Sun HPC ClusterTools™ 8.2.1c Software

User's Guide

Part No. 821-1319-10
April 2010, Revision A

Please
Recycle

Adobe PostScript

Contents

Preface

This manual explains how to use distributed resource management packages for effective resource management and utilization accounting. The following packages work in conjunction with the Open Message-Passing Interface (Open MPI) parallel applications:

- Sun Grid Engine Version 6.1 software
- Altair PBS Professional 9.2
- Cluster Resources Torque 2.3

Before You Read This Book

The *Sun HPC ClusterTools™ 8.2.1c Software Release Notes* includes release note information for the other components in this suite. For information about writing MPI programs, refer to the *Open MPI Software Programming and Reference Guide*. For information about a specific distributed resource management package, refer to the documentation supplied with that package.

For more information about Open MPI and its components, see the Open MPI web site at:

`http://www.open-mpi.org`

Using UNIX Commands

This document might not contain information on basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices.

See one or more of the following for this information:

- Software documentation that you received with your system
- Solaris™ Operating System documentation, which is at

  http://www.sun.com/documentation

Typographic Conventions

Typeface*	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123`	What you type, when contrasted with on-screen computer output	`% su` `Password:`
AaBbCc123	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this. To delete a file, type `rm` *filename*.

* The settings on your browser might differ from these settings.

Shell Prompts

Shell	Prompt
C shell	*machine-name*`%`
C shell superuser	*machine-name*`#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell superuser	`#`

Related Documentation

This book focuses on Open MPI and assumes familiarity with the *MPI Standard*. The following materials provide useful background about using Open MPI and about the

MPI Standard.

Application	Title	Part Number
Sun HPC ClusterTools Software	*Sun HPC ClusterTools 8.2.1c Software Release Notes*	821-1317-10
	Sun HPC ClusterTools 8.2.1c Software Installation Guide	821-1318-10

The Sun HPC ClusterTools documentation is available online at:

http://www.sun.com/documentation

For more information about the Sun HPC ClusterTools software, see the related Web site at:

http://www.sun.com/clustertools

For more information about Open MPI and its components, see the Open MPI web site at:

http://www.open-mpi.org

For more information about Sun Grid Engine software, see the Sun Grid Engine web site at:

http://www.sun.com/software/gridware

Documentation, Support, and Training

Sun Function	URL
Documentation	http://www.sun.com/documentation/
Support	http://www.sun.com/support/
Training	http://www.sun.com/training/

Third-Party Web Sites

Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. Submit comments about this document by clicking the Feedback[+] link at http://www.sun.com.

Please include the title and part number of your document with your feedback:

Sun HPC ClusterTools 8.2.1c Software User's Guide, part number 821-1319-10

CHAPTER 1

Introduction to Sun HPC ClusterTools Software

Sun HPC ClusterTools 8.2.1c software is a set of parallel development tools that extend the Sun network computing solutions to high-end distributed-memory applications. This chapter summarizes its required configuration and principal components. It contains the following sections:

Supported Configurations

Sun HPC ClusterTools 8.2.1c software can be used in any of the following operating system environments:

- Solaris 10 Operation System (Solaris 10 OS)
- Red Hat Linux versions 5 (RHEL)
- SUSE Linux versions 10 (SLES)
- CentOS 5.3 Linux
- OpenSolaris 2009.06

Sun HPC ClusterTools 8.2.1c software supports the following compilers in the Solaris OS and Linux environments:

- **Solaris OS**
  - Sun Studio 10, 11, 12, and 12U1 C, C++, and Fortran
- **Linux**
  - Sun Studio 12U1 C, C++, and Fortran
  - gcc 3.3.3, 3.4.6, 4.1.2
  - Intel 11.0 20081105
  - PGI 7.1-4
  - Pathscale 3.2

Sun HPC ClusterTools 8.2.1c software can run MPI jobs of up to 4096 processes on as many as 1024 nodes. It also provides support for spawning MPI processes.

The Sun HPC ClusterTools 8.2.1c software runs on clusters connected by any TCP/IP-capable interconnect, such as high-speed Ethernet, Gigabit Ethernet, Infiniband, and Myrinet MX.

Open Runtime Environment (ORTE)

Sun HPC ClusterTools 8.2.1c is based on the Open MPI message-passing interface. Open MPI operates using the Open Runtime Environment (ORTE). ORTE starts jobs and provides some status information.

The Open MPI `mpirun` and `mpiexec` commands are actually symbolic links to the `orterun` command. All three commands perform the same function, which is to launch MPI jobs.

ORTE is compatible with a number of other launchers, including rsh/ssh, Sun Grid Engine, PBS Professional, and open PBS (Torque).

Each of ORTE's primary operations is summarized in the sections that follow. Subsequent chapters contain the procedures.

Executing Programs With `mpirun`

Sun HPC ClusterTools 8.2.1c software can start both serial and parallel jobs. The syntax and use of `mpirun` are described in Chapter 5.

Integration With Distributed Resource Management Systems

Sun HPC ClusterTools 8.2.1c software provides integration facilities with three select distributed resource management (DRM) systems. These systems provide proper resource allocation, parallel job control and monitoring, and proper job accounting. They are:

- Sun Grid Engine Version 6.1 and 6.2 software
- Altair PBS Professional 9.2
- Cluster Resources Torque 2.3

Note – Open MPI itself supports other third-party launchers supported by Open MPI, such as SLURM. However, these launchers are currently not supported in Sun HPC ClusterTools software. To use these other third-party launchers, you must download the Open MPI source, compile, and link with the libraries for the launchers.

You can launch parallel jobs directly from these distributed resource management systems. The DRM interacts closely with ORTE for proper resource description and with the multiple processes comprising the requested parallel job.

For a description of the scalable and open architecture of the DRM integration facilities, see "How the Open MPI Environment Is Integrated With Distributed Resource Management Systems" on page 9. For instructions, see Chapter 6.

Open MPI Features

Open MPI is a highly optimized version of the Message Passing Interface (MPI) communications library. It implements all of the *MPI 1.2 Standard* and the *MPI 2.0 Standard*. Its highlights are:

- Integration with the Open Runtime Environment (ORTE)
- Support for MPI I/O
- Seamless use of different network protocols; for example, code compiled on a Sun HPC cluster that has a fast Ethernet network can be run without change on a cluster that has an Infiniband network

- Multiprotocol support so that MPI picks the fastest available medium for each type of connection (such as shared memory, fast Ethernet, Infiniband)
- Communication via shared memory for fast performance
- Optimized collectives for multiprocessors and clusters of multiprocessors
- Full F77, C, and C++ support, and basic F90 support

Debugging With TotalView

TotalView is a third-party multiprocess debugger from TotalView Technologies (formerly Etnus) that runs on many platforms. Support for using the TotalView debugger on Open MPI applications includes:

- Making Sun HPC ClusterTools software compatible with the TotalView debugger
- Allowing Open MPI jobs to be debugged by TotalView using the Sun Grid Engine or the Portable Batch System (PBS)
- Allowing multiple instantiations of TotalView on a single cluster
- Supporting TotalView in Sun HPC ClusterTools software

Refer to the TotalView documentation at `http://www.totalviewtech.com` for more information about using TotalView.

In addition, the Open MPI Frequently Asked Questions (FAQ) contains information about how to use the TotalView debugger with Open MPI. This information is available at:

`http://www.open-mpi.org/faq/?category=running#run-with-tv`

Performance Analysis With Sun Studio Analyzer

Analyzer is a performance analysis tool that is bundled with the Sun Studio compilers. It is available for free download at:

`http://developers.sun.com/sunstudio/downloads`

Analyzer offers the following capabilities:

- Performance profiling of both MPI calls and user application code.
- Tracing of MPI function calls and messages.

- Metrics based on a variety of conditions, such as: run time, hardware counters, memory allocations and leaks, MPI messages and bytes, MPI "wait" and "work" time.
- Attribution of metrics to user functions, callstacks, and source lines.
- Timeline displays of program execution, including message lines.
- Easy-to-use interface to construct general displays of statistical aggregation of performance data.
- Filtering to examine subsets of MPI trace data.

For example, with Analyzer, you could replace the following command line:

```
% mpirun -np 4 ./a.out
```

with the following command lines:

```
% collect -M CT mpirun -np 4 -- ./a.out
% analyzer test.1.er
```

The `-M CT` command turns on MPI tracing for ClusterTools.

Note – For older versions of `collect`, you may specify `-M CT8.2`, even to trace CT8.2.1c runs. Use `collect` without any arguments to see what values of `-M` your version of `collect` recognizes.

To use the `collect` command, you must have the Sun Studio tools in your execution path.

The -- symbol marks the executable that will be launched by `mpirun`.

The `analyzer` command starts up the Analyzer program, including a graphical user interface.

The `test.1.er` argument is the default experiment name generated by the `collect` command.

For more information on using Analyzer with MPI codes, see the web page at:

http://wikis.sun.com/display/MPIAnalyzer/Home

Analyzer documentation is available in Sun Studio manuals and at:

http://developers.sun.com/sunstudio/overview/topics/analyzing.jsp

Communications Failover Feature In Multi-Rail Infiniband Configurations

Sun HPC ClusterTools 8.2.1c software includes a failover feature for handling network failures in multi-rail Infiniband configurations. When a completion error is detected on a given rail, the failover software maps out that rail and routes future traffic through other rails available to the process.

The failover feature supports Open MPI `openib` BTL (Open Fabrics User Verbs) for the communications layer.

Note – The failover feature does not function with uDAPL or IPoIB.

CHAPTER 2

Fundamental Concepts

This chapter summarizes a few basic concepts that you should understand to get the most out of Sun's HPC ClusterTools software. It contains the following sections:

Clusters and Nodes

High performance computing clusters[1] are groups of servers interconnected by any Sun-supported interconnect. Each server in a cluster is called a *node*. A cluster can consist of a single node.

ORTE (Open Run-Time Environment) is the runtime support system for Open MPI that allows users to execute their applications in a distributed clustering environment.

When using ORTE, you can select the cluster and nodes on which your MPI programs will run and how your processes will be distributed among them. For instructions, see Chapter 5.

1. Sun™ Cluster is a completely different technology used for high availability (HA) applications.

For more information about how Open MPI allocates computing resources, see the FAQ entitled “Running MPI Jobs” at:

http://www.open-mpi.org/faq/?category=running

Processes

Open MPI allows you to control several aspects of job and process execution, such as:

- Number of processes to be launched
- Number of available slots on each node
- Process launcher to be used (such as Sun Grid Engine, PBS, rsh/ssh)
- Mapping processes to nodes

How Programs Are Launched

The exact instructions vary from one resource manager to another, and are affected by your Open MPI configuration, but they all follow these general guidelines:

1. You can launch the job either interactively or through a script. Instructions for both are provided in Chapter 5 and Chapter 6.
2. You can enter the DRM processing environment (for example, Sun Grid Engine) before launching jobs with `mpirun`.
3. You can reserve resources for the parallel job and set other job control parameters from within the DRM, or use a hosts file to specify the parameters.

For tasks and instructions, see Chapter 5.

How the Open MPI Environment Is Integrated With Distributed Resource Management Systems

As described in Chapter 1, the Open MPI/Sun HPC ClusterTools 8.2.1c environment provides close integration between ORTE and several different DRM systems, including the following:

- Sun Grid Engine
- Torque
- PBS

The integration process is similar for all DRM systems, with some individual differences. At run time, `mpirun` calls the specified DRM system (launcher), which in turn launches the job.

For information on the ways in which `mpirun` interacts with DRM systems, see Chapter 5. In addition, see the FAQ on running MPI jobs at:

`http://www.open-mpi.org/faq/?category=running`

Chapter 6 provides instructions for script-based and interactive job launching.

Using Sun Grid Engine With ORTE

HPC sites use batch systems to share resources fairly and accountably, and also to guarantee that a job can obtain the resources it needs to run at maximum efficiency. To properly monitor a job's resource consumption, the batch system must be the agent that launches the job.

Sun Grid Engine, like many other batch systems, cannot launch multiple process jobs (such as MPI applications) on its own. In Sun HPC ClusterTools 8.2.1c, ORTE launches the multiple process jobs and sets up the environment required by Open MPI.

When Sun Grid Engine launches a parallel job in cooperation with ORTE, Sun Grid Engine "owns" the resulting launched processes. Sun Grid Engine monitors the resources for these processes, thereby creating a tightly integrated environment for resource accounting. OpenRTE allows users to execute their parallel applications.

Note – There is also an open source version of Grid Engine (GE) hosted on http://www.sunsource.net. Although the Sun HPC ClusterTools 8.2.1c/Open MPI integration is developed with Sun Grid Engine, this integration should work for the open source Grid Engine as well.

Submitting Jobs Under Sun Grid Engine Integration

To submit jobs under Sun Grid Engine integration in Sun HPC ClusterTools 8.2.1c, you must first create a Sun Grid Engine (SGE) environment using `qsub`, `qsh`, and so on. Instructions about how to set up the parallel environment (PE) and queue in Sun Grid Engine are described in the *Sun HPC ClusterTools 8.2.1c Software User's Guide.*

There are two ways to submit jobs under Sun Grid Engine integration: interactive mode and batch mode. "Running Parallel Jobs in the Sun Grid Engine Environment" on page 53 explains how to submit jobs in both modes in the Sun Grid Engine environment.

MCA Parameters

Open MPI provides MCA (Modular Component Architecture) parameters for use with the `mpirun` command. These parameters and their values direct `mpirun` to perform specified functions. To specify an MCA parameter, use the `-mca` flag and the parameter name and value with the `mpirun` command.

For more information about how to use MCA parameters, see Chapter 7.

How ORTE Works With Zones in the Solaris 10 Operating System

The Solaris 10 Operating System (Solaris 10 OS) enables you to create secure, isolated areas within a single instance of the Solaris 10 OS. These areas, called *zones*, provide secure environments for running applications. Applications that execute in one zone cannot monitor or affect activity in another zone. You can create multiple non-global zones to run as virtual instances of the Solaris OS on the same hardware.

