PathScale™ InfiniPath™ Interconnect User's Guide

PathScale, Inc.

Version 1.0

PREFACE

About This Guide

This Preface describes the objectives, intended audience, and organization of the *PathScale InfiniPath Interconnect User Guide*

The *PathScale InfiniPath Interconnect User Guide* is intended to give the end users of an InifiniPath cluster what they need to know to use it. In this case, end users are understood to include both the cluster administrator and the MPI application programmers, who have different but overlapping interests in the details of the technology.

For specific instructions about installing the PathScale InfiniPath ™ HT4X Adapter, and the initial installation of the InifiniPath Software, see the *PathScale InfiniPath Interconnect Installation Guide*.

Who should read this Guide

This *Guide* is intended both for readers responsible for administration of an InfiniPath cluster network and for readers wanting to use that cluster.

This Guide assumes that all readers are familiar with cluster computing, that the cluster administrator reader is familiar with Linux administration and that the application programmer reader is familiar with MPI.

How this Guide is organized

The *PathScale InfiniPath Interconnect User Guide* is organized into these sections:

Chapter 1 “InfiniPath Interconnect Overview” briefly defines an InfiniPath cluster, consisting of a collection of nodes connected by the PathScale InfiniPath Interconnect. This is of interest to both cluster administrators and MPI programmers.

Chapter 2 “Using PathScale MPI” helps the MPI programmer make best use of the PathScale MPI implementation.

Chapter 3 “InfiniPath Cluster Administration” describes the lower levels of the supplied InfiniPath software. This would be of interest mainly to an InfiniPath cluster administrator.

Appendix A. Benchmark Programs

Appendix B. Troubleshooting

Appendix C. Recommended reading

Please refer to http://www.pathscale.com for updates to the information provided in this Guide.

Conventions used in this document

Convention	Meaning
`command`	Fixed-space font is used for literal items such as commands, functions, programs, files and path names.
`variable`	Italic fixed-space font is used for variable names in programs and command lines, indicating parameter values that you supply.
concept	Italic font is used for emphasis, concepts, and publication titles.
`user input, system output`	Fixed-space font is used for code output, and for commands or constructs you type in.
`$`	Indicates a command line prompt.
#	Indicates a command line prompt as root.
[ ]	Brackets enclose optional elements of a command or program construct.
...	Ellipses indicate that a preceding element can be repeated.
>	Right caret identifies the path of menu commands used in a procedure.
NOTE:	Indicates important information.

CHAPTER 1

InfiniPath Interconnect Overview

This chapter provides an overview of the PathScale InfiniPath Interconnect system and its software. It should be of interest both to the administrator and the user of an InfiniPath cluster.

1.1 InfiniPath Interconnect overview

The material in this *User's Guide* pertains to an InfiniPath cluster. This is defined as a collection of nodes each attached to an InfiniBand-based fabric through the PathScale InfiniPath Interconnect. The nodes are Linux-based computers, each having up to eight

processors. The PathScale InfiniPath HT-460 adapter is exclusively for AMD Opteron™-based motherboards with a standard HyperTransport™ HTX™ slot.

The InfiniPath interconnect is InfiniBand 4X, with a raw data rate of 10 Gb/s (payload rate of 8Gb/s).

InfiniPath utilizes standard, off-the-shelf InfiniBand 4X switches and cabling. At least one InfiniBand switch in the fabric must be a *managed* switch, that is, it must have an active Subnet Manager present and enabled. The InfiniPath interconnect uses InfiniBand multicast for some functions, and the Subnet Manager must support multicast group creation and management.

The currently tested and supported 24-port switches are:

- SilverStorm 9024 managed
- TopSpin 120 managed
- Voltaire ISR-9024 managed
- Mellanox MTS 2400 unmanaged

Larger switches will be tested and supported in future releases. See the PathScale Web page for updates to this list of supported switches.

InfiniPath participates in the standard InfiniBand Subnet Management protocols for configuration and monitoring. In the present version, it does not interoperate with other InfiniBand Host Channel Adapters (HCA) and Target Channel Adapters (TCA). Instead, it uses a protocol that is highly optimized for MPI and TCP between InfiniPath-equipped hosts.

The InfiniPath interconnect initially runs on AMD Opteron systems running Linux. This 1.0 release is supported on the SuSE 9.3 and Fedora Core 3 distributions, with the 2.6.11 Linux kernel. Future releases will be supported on more distributions and later kernels.

The software provided with the PathScale InfiniPath Interconnect product consists of

- PathScale MPI
- Protocol Libraries
- Layered Ethernet driver
- Subnet Management Agent,
- InfiniPath Driver

This list is ordered (roughly) from higher to lower levels of software. PathScale MPI is PathScale's implementation of the standard Message Passing Interface, which is the dominant method for programming parallel applications on clusters. Chapter 2 discusses it in some detail. The next four items constitute a base of software that supports MPI programming on an InfiniPath fabric. They are discussed in Chapter 3.

CHAPTER 2

Using PathScale MPI

2.1 PathScale MPI

PathScale's implementation of the MPI standard is derived from the MPICH reference implementation Version 1.2.6. The InfiniPath MPI libraries have been highly tuned for the InfiniPath Interconnect, and will not run over other interconnects.

PathScale MPI is an implementation of the original MPI 1.2 standard. The MPI-2 standard provides several enhancements of the original standard. Of the MPI-2 features, PathScale MPI includes only the MPI-IO features implemented in ROMIO version 124.

2.2 Getting started with MPI

In this section you will learn how to compile and run some simple example programs that are included in the InfiniPath software product

These examples assume that

• your cluster administrator has properly installed PathScale MPI and the EKOPath compilers.

• your cluster's policy allows you to use the `mpirun` script directly, without having to submit the job to a batch queuing system.

• you or your administrator has properly set up your `ssh` keys and associated files on your cluster. See Sections 2.4.1 , Configuring for ssh, and 3.6 for details on `ssh` administration.

To begin, copy the examples to your working directory:

```
cp /usr/share/mpich/examples/basic/* .
```

Next, create an MPI hosts file in the same working directory. It contains the host names of the nodes in your cluster on which you want to run the examples, with one host name per line. Name this file `mpihosts`.

Compiling and running these examples will let you verify that PathScale MPI and its components have been properly installed on your cluster. See Appendix C on troubleshooting if you have trouble compiling or running these examples.

2.2.1 An example C program

PathScale MPI makes use of some shell scripts to handle the burden of finding the appropriate include files and libraries for each supported language. Use the script `mpicc` to compile an MPI program in C and the script `mpirun` to execute it.

The supplied example program `cpi.c` computes an approximation to pi. First, compile it to an executable named `cpi`.

```
mpicc -o cpi cpi.c
```

`mpicc`, by default, runs the PathScale `pathcc` compiler, and is used for both compiling and linking, exactly as you'd use the `pathcc` or `gcc` command.

Then, run it with several different specifications for the number of processes:

```
$ mpirun -np 2 -m mpihosts ./cpi
Process 0 on host1
Process 1 on host2
pi is approximately 3.1416009869231241,
Error is 0.0000083333333309
wall clock time = 0.000149
```

Here `./cpi` designates the executable of the example program in the working directory. The `-np` parameter to `mpirun` defines the number of processes to be used in the parallel computation. Now try it with four processes.

```
$ mpirun -np 4 -m mpihosts ./cpi
Process 3 on host1
Process 0 on host2
Process 2 on host2
Process 1 on host1
pi is approximately 3.1416009869231249,
Error is 0.0000083333333318
wall clock time = 0.000603
```

Note that you may get the same output lines in a different order than shown, and if you run the program several times with the same value of the `-np` parameter, you may get the output lines in different orders because they are issued by independent asynchronous processes, and so their order is non-deterministic.

Note that the number of node programs can be greater than the number of nodes. In this four-process example, the hosts file listed only two hosts, host1 and host2. Generally, `mpirun` will try to distribute the specified number of processes evenly among the nodes listed in the hosts file, but if the number of processors exceeds the number of nodes listed in the hosts file, then some nodes will be assigned more than one instance of the program.

Up to a limit, the number of processes can even exceed the total number of processors on the specified set of nodes, although it is usually detrimental to performance to have more

than one node program per processor. See Section 2.4.8 , Multiprocessor nodes, for further discussion of this point.

