ParaStationV5

ParaStation
MPI

—

Administrator's Guide

Release 5.0.7-4
Published August 2011

ParTec CLUSTER COMPETENCE CENTER

ParaStation MPI Administrator's Guide

Release 5.0.7-4
Copyright © 2002-2011 ParTec Cluster Competence Center GmbH
August 2011
Printed 9 August 2011, 16:25

ParTec Cluster Competence Center GmbH
Possartstr. 20
D-81679 München
Phone +49-89-99809-0
Fax +49-89-99809-555

http://www.par-tec.com
`<info@par-tec.com>`

The *ParaStation MPI Administrator's Guide* and *ParaStation MPI User's Guide* provide detailed information about *ParaStation MPI*. Installation and configuration of *ParaStation MPI* including process management and communication libraries are described in-depth by the *ParaStation MPI Administrator's Guide*. Usage of MPI and related tools are described by the *ParaStation MPI User's Guide*.

Though it may seem hard to believe, this manual might contain errors. We welcome any reports on errors or problems that are found. We also would appreciate suggestions on improving this book. Please direct all comments and problems to `<support@par-tec.com>`.

The most up-to-date version of this document is available at http://docs.par-tec.com.

> Share your knowledge with others. It's a way to achieve immortality.
>
> —Dalai Lama

Table of Contents

Chapter 1. Preface

1.1. About this document

This book discusses installation, configuration and administration of *ParaStation MPI*. Furthermore, all the system management utilities are described.

For a detailed discussion of the user utilities and the programming interfaces included in the standard distribution take a look at the *ParaStation MPI User's Guide* and the API reference, respectively.

This document describes version 5.0 of the *ParaStation MPI* software. Previous versions of *ParaStation MPI* are no longer covered by this document. Information about these outdated versions can be found in previous versions of this document.

The most up-to-date version of this document is available at http://docs.par-tec.com.

1.2. This book's audience

This book is targeted to cluster system administrators maintaining Linux HPC cluster and a MPI environment using *ParaStation MPI*.

The administrators should be familiar with the concepts of general Linux system installation, Linux server administration, HPC clustering, networking and MPI environments.

1.3. *ParaStation MPI* overview

ParaStation MPI is an integrated cluster management and communication solution. It combines unique features only found in *ParaStation MPI* with common techniques, widely used in high performance computing, to deliver an integrated, easy to use and reliable compute cluster environment.

The version 5 of *ParaStation MPI* supports various communication technologies as interconnect network. It comes with an optimized communication protocol for Ethernet that enables Gigabit Ethernet to play a new role in the market of high throughput, low latency communication. Beside InfiniBand and Myrinet, it also supports the upcoming 10G Ethernet networks.

Like previous versions, *ParaStation MPI* includes an integrated cluster administration and management environment. Using communicating daemon processes on each cluster node, an effective resource management and single point of administration is implemented. This results in a single system image view of the cluster.

From the user's point of view this cluster management leads to an easier and more effective usage of the cluster. Important features like load balancing, job control and input/output management, common in classical supercomputers, but rarely found in compute clusters, are implemented by *ParaStation MPI* thus being now also available on clusters.

1.4. The history of *ParaStation MPI*

The fundamentals of the *ParaStation* software were laid in 1995, when the *ParaStation* communication hardware and software system was presented. It was developed at the chair of Professor Tichy at computer science department of Karlsruhe University.

When in 1998 *ParaStation2* was presented, it was a pure software project. The communication platform used then was Myrinet, a Gigabit interconnect developed by Myricom. The development of *ParaStation2* still took place at the University of Karlsruhe.

ParaStation became commercial in 1999 when ParTec AG was founded. This spin-off from the University of Karlsruhe owned all rights and patents connected with the *ParaStation* software. ParTec promoted the further development and improvement of the software. This included the support of a broader basis of supported processor types, communication interconnect and operating systems.

Version 3 of the *ParaStation* software for Myrinet was a rewrite from scratch fully in the responsibility of ParTec. All the know-how and experiences achieved from the former versions of the software were incorporated into this version. It was presented in 2001 and was a major breakthrough with respect to throughput, latency and stability of the software. Nevertheless it was enhanced constantly with regard to performance, stability and usability.

In 2002 the *ParaStation FE* software was presented opening the *ParaStation* software environment towards Ethernet communication hardware. This first step in the direction of independence from the underlying communication hardware brought the convenient *ParaStation* management facility to Beowulf clusters for the first time. Furthermore the suboptimal communication performance for large packets gained from the MPIch/P4 implementation of the MPI message passing interface, the de facto standard on Beowulf clusters, was improved to the limits that may to be expected from the physical circumstances.

With *ParaStation4* presented in 2003 the software became really communication platform independent. With this version of the software even Gigabit Ethernet became a serious alternative as a cluster interconnect due to the throughput and latency that could be achieved.

In the middle of 2004, all rights on *ParaStation* where transferred from ParTec AG to the ParTec Cluster Competence Center GmbH. This new company takes a much more service-oriented approach to the customer. The main goal was to deliver integrated and complete software stacks for LINUX-based compute clusters by selecting state-of-the-art software components and driving software development efforts in areas where real added value can be provided. The ParTec Cluster Competence Center GmbH continued to develop and support the *ParaStation* product as an important part of it's portfolio.

At the end of 2007, *ParaStation MPI* was released supporting MPI2 and even more interconnects and especially protocols, like DAPL. *ParaStation MPI* is backward compatible to the previous *ParaStation4* version. At the same time, it was renamed to *ParaStation MPI*, as *ParaStation5* now includes more components beside a MPI.

Chapter 2. Technical overview

Within this section, a brief technical overview of *ParaStation MPI* will be given. The various software modules constituting *ParaStation MPI* are explained.

2.1. Runtime daemon

In order to enable *ParaStation MPI* on a cluster, the *ParaStation* daemon psid(8) has to be installed on each cluster node. This daemon process implements various functions:

- Install and configure local communication devices and protocols, e.g. load the *p4sock* kernel module and set up proper routing information, if not already done at system startup.
- Queue parallel and serial tasks until requested resources are available.
- Distribute processes onto the available cluster nodes.
- Startup and monitor processes on cluster nodes. Also terminate and cleanup processes upon request.
- Monitor availability of other cluster nodes, send “I'm alive” messages.
- Handle input/output and signal forwarding.
- Service management commands from the administration tools.

The daemon processes periodically send information containing application processes, system load and others to all other nodes within the cluster. So each daemon is able to monitor each other node, and in case of absent alive messages, it will initiate proper actions, e.g. terminate a parallel task or mark this node as "no longer available". Also, if a previously unavailable node is now responding, it will be marked as "available" and will be used for upcoming parallel task. No intervention of the system administrator is required.

2.2. Libraries

In addition, a couple of libraries providing communication and management functionality, must be installed. All libraries are provided as static versions, which will be linked to the application at compile time, or as shared (dynamic) versions, which are pre-linked at compile time and folded in at runtime. There is also a set of management and test tools installed on the cluster.

ParaStation MPI comes with it's own version of MPI, based on MPIch2. The MPI library provides standard MPIch2 compatible MPI functions. For communication purposes, it supports a couple of communication paths in parallel, e.g. local communication using Shared memory, TCP or p4sock, Ethernet using p4sock and TCP, InfiniBand using verbs, Myrinet using GM or 10G Ethernet using DAPL. Thus, *ParaStation MPI* is able to spawn parallel tasks across nodes connected by different communication networks. *ParaStation MPI* will also make use of redundant interconnects, if a failure is encountered during startup of a parallel task.

There are different versions of the *ParaStation MPI* library available, depending on the compiler in use. In particular, versions for the GCC, Intel and Portland Group are available. The versions support all available languages and language options for the selected compiler, e.g. Fortran, Fortran90, C or C++. The different versions of the MPI library can be installed in parallel, thus it is possible to compile and run applications using different compilers at the same node.

2.3. Kernel modules

Beside libraries enabling efficient communication and task management, *ParaStation MPI* also provides a set of kernel modules:

- `p4sock.o`: this module implements the kernel based *ParaStation* communication protocol.

- `e1000_glue.o`, `bcm5700_glue.o`: these modules enable even more efficient communication to the network drivers coming with *ParaStation MPI* (see below).
- `p4tcp.o`: this module provides a feature called "TCP bypass". Thus, applications using standard TCP communication channels on top of Ethernet are able to use the optimized *ParaStation* protocol and therefore achieve improved performance.

  No modifications of the application, even no relinking is necessary to use this feature. To gain best performance, relinking with the MPI library provided by *ParaStation MPI* is recommended.

To enable the maximum performance on Gigabit Ethernet, *ParaStation MPI* comes with its own set of network drivers. These drivers are based on standard device drivers for the corresponding NICs and especially tuned for best performance within a cluster environment. They will also support all standard communication and protocols. To enable best performance within an Ethernet-based cluster, these drivers should replace their counterparts currently configured within the kernel.

ParaStation MPI currently comes with drivers for Intel (e1000) and Broadcom (bcm5700) network interface controllers. Dedicated helper modules (glue modules) for these drivers decrease the latency even more.

ParaStation MPI is also able to use all standard Ethernet network drivers configured into the Linux kernel. However, to get the best performance, the use of the provided drivers is recommended, if applicable.

2.4. License

This version of *ParaStation MPI* does not require a dedicated license key to run. **But usage of the software implies the acceptance of the ParaStation license!**

For licensing details, refer to Appendix B, *ParaStation license*.

Chapter 3. Installation

This chapter describes the installation of *ParaStation MPI*. At first, the prerequisites to use *ParaStation MPI* are discussed. Next, the directory structure of all installed components is explained. Finally, the installation using RPM packages is described in detail.

Of course, the less automated the chosen way of installation is, the more possibilities of customization within the installation process occur. On the other hand even the most automated way of installation, the installation via RPM, will give a suitable result in most cases.

For a quick installation guide refer to Appendix A, *Quick Installation Guide*.

3.1. Prerequisites

In order to prepare a bunch of nodes for the installation of the *ParaStation MPI* communication system, a few prerequisites have to be met.

Hardware

The cluster must have a homogeneous processor architecture. The supported architectures up to now are:

- `i586`: Intel IA32 (including AMD Athlon)
- `ia64`: Intel IA64
- `x86_64`: Intel EM64T and AMD64
- `ppc`: IBM Power4 and Power5

Multi-core CPUs are supported, as well as single and multi-CPU (SMP) nodes.

Furthermore the nodes need to be interconnected. In principle, *ParaStation MPI* uses two different kinds of interconnects:

- At first a so called administration network which is used to handle all the administrative tasks that have to be dealt with within a cluster. Besides commonly used services like sharing of NFS partitions or NIS tables, on a *ParaStation MPI* cluster, this also includes the inter-daemon communication used to implement the effective cluster administration and parallel task handling mechanisms. This administration network is usually implemented using a Fast or Gigabit Ethernet network.
- Secondly a high speed interconnect is required in order to do high bandwidth, low latency communication within parallel applications. While historically this kind of communication is usually done using specialized highspeed networks like Myrinet, nowadays Gigabit Ethernet is a much cheaper and only slightly slower alternative. *ParaStation MPI* currently supports Ethernet (Fast, Gigabit and 10G Ethernet), Myrinet, InfiniBand, QsNetII and Shared Memory.

If IP connections over the high speed interconnect are available, it is not required to really have two distinct networks. Instead it is possible to use one physical network for both tasks. IP connections are usually configured by default in the case of Ethernet. For other networks, particular measures have to be taken in order to enable IP over these interconnects.

Software

ParaStation MPI requires a RPM-based Linux installation, as the *ParaStation MPI* software is based on installable RPM packages.

All current distributions from Novell and Red Hat are supported, like

- SuSE Linux Enterprise Server (SLES) 10 and 11
- OpenSuSE up to version 11.3
- Red Hat Enterprise Linux (RHEL) 4 and 5
- Fedora Core, up to version 11
- CentOS 4 and 5

For other distributions and non-RPM based installations, please contact `<support@par-tec.com>`.

In order to use highspeed networks, additional libraries and kernel modules may be required. These packages are typically provided by the hardware vendors.

Kernel version

Using only TCP as a high speed interconnect protocol, no dedicated kernel modules are required. This is the *ParaStation MPI* default communication path and is always enabled.

Using other interconnects and protocols, additional kernel modules are required. Especially using the optimized *ParaStation* p4sock protocol, a couple of additional modules are loaded. Refer to the section called “Installing the RPMs” for details. The *ParaStation MPI* modules can be compiled for all major kernel versions.

Using InfiniBand and Myrinet requires additional modules and may restrict the supported kernels.

3.2. Directory structure

The default location to install *ParaStation MPI* is `/opt/parastation`. Underneath this directory, several subdirectories are created containing the actual *ParaStation MPI* installation:

`bin`
: contains all executables and scripts forming the *ParaStation MPI* system. This directory could be included into the `PATH` environment variable for easy access to the *ParaStation MPI* administration tools.

`config`
: contains the example configuration file `parastation.conf.tmpl`.

  Depending on the communication part of the *ParaStation MPI* system installed, more scripts and configuration files may be found within this directory.

`doc`
: contains the *ParaStation MPI* documentation after installing the corresponding RPM file. The necessary steps are described in Section 3.4, “Installing the documentation”.

`include`
: contains the header files needed in order to build *ParaStation MPI* applications. These files are primarily needed for building applications using the low level PSPort or PSE libraries.

  These header files are not needed, if only MPI applications should be build or precompiled third party applications are used.

`lib` and `lib64`
: contains various libraries needed in order to build and/or run applications using *ParaStation MPI* and the *ParaStation MPI* libraries.

`man`
: contains the manual pages describing the *ParaStation MPI* daemons, utilities and configuration files after installing the documentation package. The necessary steps are described in Section 3.4, “Installing the documentation”.

In order to enable the users to access these pages using the man(1) command, please consult the corresponding documentation [1].

`mpi2, mpi2-intel, mpi2-pgi`
: contains an adapted version of MPIch2 after installing one of the various `psmpi2` RPM files. The necessary steps are described in Section 3.5, “Installing MPI”.

  Especially the sub-directories `mpi2/bin`, `mpi2-intel/bin`, etc, contain all the commands to run (**mpiexec**) and compile (**mpicc**, **mpif90**, ...) parallel tasks.

All *ParaStation* specific kernel modules are located within the directory `/lib/modules/`*`kernel release`*`/kernel/drivers/net/ps4`.

3.3. Installation via RPM packages

The recommended way to install *ParaStation MPI* is the installation of packages using the **rpm** command. This is the preferred method on all SuSE or Red Hat based systems.

Getting the *ParaStation MPI* RPM packages

Packages containing the different parts of the *ParaStation MPI* system can be obtained from the download section of the *ParaStation MPI* homepage.

At least two packages are needed, one containing the management part, the other one providing the communication part of the *ParaStation MPI* system. Beside this core system, packages supplying MPIch for GNU, Intel and Portland Group are available. A documentation package is also obtainable.

The full names of the RPM files follow a simple structure:

```
name-x.y.z-n.arch.rpm
```

where `name` denotes the name and thus the content of the packet, `x.y.z` describes the version number, `n` the build number and `arch` is the architecture, i.e. one of `i586`, `ia64`, `x86_64`, `ppc` or `noarch`. The latter is used e.g. for the documentation packet.

The package called `psmgmt` holds the management part of *ParaStation MPI*. This package is required for any installation of the *ParaStation MPI* system, independent of the underlying communication platform.

The communication libraries and modules for *ParaStation MPI* come with the `pscom` package. As explained, all filenames are followed by an individual version number, the build date and the architecture.

The MPI libraries and tools are included in the package `psmpi2`, `psmpi2-intel` or `psmpi2-pgi` for the respective compilers.

The versions available on the *ParaStation MPI* homepage at a time are tested to work properly together. It's recommended to install always the corresponding package versions. If only a part of the installation should be updated (i.e. only the management part while keeping the communication part untouched) the corresponding release notes should be consulted in order to verify that the intended combination is supported.

The release notes of the different packages will either be found within the installation directory `/opt/parastation` or on the download section of the *ParaStation MPI* homepage.

Please note that the individual version numbers of the distinct packages building the *ParaStation MPI* system do not necessarily have to match.

[1] Usually this is either done by modifying the `MANPATH` environment variable or by editing the manpath(1) configuration file, which is `/etc/manpath.config` by default.

Compiling the *ParaStation MPI* packages from source

To build proper RPM packages suitable for a particular setup, the source code for the *ParaStation MPI* packages can be downloaded from www.parastation.com/downloads [2] .

Typically, it is not necessary to recompile the *ParaStation MPI* packages, as the provided precompiled packages will install on all major distributions.

Only the kernel modules (if required) should be compiled to provide modules suitable for the current Linux kernel, see below.

To build the `psmgmt` package, use

```
# rpmbuild --rebuild psmgmt.5.0.0-0.src.rpm
```

After installing the `psmgmt` package, the `pscom` package can be built using

```
# rpm -Uv psmgmt.5.0.0-0.i586.rpm
# rpmbuild --rebuild pscom.5.0.0-0.src.rpm
```

This will build the packages `pscom-5.0.0-0.i586.rpm` and `pscom-modules-5.0.0-0.i586.rpm`. The architecture will of course vary depending on the system the packages are built on.

While compiling the package, support for InfiniBand will be included, if one of the following files where found:

File	**Version**
`/usr/include/infiniband/verbs.h`	OpenFabrics
`/usr/mellanox/include/vapi/evapi.h`	Mellanox
`/usr/local/ofed/include/verbs.h`	OpenFabrics (Voltaire)

Table 3.1. Supported InfiniBand implementations

To enable Myrinet GM, the environment variable `GM_HOME` must be set.

To generate the `pscom-modules` package, holding the *ParaStation* protocol-specific kernel modules and patched device drivers only, use the command

```
# rpmbuild --rebuild --with modules pscom.5.0.0-0.src.rpm
```

After installing the `pscom` package, the MPIch2 package can be built using

```
# rpm -Uv pscom.5.0.0-0.i586.rpm
# rpmbuild --rebuild psmpi2-5.0.0.src.rpm
```

This will create an installable MPIch RPM package, based on gcc. Support for other compilers can be enabled using the `--with compiler` options. *`Compiler`* could be *`intel`* for Intel icc or *`pgi`* for Portland Group pgi. The option *`g77_`* will use gcc, rendering symbol names with a single underscore prefixed.

Installing the RPMs

The installation on the cluster nodes has to be performed with administrator privileges. The packages are installed using the **rpm -U** command:

```
# rpm -Uv psmgmt.5.0.0-0.i586.rpm pscom.5.0.0-0.i586.rpm \
pscom-modules.5.0.0-0.i586.rpm
```

[2] Source code for the documentation is currently not available.

This will copy all the necessary files to `/opt/parastation` and the kernel modules to `/lib/modules/`*`kernelversion`*`/kernel/drivers/net/ps4`.

On a frontend node or file server, the `pscom-modules` package is only required, if this node should run processes of a parallel task. If the frontend or fileserver node is not configured to run compute processes of parallel tasks, the installation of the `pscom-modules` package may be skipped. For details how to configure frontend nodes, refer to Section 4.1, “Configuration of the *ParaStation MPI* system”.

To enable the *ParaStation MPI* version of the e1000 or bcm5700 network drivers, rename (or delete) the original version of this driver in use which is typically located in the system directory `/lib/modules/`*`kernelversion`*`/kernel/kernel/drivers/net/e1000` or `bcm`, respectively. See **modinfo *e1000*** for details. The module dependency database must be rebuild using the command **depmod**. See Section 4.2, “Enable optimized network drivers” for details.

It is not required to use the *ParaStation MPI* version of the e1000 or bcm5700 driver, as the p4sock protocol of *ParaStation MPI* is able to use every network driver within the Linux kernel. However, to increase performance and to minimize latency, it's highly recommended.

Using the provided drivers does not influence other network communication.

While installing the *ParaStation MPI* management RPM, the file `/etc/xinetd/psidstarter` is installed. This enables remote startup of *ParaStation MPI* daemons using the xinetd(8).

The **xinetd** daemon will be triggered to read this file by executing:

```
/etc/init.d/xinetd reload
```

Refer to Section 5.22, “Changing the default ports for psid(8)” on how to change the default network port used by the psid(8).

