser.

**Table 3: Files Affected By J2EE Container Configuration Use Cases**

Use Case Name	Configuration Files Affected (Per Node)
Configure an Adapter	<ul style="list-style-type: none"> <li>j2ee/OC4J_BPEL/application-deployments/default/**/oc4j-ra.xml</li> </ul>

Use Case Name	Configuration Files Affected (Per Node)
Configure a Data Source	<ul style="list-style-type: none"> <li>j2ee/OC4J_BPEL/config/data-sources.xml</li> </ul>
Configure a JMS Destination	<ul style="list-style-type: none"> <li>j2ee/OC4J_BPEL/config/jms.xml</li> </ul>
Configure the BPEL JVM	<ul style="list-style-type: none"> <li>opmn/conf/opmn.xml</li> </ul>
Configure ONS	<ul style="list-style-type: none"> <li>opmn/conf/opmn.xml</li> </ul>
Configure Thread Count	<ul style="list-style-type: none"> <li>j2ee/OC4J_BPEL/application-deployments/orabpel/ejb_ob_engine.jar/orion-ejb-jar.xml</li> </ul>
Configure Transaction Timeout	<ul style="list-style-type: none"> <li>j2ee/OC4J_BPEL/config/server.xml</li> <li>j2ee/OC4J_BPEL/application-deployments/orabpel/ejb_ob_engine.jar/orion-ejb-jar.xml</li> </ul>

For most cases in Table 3, the substance of the use case is clear from its name, but the “Configure the BPEL JVM” use case deserves additional comment. This is a very important and frequently-executed use case whose substance involves changing JVM startup options for the OC4J BPEL VM. Typically this is done to change heap settings, pass system properties, enable logging, attach remote agents, etc.

**It is strongly recommended that customers automate environment administration using Ant or scripting languages, and keeping configuration files under version control.**

The fact that all cluster nodes are affected by these use cases, and must be kept in sync, argues strongly for an automated (script-driven) approach to BPEL administration. When these changes are made manually via Console UI, the process of keeping all nodes in sync is too tedious and error-prone. It is my preference and strong recommendation that customers automate environment administration using technologies like Ant or scripting languages, even if that means keeping master copies of the above-tabulated configuration files in a version control system and moving them into place as part of an automated deployment process.

Another impact of HA deployment architecture on this category of use case is caused by the usage of RAC in the database tier of architecture. The usage of RAC necessitates the configuration of Oracle Notification Service (ONS) and Fast Connection Failover (FCF). It also necessitates more complicated database connect descriptors appearing in JDBC URLs. These necessities manifest in the opmn.xml and data-sources.xml configuration files, and represent HA impact that is absent in simpler deployment architectures.

## BPEL Domain Management

The BPEL domain management category of use cases includes the obvious Create a BPEL Domain, and Delete a BPEL Domain, and also the less obvious use case Configure a BPEL Domain. Configuring a BPEL domain is typically accomplished using the BPEL Console UI to set configuration properties, adjust logging levels, etc. For completeness let us also include in this category the use case Configure a BPEL Server, i.e. making configuration changes that are domain-independent, through the BPEL Admin Console. Taken together, such configuration use cases have the potential to affect at least the following ORACLE\_HOME-relative files on a BPEL node:

- `integration/orabpel/domains/<domain>/config/domain.xml`; and
- `integration/orabpel/system/config/collaxa-config.xml`.

**In a clustered deployment architecture, BPEL domain management use cases must be performed on every BPEL node.**

In a single-node deployment architecture, these use cases are again simple because they only have to be performed on a single node. However in an HA deployment architecture featuring a cluster of BPEL nodes, these use cases again have to be performed on every cluster node.

Furthermore it is not clear that the performance of these use cases can be automated via scripting, with the possible exception of Configuring a BPEL Domain or Server by moving the relevant files into place from a version control system.

## **BPEL Process Management**

In the BPEL process management category are found the following fundamentally important and frequently-executed use cases:

- Deploy a BPEL Process;
- Undeploy a BPEL Process; and
- Change Process Status (retire or activate, turn off or on).

In previous versions of Oracle BPEL Process Manager, the record of what processes were deployed to a BPEL installation was stored in that installation's directory structure – and thus deployment of a process to a cluster of BPEL nodes required a separate step to deploy the process to each node. The recommended procedure was to deploy to one node using the BPEL Console, then to manually copy the deployed suitcase jar to the appropriate location on the other nodes.