Details on alternate means of specifying the MPI hosts file are given in Section 2.4.5. Further information on the mpirun options are in Sections 2.4.4, 2.4.8 and 2.4.9.

2.2.2 Other languages

`fpi.f` is a Fortran77 program that computes pi in a way similar to cpi.c. Compile and link it with

```
$ mpif77 -o pi3 pi3.f
```

and run it with

```
$ mpirun -np 2 -m mpihosts ./pi3
```

`pi3f90.f90` in the same directory is a Fortran90 program that does essentially the same computation. Compile and link it with

```
$ mpif90 -o pi3f90 pi3f90.f90
```

and run it with

```
$ mpirun -np 2 -m mpihosts ./pi3f90
```

The C++ program `hello++.cc` is a parallel processing version of the traditional “Hello, World” program. Notice that this version makes use of the external C bindings of the MPI functions if the C++ bindings are not present.

Compile it

```
$ mpicxx -o hello hello++.cc
```

and run it

```
$ mpirun -np 10 -m mpihosts ./hello
    Hello World! I am 9 of 10
```

```
Hello World! I am 2 of 10
Hello World! I am 4 of 10
Hello World! I am 1 of 10
Hello World! I am 7 of 10
Hello World! I am 6 of 10
Hello World! I am 3 of 10
Hello World! I am 0 of 10
Hello World! I am 5 of 10
Hello World! I am 8 of 10
```

Each of the scripts invokes the PathScale EKOPath compiler for the respective language and the PathScale linker. See Section 2.4.3 for how to use the `gcc` compiler. The use of `mpirun` is the same for programs in all languages.

2.3 Configuring the building of InfiniPath MPI applications

When configuring an MPI program (generating header files and/or Makefiles), for InfiniPath MPI, you will usually need to specify `mpicc`, `mpif90`, etc. as the compiler, rather than `pathcc`, `pathf90`, etc.

Typically this is done with commands similar to these (this assumes you are using `sh` or `bash` as your shell):

```
export CC=mpicc
export CXX=mpicxx
export F77=mpif77
export F90=mpif90
```

The shell variables will vary with the program being configured, but these examples show frequently used variable names. Users of `csh` would instead use commands similar to:

```
setenv CC mpicc
```

You may need to instead pass arguments to `configure` directly, in a fashion similar to this:

```
./configure -cc=mpicc -fc=mpif77 -c++=mpicxx c++linker=mpicxx
```

Sometimes you may need to edit a `Makefile` to achieve this result, adding lines similar to:

```
CC=mpicc
F77=mpif77
F90=mpif90
CXX=mpicxx
```

In some cases, the configuration process may specify the linker. It is recommended that the linker be specified as `mpicc`, `mpif90`, etc. in these cases, because that will automatically include the correct flags and libraries, rather than trying to configure to pass the flags and libraries explicitly. For example:

```
LD=mpicc
```

or

```
LD=mpif90
```

These scripts pass appropriate options to the various compiler passes to include header files, required libraries, etc. While the same effect can be achieved by passing the arguments explicitly as flags, the required arguments may vary from release to release, so it's good practice to use the provided scripts.

2.4 PathScale MPI details

This section gives more details on the use of PathScale MPI. We assume the reader has some familiarity with standard MPI. See the references in Appendix C, Recommended Reading. The PathScale implementation does include the `man` pages from the MPICH implementation for the numerous MPI functions.

2.4.1 Configuring for ssh

The command `mpirun` is generally run on the front end or head node (although it can be run from any node). In the PathScale implementation of MPI, this uses the secure shell command `ssh` to start instances of the given MPI program on the remote compute nodes. Therefore, first the user must have generated RSA or DSA keys, public and private, and have distributed the public keys to all the compute nodes to be used.

Then, provisions have to be made to allow `mpirun` to run commands on the compute nodes without requiring interactive response to password prompts from each node. Each user can accomplish this through use of the `ssh-agent` command. Alternatively, the cluster administrator can accomplish this for all users through the `shosts.equiv` mechanism, as described in Section 3.6 , Configuring secure shell,.

2.4.2 Compiling and linking

The scripts

```
mpicc
mpicxx
mpif77
mpif90
```

invoke the compiler and linker for programs in each of the respective languages, and take care of referring to the correct include files and libraries in each case. By default these call the PathScale EKOPath compiler and linker.

These scripts all provide the following command line options:

`-help` to get help.

`-show` to list each of the compiling and linking commands that would be called without actually calling them.

`-echo` to get verbose output of all the commands in the script.

`-compile_info` to show how to compile a program.

`-link_info` to show how link a program.

Further, each of these scripts admits a command line option for specifying the use of a different compiler/linker as an alternative to the EKOPath Compiler Suite. These are described in the next section.

Most other given command line options are passed on to the invoked compiler and linker. The EKOPath compiler and the usual alternatives all admit numerous command line options. See the EKOPath documentation and the `man` pages for `pathcc` and `pathf90` for complete information on its options. See the corresponding documentation for any other compiler/linker you may call for its options.

2.4.3 To use another compiler

Presently, in addition to the PathScale EKOPath Compiler Suite, PathScale MPI supports the GNU `gcc`. To use `gcc` for compiling and linking MPI programs in C, use

```
mpicc  -cc=gcc .......
```

To use `gcc` for compiling and linking C++ programs use

```
mpicxx  -CC=gcc .......
```

To use gcc for compiling and linking Fortran77 programs use

```
mpif77  -fc=g77 .......
```

where, in each case, `.....` stands for the remaining options to the `mpicxx` script, the options to the compiler in question, and the names of the files it is to operate upon.

2.4.4 Running MPI programs

The script `mpirun` lets you start your parallel MPI program on a set of nodes in a cluster. It starts, monitors, and terminates the node programs. The general syntax is

`mpirun [`*`mpirun_options...`*`] `*`program-name`*`[`*`program options`*`]`

`program-name` will generally be the pathname to the executable MPI program. If the MPI program resides in the current directory and the current directory is not in your search path, then *`program-name`* must begin with `./`.

Unless you want to run only one instance of the program, you need to use the `-np` option, as in

`mpirun -np `*`n`*` [`*`other options`*`] `*`program-name`*

which will spawn *`n`* instances of *`program-name`*. We usually call these instances *node programs*.

Each node program is started as a process on one node. While it is certainly possible for a node program to spawn child processes, the children must not themselves call MPI functions.

`mpirun` monitors the parallel MPI job, terminating when all the node programs in that job exit normally, or if any of them terminates abnormally.

2.4.5 The MPI hosts file

There must be a specified MPI hosts file (also sometimes called a “machines file”), which names the nodes on which the node programs may run. The MPI hosts file contains lines of the form

`hostname[:p]`

where the optional part *`:p`* specifies the number of node programs that can be spawned on that node. When not specified, the default value is 1. This optional part is discussed below in Section 2.4.8 , Multiprocessor nodes,.

You have several alternative ways of specifying the MPI hosts file. First, as noted in Section 2.2 , Getting started with MPI,, you can use the command line option `-m`,

```
mpirun -np n -m hfile [other options] program-name
```

In this case, if the named file cannot be opened, the MPI job fails.

If the `-m` option is omitted, `mpirun` checks the environment variable `MPIHOSTS` for the name of the MPI hosts file. If this variable is defined and the file it names cannot be opened, then the MPI job fails.

In the absence of both the `-m` option and the `MPIHOSTS` environment variable, `mpirun` uses the file `./mpihosts`, if it exists.

If none of these three methods of specifying the hosts file are used, `mpirun` looks for the file `~/.mpihosts`.

If you are working in the context of a batch queuing system, it may provide you with a job submission script that generates an appropriate machines file.

2.4.6 Console I/O in MPI programs

`mpirun` sends any output printed to `stdout` or `stderr` by any node program to the terminal. This output is line-buffered, so the lines output from the various node programs will be non-deterministically interleaved on the terminal. Using the `-l` option to `mpirun` will label each line with the rank of the node program that produced it.

Node programs do not normally use interactive input on `stdin`, and by default, `stdin` is bound to `/dev/null`. However, the `-stdin` option to `mpirun` lets you redirect a specified file to the `stdin` of all node programs or to a particular node program specified by the `-stdin-target` option.