In case the system still uses the older xinet(8) server to startup network services, please add the following lines to `/etc/services`:

```
#
# ParaStation daemon
#
psid stream tcp nowait root /opt/parastation/bin/psid psid
```

Add the next lines to `/etc/inetd.conf`:

```
# ParaStation entries
psid    888/tcp     # ParaStation Daemon Start Port
# end of ParaStation entries
```

3.4. Installing the documentation

The *ParaStation MPI* documentation is delivered in three formats: As PDF files, a browseable HTML documentation and manual pages for all *ParaStation MPI* commands and configuration files.

In order to install the documentation files, an up to date version of the documentation package `psmpi-doc` has to be retrieved. It can be found in the download section of the *ParaStation MPI* homepage. The package is called `psmpi-doc` and the architecture is `noarch`, since this part is platform independent. The name of this package follows the conventions of all other packages building the *ParaStation MPI* distribution.

To install the package, simply execute

```
# rpm -Uv psmpi-doc-5.0.0-1.noarch.rpm
```

All the PDF and HTML files will be installed within the directory `/opt/parastation/doc`, the manual pages will reside in `/opt/parastation/man`.

The intended starting point to browse the HTML version of the documentation is `file:///opt/parastation/doc/html/index.html`.

The documentation is available in two PDF files called `adminguide.pdf` for the *ParaStation MPI Administrator's Guide* and `userguide.pdf` for the *ParaStation MPI User's Guide*. Both can be found in the directory `/opt/parastation/doc/pdf`.

In order to enable the manual pages to the users please consult the documentation of the man(1) command and the remark in Section 3.2, “Directory structure”, on how to do this.

3.5. Installing MPI

The standard for the implementation of parallel applications on distributed memory machines like clusters is MPI, the Message Passing Interface. In order to enable a *ParaStation MPI* cluster for the development and execution of MPI programs, the installation of an adapted version of MPIch2 is necessary. A corresponding RPM packet can be found within the download section of the *ParaStation MPI* homepage.

The corresponding package `psmpi2` follows the common naming conventions of all *ParaStation MPI* packets.

Packages compiled on x86_64 for different distributions are available on www.par-tec.com.

After downloading the correct MPI package make sure to be root. In order to install MPIch for *ParaStation MPI* from the rpm file, the command

```
 # rpm -Uv psmpi2.5.0.0-1.i586.rpm
```

must be executed.

The command will extract the *ParaStation MPI* package to the directory `/opt/parastation/mpi2`.

In order to enable the MPI commands to the users make sure that the directory `/opt/parastation/mpi2/bin` is included into the system wide `PATH` environment variable. Furthermore the administrator might want to enable the MPI manual pages for all users. These pages reside in `/opt/parastation/mpi2/man`. Please consult the documentation of the man(1) command and the remark in Section 3.2, “Directory structure” on how to do this.

In general, all versions of *ParaStation MPI* supporting different compilers can be installed in parallel. The files will be copied to the directories `/opt/parastation/mpi2` (for GNU gcc), `/opt/parastation/mpi2-intel` (for Intel compiler) or `/opt/parastation/mpi2-pgi` (for Portland Group compiler), depending on the compiler version supported by the MPI package.

3.6. Further steps

The previous chapter described the basic installation of *ParaStation MPI*. There are still some steps to do, especially:

- configuration of the *ParaStation MPI* system
- testing

These steps will be discussed in Chapter 4, *Configuration*.

3.7. Uninstalling *ParaStation MPI*

After stoping the *ParaStation MPI* daemons, the corresponding packets can be removed using

```
  # /etc/init.d/parastation stop
  # rpm -e psmgmt pscom psmpi-doc psmpi2 psmpi2-doc
```

on all nodes of the cluster.

Chapter 4. Configuration

After installing the *ParaStation MPI* software successfully, only few modifications to the configuration file parastation.conf(5) have to be made in order to enable *ParaStation MPI* on the local cluster.

4.1. Configuration of the *ParaStation MPI* system

Within this section the basic configuration procedure to enable *ParaStation MPI* will be described. It covers the configuration of *ParaStation MPI* using TCP/IP (Ethernet) and the optimized *ParaStation* protocol *p4sock*.

The primarily configuration work is reduced to editing the central configuration file `parastation.conf`, which is located in `/etc`.

A template file can be found in `/opt/parastation/config/parastation.conf.tmpl`. Copy this file to `/etc/parastation.conf` and edit it as appropriate.

This section describes all parameters of `/etc/parastation.conf` necessary to customize *ParaStation MPI* for a basic cluster environment. A detailed description of all possible configuration parameters in the configuration file can be found within the parastation.conf(5) manual page.

The following steps have to be executed on the frontend node to configure the *ParaStation MPI* daemon psid(8):

1. **Copy template**

   Copy the file `/opt/parastation/config/parastation.conf.tmpl` to `/etc/parastation.conf`.

   The template file contains all possible parameters known by the *ParaStation MPI* daemon psid(8). Most of these parameters are set to their default value within lines marked as comments. Only those that have to be modified in order to adapt *ParaStation MPI* to the local environment are enabled. Additionally all parameters are exemplified using comments. A more detailed description of all the parameters can be found in the parastation.conf(5) manual page.

   The template file is a good starting point to create a working configuration of *ParaStation MPI* for your cluster. Beside basic information about the cluster, this template file defines all hardware components *ParaStation MPI* is able to handle. Since these definitions require a deeper knowledge of *ParaStation MPI*, it is easier to copy the template file anyway.

2. **HWType**

   In order to tell *ParaStation MPI* which general kind of communication hardware should be used, the **HWType** parameter has to be set. This could be changed on a per node basis within the nodes section (see below).

   For clusters running *ParaStation MPI* utilizing the optimized *ParaStation MPI* communication stack on Ethernet hardware of any flavor this parameter has to be set to:

   ```
   HWType { p4sock ethernet }
   ```

   This will use the optimized *ParaStation MPI* protocol, if available. Otherwise, TCP/IP will be used.

   The values that might be assigned to the **HWType** parameter have to be defined within the `parastation.conf` configuration file. Have a brief look at the various **Hardware** sections of this file in order to find out which hardware types are actually defined.

   Other possible types are: *`mvapi`*, *`openib`*, *`gm`*, *`ipath`*, *`elan`*, *`dapl`*.

To enable shared memory communication used within SMP nodes, no dedicated hardware entry is required. Shared memory support is always enabled by default. As there are no options for shared memory, no dedicated hardware section for this kind of interconnect is provided.

3. Define **Nodes**

   Furthermore *ParaStation MPI* has to be told which nodes should be part of the cluster. The usual way of using the **Nodes** parameter is the environment mode, that is already enabled in the template file.

   The general syntax of the **Nodes** environment is one entry per line. Each entry has the form

   ```
   hostname id [HWType] [runJob] [starter] [accounter]
   ```

   This will register the node *`hostname`* to the *ParaStation MPI* system with the *ParaStation MPI* ID *`id`*. The *ParaStation* ID has to be an integer number between 0 and the maximum number of nodes minus one.

   For each cluster node defined within the **Nodes** environment at least the hostname of the node and the *ParaStation MPI* ID of this node have to be given. The optional parameters **HWType**, **runJobs**, **starter** and **accounter** may be ignored for now. For a detailed description of these parameters refer to the parastation.conf(5) manual page.

   Usually the nodes will be enlisted ordered by increasing *ParaStation MPI* IDs, beginning with 0 for the first node. If a front end node exists and furthermore should be integrated into the *ParaStation MPI* system, it usually should be configured with ID 0.

   Within an cluster the mapping between hostnames and *ParaStation MPI* ID is completely unrestricted.

4. More options

   More configuration options may be set as described in the configuration file `parastation.conf`. For details refer to the parastation.conf(5) manual page.

   If using `vapi` (HwType ib) or `DAPL` (HwType dapl) layers for communication, e.g. for InfiniBand or 10G Ethernet, the amount of lockable memory must be increased. To do so, use the option *`rlimit memlock`* within the configuration file.

5. Copy configuration file to all other nodes

   The modified configuration file must be copied to all other nodes of the cluster. E.g., use **psh** to do so. Restart all *ParaStation MPI* daemons.

In order to verify the configuration, the command

```
# /opt/parastation/bin/test_config
```

could be run. This command will analyze the configuration file and report any configuration failures. After finishing these steps, the configuration of *ParaStation MPI* is done.

4.2. Enable optimized network drivers

As explained in the previous chapter, *ParaStation MPI* comes with its own versions of adapted network drivers for Intel (e1000) and Broadcom (bcm5700) NICs. If the optimized *ParaStation MPI* protocol **p4sock** is used to transfer application data across Ethernet, this adapted drivers should be used, too. To enable these drivers, the simplest way is to rename the original modules and recreate the module dependencies:

```
# cd /lib/modules/$(uname -r)/kernel/drivers/net
# mv e1000/e1000.o e1000/e1000-orig.o
```

```
# mv bcm/bcm5700.o bcm/bcm5700-orig.o
# depmod -a
```

If your system uses the e1000 driver, a subsequent **modinfo** command for kernel version 2.4 should report that the new *ParaStation MPI* version of the driver will be used:

```
# modinfo e1000
filename: /lib/modules/2.4.24/kernel/drivers/net/ps4/e1000.o
description: "Intel(R) PRO/1000 Network Driver"
author:      "Intel Corporation, <linux.nics@intel.com>"
...
```

The "filename" entry reports that the *ParaStation MPI* version of the driver will be used. The same should apply for the **bcm5700** network driver.

For kernel version 2.6, use the **modprobe** command:

```
# modprobe -l e1000
/lib/modules/2.6.5-7.97/kernel/drivers/net/ps4/e1000.ko
```

To reload the new version of the network drivers, it is necessary to reboot the system.

4.3. Testing the installation

After installing and configuring *ParaStation MPI* on each node of the cluster, the *ParaStation MPI* daemons can be started up. These daemons will setup all necessary communication relations and thus will form the virtual cluster consisting of the available nodes.

The *ParaStation MPI* daemons are started using the **psiadmin** command. This command will establish a connection to the local **psid**. If this daemon is not already up and running, the **inetd** will start up the daemon automatically.

If the daemon is not configured to be automatically started by xinetd, it must be started using `/etc/init.d/parastation start`.

```
# /opt/parastation/bin/psiadmin
```

After connecting to the local psid daemon, this command will issue a prompt

```
psiadmin>
```

To start up the *ParaStation MPI* daemons on all other nodes, use the add command:

```
psiadmin> add
```

The following status inquiry command

```
psiadmin> list
```

should list all nodes as "up". To verify that all nodes have installed the proper kernel modules, type

```
psiadmin> list hw
```

The command should report for all nodes all hardware types configured, e.g. *`p4sock`*, *`ethernet`*.

Alternatively, it is possible to use the single command form of the psiadmin command:

```
# /opt/parastation/bin/psiadmin -s -c "list"
```

The command should be repeated until all nodes are up. The *ParaStation MPI* administration tool is described in detail in the corresponding manual page psiadmin(1).

If some nodes are still marked as "down", the logfile `/var/log/messages` for this node should be inspected. Entries like “psid: ....” at the end of the file may report problems or errors.

After bringing up all nodes, the communication can be tested using

```
# /opt/parastation/bin/test_nodes -np nodes
```

where *nodes* has to be replaced by the actual number of nodes within the cluster. After a while a result like

```
---------------------------------------
Master node 0
Process 0-31 to 0-31 ( node 0-31 to 0-31 ) OK
All connections ok

PSIlogger: done
```

should be reported. Of course the number '31' will be replaced by a the actual number of nodes given on the command line, i.e. *nodes*-1.

in case of failure, **test_nodes** may give continuously results like

```
---------------------------------------
Master node 0
Process 0-2,4-6 to 0-7 ( node 0-2,4-6 to 0-7 ) OK
Process 3 to 0-6 ( node 3 to 0-6 ) OK
Process 7 to 0-2,4-7 ( node 7 to 0-2,4-7 ) OK
```

A detailed description of **test_nodes** can be found within the corresponding manual page test_nodes(1).

Chapter 5. Insight *ParaStation MPI*

This chapter provides more technical details and background information about *ParaStation MPI*.

5.1. *ParaStation* `pscom` communication library

The *ParaStation MPI* communication library `libpscom` offers secure and reliable end-to-end connectivity. It hides the actual transport and communication characteristics from the application and higher level libraries.

The `libpscom` library supports a wide range of interconnects and protocols for data transfers. Using a generic plug-in system, this library may open connections using the following networks and protocols:

- `TCP:` uses standard TCP/IP sockets to transfer data. This protocol may use any interconnect. Support for this protocol is built-in to the `libpscom`.
- `P4sock:` uses an optimized network protocol for Ethernet (see Section 5.2, “*ParaStation* protocol *p4sock*”, below). Support for this protocol is built-in to the `libpscom`.
- `InfiniBand:` based on a verbs kernel layer and a `libverbs` library, typically provided by the hardware vendor, the `libpscom` may use InfiniBand to actually transfer data. The corresponding plug-in library is called `libpscom4openib`.
- `Myrinet:` using the GM library and kernel level module, the `libpscom` library is able to use Myrinet for data transfer. The particular plug-in library is called `libpscom4gm`.
- `Shared Memory:` for communication within a SMP node, the `libpscom` library uses shared memory. Support for this protocol is built-in to the `libpscom`.
- `DAPL:` The `libpscom` supports a DAPL transport layer. Using the `libpscom4dapl` plug-in, it may transfer data across various networks like InfiniBand or 10G Ethernet using a vendor-provided `libdapl`.
- `QsNet:` The `libpscom` supports the QsNetII transport layer. Using the `libpscom4elan` plug-in, it may transfer data using the `libelan`.

The interconnect and protocol used between two distinct processes is chosen while opening the connection between those processes. Depending on available hardware, configuration (see Section 4.1, “Configuration of the *ParaStation MPI* system”) and current environment variables (see Section 5.8, “Controlling *ParaStation MPI* communication paths”), the library automatically selects the fastest available communication path.

The library routines for sending and receiving data handle arbitrary large buffers. If necessary, the buffers will be fragmented and reassembled to meet the underlying transport requirements.

The application is dynamically linked with the `libpscom.so` library. At runtime, this library loads plug-ins for various interconnects, see above. For more information on controlling *ParaStation MPI* communication pathes, refer to Section 5.8, “Controlling *ParaStation MPI* communication paths”.

5.2. *ParaStation* protocol *p4sock*

ParaStation provides its own communication protocol for Ethernet, called *p4sock*. This protocol is designed for extremely fast and reliable communication within a closed and homogeneous compute cluster environment.

The protocol implements a reliable, connection-oriented communication layer, especially designed for very low overhead. As a result, it delivers very low latencies.

The *p4sock* protocol is encapsulated within the kernel module *p4sock.ko*. This module is loaded on system startup or whenever the *ParaStation* daemon psid(8) starts up and the *p4sock* protocol is enabled within the configuration file parastation.conf(5).

The *p4sock.ko* module inserts a number of entries within the `/proc` filesystem. All *ParaStation* entries are located within the subdirectory `/proc/sys/ps4`. Three different subdirectories, listed below, are available.

To read a value, e.g. just type

```
# cat /proc/sys/ps4/state/connections
```

to get the number of currently open connections. To modify a value, for e.g. type

```
# echo 10 > /proc/sys/ps4/state/ResendTimeout
```

to set the new value for `ResendTimeout`.

5.2.1. Directory `/proc/sys/ps4/state`

Within this `state` directory, various entries showing protocol counters. All these entries, except `polling`, are read only!

- `HZ:` reads the number of timer interrupts per second for this kernel ("jiffies").

A jiffy is the base unit for system timers, used by the Linux kernel. So all timeouts within the kernel are based on this timer resolution. On kernels with version 2.4, this it typically 100Hz (= 10ms). But there are kernel versions available, e.g. for newer SuSE Linux versions, which include patches to change this to a much higher value!

- `connections:` reads the current number of open connections.
- `polling:` returns the current value for the polling flag: 0 = never poll, 1 = poll if otherwise idle (number of runable processes < number of CPUs), 2 = always poll. Writing this value will immediately change the polling strategy.
- `recv_net_ack:` number of received ACKs.
- `recv_net_ctrl:` number of received control packets (ACK, NACK, SYN, SYNACK, ...).
- `recv_net_data:` number of received data packets.
- `recv_net_nack:` number of received NACKs.
- `recv_user:` number of packets delivered to application buffers.
- `send_net_ack:` number of sent ACKs.
- `send_net_ctrl:` number of sent control packets.
- `send_net_data:` number of sent data packets.
- `send_net_nack:` number of sent NACKs.
- `send_user:` number of packets sent by the application.
- `sockets:` number of open sockets connecting to the *ParaStation* protocol module.
- `timer_ack:` number of expired delayed ACK timers.
- `timer_resend:` number of expired resend timers.

5.2.2. Directory `/proc/sys/ps4/ether`

Within this directory, all Ethernet related parameters for the *ParaStation p4sock* protocol are grouped. All these entries can be read and written, newly written values will be used immediately.

- `AckDelay:` maximum delay in "jiffies" for ACK messages. If no message is sent within this time frame, where an ACK for already received packets can be "hooked up", a single ACK message will generated. Must be less then `ResendTimeout`.

- `MaxAcksPending:` maximum number of pending ACK messages until an "urgent" ACK messages will be sent.
- `MaxDevSendQSize:` maximum number of entries of the (protocol internal) send queue to the network device.
- `MaxMTU:` maximum packet size used for network packets. For sending packets, the minimum of `MaxMTU` and service specific MTU will be used.
- `MaxRecvQSize:` size of the protocol internal receive queue.
- `MaxResend:` Number of retries until a connection is declared as dead.
- `MaxSendQSize:` size of the protocol internal send queue.
- `ResendTimeout:` delay in "jiffies" for resending packets not acknowledged up to now. Must be greater then `AckDelay`.

5.2.3. Directory `/proc/sys/ps4/local`

Currently, there are no entries defined for this directory.

5.2.4. `p4stat`

The command **p4stat** can be used to list open sockets and network connections of the *p4sock* protocol.

```
$ /opt/parastation/bin/p4stat -s
Socket #0 : Addr: <00><00><00><00><00><'........' last_idx 0 refs 2
Socket #1 : Addr: <70><6f><72><74><33><'port384.' last_idx 0 refs 10
Socket #2 : Addr: <70><6f><72><74><31><'port144.' last_idx 0 refs 10

$ /opt/parastation/bin/p4stat -n
net_idx  SSeqNo SWindow RSeqNo RWindow lusridx lnetidx rnetidx snq rnq refs
     84   30107   30467  30109   30468      84      84     230   0   0    2
     85   30106   30466  30106   30465      85      85     231   0   0    2
     86   30107   30467  30109   30468      86      86     84    0   0    2
     87   30106   30466  30106   30465      87      87     85    0   0    2
     88   30107   30467  30109   30468      88      88     217   0   0    2
     89   30106   30466  30106   30465      89      89     218   0   0    2
     90   30106   30466  30106   30465      90      90     220   0   0    2
     91   30106   30466  30106   30465      91      91     221   0   0    2
     92   30001   30361  30003   30362      92      92     232   0   0    2
     93   30001   30361  30003   30362      93      93     219   0   0    2
     94   30000   30000  30001   30360      94      94     233   0   0    2
     95   30000   30000  30001   30360      95      95     222   0   0    2
     96   30000   30000  30001   30360      96      96     222   0   0    2
```

This command shows some protocol internal parameters, like open connections, sequence numbers, reference counters, etc. For more information, see p4stat(8).

5.3. Controlling process placement

ParaStation MPI includes sophisticated functions to control the process placement for newly created parallel and serial tasks. These processes typically require a dedicated CPU (core). Upon task startup, the environment variables *`PSI_NODES`*, *`PSI_HOSTS`* and *`PSI_HOSTFILE`* are looked up (in this order) to get a predefined node list. If not defined, all currently known nodes are taken into account. Also, the variables *`PSI_NODES_SORT`*, *`PSI_LOOP_NODES_FIRST`*, *`PSI_EXCLUSIVE`* and *`PSI_OVERBOOK`* are observed. Based on these variables and the list of currently active processes, a sorted list of nodes is constructed, defining the final node list for this new task.