The *global zone* is the default zone for the Solaris system. You install Sun HPC ClusterTools software into the global zone. Any non-global zones running under that Solaris system "inherit" that installation. This means that you may install and configure Sun HPC ClusterTools and compile/run/debug your programs in either a global or a non-global zone.

Note – The non-global zones do not inherit the links set up in the global zone. This means that you must type out the full path to the Sun HPC ClusterTools executables on the command line.

CHAPTER 3

Setting Up Your Environment

This chapter provides miscellaneous information about the runtime environment that you should know before you begin to use it. It contains the following sections:

Prerequisites

If your program uses Sun HPC ClusterTools components, compile and link it on a cluster that contains the Sun HPC ClusterTools software.

It is strongly suggested that you use the Sun HPC ClusterTools 8.2.1c compiler wrapper to compile applications. These compiler wrappers add the appropriate compiler and linker flags to the command line and call the underlying compiler and linker for you. Compiler wrappers are available for C, C++, Fortran 77, and Fortran 90.

For more information about compiling MPI applications, see the "Compiling MPI Applications" FAQ at:

`http://www.open-mpi.org/faq/?category=mpi-apps`

Command and Man Page Paths

Sun HPC ClusterTools commands typically reside in the following directories:

- Solaris OS: `/opt/SUNWhpc/HPC8.2.1c/sun/bin`

- Linux:
  - `/opt/SUNWhpc/HPC8.2.1c/gnu/bin` for the GNU compiled version
  - `/opt/SUNWhpc/HPC8.2.1c/sun/bin` for the Sun Studio compiled version
  - /opt/SUNWhpc/HPC8.2.1c/pgi/bin for the PGI compiled version
  - /opt/SUNWhpc/HPC8.2.1c/intel/bin for the Intel compiled version
  - /opt/SUNWhpc/HPC8.2.1c/pathscale/bin for the Pathscale compiled version

You can run the Sun HPC ClusterTools software from the directory in which your ClusterTools commands are installed, or you may add the directory to your PATH, or set the PATH environment variable.

The man pages for Sun HPC ClusterTools commands reside in `/opt/SUNWhpc/HPC8.2.1c/`*compiler-name*`/man`.

Note – The path examples in this manual refer to the default location for the Solaris and Sun Studio compiled Linux binaries. Be certain to use the path name that corresponds to your operating system and compiled version of ClusterTools.

Setting Up Your Path

You should set your path to include Sun HPC ClusterTools commands. If they are in a nonstandard location, you also need to set the `OPAL_PREFIX` environment variable. If you are uncertain whether this step is needed, setting the `OPAL_PREFIX` environment variable does no harm if the commands are in the standard location.

Use the following command line format for a standard installation on a system running the Solaris OS:

```
% setenv PATH /opt/SUNWhpc/HPC8.2.1c/sun/bin:${PATH}
```

The `setenv` command prefixes the `PATH` on both the local and remote hosts with `/opt/SUNWhpc/HPC8.2.1c/sun/bin`.

Use the following command line format for a standard installation on a system running a GNU-compiled Linux OS:

```
% setenv PATH /opt/SUNWhpc/HPC8.2.1c/gnu/bin:${PATH}
```

The `setenv` command prefixes the `PATH` on both the local and remote hosts with `/opt/SUNWhpc/HPC8.2.1c/gnu/bin`.

Use the following command line format for systems that have a nonstandard installation. In this example, the Sun HPC ClusterTools software is located in `/my/clustertools/installation`:

```
% setenv OPAL_PREFIX /my/clustertools/installation
% setenv PATH $OPAL_PREFIX/bin:$(PATH)
```

Core Files

Core files are produced as they normally are in the Solaris environment. However, if more than one process dumps core in a multiprocess program, the resulting core file may be overwritten in the same directory. Use `coreadm`(1M) to control the naming and placement of core files.

To disable the core dump, use the `limit`(1) command. You can use the following command in the C shell:

```
% limit coredumpsize 0
```

CHAPTER 4

Compiling MPI Programs

This chapter describes the compilers that Sun HPC ClusterTools Software supports for both the Solaris OS and Linux. In addition, it describes changes you might make in your application code to recompile and run programs developed with a previous version of Sun HPC ClusterTools software in Sun HPC ClusterTools 8.2.1c.

This chapter contains the following topics:

If you previously compiled your application with the `tmcc` compiler, then you must recompile your applications using the `mpicc` compiler if you want them to be compatible with Sun HPC ClusterTools 8.2.1c software.

Supported Compilers

For the Solaris OS, Sun HPC ClusterTools 8.2.1c software supports Sun Studio 10, 11, 12, and 12 U1 C, C++, and Fortran compilers.

For the Linux OS, the ClusterTools 8.2.1c software supports

- Sun Studio 12 U1 compilers
- `gcc` Linux 3.3.3, 3.4.6, and 4.1.2 compilers
- Intel 11.0 20081105 compiler
- PGI 7.1-4 compiler
- Pathscale 3.2 compiler

Using the Compiler Wrappers

Sun HPC ClusterTools 8.2.1c supplies compiler wrappers for you to use instead of directly calling the compilers when compiling applications for use with the Sun HPC ClusterTools 8.2.1c software. These compiler wrappers do not actually perform the compilation and linking steps themselves, but they add the appropriate compiler and linker flags and call the compiler and linker.

Note – Using the compiler wrappers is strongly suggested. If you decide not to use them, the Open MPI Web site at `http://www.open-mpi.org` contains instructions about how to compile without using them.

The following compiler wrappers are available:

TABLE 4-1 Compiler Wrappers

Language	Compiler Wrapper
C	`mpicc`
C++	`mpiCC`, `mpicxx`, or `mpic++` (Note: `mpiCC` is for use on case-sensitive file systems only)
Fortran 77	`mpif77`
Fortran 90	`mpif90`

For more information about the compiler wrappers, their use, and troubleshooting, see the Open MPI FAQ at:

`http://www.open-mpi.org/faq/?category=mpi-apps#cant-use-wrappers`

Using Non-Default Error Handlers

In Open MPI, the non-default error handler does not persist, and the default error handler is used. This causes any call used after MPI_Finalize to be aborted.

Compiling Fortran 90 Programs

When you are compiling MPI programs written in Fortran 90, you must use the `-xalias=actual` switch. Otherwise, your program could fail.

This condition is due to a known condition in the MPI standard. The standard states that "The MPI Fortran binding is inconsistent with the Fortran 90 standard in several respects." Specifically, the Fortran 90 compiler could break MPI programs that use non-blocking operations.

For more information about this issue, see

`http://www-unix.mcs.anl.gov/mpi/mpi-standard/mpi-report-2.0/node19.htm#Node19`

CHAPTER 5

Running Programs With the `mpirun` Command

This chapter describes the general syntax of the `mpirun` command and lists the command's options. This chapter also shows some of the tasks you can perform with the `mpirun` command. It contains the following sections:

Note – The `mpirun`, `mpiexec`, and `orterun` commands all perform the same function, and they can be used interchangeably. The examples in this manual all use the `mpirun` command.

About the `mpirun` Command

The `mpirun` command controls several aspects of program execution in Open MPI. `mpirun` uses the Open Run-Time Environment (ORTE) to launch jobs. If you are running under distributed resource manager software, such as Sun Grid Engine or PBS, ORTE launches the resource manager for you.

If you are using `rsh/ssh` instead of a resource manager, you must use a hostfile or host list to identify the hosts on which the program will be run. When you issue the `mpirun` command, you specify the name of the hostfile or host list on the command line; otherwise, `mpirun` executes all the copies of the program on the local host, in round-robin sequence by CPU slot. For more information about hostfiles and their syntax, see "Specifying Hosts By Using a Hostfile" on page 27.

Both MPI programs and non-MPI programs can use `mpirun` to launch the user processes.

Some example programs are provided in the `/opt/SUNWhpc/HPC8.2.1c/sun/examples` directory for you to compile and run as sanity tests.

Syntax for the `mpirun` Command

The following example shows the general single-program, multiple data (SPMD) syntax for `mpirun`:

```
% mpirun [options] [program–name]
```

For example:

```
% mpirun -np x program-name
```

For an MPMD (Multiple Program, Multiple Data) application, the command syntax appears similar to the following:

```
% mpirun [options] [program-name] : [options2] [program-name2] ...
```

For example:

```
% mpirun -np x program1 : -np y program2
```

This command starts *x* number of copies of the program *program1*, and then starts *y* copies of the program *program2*.

mpirun Options

The *options* control the behavior of the mpirun command. They might or might not be followed by arguments.

Caution – If you do not specify an argument for an option that expects to be followed by an argument (for example, the --app <*filename*> option), that option will read the next option on the command line as an argument. This might result in inconsistent behavior.

Use the mpirun -h (or --help) option to see a complete list of supported options.

```
% mpirun -h
```

Using Environment Variables With the mpirun Command

Use the -x *args* option (where *args* is the environment variable(s) you want to use) to specify any environment variable you want to pass during runtime. The -x option exports the variable specified in *args*. If no value is specified on the mpirun command line, the value is inherited from the current environment. For example:

```
% setenv DISPLAY myworkstation:0
% mpirun -x DISPLAY -x LD_LIBRARY_PATH=/opt/SUNWhpc/HPC8.2.1c/sun/lib -np 4
a.out
```

Using MCA Parameters With the mpirun Command

The mpirun command uses MCA (Multiple Component Architecture) parameters to pass environment variables. To specify an MCA parameter, use the -mca option with the mpirun command, and then specify the parameter type, the parameter you want to pass as an environment variable, and the value you want to set. For example:

```
% mpirun --mca mpi_show_handle_leaks 1 -np 4 a.out
```

This sets the MCA parameter mpi_show_handle_leaks to the value of 1 before running the program named a.out with four processes. In general, the format used on the command line is --mca *parameter_name value*.

Note – There are multiple ways to specify the values of MCA parameters. This chapter discusses how to use them from the command line with the `mpirun` command. MCA parameters are discussed in more detail in Chapter 7.

`mpirun` Command Examples

The examples in this section show how to use the `mpirun` command options to specify how and where the processes and programs run.

TABLE 5-1 shows the process control options for the `mpirun` command. The procedures that follow the table explain how these options are used and show the syntax for each.

TABLE 5-1 Program/Process Control Options

Task	mpirun option
To run a program with default settings	(no need to specify an option)
To run multiple parallel processes	-c or –np <number of processes>
To display command help	-h or --help
To change the working directory	-wdir or --wdir <directory>
To specify the list of hosts on which to invoke processes	`-host` or `--host` or `-H` <*host1*, *host2*, . . . , *hostN*>
To specify the list of hosts on which to execute the program (also known as the rankmap file)	`-hostfile` <*filename*> or `--hostfile` <*filename*> or `-machinefile` <*filename*> or `--machinefile` <*filename*>
To start up in debugging mode	`--debug` or `-debugger` or `--debugger` <*sequence*>
To specify verbose output	`-v`
To specify multiple executables	`-np 2 exe1 : -np 6 exe2`
To bind processes to processor sockets	`--bind-to-socket --bysocket`

▼ To Run a Program With Default Settings

- **To run the program with default settings, enter the command and program name, followed by any required arguments to the program:**

```
% mpirun program–name
```

▼ To Run Multiple Processes

By default, an MPI program started with `mpirun` runs as one process.

- **To run the program as multiple processes, use the `-np` option:**

```
% mpirun -np process–count program–name
```

When you request multiple processes, ORTE attempts to start the number of processes you request, regardless of the number of CPUs available to run those processes. For more information, see "Oversubscribing Nodes" on page 28.

▼ To Direct `mpirun` By Using an Appfile

You can use a type of text file (called an appfile) to direct `mpirun`. The appfile specifies the nodes on which to run, the number of processes to launch on each node, and the programs to execute in a parallel application. When you use the `--app` option, `mpirun` takes all its direction from the contents of the appfile and ignores any other nodes or processes specified on the command line.

For example the following shows an appfile called `my_appfile`:

```
# Comments are supported; comments begin with #
# Application context files specify each sub-application in the
# parallel job, one per line. The first sub-application is the 2
# a.out processes:
-np 2 a.out
# The second sub-application is the 2 b.out processes:
-np 2 b.out
```

- **To use the `--app` option with the `mpirun` command, specify the name and path of the appfile on the command line. For example:**

```
% mpirun --app my_appfile
```

This command produces the same results as running `a.out` and `b.out` from the command line.

Mapping MPI Processes to Nodes

When you issue the `mpirun` command from the command line, ORTE reads the number of processes to be launched from the `-np` option, and then determines where the processes will run.

To determine where the processes will run, ORTE uses the following criteria:

- Available hosts (also referred to as nodes), specified by a hostfile or by the `--host` option
- Scheduling policy (round-robin or by-slot)
- Default and maximum numbers of slots available on each host
- ORTE also checks to see whether the current environment/shell runs with any third-party launcher (such as Sun Grid Engine or PBS) to determine where the processes will launch.

Specifying Available Hosts

You specify the available hosts to Open MPI in three ways:

- Through the batch scheduler in your resource management software. This option is described in detail in Chapter 6.
- By using a hostfile with the `--hostfile` option. The hostfile is a text file that contains the names of hosts, the number of available slots on each host, and the maximum slots on each host.
- By using the `--host` option. Use this option to specify which hosts to include or exclude.

Specifying Hosts By Using a Hostfile

The hostfile lists each node, the available number of slots, and the maximum number of slots on that node. For example, the following listing shows a simple hostfile:

```
node0
node1 slots=2
node2 slots=4 max_slots=4
node3 slots=4 max_slots=20
```

In this example file, `node0` is a single-processor machine. `node1` has two slots. `node2` and `node3` both have 4 slots, but the values of `slots` and `max_slots` are the same (4) on `node2`. This disallows the processors on `node2` from being oversubscribed. The four slots on `node3` can be oversubscribed, up to a maximum of 20 processes.

When you use this hostfile with the `--nooversubscribe` option (see "Oversubscribing Nodes" on page 28), `mpirun` assumes that the value of `max_slots` for each node in the hostfile is the same as the value of `slots` for each node. It overrides the values for `max_slots` set in the hostfile.

Open MPI assumes that the maximum number of slots you can specify is equal to infinity, unless explicitly specified. Resource managers also do not specify the maximum number of available slots.

