ParaStation V5

ParaStation GRIDMONITOR

—

Administrator's Guide

Release 1.9.2-2
Published September 2010

ParTec CLUSTER COMPETENCE CENTER

GridMonitor Administrator's Guide

Release 1.9.2-2

September 2010

Printed 10 November 2010, 15:38

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>

This document provides detailed information about the *ParaStation GridMonitor*. Installation and configuration of the *ParaStation GridMonitor* as well as usage of the *ParaStation GridMonitor* commands and graphical user interface are explained in-depth.

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

List of Figures

List of Examples

Chapter 1. Preface

1.1. About this book

This book discusses installation, configuration and usage of *ParaStation GridMonitor*, based on the packages `pscollect`, version 5.0 and `psgridmon`, version 5.0.

The *ParaStation GridMonitor* uses functions of the *ParaStation MPI* job management. For more information on *ParaStation MPI*, refer to the *ParaStation MPI User's Guide* and *ParaStation MPI Administrator's Guide* or to http://www.par-tec.com.

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

1.2. This book's audience

The *ParaStation GridMonitor* and thus this book is targeted at supporting cluster system administrators. Despite the fact that those administrators are users in a sense that they may use the software on a regular basis, and they will find a lot of of information on how to use the software in this book, we still call the book *GridMonitor Administrator's Guide*, as there are also a couple of chapters describing how to set up the *GridMonitor*.

1.3. *ParaStation GridMonitor* overview

The *ParaStation GridMonitor* is a versatile monitoring application for Linux-based compute clusters. It provides the administrator with various aspects of the available information, from an overall status to in-depth view of details. The *GridMonitor* may retrieve and store all kind of data from within one or more clusters. Data can be visualized in various ways using a web browser. Furthermore, the data is constantly monitored and in case of an abnormal situation, this is reported to the administrator.

The *GridMonitor* is designed for high scalability, robustness and low system overhead.

Part I. Setting up the *GridMonitor*

The sections in Part I give a brief overview of the *GridMonitor* and explain how to install and configure all parts of the *GridMonitor*.

Chapter 2. Introduction

2.1. What is the *ParaStation GridMonitor*?

The *ParaStation GridMonitor* is a versatile monitoring application for Linux-based compute clusters. It provides the administrator with various aspects of the available information, from an overall status to in-depth view of details. The *GridMonitor* may retrieve and store all kind of data from within one or more clusters. Data can be visualized in various ways using a web browser. Furthermore, the data is constantly monitored and in case of an abnormal situation, this is reported to the administrator.

The *GridMonitor* is designed for high scalability, robustness and low system overhead.

2.2. Data collection process: *collector*

All available data is retrieved and managed by a central process called *collector*, using various protocols, like SNMP, and others talking to remote agents. To minimize overall system and network load, only data requested by a client application is read from agents by the *collector*. If no one is interested, no data is transfered and therefore no compute cycles and network bandwidth is wasted. In addition, the *collector* is especially designed to handle dead or overloaded nodes, broken network connections and limited network bandwidth.

The *collector* gathers various data from (almost) all information sources available within a cluster describing:

- compute nodes,
- fileservers, frontend nodes,
- baseboard management controllers (BMCs),
- network switches,
- disks and storage devices,
- environment monitoring devices,
- runtime systems,
- batch queuing systems.

For all types of data sources, dedicated plug-ins for the *collector* controlling local or remote agents are provided. These plug-ins read the requested data using the appropriate agent and format them to be usable by the *collector*.

Parameters not only include operating system values like system load, network counters or temperatures, but also parameters describing active jobs provided by the ParaStation process management, queued jobs provided by a batch queuing system, etc. If available, they may even include information provided by rack/room environment monitoring devices, uninterruptible power supplies or similar devices.

Each parameter is cached within the *collector* for a certain period of time. Clients reading a parameter within this cache timeout will be provided with the cached value and are therefore not able to cause excessive network traffic.

Data can be stored to and retrieved from a database; therefore a data history is available, e.g. for plotting diagrams. Parameters and sample frequencies can be configured independently.

'Virtual' parameters can be computed, monitored and stored to the database based on actual read data, e.g. the total system load as sum of all node load values.

Each known numerical parameter can be compared to an upper and lower limit. In case these value under-runs or over-runs those limits, actions can be triggered, e.g. generating an event. In addition, string

parameters can be compared to constant strings, too. To constantly monitor these parameter, reading cycles can be defined. This is fully configurable with respect to parameter name, upper and lower limit, cycle time, etc.

Events describing abnormal situations within a cluster can be generated by monitoring parameter limits, node availablity, etc. Events will be stored within the database and reported by email.