Beside this environment variables, node reservations for users and groups are also observed. See psiadmin(1).

In addition, only available nodes will be used to start up processes. Currently not available nodes will be ignored.

Obeying all these restrictions, the processes constructing a parallel task will be spawned on the nodes listed within the final node list. For SMP systems, all available CPUs (cores) on this node may be used for consecutive ranks, depending on the environment variable *`PSI_LOOP_NODES_FIRST`*.

For administrative tasks not requiring a dedicated CPU (core), e.g. processes spawned using **pssh**, other strategies take place. As this type of processes are intended to run on dedicated nodes, predefined by the user, the described procedure will be circumvented and the processes will be run on the user-defined nodes.

For a detailed discussion of placing processes within *ParaStation*, please refer to process placement(7), ps_environment(5), pssh(8) and mpiexec(8).

5.4. Using the *ParaStation MPI* queuing facility

ParaStation MPI is able to queue task start requests if required resources are currently in use. This queuing facility is disabled by default. It can be enabled by each user independently. All requests of all users are held within one queue and managed on a first-come-first-serve strategy.

For details, refer to *ParaStation MPI User's Guide*.

5.5. Exporting environment variables for a task

ParaStation MPI by default exports only a limited set of environment variables to newly spawned processes, like *`HOME`*, *`USER`*, *`SHELL`* or *`TERM`*.

Additional variables can be exported using `PSI_EXPORTS`.

For a complete list of environment variables known to and exported automatically by *ParaStation MPI*, refer to ps_environment(5).

For more details, refer to *ParaStation MPI User's Guide*.

Environment variables may also be inherited by the *ParaStation* daemon. See parastation.conf(5) and psiadmin(1) for details.

5.6. Using non-*ParaStation MPI* applications

It is possible to run programs linked with 3rd party MPI libraries within the *ParaStation MPI* environment. Currently supported MPI1 compatible libraries are:

- MPIch using ch_p4 (mpirun_chp4)
- MPIch using GM (mpirun_chgm)
- InfiniPath (mpirun-ipath-ps)
- MVAPIch (mpirun_openib)
- QsNet MPI (mpirun_elan)

In order to run applications linked with one of those MPI libraries, *ParaStation MPI* provides dedicated **mpirun** commands. The processes for those type of parallel tasks are spawned obeying all restrictions described in Section 5.3, “Controlling process placement”. Of course, the data transfer will be based on the communication channels supported by the particular MPI library. For MPIch using ch_p4 (TCP), *ParaStation* provides an alternative, see Section 5.7, “*ParaStation MPI* TCP bypass”.

The command **mpirun-ipath-ps** running programs linked with InfiniPath™ MPI is part of the `psipath` package. For details how to obtain this package, please contact <`support@par-tec.com`>.

For more information refer to mpirun_chp4(8), mpirun_chgm(8), mpirun-ipath-ps(8), mpirun_openib(8) and mpirun_elan(8).

Using the *ParaStation MPI* command **mpiexec**, any parallel application supporting the PMI protocol, which is part of the MPI2 standard, may be run using the *ParaStation MPI* process environment. Therefore, many other MPI2 compatible MPI libraries are now supported by *ParaStation MPI*.

It is also possible to run serial applications, thus applications not parallelized with MPI, within *ParaStation MPI*. *ParaStation MPI* distinguishes between serial tasks allocating a dedicated CPU within the resource management system and administrative tasks not allocating a CPU. To execute a serial program, run **mpiexec** `-n 1` To run an administrative task, use **pssh** or **mpiexec** `-A -n 1`.

For more details on how to start-up serial and parallel jobs refer to mpiexec(8), pssh(8) and the *ParaStation MPI User's Guide*.

5.7. *ParaStation MPI* TCP bypass

ParaStation MPI offers a feature called "TCP bypass", enabling applications based on TCP to use the efficient *p4sock* protocol. The data will be redirected within the kernel to the *p4sock* protocol. No modifications to the application are necessary!

To automatically configure the TCP bypass during *ParaStation MPI* startup, insert a line like

```
Env PS_TCP FirstAddress-LastAddress
```

in the `p4sock`-section of the configuration file `parastation.conf`, were *`FirstAddress`* and *`LastAddress`* are the first and last IP addresses for which the bypass should be configured.

To enable the bypass for a pair of processes, the library `libp4tcp.so`, located in the directory `/opt/parastation/lib64` must be preloaded by both processes using:

```
export LD_PRELOAD=/opt/parastation/lib64/libp4tcp.so
```

For parallel and serial tasks launched by *ParaStation MPI*, this environment variable is exported to all processes by default. Please refer to ps_environment(5).

It's not recommended to insert `libp4tcp.so` in the global preload configuration file `/etc/ld.so.preload`, as this may hang connections to daemon processes started up before the bypass was configured.

See also p4tcp(8).

5.8. Controlling *ParaStation MPI* communication paths

ParaStation MPI uses different communication paths, see Section 5.1, “*ParaStation* `pscom` communication library” for details. In order to restrict or prioritize the paths to use, a number of environment variables are recognized by *ParaStation MPI*.

PSP_SHM or PSP_SHAREDMEM (deprecated)
: Don't use shared memory for communication within the same node.

PSP_TCP
: Don't use TCP for communication.

PSP_P4S or PSP_P4SOCK (deprecated)
: Don't use *ParaStation MPI* p4sock protocol for communication.

PSP_MVAPI
: Don't use Mellanox InfiniBand vapi for communication.

PSP_OPENIB
: Don't use InfiniBand verbs for communication.

PSP_GM
: Don't use GM (Myrinet) for communication.

PSP_DAPL
: Don't use DAPL for communication.

To disable the particular transport, the corresponding variable must be set to 0. To enable a transport, the variable must be set to 1 or the variable must not be defined. To prioritize a communication path, assign a value greater than 1.

Using the environment variable PSP_LIB, it is possible to define the communication library to use, independent of the variables mentioned above. This library must match the currently available interconnect and protocol, otherwise an error will occur.

The library name must be specified using the full path and filename, e.g.

```
export PSP_LIB=/opt/parastation/lib64/libpscomopenib.so
```

This variable is automatically exported to all processes started by *ParaStation MPI*. Refer to Section 5.1, "*ParaStation* pscom communication library" for a full list of available library variants.

If more than one path for a particular interconnect exist, e.g. if the nodes are connected by two Gigabit Ethernet networks in parallel, it is desirable to pretend the interface and therefore the network to be used for application data. To do so, the environment variable PSP_NETWORK has to be defined.

Assuming the network 192.168.1.0 is dedicated to management data and the network 192.168.2.0 is intended for application data, the following configuration within parastation.conf would re-direct the application data to the network 192.168.2.0:

```
Env PSP_NETWORK 192.168.2.0
Nodes {
  node0   0  # 192.168.1.1
  node1   1  # 192.168.1.2
  ...
}
```

Refer to ps_environment(5) for details.

5.9. Authentication within *ParaStation MPI*

Whenever a process of a parallel task is spawned within the cluster, *ParaStation MPI* does not authenticate the user. Only the user and group ID is copied to the remote node and used for starting up processes.

Thus, it is not necessary for the user to be known by the compute node, e.g. having an entry in /etc/passwd. On the contrary, the administrator may disallow logins for users by removing the entries from /etc/passwd. Usage of common authentication schemes like NIS is not required and therefore limits user management to the frontend nodes.

Authentication of users is restricted to login or frontend nodes and is outside of the scope of *ParaStation MPI*.

5.10. Homogeneous user ID space

As explained in the previous section, *ParaStation MPI* uses only user and group IDs for starting up remote processes. Therefore, all processes will have identical user and group IDs on all nodes.

A homogeneous user ID space is stretched across the entire cluster.

5.11. Single system view

The *ParaStation MPI* administration tool collects and displays information from all or a selected subset of nodes in the cluster. Actions can be initiated on each node and will be automatically and transparently forwarded to the destination node(s), if necessary. From a management perspective, all the nodes are seen as a homogeneous system. Thus, the administrator will have a *single system view* of the cluster.

5.12. Parallel shell tool

ParaStation MPI provides a parallel shell tool called `psh`, which allows to run commands on all or selected nodes of the cluster in parallel. The output of the individual commands is presented in a sophisticated manner, showing common parts and differences.

`psh` may also be used to copy files to all nodes of the cluster in parallel.

This command is not intended to run interactive commands in parallel, but to run a single task in parallel on all or a bunch of nodes and prepare the output to be easily read by the user.

5.13. Nodes and CPUs

Care must be taken if the hardware is able to simulate virtual CPUs, e.g. Intel Xeon CPUs using Hyperthreading. The *ParaStation MPI* daemon detects virtual CPUs and uses all the virtual CPUs found for placing processes. Detecting virtual CPUs requires that the kernel module *cpuid* is loaded prior to starting the *ParaStation MPI* daemon. Use

```
# psiadmin -c "s hw"
Node     CPUs    Available Hardware
   0      4/ 2   ethernet p4sock
   1      4/ 2   ethernet p4sock
```

to show the number of virtual and physical CPUs per node.

It's possible to spawn more processes than physical or virtual CPUs are available on a node ("overbooking"). See *ParaStation MPI User's Guide* for details.

5.14. Integration with AFS

To run parallel tasks spawned by *ParaStation MPI* on clusters using `AFS`, *ParaStation MPI* provides the scripts `env2tok` and `tok2env`.

On the frontend side, calling

```
. tok2env
```

will create an environment variable `AFS_TOKEN` containing an encoded access token for AFS. This variable must be added to the list of exported variables

```
PSI_EXPORTS="AFS_TOKEN,$PSI_EXPORTS"
```

In addition, the variable

```
PSI_RARG_PRE_0=/some/path/env2tok
```

must be set. This will call the script `env2tok` before running the actual program on each node. `Env2tok` itself will decode the token and will setup the AFS environment.

The commands `SetToken` and `GetToken`, which are part of the AFS package, must be available on each node. Also, the commands `uuencode` and `uudecode` must be installed.

Script `tok2env`:

```
#!/bin/bash
tmp=$IFS
IFS=" "
export AFS_TOKEN=`GetToken | uuencode /dev/stdout`
IFS=$tmp
```

Script `env2tok`:

```
#!/bin/bash
IFS=" "
echo $AFS_TOKEN | uudecode | SetToken

exec $*
```

5.15. Integrating external queuing systems

ParaStation MPI can be easily integrated with batch queuing and scheduling systems. In this case, the queuing system will decide, where (and when) to run a parallel task. *ParaStation MPI* will then start, monitor and terminate the task. In case of higher prioritized jobs, the batch system may also suspend a task using the *ParaStation MPI* signal forwarding.

Integration is done by setting up *ParaStation MPI* environment variables, like `PSI_HOSTFILE`. *ParaStation MPI* itself need not be modified in any way. It is not necessary to use a remote shell (`rsh`) to start `mpirun` on the first node of the selected partition. The batch system should only run the command on the same node where the batch system is running, *ParaStation MPI* will start all necessary processes on the remote nodes. For details about spawning processes refer to *ParaStation MPI User's Guide*.

If an external queuing system is used, the environment variable `PSI_NODES_SORT` should be set to "none", thus no sorting of any predefined node list will be done by *ParaStation MPI*.

ParaStation MPI includes its own queuing facility. For more details, refer to Section 5.4, “Using the *ParaStation MPI* queuing facility” and *ParaStation MPI User's Guide*.

5.15.1. Integration with PBS PRO

Parallel jobs started by PBS PRO using the *ParaStation MPI* `mpirun` command will be automatically recognized. Due to the environment variable `PBS_NODEFILE`, defined by PBS PRO, *ParaStation MPI* will automatically setup the PSI_HOSTFILE to PBS_NODEFILE. The environment variable PSI_NODES_SORT is set to "none", thus no sorting of the predefined node list will occur. The tasks will be spawned in the given order on the predefined list of nodes.

Therefore, *ParaStation MPI* will use the (unsorted) hostfile supplied by PBS PRO to startup the parallel task.

5.15.2. Integration with OpenPBS

Refer to previous Section 5.15.1, “Integration with PBS PRO”.

5.15.3. Integration with Torque

Refer to previous Section 5.15.1, “Integration with PBS PRO”.

5.15.4. Integration with LSF

Similar to Section 5.15.1, “Integration with PBS PRO”, *ParaStation MPI* will also recognize the variable LSB_HOSTS, provided by LSF. This variable holds a list of nodes for the parallel task. It is copied to the *ParaStation MPI* variable PSI_HOSTS, consequently it will be used for starting up the task. The environment variable PSI_NODES_SORT is set to "none", thus no sorting of the predefined node list will occur. The tasks will be spawned in the given order on the predefined list of nodes.

5.15.5. Integration with LoadLeveler

ParaStation MPI recognizes the variable LOADL_PROCESSOR_LIST, provided by IBM LoadLeveler. This variable holds a list of nodes for the parallel task. It is copied to the *ParaStation MPI* variable PSI_HOSTS, consequently it will be used for starting up the task. The environment variable PSI_NODES_SORT is set to "none", thus no sorting of the predefined node list will occur. The tasks will be spawned in the given order on the predefined list of nodes.

5.16. Multicasts

This version of *ParaStation MPI* uses the *ParaStation MPI* RDP protocol to exchange status information between the psid(8) daemons. Therefore, multicast functionality is no longer required. It is still possible to use multicasts, if requested.

To enable Multicast message exchange, edit parastation.conf and uncomment the

```
 # UseMCast
```

statement.

If Multicast is enabled, the *ParaStation MPI* daemons exchange status information using multicast messages. Thus, a Linux kernel supporting multicast on all nodes of the cluster is required. This is usually no problem, since all standard kernels from all common distribution are compiled with multicast support. If a customized kernel is used, multicast support must be enabled within the kernel configuration! In order to learn more about multicast take a look at the *Multicast over TCP/IP HOWTO*.

In addition, the hardware also has to support multicast packets. Since all modern Ethernet switches support multicast and the nodes of a cluster typically live in a private subnet, this should be not a problem. If the cluster nodes are connected by a gateway, it has to be configured appropriately to allow multicast packets to reach all nodes of the cluster from all nodes.

Using a gateway in order to link parts of a cluster is not a recommended configuration.

On nodes with more than one Ethernet interface, typically frontend or head nodes, or systems where the default route does not point to the private cluster subnet, a proper route for the multicast traffic must be setup. This is done by the command

```
  route add -net 224.0.0.0 netmask 240.0.0.0 dev ethX
```

where *`ethX`* should be replaced by the actual name of the interface connecting to all other nodes. In order to enable this route at system startup, a corresponding entry has to be added to `/etc/route.conf` or `/etc/sysconfig/networks/routes`, depending on the type of Linux distribution in use.

5.17. Copying files in parallel

To copy large files to many or all nodes in a cluster at once, **pscp** is very handy. It overlaps storing data to disk and transferring data on the network, therefore it scales very well with respect to the number of nodes. Arbitrary size of files may be copied, even archives containing large lists of files may be created and unpacked on-the-fly.

Pscp uses the *ParaStation MPI* `pscom` library for data transfers, that automatically will use the most effective communication channel available. If required, the communication layer may be controlled using environment variables, refer to ps_environment(7) for details. The client process on each node is spawned using the *ParaStation MPI* process management.

As **pscp** uses administrative *ParaStation MPI* tasks to spawn the client processes, the user must be a member of the `adminuser` list or the user's group must be a member of the `admingroup` list. By default, only root is a member of the `adminuser` list and therefore allowed to use **pscp**. Refer to *ParaStation MPI User's Guide* and psiadmin(8) for details.

For more details refer to *ParaStation MPI User's Guide* and pscp(8).

5.18. Using *ParaStation MPI* accounting

ParaStation MPI may write accounting information about each finished job run on the cluster to `/var/account/yyyymmdd`, where `yyyymmdd` denotes the current accounting file in the form year, month and day.

To enable accounting, the special hardware *`accounter`* must be set within the *ParaStation MPI* configuration file for at least one node. On each configured node, an accounting daemon collecting all information for all jobs within the cluster will store the job information in the accounting file.

To list, sort and filter all the collected information, the command **psaccview** is available.

See psaccounter(8) and psaccview(8) for details.

5.19. Using *ParaStation MPI* process pinning

ParaStation MPI is able to pin down compute tasks to particular cores. This will avoid 'hoping' processes between different cores or CPUs during runtime, controlled by the OS scheduler.

While placing tasks to particular nodes, *ParaStation MPI* will also decide which CPU-slots (= virtual cores) on this node will be used. The physical core assigned to this CPU-slot will be calculated using the mapping defined in the configuration file or by the environment variable `__PSI_CPUMAP`, provided by the job environment.

Process pinning may be enabled or disabled globally or on a per node basis. Refer to **pinProcs**, **CPUmap** and **allowUserMap** entries in `parastation.conf` and the **set pinprocs**, **set cpumap** and **set allowusermap** directives of **psiadmin** for details.

Process pinning may be disabled for a particular job by defining the environment variable `__PSI_NO_PINPROC`. The value itself is thereby irrelevant.

See also parastation.conf(5) and psiadmin(1) for more information.

5.20. Using memory binding

Beside pinning down compute tasks to particular cores, *ParaStation MPI* is also able to use memory binding techniques on NUMA based systems. This will give hints to the memory management subsystem of the operating system to select 'nearest' memory, if available.

Memory binding may be enabled or disabled globally or on a per node basis. Refer to the **bindMem** entry in `parastation.conf` and **set bindmem** directive of **psiadmin** for details.

Memory binding may be disabled for a particular job by defining the environment variable `__PSI_NO_BINDMEM`. The value itself is thereby irrelevant.

See also parastation.conf(5) and psiadmin(1) for more information.

5.21. Spawning processes belonging to all groups

By default, newly created processes only belong to the primary group for the user ID as defined on the spawning node. To add a process to all groups a user belongs to on the current node, enable this flag using the **supplementaryGroups** directive or set the configuration flag **suplGrps** in `parastation.conf` to true.

Enabling this behavior may trigger extensive network traffic, depending on how the user authentication is configured on the nodes. E.g., using LDAP will open a connection to the LDAP server.

See also parastation.conf(5) and psiadmin(1) for more information.

5.22. Changing the default ports for psid(8)

By default, the *ParaStation MPI* daemon psid(8) uses the port `888` for TCP connections. To change this port, modify the files `/etc/services` and `/etc/xinet.d/psidstarter`.

Add the following line to `/etc/services`:

```
psid    888/tcp     # ParaStation Daemon Start Port
```

and change the default port number `888`.

Modify the entry

```
        port            = 888
```

within the file `/etc/xinet.d/psidstarter` to reflect the newly assigned port numbers.

In addition, the *ParaStation MPI* daemon psid(8) uses the UDP port `886` for RDP connections. To change this port, use the **RDPPort** directive within `parastation.conf`. See parastation.conf(5) for details.

The port numbers must be identical on all cluster nodes! Restart **xinetd** and **psid** on all nodes to activate the modifications.

5.23. Relocating packages during installation

The *ParaStation MPI* packages are installed by default in `/opt/parastation`. While installing the `psmpi2` packages, this path might be modified using the `--prefix` option of **rpm**.

```
  # rpm -U --prefix /new/path psmpi2-5.0.25-2.x86_64.rpm
  # rpm -U --prefix /new/path psmpi2-doc-5.0.25-2.x86_64.rpm
```

All other packages are currently not relocatable.

Chapter 6. Troubleshooting

This chapter provides some hints to common problems seen while installing or using *ParaStation MPI*. Of course, more help will be provided by <`support@par-tec.com`>.

6.1. Problem: psiadmin returns error

When starting up the *ParaStation MPI* admin command **psiadmin**, an error is reported:

```
# psiadmin
PSC: PSC_startDaemon: connect() fails: Connection refused
```

Reason: the local *ParaStation MPI* daemon could not be contacted. Verify that the psid(8) daemon is up and running. Check if the daemon is known to the xinetd:

```
# netstat -ant | grep 888
tcp        0      0 *:888                   *:*       LISTEN
```

If no "listening" socket is reported, check that the *ParaStation MPI* daemon is configured within the xinet(8) configuration. Check the file `/etc/xinet.d/psidstarter`.