Starting in version 10.1.3.1 of Oracle BPEL Process Manager, the record of what processes are deployed to a BPEL domain is stored in the dehydration store database. Thus all BPEL nodes using the same dehydration store should share the record of what is deployed to a given domain, and deployment should only need to occur once, to any BPEL node in the cluster, for all nodes to see the deployment. The same storage location change, and once-per-cluster update expectation, also holds for BPEL process status in version 10.1.3.1.

**Explicit process deployment to each cluster node is most reliable (perhaps redundant, but also harmless).**

Because of these changes for version 10.1.3.1, clustering and HA architecture should have no impact, compared to single-node deployment architectures, on BPEL process management use cases in version 10.1.3.1. But in practice, the collective experience has been that the once-per-cluster expectation is not always met satisfactorily, and it becomes necessary to explicitly deploy BPEL processes to each cluster node (which should be redundant, but also appears harmless). Given this finding, clustering and HA architecture do have an impact on these use cases. The recommended deployment approach is to use an Ant script and Oracle's custom BPEL-specific Ant tasks to deploy a BPEL process to each BPEL node in a cluster. This approach facilitates substituting node-specific properties into

suitcase jars at deployment time, but we also understand that *any* difference between two suitcase jars' contents causes BPEL Process Manager to treat them as different processes with different GUIDs.

### **BPEL System Monitoring and Management**

Some use cases in this category are, predictably, Start a BPEL Server, Stop a BPEL Server, and Check BPEL Server Status. Other use cases in this category, having to do with monitoring, are less readily named, but one can imagine monitoring load, performance, and resource utilization within a BPEL system, for example.

The impact of HA architecture, specifically clustering in the BPEL middle tier and also in the database tier, obviously impacts these use cases with a multiplying effect. Now not just one node needs to be monitored and managed, but many nodes. However the impact of HA architecture in this particular area may be softened with the adoption of Oracle Enterprise Manager and its new management pack for SOA Suite.

**In a clustered deployment architecture, monitoring and management is impacted with a multiplying effect.**

### **Message Recovery**

The “recovery”, i.e. re-submittal, of unprocessed SOAP messages is impacted in an interesting way by HA architecture i.e. clustering. Specifically, in a single-node architecture, there is no ambiguity about the BPEL Server to which an unprocessed message will be re-submitted; there is only one to choose from.

In a clustered deployment architecture, however, there are multiple BPEL Servers in which to resubmit unprocessed messages. This causes BPEL system architects to have to select a load-balancing strategy, effectively, for message recovery. All BPEL nodes in a cluster “see” the same recovery queue, but only one of them will dequeue and process a given message, and only when explicitly requested to do so (by UI or API). Especially with automated recovery scripts that re-submit unprocessed messages via API, there is a risk of “flooding” a given node with unprocessed messages. Therefore in a cluster, the automated recovery script, or manual recovery procedure, had better provide for balancing the load of re-submitted messages over the cluster. One way to achieve this is to use ORMI load balancing in conjunction with BPEL PM's Java API.

**In a clustered deployment architecture, message re-submission should be load-balanced.**

### **Dehydration Store Administration**

Finally we come to use cases in the category of dehydration store administration, the most well-known of which is Purge Tenured Data, motivated by controlling database growth. Disaster recovery considerations, however, give rise to additional use cases, such as Synchronize Backup Dehydration Store With Primary, which are absent without HA architecture.

Using RAC in the database tier also potentially gives rise to additional dehydration store administration use cases caused by the adoption of HA architecture. For example it has become apparent with BPEL HA customers that a use case with intent/name Configure Database Initialization Parameters Specially for RAC may

be necessary to ensure correct behavior of the BPEL system. For example the initialization parameter MAX\_COMMIT\_PROPAGATION\_DELAY may need to be set to zero for a RAC database tier and clustered BPEL mid-tier, depending on the design of the deployed BPEL processes.

In summary, we see that a great many use cases of BPEL system administration, in a whole spectrum of categories, are affected by the inclusion of HA architecture in a BPEL system. Administrators and architects need to be aware of these impacts, and consider them when choosing the level of investment in HA architecture for their BPEL systems.