Note – Open MPI includes a commented default hostfile at `/opt/SUNWhpc/HPC8.2.1c/sun/etc/openmpi-default-hostfile`. Unless you specify a different hostfile at a different location, this is the hostfile that OpenMPI uses. It is empty by default, but you may edit this file to add your list of nodes. See the comments in the hostfile for more information.

Specifying Hosts By Using the `--host` Option

You can use the --host option to `mpirun` to specify the hosts you want to use on the command line in a comma-delimited list. For example, the following command directs `mpirun` to run a program called a.out on hosts a, b, and c:

```
% mpirun -np 3 --host a,b,c a.out
```

Open MPI assumes that the default number of slots on each host is one, unless you explicitly specify otherwise.

▼ To Specify Multiple Slots Using the `--host` Option

To specify multiple slots with the `-host` option for each host repeat the host name on the command line for each slot you want to use. For example:

```
% mpirun -host node1,node1,node2,node2 ...
```

If you are using a resource manager such as Sun Grid Engine or PBS, the resource manager maintains an accurate count of available slots.

Excluding Hosts From Scheduling By Using the `--host` Option

You can also use the `--host` option in conjunction with a hostfile to exclude any nodes not explicitly specified on the command line. For example, assume that you have the following hostfile called `my_hosts`:

```
a slots=2 max_slots=20
b slots=2 max_slots=20
c slots=2 max_slots=20
d slots=2 max_slots=20
```

Suppose you issue the following command to run program `a.out`:

```
% mpirun -np 1 --hostfile my_hosts --host c a.out
```

This command launches one instance of `a.out` on host c, but excludes the other hosts in the hostfile (a, b, and d).

Note – If you use these two options (`--hostfile` and `--host`) together, make sure that the host(s) you specify using the `--host` option also exist in the hostfile. Otherwise, `mpirun` exits with an error.

Oversubscribing Nodes

If you schedule more processes to run than there are available slots, this is referred to as *oversubscribing*. Oversubscribing a host is not suggested, as it might result in performance degradation.

`mpirun` has a `--nooversubscribe` option. This option implicitly sets the `max_slots` value (maximum number of available slots) to the same value as the `slots` value for each node, as specified in your hostfile. If the number of processes

requested is greater than the `slots` value, `mpirun` returns an error and does not execute the command. This option overrides the value set for `max_slots` in your hostfile.

For more information about oversubscribing, see the following URL:

http://www.open-mpi.org/faq/?category=running#oversubscribing

Scheduling Policies

ORTE uses two types of scheduling policies when it determines where processes will run:

- By slot (default). This scheme schedules processes to run on each successive slot on one host. When all those slots are filled, scheduling begins on the next host in the hostfile.
- By node. In this scheme, Open MPI schedules the processes by finding the first available slot on a host, then the first available slot on the next host in the hostfile, and so on, in a round-robin fashion.

Scheduling By Slot

This is the default scheduling policy for Open MPI. If you do not specify a scheduling policy, this is the policy that is used.

In by-slot scheduling, Open MPI schedules processes on a node until all of its available slots are exhausted (that is, all slots are running processes) before proceeding to the next node. In MPI terms, this means that Open MPI tries to maximize the number of adjacent ranks in MPI_COMM_WORLD on the same host without oversubscribing that host.

▼ To Specify By-Slot Scheduling

If you want to explicitly specify by-slot scheduling for some reason, there are two ways to do it:

1. Specify the `--byslot` option to `mpirun`. For example, the following command specifies the `--byslot` and `--hostfile` options:

```
% mpirun -np 4 --byslot --hostfile myfile a.out
```

   The following example uses the -host option:

```
% mpirun -np 4 --byslot -host node0,node0,node1,node1 a.out
```

2. Set the MCA parameter `rmaps_base_schedule_policy` to the value `slot`. For example:

```
% mpirun --mca rmaps_base_schedule_policy slot -np 4 a.out
```

Note – The examples in this chapter set MCA parameters on the command line. For more information about the ways in which you can set MCA parameters, see Chapter 7. In addition, the Open MPI FAQ contains information about MCA parameters at the following URL:

`http://www.open-mpi.org/faq/?category=tuning#setting-mca-params`

The following output example shows the contents of a simple hostfile called `my-hosts` and the results of the `mpirun` command using by-slot scheduling.

```
% cat my-hosts
node0 slots=2 max_slots=20
node1 slots=2 max_slots=20
% mpirun --hostfile my-hosts -np 8 --byslot hello | sort
Hello World I am rank 0 of 8 running on node0
Hello World I am rank 1 of 8 running on node0
Hello World I am rank 2 of 8 running on node1
Hello World I am rank 3 of 8 running on node1
Hello World I am rank 4 of 8 running on node0
Hello World I am rank 5 of 8 running on node0
Hello World I am rank 6 of 8 running on node1
Hello World I am rank 7 of 8 running on node1
```

Scheduling By Node

In by-node scheduling, Open MPI schedules a single process on each node in a round-robin fashion (looping back to the beginning of the node list as necessary) until all processes have been scheduled. Nodes are skipped once their default slot counts are exhausted.

▼ To Specify By-Node Scheduling

There are two ways to specify by-node scheduling:

- Specify the `--bynode` option to `mpirun`. For example:

```
% mpirun -np 4 --bynode --hostfile my-hosts a.out
```

- Set the MCA parameter `rmaps_base_schedule_policy` to the value `node`. For example:

```
% mpirun --mca rmaps_base_schedule_policy node -np 4 a.out
```

The following output example shows the contents of the same hostfile used in the previous example and the results of the `mpirun` command using by-node scheduling.

```
% cat my-hosts
node0 slots=2 max_slots=20
node1 slots=2 max_slots=20
% mpirun --hostfile my-hosts -np 8 --bynode hello | sort
Hello World I am rank 0 of 8 running on node0
Hello World I am rank 1 of 8 running on node1
Hello World I am rank 2 of 8 running on node0
Hello World I am rank 3 of 8 running on node1
Hello World I am rank 4 of 8 running on node0
Hello World I am rank 5 of 8 running on node1
Hello World I am rank 6 of 8 running on node0
Hello World I am rank 7 of 8 running on node1
```

Comparing By-Slot to By-Node Scheduling

In the examples in this section, node0 and node1 each have two slots. The diagrams show the differences in scheduling between the two methods.

By-slot scheduling for the two nodes can be represented as follows:

node0	node1
0	2
1	3
4	6
5	7

By-node scheduling for the same two nodes can be represented this way:

node0	node1
0	1
2	3
4	5
6	7

Binding MPI Processes

Binding MPI processes to specific hardware processors can benefit performance in several ways:

- It can prevent the operating system from migrating processes excessively. Process migration can cost performance, especially if processes are migrated away from "local" NUMA memory or from caches that hold data that the process must access.
- It can be used to place processes relative to one another. For example, processes can be spread out over the processor sockets in a node to minimize how much processes must contend for shared hardware resources like caches or memory bandwidth. Alternatively, processes can be placed together on the same processor sockets to minimize the cost of data communications between them.

While default process binding often benefits performance, it has the potential to induce undesirable side effects:

- Spreading processes out over multiple processor sockets can increase interprocess communication costs. This effect can be most noticeable for MPI communications microbenchmarks.
- Oversubscribing particular processor sockets or cores can lead to exceptionally bad performance. Such oversubscription can result if other MPI jobs use the same binding arrangement as yours or if your processes are multithreaded.

There are three methods for specifying process binding:

- `mpirun` options
- MCA parameters
- Rankfiles

MPI processes are bound at the time that they are launched.

mpirun Options

The following bind-to-* options were introduced in the Sun HPC ClusterTools 8.2.1 software release.

[-]-bind-to-core

[-]-bind-to-socket

[-]-bind-to-board

[-]-bind-to-none

Each can be used with the process placement options:

[-]-bycore

[-]-bysocket

[-]-byboard

[-]-bynode

You can use the -report-bindings option to get a report on how the processes are bound.

Note – See the mpirun man page for detailed descriptions of the command-line binding options.

MCA Parameters

You can produce the same process binding behavior as is available with mpirun options by setting MCA parameters in the ~/.openmpi/mca-params.conf configuration file.

The MCA parameter method offers the advantage of allowing you to associate environment variables with individual process binding settings. For example, if a configuration file included the following entries, processes would be bound to successive processor sockets if the node's environment supports binding, but not otherwise:

```
orte_process_binding        = [none|core|socket|board][:if-avail]
rmaps_base_schedule_policy = [slot|socket|board|node]
```

See Chapter 7 for more information on setting MCA parameters as environment variables or in text configuration files.

Rankfiles

You can use the mpirun command's rankfile option to specify detailed bindings of MPI processes. The syntax for this option is:

`-rf` or `--rankfile` <*rankfile*>

Note – See the `mpirun` man page for more information on the rankfile option.

Controlling Input/Output

Open MPI directs UNIX standard input to `/dev/null` on all processes except the rank 0 process of MPI_COMM_WORLD. The MPI_COMM_WORLD rank 0 process inherits standard input from `mpirun`. The node from which you invoke `mpirun` need not be the same as the node where the MPI_COMM_WORLD rank 0 process resides. Open MPI handles the redirection of the `mpirun` standard input to the rank 0 process.

Open MPI directs UNIX standard output and standard error from remote nodes to the node that invoked `mpirun`, and then prints the information from the remote nodes on the standard output/error of `mpirun`. Local processes inherit the standard output/error of `mpirun` and transfer to it directly.

▼ To Redirect Standard I/O

To redirect standard I/O for Open MPI applications, use the typical shell redirection procedure on `mpirun`. For example:

```
% mpirun -np 2 my_app < my_input > my_output
```

In this example, only the MPI_COMM_WORLD rank 0 process will receive the stream from `my_input` on `stdin`. The `stdin` on all the other nodes will be tied to `/dev/null`. However, the `stdout` from all nodes will be collected into the `my_output` file.

Controlling Other Job Attributes

To Perform This Task	Use This Option
To change the working directory	`-wdir` or `--wdir`
To display debugging output	`-d`
To display command help	`-h`

▼ To Change the Working Directory

Use the `-wdir` or `--wdir` option to specify the path of an alternative working directory to be used by the processes spawned when you run your program:

```
% mpirun --wdir working–directory  program–name
```

Setting a path with `--wdir` does not affect where the runtime environment looks for executables. If you do not specify `--wdir`, the default is the current working directory. For example:

```
% mpirun --wdir /home/mystuff/bin a.out
```

The syntax above changes the working directory for `a.out` to `/home/mystuff/bin`.

▼ To Specify Debugging Output

Use this syntax to specify debugging output. For example:

```
% mpirun -d a.out
```

The `-d` option shows the user-level debugging output for all of the ORTE modules used with `mpirun`. To see more information from a particular module, you can set additional MCA debugging parameters. The availability of the additional debugging information depends on how the module of interest is implemented.

For more information on MCA parameters, see Chapter 7. For more information about whether a module provides additional verbose or debug mode, run the `ompi_info` command on that module.

▼ To Display Command Help (`-h`)

To display a list of `mpirun` options, use the `-h` option (alone).

```
% ./mpirun -h
```

Submitting Jobs Under Sun Grid Engine Integration

There are two ways to submit jobs under Sun Grid Engine integration: interactive mode and batch mode. The instructions in this chapter describe how to submit jobs interactively. For information about how to submit jobs in batch mode, see Chapter 6.

Defining Parallel Environment (PE) and Queue

A PE needs to be defined for all the queues in the Sun Grid Engine cluster to be used as ORTE nodes. Each ORTE node should be installed as an Sun Grid Engine execution host. To allow the ORTE to submit a job from any ORTE node, configure each ORTE node as a submit host in Sun Grid Engine.

Each execution host must be configured with a default queue. In addition, the default queue set must have the same number of slots as the number of processors on the hosts.

▼ To Use PE Commands

- **To display a list of available PEs (parallel environments), type the following:**

```
% qconf -spl
make
```

- **To define a new PE, you must have Sun Grid Engine manager or operator privileges. Use a text editor to modify a template for the PE. The following example creates a PE named** `orte`**.**

```
% qconf -ap orte
```

- **To modify an existing PE, use this command to invoke the default editor:**

```
% qconf -mp orte
```

- **To show a particular PE that has been defined, type this command:**

```
% qconf -sp orte
pe_name           orte
slots             8
user_lists        NONE
xuser_lists       NONE
start_proc_args   /bin/true
stop_proc_args    /bin/true
allocation_rule   $round_robin
control_slaves    TRUE
job_is_first_task FALSE
urgency_slots     min
```

The value NONE in `user_lists` and `xuser_lists` mean enable everybody and exclude nobody.

The value of `control_slaves` must be TRUE; otherwise, `qrsh` exits with an error message.

The value of `job_is_first_task` must be FALSE or the job launcher consumes a slot. In other words, `mpirun` itself will count as one of the slots and the job will fail, because only n-1 processes will start.

▼ To Use Queue Commands

- **To show all the defined queues, type the following command:**

```
% qconf -sql
all.q
```

The queue `all.q` is set up by default in Sun Grid Engine.

- **To configure the `orte` PE from the example in the previous section to the existing queue, type the following:**

```
% qconf -mattr queue pe_list "orte" all.q
```

You must have Sun Grid Engine manager or operator privileges to use this command.

Submitting Jobs in Interactive Mode

▼ To Set the Interactive Display

Before you submit a job, you must have your DISPLAY environment variable set so that the interactive window will appear on your desktop, if you have not already done so.