If this is ok, reload xinetd:

```
# kill -HUP pid of xinetd
```

If everything seems to be ok up to now, check for recent entries within the log file`/var/log/messages`. Be aware, the log facility can be modified using the *`LogDestination`* within the config file `parastation.conf`. Look for lines like

```
Mar 24 17:19:12 pan psid[7361]: Starting ParaStation DAEMON
Mar 24 17:19:12 pan psid[7361]: Protocol Version 329
Mar 24 17:19:12 pan psid[7361]:  (c) Cluster Competence \
  Center GmbH
```

These lines indicate a normal startup of the psid. Other messages may indicate problems found by the psid, e.g. errors within the configuration file.

6.2. Problem: node shown as "down"

Maybe the node is currently not available (shutdown or crashed), or the network connection to this node is not available.

Try to `ping` this node. If ok, try to startup *ParaStation MPI*. From an other node, "add" this node:

```
psiadmin> add nodeid
```

Or logged on to this node, run **psiadmin** which also starts up the *ParaStation MPI* daemon `psid`. See Section 6.1, " Problem: **psiadmin** returns error " for more details.

Check the logfile `/var/log/messages` on this node for error messages. Verify that all nodes have an identical configuration (`/etc/parastation.conf`).

6.3. Problem: cannot start parallel task

Problem: a parallel task cannot be launched, an error is reported:

```
PSI: PSI_createPartition: Resource temporarily unavailable
```

Check for available nodes and active parallel tasks. Check for user or group restrictions.

If the error

```
PSI: dospawn: spawn to node 1 failed.
PSE: Could not spawn './mpi_latency' process 1, error = Bad \
  file descriptor.
```

is reported, check if the current directory holding the program **mpi_latency** is accessible on all nodes. Verify that the program is executable on all nodes.

6.4. Problem: bad performance

Verify that the proper interconnect and/or transport is used: check for environment variables controlling transport (see Section 5.8, “Controlling *ParaStation MPI* communication paths” and ps_environment(5)).

Watch protocol counters, e.g. counters indicating timeouts, retries, errors or other bad conditions. For p4sock, check `recv_net_data` and `recv_user`. See Section 5.2, “*ParaStation* protocol *p4sock*”.

Look for a crystal bowl!

Or contact <`support@par-tec.com`>.

6.5. Problem: different groups of nodes are seen as up or down

Problem: depending on which node the **psiadmin** is run, different groups of nodes are seen as "up" or "down".

Check for identical configuration on each node, e.g. compare the configuration file `/etc/parastation.conf` on each node.

6.6. Problem: cannot start process on frontend

Problem: Starting a job is canceled giving the error message

```
 Connecting client 139.27.166.22:44784 (rank 6) failed : Network is
unreachable
 PSIlogger: Child with rank 12 exited with status 1.
```

This typically happens, if the frontend or head node is included as compute node and also acts as gateway for the compute nodes. The "external" address of the frontend is not known to the compute nodes.

Use the `PSP_NETWORK` environment variable to re-direct all traffic to the cluster-internal network. See ps_environment(5) and Section 5.8, “Controlling *ParaStation MPI* communication paths” for details.

6.7. Warning issued on task startup

While starting up a parallel task, the message

```

execClient: chdir(/usr/tmp/username/./.): No such file or \
  directory
Will use user's home directory
---------------------------------------

```

is displayed.

The current directory `/usr/tmp/username` does not exist on one or more of the remote nodes. The user's home directory, defined by the environment variable `HOME` will be used instead.

Make sure that the directory `/usr/tmp/username` is accessible on each node or change your current directory to a globally accessible directory.

6.8. Problem: pssh fails

Problem: users other than root cannot run commands on remote nodes using the **pssh** command.

```
$ pssh -n 0 date
PSI: dospawn: spawn to node 0 failed: Permission denied
```

By default, only root may spawn processes which are not consuming CPUs. The command **pssh** uses this way to run a process on a remote node.

To allow other users to spawn this type of processes, add the user or group to the adminuser or admingroup list within *ParaStation MPI* using the command

```
psiadmin> set adminuser +username
psiadmin> set admingroup +group
```

or add appropriate `adminuser` and/or `admingroup` entries to the *ParaStation MPI* configuration file `parastation.conf`.

6.9. Problem: psid does not startup, reports port in use

Problem: the **psid** terminates after startup reporting that the port 886 is in use.

By default, the **psid** uses the port 886 (UDP) for the RDP protocol (inter-daemon communication). If this port is already in use, the daemon refuses to start-up and terminates immediately.

Make sure no other process uses this port. Or use the **RDPPort** directive within `parastation.conf` to re-define this port for all daemons within the cluster.

See also parastation.conf(5).

6.10. Problem: processes cannot access files on remote nodes

Problem: processes created by *ParaStation MPI* on remote nodes are not able to access files, if this files have enabled access only for a supplementary group the current user belongs to.

By default, only the primary group is set for newly created processes. To add all groups to a process, set the **supplGrps** flag within `parastation.conf` or use the **supplementaryGroups** directive within **psiadmin**.

See also Section 5.21, “Spawning processes belonging to all groups”.

6.11. Warning: PSI barrier timeout

The warning

```
 PSIlogger: Timeout: Not all clients joined the first pmi barrier:
joined=337 left=175 round=1
```

is output during job startup.

The PMI protocol just throws a warning. It just reports that after the first round of timeouts (which is 60 seconds plus 500 usec per process, i.e. in this case 60.256 sec) not all processes have joined the first barrier. Since there are no more warnings after that, the remaining 175 processes have joined during the second timeout-period. The slow startup might be due to network or file system problems.

See also ps_environment(5), `PMI_BARRIER_TMOUT` and `PMI_BARRIER_ROUNDS`.

Reference Pages

This appendix lists all reference pages related to *ParaStation MPI* administration tasks. For reference pages describing user related commands and information, refer to the *ParaStation MPI User's Guide*.

parastation.conf

parastation.conf — the *ParaStation MPI* configuration file

Description

Upon execution, the *ParaStation MPI* daemon psid(8) reads its configuration information from a configuration file which, by default, is `/etc/parastation.conf`. There are various parameters that can be modified persistently within this configuration file.

The main syntax of the configuration file is one parameter per line. Due to ease of use there are some parameters, e.g. **Nodes**, that are implemented in an environment mode. This mode enables the setting of multiple parameters by a single command. Environment mode parameters may comprise more than one line.

Line continuation is possible. If the last character within a line before the newline character is a "\", the newline character will be ignored and the next line is appended to the current line.

Comments are starting with a "#". All remaining characters on the line will be ignored. Keep in mind that line continuation also works within comments, i.e. if the last character of the line is a "\", the next line will be ignored, too.

The parser used to analyze `parastation.conf` is **not** case sensitive. This means, that all keywords within the configuration file may be written in any combination of upper- and lowercase characters. Within this document a mixed upper-/lowercase notation is used to provide more readable keywords. The same notation is used in the configuration file template `parastation.conf.tmpl` contained in the distributed *ParaStation MPI* system. The template file can be found in `/opt/parastation/config`.

Parameters

The different parameters are discussed in the order they should appear within the configuration file. Dependencies between parameters - resulting in a defined order of parameters - are marked explicitly.

Some parameters may be modified using different keywords, e.g. both **InstallDir** and **InstallationDir** modify the directory where the *ParaStation MPI* daemon psid(8) expects the *ParaStation MPI* system installed. In case of different keywords modifying the same resource, all keywords are mentioned in front of the parameter's discussion.

Only few parameters have to be declared in any case in order to enable *ParaStation MPI* to run on a cluster. These parameters are **HWType** and **Nodes**.

If parameters are declared more than once, the latest declaration is the one to use. Do not make use of this behavior as a feature since it may create great pitfalls.

InstallDir *`inst-dir`* **, InstallationDir** *`inst-dir`*

Tell the *ParaStation MPI* daemon to find all the *ParaStation MPI* related files in *`inst-dir`*. The default is `/opt/parastation`.

Hardware *name*

Tell the *ParaStation MPI* daemon how to handle a distinct hardware. Usually it is not necessary to edit these entries, since the template version of the configuration file contains up to date entries of all supported hardware types. Furthermore a deeper insight into the low-level functionality of *ParaStation MPI* is needed in order to create such an entry.

Nevertheless a brief overview on the structure of the **Hardware** entries is given here.

The following five types of parameters within the **Hardware** environment will get a special handling from the *ParaStation MPI* daemon psid(8). These define different script files called in order to execute various operations towards the corresponding communication hardware.

All these entries have the form of the parameter's name followed by the corresponding value. The value might be enclosed by single or double quotes in order to allow a space within.

The values are interpreted as absolute or relative paths. Relative paths will be looked up relative to *`InstallDir`*. If one or more of the scripts are not defined, no corresponding action will take place for this hardware.

startscript

Define a script called in order to startup the corresponding communication hardware. This script will be executed when the daemon starts up or after a reset of the communication hardware.

stopscript

Define a script called in order to shutdown the corresponding communication hardware. This script will be executed when the daemon exits or before a reset of the communication hardware.

setupscript

Define a script called in order to set special parameters on the corresponding communication hardware.

statusscript

Define a script called in order to get a status message from the corresponding communication hardware. This is mainly used in order to generate the lines shown be the **`status counter`** directive of the *ParaStation MPI* administration tool psiadmin(1).

headerscript

Define a script called in order to get a header line for the status message produced by the above discussed **statusscript** .

All further parameters defined within a **Hardware** section are interpreted as environment variables when calling the above defined scripts. Again these parameters have the form of the parameters name - interpreted as the environments variables name - followed by the corresponding value. The values might be single strings not containing whitespace characters or enclosed by single or double quotes, too.

The impact of the environment variables on the scripts of course depend on the scripts itself.

Various hardware types are defined within the template configuration file coming with the *ParaStation MPI* software distribution. These hardware types, the corresponding scripts and the environment variables the scripts understand are briefly discussed within the following lines.

Shared memory will be used as hardware type for communication within a SMP node. As there are no options for this kind of hardware, no dedicated section is provided.

ethernet

Use classical TCP/IP communication over Ethernet via an optimized MPI implementation.

Since TCP/IP has to be configured before *ParaStation MPI* starts up, the corresponding script **ps_ethernet** has almost nothing to do and hence does not understand a single environment variable.

p4sock

Use optimized communication via (Gigabit) Ethernet.

The script handling this hardware type **ps_p4sock** is also located in the `config` subdirectory. It understands the following two environment variables:

`PS_TCP`
: If set to an address range, e.g. 192.168.10.0-192.168.10.128, the TCP bypass feature of the **p4sock** protocol is enabled for the given address range.

openib

Use the OpenFabrics verbs layer for communication over InfiniBand.

No script is currently implemented for this communication protocol, therefore no environment variables are recognized.

mvapi

Use the Mellanox verbs layer for communication over InfiniBand.

No script is currently implemented for this communication protocol, therefore no environment variables are recognized.

gm

Use communication over GM (Myrinet).

The script **ps_gm** will load the Myrinet gm driver.

`PS_IPENABLED`
: If set to 1, the IP device myri0 is enabled after loading.

elan

Use communication over QsNet (libelan).

No script is currently implemented for this communication protocol, therefore no environment variables are recognized.

This communication layer is currently not supported by the *ParaStation MPI* communication library, therefore only programs linked with the QsNet MPI will work.

ipath

Use communication over InfiniPath.

No script is currently implemented for this communication protocol, therefore no environment variables are recognized.

This communication layer is currently not supported by the *ParaStation MPI* communication library, therefore only programs linked with the InfiniPath MPI will work.

dapl

Use communication over a generic DAPL layer.

No script is currently implemented for this communication protocol, therefore no environment variables are recognized.

accounter

This is actually a pseudo communication layer. It is only used for configuring nodes running the *ParaStation MPI* accounting daemon and should be used only in a particular **Nodes** entry.

NrOfNodes *`num`*

This configuration parameter is no longer required and will be silently ignored.

HWType { ethernet | p4sock | openib | mvapi | gm | elan | dapl | none }

HWType { { ethernet | p4sock | openib | mvapi | gm | elan | dapl | none }... **}**

Define the default communication hardware available on the nodes of the *ParaStation MPI* cluster. This may be overruled by an explicit `HWType` option in a **Node** statement.

The hardware types used within this command have to be defined in **Hardware** declarations before.

Further hardware declarations might be defined by the user, but this is pretty much undocumented.

It is possible to enable more than one hardware type, either as default or on a per node basis.

The default value of **HWType** is *`none`*.

starter { true | yes | 1 | false | no | 0 }

If the argument is one of *`yes`*, *`true`* or *`1`*, all nodes declared within a **Node** statement will allow to start parallel tasks, unless otherwise stated.

If the argument is one of *`no`*, *`false`* or *`0`*, starting will be not allowed.

It might be useful to prohibit the startup of parallel task from the frontend machine if a batch system is used. This will force all users to use the batch system in order to start their tasks. Otherwise it would be possible to circumvent the batch system by starting parallel task directly from the frontend machine.

The default is to allow the starting of parallel tasks from all nodes.

runJobs { true | yes | 1 | false | no | 0 }

If the argument is one of *`yes`*, *`true`* or *`1`*, all nodes declared within a **Node** statement will allow to run processes of parallel tasks, unless otherwise stated.

If the argument is one of *`no`*, *`false`* or *`0`*, *ParaStation MPI* will not start processes on these nodes.

It might be useful to prohibit the start of processes on a frontend machine since usually this machine is reserved for interactive work done by the users. If the execution of processes is forbidden on a distinct node, parallel tasks might be started from this node anyhow.

The default is to allow all nodes to run processes of parallel tasks.

Node[s] *`hostname id`* [HWType-entry] [starter-entry] [runJobs-entry] [env *`name value`*] [env { *`name value`* ... }]

Node[s] { {*`hostname id`* [HWType-entry] [starter-entry] [runJobs-entry] [env *`name value`*] [env { *`name value`* ... }] }... **}**

Node[s] $GENERATE *`from-to/step nodestr idstr`* [HWType-entry] [starter-entry] [runJobs-entry] [env *`name value`*] [env { *`name value`* ... }]

Define one or more nodes to be part of the *ParaStation MPI* cluster.

This is the first example of a parameter that supports the environment mode. This means there are two different notations to use this parameter. The first one may be used to define a single node, the second one will allow to register more than one node within a single command. It is a convenient form that prevents from typing the keyword once per entry again and again.

Each entry has to have at least two items, the *`hostname`* and the *`id`*. This will tell the *ParaStation MPI* system that the node called *`hostname`* will act as the physical node with *ParaStation MPI* ID *`id`*.

`hostname` is either a resolvable hostname or an IP address in dot notation (e.g. 192.168.1.17). *`Id`* is an integer number in the range from 0 to maximum number of nodes minus one.

Further optional items as `HWType-entry`, `starter-entry` or `runJobs-entry` may overrule the default values of the hardware type on the node, the ability to start parallel jobs from this node or the possibility to run processes on this node respectively. These entries have the same syntax as the stand alone commands to set the corresponding default value.

E.g. the line

```
Node node17 16 HWType { ethernet p4sock } starter yes runJobs no
```

will define the node *`node17`* to have the *ParaStation MPI* ID 16. Furthermore it is expected to have a Ethernet communication using both TCP and *p4sock* protocols. It is allowed to start parallel tasks from this node but the node itself will not run any process of any parallel task (except the *ParaStation MPI* logger processes of the tasks started on this node).

The option `environment` or `env` allows per node environment variables to be set. Using the first form, the variable *`name`* is set to *`value`*. More then one name/value pair may be given. More complex *`values`* may be given using quotation marks:

```
Node node17 16 environment LD_LIBRARY_PATH /mypath
Node node18 17 env { PSP_P4S "2" PSP_OPENIB "0" }
```

This example will define the variable *`LD_LIBRARY_PATH`* to `/mypath` for node *`node17`* and the variables *`PSP_P4S`* and *`PSP_OPENIB`* to 2 and `0` for node *`node18`*.

The `$GENERATE` allows to define a group of nodes at once using a simple syntax. Using the parameters *`from`* and *`to`*, a range may be defined, incremented by *`step`*. Each entry in this range may be referenced within the *`nodestr`* and *`idstr`* using a syntax of `$[{offset[,width[,base]]}]`. Eg., the entry

```
$GENERATE 1-96  node${0,2} ${0}
```

define the nodes *`node01`* up to *`node96`* using the id's 1 - 96, respectively. More node specific attributes may be defined as described above.

LicenseServer *`hostname`* **, LicServer** *`hostname`*

LicenseFile *`lic-file`* **, LicFile** *`lic-file`*

LicenseDeadInterval *`num`* **, LicDeadInterval** *`num`*

These entries are silently ignored by this version of *ParaStation MPI*.

SelectTime *`time`*

Set the timeout of the central select(2) of the *ParaStation MPI* daemon psid(8) to *`time`* seconds.

The default value is 2 seconds.

This parameter can be set during runtime via the **set `selecttime`** directive within the *ParaStation MPI* administration and management tool psiadmin(1).

DeadInterval *`num`*

The *ParaStation MPI* daemon psid(8) will declare other daemons as dead after *`num`* consecutively missing multicast pings.

After declaring a node as dead, all processes residing on this node are also declared dead. This results in sending signals to all processes on the local node that have requested to get informed about the death of one of these processes.

The default value is 10.

For now, the multicast period is set to two seconds, i.e. every daemon sends a multicast ping every two seconds. This results in declaring a daemon as dead after 20 seconds for the default value.

LogLevel *`num`*

Set the debugging level of the *ParaStation MPI* daemon psid(8) to `num`.

For values of `level` larger than `10` the daemon logs a huge amount of message in the logging destination, which is usually the syslog(3).

This parameter can be set during runtime via the **set `psiddebug`** directive within the *ParaStation MPI* administration and management tool psiadmin(1).

LogDest { LOG_DAEMON | LOG_KERN | LOG_LOCAL[0-7] }

LogDestination { LOG_DAEMON | LOG_KERN | LOG_LOCAL[0-7] }
Set the logging output's destination for the *ParaStation MPI* daemon psid(8). Usually the daemon prints logging output using the syslog(3) mechanism, unless an alternative logging file is requested via psid(8)'s `-l` option.

In order to collect all the *ParaStation MPI* specific log messages into a special file, the *`facility`* argument of the openlog(3) function call in cooperation with a suitable setup of the syslogd(8) may be used. This parameter will set the argument to one of the mentioned values.

The default value is **LOG_DAEMON**.

MCastGroup *`group-num`*

Tell psid(8) to use the multicast group *`group-num`* for multicast communication to other daemons.

The default group to use is `237`

MCastPort *`portno`*

Tell psid(8) to use the UDP port *`portno`* for multicast communication to other daemons.

The default port to use is `1889`

RDPPort *`portno`*

Tell psid(8) to use the UDP port *`portno`* for the RDP communication protocol to other daemons.

The default port to use is `886`.

RLimit { Core *`size`* | CPUTime *`time`* | DataSize *`size`* | MemLock *`size`* | StackSize *`size`* | RSSize *`size`* | NoFile *`num`* }

RLimit { { Core *`size`* | CPUTime *`time`* | DataSize *`size`* | MemLock *`size`* | StackSize *`size`* | RSSize *`size`* | NoFile *`num`* }... **}**
Set various resource limits to the psid(8) and thus to all processes started from it.

All limits are set using the setrlimit(2) system call. For a detailed description of the different types of limits please refer to the corresponding manual page.

If no RLimits are set within the *ParaStation MPI* configuration files, no changes are made to the systems default value.

The following (soft) resource limits may be set:

Core `size`

Set the maximum size of a core-file to `size` kilobytes. `size` is an integer number, the string “infinity” or the string “unlimited”. In the two latter cases the data size is set to RLIM_INFINITY.

Starting with version 5.0.3, this configuration will also control the writing of core-files for the **psid** itself, in case a catastrophic failure occurs.

CPUTime `time`

Set the maximum CPU time that might be consumed by the daemon to `time` seconds. `time` has to be an integer number, the string “infinity” or the string “unlimited”. In the two latter cases the data size is set to RLIM_INFINITY.

DataSize `size`

Set the maximum data size to `size` kilobytes. `size` is an integer number, the string “infinity” or the string “unlimited”. In the two latter cases the data size is set to RLIM_INFINITY.