2.4.7 Environment for node programs

The environment variables extant on the front end node on which you run `mpirun` are not propagated to the other nodes. You can set the paths, such as `LD_LIBRARY_PATH`, and other environment variables for the node programs through the use of the `-rcfile` option of `mpirun`:

```
mpirun -np n -m hostfile -rcfile mpirunrc program
```

In the absence of this option, `mpirun` checks to see if a file called `$HOME/.mpirunrc` exists in the user's home directory. In either case, the file is sourced by the shell on each node at time of startup of the node program.

The *`mpirunrc`* should not contain any interactive commands. It may contain commands that output on `stdout` or `stderr`.

When you do not specify an *`mpirunrc`* file, either through the option or the default ~/`mpirunrc`, the environment on each node is whatever it would be for the user's login via `ssh`, unless you are using mpd. (See Section 2.5 , Using MPD,)

2.4.8 Multiprocessor nodes

Another command line option, *`-ppn`*, instructs `mpirun` to assign a fixed number *p* of node programs to each node, as it distributes the *n* instances among the nodes.

```
mpirun -np n -m hostfile -ppn p program-name
```

This option overrides the `:p` specifications, if any, in the lines of the MPI hosts file.

As a general rule, `mpirun` tries to distribute the *n* node programs among the nodes without exceeding on any node the maximum number of instances specified by the `:p` option from the MPI hosts file or the value specified by the `-ppn` command line option. Normally, the number of node programs on a node should be no larger than the number of processors on the node, at least not for compute-bound problems. In the current implementation of the InfiniPath interconnect, no node can run more than eight node programs.

For improved performance, PathScale MPI uses shared memory to pass messages between node programs running on the same host.

2.4.9 mpirun options

Here is a list summarizing the most commonly used options to `mpirun`. See the man page for a more complete listing

`-np` *`n`*

Number of processes to create.

`-ppn` *`n`*

Number of processes per node.

`-machinefile, -m` *`filename`*

Machines file, the list of hosts to be used for this job.
Default: `$MPIHOSTS`, then `./mpihosts`, then `~/.mpihosts`

`-verbose`

Print diagnostic messages from mpirun itself. Can be useful in trouble shooting
Default: Off

`-version, -v`

Print MPI version. Default: Off

`-help, -h`

Print mpirun help message. Default: Off

`-rcfile` *`filename`*

Startup script for setting environment on nodes.
Default: `$HOME/.mpirunrc`

`-in-xterm`

Run each process in an xterm window. Default: Off

`-display` *`display-name`*

X Display for xterm. Default: None

`-debug`

Run each process under debugger in an xterm window. Default: Off

`-debug-no-pause`

Like debug, except doesn't pause at beginning. Default: Off

`-debugger` *`debugger-name`*

Which debugger to use. Default: gdb

`-psc-debug-level, -d` *`level`*

Controls the verbosity of MPI and InfiniPath debug messages for node programs Default: 1

`-nonmpi`

Run a non-MPI program. Required if the node program makes no MPI calls. Default: Off

`-quiescence-timeout, -q` *`nsecs`*

Wait time in seconds for quiescence (absence of MPI communication) on the nodes. Useful for detecting deadlocks. 0 disables quiescence detection. Default: 900

`-label-output, -l`

Label lines of output on `stdin` and `stderr` with the MPI rank of the process that wrote them. Default: Off

`-stdin` *`filename`*

Filename that should be fed as stdin to the node program. Default: /dev/null

`-stdin-target` *`rank`*

Process rank that should receive the `stdin` file specified with the `-stdin` option. -1 means all ranks. Default: -1

`-wdir` *`dir`*

Sets the working directory for the node program. Default: `-wdir` *`dir`*

`-print-stats`

Causes each node program to print various MPI statistics to `stderr` on job termination. Can be useful in troubleshooting. See Appendix B. Default off.

`-statsfile` *`filename`*

Specifies alternate file to receive the output from the `-print-stats` option. See Appendix B. Default `stderr`.

.

2.5 Using MPD

2.5.1 What is MPD?

MPD is an alternative to `mpirun` for launching MPI jobs. It was also developed by Argonne National Laboratory as part of the MPICH-2 system. While the ANL MPD had certain advantages over the use of their `mpirun` (faster launching, better cleanup after crashes, better tolerance of node failures), the PathScale `mpirun` offers the same advantages.

The disadvantage of MPD is reduced security, since it does not use `ssh` to launch node programs. It is also a little more complex to use than `mpirun` because it requires starting a ring of MPD daemons on the nodes. Therefore, most users should use the normal `mpirun` mechanism for starting jobs as described in the previous chapter. However, for users who wish to use MPD, a PathScale implementation is included in the InfiniPath software.

2.5.2 Using MPD

To start an MPD environment, use the `mpdboot` program. You must provide `mpdboot` with a file listing the machines on which to run the `mpd` daemon. The format of this file is the same as for the `mpirun` command.

Here is an example of how to run mpdboot:

```
mpdboot -f hostsfile
```

After `mpdboot` has started the MPD daemons, it will print a status message and drop you into a new shell.

To leave the MPD environment, exit from this shell. This will terminate the daemons.

To run an MPI program from within the MPD environment, use the `mpirun` command. You do not need to provide a node file or a count of CPUs; by default, `mpirun` will use all nodes and CPUs available within the MPD environment.

To check the status of the MPD daemons, use the `mpdping` command.

2.6 File I/O in MPI

2.6.1 Linux File I/O in MPI programs

MPI node programs are Linux programs, which can do file I/O to local or remote files in the usual ways through APIs of the language in use. Remote files are accessed via some network file system, typically NFS. Parallel programs usually need to have some data in files to be shared by all of the processes of an MPI job. Node programs may also use non-shared, node-specific files, such as for scratch storage for intermediate results or for a node's share of a distributed database.

There are different styles of handling file I/O of shared data in parallel programming. You may have one process, typically on the front end node or on a file server, which is the only process to touch the shared files, and which passes data to and from the other processes via MPI messages. On the other hand, the shared data files could be accessed directly by each node program. In this case, the shared files would be available through some network file support, such as NFS. Also, in this case, the application programmer would be responsible for ensuring file consistency, either through proper use of file locking mechanisms offered by the OS and the programming language, such as `fcntl` in C, or by the use of MPI synchronization operations.

2.6.2 MPI-IO with ROMIO

MPI-IO is the part of the MPI2 standard, supporting collective and parallel file IO. One of the advantages in using MPI-IO is that it can take care of managing file locks in case of file data shared among nodes.

The PathScale MPI implementation includes ROMIO, a high-performance, portable implementation of MPI-IO from Argonne National Laboratory. ROMIO includes everything defined in the MPI-2 I/O chapter of the MPI-2 standard except support for file interoperability and user-defined error handlers for files. See the ROMIO documentation in http://www.mcs.anl.gov/romio for details.

2.7 Debugging MPI programs

Debugging parallel programs is substantially more difficult than debugging serial programs. Thoroughly debugging the serial parts of your code before parallelizing is good programming practice.

2.7.1 MPI errors

Almost all MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error code; as the function return value in C functions or as the last argument in a Fortran subroutine call. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. Therefore you can get information about MPI exceptions in your code by providing your own handler for `MPI_ERRORS_RETURN`. See the `man` page for `MPI_Errhandler_set` for details. Note that MPI does not guarantee that an MPI program can continue past an error.

See the standard MPI documentation referenced in Appendix C for details on the MPI error codes.

2.7.2 Using debuggers

The InfiniPath software supports the use of multiple debuggers, including `pathdb`, `gdb`, and the stack trace utility `strace`. Both debuggers let you set breakpoints in a running program, and examine and set its variables.

Of course, symbolic debugging is easier than machine language debugging. To enable symbolic debugging you must have compiled with the `-g` option to `mpicc` (or its friends) so that the compiler will have included symbol tables in the compiled object code.

To run your MPI program with a debugger use the `-debug` or `-debug-no-pause` and `-debugger` options to `mpirun`. See the `man` pages to `pathdb`, `gdb`, and `strace` for details. When you run under a debugger, you get an xterm window on the front end machine for each node process. Thus, you can control the different node processes as desired.

To use `strace` with your MPI program, the syntax would be

```
mpirun -np n -m hostsfile strace program-name
```

The following features of PathScale MPI especially facilitate debugging:

- Stack backtraces are provided for programs that crash.