For example, if you are working in the C shell, type the following command:

```
% setenv DISPLAY desktop:0.0
```

▼ To Submit Jobs Interactively

1. **Use the `source` command to set the Sun Grid Engine environment variables from a file:**

```
mynode4% source /opt/sge/default/common/settings.csh
```

2. **Use the** `qsh` **command to start the interactive X Windows session, and specify the parallel environment (in this example, ORTE) and the number of slots to use:**

```
mynode4% qsh -pe orte 2
waiting for interactive job to be scheduled...
Your interactive job 324 has been successfully scheduled.
```

3. **On a different node in the cluster, use the cd command to switch to the directory where your executable is located.**

```
mynode5% cd /workspace/joeuser/ompi/trunk/builds/sparc32-g/bin
```

4. **Issue the** `mpirun` **command.**

```
mynode5% /opt/SUNWhpc/HPC8.2.1c/sun/bin/mpirun -np 4 hostname
```

In the above example, Sun Grid Engine starts the user executable *hostname* with 4 processes on the two Sun Grid Engine assigned slots. The following example shows the output from the `mpirun` command with the specified options.

```
mynode5% /opt/SUNWhpc/HPC8.2.1c/sun/bin/mpirun -np 4 --hostname
mynode5

mynode5
```

▼ To Verify That Sun Grid Engine Is Running

The following is not required for normal operation, but if you want to verify that Sun Grid Engine is being used, add `--mca ras_gridengine_verbose` to the `mpirun` command line. For example:

```
% ./mpirun -np 4 -mca ras_gridengine_verbose 100 hostname
[mynode6:04234] ras:gridengine: JOB_ID: 28
[mynode6:04234] ras:gridengine: mynode6: PE_HOSTFILE shows slots=2
[mynode6:04234] ras:gridengine: mynode7: PE_HOSTFILE shows slots=2
mynode6
mynode6
mynode7
mynode7
%
```

▼ To Start an Interactive Session Using `qrsh`

An alternate way to start an interactive session is by using `qrsh` instead of `qsh`. For example:

```
% qrsh -V -pe orte 8 mpirun -np 4 -byslot hostname
```

Using MPI Client/Server Applications

The instructions in this section explain how to get best results when starting Open MPI client/server applications.

▼ To Launch the Client/Server Job

1. **Type the following command to launch the server application. Substitute the name of your MPI job's universe for *univ1*:**

```
% ./mpirun -np 1 --universe univ1 t_accept
```

2. **Type the following command to launch the client application, substituting the name of your MPI job's universe for *univ1*:**

```
% ./mpirun -np 4 --universe univ1 t_connect
```

If the client and server jobs span more than 1 node, the first job (that is, the server job) must specify on the mpirun command line all the nodes that will be used. Specifying the node names allocates the specified hosts from the entire universe of server and client jobs.

For example, if the server runs on `node0` and the client job runs on `node1` only, the command to launch the server must specify both nodes (using the `-host node0,node1` flag) even it uses only one process on `node0`.

Assuming that the persistent daemon is started on `node0`, the command to launch the server would look like this:

```
node0% ./mpirun -np 1 --universe univ1 -host node0,node1 t_accept
```

The command to launch the client is:

```
node0% ./mpirun -np 4 --universe univ1 -host node1 t_connect
```

Using Name Publishing

If you are planning on using name publishing, you must perform some additional tasks. You need to start up an ompi-server process on your server so that both the clients and servers can exchange information using that server.

For information about how to start the ompi-server process, type the following command on your server:

```
% man ompi-server
```

Troubleshooting Client/Server Jobs

If the MPI client/server job fails to start, you might see error messages similar to this:

```
node0% ./orted --persistent --seed --scope public --universe univ4
--debug
[node0:21760] procdir: (null)
[node0:21760] jobdir: (null)
[node0:21760] unidir:
/tmp/openmpi-sessions-joeuser@node0_0/univ4
[node0:21760] top: openmpi-sessions-joeuser@node0_0
[node0:21760] tmp: /tmp
[node0:21760] orte_init: could not contact the specified
universe name univ4
[node0:21760] [NO-NAME] ORTE_ERROR_LOG: Unreachable in file
/opt/SUNWhpc/HPC8.2.1c/sun/bin/orted/runtime/orte_init_stage1.c
at line 221
```

These messages indicate that there is residual data left in the `/tmp` directory. This can happen if a previous client/server job has already run from the same node.

To empty the `/tmp` directory, use the `orte-clean` utility. For more information about `orte-clean`, see the `orte-clean` man page.

You might also need to run orte-clean if you see error messages similar to the following:

```
node0% ./orted --persistent --seed --scope public --universe univ4 --debug
[node0:21760] procdir: (null)
[node0:21760] jobdir: (null)
[node0:21760] unidir:
/tmp/openmpi-sessions-joeuser@node0_0/univ4
[node0:21760] top: openmpi-sessions-joeuser@node0_0
```

```
[node0:21760] tmp: /tmp
[node0:21760] orte_init: could not contact the specified
universe name univ4
[node0:21760] [NO-NAME] ORTE_ERROR_LOG: Unreachable in file
/opt/SUNWhpc/HPC8.2.1c/sun/bin/orted/runtime/orte_init_stage1.c
at line 221
--------------------------------------------------------------
It looks like orte_init failed for some reason; your parallel process is likely
to abort.  There are many reasons that a parallel process can fail during
orte_init; some of which are due to configuration or environment problems.  This
failure appears to be an internal failure; here's some additional information
(which may only be relevant to an Open MPI developer):
   orte_sds_base_contact_universe failed
   --> Returned value -12 instead of ORTE_SUCCESS
--------------------------------------------------------------
[node0:21760] [NO-NAME] ORTE_ERROR_LOG: Unreachable in file
/opt/SUNWhpc/HPC8.2.1c/sun/bin/orted/runtime/orte_system_init.c
at line 42
[node0:21760] [NO-NAME] ORTE_ERROR_LOG: Unreachable in file
/opt/SUNWhpc/HPC8.2.1c/sun/bin/orte/runtime/orte_init.c
at line 52
Open RTE was unable to initialize properly.  The error occured while attempting
to orte_init().  Returned value -12 instead of ORTE_SUCCESS.
```

Handling Network Failures In Multi-Rail Infiniband Configurations

A communication failover feature for handling network failures in multi-rail Infiniband configurations was introduced in the Sun HPC ClusterTools 8.2.1c software release. When a completion error is detected on a given rail, the failover software maps out that rail and routes future traffic through other rails available to the process.

The failover feature supports Open MPI `openib` BTL (Open Fabrics User Verbs) for the communications layer.

Note – The failover feature does not function with uDAPL or IPoIB.

You enable the failover feature by setting the MCA parameter `pml_obl_enable_failover`, either from the `mpirun` command line or in the `openmpi-mca-parameter.conf` file.

From the `mpirun` Command Line:

```
--mca pml_obl_enable_failover 1
```

In the `openmpi-mca-parameter.conf` File:

```
pml_obl_enable_failover 1
```

Note – For the failover feature to function correctly, you must keep the default settings of the `btl_openib_flags` parameter.

Optimizing Failover Timing

Failover occurs when we get a completion error on a connection. A completion error is typically generated when a message transfer fails to complete within a defined time period. By default, this time period is probably larger than is optimal when the the failover feature enabled. This section explains how the timeout value is determined and how to adjust it.

- Transport Timer Timeout (Ttt) - This is the local ACK timeout period. The MCA parameter `btl_openib_ib_timeout` regulates the Ttt duration according to this formula:

  Ttt = 4.096 microseconds * $2^{\texttt{btl\_openib\_ib\_timeout}}$

  The `btl_openib_ib_timeout` parameter corresponds to the `IBV_QP_TIMEOUT` attribute.

- Retry Count - This is the number of times send operations will be retried following repeated Transport Timer Timeouts. The MCA parameter `btl_openib_ib_retry_count` specifies the retry value.

  The `btl_openib_ib_retry_count` parameter corresponds to the `IBV_QP_RETRY` attribute.

Each of these values can be set on a queue pair.

The default Transport Timer Timeout parameter value is 20, which results in a minimum Ttt period of 4.29 seconds.

Ttt = 4.096 microseconds * 2^{20} = 4.096 microseconds * 1048576 = 4.29 seconds

The upper limit of the Transport Timer Timeout is four times this basic timeout period, or 17.18 seconds. That is, with a Ttt parameter value of 20, the Ttt period is guaranteed to fall within the following range:

4.096 microseconds * 2^{20} <= timeout <= 4 * 4.096 microseconds * 2^{20}

4.29 seconds <= timeout <= 17.18 seconds

The default Retry Count parameter value is 7. When both default values are used, the combined Ttt and Retry Count timeout period will fall within the following range:

7 * 4.096 microseconds * 2^{20} <= timeout <= 7 * 4 * 4.096 microseconds * 2^{20}

30.06 seconds <= timeout <= 120.26 seconds

If you want to cause the network failover action to take effect more quickly, set the `IBV_QP_TIMEOUT` attribute to a value smaller than the default. For example, if you set the timeout attribute to 15, the failover will occur within the following range:

7 * 4.096 microseconds * 2^{15} <= timeout <= 7 * 4 * 4.096 microseconds * 2^{15}

7 * 4.096 microseconds * 32768 <= timeout <= 7 * 4 * 4.096 microseconds * 32768

0.94 seconds <= timeout <= 3.76 seconds

The MPI library may also detect asynchronous errors. One example of these is the `PORT_EVENT` error. This error is ignored because it will ultimately result in a timeout error, which will be handled by the failover software. Other asynchronous errors, which are less likely to occur, will cause the MPI library to abort.

When running with failover enabled, you are likely to have best results if you run with the Ttt value set as follows:

```
--mca btl_openib_ib_timeout 15
```

Experimentally, this yields a timeout value of approximately 2 seconds.

Viewing Failures

By default, no information about rail loss is generated during a job run. However, if you enable some level of verbosity for the MPI jobs you run, you will be able to determine when a network failover occurs.

For example, if you include the following entry on the mpirun command line, minimal output will be displayed that shows when network interfaces are mapped out.

```
--mca pml_obl_verbose 10
```

To see more details about failover activities, use verbosity level 20 or 30.

You can also view failover activity by using the MCA parameter `btl_openib_verbose_failover`. As with the parameter `pml_obl_verbose`, you can choose among three verbosity levels: 10, 20, and 30. For example:

```
--mca btl_openib_verbose_failover 30
```

Forcing Failovers for Port Event Errors

Some error events do not directly trigger a failover, but are likely to cause one through cascading timeout effects. Port Event errors are an example of this. The failover software catches Port Event errors, but does not take action. A failover may subsequently result from a Transport Timer Timeout failure, which could be a natural consequence of the Port Event.

When you suspect that Port Event error may occur, you might want to force the failover, without waiting for an eventual Ttt-driven failover to occur. You can do this by setting a flag with the following MCA parameter:

```
--mca btl_openib_port_event_error_failover 1
```

Standard Out Example

The following example illustrates the kind of information that is displayed on standard out when a failover occurs. In this example, the Ttt parameter is set to 15 and the verbosity level is set to 10.

```
[ct-x2200-4 osu]$ mpirun -np 2 -host ct-x2200-11,ct-x2200-12 -mca btl
self,sm,openib -mca btl_openib_ib_timeout 15 -mca pml_ob1_verbose 10
osu_latency
------------------------------------------------------------------------
# OSU MPI Latency Test (Version 2.1)
# Size          Latency (us)
0               3.25
1               3.50
2               3.49
[...snip...]
65536           68.93
131072          111.90
[ct-x2200-12:04711] BTL openib error: rank=1 mapping out lid=45:name=mlx4_1 to
rank=0 on node=ct-x2200-11
[ct-x2200-11:09390] BTL openib error: rank=0 mapping out lid=42:name=mlx4_1 to
rank=1 on node=ct-x2200-12
262144          41616.97
524288          349.64
1048576         687.25
```

```
2097152         1372.68
4194304         2749.38
[ct-x2200-4 osu]$
```

This output shows the `openib` error that caused the rail to be mapped out. It also shows a sudden increase in latency for the data affected by the network failure. The 41616.97 microseconds latency includes the time required for the seven retries to complete.

Syslog Example

You can use the `syslog` facility to record information about network failures and any consequent remapping of interfaces. Run with the following `mca` parameter to enable the use of `syslog`:

```
--mca notifier syslog
```

This will cause information to be stored in `/var/log/messages`. Messages about the job will also be written to standard out.

In this second example, the timeout parameter is again set to 15, but we have enabled the logging of failures to the syslog facility.

```
[ct-x2200-4 osu] $mpirun -np 2 -host ct-x2200-11,ct-x2200-12 -mca btl
self,sm,openib -mca btl_openib_ib_timeout 15 -mca notifier syslog osu_latency
------------------------------------------------------------------------
# OSU MPI Latency Test (Version 2.1)
# Size          Latency (us)
0               3.24
1               3.52
2               3.50
[...snip...]
4096            12.99
Open MPI Error Report:[5246]: BTL openib error: rank=1 mapping out
lid=45:name=mlx4_1 to rank=0 on node=ct-x2200-11Open MPI Error Report:[10046]:
BTL openib error: rank=0 mapping out lid=42:name=mlx4_1 to rank=1 on
node=ct-x2200-12
8192            440.22
16384           22.15
32768           33.46
65536           53.55
131072          93.33
262144          177.53
524288          344.23
1048576         674.95
2097152         1346.90
4194304         2703.70
```

The /var/log/messages file on one of the nodes will include the following information:

```
Dec  2 17:02:05 ct-x2200-11 Open MPI Error Report:[10046]: BTL openib error:
rank=0 mapping out lid=42:name=mlx4_1 to rank=1 on node=ct-x2200-12
```

For More Information

For more information about the mpirun command and its options, see the following:

- Chapter 7, "Using MCA Parameters With mpirun" on page 61
- the mpirun(3) man page
- Open MPI FAQ at http://www.open-mpi.org

CHAPTER 6

Running Programs With `mpirun` in Distributed Resource Management Systems

This chapter describes the options to the `mpirun` command that are used for distributed resource management, and provides instructions for each resource manager. It contains the following sections:

`mpirun` Options for Third-Party Resource Manager Integration

ORTE is compatible with a number of other launchers, including `rsh/ssh`, Sun Grid Engine, and PBS.

Note – Open MPI itself supports other third-party launchers supported by Open MPI, such as SLURM and Torque. However, these launchers are currently not supported in Sun HPC ClusterTools software. To use these other third-party launchers, you must download the Open MPI source, compile, and link with the libraries for the launchers.

Checking Your Open MPI Configuration

To see whether your Open MPI installation has been configured for use with the third-party resource manager you want to use, issue the `ompi_info` command and pipe the output to `grep`. The following examples show how to use `ompi_info` to check for the desired third-party resource manager.