MemLock `size`

Set the maximum amount of memory that might be locked into RAM to `size` kilobytes. `size` is an integer number, the string “infinity” or the string “unlimited”. In the two latter cases the data size is set to RLIM_INFINITY.

StackSize `size`

Set the maximum stack size to `size` kilobytes. `size` is an integer number, the string “infinity” or the string “unlimited”. In the two latter cases the stack is set to RLIM_INFINITY.

RSSize `size`

Set the maximum Resident Set Size (RSS) to `size` pages. `size` is an integer number, the string “infinity” or the string “unlimited”. In the two latter cases the RSS is set to RLIM_INFINITY.

NoFile `num`

Set the maximum number of open files to `num`. Be aware of the fact that inherited limits are confined by **psid's** hard limits.

Env | Environment `name value`

Env | Environment { {`name value` }... **}**

Set environment variables for the *ParaStation MPI* daemon psid(8) and any application started via this daemon.

This command again has two different modes. While within the first form exactly one variable is set, within the environment form of this command as many variables as wanted may be set. The general form of the latter case is one variable per line.

The `value` part of each line either is a single word or an expression enclosed by single or double quotes. The expression might contain whitespace characters. If the expression is enclosed by single quotes, it is allowed to use balanced or unbalanced double quotes within this expression and vice versa.

This command might be used for example in order to set the `PSP_NETWORK` environment variable globally without the need of every user to adjust this parameter in his own environment.

freeOnSuspend { true | yes | 1 | false | no | 0 }

If the argument is one of *`yes`*, *`true`* or *`1`*, suspending a task by sending the signal `SIGTSTP` to the logger will handle all resources (CPUs) currently claimed by this task as free.

If the argument is one of *`no`*, *`false`* or *`0`*, *ParaStation MPI* will not claim resources as free after sending `SIGTSTP`.

handleOldBins { true | yes | 1 | false | no | 0 }

If the argument is one of *`yes`*, *`true`* or *`1`*, compatibility mode for applications linked with *ParaStation MPI* version 4.0 up to 4.0.6 will be enabled. Keep in mind that this behavior might collide with the **freeOnSuspend** feature.

If the argument is one of *`no`*, *`false`* or *`0`*, *ParaStation MPI* will disable compatibility mode.

UseMCast { true | yes | 1 | false | no | 0 }

If the argument is one of *`yes`*, *`true`* or *`1`*, keep alive messages from the *ParaStation MPI* daemon psid(8) are sent using Multicast messages.

If the argument is one of *`no`*, *`false`* or *`0`*, *ParaStation MPI* will use it's own RDP protocol for keep alive messages. This is the default.

PSINodesSort { PROC | LOAD_1 | LOAD_5 | LOAD_15 | PROC+LOAD | NONE }

Define the default sorting strategy for nodes when attaching them to a partition. The different possible values have the following meaning:

PROC

Sort by the number of processes managed by *ParaStation MPI* on the corresponding nodes

LOAD_1

Sort by the load average during the last minute on the corresponding nodes

LOAD_5

Sort by the load average during the last 5 minutes on the corresponding nodes

LOAD_15

Sort by the load average during the last 15 minutes on the corresponding nodes

PROC+LOAD

Sort conforming to the sum of the processes managed by *ParaStation MPI* and the load average during the last minute on the corresponding nodes

NONE

Do not sort at all.

This only comes into play, if the user does not define a sorting strategy explicitly via `PSI_NODES_SORT`. Be aware of the fact that using a batch-system like PBS or LSF *will* set the strategy explicitly, namely to NONE.

overbook { true | yes | 1 | false | no | 0 }

If the argument is one of *`yes`*, *`true`* or *`1`*, all nodes may be overbooked by the user using the `PSI_OVERBOOK` environment variable.

If the argument is one of *`no`*, *`false`* or *`0`*, *ParaStation MPI* will deny overbooking of the nodes, even if `PSI_OVERBOOK` is set.

It might be useful to prohibit the start of processes on a frontend machine since usually this machine is reserved for interactive work done by the users. When the execution of processes is forbidden on a distinct node, parallel task might be started from this node anyhow.

The default is to allow all nodes to run processes of parallel tasks.

processes *`maxprocs`*

Define the maximum number of processes per node.

This parameter can be set during runtime via the **set `maxproc`** directive within the *ParaStation MPI* administration and management tool psiadmin(1).

pinProcs { true | yes | 1 | false | no | 0 }

Enables or disables process pinning for compute tasks. If enabled, tasks will be pinned down to particular CPU-slots. The mapping between those CPU-slots and physical CPUs and cores is made using a mapping list. See **CPUmap** below.

The **pinProcs** parameter can be set during runtime via the **set `pinprocs`** directive within the *ParaStation MPI* administration and management tool psiadmin(1).

bindMem { true | yes | 1 | false | no | 0 }

This parameter must be set to true if nodes providing non-Uniform memory access (NUMA) should use 'local' memory for the tasks.

This parameter can be set during runtime via the **set `bindmem`** directive within the *ParaStation MPI* administration and management tool psiadmin(1).

CPUmap { map }

Set the map used to assign CPU-slots to physical cores to *`map`*. `Map` is a quoted string containing a space-separated permutation of the number 0 to *`Ncore`*-1. Here *`Ncore`* is the number of physical cores available on this node. The number of cores within a distinct node may be determined via **list hw**. The first number in *`map`* is the number of the physical core the first CPU-slot will be mapped to, and so on.

This parameter can be set during runtime via the **set `bindmem`** directive within the *ParaStation MPI* administration and management tool psiadmin(1).

supplGrps { true | yes | 1 | false | no | 0 }

This parameter must be set to true if processes spawned by *ParaStation MPI* should belong to all groups defined for this user. Otherwise, they will only belong to the primary group.

This parameter can be set during runtime via the **set `supplementaryGroups`** directive within the *ParaStation MPI* administration and management tool psiadmin(1).

rdpMaxRetrans number

Set the maximum number of retransmissions within the RDP facility. If more than this number of retransmission would have been necessary to deliver the packet to the remote destination, this connection is declared to be down.

See also psiadmin(1).

statusBroadcasts number

Set the maximum number of status broadcasts per round. This is used to limit the number of status-broadcasts per status-iteration. Too many broadcast might lead to running out of message-buffers within RDP on huge clusters.

If more than this number of broadcasts are triggered during one status-iteration, all future broadcasts will be ignored. The corresponding counter is reset upon start of the next status iteration.

A value of 0 will completely suppress sending of status-broadcasts. In this case information on dead nodes will be propagated by sending ACTIVENODES messages upon receive of too many wrong LOAD messages, only.

Only relevant, if MCast is *not* used.

See also psiadmin(1).

rdpTimeout ms

The timeout of the actual timer registered by RDP in milliseconds. Each time the corresponding timer is elapsed, handleTimeoutRDP() is called handling all resend activities necessary. This parameter steers the actual load introduced by RDP. Within the daemon, there is a lower limit for all timeout-timers of 100 msec. Thus, the minimal value here is 100, too.

deadLimit number

Dead-limit of the RDP status module. After this number of consecutively missing RDP-pings the master declares the node to be dead.

Only relevant, if MCast is *not* used.

statusTimeout ms

Timeout of the RDP status module. After this number of milliseconds a RDP-ping is sent to the master daemon. Additionally, the master daemon checks for received ping-messages. Within the daemon, there is a lower limit for all timeout-timers of 100 msec. Thus, the minimal value here is 100, too.

Only relevant, if MCast is *not* used.

rdpClosedTimeout ms

The closed timeout within the RDP facility in milliseconds. If a RDP-connection is closed, during this timeout all messages from the corresponding partner are ignored. Thus, reconnection is avoided during this period. This helps handling packets still on the wire on connection close.

rdpResendTimeout ms

The resend timeout within the RDP facility in milliseconds. If a pending message is available and not yet acknowledged, this is the timeout after which the message is retransmitted to the remote host.

rdpMaxACKPend number

The maximum number of pending ACKs within the RDP facility. If this number of packets is received from a remote node consecutively without any retransmission, an explicit ACK is sent. Otherwise the ACK is sent piggyback within the next regular packet to this node or as soon as a retransmission occurred.

If set to 1, each RDP packet received is acknowledged by an explicit ACK.

pinProcs { true | yes | 1 | false | no | 0 }

Enable pinning of processes to distinct processor cores.

bindMem { true | yes | 1 | false | no | 0 }

Enable binding of processes to distinct memory nodes on NUMA systems.

CPUmap { list of ids }

Define map assigning logical process-slots to physical processor cores.

allowUserMap { true | yes | 1 | false | no | 0 }

Enable users to influence local mapping of processes via providing a `__PSI_CPUMAP` environment variable.

startupScript script

This script is called during startup of the ParaStation daemon. It's either an absolute path or relative to *`InstallDir`*. Parsing the configuration file will fail, if the script is not found. Depending on its return value the daemon continues startup without any action (0), some output of the script is written to the daemons log (-1) or the daemon stops immediately (-2).

As a default no script is defined and, thus, nothing is called.

maxStatTry num

Maximum number of tries to stat() an executable before spawning new processes. Increasing this number might help on overloaded NFS-servers.

RDPstatistics { true | yes | 1 | false | no | 0 }

Flag the RDP statistics. If set to 1, statistics on total number of messages sent and mean-time to ACK are determined per connection.

nodeUpScript script

This script is called by the currently elected master of the *ParaStation* daemons every time a node becomes active from *ParaStation*'s point of view. I.e. the node's daemon connects to the master daemon for the first time after being down.

This might be used to pass this type of information into a batch-system or some monitoring facility like the GridMonitor.

While calling the script two additional arguments are passed within the environment:

NODE_NAME
: The hostname of the node that appeared. The actual value is the IP address implicitly defined in the Nodes-section of this file resolved to the official name of the host using `gethostbyaddr()`.

NODE_ID
    The *ParaStation MPI* ID of the node that appeared.

As a default no script is defined and, thus, nothing is called.

nodeDownScript script

This script is called by the currently elected master of the *ParaStation* daemons every time a node goes down from ParaStation's point of view. I.e. the node's daemon disconnects from the master daemon after being connected.

This might be used to pass this type of information into a batch-system or some monitoring facility like the GridMonitor.

While calling the script two additional arguments are passed within the environment:

NODE_NAME
    The hostname of the node that appeared. The actual value is the IP address implicitly defined in the Nodes-section of this file resolved to the official name of the host using `gethostbyaddr()`.

NODE_ID
    The *ParaStation MPI* ID of the node that appeared.

As a default no script is defined and, thus, nothing is called.

Errors

No known errors.

See also

psid(8), psiadmin(1)

psiadmin

psiadmin — the *ParaStation MPI* administration and management tool

Synopsis

psiadmin [-denqrsv?] [-c `command`] [-f `program-file`] [--usage]

Description

The psiadmin command provides an administrator interface to the *ParaStation MPI* system.

The command reads directives from standard input in interactive mode. The syntax of each directive is checked and the appropriate request is sent to the local *ParaStation MPI* daemon psid(8).

In order to send psiadmin into batch mode, either use the `-c` or the `-f`. The syntax of the directives is exactly the same as in interactive mode for both options.

Most of the directives listed below can be executed by general users. Only modifying parameters, killing foreign jobs and shutting down single nodes or the whole system requires root privilege.

Options

`-c , --command=command`
: Execute the single directive **command** and exit.

`-d`
: Do not automatically start up the local psid(8).

`-e , --echo`
: Echo each executed directive to stdout.

`-f , --file=program-file`
: Read commands from the file `program-file`. Exit as soon as EOF is reached.

  It might be useful to enable echoing (`-e`) when acting on a script file.

  This option silently enables the `-q` option suppressing the prompt.

`-n , --noinit`
: Ignore the initialization file `.psiadminrc`.

`-q , --quiet`
: Suppress printing the prompt each time waiting for a new command. This is useful in combination with the `-f` option.

`-s , --start-all`
: Try to start all daemons within the cluster. This option is equivalent to the execution of the **add** directive straight after the startup of the administration tool.

`-r , --reset`
: Do a reset of the *ParaStation MPI* system on startup.

`-v , --version`
: Output version information and exit.

`-? , --help`
: Show a help message.

`--usage`
: Display a brief usage message.

Standard Input

The psiadmin command reads standard input for directives until end of file is reached, or the **exit** or **quit** directive is read.

Standard Output

If Standard Output is connected to a terminal, a command prompt will be written to standard output when psiadmin is ready to read a directive.

If the `-e` option is specified, psiadmin will echo the directives read from standard input to standard output.

Standard Error

The psiadmin command will write a diagnostic message to standard error for each error occurred.

Extended description

If psiadmin is invoked without the `-c` or `-f` option and standard output is connected to a terminal, psiadmin will repeatedly write a prompt to standard output and read a directive from standard input.

Directives can be abbreviated to their minimum unambiguous form. A directive is terminated by a new line character or a semicolon. Multiple directives may be entered on a single line. A directive may extend across lines by escaping the new line character with a back-slash "\".

Comments begin with the # character and continue to end of the line. Comments and blank lines are ignored by psiadmin.

Upon startup psiadmin tries to find the file `.psiadminrc` first in the current directory and then in the user's home directory. Only the first one found is really considered. Each directive found within this file is handled silently before going either into interactive or batch mode (using the `-f` flag).

Interactive directives

Whenever the psiadmin is started into interactive mode, it will prompt for directives unless the `-q` flag is used. The same directives are accepted in batch mode, too. Directives may be abbreviated as long as they are unique. They can be expanded using the TAB-key, analogous to some shell tab expansion features. A command history is stored in ~/.psiadm_history. See readline(3) for more information on command expansion and command history.

Almost all directives accept an optional parameter *`nodes`*. This contains either a comma-separated list of node ranges to act on, each of the form *`from`*[-*`to`*]. If the *`to`* part is missing, the single node *`from`* is represented by this range. In principle *`nodes`* might contain an unlimited number of ranges.

Otherwise the value of *`nodes`* might be `all`. Then all nodes of the *ParaStation MPI* cluster are selected within this directive.

If *`nodes`* is empty, the node range preselected via the **range** command is used. The default preselected node range contains all nodes of the ParaStation cluster.

The *`from`* and *`to`* parts of each range are node IDs. They might be given in decimal or hexadecimal notation and must be in the range between `0` and `NumberOfNodes-1`.

As an extension *`nodes`* might also be a hostname that can be resolved into a valid *ParaStation MPI* ID.

Using hostnames containing "-" might confuse this algorithm and is therefore not recommended.

exit

Exit the interactive mode of psiadmin. Same as **quit**.

help [*`directive`*]

Print a help message concerning *`directive`*. If *`directive`* is missing, a general help message is displayed.

kill [-*`sig`*] *`tid`*

Send the process with the task ID `tid` the signal *`sig`*.

`sig` has to be a positive number representing a UNIX signal. For a list of available signals please consult the signal(7) manual page. If *`sig`* is not given, a SIGTERM signal (i.e. 15) is sent to the corresponding process.

Processes can only be signaled by their owner or by root.

list [all | allproc [cnt *`count`*] | count [hw *`hw`*] | down | hardware | load | mcast | memory | node | proc [cnt *`count`*] | rdp | summary [max *`max`*] | up | version | startupscript | starttime | environment | rdpconnection | nodeupscript | nodedownscript] [*`nodes`*]

list jobs [state running | state pending | state suspended] slots [*`tid`*]

Report various states of the selected node(s) or job(s). Depending on the given argument, different information can be requested from the *ParaStation MPI* system. If no argument is given, the `node` information is retrieved.

all

Show the information given by **node**, **count** and **proc** on the selected node(s).

allproc [cnt *`count`*]

Show all processes managed by the *ParaStation MPI* system on the selected node(s).

All processes - including forwarder and other special processes - managed by *ParaStation MPI* are displayed. If forwarder processes should not be displayed, use the **list `proc`** directive.

Up to *`count`* processes per node are displayed. If more processes are controlled by *ParaStation MPI* on this node, a short remark will tell the number of not displayed processes. The default is to show 10 processes.

The output fields of the process list are described within the **list `proc`** directive. In addition to the process classes described there, *ParaStation MPI* Forwarder processes, i.e. processes spawned by the *ParaStation MPI* daemon psid(8) in order to control a spawned process, are marked by “(F)” after the user ID. Further helper processes needed in order to spawn non *ParaStation MPI* applications are marked with “(S)”.

count [hw *`hw`*]

List the status of the communication system(s) on the selected node(s). Various counters are displayed.

If the `hw` option is given, only the counters concerning the *`hw`* hardware type are displayed. The default is to display the counters of all enabled hardware types on this node.

down

List all nodes which are marked as "DOWN".

hardware

Show the hardware setup on the selected node(s).

Besides the types of the communication hardware enabled within the *ParaStation MPI* system on each node also the number of available CPUs are displayed. The two numbers shown in this column mark the number of virtual and physical CPUs respectively. These number might differ due to technologies like Intel's HyperThreading or multi core CPUs.

load

Show the load and the number of processes managed by the *ParaStation MPI* system on the selected node(s).

The three load values displayed are the averages for 1, 5 and 15 minutes respectively. The two numbers of processes are as follows: The total number of processes contains all processes managed by the *ParaStation MPI* system, including Logger, Forwarder and psiadmin(1) processes. Furthermore of course the actual working processes started by the users are included. The latter ones are the “normal” processes, additionally displayed in the last column of the output.

mcast

List the status of the MCast facility of the *ParaStation MPI* daemon psid(8) on the selected node(s).

memory

Show the overall and available memory on the selected node(s).

node

List the status of the selected node(s). Depending on the state of the *ParaStation MPI* daemons, the node(s) are marked to be "UP" or "DOWN".

proc [cnt *`count`*]

Show the processes managed by the *ParaStation MPI* system on the selected node(s).

Only user, logger and admin processes are displayed. If forwarder and other special processes should also be displayed, use the **list `allproc`** directive.

Up to *`count`* processes per node are displayed. If more processes are controlled by *ParaStation MPI* on this node, a short remark will tell the number of not displayed processes. The default is to show 10 processes.

The listed fields have the following meaning:

Node
: The *ParaStation MPI* ID of the node the process is running on.

TaskID
: The *ParaStation MPI* task ID of the process, both as decimal and hexadecimal number. The task ID of a process is unique within the cluster and is composed out of the *ParaStation MPI* ID of the node the process is running on and the local process ID of the process, i.e. the result of calling getpid(2).

ParentTaskID
: The *ParaStation MPI* task ID of the parent process. The parent process is the one which has spawned the current process. If the process was not spawned by any other controlled by *ParaStation MPI*, i.e. it is the first process started within a parallel task, the parent *ParaStation MPI* task ID is 0.

Con

Flag to mark if the process has reconnected to its local *ParaStation MPI* daemon psid(8). If a 1 is displayed, the process has connected the daemon, otherwise 0 is reported.

UserID

The user ID under which the process runs. This is usually identical to the user ID of the parent process.

Furthermore administrative processes, i.e. psiadmin(1) processes connected to a local daemon are marked with “(A)” after the user ID.

Logger processes, i.e. root processes of parallel tasks which converted to a *ParaStation MPI* Logger process, are tagged with “(L)” after the user ID.

System processes, which are not counted, are marked as “(*)”. Accounting processes are indicated by “(C)”. Other helper processes are marked with “(S)”.

jobs [state running | state pending | state suspended] [slots] [*`tid`*]

Show all or selected jobs managed by the *ParaStation MPI* system.

If selected, only jobs with state running, pending or suspended are shown. if slots is provided, node and CPU count information for this job is printed, too. If *`tid`* is given, information for this particular job is shown.

For each job, information about the RootTaskId, state (= 'R', 'P' or 'S'), size (= number of CPUs), UID, GID, target slots and start time is printed.

rdp

List the status of the RDP protocol of the *ParaStation MPI* daemon psid(8) on the selected node(s).