## TESTING AND TROUBLESHOOTING

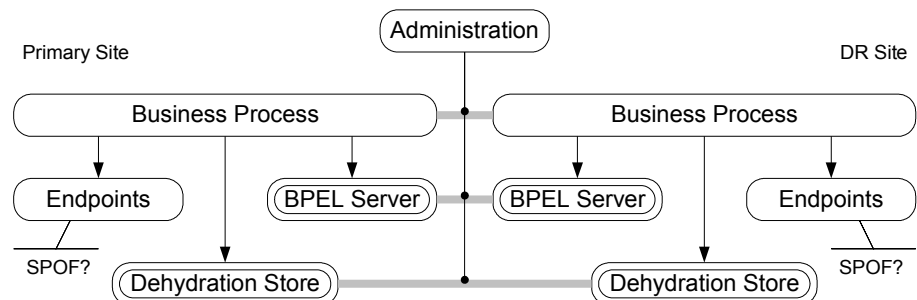
After an organization has chosen a level of investment in HA architecture, selected a deployment architecture, designed BPEL processes accordingly, installed and configured software in various environments, and administered these installations' containers and domains and processes, there comes a moment of truth: will the system so developed actually exhibit high availability? Will it be resilient to failures in its components, and redundant and virtual for the clients and businesses that depend on it?

Rather than finding out these answers during production operations, it is of course better to find them out in testing prior to the commencement of production operations. Therefore this section discusses how to test and troubleshoot HA BPEL systems, and begins with some statements of expected behavior under various failure conditions.

**It is better to discover a system's availability characteristics during testing than during production operations.**

### Failure Conditions and Expected Behavior

The generic HA BPEL system dependency diagram of Figure 1, reproduced below as Figure 6, gives a clear picture of possible failures in a BPEL HA system.



**Figure 6: Generic HA BPEL System Dependency Diagram**



In general there are several major categories of things that can fail:

- A RAC node (or process) in the dehydration store database tier;
- A BPEL node (or Server process) in the middle tier;
- An endpoint upon which a BPEL process depends; and
- An entire site hosting a BPEL system.

For each of these potential failures, the following subsections discuss the expected behavior in the event of such a failure.

#### **RAC Node Failure**

If a dehydration store RAC node should fail, it is expected, under the configuration assumptions documented in the Installation and Configuration section of this paper, that all of the following responses will occur:

- The “Implicit Connection Caches” (i.e. connection pools) maintained by the Oracle JDBC driver used in each BPEL Server VM will be cleaned of connections to the failed RAC node, so that future connection requests will be satisfied only by connections to surviving nodes. This pool management is by virtue of the Fast Connection Failover configuration.
- Any thread in a BPEL Server VM that is executing a BPEL process instance and that attempts to use a connection to the failed RAC node (by executing a statement over that connection) will receive a `SQLException`, discard the connection, acquire a new one, and restart process execution from the last message arrival. The FCF contract mandates discarding and reacquiring connections and retrying transactions against the newly acquired transaction ([B14355], p.27-7). BPEL Server implements this contract by retrying the execution of the BPEL process from the last message arrival. Readers are cautioned that this implementation could result in redundant invocation of non-idempotent endpoints by a BPEL process in the event of RAC node failure.
- If the failed RAC node should come back online, the Implicit Connection Caches in the JDBC drivers of the BPEL Server VMs will re-balance themselves to include connections from the restarted RAC node. This rebalancing behavior is a feature of the Fast Connection Failover capability.

**BPEL responds to RAC node failure by  
retrying process execution from last  
message arrival.**

#### **BPEL Node Failure**

If a BPEL node should fail, the expected behavior of the system under the configuration assumptions is as follows. First, new arriving invocation messages should initiate new BPEL process instances in the surviving BPEL nodes, assuming the implementation of appropriate load-balancing strategies for the various process initiation methods as discussed in the Process Design section of this paper.