- `-debug` and `-debug-no-pause` options are provided for `mpirun` that can make each node program start with debugging enabled. The `-debug` option allows you to set breakpoints, and start running programs individually. The `-debug-no-pause` option allows postmortem inspection.

- Communication between mpirun and node programs can be printed by specifying the `mpirun -verbose` option. MPI implementation debug messages can be printed by specifying the `mpirun -psc-debug-level` option (this can substantially impact the performance of the node program).

- Support is provided for progress timeout specifications, deadlock detection, and generating information about where a program is stuck.

- Several misconfigurations (such as mixed use of 32-bit/64-bit executables) are detected by the runtime.

2.8 PathScale MPI limitations

The current version of PathScale MPI has the following limitations:

- At most eight node programs per node.
  The error message when this limit is exceeded is
  "All InfiniPath ports in use".
- At most 2048 nodes.
- At most 8192 node programs per MPI job.
- No C++ bindings to MPI -- use the extern C MPI function calls.
- In MPI-IO file I/O calls in the Fortran binding, offset or displacement arguments are limited to 32 bits. Thus, for example, the second argument of `MPI_File_seek` must lie between -2^{31} and $2^{31}-1$, and the argument to `MPI_File_read_at` must lie between 0 and $2^{32}-1$.

.

CHAPTER 3

InfiniPath Cluster Administration

This chapter describes what the cluster administrator needs to know about the InfiniPath software and system administration.

3.1 Introduction

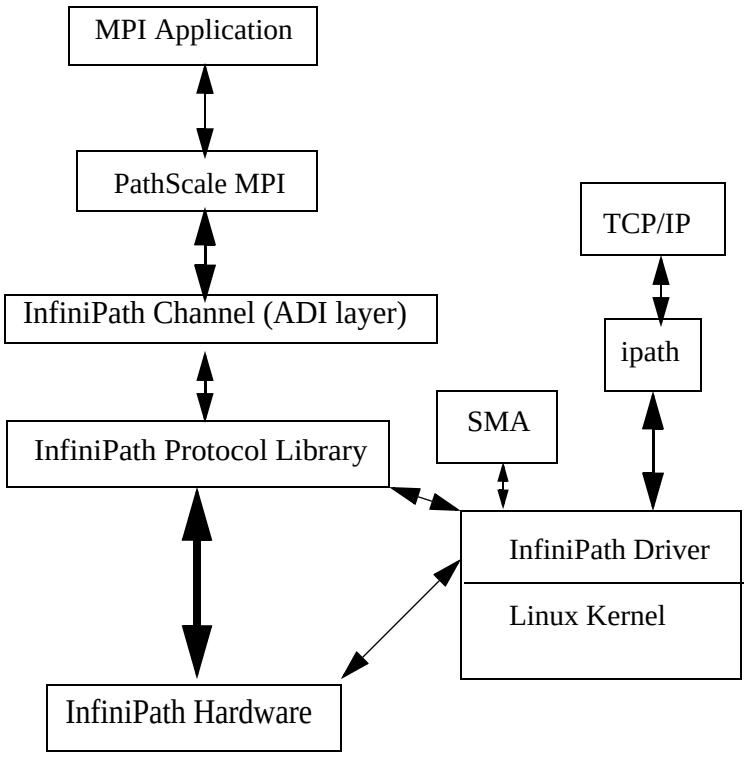
The following components of the InfiniPath software provide the foundation supporting the MPI implementation:

- InfiniPath Driver `infinipath`
- Subnet Management Agent `ipath_sma`
- Layered Ethernet driver `ipath`

• Protocol and MPI support libraries

Figure 1 indicate the relations of these software elements.

Figure 1:



3.2 Installed layout

The InfiniPath software is supplied as a set of RPM files, described in detail in the *Installation Guide*. This section describes the directory structure that the installation leaves on each node's file system.

The InfiniPath shared libraries are installed in `/usr/lib` for 32-bit applications, and `/usr/lib64` for 64-bit applications. See the Installation Guide for more details.

MPI include files are in `/usr/include`

MPI programming examples from the MPICH distribution are in `/usr/share/mpich/examples`

InfiniPath utility programs, as well as MPI utilities and benchmarks are installed in `/usr/bin`.

The InfiniPath kernel modules are installed in the standard module locations in `/lib/modules` (version dependent). They are compiled and installed when the `infinipath` RPM is installed. They must be rebuilt and re-installed when the kernel is upgraded. This can be done by running the script `/usr/src/pathscale/drivers/make-install.sh`

3.3 Configuration and startup

The InfiniPath driver software is generally started at system startup under control of the scripts
`/etc/rc.d/init.d/infinipath` and `/etc/sysconfig/infinipath`.

These scripts are configured by the installation. The cluster administrator does not normally need to be concerned with the configuration parameters.

After startup, the InfiniPath processes that are normally running in each node are:

`ipath_mux` (two instances)
`ipath_sma`

`ipath_mux` is the InfiniPath Management Packet Multiplexer. It allows multiple management applications and diagnostic programs to send and receive InfiniBand management packets.

You can check the version of the installed InfiniPath software by looking in
`/proc/driver/infinipath/version`.

Example contents are:

```
Driver 2.0, InfiniPathHT4X_Link1, InfiniPath1 3.2, PCI 2,
SW Compat 2
```

This information is useful when for reporting problems when requesting support.

3.3.1 InfiniPath software configuration

The driver has several configuration variables which provide for setting reserved buffers for the software, defining events to create trace records, and setting debug level. See the `infinipath` man page for details.

3.3.2 How to start and stop the InfiniPath software outside of system startup

The InfiniPath driver software runs as a system service, normally started at system startup. You can start, stop, or restart the InfiniPath support with

```
/etc/rc.d/init.d/infinipath [start | stop | restart]
```

Normally you will not need to restart the software, but you may wish to do so after installing a new InfiniPath release, or after changing driver or SMA options.

If you stop or restart the InfiniPath software after system boot, and are using the `ipath` interface, it is a good idea to use the `ifdown` and `ifup` commands before and after restart, because the restart will also stop the `ipath` driver.

.

3.4 Subnet Management Agent

Each node in an InfiniPath cluster runs a Subnet Management Agent (SMA), which carries out two-way communication with the Subnet Manager (SM) running on one or more managed switches. The Subnet Manager discovers and configures all the reachable nodes in the InfiniBand fabric. It discovers them at switch startup, and continues monitoring changes in the physical network connectivity and topology. It is responsible for assigning local identifiers, called *LID*s, to the visible nodes. In case the network contains multiple managed switches, they negotiate among themselves which will run the unique subnet manager.

The Subnet Manager also assigns and manages InfiniBand multicast groups, such as the group used for broadcast purposes by the `ipath` driver.

The primary functions of the SMA are to keep the SM informed whether this node is alive and to get its assigned identifier from the SM. The SM tells the other switches in the fabric how to set up their internal tables that map node identifiers to ports.

Appendix B gives some SMA `syslog` messages that can be of help in trouble shooting.

Check the file `/proc/driver/infinipath/status` to verify that the InfiniPath software is loaded and functioning. Normally, it should contain node's LID, MLID, GUID, and serial number plus the following items,

```
Initted
InfiniPath_found
IB_link_up
IB_configured
```

plus, if `ipath` is in use, the following items

```
ipath_loaded
ipath_up
```

In case of trouble the entry `Fatal_Hardware_Error` might appear. In this case, try rebooting the node.

3.5 Layered Ethernet driver

The layered Ethernet component provides almost complete Ethernet software functionality over the InfiniPath fabric. The driver module name is `ipath`. At startup this is bound to some Ethernet device `eth`*x*. All Ethernet functions are available through this device in a transparent way, except that Ethernet multicasting is not supported. You can use all the usual command line and GUI-based configuration tools on this Ethernet.