▼ To Check for `rsh/ssh`

To see whether your Open MPI installation has been configured to use the `rsh/ssh` launcher:

```
% ompi_info | grep rsh
MCA plm: rsh (MCA v2.0, API v2.0, Component v1.3)
```

▼ To Check for PBS/Torque

To see whether your Open MPI installation has been configured to use the PBS/Torque launcher:

```
% ompi_info | grep tm
MCA ras: tm (MCA v2.0, API v2.0, Component v1.3)
MCA plm: tm (MCA v2.0, API v2.0, Component v1.3)
```

▼ To Check for Sun Grid Engine

To see whether your Open MPI installation has been configured to use Sun Grid Engine:

```
% ompi_info | grep gridengine
MCA ras: gridengine (MCA v2.0, API v2.0, Component v1.3)
```

Running Parallel Jobs in the PBS Environment

If your Open MPI environment is set up to include PBS, Open MPI automatically detects when `mpirun` is running within PBS, and will execute properly.

First reserve the number of resources by invoking the `qsub` command with the `-l` option. The `-l` option specifies the number of nodes and the number of processes per node. For example, this command sequence reserves four nodes with four processes per node for the job `myjob.sh`:

```
% qsub -l nodes=4:ppn=4 myjob.sh
```

When you enter the PBS environment, you can launch an individual job or a series of jobs with `mpirun`. The `mpirun` command launches the job using the nodes and processes information from PBS.The resource information is accessed using the `tm` calls provided by PBS; hence, `tm` is the name used to identify the module in ORTE. The job ranks are children of PBS, not ORTE.

You can run an ORTE job within the PBS environment in two different ways: interactive and scripted.

▼ To Run an Interactive Job in PBS

1. **Enter the PBS environment interactively with the `-I` option to `qsub`, and use the `-l` option to reserve resources for the job.**

   Here is an example.

```
% qsub -l nodes=2:ppn=2 -I
```

   The command sequence shown above enters the PBS environment and reserves one node called `mynode` with two processes for the job. Here is the output:

```
qsub: waiting for job 20.mynode to start
qsub: job 20.mynode ready
Sun Microsystems Inc. SunOS 5.10 Generic June 2006
pbs%
```

2. **Launch the `mpirun` command.**

   Here is an example that launches the `hostname` command with a verbose output:

```
pbs% /opt/SUNWhpc/HPC8.2.1c/sun/bin/mpirun -np 4 -mca
plm_tm_verbose 1 hostname
```

The output shows the hostname program being run on ranks `r0` and `r1`:

```
% /opt/SUNWhpc/HPC8.2.1c/sun/bin/mpirun -np 4 -mca plm_tm_verbose
1 hostname
[hostname1:09064] plm:tm: launching on node mynode1
[hostname2:09064] plm:tm: launching on node mynode2
hostname2
hostname1
hostname2
hostname1
```

▼ To Run a Batch Job in PBS

1. **Write a script that calls** `mpirun`**.**

   In the following examples, the script is called `myjob.csh`. The system is called `mynode`. Here is an example of the script.

```
#!/bin/csh

/opt/SUNWhpc/HPC8.2.1c/sun/bin/mpirun -np 2 -mca plm_tm_verbose 1
hostname
```

2. **Enter the PBS environment and use the** `-l` **option to** `qsub` **to reserve resources for the job.**

   Here is an example of how to use the `-l` option with the `qsub` command.

```
% qsub -l nodes=2:ppn=2 myjob.csh
```

   This command enters the PBS environment and reserves one node with two processes for the job that will be launched by the script named `myjob.csh`.

Here is the output to the script `myjob.csh`.

```
% more myjob.csh.*
::::::::::::::
myjob.csh.e2365
::::::::::::::
::::::::::::::
myjob.csh.o2365
::::::::::::::
Warning: no access to tty (Bad file number).
Thus no job control in this shell.
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
hostname5
hostname4
hostname5
hostname4
```

After the job finishes, it generates two output files:

- *name_of_job*`.e`*job_id* is the file that shows the error outputs (for example, `myjob.csh.e2365` in the above example).
- *name_of_job*`.o`*job_id* is the file that shows the standard outputs (for example, `myjob.csh.o2365` in the above example).

As you can see, the `pbsrun` command calls `mpirun`, which forks into two calls of the `hostname` program, one for each node.

Running Parallel Jobs in the Sun Grid Engine Environment

Sun Grid Engine 6.1 is the supported version of Sun Grid Engine for Sun HPC ClusterTools 8.2.1c.

Before you can run parallel jobs, make sure that you have defined the parallel environment and queue before running the job.

Defining Parallel Environment (PE) and Queue

A PE needs to be defined for all the queues in the Sun Grid Engine cluster to be used as ORTE nodes. Each ORTE node should be installed as a Sun Grid Engine execution host. To allow the ORTE to submit a job from any ORTE node, configure each ORTE node as a submit host in Sun Grid Engine.

Each execution host must be configured with a default queue. In addition, the default queue set must have the same number of slots as the number of processors on the hosts.

▼ To Use PE Commands

- **To display a list of available PEs (parallel environments), type the following:**

```
% qconf -spl
make
```

- **To define a new PE, you must have Sun Grid Engine manager or operator privileges. Use a text editor to modify a template for the PE. The following example creates a PE named** `orte`**.**

```
% qconf -ap orte
```

- **To modify an existing PE, use this command to invoke the default editor:**

```
% qconf -mp orte
```

- **To show a particular PE that has been defined, type this command:**

```
% qconf -sp orte
pe_name           orte
slots             8
user_lists        NONE
xuser_lists       NONE
start_proc_args   /bin/true
stop_proc_args    /bin/true
allocation_rule   $round_robin
control_slaves    TRUE
job_is_first_task FALSE
urgency_slots     min
```

The value NONE in `user_lists` and `xuser_lists` mean enable everybody and exclude nobody.

The value of `control_slaves` must be TRUE; otherwise, `qrsh` exits with an error message.

The value of `job_is_first_task` must be FALSE or the job launcher consumes a slot. In other words, `mpirun` itself will count as one of the slots and the job will fail, because only n-1 processes will start.

▼ To Use Queue Commands

- **To show all the defined queues, type the following command:**

```
% qconf -spl
all.q
```

The queue `all.q` is set up by default in Sun Grid Engine.

- **To configure the `orte` PE from the example in the previous section to the existing queue, type the following:**

```
% qconf -mattr queue pe_list "orte" all.q
```

You must have Sun Grid Engine manager or operator privileges to use this command.

Submitting Jobs Under Sun Grid Engine Integration

There are two ways to submit jobs under Sun Grid Engine integration: interactive mode and batch mode. The instructions in this section describe how to submit jobs in batch mode. For information about how to use interactive mode, see Chapter 5.

▼ To Set the Interactive Display

Before you submit a job, you must have your DISPLAY environment variable set so that the interactive window will appear on your desktop, if you have not already done so.

For example, if you are working in the C shell, type the following command:

```
% setenv DISPLAY desktop:0.0
```

▼ To Submit Jobs in Batch Mode

Note – Before you can use the parallel environment, make sure that you have set it up before running the job. See "Defining Parallel Environment (PE) and Queue" on page 54 for more information.

1. **Create the script. In this example,** `mpirun` **is embedded within a script to** `qsub`**.**

```
mynode4% cat SGE.csh
#!/usr/bin/csh

# set PATH: including location of MPI program to be run
setenv PATH
/opt/SUNWhpc/HPC8.2.1c/sun/examples/connectivity:${PATH}

mpirun -np 4 -mca ras_gridengine_verbose 100 connectivity.sparc -v
```

Note – The `--mca ras_gridengine_verbose 100` setting is used in this example only to show that Sun Grid Engine is being used. This would not be needed for normal operation.

2. **Next, source the Sun Grid Engine environment variables from a** `settings.csh` **file where $SGE_ROOT is set to** `/opt/sge`**:.**

```
% source $SGE_ROOT/default/common/settings.csh
```

3. **To start the batch (or scripted) job, specify the parallel environment, slot number and the user executable.**

```
% qsub -pe orte 2 sge.csh
your job 305 ("sge.csh") has been submitted
```

   Since this is submitted as a batch job, you would not expect to see output at the terminal. If no indication is given for where the output should go, Sun Grid Engine redirects to your home directory and creates <*job_name*>.o<*job_number*>.

The job creates the output files. The file name with the format *name_of_job*.o*job_id* contains the standard output. The file name with the format *name_of_job*.e*job_id* contains the error output. If the job executes normally, the error output files will be empty.

The following example lists the files produced by a job called `sge.csh` with the job ID number 866:

```
% ls -rlt ~ | tail
-rw-r--r--   1 joeuser   mygroup       0 Jan 16 16:42 sge.csh.po866
-rw-r--r--   1 joeuser   mygroup       0 Jan 16 16:42 sge.csh.pe866
-rw-r--r--   1 joeuser   mygroup       0 Jan 16 16:42 sge.csh.e866
-rw-r--r--   1 joeuser   mygroup     194 Jan 16 16:42 sge.csh.o866
```

By default, the output files are located in your home directory, but you can use Sun Grid Engine software to change the location of the files, if desired.

Note – In most cases, you do not need to change the values set in the `gridengine` MCA parameters. If you run into difficulty and want to change the values for debugging purposes, the option is available. For more information about MCA parameters, see Chapter 7.

▼ To See a Running Job

- **Type the following command:**

```
% qstat -f
```

▼ To Delete a Running Job

- **Type the following command:**

```
% qdel job-number
```

where *job-number* is the number of the job you want to delete.

For more information about Sun Grid Engine commands, refer to the Sun Grid Engine documentation.

`rsh` Limitations

Note – This issue affects both `rsh` and the Sun Grid Engine program `qrsh`. `qrsh` uses `rsh` to launch jobs.

If you are using `rsh` or `qrsh` as the job launcher on a large cluster with hundreds of nodes, `rsh` might show the following error messages when launching jobs on the remote nodes:

```
rcmd: socket: Cannot assign requested address
rcmd: socket: Cannot assign requested address
rcmd: socket: Cannot assign requested address
[node0:00749] ERROR: A daemon on node m2187 failed to start as
expected.
[node0:00749] ERROR: There may be more information available from
[node0:00749] ERROR: the 'qstat -t' command on the Grid Engine
tasks.
[node0:00749] ERROR: If the problem persists, please restart the
[node0:00749] ERROR: Grid Engine PE job
```

This indicates that `rsh` is running out of sockets when launching the job from the head node.

Using `rsh` as the Job Launcher

If you are using `rsh` as your job launcher, use `ssh` instead. Add the following to your command line:

```
-mca plm_rsh_agent ssh
```

Using Sun Grid Engine as the Job Launcher

If you are using Sun Grid Engine version 6.1 or earlier as your job launcher, you can modify the Sun Grid Engine configuration to allow Sun Grid Engine to use `ssh` instead of `rsh` to launch tasks on the remote nodes. The following web site describes how to perform this workaround:

http://gridengine.sunsource.net/howto/qrsh_qlogin_ssh.html

Note that this workaround does not properly track resource usage, nor does it allow proper job accounting. Sun Grid Engine tracks resource usage by attaching an extra groupid when launching tasks as a user of the remote connection.

Sun Grid Engine version 6.2 fixes this issue by not using `rsh` to start jobs on remote nodes. Instead Sun Grid Engine version 6.2 makes use of a native Interactive Job Support (IJS), which removes any dependencies on `rsh`, `ssh`, or `telnet`. It is recommended that you upgrade to the latest available version of Sun Grid Engine.

For More Information

For more information about using the `mpirun` command to perform batch processing, see the following:

- `mpirun`(3) man page
- Running MPI Jobs section of the Open MPI FAQ at `http://www.open-mpi.org/faq/?category=running#plm-available`

CHAPTER 7

Using MCA Parameters With `mpirun`

Open MPI uses Modular Component Architecture (MCA) parameters to provide a way to tune your runtime environment. Each parameter corresponds to a specific function. You change the value of the parameter in order to change the function.

Developing an Open MPI application that uses MCA parameters poses a number of advantages. Developers and administrators can customize the Open MPI environment to suit the specific needs of hardware or the operating environment. For example, a system administrator might use MCA parameters to optimize an Open MPI installation on a network so that users only need to run with the default values to obtain the best performance.

This chapter contains the following topics:

In order to understand how MCA parameters fit within Open MPI, you must understand how the Modular Component Architecture is constructed.

About the Modular Component Architecture

The Modular Component Architecture (MCA) is the backbone for much of Open MPI's functionality. It is a series of *frameworks*, *components*, and *modules* that are assembled at runtime to create an MPI implementation.

An MCA *framework* manages a specific Open MPI task (such as process launching for ORTE). Each MCA framework supports a single component type, but can support multiple versions of that type. The framework uses the services from the MCA base functionality to find and/or load components.

An MCA *component* is an implementation of a framework's interface. It is a standalone collection of code that can be bundled into a plug-in that can be inserted into the Open MPI code base, either at runtime and/or at compile time.

An MCA *module* is an instance of a component. For example, if a node running an Open MPI application has multiple Ethernet NICs, the Open MPI application will contain one TCP MPI point-to-point component, but two TCP point-to-point modules.

For more information about the Open MPI Modular Component Architecture, see the Open MPI FAQ on runtime tuning at:

`http://www.open-mpi.org/faq/?category=tuning`

Open MPI Frameworks

There are three types of frameworks in Open MPI:

- In the MPI layer (OMPI)
- In the run-time layer (ORTE)
- In the operating system/platform layer (OPAL)

You might think of these frameworks as ways to group MCA parameters by function. For example, the OMPI `btl` framework controls the functions in the byte transfer layer, or BTL (point-to-point byte movement) in the network. All of the MCA parameters that are grouped under `btl` affect the BTL layer.

In addition to the parameters that are grouped under the individual frameworks, there are top-level MCA parameters that affect the frameworks themselves and specify values to your Open MPI installation.