Beside the actual data, the *collector* also provides information about the type of available data. This is called the `parameter type system`. Using this system, it's easy for the *GridMonitor* GUI to construct dynamic selection boxes without actually reading the data and therefore wasting network bandwidth and compute cycles. The parameter type system also enables a *GridMonitor* GUI to dynamically include new parameters without modifying the scripts or page layout.

2.3. Graphical client (*GridMonitor* GUI)

The web browser-based *GridMonitor* GUI is the first available client for the *collector*. Based on a webserver using PHP scripts, it visualizes the information provided by the *collector*. The data is grouped within various views, like cluster overview, node overview, process details, environmental details, parameter lists, etc.

The graphical user interface also provides information about currently pending events and event history. Parameters can be shown graphically as history diagrams for periodically sampled data, or as bar and radar diagrams for current data.

The following figure examplarily shows the overview of the cluster `Softcomp`.

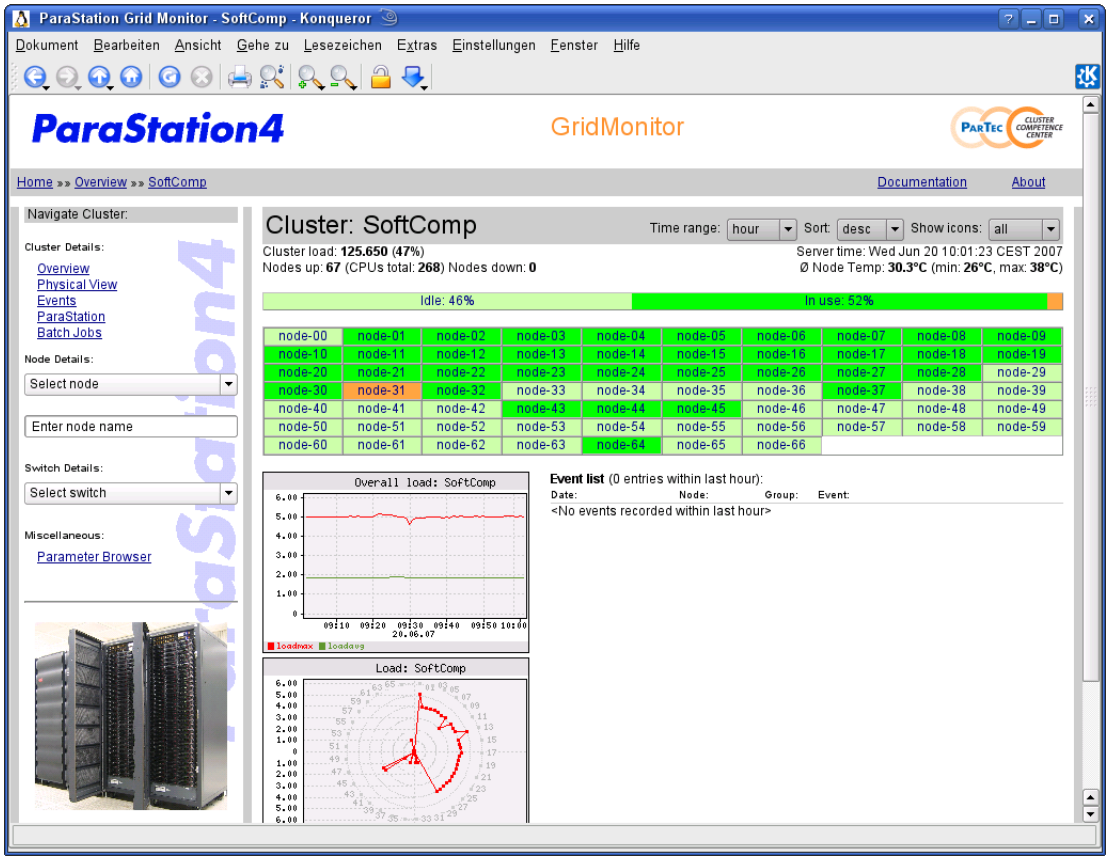

Figure 2.1. Example *GridMonitor* GUI picture

Chapter 3. Installation

The installation of the *GridMonitor* is divided into two parts: installing the *collector* and installing the *GridMonitor* GUI. In addition, the available documentation could be installed to provide online manuals and this document. This chapter explains how to install everything within a cluster environment.

3.1. Installation prerequisites

The *GridMonitor* is available as a set of RPM packages for Linux only. The `pscollect` package is available for i586 and x86_64 architecutures, whereas the `psgridmon` and `psgridmon-doc` packages are architecture-independent and therefore suitable for all Linux architectures.