3.5.1 ipath configuration on Fedora

The Ethernet driver is loaded at startup by a line in

```
/etc/modprobe.conf
```

such as

```
alias eth2 ipath
```

and by corresponding lines in

```
/etc/sysconfig/network-scripts/ifcfg-eth2
      # PathScale Interconnect Ethernet
      DEVICE=eth2
      ONBOOT=yes
      BOOTPROTO=dhcp
```

You can check whether the Ethernet driver has been loaded with

```
lsmod | grep ipath
```

3.5.2 ipath configuration on SuSE

1. Check to see that the ipath module is loaded:

```
lsmod | grep -w ipath
```

If it is not listed, load it with the command:

```
modprobe ipath
```

2. Determine the MAC address to be used with the following command:

```
sed -n -e 's/.*GUID=[0-9a-f]*:[0-9a-f]*//' \
-e 's/:\(.\):/:0\1:/g' \
-e 's/://' -e 's/,.*//p' /proc/driver/infinipath/status
```

The output should appear similar to this (6 hex digit pairs, separated by colons):

```
00:11:75:04:e0:11
```

This is referred to as $MAC in the rest of the instructions, where $MAC must be replaced by the actual MAC address output by the `sed` command.

2. Create the file

```
/etc/sysconfig/hardware/hwcfg-eth-id-$MAC
```

with the following lines:

```
MODULE=ipath
STARTMODE=auto
```

3. Create the file

```
/etc/sysconfig/network/ifcfg-eth-eth#
```

where # is the number of the next Ethernet interface. If `ifconfig -a` showed `eth0` and `eth1`, then # would typically be `2` and the file name would be: `/etc/sysconfig/network/ifcfg-eth-eth2`. It should contain the following lines if you are using DHCP:

```
BOOTPROTO=dhcp
NAME=ipath
STARTMODE=auto
_nm_name=eth-id-$MAC
```

If you are using static IP addresses, then it will instead appear similar to the lines below, with the actual network and IP address that you are using (this example uses a private IP address of 192.168.1.2, with the normal matching netmask and broadcast addresses):

```
BOOTPROTO=static
IPADDR=192.168.1.2
NETMASK=255.255.255.0
NETWORK=192.168.1.0
NAME=ipath
STARTMODE=auto
_nm_name=eth-id-$MAC
```

It is important that the name in `_nm_name` exactly match the name of the file created in step 2, with `hwcfg-` prepended.

4. When the system is next rebooted, the `ipath` Ethernet device should be correctly configured. Verify this with the command:

```
ifconfig -a
```

If you do not see a device listed with a HWaddr matching $MAC, check for errors in steps 1, 2, or 3. If it is listed, but does not have a valid "inet addr" listed, check for errors with the IP address or DHCP server configuration.

3.6 Configuring secure shell

Running MPI programs on a PathScale cluster depends, by default, on secure shell `ssh` to launch node programs on the nodes. As a practical matter, jobs must be able to start up without the need for interactive password entry on every node. In Section 2.4.1, we showed how each user can accomplish this end through the use of `ssh-agent`. Here we see how the cluster administrator can lift this burden from the user through the use of the `shosts.equiv` mechanism. This method is recommended, provided that your cluster is behind a firewall and accessible only to trusted users.

This example assumes that both the cluster nodes and the front end system are running the `openssh` package as distributed in current Linux systems. It also assumes that all cluster users have accounts with the same account name on both the front end, and on each node, either by using NIS, or some other means of distributing the password file.

This example assumes the front end is called `ip-fe.pathscale.com`. All steps to configure `ssh` require root or superuser access on `ip-fe`, and on each node. This example also assumes that `ssh`, including the hosts key, has already been configured on the system `ip-fe`. See the `sshd` and `ssh-keygen` man pages for more information.

On the system `ip-fe`, change `/etc/ssh/ssh_config` to allow host-based authentication. Specifically, this file must contain the following four lines, set to 'yes'. If they arepresent and commented out with an initial #, remove the #. If not present, them add them.

```
RhostsAuthentication yes
RhostsRSAAuthentication yes
HostbasedAuthentication yes
EnableSSHKeysign yes
```

On each of the InfiniPath node systems, add two files in the `/etc/ssh` directory, if they are not already there. If there, you will need to edit them. Both files need to be changed to mode 600 when you have finished editing them.

Create or edit the file `/etc/ssh/shosts.equiv`, adding the name of the front end system. You'll need to add the line:

```
ip-fe.pathscale.com
```

Create or edit the file `/etc/ssh/ssh_known_hosts`. You'll need to copy the contents of the file `/etc/ssh/ssh_host_dsa_key.pub` from `ip-fe.pathscale.com` to this file (as a single line), and then edit that line to insert `ip-fe.pathscale.com ssh-dss` at the beginning of the line. This is very similar to the standard `known_hosts` file for `ssh`. An example line might look like this (displayed as multiple lines, but a single line in the file):

```
ip-fe.pathscale.com  ssh-dss
AAzAB3NzaC1kc3MAAACBAPoyES6+Akk+z3RfCkEHCkmYuYzqL2+1nwo4LeTVWpCD1Q
svrYRmpsfwpzYLXiSJdZSA8hfePWmMfrkvAAk4ueN8L3ZT4QfCTwqvHVvSctpibf8n
aUmz
loovBndOX9TIHyP/Lj/fzzep4wL17+5hr1AHXldzrmgeEKp6ect1wxAAAAFQDR56d
AKFA4WgAiRmUJailtLFp8swAAAIBB1yrhF5P0jO+vpSnZrvrHa0Ok+Y9apeJp3ses
see30NlqKbJqWj5/DOoRejr2VfTxZROf8LKuOY8tD6I59I0vlcQ812E5iw1GCZfNe
fBmWbegWVKFwGlNbqBnZK7kDRLSOKQtuhYbGPcrVlSjuVpsfWEju64FTqKEetA8l
8QEgAAAIBNtPDDwdmXRvDyc0gvAm6lPOIsRLmgmdgKXTGOZUZ0zwxSL7GP1nEyFk9
wAxCrXv3xPKxQaezQKs+KL95FouJvJ4qrSxxHdd1NYNR0DavEBVQgCaspgWvWQ8cL
0aUQmTbggLrtD9zETVU5PCgRlQL6I3Y5sCCHuO7/UvTH9nneCg==
```

On each node, system file `/etc/ssh/sshd_config` must be edited, so that the following four lines are not commented out (no # at the start of the line) and are set to `yes`. Each of these lines is normally present, but commented out and set to `no` by default.

```
RhostsAuthentication yes
RhostsRSAAuthentication yes
HostbasedAuthentication yes
PAMAuthenticationViaKbdInt yes
```

After creating or editing these three files, `sshd` must be restarted on each system. If you are already logged in via `ssh` (or any other user is logged in via `ssh`), their sessions or programs will be terminated, so do this only on idle nodes. Tell `sshd` to use the new configuration files by typing:

```
killall -HUP sshd
```

At this point, any user should be able to login to the `ip-fe` front end system, and then use `ssh` to login to any InfiniPath node without being prompted for a password, or pass phrase.

3.7 Performance and management tips

3.7.1 Remove unneeded services.

An important step that the cluster administrator can take to enhance application performance is to minimize the set of system services running on the compute nodes. Since these are presumed to be specialized computing appliances, they do not need many of the service daemons normally running on a general Linux computer.

Following are several groups constituting a minimal necessary set of services. These are all services controlled by `chkconfig`. To see the list of services that are enabled, use the command

```
/sbin/chkconfig --list | grep -w on
```

Basic network services:

```
network
ntpd
syslog
xinetd
sshd
```

For system housekeeping:

```
anacron
atd
crond
```

If you are using NFS or yp passwords:

```
rpcidmapd
ypbind
portmap
nfs
nfslock
autofs
```

To watch for disk problems:

```
smartd
readahead
```

The service comprising the InfiniPath driver and SMA:

```
infinipath
```

And any services required by your batch queuing system.

3.7.2 Balanced processor power.

Of course, higher processor speed is good. Adding more processors is good only if processor speed is balanced. Adding slower processors can result in load imbalance.

3.7.3 Homogenous nodes

To minimize management problems, the compute nodes of the cluster should have very similar hardware configurations and identical software installations. The tool `ipathbug-helper`, described in Appendix B, is useful for verifying homogeneity.

3.8 Integration with a batch queuing system

Most cluster systems use some kind of batch queuing system as an orderly way to provide users with access to the resources they need to meet their job's performance requirements. In the context of a batch queuing system, the cluster administrator needs to provide means for users to submit MPI jobs through it. This can take the form of a script, call it `batch_mpirun`, in your favorite scripting language, which your users can invoke much as they would invoke `mpirun`, to submit their MPI jobs. This script would in turn make the appropriate calls to the batch system and to the shell. In this section we sketch an example of the some of the functions that such a script might perform, in the context of the Simple Linux Utility Resource Manager (SLURM) developed at Lawrence Livermore National Laboratory. We will describe these functions in terms of commands in the `bash` shell.