**In the event of BPEL node failure, any in-memory state of process instance execution will be lost. Dehydrated process instances in a wait state will continue execution on a surviving node, upon next message arrival.**

**On endpoint failure, a process instance will terminate execution in a faulted state, if no resilience mechanisms are used.**

**The test plan for an HA BPEL system should include various outage tests.**

Second, and this may not match the expectations of customers, the execution of BPEL process instances that were in the middle of execution in memory on the failed BPEL node will ***not*** be automatically continued or resumed on surviving BPEL nodes. The last invocation message or callback message to arrive for such a process instance may be preserved in the dehydration store’s delivery queue, so that manual or scripted recovery can be accomplished. But, again, there is no *automatic* recovery and resubmission of such messages. Readers need to be aware of this expected behavior and plan accordingly. One exception or special case is that, if a running process instance was dehydrated, waiting for a callback message, for example, and the BPEL node that last executed the instance should fail, and then the callback message arrives, the instance’s execution will continue in a surviving BPEL node (note there can be complications if the process has a synchronous interface to its clients).

**Endpoint Failure**

The expected behavior of a BPEL process upon the failure of an endpoint it invokes is very much a function of how the process was designed, and not amenable to categorical a-priori prediction. First there is the question of what type of endpoint it is, despite its web services facade – a technology adapter, a remote SOAP service, a packaged application, etc. Second there is the question of whether the BPEL process designer used any of the mechanisms for implementing resilience to failure in endpoints, as discussed in the Process Design section of this paper. In the default case, where the endpoint is an arbitrary service and the BPEL process designer did not use any resilience mechanisms, it is expected that the process will terminate execution in a faulted state.

One exception to this general rule is the case in which the endpoint in question is a DbAdapter to a RAC database, and the failure in question is a RAC node failure. In this case, the expected behavior is as described in the previous subsection on the failure of a dehydration store RAC node, including the potential for redundant invocation of non-idempotent endpoints.

**Testing and Validating HA BPEL Systems**

The generic HA BPEL system dependency diagram of Figures 1 and 6, and the statements of expected behavior given above, suggest obvious ways to test and validate the behavior of HA BPEL systems. The general outline of an HA BPEL system test plan is to start with a baseline case, in which no failures are injected, load balancing is verified, and correct system behavior is generally verified under nominal conditions. Then there should be a series of test cases in which a different type of failure is injected, and the system is observed for the correct expected behavior in that failure condition. The following table tabulates the outline of such a test plan.

**Table 4: HA BPEL System Test Plan Outline**

Test Case	Scenario	Acceptance Criteria
Baseline Case	Nominal system operation; no failures injected	<b>System behaves as expected when all components are reliable; load balancing operates correctly</b>
Dehydration Store RAC Node Failure Case	Inject dehydration store RAC node failure via shutdown abort	<b>Expect all in-flight process instances to run to completion with possible repeated execution; expect new process instances to run normally</b>
BPEL Node Failure Case	Inject BPEL node failure by killing a BPEL Server process	<b>Expect to lose execution state of in-flight instances in killed node, but be able to recover from their last-arrived message in a surviving node; expect new load to be balanced across surviving nodes</b>
Endpoint Failure Cases	Inject failures of various endpoints	<b>Expect resilience (or not) according as process design</b>

## Troubleshooting HA BPEL Systems

Troubleshooting starts with symptoms, then moves to possible causes and corrective actions.

Troubleshooting of systems can be facilitated with a table relating symptoms to possible problems to corrective action. Table 5, following, attempts to act as such a troubleshooting guide for HA BPEL systems. Over future revisions of this paper, as experience with HA BPEL systems accumulates, this table will grow to collect known troubleshooting solutions.

**Table 5: HA BPEL System Troubleshooting**

Symptom	Possible Problem	Corrective Action
BPEL process instances stuck in “Running” state after dehydration store RAC node failure	FCF not properly configured; stale connections remain in pools	<ol style="list-style-type: none"> <li><b>1. Cancel process instances.</b></li> <li><b>2. Verify correctness of FCF configuration and restart.</b></li> <li><b>3. Re-initiate processes.</b></li> </ol>
<b>Processes don’t receive callback messages, callback messages accumulate in manual recovery queue after BPEL node failure</b>	<b>soapServerUrl and soapCallbackUrl not set to use load balancer’s address</b>	<b>Set soapServerUrl and soapCallbackUrl to use load balancer’s address in every installed BPEL Server</b>

## SUMMARY AND FUTURE WORK

This paper has presented collected knowledge on the subject of architecting BPEL systems for high availability, within the organizational framework of business case, deployment architecture, installation and configuration, process design, administration, and testing and troubleshooting.