This directive now displays for each connection the state, the partner's IP-address, total number of frames sent, number of pending ACKs, number of pending frames, number of retransmissions of the frame in flight and total number of retransmission during connection. If collecting the RDP statistics is enabled, the mean-time to ACK is displayed, too.

summary [max *`max`*]

Print a brief summary of the active and down nodes. Thus the number of up and down nodes will be printed out in one line. If any node is down and the number of down nodes is less than 20 or *`max`*, then the node IDs of this nodes will also printed out in a second line.

up

List all nodes which are marked as "UP".

version

List the ID, psid revision and RPM version for the selected node(s).

startupscript

Show the daemon's script called during startup in order to test the local situation on the selected nodes.

starttime

List the startup time of the *ParaStation* daemons currently running.

environment key *env*

List the daemons environment variable *env* and its value on the selected nodes. If *key env* is omitted, the entire environment will be displayed.

rdpconnection

Show info on RDP connections on the selected nodes.

nodeupscript

Show the daemon's script called on the master node whenever a daemon connects after being down before on the selected nodes.

nodedownscript

Show the daemon's script called on the master node whenever a daemon disconnects on the selected nodes.

quit

Exit the interactive mode of psiadmin. Same as **exit**.

range {[n1-n10] | [n1,n2,..] | all }

Preselect or display the default set of nodes

The **range** command is used to preselect a range of nodes. Any subsequent **PSIAdmin** commands issued after the **range** command will only be applied to those nodes defined in the range. Ranges can be specified as contiguous sets `n1-n2` or as individual comma separated hosts `n1,n2`. Mixes of continuous sets and individual hosts are permitted as shown in the following example:

```
PSIAdmin> range n1-n3,n5,n7-n10
```

show { maxproc | user | group | psiddebug | selecttime | statustimeout | statusbroadcasts | deadlimit | rdpdebug | rdptimeout | rdppktloss | rdpmaxretrans | rdpresendtimeout | rdpretrans | rdpclosedtimeout | rdpmaxackpend | rdpstatistics | mcastdebug | master | freeonsuspend | fos | handleoldbins | hob | starter | runjobs | overbook | exclusive | pinprocs | bindmem | cpumap | allowUserMap | nodessort | supplementaryGroups | maxStatTry | adminuser | admingroup | accounters | accountpoll | rl_addressspace | rl_as | rl_core | rl_cpu | rl_data | rl_fsize | rl_locks | rl_memlock | rl_msgqueue | rl_nofile | rl_nproc | rl_rss | rl_sigpending | rl_stack } [*nodes*]

Show various parameters of the *ParaStation MPI* system.

accounters [*nodes*]

Show information on which node(s) *ParaStation MPI* accounting processes are running.

user [*nodes*]

Show who grants exclusive access on the selected node(s).

group [*nodes*]

Show which group grants exclusive access on the selected node(s).

maxproc [*nodes*]

Show the maximum number of *ParaStation MPI* processes on the selected node(s).

selecttime [*nodes*]

Show the timeout of the central select(2) of the *ParaStation MPI* daemon psid(8) on the selected node(s).

psiddebug [*nodes*]

Show the debugging mask of the *ParaStation MPI* daemon psid(8) on the selected node(s).

rdpdebug [*nodes*]

Show the debugging mask of the RDP protocol within the *ParaStation MPI* daemon psid(8) on the selected node(s).

rdpretrans [*nodes*]

Show the RDP retransmit counters off the selected node(s).

mcastdebug [*nodes*]

Show the debugging mask of the MCast protocol within the *ParaStation MPI* daemon psid(8) on the selected node(s).

master [*nodes*]

Show the current master on the selected node(s).

The master node's task is the management and allocation of resources within the cluster. It is elected among the running nodes during runtime. Thus usually all nodes should give the same answer to this question. In rare cases - usually during startup or immediately after a node failure - the nodes might disagree on the elected master node. This command helps on identifying these rare cases.

freeOnSuspend [*nodes*]

Show the `freeOnSuspend` flag on the selected nodes.

The `freeOnSuspend` flag steers the behavior of the resource management concerning suspended jobs. Basically there are two possible approaches: Either the resources used by the suspended job are freed for other jobs (this is done, if the flag is set to 1) or they are kept occupied in order to preserve them exclusively for the time the job continues to run (this is the behavior as long as the flag has the value 0).

Since the master node does all the resource management within the cluster, only the value on this node actually steers the behavior.

handleOldBins [*nodes*]

Show the compatibility flag for applications linked against version 4.0.x of *ParaStation MPI* on the selected nodes.

nodesSort [*nodes*]

Show the default sorting strategy used when attaching nodes to partitions.

Since the master node does all the resource management within the cluster, only the value on this node actually steers the behavior.

starter [*nodes*]

Show if the selected node(s) are allowed to start parallel tasks.

runjobs [*nodes*]

Show if the selected node(s) are allowed to run tasks.

overbook [*nodes*]

Show if the selected node(s) are allowed to be overbooked on user request.

rdppktloss [*nodes*]

Show RDP protocol's packet-loss rate.

rdpmaxretrans [*nodes*]

Show RDP protocol's maximum retransmission count.

exclusive [*nodes*]

Show flag marking if this nodes can be requested by users exclusively.

pinprocs [*nodes*]

Show flag marking if this nodes uses process pinning.

cpumap [*nodes*]

Show the CPU-slot to core mapping list for the selected nodes.

bindmem [*nodes*]

Show flag marking if this nodes uses binding as NUMA policy.

adminuser [*nodes*]

Show users allowed to start admin-tasks, i.e. unaccounted tasks.

admingroup [*nodes*]

Show groups allowed to start admin-tasks, i.e. unaccounted tasks.

rl_addressspace [*nodes*]

Show RLIMIT_AS on this node.

rl_core [*nodes*]

Show RLIMIT_CORE on this node.

rl_cpu [*nodes*]

Show RLIMIT_CPU on this node.

rl_data [*nodes*]

Show RLIMIT_DATA on this node.

rl_fsize [*nodes*]

Show RLIMIT_FSIZE on this node.

rl_locks [*nodes*]

Show RLIMIT_LOCKS on this node.

rl_memlock [*nodes*]

Show RLIMIT_MEMLOCK on this node.

rl_msgqueue [*nodes*]

Show RLIMIT_MSGQUEUE on this node.

rl_nofile [*nodes*]

Show RLIMIT_NOFILE on this node.

rl_nproc [*nodes*]

Show RLIMIT_NPROC on this node.

rl_rss [*nodes*]

Show RLIMIT_RSS on this node.

rl_sigpending [*nodes*]

Show RLIMIT_SIGPENDING on this node.

rl_stack [*nodes*]

Show RLIMIT_STACK on this node.

supplementaryGroups [*nodes*]

Show supplementaryGroups flag.

statusBroadcasts [*nodes*]

Show the maximum number of status broadcasts initiated by lost connections to other daemon.

rdpTimeout [*nodes*]

Show the RDP timeout configured in ms.

deadLimit [*nodes*]

Show the dead-limit of the RDP status module. See also parastation.conf(5).

statusTimeout [*nodes*]

Show the timeout of the RDP status module. See also parastation.conf(5).

rdpClosedTimeout [*nodes*]

Show the closed timeout within the RDP facility in milliseconds. See also parastation.conf(5).

rdpResendTimeout [*nodes*]

Show the resend timeout within the RDP facility in milliseconds. See also parastation.conf(5).

rdpMaxACKPend [*nodes*]

Show the maximum ACK pending counter within the RDP facility. See also parastation.conf(5).

rdpStatistics [*nodes*]

Show if RDP statistics are currently collected.

allowUserMap [*nodes*]

Show flag marking if this nodes will allow user to influence the mapping of processes to physical core.

maxStatTry [*nodes*]

Show the maximum number of tries to stat() an executable while spawning new processes.

accountPoll [*nodes*]

Show polling interval in seconds of accounter to retrieve more detailed information.

sleep [*sec*]

Sleep for *sec* seconds before continuing to parse the input.

version

Print various version numbers.

environment { list } [*nodes*]

Manage the *ParaStation* daemon environment.

list [key *env*] [*nodes*]

List the daemons environment variable *env* and its value on the selected nodes. If the option *key env* is omitted, the entire environment will be displayed.

echo *string*

Echo the given *string* to stdout. This command does not support control sequences like its counterpart **/bin/echo**.

Privileged directives

Some directives are only available for privileged users, i.e. only root can execute these directives.

add [*nodes*]

Start the *ParaStation MPI* daemon psid(8) on the selected node(s).

add only tries to start the *ParaStation MPI* daemon on the selected node(s). If it is not possible to start the daemon, no error message occurs. The current status of the nodes can be checked using the **list** directive.

hwstart [hw { *hw* | all }] [*nodes*]

Start the declared hardware on the selected nodes.

Starting a specific hardware will be tried on the selected nodes regardless, if this hardware is specified for this nodes within the `parastation.conf` configuration file or not. On the other hand, if `hw all` is specified or the `hw` option is missing at all, only the hardware types specified within the configuration file are started.

Starting or stopping a specific communication hardware only concerns the *ParaStation MPI* part of hardware handling. I.e. stopping `ethernet` hardware should not touch the normal IP traffic running over this specific device.

hwstop [hw { *hw* | all }] [*nodes*]

Stop the declared hardware on the selected nodes.

If `hw all` is specified or the `hw` option is missing at all, all running hardware for this node is stopped.

Starting or stopping a specific communication hardware only concerns the *ParaStation MPI* part of hardware handling. I.e. stopping `ethernet` hardware should not touch the normal IP traffic running over this specific device.

resolve [*nodes*]

Resolves a list of IDs to node names. *Nodes* selects one or more ranges of nodes. *Nodes* is either of the form s1[-e1]{,si[-ei]}*, where the s and e are positiv numbers representing *ParaStation MPI* IDs, or 'all'. Each comma-separated part of *nodes* denotes a range of nodes. If a range's '-e' part is missing, it represents a single node. In principle *nodes* might contain an unlimited number of ranges. If *nodes* value is 'all', all nodes of the ParaStation cluster are selected. If *nodes* is empty, the node range preselected via the 'range' command is used. The default preselected node range contains all nodes of the *ParaStation MPI* cluster.

As an extension *nodes* might also be a hostname that can be resolved into a valid *ParaStation MPI* ID.

reset [hw] [*nodes*]

Reset the *ParaStation MPI* daemon on all selected node(s). As a consequence **all** processes using the selected node(s) are **killed**!

If the option `hw` is given, additionally the communication hardware is brought into a known state. Executing **reset hw** is the same as using **restart**.

restart [*nodes*]

Restart the *ParaStation MPI* system on all selected node(s). This includes re-initialization of the communication hardware. On the selected node(s) the *ParaStation MPI* daemon processes are forced to reinitialize the *ParaStation MPI* cluster. As a consequence **all** processes using the selected node(s)s are **killed**!

This is the same as using **reset hw**.

set { maxproc { *num* | any } | user [+ | -] { *name* | any } | group [+ | -] { *name* | any } | psiddebug *mask* | master *id* | selecttime *time* | statusTimeout *ms* | statusBroadcasts *num* | deadLimit *num* | rdpdebug *mask* | rdpTimeout *ms* | rdpmaxretrans *val* | rdpResendTimeout *ms* | rdpRetrans *count* | rdpClosedTimeout *ms* | rdpMaxACKPend *num* | rdpStatistics *bool* | mcastdebug *mask* | freeOnSuspend { 0 | 1 } | handleOldBins { 0 | 1 } | starter { 0 | 1 } | runjobs { 0 | 1 } | overbook { 0 | 1 } | exclusive *bool* | pinprocs *bool* | bindmem *bool* | supplementaryGroups *bool* | maxStatTry *num* | cpumap *map* | allowUserMap *bool* | nodesSort { PROC | LOAD_1 | LOAD_5 | LOAD_15 | PROC+LOAD | NONE } | adminuser [+ | -] { *name* | any } | admingroup [+ | -] { *name* | any } } [*nodes*]

Modify various parameters of the *ParaStation MPI* system.

adminuser [+ | -] { *name* | any } [*nodes*]

Grant authorization to start admin-tasks, i.e. task not blocking a dedicated CPU, to a particular or any user. *Name* might be a user name or a numerical UID. If *name* is preceded by a '+' or '-', this user is added to or removed from the list of adminusers respectively.

admingroup [+ | -] { *name* | any } [*nodes*]

Grant authorization to start admin-tasks, i.e. task not blocking a dedicated CPU, to a particular or any group. *Name* might be a group name or a numerical GID. If *name* is preceded by a '+' or '-', this group is added to or removed from the list of admingroups respectively.

user [+ | -] { *name* | any } [*nodes*]

Grant exclusive access on the selected node(s) to the special user `name` or to any user. If *name* is preceded by a '+' or '-', this user is added to or removed from the list of users respectively.

group [+ | -] { *name* | any } [*nodes*]

Grant exclusive access on the selected node(s) to the special group `name` or to any group. If *name* is preceded by a '+' or '-', this group is added to or removed from the list of groups respectively.

maxproc { *num* | any } [*nodes*]

Limit the number of running *ParaStation MPI* processes on the selected node(s) to `num` or remove the limit.

selecttime *time* [*nodes*]

Set the timeout of the central select(2) of the *ParaStation MPI* daemon psid(8) to `time` seconds on the selected node(s).

This parameter can be set persistently via the **SelectTime** option within the *ParaStation MPI* configuration file parastation.conf(5).

master *id* [*nodes*]

Give the ParaStation daemon's some hints concerning the master node. This will actually trigger the daemon to connect the node with *ParaStation MPI* ID *id*.

psiddebug *mask* [*nodes*]

Set the debugging mask of the *ParaStation MPI* daemon psid(8) to `mask` on the selected node(s).

`Mask` is the bit-wise disjunction of the following bit-patterns:

Pattern	Name	Description
`0x0000001`	PSC_LOG_PART	Partitioning functions (i.e. PSpart_())
`0x0000002`	PSC_LOG_TASK	Task structure handling (i.e. PStask_())
`0x0000004`	PSC_LOG_VERB	Various, less interesting messages
`0x0000010`	PSID_LOG_SIGNAL	Signal handling
`0x0000020`	PSID_LOG_TIMER	Timer stuff
`0x0000040`	PSID_LOG_HW	Hardware stuff
`0x0000080`	PSID_LOG_RESET	Messages concerning (partial) resets
`0x0000100`	PSID_LOG_STATUS	Status determination
`0x0000200`	PSID_LOG_CLIENT	Client handling
`0x0000400`	PSID_LOG_SPAWN	Spawning clients
`0x0000800`	PSID_LOG_TASK	PStask_cleanup() call etc.
`0x0001000`	PSID_LOG_RDP	RDP messages
`0x0002000`	PSID_LOG_MCAST	Multicast messages
`0x0004000`	PSID_LOG_VERB	Higher verbosity (function call, etc.)
`0x0008000`	PSID_LOG_SIGDBG	More verbose signaling stuff
`0x0010000`	PSID_LOG_COMM	General daemon communication
`0x0020000`	PSID_LOG_OPTION	Option handling
`0x0040000`	PSID_LOG_INFO	Handling of info request messages
`0x0080000`	PSID_LOG_PART	Partition creation and management
`0x0100000`	PSID_LOG_ECHO	Echo each line to parse
`0x0200000`	PSID_LOG_FILE	Logs concerning the file to parse
`0x0400000`	PSID_LOG_CMNT	Comment handling
`0x0800000`	PSID_LOG_NODE	Info concerning each node
`0x1000000`	PSID_LOG_RES	Info on various resource to define
`0x2000000`	PSID_LOG_VERB	More verbose stuff

Table 2. psid debug flags

This parameter can be set persistently via the **LogMask** option within the *ParaStation MPI* configuration file parastation.conf(5).

rdpdebug *`mask`* [*`nodes`*]

Set the debugging mask of the RDP protocol within the *ParaStation MPI* daemon psid(8) to `mask` on the selected node(s).

Unless you want to debug the RDP protocol (i.e. the secure protocol used by the daemons to talk to each other) this parameter is not really useful.

`Mask` is the bit-wise disjunction of the following bit patterns:

Pattern	Name	Description
`0x0001`	RDP_LOG_CONN	Uncritical errors on connection loss
`0x0002`	RDP_LOG_INIT	Info from initialization (IP, FE, NFTS etc.)
`0x0004`	RDP_LOG_INTR	Interrupted syscalls
`0x0008`	RDP_LOG_DROP	Message dropping and resequencing
`0x0010`	RDP_LOG_CNTR	Control messages and state changes
`0x0020`	RDP_LOG_EXTD	Extended reliable error messages (on linux)
`0x0040`	RDP_LOG_COMM	Sending and receiving of data (huge! amount)
`0x0080`	RDP_LOG_ACKS	Resending and acknowledging (huge! amount)

Table 3. RDP debug flags

mcastdebug `mask` [*nodes*]

Set the debugging mask of the MCast protocol within the *ParaStation MPI* daemon psid(8) to `mask` on the selected node(s).

Unless you want to debug the MCast protocol (i.e. the protocol used by the daemons to ping alive-messages to each other) this parameter is not really useful.

`Mask` is the bit-wise disjunction of the following bit patterns:

Pattern	Name	Description
`0x0001`	MCAST_LOG_INIT	Info from initialization (IP etc.)
`0x0002`	MCAST_LOG_INTR	Interrupted syscalls
`0x0004`	MCAST_LOG_CONN	T_CLOSE and new pings
`0x0008`	MCAST_LOG_5MIS	Every 5th missing ping
`0x0010`	MCAST_LOG_MSNG	Every missing ping
`0x0020`	MCAST_LOG_MSNG	Every received ping
`0x0040`	MCAST_LOG_SENT	Every sent ping

Table 4. Multicast debug flags

freeOnSuspend [0 | 1] [*nodes*]

Switch the `freeOnSuspend` flag on or off on the selected nodes.

The `freeOnSuspend` flag steers the behavior of the resource management concerning suspended jobs. Basically there are two possible approaches: Either the resources used by the suspended job are freed for other jobs (this is done, if the flag is set to 1) or they are kept occupied in order to preserve them exclusively for the time the job continues to run (this is the behavior as long as the flag has the value 0).

Since the master node does all the resource management within the cluster, only the value on this node actually steers the behavior.

This flag can be set persistently via the **freeOnSuspend** option within the *ParaStation MPI* configuration file parastation.conf(5).

handleOldBins [0 | 1] [*nodes*]

Switch the compatibility flag for applications linked against version 4.0.x of *ParaStation MPI* on or off on the selected nodes.

nodesSort { PROC | LOAD_1 | LOAD_5 | LOAD_15 | PROC+LOAD | NONE } [*nodes*]

Define the default sorting strategy for nodes when attaching them to a partition. The different possible values have the following meaning:

PROC

Sort by the number of processes managed by *ParaStation MPI* on the corresponding nodes

LOAD_1

Sort by the load average during the last minute on the corresponding nodes

LOAD_5

Sort by the load average during the last 5 minutes on the corresponding nodes

LOAD_15

Sort by the load average during the last 15 minutes on the corresponding nodes

PROC+LOAD

Sort conforming to the sum of the processes managed by *ParaStation MPI* and the load average during the last minute on the corresponding nodes

NONE

Do not sort at all.