3.8.1 Allocating resources

As described in the preceding chapter, when the `mpirun` command starts, it requires specification of the number of node programs it must spawn (via the `-np` option) and specification of an MPI hosts file listing the nodes on which it may run them. Normally, since performance is usually important, a user might require that his node program be the only application running on each node CPU. In a typical batch environment, the MPI user would still specify the number of node programs, but would depend on the batch system to allocate specific nodes when the required number of CPUs becomes available. Thus, `batch_mpirun` would take at least an argument specifying the number of node programs and an argument specifying the MPI program to be instantiated. For example,

```
batch_mpirun -np n my_mpi_program
```

After parsing the command line arguments, the next step of `batch_mpirun` would be to request an allocation of *n* processors from the batch system. In SLURM, this would use the command

```
srun --allocate --ntasks=$np
```

where `np` is the shell variable that your script has set from the parsing of its command line options.

With these specified arguments, the SLURM function `srun` blocks until there are `$np` processors available to commit to the caller. When the requested resources are available, this command opens a new shell and allocates the requested number of processors to it.

3.8.2 Generating a node file

Once the batch system has allocated the required resources, your script must generate an MPI hosts file. To do this, it must find out which nodes the batch system has allocated, and how many processes we can start on each node. Here is how we can do this in `bash`, using calls to SLURM functions:

```
node_file=`mktemp -p /tmp node_file.XXXXXX`
srun hostname -s | sort | uniq -c \
| awk '{printf "%s:%s\n", $2, $1}' > $node_file
echo $node_file
```

The first command creates a temporary hosts file with a random name, and assigns the name to the variable `node_file` it has generated.

The next instance of the SLURM `srun` command runs `hostname -s` once per process slot that SLURM has allocated to us. If SLURM has allocated two slots on one node, we thus get the output of `hostname -s` twice for that node.

The `sort | uniq -c` component tells us the number of times each unique line was printed. The `awk` command converts the result into the node file format used by `mpirun`. Each line consists of a node name, a colon, and the number of processes to start on that node. These lines are redirected to the file that was created by the first command.

3.8.3 Simple process management

At this point, your script has enough information to be able to run an MPI program. All that remains is to start the program when the batch system tells us that we can do so, and notify the batch system when the job completes.

```
mpirun -np $np -m $node_file $my_mpi_program
exit_code=$?
rm -f $node_file
exit $exit_code
```

The InfiniPath software will normally ensure clean termination of all MPI programs when a job ends, but in some rare circumstances an MPI process will remain alive, and potentially interfere with future MPI jobs. To avoid this problem, the usual solution is to run a script before and after each batch job which kills all unwanted processes. PathScale does not provide such a script, but it is useful to know how to find out which processes on a node are using the InfiniPath interconnect.. The easiest way to do this is through use of the "fuser" command, which is normally installed in `/sbin`:

```
/sbin/fuser /dev/ipath
/dev/ipath:     22648m 22651m
```

In this example, process 22648 and 22651 are using the InfiniPath interconnect..

You can use

```
/sbin/fuser -k /dev/ipath
```

to terminate all processes using the InfiniPath interconnect..

3.9 Customer acceptance utility

`ipath_checkout` is a `bash` script to verify that the installation is correct and that all the nodes of the network are functioning and mutually connected by the InfiniPath fabric. It is to be run on a frontend node, and requires specification of a hosts file:

```
ipath_checkout [options] hostsfile
```

where *`hostsfile`* designates a file listing the hostnames of the nodes of the cluster, one hostname per line, just as the hosts file for the `mpirun` command.

`ipath_checkout` performs the following seven tests on the cluster:

1. `ping` all nodes to verify all are reachable from the frontend.
2. `ssh` to each node to verify correct configuration of ssh.
3. Gather and analyze system configuration from nodes.
4. Gather and analyze RPMs installed on nodes
5. Verify InfiniPath hardware and software status and configuration.
6. Verify ability to run MPI jobs on nodes by use of `mpirun`.
7. Run bandwidth and latency test on every pair of nodes and analyze results.

The possible options to `ipath_checkout` are:

`-h, --help`
Display help message giving defined usage.

`-v, --verbose`
`-vv, --vverbose`
`-vvv, --vvverbose`
These specify successively higher levels of detail in reporting results of tests. So, there are four levels of detail in all, including the case of where none of these options is present..

`-c, --continue`
When not specified, the test terminates when any test fails. When specified, the tests continue after a failure, with failing nodes excluded from subsequent tests.

`--workdir=DIR`
Use DIR to hold intermediate files created while running tests. DIR must not already exist.

`-k, --keep`
Keep intermediate files that were created while performing tests and compiling reports. Results will be saved in a directory named pathscale_* or the directory name given to `--workdir`.

`--skip=`*`LIST`*
Skip the tests in *`LIST`* (e.g. --skip=2457 will skip tests 2, 4, 5, and 7.)

In most cases of failure, the script suggests recommended actions. Please see the `ipath_checkout` man page for further information and updates..

Appendices

Appendix A. Benchmark Programs

Several MPI performance measurement programs are installed from the `mpi-benchmark` RPM. This appendix describes these useful benchmarks and how to run them. These programs are based on code from the group of Dr. Dhabaleswar K. Panda at the Network-Based Computing Laboratory at the Ohio State University. See http://nowlab.cis.ohio-state.edu/

These programs allow you to measure the MPI latency and bandwidth between two or more nodes in your cluster. Both the executables, and the source for those executables, are shipped. The executables are shipped in the `mpi-benchmark` RPM, and installed under `/usr/bin`. The source is shipped in the `mpi-devel` RPM and installed under `/usr/share/mpich/examples/performance`.

The examples given below are intended only to show the syntax for invoking these programs and the meaning of the output. They are NOT representations of actual InfiniPath performance characteristics.

Benchmark 1: Measuring MPI latency between two nodes

In the MPI community, *latency for a message of given size* is defined to be the time difference between a node program's calling MPI_Send and the time that the corresponding MPI_Recv in the receiving node program returns. By *latency*, alone without a qualifying message size, we mean the latency for a message of size zero. This latency represents the minimum overhead for sending messages, due both to software overhead and to delays in the electronics of the fabric. To simplify the timing measurement, latencies are usually measured with a ping-pong method, timing a round-trip and dividing by two.

The program `osu_latency`, from Ohio State University, measures the latency for a range of messages sizes from 0 to 4 megabytes. It uses a ping-pong method, in which the rank 0 process initiates a series of sends and the rank 1 process echos them back, using the blocking MPI send and receive calls for all operations. Half the time interval observed by the rank 0 process for each such exchange is a measure of the latency for messages of that size, as defined above. The program uses a loop, executing many such exchanges for each message size, in order to get an average. It defers the timing until the message has been sent and received a number of times, in order to be sure that all the caches in the pipeline have been filled.

This benchmark always involves just two node programs. You can run it with the command

```
mpirun -np 2 -ppn 1 -m hostsfile osu_latency
```

The `-ppn 1` option is needed to be certain that the two communicating processes are on different nodes. Otherwise, in the case of multiprocessor nodes, `mpirun` might assign the two processes to the same node, and so the result would not be indicative of the latency of the InfiniPath fabric, but rather of the shared memory transport mechanism. Here is what the output of the program looks like:

```
# OSU MPI Latency Test (Version 2.0)
# Size          Latency (us)
0               1.47
1               1.48
2               1.49
4               1.48
8               1.48
```

```
16              1.72
32              1.75
64              1.79
128             1.90
256             2.18
512             2.65
1024            3.62
2048            5.58
4096            7.69
8192            12.1
16384           22.43
32768           45.16
65536           96.69
131072          165.61
262144          311.65
524288          592.81
1048576         1147.42
2097152         2246.39
4194304         4444.51
```

The first column gives the message size in bytes, the second gives the average (one-way) latency in microseconds. Again, this example is given to show the syntax of the command and the format of the output, and is not meant to represent actual values that might be obtained on any particular InfiniPath installation. The numbers shown are from an actual run using 2.0 GHz Opteron processors.