In addition, a valid license for the *collector* is required.

Superuser privileges are required to install all components of the *ParaStation GridMonitor*.

Please contact `<support@par-tec.com>` for information on how to obtain the latest version of the *ParaStation GridMonitor* software.

The *collector* relies on additional tools to collect information. *ParaStation* management, version 4.3.4, is required to provide a list of currently active jobs on a cluster. If not installed or not configured, no information about active jobs will be displayed. Refer to Section 4.2.5, “Configuring the *collector* – step 4” for details.

The *GridMonitor* uses the `netsnmp` libraries to collect information from SNMP devices. These libraries are required even if no SNMP devices are configured. For information on how to configure SNMP devices, refer to Section 4.2.8, “Configuring the *collector* – step 7”.

In addition, the *GridMonitor* is able to provide information collected from a batch queuing system. Currently, only `Torque` is supported. To actually read the data, the command **qstat** provided by Torque must be installed on the configured node. For configuration details, refer to Section 4.2.6, “Configuring the *collector* – step 5”.

If data from BMCs should be read using the IPMI protocol, the command **ipmitool** from the package `ipmitool` is required.

The *ParaStation GridMonitor* GUI requires an installed `apache` webserver providing support for PHP. Especially, the following PHP modules must be installed: apache2-mod_php, php5, php5-gd, php5-iconv, php5-tokenizer and php5-ctype. Supported are PHP version 4.2 up to version 5.2.

3.2. Installing the *collector*

The RPM package `pscollect-<version>.<arch>.rpm` holds all files required for configuring and running the *collector* and the agents. This package must be installed on each compute node monitored afterwards. It also must be installed on an additional collector node, unless this node is already part of the cluster. On the collector node, the command **pscollect** implementing the *collector* and database is run. Whereas on the monitored nodes, only the agent script **psvalue** is started, remotely controlled by the *collector*. Installing this package requires 10MB of disk space located in `/opt`.

The **pscollect** command requires a `glibc` version of 2.3.3-118 or higher. Using previous versions of `glibc` will result in crashes of **pscollect** command due to a `glibc` problem.

To install the *collector*, the RPM package `pscollect-<version>-<arch>.rpm` must be installed using the command

```
rpm -Uv pscollect-5.0.11-1.x86_64.rpm
```

While installing the package, an additional unprivileged user account called `pscd` is created on this node. This user will run the *collector* and agent scripts.

To enable the startup of the agents from the *collector*, a **ssh** auto-login for user `pscd` from the collector node to all other nodes must be enabled. See Section 4.3.2, “Configuring the *collector* – enable `pscd` auto-login” for more information.

On the collector node, the configuration file `/etc/pscollect/cluster.conf` must be edited to set up the list of monitored nodes, switches, etc. See Section 4.2, “Configuring the *collector* – step by step” for details. In addition, a valid license file must be copied to `/etc/pscollect/license`.

After installing the package on all nodes and configuring the *collector*, the `pscollect` daemon may be started on the collector node using

```
/etc/init.d/pscollect start
```

All agents are started automatically by the *collector* (see note about **ssh** auto-login above). In addition, the required database files are created and initialized, if not already present. This script may also be used to install the *collector* as a service which might be started up while booting the system. Refer to insserv(8) and chkconfig(8) for details.

3.3. Installing the *GridMonitor* GUI

The *GridMonitor* uses the `Apache` webserver including the PHP extension `mod_php` to generate and deliver the HTML pages. Version 1.3 and 2.0 of `Apache` and version 4 and 5 of `mod_php` are supported. Currently, only Linux istallations of `Apache` are supported. Installing the *GridMonitor* GUI package requires about 10MB of disk space located in `/opt`.

To install the graphical user interface for the *GridMonitor*, the RPM package `psgridmon-<version>-noarch.rpm` must be installed using the command

```
rpm -Uv psgridmon-5.0.7-1.noarch.rpm
```

After configuring the *GridMonitor* GUI (see Section 4.4, “Configuring the graphical user interface (*GridMonitor* GUI)”) and reloading the webserver by running the command

```
/etc/init.d/apache reload
```

the *GridMonitor* may be accessed using the link http://localhost/gridmon, whereas `localhost` must be replaced by the actual hostname, if accessed from different nodes.

Currently, all users are allowed to view the information provided by the *GridMonitor*. No restrictions are applied.

To get the full navigation capabilities, JavaScript must be enabled within the web browser.