Organizations contemplating HA BPEL systems are encouraged to choose an appropriate level of investment in HA architecture before undertaking development and operations. Organizations are further encouraged to carefully consider the BPEL process design aspects of load balancing over a cluster, given different process initiation methods, of virtualizing endpoints, and of whether to implement resilience to component failure, or pass notification of failure to calling clients. Finally organizations are encouraged to plan for the additional complexity involved in administering HA BPEL systems, and are strongly encouraged to execute comprehensive HA-focused system test plans prior to beginning production operations.

A considerable amount of knowledge remains to be collected on the subject of architecting BPEL systems for high availability, and on related subjects. For example separate papers on the quality attributes of correctness (in transaction design), performance, and scalability have already been called for herein. The current frontiers of knowledge are in areas such as HA for packaged application endpoints, HA for the Human Workflow capabilities of BPEL Process Manager, and HA during planned downtimes (e.g. rolling upgrades). Hopefully future revisions of this paper will be able to address those knowledge areas. Finally, the scope of this paper, or one like it, could be expanded to address HA architecture with other SOA Suite components, or across the SOA Suite as a whole.

## REFERENCES

- [ATN009] Oracle Adapter Technical Note #009: SAP Adapter Scalability and security, Meera Srinivasan.
- [ATN011] Oracle Adapter Technical Note #011: Oracle Adapter for Seibel Scalability & HA, Meera Srinivasan.
- [B14203] Oracle Database Oracle Clusterware and Oracle Real Application Clusters Installation Guide, 10g Release 2 (10.2) for Linux, part number 14203-08, September 2006.
- [B14355] Oracle Database JDBC Developer's Guide and Reference, 10g Release 2 (10.2), part number 14355-02, January 2007.
- [B28939] Oracle Application Server Enterprise Deployment Guide, 10g Release 3 (10.1.3.1.0), part number 28939-03, October 2006.
- [B28941] Oracle Application Server High Availability Guide, 10g Release 3 (10.1.3.1.0), part number 28941-03, November 2006.

- [B28958] Oracle Containers for J2EE Services Guide, 10g (10.1.3.1.0), part number 28958-01, September 2006.
- [B28980] Oracle BPEL Process Manager Installation Guide, 10g (10.1.3.1.0) for UNIX and Microsoft Windows, part number 28980-01, September 2006.
- [B28981] Oracle BPEL Process Manager Developer's Guide, 10g (10.1.3.1.0), part number 28981-03, January 2007.
- [B28994] Oracle Application Server Adapters for Files, FTP, Databases, and Enterprise Messaging User's Guide, 10g Release 3 (10.1.3.1.0), part number 28994-01, September 2006.
- [B31005] Oracle Application Server Adapter Concepts Guide, 10g Release 3 (10.1.3.1.0), part number 31005-01, September 2006.
- [B31014] Oracle Application Server Release Notes, 10g Release 3 (10.1.3.1.0), for Linux x86, part number 31014-07, April 2007.
- [Castro] Castro, Fermin, "Oracle Application Server 10g Fast Connection Failover Configuration Guide", October, 2006, [http://www.oracle.com/technology/products/ias/high\\_availability/OracleApplicationServer10gFCF.pdf](http://www.oracle.com/technology/products/ias/high_availability/OracleApplicationServer10gFCF.pdf)
- [Cockburn] Cockburn, Alistair, "Prioritizing Forces in Software Design", in Pattern Languages of Program Design 2, Vlissides et.al., eds., Addison-Wesley 1996.
- [Hohpe] Hohpe, Gregor, and Bobby Woolf, Enterprise Integration Patterns, Addison-Wesley, 2004.
- [Schmidt] Schmidt, Klaus, High Availability and Disaster Recovery, Springer, 2006.
- [Wirfs] Wirfs-Brock, Rebecca, and Alan McKean, Object Design, Addison-Wesley, 2003.



Architecting BPEL Systems for High Availability  
May 2007  
Author: Randy Stafford

Oracle Corporation  
World Headquarters  
500 Oracle Parkway  
Redwood Shores, CA 94065  
U.S.A.

Worldwide Inquiries:  
Phone: +1.650.506.7000  
Fax: +1.650.506.7200  
[oracle.com](http://oracle.com)

Copyright © 2007, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.