Benchmark 2: Measuring MPI bandwidth between two nodes

The `osu_bw` benchmark is meant to measure the maximum rate at which you can pump data between two nodes. It also uses a ping-pong mechanism, similar to the `osu_latency` code, except in this case, the originator of the messages pumps a number of them (64 in the installed version) in succession using the non-blocking MPI_Isend function, while the receiving node consumes them as quickly as it can using the non-blocking MPI_Irecv, and then returns a zero-length acknowledgement when all of the set has been received.

You can run this program with

```
mpirun -np 2 -ppn 1 -m hostsfile osu_bw
```

Typical output might look like

```
# OSU MPI Bandwidth Test (Version 2.0)
# Size          Bandwidth (MB/s)
1               2.250465
2               4.475789
4               8.979276
8               17.952547
16              27.615041
32              52.676363
64              104.704225
128             198.347505
256             335.396929
512             521.273433
1024            829.369420
2048            884.249845
4096            926.723948
8192            934.093084
16384           941.191459
32768           938.179872
65536           945.163478
131072          950.206048
262144          951.938802
524288          952.912385
1048576         953.716825
2097152         953.922714
4194304         954.119999
```

Of course, the increase in measured bandwidth with messages size results from the fact that latency's contribution to the measured time interval becomes relatively smaller.

Benchmark 3: Measuring mpi latency in host rings

The program `mpi_latency` can be used to measure latency in a ring of hosts. Its syntax is a bit different from Benchmark 1 in that it takes command line arguments that let you specify the message size and the number of messages over which to average the results. So, for example, if you have a hosts file listing four or more nodes, the command

```
mpirun -np 4 -ppn 1 -m hostsfile mpi_latency 100 0
```

might produce output like this:

```
0       1.760125
```

indicating it took an average of 1.76 microseconds per hop for sending a zero-length message from the first host, to the second, to the third, to the fourth, and then get replies back in the other direction.

Appendix B. Troubleshooting

This appendix describes some of the existing provisions for diagnosing problems and fixing them.

1. Mechanical and Electrical Considerations

If the InfiniPath interconnect boards are all installed correctly and solidly connected to an appropriate InfiniBand switch, then, on powering up, each board should show both LEDs lit.

If a node repeatedly and spontaneously reboots when attempting to load the InfiniPath driver, it may be a symptom that its InfiniPath interconnect board is not well seated in the HTX slot.

2. Driver loading and initialization

Initiation message

Symptom: When I start the program, it fails with messages similar to:

```
userinit: userinit ioctl failed: Network is down [1]: device
init failed
userinit: userinit ioctl failed: Fatal Error in
keypriv.c(520): device init failed
```

Suggestion: One or more nodes do not have the Interconnect in a usable state. A cable is not connected, the switch is down, SMA is not running, or a hardware error has occurred.

You can check the file `/proc/driver/infinipath/status` to verify that the Infinipath software is loaded and functioning Normally, it should contain node's LID, MLID, GUID, and serial number plus the following items,

```
Initted
InfiniPath_found
IB_link_up
IB_configured
```

plus, if `ipath` is in use, the following items

```
ipath_loaded
ipath_up
```

In case of trouble the entry `Fatal_Hardware_Error` might appear.

If, on any node, the driver status appears abnormal, you can try restarting it with

```
/etc/rc.d/init.d/infinipath restart
```

or

```
/etc/rc.d/init.d/infinipath stop
```

```
/etc/rc.d/init.d/infinipath start
```

3. Broken intermediate link

Symptom: Some message traffic passes through the fabric while other traffic appears to be blocked. MPI jobs fail to run.

Explanation: In large cluster configurations, switches may be attached to other switches in order to supply the necessary inter-node connectivity. Problems with these inter-switch (or intermediate) links are sometime more difficult to diagnose than failure of the final link between a switch and a node. The failure of an intermediate link may allow some traffic to pass through the fabric while other traffic is blocked or degraded.

Suggestion: If you encounter such behavior in a multi-layer fabric, check that all switch cable connections are correct.

4. SMA syslog messages

Most of these are written once at SMA startup, or once for each unit. A few are printed at state changes after startup.

`Failed to take IB link to DOWN state: errno`

Not fatal, but likely a sign of other troubles. Printed when the SMA is bouncing the link in order to get the attention of the Subnet Manager and the link won't go down.

The following messages are normally written just once when `infinipath` is started..

`unit u devstatus = hex_str desc_str`

The `hex_str` holds a status code, encoding a set of binary status variables. The *`desc_str`* is a concatenation of descriptive strings from the following list, each indicating the corresponding bit being on:

```
IB_NOCABLE
IB_CONF
```

```
IB_READY
CHIP_PRESENT
SMA
PLACEHOLDER
LAYERUP
LAYERLOADED
INITTED
```

or, if the determination of status failed:

`unable to get status for unit u`

That occurrence is not fatal, but indicates something anomalous.

`connected to mux, id=`*`mux_id`*

Indicates normal operation.

`looking for` *`n`* `units`

Indicates normal operation and shows the maximum number of units supported by the installed driver and SMA.

`unit` *`u`* `found`
or
`unit` *`u`* `not found`

Written once at startup for each unit. “Found” is normal. “Not found” is not fatal, but usually means that not all possible units are configured..

`expected 1 port on unit` *`u`*`, got` *`n`*

A unit claimed to support more than one port. Not fatal, but it almost certainly means a driver problem or library mismatch.

`setting unit` *`u`* `LID to` *`hx`*`(`*`n`*`) from command line`

Only printed if someone sets the LID on the command line.

`setting unit` *`u`* `GUID to` *`xx`*`:`*`xx`*`:`*`xx`*`:`*`xx`*`:`*`xx`*`:`*`xx`*`:`*`xx`*`:`*`xx`* `from command line`

Only printed if someone set the GUID on the command line.

`calloc() failure in mcast_register()`

Printed if someone (likely an MPI job) tries to set up a multicast group and there isn't enough memory to `malloc` the struct, which is tiny. Not fatal, but it means the system is in bad trouble.

`unexpected state transition` *`s1`*`-->`*`s2`* `in mcast_mp_state()`
`unexpected state transition` *`s1`*`-->`*`s2`* `in mcast_state()`
Not fatal but they shouldn't happen. They might indicate a problems with a particular SM implementation in setting up a multicast group or handling link state transitions.

`no subnet management activity detected on unit` *`u`*`, bouncing link`

At startup, if we haven't made sufficient progress in setting up our link, we start taking the link up and down periodically to try to get the attention of the SM. This message may indicate that the SM is down, that the fabric is partitioned and the SM is unreachable, or that there is a compatibility problem with a particular implementation of an SM.

`SM does not support Multicast`

Self explanatory. The SM managing the fabric does not support multicast groups. This implies that neither the layered Ethernet driver nor MPI will work. Otherwise, not fatal.

`SM set MLID to 0x`*`xxxx`*

Printed once per multicast group creation. One multicast group is created at startup for `ipath`, so there is always one.

or

`Multicast join failed, status = 0x`*`xxxx`*

The multicast join request was refused by the SM for some reason.

`SM set unit` *`u`* `LID to 0x`*`xxxx`*`(`*`n`*`)`
`SM set unit %d link state to 0x`*`xxxx`*

Printed whenever the SM wants this node to change its link state. Normally, there may be a few such transitions at startup, and then whenever there is an unplug/replug event.

`SM set unit` *`u`* `MTU to 0x`*`xxxx`*

Printed whenever the SM assigns this node an MTU. Normally only once, at startup.

The following seven are all non fatal diagnostics indicating something abnormal.