This only comes into play, if the user does not define a sorting strategy explicitly via `PSI_NODES_SORT`. Be aware of the fact that using a batch-system like PBS or LSF *will* set the strategy explicitly, namely to NONE.

overbook [0 | 1] [*nodes*]

Define if this nodes shall be overbooked upon user-request (if flag is true) or if overbooking should be denied at all (false).

starter [0 | 1] [*nodes*]

Define if starting jobs from this nodes should allowed (flag is true) or denied (false).

runjobs [0 | 1] [*nodes*]

Define if running tasks on this nodes should be allowed (flag is true) or denied (false).

rdpmaxretrans *val* [*nodes*]

Set RDP protocol's maximum retransmission count.

exclusive [0 | 1] [*nodes*]

Set flag marking if this nodes can be requested by users exclusively to *bool*. Relevant values are 'false', 'true', 'no', 'yes', 0 or different from 0.

pinprocs [0 | 1] [*nodes*]

Set flag marking if this nodes will use process-pinning to bind processes to cores. Relevant values are 'false', 'true', 'no', 'yes', 0 or different from 0.

bindmem [0 | 1] [*nodes*]

Set flag marking if this nodes will use memory-binding as NUMA policy. Relevant values are 'false', 'true', 'no', 'yes', 0 or different from 0.

cpumap *map* [*nodes*]

Set the map used to assign CPU-slots to physical cores to *map*. *Map* is a quoted string containing a space-separated permutation of the number 0 to `Ncore`-1. Here `Ncore` is the number of physical cores available on this node. The number of cores within a distinct node may be determined via 'list hw'. The first number in *map* is the number of the physical core the first CPU-slot will be mapped to, and so on.

allowUserMap [0 | 1] [*nodes*]

Set flag marking if this nodes will allow user to influence the mapping of processes to physical core. Relevant values are 'false', 'true', 'no', 'yes', 0 or different from 0.

supplementaryGroups [0 | 1] [*nodes*]

The supplementaryGroups flag defines whether a process spawned should belong to all groups (true) defined for this user or only to the primary group (false). Relevant values are 'false', 'true', 'no', 'yes', 0 or different from 0.

maxStatTry num [*nodes*]

Set the maximum number of tries to stat() an executable while spawning new processes to *num*. All numbers larger than 0 are allowed.

statusBroadcasts [num] [*nodes*]

Set the maximum number of status broadcasts initiated by lost connections to other daemons. See also parastation.conf(5).

rdpTimeout [ms] [*nodes*]

Set the RDP timeout in ms for all selected nodes. See also parastation.conf(5).

deadLimit [num] [*nodes*]

Set the dead-limit of the RDP status module. After this number of consecutively missing RDP-pings, the master declares the node to be dead. Only relevant, if MCast is *not* used. See also parastation.conf(5).

statusTimeout [ms] [*nodes*]

Set the Timeout of the RDP status module. After this number of milliseconds a RDP-ping is sent to the master daemon. Additionally, the master daemon checks for received ping-messages. Only relevant, if MCast is *not* used. See also parastation.conf(5).

rdpClosedTimeout [ms] [*nodes*]

Set the RDP closed timeout of the RDP status module. See also parastation.conf(5).

rdpResendTimeout [ms] [*nodes*]

Set the RDP resend timeout of the RDP status module. See also parastation.conf(5).

rdpRetrans [count] [*nodes*]

Set RDP protocol's total retransmission count. Most probably you want to reset this to 0.

rdpMaxACKPend [num] [*nodes*]

Set the maximum number of pending ACKs within the RDP facility. See also parastation.conf(5).

rdpStatistics [bool] [*nodes*]

Turn on or off collecting RDP statistics.

shutdown [*nodes*]

Shutdown the *ParaStation MPI* daemon on all selected node(s). As a consequence **all** processes using the selected node(s) are **killed**!

test [quiet | normal | verbose]

All communications links in a *ParaStation MPI* network are tested.

quiet

Quiet execution. Only a short message is printed if the test was successful.

normal

Normal execution with some messages during runtime. This is the default.

verbose

Very verbose execution with many message during runtime.

environment [set | unset] [*nodes*]

Manage the *ParaStation* daemon environment.

set key *env* [*nodes*]

Set the environment variable *key* to *env* on the selected nodes. *Env* might get quoted in order to include whitespace characters.

unset key [*nodes*]

Unset the environment variable *key* on the selected nodes.

Files

Upon startup, psiadmin tries to find `.psiadminrc` in the current directory or in the user's home directory. The first file found is parsed and the directives within are executed. Afterwards psiadmin goes into interactive mode unless the `-f` is used.

This file might be used to set some default ranges whenever psiadmin is invoked.

The startup file is ignored if the option `-c` is used.

Errors

No known errors.

See also

parastation.conf(5), psid(8), mlisten(1)

psid

psid — the *ParaStation MPI* daemon. The organizer of the *ParaStation MPI* software architecture.

Synopsis

psid [-v?] [-d *`level`*] [-f *`configfile`*] [-l *`logfile`*] [--usage]

Description

The *ParaStation MPI* daemon is implemented as a Unix daemon process. It supervises allocated resources, cleans up after application shutdowns, and controls access to common resources. Thus, it takes care of tasks which are usually managed by the operating system.

The local daemon is usually started by executing psiadmin(1). If it is not running at the time a *ParaStation MPI* process is starting, the inetd(8) or xinetd(8) daemon is starting up **psid** automatically. The daemon can also be started using the command line. Parameters can be given at the command line or in the configuration file inetd.conf(5) or xinetd.conf(5). Most of the parameters can also be given in the *ParaStation MPI* configuration file parastation.conf(5). As an alternative, the **psid** can be installed as a service using the start/stop script `/etc/init.d/parastation`.

Nodes without a running *ParaStation MPI* daemon are not visible within the cluster. Be aware of the fact that psiadmin(1) usually only starts the local daemon. All other daemons managing the nodes configured to belong to the cluster may be started using the **add** directive from within the *ParaStation MPI* administration tool psiadmin(1).

If psiadmin(1) is started with the `-s` option, all daemons within the cluster will be started automatically.

The *ParaStation MPI* daemon must always run with root privileges.

Before a process can communicate with the *ParaStation MPI* system, it has to register with the daemon. Access may be granted or denied. The daemon can deny the access due to several reasons:

- the *ParaStation MPI* system library of the process and the *ParaStation MPI* daemon are incompatible.
- the daemon is in a state where it does not accept new connections.
- insufficient resources.
- the user is temporally not allowed to access *ParaStation MPI* (see psiadmin(1)).
- the group is temporally not allowed to access *ParaStation MPI* (see psiadmin(1)).
- the number of processes exceed the maximum set by psiadmin(1).

The *ParaStation MPI* daemon can restrict the access to the communication subsystem to a specific user or a maximum number of processes. This enables the cluster to run in an optimized way, since multiple processes slow down application execution due to scheduling overhead. (See psiadmin(1), **set user**, **set group** and **set maxproc** for this features.)

All *ParaStation MPI* daemons are connected to each other. They exchange local information and transmit demands of local processes to the **psid** of the destination node. With this cooperation, the *ParaStation MPI* system offers a distributed resource management.

The *ParaStation MPI* daemon spawns and kills client processes on demand of a parent process. The *ParaStation MPI* system transfers remote spawning or killing requests to the daemon of the destination node. Then operating system functionality is used to spawn and kill the processes on the local node. The spawned process runs with same user and group id as the spawning process. The *ParaStation MPI* system redirects the output of spawned process to the terminal of the parent process.

Options

`-d , --debug=`*`level`*
: Activate the debugging mode and set the debugging level to *`level`*. If debugging is enabled, i.e. if *`level`* is larger than 0 and option `-l` is set to *`stdout`*, no fork(2) is made on startup, which is usually done in order to run **psid** as a daemon process in background.

  The debugging level of the daemon can also be modified during runtime using the **set psiddebug** command of psiadmin(1).

  Be aware of the fact that high values of *`level`* lead to excessively much debugging output spoiling the syslog(3) or the logfile.

`-f , --configfile=`*`file`*
: Choose *`file`* to be the *ParaStation MPI* configuration file. The default is to use `/etc/parastation.conf`.

`-l , --logfile=`*`file`*
: Choose *`file`* to be the destination for logging output. *`file`* may be the name of an ordinary file or `stdin` or `stdout`. The default is to use syslog(3) for any logging output.

`-v , --version`
: Output version information and exit.

`-? , --help`
: Show a help message.

`--usage`
: Display a brief usage message.

See also

parastation.conf(5), psiadmin(1), mlisten(1)

test_config

test_config — verify the *ParaStation4* configuration file.

Synopsis

test_config [-vad?] [-v] [-a] [-d] [-?] [-f filename]

Description

test_config reads and analyzes the *ParaStation4* configuration file. Any errors or anomalies are reported. By default, the configuration file `/etc/parastation.conf` will be used.

Options

`-f filename`
: Use configuration file `filename`.

`-d num`
: Set debug level to `num`.

`-v`
: Output version information and exit.

`-h -? , --usage`
: Show a help message.

test_nodes

test_nodes — test physical connections within a cluster.

Synopsis

test_nodes [-np *num*] [-cnt *count*] [-map] [-type]

Description

Tests all or some physical (low level) connections within a cluster. Therefore the program is started on *num* nodes. After all processes came up correctly, each of them starts to send test packets to every other node of the cluster. For this purpose the PSP_IReceive(3) and PSP_ISend(3) calls of the *ParaStation MPI* PSPort library are used.

After every node has received data from any node (i.e. an all to all communication was executed), a success message is printed and **test_nodes** exits. Otherwise after a certain timeout a message concerning the current status about the tested connection is posted.

test_nodes will run as long as any connection between two tested nodes is unable to transport the test packets.

Options

`-np` *num*
: Run the testing program on *num* nodes.

`-cnt` *num*
: Send *num* test packets at once instead of only one.

`-map`
: Don't print error messages but rather print a map with all working connections marked as 1 and all failed connections as 0.

`-type`
: Print the type of each connection used instead of packets send/received. Implies `-map`.

See also

psid(8)

test_pse

test_pse — test virtual connections within a cluster.

Synopsis

test_pse [-np *`num`*]

Description

This command spawns *`num`* processes within the cluster.

It's intended to test the process spawning capabilities of *ParaStation MPI*. It does not test any communication facilities within *ParaStation MPI*.

Options

`-np` *`num`*

  Spawn *`num`* processes.

See also

psid(8)

p4stat

p4stat — display information about the `p4sock` protocol.

Synopsis

p4stat [-v] [-s] [-n] [-?] [--sock] [--net] [--version] [--help] [--usage]

Description

Display information for sockets and network connections using the *ParaStation4* protocol *p4sock*.

Options

`-s`, `--sock`
: Display information about open *p4sock* sockets.

`-n`, `--net`
: Display information of network connections using *p4sock*.

`-v`, `--version`
: Output version information and exit.

`-?`, `--help`
: Show a help message.

`--usage`
: Display a brief usage message.

See also

p4tcp(8), parastation.conf(5)

p4tcp

p4tcp — configure the *ParaStation4* TCP bypass.

Synopsis

p4tcp [-v] [-a] [-d] [-?] [*from* [*to*]]

Description

p4tcp configures the *ParaStation4* TCP bypass. Without an argument, the current configuration is printed.

From and *to* are IP addresses forming an address range for which the bypass feature should be activated. Multiple addresses or address ranges may be configured by using multiple **p4tcp** commands.

To enable the bypass for a pair of processes, the library `libp4tcp.so` located in the directory `/opt/parastation/lib64` must be preloaded by both processes using:

```
  export LD_PRELOAD=/opt/parastation/lib64/libp4tcp.so
```

For parallel and serial tasks launched by *ParaStation MPI*, this environment variable is exported to all processes by default. Please refer to ps_environment(5).

Options

`-a`, `--add`
: Add an address or an address range to the list of redirected addresses. New TCP connections directed to a node within this address range may use the `p4sock` protocol for data transfer.

`-d`, `--delete`
: Delete an address or address range from the list of redirected addresses. New TCP connections to this address(es) will no longer use the `p4sock` protocol.

`-v`, `--version`
: Output version information and exit.

`-?`, `--help`
: Show a help message.

`--usage`
: Display a brief usage message.

See also

p4stat(8)

psaccounter

psaccounter — Write accounting information from the *ParaStation MPI* psid to the accounting files.

Synopsis

psaccounter [-e | --extend] [-d | --debug=*pattern*] [-F | --foreground] [-l | --logdir=*dir*] [-f | --logfile=*filename*] [-p | --logpro] [-c | --dumpcore] [--coredir=*dir*] [-v | --version] [-? | --help] [--usage]

Description

The command **psaccounter** collects information about jobs from the *ParaStation MPI* psid daemon and writes this information to the accounting files. For each day, a particular file called `yyyymmdd` is created within the accounting directory `/var/account`.

The **psaccounter** is typically started by the psid(8) using the pseudo hardware configuration `accounter` entry within `parastation.conf`.

Options

`-e`, `--extend`
: Write extended information to the accounting file. In addition to entries for job terminations, it will also record job queue, start and delete information.

`-d`, `--debug=`*`flag`*
: Print debug information. *`Pattern`* can be a combination of the following bits:

Pattern	Description
0x010	More warning messages
0x020	Show process information (start, exit)
0x040	Show received messages
0x080	Very verbose output
0x100	Show node information on startup

Table 5. Psaccounter debug flags

As the accounter is typically not run directly, but started by the psid(8), the start script `/opt/parastation/config/ps_acc` should be modified to enable this debugging facilities.

`-F`, `--foreground`
: Don't fork and go to background.

`-l`, `--logdir=`*`accountdir`*
: Create accounting files within directory *`accountdir`*. Using `-`, all accounting information is written to stdout.

`-f`, `--logfile=`*`filename`*
: Write debug and error messages to *`filename`*. Using `-` as filename, messages are written to stdout. By default, all debug and error messages are sent to syslog.

`-p`, `--logpro=`*`command`*
: Define a post processing command for accounting files. If defined, this command is called everytime a new accounting file is created. The filename of the current accounting file is appended to the command.

Calling **psaccounter** with `-p` *`gzip`* would call the command **gzip** *`yyyymmdd`* and therefore compress least recently used accounting file.

`-c`, `--dumpcore`
: Define that a core file should be written in case of a catastrophe. By default, the core file will be written to `/tmp`.

`--coredir=`*`dir`*
: Defines where to save core files.

`-v, --version`
: Output version information and exit.

`-?, --help`
: Show this help messages.

`--usage`
: Display brief usage message.

Files

`/var/account/`*`yyyymmdd`*
: Accounting files, one per day.

See also

psid(8), psaccview(8) and parastation.conf.

psaccview

psaccview — Print *ParaStation MPI* accounting information.

Synopsis

psaccview [-? | --help] [-h | --human] [-nh | --noheader] [-l | --logdir=`dir`] [-e | --exit=`exitcode`] [-q | --queue=`queue`] [-u | --user=`user`] [-g | --group=`group`] [-j | --jobname=`jobname`] [-lj | --ljobs] [-lu | --ltotuser] [-lg | --ltotgroup] [-ls | --ltotsum] [-st | --stotopt=`optstring`] [-sj | --sjobopt=`optstring`] [-t | --timespan=`period`] [-b | --begin=`yyyymmdd`] [-e | --end=`yyyymmdd`] [--jsort=`criteria`] [--usort=`criteria`] [--gsort=`criteria`] [-v | --version] [--usage]

Description

Psaccview prints various accounting information from one or more accounting files written by the *ParaStation MPI* **psaccounter**. The data may be printed to be "human readable" or may be formated to be easily post-processed by other tools.

Options

Output formatting output

`-h`, `--human`
: Print times and timestamps in more human readable form.

`-nh`, `--noheader`
: Suppress headers.

`-st`, `--stotopt=optstring`
: Defines columns displayed within the user list, group list and the total summary list. Valid entries are: `user`, `group`, `walltime`, `qtime`, `mem`, `vmem`, `cputime`, `jobs`, `cpuweight`, `aqtime` and `usage`.

`-sj`, `--sjobopt=optstring`
: Defines columns displayed within the detailed job list. Valid entries are: `user`, `group`, `jobid`, `jobname`, `start`, `end`, `walltime`, `qtime`, `mem`, `vmem`, `cputime`, `cpuweight`, `queue`, `procs` and `exit`.

Selecting entries in the input files

`-e`, `--exit=exitcode`
: Show only entries with the given exit code. You can use "false" as exit code to get all jobs where the exit code is not equal to 0.

`-q`, `--queue=queue`
: Shows only entries for the given queue.

`-u`, `--user=user`
: Show only entries for the given user name.

`-g`, `--group=group`
: Show only entries for the given group.

`-j`, `--jobname=jobname`
: Show only entries for the given jobname.

Grouping jobs

`-lj`, `--ljobs`
: Print detailed jobs list. Lists all jobs, one per line.

`-lu`, `--ltotuser`
: Print user list. Lists job summary per user, one user per line.

`-lg`, `--ltotgroup`
: Print group list. Lists job summary per group, one group per line.

`-ls`, `--ltotsum`
: Print total job summary. Lists a summary of all jobs, only one line in total.

Defining time periods considered

`-t`, `--timespan=`*`period`*
: Selects a period of time shown. Valid entries are `today`, `week`, `month` or `all`.

`-b`, `--begin=`*`yyyymmdd`*
: Defines the first day of the period of time shown.

`-e`, `--end=`*`yyyymmdd`*
: Defines the last day of the period of time shown.

Sort options

`--jsort=`*`criteria`*
: Selects the criteria where the job list is sorted by. Valid entries are `user`, `group`, `jobid`, `jobname`, `start`, `end`, `walltime`, `qtime`, `mem`, `vmem`, `cputime`, `queue`, `procs` and `exit`.

`--usort=`*`criteria`*
: Selects the criteria where the user list is sorted by. Valid entries are `user`, `jobs`, `walltime`, `qtime`, `mem`, `vmem`, `cputime`, `procs` and `cpuweight`.

`--gsort=`*`criteria`*
: Selects the criteria where the group list is sorted by. Valid entries are `group`, `jobs`, `walltime`, `qtime`, `mem`, `vmem`, `cputime`, `procs` and `cpuweight`.

General Options

`-l, --logdir=`*`logdir`*
: Look for accounting files in directory *`logdir`*.

`-v, --version`
: Output version information and exit.

`-?, --help`
: Show this help messages.

`--usage`
: Display brief usage message.

Extended description

The command **psaccview** reads entries of the type "job terminated" from the specified input files. Files could be plain text, as written be the **psaccounter**, or compressed by **gzip** or **bzip2**. Refer to psaccounter(8) for more information.

Upon startup **psaccview** tries to find the file `.psaccviewrc` in the user's home directory. Within this file, predefined variables in the command my be re-defined. See the `configuration` section within the **psaccview** script.

The command expects one file per day, named as *`yyyymmdd`*, where *`yyyy`* represents the year, *`mm`* the month and *`dd`* the day for the data contained. If not otherwise specified by using options defining periods of time to be printed, all files within the accounting directory `/var/account` for the last week will read. Unavailable files will be silently ignored.

The output may be formated to print information of each particular job (job list), of a summary of all jobs per user (user list), of a summary of all jobs per group (group list) or as a total summary of all jobs. Multiple lists can be selected, by default all information is shown. Lists may be sorted by columns and may be filtered to only show information about a particular user, group, queue, jobname or job exit code.

The columns to be printed may be defined using formatting options. Available column names are:

`aqtime`
: Average queue time, only for total summary.

`cputime`
: Total CPU time for this job.

`cpuweight`
: Walltime * number of processes.

`end`
: End time.

`exit`
: Exit code of the job.

`group`
: Group the job owner belongs to.

`jobid`
: *ParaStation MPI* internal task ID.

`jobname`
: Job name.

`jobs`
: Number of jobs, only for total summary.

`mem`
: Currently not supported.

`procs`
: Number of processes.

`qtime`
: Queuetime, giving the delay between queuing a job and running it.

`queue`
: Queue name.

`start`
: Start time.

`usage`
: Usage of a user or group of the overall walltime.

`user`
: Job owner (user name).

`vmem`
: Currently not supported.

`walltime`
: Overall wall clock time for the job.

These column names may also be used for sorting lists, where applicable.

Files

`/var/account/*` , `/var/account/*.gz` , `/var/account/*.bz2`
: Accounting files, one per day.

`$HOME/.psaccviewrc`
: Initialization file.

See also

psaccounter(8).

mlisten

mlisten — display multicast pings from the *ParaStation MPI* daemon psid(8)

Synopsis

mlisten [-dv?] [-m `MCAST`] [-p `PORT`] [-n `IP`] [-# `NODES`] [--usage]

Description

Display the multicast pings the *ParaStation MPI* daemon psid(8) is emitting continuously. These pings are displayed by spinning bars.

Each ping received from node N lets the Nth bar spin around one more step. For each node never received a multicast ping from a '.' is displayed.

ParaStation MPI by default no longer uses multicast messages. Thus, no multicast messages will be ever received and displayed using the default configuration!

Options

`-d , --debug`
: Activate the debugging mode. This will disable the spinning bars. Instead a detailed message about each received multicast ping is displayed.

`-m , --mcast=MCAST`
: Listen to multicast group `MCAST`. Set this to the value of `MCastGroup` in the *ParaStation MPI* configuration file parastation.conf(5). The default is 237, which is also the default within psid(8).

`-p , --port=PORT`
: Listen to UDP port `PORT`. Set this to the value of `MCastPort` in the *ParaStation MPI* configuration file parastation.conf(5). The default is 1889, which is also the default within psid(8).

`-n , --network=IP`
: Listen on the network interface with address `IP`. The default is to listen on any interface. `IP` is the IP address of the network interface in dot notation.