TABLE 7-1 Top-Level MCA Parameters

Parameter Group	Description
`mca`	Specify paths or functions for MCA parameters
`mpi`	Specify MPI behavior at runtime
`orte`	Specify debugging functions and components for ORTE
`opal`	Specify stack trace information

To view the available top-level parameters in each group, type the following command:

```
% ompi_info --param groupname groupname
```

where groupname stands for the parameter group you want to view. For example, to view the available MPI parameters, you would type:

```
% ompi_info --param mpi mpi
```

The `ompi_info` Command

The `ompi_info` command returns information about your Sun HPC ClusterTools/Open MPI installation. When you issue the command without any modifiers, `ompi_info` returns the following information:

- Revision information for Open MPI, ORTE, and OPAL
- Installed compilers
- Architecture of the node on which Open MPI is installed
- Version information for the installed frameworks

Using the ompi_info Command With MCA Parameters

The ompi_info command can list the parameters for a given component, all the parameters for a specific framework, or all parameters. The ompi_info output for most parameters contains a description of the parameter. The output for any parameter shows the current value of that parameter.

▼ To List All MCA Parameters

- **Type the following command at the system prompt:**

```
% ompi_info --param all all
```

The output from **ompi_info** lists all of the installed frameworks, their MCA parameters, and their current values.

▼ To List All MCA Parameters For a Framework

- **Type the following command at the system prompt:**

```
% ompi_info --param btl all
```

In this example, the command output will list all available MCA parameters for the btl framework.

▼ To Display All MCA Parameters For a Selected Component

- **Type the following command at the system prompt:**

```
% ompi_info --param btl tcp
```

In this example, the command output will list all available MCA parameters for the tcp component, which is part of the btl framework.

Using MCA Parameters

There are three ways to use MCA parameters with Open MPI:

1. Setting the parameter from the command line using the `mpirun --mca` command. This method assumes the highest precedence; values set for parameters using this method override any other values specified for the same parameter.
2. Using the parameter as an environment variable. Values for parameters set in this fashion assume the next highest priority.
3. Setting the parameter values in a text file. Parameter values specified using this method have the lowest priority.

▼ To Set MCA Parameters From the Command Line

- **Type the following command at the system prompt:**

```
% mpirun --mca param-name value
```

In this example, *param-name* stands for the name of the MCA parameter you want to set, and *value* stands for the new value you want to specify for the parameter. For example, the following command sets the value of the `mpi_show_handle_leaks` parameter to 1 for the specified job:

```
% mpirun --mca mpi_show_handle_leaks 1 -np 4 a.out
```

This sets the value of MCA parameter `mpi_show_handle_leaks` to 1 before running the program `a.out` with four processes.

Using MCA Parameters As Environment Variables

As with other types of environment variables, the syntax for setting MCA parameters as environment variables varies with the type of command shell.

▼ To Set MCA Parameters in the `sh` Shell

1. **Type the following command at the prompt:**

```
% OMPI_MCA_param-name=value
```

   where *param-name* is the name of the MCA parameter you want to set, and *value* is the desired value for the parameter. For example, the following command sets the `mpi_show_handle_leaks` parameter to 1:

```
% OMPI_MCA_mpi_show_handle_leaks=1
```

2. **Type the following command:**

```
% export OMPI_MCA_param-name
```

   For example, an export command using the parameter used in the previous step would look like this:

```
% export OMPI_MCA_mpi_show_handle_leaks
```

3. **Issue the `mpirun` command with the desired options. For example:**

```
% mpirun -np 4 a.out
```

▼ To Set MCA Parameters in the C Shell

1. **Use the `setenv` command to set the MCA parameter.**

```
% setenv OMPI_MCA_param-name  value
```

   where *param-name* is the name of the MCA parameter you want to set, and *value* is the desired value for the parameter. The following example shows how to set the `mpi_show_handle_leaks` parameter to 1.

```
% setenv OMPI_MCA_mpi_show_handle_leaks 1
```

2. **Issue the `mpirun` command for the program (in this example, `a.out`).**

```
% mpirun -np 4 a.out
```

▼ To Specify MCA Parameters Using a Text File

1. **Create a text file, specifying each parameter/value pair on a separate line. Comments are allowed. For example:**

```
# This is a comment
# Set the same MCA parameter as in previous examples
mpi_show_handle_leaks = 1

# Default to rsh always
plm_rsh_agent = rsh

mpi_preconnect_all = 1
mpi_param_check = 0
#
# udapl parameters - comment or uncomment as needed
#
#btl = self,tcp,sm
#btl = self,udapl,sm
btl = ^tcp
```

2. **Name the file** `mca-params.conf` **and save it.**

   You can save the file either to your home directory under `$HOME/.openmpi/mca-params.conf`, where the parameter values in the file will only affect your jobs, or you can save it to `/opt/SUNWhpc/HPC8.2.1c/sun/etc/openmpi-mca-params.conf`, where the parameter values in the file affect all users.

   The following example shows the output from the `ompi_info` command for `mca_param_files`.

```
% ompi_info --param mca mca_param_files
MCA mca: parameter "mca_param_files" (current value:
"/home/joeuser/.openmpi/mca-params.conf:
/opt/SUNWhpc/HPC8.2.1c/sun/etc/openmpi-mca-params.conf")
Path for MCA configuration files containing default parameter values
MCA mca: parameter "mca_component_path" (current value:
"/opt/SUNWhpc/HPC8.2.1c/sun/etc/openmpi:/home/joeuser/.openmpi/components
")
Path where to look for Open MPI and ORTE components
MCA mca: parameter "mca_verbose" (current value: <none>)
Top-level verbosity parameter
MCA mca: parameter "mca_component_show_load_errors" (current value: "1")
Whether to show errors for components that failed to load or not
MCA mca: parameter "mca_component_disable_dlopen" (current value: "0")
Whether to attempt to disable opening dynamic components or not
```

The MCA parameter `mca_param_files` specifies a colon-delimited path of files to search for MCA parameters. Files to the left of the colon have lower precedence; files to the right of the colon have higher precedence. At runtime, `mpirun` searches the following two files in order when the `mca_param_files` parameter is set:

1. `$HOME/.openmpi/mca-params.conf`: The user-supplied set of values takes the highest precedence.
2. `$prefix/etc/openmpi-mca-params.conf`: The system-supplied set of values has a lower precedence.

In the above example, Open MPI first searches `/home/joeuser/.openmpi/mca-params.conf` for MCA parameters, and then searches `/opt/SUNWhpc/HPC8.2.1c/sun/etc/openmpi-mca-params.conf`. If a parameter appears in both locations, the value set in the second file (the file to the right of the colon) is used.

Including and Excluding Components

Each MCA framework has a top-level MCA parameter that you can use to select which components are to be used at runtime. In other words, there is an MCA parameter of the same name as each MCA framework (for example, `btl`) that you can use to include or exclude components from a given run.

You can use top-level parameters in the same way you would use other MCA parameters (for example, you can set them from the command line, as environment variables, or in text files).

For example, the `btl` MCA parameter is used to control which byte transfer layer (BTL) components are used with `mpirun`. The value for the `btl` parameter is a list of components separated by commas, with the optional prefix ^ (caret symbol).

Note – Do not mix “include” instructions with “exclude” instructions in the same command; otherwise, `mpirun` returns an error.

▼ To Include and Exclude Components Using the Command Line

- **Type the following command at the system prompt:**

```
% mpirun --mca framework comp1, comp2 ^comp3
```

In this example, the components *comp1* and *comp2* are included for the framework specified by `--mca` *framework*. Component *comp3* is excluded, since it is preceded by the ^ (caret) symbol.

For example, the following command excludes the `tcp` and `openib` components from the BTL framework, and implicitly includes all the other components:

```
% mpirun --mca btl ^tcp,openib ...
```

The use of the caret followed by the ellipsis in the command means "Perform the opposite action with the rest of the components." When the `mpirun -- mca` command specifies components to be excluded, the caret followed by the ellipsis ellipsis implicitly includes the rest of the components in that framework. When the `mpirun --mca` command specifically includes components, the caret followed by the ellipsis means "and exclude the components not specified."

For example, the following command includes only the `self`, `sm`, and `gm` components of `btl` and implicitly excludes the rest:

```
% mpirun --mca btl self,sm,gm ...
```

Using MCA Parameters With Sun Grid Engine

The `ras_gridengine` parameters enable you to specify output from the Open MPI RAS (Resource Allocation Subsystem). The rsh PLM (Process Launch Module) contains the `gridengine` parameters.

The following example shows the `mpirun` command line being used to set the MCA parameter `plm_gridengine_debug` value to 100.

```
% mpirun -np 4 -mca plm_gridengine_debug 100 connectivity.sparc -v
```

To view a list of the RAS parameters from the command line, use the `ompi_info` command. The following example shows how to specify the RAS parameters.

```
% ompi_info -param ras gridengine
```

Changing the Default Values in MCA Parameters

Note – In most cases, you do not need to change the default values in the gridengine MCA parameters. If you encounter a difficulty and want to change the values for debugging purposes, the options are available.

There are options available in the MCA PLM and RAS components and modules to allow changes of the default values.

For more information about how to change the values in MCA parameters, see the General Run-time Tuning FAQ on the Open MPi Web site at:

`http://www.open-mpi.org/faq/?category=tuning#setting-mca-params`

For More Information

For more information about the Modular Component Architecture and MCA parameters, refer to the following sources:

- Chapter 5, "Running Programs With the `mpirun` Command" on page 21
- Open MPI FAQ about runtime tuning: `http://www.open-mpi.org/faq/?category=tuning`
- The `ompi_info` man page
- The `ompi_info --help` command

CHAPTER 8

Using the DTrace Utility With Open MPI

This chapter describes how to use the Solaris™ Dynamic Tracing (DTrace) utility with Open MPI. DTrace is a comprehensive dynamic tracing utility that you can use to monitor the behavior of applications programs as well as the operating system itself. You can use DTrace on live production systems to understand those systems' behavior and to track down any problems that might be occurring.

The D language is the programming language used to create the source code for DTrace programs.

The content of this chapter assumes knowledge of the D language and how to use DTrace.

The following topics are covered in this chapter:

For more information about the D language and DTrace, refer to the *Solaris Dynamic Tracing Guide* (Part Number 817-6223). This guide is part of the Solaris 10 OS Software Developer Collection.

Solaris 10 OS documentation can be found on the web at the following location:

http://www.sun.com/documentation

Follow these links to the *Solaris Dynamic Tracing Guide:*

Solaris Operating Systems -> Solaris 10 -> Solaris 10 Software Developer Collection

---

Note – The programs and script mentioned in the sections that follow are located at:

`/opt/SUNWhpc/examples/mpi/dtrace`

Checking the `mpirun` Privileges

Before you run a program under DTrace, you need to make sure that you have the correct `mpirun` privileges.

In order to run the script under `mpirun`, make sure that you have `dtrace_proc` and `dtrace_user` privileges. Otherwise, DTrace will return the following error because it does not have sufficient privileges:

```
dtrace:  failed to initialize dtrace:  DTrace requires additional
privileges
```

▼ To Determine the Correct Privileges on the Cluster

To determine whether you have the appropriate privileges on the entire cluster, perform the following steps:

1. **Use your favorite text editor to create the following shell script, called `mpppriv.sh`:**

```
#!/bin/sh
# mpppriv.sh -  run ppriv under a shell so you can get the privileges
#     of the process that mpirun creates
ppriv $$
```

2. **Type the following command, replacing `host1` and `host2` with the names of hosts in your cluster:**

```
% mpirun -np 2 --host host1,host2 mpppriv.sh
```

If the output of `ppriv` shows that the E privilege set has the `dtrace` privileges, then you will be able to run `dtrace` under `mpirun` (see the following two examples). Otherwise, you must adjust your system to get `dtrace` access.

The following example shows the output from `ppriv` when the privileges have not been set:

```
% ppriv $$
4084:  -csh
flags = <none>
  E:  basic
  I:  basic
  P:  basic
  L:  all
```

This example shows `ppriv` output when the privileges have been set:

```
% ppriv $$
2075:  tcsh
flags = <none>
  E:basic,dtrace_proc,dtrace_user
  I:basic,dtrace_proc,dtrace_user
  P:basic,dtrace_proc,dtrace_user
  L:  all
```

Note – To update your privileges, ask your system administrator to add the `dtrace_user` and `dtrace_proc` privileges to your account in the `/etc/user_attr` file.

After the privileges have been changed, you can use the `ppriv` command to view the changed privileges.

Running DTrace with MPI Programs

There are two ways to use dynamic tracing with MPI programs:

- Run the MPI program directly under DTrace
- Attach DTrace to a running MPI program

Running an MPI Program Under DTrace

For illustration purposes, assume you have a program named `mpiapp`.

▼ To Trace a Program Using the `mpitrace.d` Script

- **Type the following command:**

```
% mpirun -np 4 dtrace -s mpitrace.d -c mpiapp
```

The advantage of tracing an MPI program in this way is that all the processes in the job will be traced from the beginning. This method is probably most useful in doing performance measurements, when you need to start at the beginning of an application and you need all the processes in a job to participate in collecting data.

This approach also has some disadvantages. One disadvantage of running a program like the one in the above example is that all the tracing output for all four processes is directed to standard output (`stdout`). One way around this problem is to create a script similar to the script in the following section.:

▼ To Trace a Parallel Program and Get Separate Trace Files

1. **Create a shell script (called `partrace.sh` in this example) similar to the following:**

```
#!/bin/sh
# partrace.sh - a helper script to dtrace Open MPI jobs from the
#     start of the job.
dtrace -s $1 -c $2 -o $2.$OMPI_COMM_WORLD_RANK.trace
```

2. **Type the following command to run the `partrace.sh` shell script:**

```
% mpirun -np 4 partrace.sh mpitrace.d mpiapp
```

This will run `mpiapp` under dtrace using the `mpitrace.d` script. The script saves the trace output for each process in a job under a separate file name, based on the program name and rank of the process. Note that subsequent runs will append the data into the existing trace files.

Note – The status of the `OMPI_COMM_WORLD_RANK.trace` variable is unstable and subject to change. Use this variable with caution.

Attaching DTrace to a Running MPI Program

The second way to use `dtrace` with Open MPI is to attach `dtrace` to a running MPI program.