`handle_sigchld: waitpid() returned error, err_no`
`handle_sigchld: waitpid() returned 0`
`handle_sigchld: waitpid() returned unexpected pid` *`pid`*

```
clock_sanity: fstat(\"tmpname\") failed -- err_no
clock_sanity: mkstemp(\"tmplat\") failed -- err_no
clock_sanity: gmtime(long) failed -- err_no
clock_sanity: year yyyy is likely bogus -- check hwclock
```

The next is normally written at shutdown

```
child process exited, exiting
```

The following may be written by `ipath_sma` or by `ipath_dumpmads`. They are all diagnostics indicating non-fatal problems opening, writing, or rotating the log file.

```
backup log file ipath_mux_logfile is not a regular file"
unable to rename log file ipath_mux_logfile, deleting
unable to delete log file ipath_mux_logfile
unable to open log file ipath_mux_logfile
unable to stat() log file ipath_mux_logfile
unable to open log file ipath_mux_logfile
unable to stat() log file ipath_mux_logfile
unable to open log file ipath_mux_logfile
```

Here is an example of the syslog record for a normal startup.

```
May 16 20:45:26 iqa-16 /usr/bin/ipath_sma[5301]: connected to mux,
id=4
May 16 20:45:26 iqa-16 /usr/bin/ipath_sma[5301]: looking for 1
units
May 16 20:45:26 iqa-16 /usr/bin/ipath_sma[5301]: unit 0 found
May 16 20:45:26 iqa-16 /usr/bin/ipath_sma[5301]: unit 0 devstatus
= 0x61 IB_READY CHIP_PRESENT INITTED
May 16 20:45:26 iqa-16 /usr/bin/ipath_sma[5301]: SM set unit 0 LID
to 0x12(18)
May 16 20:45:26 iqa-16 /usr/bin/ipath_sma[5301]: SM set unit 0 MTU
to 0x4
May 16 20:45:26 iqa-16 /usr/bin/ipath_sma[5301]: SM set unit 0
link state to 0x4
May 16 20:45:36 iqa-16 /usr/bin/ipath_sma[5301]: SM set MLID to
0xc010
```

5. *Messages from the InfiniPath library.*

These messages may appear in the `mpirun` output. The first set are error messages, which indicate internal problems and should be reported to PathScale.

```
Trying to cancel invalid timer (EOC)
sender rank rank is out of range (notification)
sender rank rank  is out of range (ack)
Reached TIMER_TYPE_EOC while processing timers
Found unknown timer type type
unknown frame type type
recv done: available_tids now n, but max is m (freed p)
cancel recv available_tids now n, but max is m (freed %p)
[n] Src lid error: sender: x, exp send: y
Frame receive from unknown sender. exp. sender = x, came from y
Failed to allocate memory for eager buffer addresses: str
```

The following error messages probably indicate a hardware or connectivity problem-

```
Failed to get IB Unit LID for any unit
Failed to get our IB LID
Failed to get number of Infinipath units
```

In these cases you can try to reboot, then call PathScale support.

The following indicate a mismatch between the InfiniPath interconnect hardware in use and the version for which the software was compiled.

```
Number of buffer avail registers is wrong; have n, expected m
build mismatch, tidmap has n bits, ts_map m
```

These indicate a mismatch between the InfiniPath software and hardward versions. Consult PathScale support.

The following are all informative messages. They are not necessarily fatal themselves, but sometimes indicate problems interfere with the application. All of them are prefixed with the name of the function that produced them.

The following indicate driver initialization problems, and should never occur. Please inform PathScale if they do.

```
Failed to get LID for unit u: str
Failed to get number of units: str
GETPORT ioctl failed: str
can't allocate memory for ipath_ctrl_typ: type
can't stat infinipath device to determine type: type
file descriptor is not for a real device, failing
get info ioctl failed: str
ipath_get_num_units called before init
ipath_get_unit_lid called before init
mmap64 of egr bufs from h failed: str
mmap64 of pio buffers at %llx failed: str
mmap64 of pioavail registers (%llx) failed: str
mmap64 of rcvhdr q failed: str
mmap64 of user registers at %llx failed: str
userinit allocation of rcvtail memory failed: str
userinit ioctl failed: str
Failed to set close on exec for device: str
```

The following message indicates that a node program may not be processing incoming packets, perhaps due to a very high system load.

```
eager array full after overflow, flushing (head h, tail t
```

The following indicates an invalid InfiniPath link protocol version

```
InfiniPath version ERROR: Expected version v, found w (memkey h)
```

The following error messages should occur rarely or never and indicate internal software problems.

```
ExpSend opcode h tid=j, rhf_error k: str
Asked to set timeout w/delay l, gives time in past (t2 < t1)
Error in sending packet: str
Fatal error in sending packet, exiting: str
Fatal error in sending packet: str
```

Here the *str* can give additional clues to the reason for the failure.

The following probably indicates a node failure or malfunctioning link in the fabric:

`Logic Link to` *str*`, rank` *r* `could not be established. Time elapsed` *hr*`:`*mn*`:`*sc*`. Still trying`

6. MPI messages

Some MPI error messages are issued from the parts of the code inherited from the MPICH implementation. See the MPICH documentation for descriptions of these. This section presents the error messages specific to the PathScale implementation of MPI.

These messages appear in the `mpirun` output. Most are followed by an abort, and possibly a backtrace. Each is preceded by the name of the function in which the exception occurred.

`Error sending packet:` *description*

`Error receiving packet :` *description*

A fatal protocol error occurred while trying to send an InfiniPath packet.

`On Node` *n*`, process` *p* `seems to have forked.`
`The new process id is` *q*`. Forking is illegal under`
`InfiniPath. Exiting.`

An MPI process has forked and its child process has attempted to make MPI calls. This is not allowed.

processlabel `Fatal Error in` *filename line_no*`:` *error_string*

This is always followed by an abort. The *processlabel* usually takes the form of host name followed by process rank. At time of writing, the possible *error_string*s are

```
Illegal label format character.
Recv Error.
Memory allocation failed.
```

```
Error creating shared memory object.
Error setting size of shared memory object.
Error mmapping shared memory.
Error opening shared memory object.
Error attaching to shared memory.
invalid remaining buffers !!
Node table has inconsistent length!
Timeout waiting for nodetab!
```

`Internal Error: NULL function/argument found:`*`func_ptr`*`(`*`arg_ptr`*`)`

This should never occur. Please inform PathScale if it occurs.

7. MPI stats

Using the `-print-stats` option to `mpirun` will result in a listing to `stderr` of various MPI statistics from each node program. Using the option

```
-statsfile filename
```

will send the output of each node program to the file *`filename-rank`* rather than to `stderr`, where *`rank`* is the node program's MPI rank.

The occurence of certain message types in the output may indicate problems with the system or with the MPI program itself.

For example:,

`retransmit` indicates dropped packets, which may be a symptom of errors or resource exhaustion in the interconnect or switch.likely, or of the receiving processor not being able to process incoming data quickly enough.

`piobuf_unavail` indicates that send-side resources on the chip are not available. This happens in a send-side flood.

`recv_dropped` indicates too many unexpected messages. This means that MPI has not posted enough receives to match incoming messages.

`hdrq_full` and `etid_full`: indicate that MPI is not processing messages fast enough, that is, the user-application is not giving control to MPI (via MPI functions) often enough.

8. `ipathbug-helper`

The InfiniPath software includes a shell script `ipathbug-helper`, which can gather status and history information for use in analyzing InfiniPath problems. When seeking assistance from PathScale technical support, you should run this script on the head node of your cluster and perhaps on some of the compute nodes which are suspected to have problems, and send its `stdout` output to your reseller.

It is best to run `ipathbug-helper` with root privilege, since some of the queries it makes requires it. There is also a `--verbose` which greatly increases the amount of gathered information.

`ipathbug-helper` is useful for checking for the desired software homogeneity among the nodes of your cluster, mentioned in Section3.7.3. Simply run it on several modes and examine the output for differences.

Appendix C. Recommended Reading

References for MPI

The MPI Standard specification documents are at
http://www.mpi-forum.org/docs

Good books for learning MPI programming:

Gropp, Lusk and Skjellum, *Using MPI*, Second Edition, 1999, MIT Press, ISBN 0-262-57134-X

Gropp, Lusk and Skjellum, *Using MPI-2*, Second Edition, 1999, MIT Press, ISBN 0-262-57133-1

Pacheco, *Parallel Programming with MPI*, 1997, Morgran Kaufman Publishers, ISBN 1-55860

The MPICH implementation of MPI and its documentation are at
http://www-unix.mcs.anl.gov/mpi/mpich/

The ROMIO distribution and its documentation is in
http://www.mcs.anl.gov/romio

Reference and source for SLURM

http://www.llnl.gov/linux/slurm/

InfiniBand

The InfiniBand specification is at
http://www.infinibandta.org/specs

Clusters

A good book on many of the issues concerning clusters is

Gropp, Lusk, Sterling, *Beowulf Cluster Computing with Linux*, Second Edition, 2003, MIT Press, ISBN 0-262-69292-9