`-# , --nodes=NODES`
: Display information for `NODES` nodes. The default is 128, i.e. 128 nodes.

`-v , --version`
: Output version information and exit.

`-? , --help`
: Show a help message.

`--usage`
: Display a brief usage message.

See also

psid(8), parastation.conf(5)

Appendix A. Quick Installation Guide

This appendix gives a brief overview how to install *ParaStation MPI* on a cluster. A detailed description can be found in Chapter 3, *Installation* and Chapter 4, *Configuration*.

1. **Shutdown**

   If this is an update of *ParaStation MPI*, first shut down the *ParaStation MPI* system. In order to do this, startup **psiadmin** and issue a `shutdown` command.

   ```
   # /opt/parastation/bin/psiadmin
   psiadmin> shutdown
   ```

   This will terminate all currently running tasks controlled by *ParaStation MPI*, including **psiadmin**.

2. **Get the installation packages**

   Get the necessary installation packages from the download section of the *ParaStation MPI* homepage www.parastation.com. Required packages are `psmgmt`, `pscom` and `psmpi2`. The documentation packages `psmpi-doc` and `psmpi2-doc` are optional.

   If you want to compile the packages yourself, download the source packages (*.src.rpm) and rebuild it, using the **rpmbuild** command, e.g.:

   ```
   # rpmbuild --rebuild psmgmt.5.0.0-0.src.rpm
   # rpm -U psmgmt.5.0.0-0.i586.rpm
   # rpmbuild --rebuild pscom.5.0.0-0.src.rpm
   # rpm -U pscom.5.0.0-0.i586.rpm
   # rpm -U pscom-modules.5.0.0-0.i586.rpm
   # rpmbuild --rebuild psmpi2.5.0.0-1.src.rpm
   # rpm -U psmpi2.5.0.0-1.i586.rpm
   ```

   The `psmgmt` package must be installed before the `pscom` package may be built, similar for `pscom` and `psmpi2`. If you **only** want to rebuild the kernel modules for the p4sock protocol, use

   ```
   # rpmbuild --rebuild --with modules pscom.5.0.0-0.src.rpm
   ```

   This will render a RPM package with the *ParaStation MPI* kernel modules suitable for your setup.

3. **Install software on the server**

   Install the *ParaStation MPI* distribution files on the server machine, if not yet done:

   ```
   # rpm -U psmgmt.5.0.0-0.i586.rpm pscom.5.0.0-0.i586.rpm \
   pscom-modules.5.0.0-0.i586.rpm psmpi2.5.0.0-1.i586.rpm \
   psmpi-doc.5.0.0-0.noarch.rpm
   ```

4. **Install software on the compute nodes**

   Repeat step 3 for each node. You may omit the documentation package.

5. **Configuration**

   Next, the configuration file `parastation.conf` has to be adapted to the local settings. The template file `/opt/parastation/config/parastation.conf.tmpl` should be copied to `/etc/parastation.conf` and adjusted to the local needs. The configuration could be verified using the command test_config(1) located in `/opt/parastation/bin`.

   This configuration file must be copied to all other nodes.

6. **Startup ParaStation MPI**

Provided the *ParaStation MPI* daemon is started by the **xinetd**, run the psiadmin(1) command located in `/opt/parastation/bin` and execute the `add` command. This will bring up the *ParaStation MPI* daemon psid(8) on every node.

```
# /opt/parastation/bin/psiadmin
psiadmin> add
```

Alternatively you can start psiadmin(1) with the `-s` option.

To install the *ParaStation MPI* daemon as a system service, started up at boot time, use

```
# chk_config -a /etc/init.d/parastation
```

This step must be repeated for each node.

7. **Testing**

   A brief test of the entire communication and management system can be accomplished by using the test_nodes(1) command. For a detailed description please refer to Section 4.3, “Testing the installation”.

Appendix B. *ParaStation* license

The *ParaStation* software may be used under the following terms and conditions only.

Software and Know-how License Agreement

Version 1.0

between ParTec Cluster Competence Center GmbH

place of business: Possartstr. 20, 81679 München

represented by: Bernhard Frohwitter

- in the following referred to as ParTec -

and you

- in the following referred to as "Licensee" -

Preamble

ParTec has developed a cluster middleware software, comprising a high-performance communication layer. ParTec decided to license a base version of such software including a significant portion of the high-performance communication layer royalty free to educational institutions, such as universities, in order to enable them to evaluate, study and enhance the software. It should, however, be noted that the use of the software is solely allowed for noncommercial purposes.

If any party, such as enterprises or governmental authorities, wishes to use the software for commercial purposes in any sense, need to contact ParTec in order to ask for a commercial license. They have, however, the opportunity to use the software according to this license one-time for a limited period of three (3) months.

It is acknowledged that ParTec has invested an massive amount of labour and financial means into the development of the software. It is therefore, requested from each licensee to return the results of their studies, amendments and enhancements free of charge to ParTec in return for the granted rights as specified in this contract.

§ 1 Definitions

Technology
: means ParTec's cluster middleware solution ParaStation Base Version.

Software
: means the computer implementation of the Technology, in object code, source code or any other machine readable form, the source code of which is available for download. means ParTec's cluster middleware solution ParaStation Base Version.

Base Version Code
: means the Software in form of source code.

Modifications
: means any improvement and/or extensions by Licensee of the Base Version Code, including the underlying concepts.

Derivative Work Code
: means the Base Version Code with Modifications.

Documentation
: means specifications and description of the Technology.

University Use
: means evaluation, research, development and educational use within the educational institution, excluding use for direct or indirect commercial (including strategic) gain or advantage.

Commercial Use
: means any non-consumer use that is not covered by University Use.

Know-how
: means program documents and information which relates to Software, also in machine readable form, in particular the Base Version Code and the detailed comments on the Base Version Code, provided together with the Base Version Code.

§ 2 Granted Rights

Subject to and conditioned upon Licensees full compliance with the terms and conditions of this license, ParTec grants Licensee of this contract a non-exclusive, worldwide and royaltyfree license for University Use and Commercial Use only to:

a. reproduce, prepare derivative works of, display and perform the Base Version Code, in whole or in part, alone or as part of Derivative Work Code;
b. reproduce, prepare derivative works of and display the Documentation;
c. use the Know-how.

§ 3 Assignment and Sublicenses

Licensee does not have the right to assign the license to third parties or to grant sublicenses.

§ 4 Confidentiality

1. Licensee promises to maintain in confidence the Know-how provided to him by ParTec, in particular not to transfer it to third parties, and to use the Know-how only in the scope of this agreement.
2. For this purpose Licensee will keep all documents and data carriers containing Knowhow of ParTec locked up in the same way as he keeps its own secret documents, and Licensee shall require all of its staff having access to the know-how of ParTec to sign a written confidentiality obligation, which complies with this agreement.
3. Furthermore Licensee promises not to publish the Software as object code or as source code, nor the corresponding comments either totally or in part on his own publications or other documentation. Any functional description of Licensee's Modifications, in particular source code of Modifications, which shows Know-how, such as the structure of the Software, is prohibited.
4. The above limitations do not apply to information
   a. which Licensee already had in written form prior to signing this agreement,
   b. which have become accessible to the public due to publication of third parties without the activity of Licensee.

   The Licensee has the burden of proof for the existence of such an exception.
5. The above obligations also remain after the termination of this agreement.

§ 5 Licensee's Duties

1. Licensee shall submit to ParTec any suggestions for improvements and further developments of the Technology. ParTec may on its own discretion use, implement, publish, exploit, commercialize those suggestions.
2. Licensee shall provide source code and any documentation for its Modification to ParTec as soon as practicable, but before the publication of a functional description of Licensee's Modifications. Licensee shall include
   a. a declaration about the origin of the contributed material to the Modification, as in Attachment I,

   b. a file with the source code of Licensee's Modification showing all changes and additions made,
   c. proper description of its Modification in English language.
3. Licensee shall keep intact all existing copyright notices, any notices referring to particular licenses and any warranty related statements.

§ 6 Grant-Back

1. Licensee grants ParTec for Modifications being severable improvements a nonexclusive, perpetual, irrevocable, worldwide and royalty-free license, and for Modifications being non-severable improvements an exclusive, perpetual, irrevocable, worldwide and royalty-free license to
   a. use, reproduce, modify, display, prepare derivative works of and distribute its Modifications and derivative works thereof, in whole or in part, in source code and object code form, as part of the Software or other technologies based in whole or in part on Base Version Code or Technology;
   b. use, reproduce, modify, display, prepare derivative works of and distribute Licensee's documentation of its Modifications;
   c. sublicense any of the foregoing through multiple tiers of distribution
2. As far as the license granted in section §6(1) covers Licensee's patents and patent applications, Licensee grants ParTec a non-exclusive, perpetual, irrevocable, worldwide and royalty-free license.

§ 7 Procedure against Infringers

Licensee shall promptly inform ParTec about infringement acts related to the Software, particularly about offers and distribution of piracy copies.

§ 8 Trademarks and Source Indication

Licensee does not have the right, except in the copyright notice, to use the company name or any trademark of ParTec. Licensee may, however, in the usual way refer to the fact that Licensee uses ParTec's Software.

§ 9 Exclusion of Liability

1. ParTec is not liable for errors and does not guarantee the specific utility of the Technology for Licensee's purpose. In particular, ParTec is not liable for indirect or subsequent damages due to errors of the licensed Software.
2. ParTec is not aware of any rights of third parties which would oppose University Use or Commercial Use. ParTec is not liable however for the licensed Software and the licensed Know-how being free of rights of third parties.
3. If Licensee is accused by third parties of infringing intellectual property rights due to the use of the licensed Software or the licensed Know-how, ParTec promises to provide Licensee with information and documents in defense against such claims as far as ParTec is able to do so without breach of third party obligations and while maintaining its own confidentiality interests. All costs involved in such activities shall be borne by Licensee.

§ 10 Duration

1. If licensee solely makes University Use of the licensed Software and Know-how, this agreement is entered into for a duration of 1 year. It is extended year by year unless it is terminated 6 months beforehand by one of the parties.
2. For licensees, which make Commercial Use of the licensed Software or Know-how, this agreement is entered into for a duration of three (3) months only. Licensee acknowledges that he may only take advantage of this license for Commercial Use once. For any subsequent retrieval of the Software, licensee needs to obtain a separate License from ParTec.

§ 11 Termination

1. A termination for cause is possible particularly in the case of one of the following reasons.
   a. A breach by the Licensee of one of the obligations under this agreement and the fruitless expiration of a period of 10 days after warning by ParTec.
   b. Beginning of bankruptcy or receivership proceedings relating to the property of the other party.
   c. An essential change in the control over the other party, in particular merger or major control by a third company.
2. A breach by Licensee of any one of the obligations under sections §4, §5 and §6, will automatically terminate Licensee's rights under this license.

§ 12 Rights after Expiration of the Agreement

1. All rights of Licensee on the use of the Base Version Code end at the expiration or termination of this agreement.
2. Licensee promises to return to ParTec within one week after the expiration or termination of the agreement all relevant documents relating to the Base Version Code, whether secret or not, as well as all copies made and to delete any copies retrieved by downloading or copies thereof.

§ 13 Salvatorian clause

1. Should a provision of this agreement be invalid or become invalid or should this agreement contain an omission, then the legal effect of the other provisions shall not thereby be affected.
2. Instead of the invalid provision a valid provision is deemed to have been agreed upon which comes closest to what the parties intended commercially; the same applies in the case of an omission.

§ 14 Changes to this agreement

Any change of this agreement shall be made in writing, no collateral agreements to this agreement have been made.

§ 15 Jurisdiction and Applicable Law

For all controversies out of this agreement the patent chamber of the District Court Munich I shall have jurisdiction. The applicable law shall be that of the Federal Republic of Germany excluding United Nations Convention on Contracts for the International Sale of Goods (CISG) and International Private Law.

Attachment I - Declaration of Origin

Material covered by this certificate (version, release, etc.):

____

Was any portion of the software material written by anyone other than you or your employees within the scope of their employment? YES/NO

Was any portion of the software material (e.g., Code, associated documentation, etc.) derived from preexisting works (either yours or a third party's), including any code from freeware, shareware, electronic bulletin boards, or the Internet? YES/NO

Please let us know any circumstance which might affect ParTec's ability to reproduce, distribute and market this software material, including whether your software material was prepared from any preexisting materials which have any: (a) confidentiality or trade secret restrictions to others; (b) known or possible royalty obligations to others; (c) used other preexisting materials developed for another party or customer (including government) where you may not have retained full rights to such other preexisting materials.

You recognize that, for copyright registration or enforcement of legal rights relating to the furnished software material, ParTec may need you to produce additional information related to the software material. You hereby agree to cooperate with ParTec and provide such information to ParTec at ParTec s request. As an authorized representative of your institution, you hereby certify the above to be true and accurate.

BY: ________________________________ (Authorized Signature)

Name: ________________________________ (Type or Print)

Title: ________________________________

Appendix C. Upgrading *ParaStation4* to *ParaStation MPI*, version 5

This appendix explains how to upgrade an existing *ParaStation4* installation to the current *ParaStation MPI* version.

C.1. Building and installing *ParaStation MPI* packages

Just recompile the packages:

```
# rpmbuild --rebuild psmgmt.5.0.0-0.src.rpm
# rpm -U psmgmt.5.0.0-0.i586.rpm
# rpmbuild --rebuild pscom.5.0.0-0.src.rpm
# rpm -U pscom.5.0.0-0.i586.rpm
# rpm -U pscom-modules.5.0.0-0.i586.rpm
# rpmbuild --rebuild psmpi2.5.0.0-1.src.rpm
# rpm -U psmpi2.5.0.0-1.i586.rpm
```

Before installing the new packages, stop the *ParaStation MPI* psid daemon on each node. This will terminate all currently running jobs!

Install the `psmgmt`, `pscom` and `psmpi2` packages on all nodes using the **rpm** command. Install the `psmpi-doc` package holding the documentation on the login node(s) and frontend servers.

The existing *ParaStation4* configuration file `/opt/parastation/config/parastation.conf` is just fine for *ParaStation MPI*. If new features like process pinning should be used, adjust the existing configuration file. Look for `pinProcs`, `CPUmap`, `bindMem`, `supplGrps` and `RLimit Core` entries in the new template file `parastation.conf.tmpl`, copy them to the current configuration file and adjust them to your needs.

The configuration file of *ParaStation MPI* located in `/etc` is no longer a symbolic link to `/opt/parastation/config/parastation.conf`, therefore we recommend to copy this file to `/etc`.

Restart the *ParaStation MPI* daemons:

```
# psiadmin -s
```

This will startup the **psid** on all nodes of the cluster. Check with **psiadmin** command if all nodes and interconnects are available.

C.2. Changes to the runtime environment

When running applications with *ParaStation MPI*, some differences to *ParaStation4* must be observed.

First of all, jobs must be started using the new **mpiexec** command. Refer to mpiexec(8) for options and details.

The former **mpirun** command is no longer supported! If it is still available, due to an *ParaStation4* `mpich-ps4` package still installed, this command will not run executables compiled and linked with *ParaStation MPI*!

In addition, jobs may no longer be started by running the executable, like

```
$ ./myprog …
```

Use the **mpiexec** command instead!

Executables linked with *ParaStation4* can be run using the new **mpiexec** command. In this case, the option `-b` or `--bnr` is required.

The environment variable `PSP_P4SOCK` was renamed to `PSP_P4S`, but still recognized. Within this version of *ParaStation MPI*, both names may be used. Likewise, The environment variable `PSP_SHAREDMEM` was renamed to `PSP_SHM`, but also still recognized.

Glossary

Address Resolution Protocol	A sending host decides, through a protocols routing mechanism, that it wants to transmit to a target host located some place on a connected piece of a physical network. To actually transmit the hardware packet usually a hardware address must be generated. In the case of Ethernet this is 48 bit Ethernet address. The addresses of hosts within a protocol are not always compatible with the corresponding hardware address (being different lengths or values). The Address Resolution Protocol (ARP) is used by the sending host in order to resolve the Ethernet address of the target host from its IP address. It is described in the RFC 826. The ARP is part of the TCP/IP protocol family.
Administration Network	The administration network is used for exchanging (meta) data used for administrative tasks between cluster nodes. This network typically carries only a moderate data rate and can be entirely separated from the data network. Almost always, Ethernet (Fast or more and more Gigabit) is used for this purpose.
Administrative Task	A single process running on one of the compute nodes within the cluster. This process does not communicate with other processes using MPI. This task will not be accounted within the *ParaStation MPI* process management, ie. it will not allocate a dedicated CPU. Thus, administration tasks may be started in addition to parallel tasks. See also Serial Task for tasks accounted with *ParaStation MPI*.
admin-task	See Administrative Task.
ARP	See Address Resolution Protocol.
Data Network	The data network is used for exchanging data between the compute processes on the cluster nodes. Typically, high bandwidth and low latency is required for this kind of network. Interconnect types used for this network are Myrinet or InfiniBand, and (Gigabit) Ethernet for moderate bandwidth and latency requirements. Especially for Ethernet based clusters, the administration and data network are often collapsed into a single interconnect.
CPU	Modern multi-core CPUs provide multiple CPU cores within a physical CPU package. Within this document, the term `CPU` will be used to refer to a independing computing core, independent of the physical packaging.
DMA	See Direct Memory Access.
Direct Memory Access	In the old days devices within a computer were not able to put data into memory on their own but the CPU had to fetch it from them and to store it to the final destination manually. Nowadays devices as Ethernet cards, harddisk controllers, Myrinet cards etc. are capable to store chunks of data into memory on their own. E.g. a disk controller is told to fetch an amount of memory from a hard disk and

	to store it to a given address. The rest of the jobs is done by this controller without producing further load to the CPU. Obviously this concept helps to disburden the CPU from work which is not its first task and thus gives more power to solve the actual application.
Forwarder	See *ParaStation MPI* Forwarder.
Logger	See *ParaStation MPI* Logger.
Master Node	The evaluation of temporary node lists while spawning new tasks is done only by one particular psid(8) within the cluster. The node running this daemon is called *master node*. The master node is dynamically selected within the cluster and may change, if the current master node is no longer available. Election is based on the node IDs, refer to parastation.conf(5).
Network Interface Card	The physical device which connects a computer to a network. Examples are Ethernet cards (which are nowadays often found to be on board) or Myrinet cards.
NIC	See Network Interface Card.
Non-Uniform memory access (NUMA)	Non-Uniform memory access describes the fact that for some multiprocessor design the access time to the memory depends on the location of this memory. Within this designs, the memory is typically closely attached to a CPU. CPUs have access to memory attached to other CPUs using additional logic inducing additional latency. Therefore the access time for different memory addresses may vary.
Parallel Task	A bunch of processes distributed within the cluster forming an instance of a parallel application. E.g. a MPI program running on several nodes of a cluster can only act as a whole but consists of individual processes on each node. *ParaStation MPI* knows about their relationship and can handle them as a distributed parallel task running on the cluster. Sometimes also referred as *job*.
ParaStation MPI Logger	The counterpart to the *ParaStation MPI* Forwarder. This process receives all output collected by the forwarder processes and sends it to the final destination, stdout or stderr. Furthermore input to the *ParaStation MPI* task is forwarded to a specific process. The first process of the task started usually converts to the logger processes after spawning all the other processes of the parallel task.
ParaStation MPI Forwarder	Collects output written by *ParaStation MPI* controlled processes to `stdout` or `stderr` and sends it to the *ParaStation MPI* Logger. Furthermore the forwarder controls the process and sends information about its exit status to the local daemon.
PMI	Process Manager Interface: protocol to standardize startup of tasks of a parallel job. Implemented in **mpd** and *ParaStation MPI* **psid**.
Process	The atomic part of a Parallel Task. A process is at first a standard Unix process. Since *ParaStation MPI* knows about its membership in a parallel task, it can be handled in a peculiar way if an event takes place on some

	other node (e.g. another process of the task dies unexpectedly, a signal is send to the task, etc.).
Serial Task	A single process running on one of the compute nodes within the cluster. This process does not communicate with other processes using MPI. *ParaStation MPI* knows about this process and where it is started from. A serial task may use multiple threads to execute, but all this threads have to share a common address space within a node.