▼ To Attach DTrace to a Running MPI Program

Perform the following procedure:

1. **Log in to the node in which you are interested.**
2. **Type commands similar to the following command to get the process ID (PID) of the running program on the node of interest.**

```
% prstat 0 1 | grep mpiapp
24768 joeuser     526M 3492K sleep   59    0   0:00:08 0.1% mpiapp/1
24770 joeuser     518M 3228K sleep   59    0   0:00:08 0.1% mpiapp/1
```

3. **Decide which rank you want to use to attach `dtrace`.**

   The lower PID number is usually the lower rank on the node.

4. **Type the following command to attach to the rank 1 process (identified by its process ID, which is 24770 in the example) and run the DTrace script `mpitrace.d`:**

```
% dtrace -p 24770 -s mpitrace.d
```

Simple MPI Tracing

DTrace enables you to easily trace programs. When used in conjunction with MPI and the more than 200 functions defined in the MPI standard, DTrace provides an easy way to determine which functions might be in error during the debugging process, or those functions that might be of interest. After you determine the function showing the error, it is easy to locate the desired job, process, and rank on which to run your scripts. As demonstrated above, DTrace allows you to perform these determinations while the program is running

Although the MPI standard provides the MPI profiling interface, using DTrace does provide a number of advantages. The advantages of using DTrace include the following:

- The PMPI interface requires you to restart a job every time you make changes to the interposing library.

- DTrace allows you to define probes that let you capture tracing information on MPI without having to code the specific details for each function you want to capture.
- The DTrace scripting language D has several built-in functions that help in debugging problematic programs.

The following example shows a simple script that traces the entry and exit into all the MPI API calls.

```
mpitrace.d:
pid$target:libmpi:MPI_*:entry
{
printf("Entered %s...", probefunc);
}

pid$target:libmpi:MPI_*:return
{
printf("exiting, return value = %d\n", arg1);
}
```

When you use this example script to attach DTrace to a job that performs `send` and `recv` operations, the output looks similar to the following:

```
% dtrace -q -p 24770 -s mpitrace.d
Entered MPI_Send...exiting, return value = 0
Entered MPI_Recv...exiting, return value = 0
Entered MPI_Send...exiting, return value = 0
Entered MPI_Recv...exiting, return value = 0
Entered MPI_Send...exiting, return value = 0 ...
```

You can easily modify the `mpitrace.d` script to include an argument list. The resulting output resembles `truss` output. For example:

```
mpitruss.d:
pid$target:libmpi:MPI_Send:entry,
pid$target:libmpi:MPI_*send:entry,
pid$target:libmpi:MPI_Recv:entry,
pid$target:libmpi:MPI_*recv:entry
{
printf("%s(0x%x, %d, 0x%x, %d, %d, 0x%x)",probefunc, arg0, arg1,
arg2, arg3, arg4, arg5);
}
pid$target:libmpi:MPI_Send:return,
pid$target:libmpi:MPI_*send:return,
pid$target:libmpi:MPI_Recv:return,
pid$target:libmpi:MPI_*recv:return
{
printf("\t\t = %d\n", arg1);
}
```

The `mpitruss.d` script shows how you can specify wildcard names to match the functions. Both probes will match all send and receive type function calls in the MPI library. The first probe shows the usage of the built-in `arg` variables to print out the `arglist` of the function being traced.

Take care when wildcarding the entrypoint and the formatting argument output, because you could end up printing either too many arguments, or not enough arguments, for certain functions. For example, in the above case, the `MPI_Irecv` and `MPI_Isend` functions will not have their Request handle parameters printed out.

The following example shows a sample output of the `mpitruss.d` script:

```
% dtrace -q -p 24770 -s mpitruss.d
MPI_Send(0x80470b0, 1, 0x8060f48, 0, 1,0x8060d48) = 0
MPI_Recv(0x80470a8, 1, 0x8060f48, 0, 0, 0x8060d48) = 0
MPI_Send(0x80470b0, 1, 0x8060f48, 0, 1, 0x8060d48) = 0
MPI_Recv(0x80470a8, 1,0x8060f48, 0, 0, 0x8060d48) = 0 ...
```

Tracking Down Resource Leaks

One of the biggest issues with programming is the unintentional leaking of resources (such as memory). With MPI, tracking and repairing resource leaks can be somewhat more challenging because the objects being leaked are in the middleware, and thus are not easily detected by the use of memory checkers.

DTrace helps with debugging such problems using variables, the profile provider, and a callstack function. The `mpicommcheck.d` script (shown in the example below) probes for all the MPI communicator calls that allocate and deallocate communicators, and keeps track of the stack each time the function is called. Every 10 seconds the script dumps out the current count of MPI communicator calls and the total calls for the allocation and deallocation of communicators. When the `dtrace` session ends (usually by pressing Ctrl-C, if you attached to a running MPI program), the script will print out the totals and all the different stack traces, as well as the number of times those stack traces were reached.

In order to perform these tasks, the script uses DTrace features such as variables, associative arrays, built-in functions (`count`, `ustack`) and the predefined variable `probefunc`.

The following example shows the `mpicommcheck.d` script.

```
mpicommcheck.d:
BEGIN
{
  allocations = 0;
  deallocations = 0;
  prcnt = 0;
}

pid$target:libmpi:MPI_Comm_create:entry,
pid$target:libmpi:MPI_Comm_dup:entry,
pid$target:libmpi:MPI_Comm_split:entry
{
  ++allocations;
  @counts[probefunc] = count();
  @stacks[ustack()] = count();
}

pid$target:libmpi:MPI_Comm_free:entry
{
  ++deallocations;
  @counts[probefunc] = count();
  @stacks[ustack()] = count();
}

profile:::tick-1sec
/++prcnt > 10/
{
  printf("============================================================
==============");
  printa(@counts);
  printf("Communicator Allocations = %d \n", allocations);
  printf("Communicator Deallocations = %d\n", deallocations);
  prcnt = 0;
}

END
{
  printf("Communicator Allocations = %d, Communicator
Deallocations = %d\n",
  allocations, deallocations);
}
```

This script attaches `dtrace` to a suspect section of code in your program (that is, a section of code that might contain a resource leak). If, during the process of running the script, you see that the printed totals for allocations and deallocations are starting to steadily diverge, you might have a resource leak. Depending on how your

program is designed, it might take some time and observation of the allocation/deallocation totals in order to definitively determine that the code contains a resource leak. Once you do determine that a resource leak is definitely occurring, you can press Ctrl-C to break out of the `dtrace` session. Next, using the stack traces dumped, you can try to determine where the issue might be occurring.

The following example shows code containing a resource leak, and the output that is displayed using the `mpicommcheck.d` script.

The sample MPI program containing the resource leak is called `mpicommleak`. This program performs three MPI_Comm_dup operations and two MPI_Comm_free operations. The program thus "leaks" one communicator operation with each iteration of a loop.

When you attach `dtrace` to `mpicommleak` using the `mpicommcheck.d` script above, you will see a 10-second periodic output. This output shows that the count of the allocated communicators is growing faster than the count of deallocations.

When you finally end the `dtrace` session by pressing Ctrl-C, the session will have output a total of five stack traces, showing the distinct three MPI_Comm_dup and two MPI_Comm_free call stacks, as well as the number of times each call stack was encountered.

For example:

```
% prstat 0 1 | grep mpicommleak
 24952 joeuser    518M 3212K sleep   59    0   0:00:01 1.8% mpicommleak/1
 24950 joeuser    518M 3212K sleep   59    0   0:00:00 0.2% mpicommleak/1
% dtrace -q -p 24952  -s mpicommcheck.d
=============================================================================
  MPI_Comm_free                                                           4
  MPI_Comm_dup                                                            6
Communicator Allocations = 6
Communicator Deallocations = 4
=============================================================================
  MPI_Comm_free                                                           8
  MPI_Comm_dup                                                           12
Communicator Allocations = 12
Communicator Deallocations = 8
=============================================================================
  MPI_Comm_free                                                          12
  MPI_Comm_dup                                                           18
Communicator Allocations = 18
Communicator Deallocations = 12
^C
Communicator Allocations = 21, Communicator Deallocations = 14

libmpi.so.0.0.0`MPI_Comm_free
              mpicommleak`deallocate_comms+0x19
              mpicommleak`main+0x6d
              mpicommleak`0x805081a
                7

              libmpi.so.0.0.0`MPI_Comm_free
              mpicommleak`deallocate_comms+0x26
              mpicommleak`main+0x6d
              mpicommleak`0x805081a
                7

              libmpi.so.0.0.0`MPI_Comm_dup
              mpicommleak`allocate_comms+0x1e
              mpicommleak`main+0x5b
              mpicommleak`0x805081a
                7

              libmpi.so.0.0.0`MPI_Comm_dup
              mpicommleak`allocate_comms+0x30
              mpicommleak`main+0x5b
              mpicommleak`0x805081a
                7

              libmpi.so.0.0.0`MPI_Comm_dup
              mpicommleak`allocate_comms+0x42
              mpicommleak`main+0x5b
              mpicommleak`0x805081a
                7
```

Using the DTrace `mpiperuse` Provider

PERUSE is an MPI interface that allows you to obtain detailed information about the performance and interactions of processes, software, and MPI. PERUSE provides a greater level of detail about process performance than does the standard MPI profiling interface (PMPI).

For more information about PERUSE and the current PERUSE specification, see:

`http://www.mpi-peruse.org`

Open MPI includes a DTrace provider named `mpiperuse`. This provider enables you to configure Open MPI to support DTrace probes into the Open MPI shared library `libmpi`.

DTrace Support in the ClusterTools Software

In Sun HPC ClusterTools 8.2.1c software, there are preconfigured executables and libraries with the `mpiperuse` provider probes built in. They are located in the `/opt/SUNWhpc/HPC8.2.1c/sun/instrument` directory. Use the wrappers and utilities located in this directory to access the `mpiperuse` provider.

Note – No recompilation is necessary in order to use the `mpiperuse` provider. Just run the application to be DTraced using `/opt/SUNWhpc/HPC8.2.1c/sun/instrument/bin/mpirun`.

Available `mpiperuse` Probes

The DTrace `mpiperuse` probes expose the events specified in the current PERUSE specification. These events track the life cycle of requests within the MPI library. For more information about this life cycle and the actual events provided by PERUSE, see Section 4 of the PERUSE Specification.

Sections 4.3.1 and 4.4 of the PERUSE Specification list and describe the individual events exposed by PERUSE.

The `mpiperuse` provider makes these events available to DTrace. The probe names correspond to the event names listed in Sections 4.3.1 and 4.4 of the PERUSE specification. For each event, the corresponding probe name is similar, except that

the leading PERUSE is removed, the probe name is all lowercase, and underscores are replaced with hyphens. For example, the probe for PERUSE_COMM_MSG_ARRIVED is comm-msg-arrived.

All of the probes are classified under the `mpiperuse` provider. This means that to find the probe names, you would look under the `mpiperuse` name. It also means that when you make a DTrace statement, you can include a wildcard for all probes simply by using the `mpiperuse` classification.

Specifying an `mpiperuse` Probe in a D Script

In the D scripting language, specifying an `mpiperuse` provider takes the following form:

```
mpiperuse$target:::probe-name
```

where *probe-name* is the name of the `mpiperuse` probe you want to use.

For example, to specify a probe to capture a PERUSE_COMM_REQ_ACTIVATE event, add the following line to a D script:

```
mpiperuse$target:::comm-req-activate
```

This alerts DTrace that you want to use the `mpiperuse` provider to capture the PERUSE_COMM_REQ_ACTIVATE event. In this example, the optional object and function fields in the probe description are omitted. This directs DTrace to find all occurrences of the `comm-req-activate` probes in the MPI library and its plugins instead of a specific probe. This is necessary because certain probes can happen in multiple places in the MPI library.

For more information about the D language and its syntax, refer to the *Solaris Dynamic Tracing Guide* (Part Number 817-6223). This guide is part of the Solaris 10 OS Software Developer Collection.

Available Arguments

All of the `mpiperuse` probes receive the following arguments:

TABLE 8-1 Available `mpiperuse` Arguments

`args[0] = mpiconninfo_t *i`	This provides a basic source and destination for the request and which protocol is expected to be used for the transfer. This typedef is defined in `/usr/lib/dtrace/mpi.d`.
`args[1] = uintptr_t uid`	This is the PERUSE unique id for the request that fired the probe (as defined by the PERUSE specifications). For OMPI this is the address of the actual request.
`args[2] = uint_t op`	This value indicates whether the probe is for a `send == 0` or `recv == 1` request.
`args[3] = mpicomm_spec_t *cs`	This structure is defined in `/usr/lib/dtrace/mpi.d` and mimics the `spec` structure, as defined on page 22 of the PERUSE specification.

How To Use `mpiperuse` Probes to See Message Queues

To use the `mpiperuse` provider, make reference to the appropriate `mpiperuse` provider probes and arguments in a DTrace script, as you would for any other provider (such as the `pid` provider).

The procedure for running scripts with `mpiperuse` probes follows the same steps as those shown in "Running an MPI Program Under DTrace" on page 74 and "Attaching DTrace to a Running MPI Program" on page 75, except that you must edit the `partrace.sh` script before you run it.

Change `partrace.sh` to include a `-Z` switch after the `dtrace` command, as shown in the following example.

```
#!/bin/sh
# partrace.sh - a helper script to dtrace Open MPI jobs from the
#     start of the job.
dtrace -Z -s $1 -c $2 -o $2.$OMPI_COMM_WORLD_RANK.trace
```

This change allows probes that do not exist at initial load time to be used in a script (that is, the probes are in plugins that have not been `dlopen`ed).

The following example shows how to use the `mpiperuse` probes when running a DTrace script. Use the example script provided in `/opt/SUNWhpc/HPC8.2.1c/sun/examples/dtrace/mpistat.d`

1. **Compile and run a script against a program.**

   In this example, the script file is called `dtest.c`. Substitute the name and path of your script for `dtest.c`.

```
% /opt/SUNWhpc/HPC8.2.1c/sun/instrument/bin/mpicc
~myhomedir/scraps/usdt/examples/dtest.c -o dtest
% /opt/SUNWhpc/HPC8.2.1c/sun/instrument/bin/mpirun -np 2 dtest
Initing MPI...
Initing MPI...
Do communications...
Do communications...
attach to pid 13371 to test tracing.
```

2. **In another window, type the following command:**

```
% dtrace -q -p 13371 -s /opt/SUNWhpc/HPC8.2.1c/sun/examples/dtrace/mpistat.d
input(Total) Q-sizes      Q-Matches            output
bytes active posted unexp posted unexp         bytes active
    0      0      0     0      0     0           0      0
    0      0      0     0      0     0           0      0
    0      0      0     0      0     0           0      0
    0      0      0     0      0     0           0      0
    0      0      0     0      1     0           5      0
    0      0      0     0      1     0           5      0
    0      0      0     0      1     0           5      0
    0      0      0     0      1     0           5      0
    0      0      0     0      1     0           5      0
    0      0      0     0      1     0           5      0
    0      0      0     0      1     0           5      0
    0      0      0     0      1     0           5      0
    0      0      0     0      2     0          10      0
    0      0      0     0      2     0          10      0
    0      0      0     0      2     0          10      0
    0      0      0     0      2     0          10      0
    0      0      0     0      2     0          10      0
    0      0      0     0      2     0          10      0
    0      0      0     0      2     0          10      0
    0      0      0     0      2     0          10      0
    0      0      0     0      3     0          15      0
    0      0      0     0      3     0          15      0
    0      0      0     0      3     0          15      0
    0      0      0     0      3     0          15      0
    0      0      0     0      3     0          15      0
    0      0      0     0      3     0          15      0
    0      0      0     0      3     0          15      0
```

`mpiperuse` Usage Examples

The examples in this section show how to perform the described DTrace operations from the command line.

▼ To Count the Number of Messages To or From a Host

- **Issue the following DTrace command, substituting the process ID of the process you want to monitor for *pid*:**

```
dtrace -p pid -n 'mpiperuse$target:::comm-req-xfer-end { @[args[0]->ci_remote]
= count(); }'
```

DTrace returns a result similar to the following. In this example, the process ID is 25428 and the host name is `joe-users-host2`.

```
% dtrace -p 25428 -n 'mpiperuse$target:::comm-req-xfer-end {@[args[0]-
>ci_remote] = count();}'
dtrace: description 'mpiperuse$target:::comm-req-xfer-end ' matched 17 probes
^C
joe-users-host2 recv 3
joe-users-host2 send 3
```

▼ To Count the Number of Messages To or From Specific BTLs

- **Issue the following DTrace command, substituting the process ID of the process you want to monitor for *pid*:**

```
dtrace -p pid -n 'mpiperuse$target:::comm-req-xfer-end { @[args[0]->ci_protocol]
= count(); }'
```

DTrace returns a result similar to the following. In this example, the process ID is 25445.

```
 % dtrace -p 25445 -n 'mpiperuse$target:::comm-req-xfer-end {@[args[0]-
>ci_protocol] = count();}'
dtrace: description 'mpiperuse$target:::comm-req-xfer-end ' matched 17 probes
^C

  sm 60
```

▼ To Obtain Distribution Plots of Message Sizes Sent or Received From a Host

- **Issue the following DTrace command, substituting the process ID of the process you want to monitor for *pid*:**

```
dtrace -p pid -n 'mpiperuse$target:::comm-req-xfer-end { @[args[0]->ci_remote]
= quantize(args[3]->mcs_count); }'
```

DTrace returns a result similar to the following. In this example, the process ID is 25445.

```
 % dtrace -p 25445 -n 'mpiperuse$target:::comm-req-xfer-end {@[args[0]-
>ci_remote] = quantize(args[3]->mcs_count);}'
dtrace: description 'mpiperuse$target:::comm-req-xfer-end ' matched 17 probes
^C

  myhost
           value  ------------- Distribution ------------- count
               2 |                                         0
               4 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 4
               8 |                                         0
```

▼ To Create Distribution Plots of Message Sizes By Communicator, Rank, and Send/Receive

- **Issue the following DTrace command, substituting the process ID of the process you want to monitor for *pid*:**

```
dtrace -p pid -n 'mpiperuse$target:::comm-req-xfer-end {@[args[3]->mcs_comm,
args[3]->mcs_peer, args[3]->mcs_op] = quantize(args[3]->mcs_count);}'
```

DTrace returns a result similar to the following. In this example, the process ID is 24937.

```
% dtrace -p 24937 -n 'mpiperuse$target:::comm-req-xfer-end {@[args[3]->mcs_comm,
args[3]->mcs_peer, args[3]->mcs_op] = quantize(args[3]->mcs_count);}'
dtrace: description 'mpiperuse$target:::comm-req-xfer-end ' matched 19 probes
^C
        134614864        1  recv
           value  ------------- Distribution ------------- count
               2 |                                         0
               4 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 9
               8 |                                         0
```

```
        134614864        1  send
           value  ------------- Distribution ------------- count
               2 |                                         0
               4 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 9
               8 |                                         0
```

APPENDIX **A**

Troubleshooting

This appendix describes some common problem situations, resulting error messages, and suggestions for fixing the problems. Open MPI error reporting, including I/O, follows the *MPI-2 Standard*. By default, errors are reported in the form of standard error classes. These classes and their meanings are listed in TABLE A-1 (for non-I/O MPI) and TABLE A-2 (for MPI I/O), and are also available on the `MPI` man page.

MPI Messages

Standard Error Classes

Listed below are the error return classes you might encounter in your MPI programs. Error values can also be found in `mpi.h` (for C), `mpif.h` (for Fortran), and `mpi++.h` (for C++).

TABLE A-1 Open MPI Standard Error Classes

Error Code	Value	Meaning
`MPI_SUCCESS`	`0`	Successful return code.
`MPI_ERR_BUFFER`	`1`	Invalid buffer pointer.
`MPI_ERR_COUNT`	`2`	Invalid count argument.
`MPI_ERR_TYPE`	`3`	Invalid datatype argument.
`MPI_ERR_TAG`	`4`	Invalid tag argument.
`MPI_ERR_COMM`	`5`	Invalid communicator.

TABLE A-1 Open MPI Standard Error Classes *(Continued)*

Error Code	Value	Meaning
MPI_ERR_RANK	6	Invalid rank.
MPI_ERR_ROOT	7	Invalid root.
MPI_ERR_GROUP	8	Null group passed to function.
MPI_ERR_OP	9	Invalid operation.
MPI_ERR_TOPOLOGY	10	Invalid topology.
MPI_ERR_DIMS	11	Illegal dimension argument.
MPI_ERR_ARG	12	Invalid argument.
MPI_ERR_UNKNOWN	13	Unknown error.
MPI_ERR_TRUNCATE	14	Message truncated on receive.
MPI_ERR_OTHER	15	Other error; use `Error_string`.
MPI_ERR_INTERN	16	Internal error code.
MPI_ERR_IN_STATUS	17	Look in status for error value.
MPI_ERR_PENDING	18	Pending request.
MPI_ERR_REQUEST	19	Illegal `MPI_Request()` handle.
MPI_ERR_KEYVAL	36	Illegal key value.
MPI_ERR_INFO	37	Invalid info object.
MPI_ERR_INFO_KEY	38	Illegal info key.
MPI_ERR_INFO_NOKEY	39	No such key.
MPI_ERR_INFO_VALUE	40	Illegal info value.
MPI_ERR_TIMEDOUT	41	Timed out.
MPI_ERR_SYSRESOURCES	42	Out of resources.
MPI_ERR_SPAWN	45	Error spawning.
MPI_ERR_WIN	46	Invalid window.
MPI_ERR_BASE	47	Invalid base.
MPI_ERR_SIZE	48	Invalid size.
MPI_ERR_DISP	49	Invalid displacement.
MPI_ERR_LOCKTYPE	50	Invalid locktype.

TABLE A-1 Open MPI Standard Error Classes *(Continued)*

Error Code	Value	Meaning
MPI_ERR_ASSERT	51	Invalid assert.
MPI_ERR_RMA_CONFLICT	52	Conflicting accesses to window.
MPI_ERR_RMA_SYNC	53	Erroneous RMA synchronization.
MPI_ERR_NO_MEM	54	Memory exhausted.
MPI_ERR_LASTCODE	55	Last error code.

MPI I/O Error Handling

Open MPI I/O error reporting follows the *MPI-2 Standard*. By default, errors are reported in the form of standard error codes (found in `/opt/SUNWhpc/include/mpi.h`). Error classes and their meanings are listed in TABLE A-2. They can also be found in `mpif.h` (for Fortran) and `mpi.h` (for C).

You can change the default error handler by specifying `MPI_FILE_NULL` as the file handle with the routine `MPI_File_set_errhandler()`, even if no file is currently open. Or, you can use the same routine to change the error handler for a specific file.

TABLE A-2 Open MPI I/O Error Classes

Error Class	Value	Meaning
MPI_ERR_FILE	20	Bad file handle.
MPI_ERR_NOT_SAME	21	Collective argument not identical on all processes.
MPI_ERR_AMODE	22	Unsupported `amode` passed to open.
MPI_ERR_UNSUPPORTED_DATAREP	23	Unsupported `datarep` passed to `MPI_File_set_view()`.
MPI_ERR_UNSUPPORTED_OPERATION	24	Unsupported operation, such as seeking on a file that supports only sequential access.
MPI_ERR_NO_SUCH_FILE	25	File (or directory) does not exist.
MPI_ERR_FILE_EXISTS	26	File exists.

TABLE A-2 Open MPI I/O Error Classes *(Continued)*

Error Class	Value	Meaning
MPI_ERR_BAD_FILE	27	Invalid file name (for example, path name too long).
MPI_ERR_ACCESS	28	Permission denied.
MPI_ERR_NO_SPACE	29	Not enough space.
MPI_ERR_QUOTA	30	Quota exceeded.
MPI_ERR_READ_ONLY	31	Read-only file system.
MPI_ERR_FILE_IN_USE	32	File operation could not be completed, as the file is currently open by some process.
MPI_ERR_DUP_DATAREP	33	Conversion functions could not be registered because a data representation identifier that was already defined was passed to MPI_REGISTER_DATAREP.
MPI_ERR_CONVERSION	34	An error occurred in a user-supplied data-conversion function.
MPI_ERR_IO	35	I/O error.
MPI_ERR_INFO	37	Invalid info object.
MPI_ERR_INFO_KEY	38	Illegal info key.
MPI_ERR_INFO_NOKEY	39	No such key.
MPI_ERR_INFO_VALUE	40	Illegal info value.
MPI_ERR_LASTCODE	55	Last error code.

Exceeding the File Descriptor Limit

If your application tries to open a file descriptor when the maximum limit of open file descriptors has been reached, the job will fail and display the following message:

```
% =>mpirun -np 64 foo -v
% ORTE_ERROR_LOG: The system limit on number of pipes a process can open was
reached in file base/iof_base_setup.c at line 115
% ORTE_ERROR_LOG: The system limit on number of pipes a process can open was
reached in file odls_default_module.c at line 233
```

```
% The system limit on number of network connections a process can open was reached
in file oob_tcp.c at line 447
--------------------------------------------------------------------------
Error: system limit exceeded on number of network connections that can be open

This can be resolved by setting the mca parameter opal_set_max_sys_limits to 1,
increasing your limit descriptor setting (using limit or ulimit commands),
or asking the system administrator to increase the system limit.
--------------------------------------------------------------------------
```

Should this occur, do as the message says and try `--mca opal_set_max_sys_limits 1`. Alternatively, you need to increase the number of file descriptors.

The Solaris OS default file descriptor limit is 256. When you start an MPI job, a program called an `orted` (for ORTE daemon) spawns the user processes. For each user process spawned, the `orted` takes up four file descriptors. In addition, the job takes 12 additional file descriptors regardless of the number of processes spawned.

To calculate the number of file descriptors needed to run a certain job, use the following formula:

```
file descriptors = 12 + 4 * np
```

where `np` is the number of processes launched.

If the number of file descriptors needed is greater than 256, you must increase the number of available descriptors to a value equal to or greater than the number you calculated. Otherwise, the processes fail and the error message is displayed.

Increasing the Number of Available File Descriptors

▼ To View the Hard Limit from the C Shell

1. **Log in to a C shell as superuser.**
2. **Determine the current hard limit value for your Solaris implementation. Type the following command:**

```
# limit -h descriptors
```

▼ To View the Hard Limit from the Bourne Shell

1. **Log in to a Bourne shell as superuser.**
2. **Use the `ulimit` function. Type the following command:**

```
# ulimit -Hn
```

Each function returns the file descriptor hard limit that was in effect. The new value you set for the number of available file descriptors must be less than or equal to this number. The usual default value for the hard limit in the Solaris OS is 64000 (64K).

▼ To Increase the Number of File Descriptors

Note – You must perform this procedure on each of the nodes on which you plan to run.

---

1. **Open the `/etc/system` file in a text editor.**
2. **Add the following line to the file:**

```
set rlim_fd_cur=value
```

where value is the new maximum number of file descriptors. For example, the following line added to the /etc/system file increases the maximum number of file descriptors to 1024:

```
set rlim_fd_cur=1024
```

3. **Save the file and exit the text editor.**
4. **Reboot the system.**

Setting File Descriptor Limits When Using Sun Grid Engine

If you are using Sun Grid Engine to launch your jobs on very large multi-processor nodes, you might see an error message about exceeding your file descriptor limit, and your jobs might fail. This can happen because Sun Grid Engine cannot set the file descriptor limit in its queue.

There are three ways in which you can adjust the number of available file descriptors when you use Sun Grid Engine:

1. Set the file descriptor limit in your login shell (`.cshrc`, `.tcshrc`, `.bashrc`, and so on).
2. Modify the `/etc/shell` file for each of the nodes on your cluster as described in the previous section, "To Increase the Number of File Descriptors" on page 94. Remember that you must reboot all of the nodes in the cluster once you have finished modifying the files.
3. On a Sun Grid Engine execution host, modify the `$SGE_ROOT/default/common/sgeexecd` startup script to increase the file descriptor limit to the same value as the hard limit (as described in the previous section). You must restart the `sgeexecd` daemon on the host. Since this script is shared among the Sun Grid Engine execution hosts in the cluster using NFS, you may make the change on one host, and it will be propagated to the other Sun Grid Engine hosts in the cluster.

Index

W