3.4. Installing the documentation

To install the documentation for the *GridMonitor*, the package `psgridmon-doc-<version>-noarch.rpm` must be installed using the command

```
rpm -Uv psgridmon-doc-4.5.0-0-noarch.rpm
```

After installing the documentation package, the online version of the *ParaStation GridMonitor Administrator's Guide* is available using the '`Documentation`' button within the *GridMonitor* GUI or using the link file://opt/parastation/doc/html/psgridmon-adminguide/index.html. A PDF version of the *ParaStation GridMonitor Administrator's Guide* is also available in the directory `/opt/parastation/doc/pdf`.

Installing this packages requires 2MB of disk space located in `/opt`. It may be installed without the `pscollect` and/or the `psgridmon` package(s).

3.5. Uninstalling the *GridMonitor*

All parts of the *GridMonitor* can be removed using the command

```
rpm -e pscollect psgridmon psgridmon-doc
```

Uninstalling the `pscollect` package will not delete the various database files (`events.db3`, `history.psdb` and `{monitor,state,users}.ini`) in the directory `/var/lib/pscollect`. As this files may take a lot of disk space, they should be deleted by the administrator if no longer needed.

Chapter 4. Configuration

This chapter describes the configuration of both the *collector* and the graphical user interface (*GridMonitor* GUI).

4.1. Default configuration

Installing both packages of the *GridMonitor* renders a default configuration for a cluster called `Cluster1`. The *collector* and web server are expected to run on the same host, therefore the PHP scripts use `localhost` to access the *collector*.

Currently, there is no way to automatically configure the known nodes within the *collector*, this has to be done manually. See next section for details.

4.2. Configuring the *collector* – step by step

The *collector* reads its initial configuration from the default configuration file `cluster.conf`, located in the directory `/etc/pscollect`. The *collector* is running as non-privileged user `pscd`, therefore, this file should be owned and be readable by this user.

Comments start with a `--` and are ignored until the end of line. The complete configuration is defined using LUA, a powerful scripting language. Refer to http://www.lua.org for more details.

The configuration is devided into particular steps, each of them is describing a particular configuration aspect. The next capters will describe these steps in detail.

The *collector* must be restarted to activate the newly created or modified configuration. To do so, run the command

```
/etc/init.d/pscollect restart
```

4.2.1. Configuring the *collector* – basics

The first section within the *collector* configuration file defines global configuration entries, required for the *collector*. These entries typically don't have to be modified. Note the final call of the `init()` method. This is important bring the *collector* into a well defined state.

```
--# -*- lua -*-
--# pscollect configuration
--#
--# Lua 5.1 Reference Manual: http://www.lua.org/manual/5.1/
--#

--# Set root password to abc321 via:
--# echo "passwd root abc321" | psget /var/run/pscollect/socket     (1)

debug=false

--# Load defaults
include "/opt/parastation/config/sys_cluster.pscc"

--# Overwrite defaults?
-- put('config/db_root', '/var/lib/pscollect')
-- put('config/bin_dir', '/opt/parastation/bin')
-- put('config/rcmd', 'ssh')


--# Initialize
init()
```

Example 4.1. Basic *collector* configurations

4.2.2. Configuring the *collector* – step 1

These section defines whether the *collector* accepts connections from each node or only connections initiated by clients on the local node.

```
--# ##############################################################
--# Step 1: Should any node be able to read information from
--#         collector (required, security)?
--# Accept connections only from localhost:
listen {host="localhost", port=4000}                                (1)
--# Or accept connections from any host:
--# (required if your webserver does not run on the same node)
--listen {host="0.0.0.0", port=4000}
--# Accept local socket connections:
listen {socket="/var/run/pscollect/socket"}                         (2)
```

Example 4.2. Configuring *collector* accessability

The configuration entry (1) above tells the *collector* to accept connections from `localhost`, via TCP port `4000`, and (2) configures it to open a socket in the local file system.

When connected through a local socket, the *collector* checks the user id of the peer process. If the owner is either `root` or the same user as that running the *collector* (normally `pscd`), that client automatically gains administrator privileges, without needing to provide a password. This can e.g. be used to change the password, as shown at the beginning (1) of the configuration file..

See also Section 4.4.1, “Configuring basic *GridMonitor* GUI parameters”.

4.2.3. Configuring the *collector* – step 2

Within this section, the cluster name is defined:

```
--# ##############################################################
--# Step 2: define first cluster (required)
cluster("Cluster1")                                                (1)
```

Example 4.3. Configuring cluster name

The entry `cluster` (1) tells the *collector* about a cluster called `Cluster1`. All further configuration steps refer to this cluster. The entry has to be modified to meet your current cluster name.

4.2.4. Configuring the *collector* – step 3

Within this section, all nodes belonging to a particular cluster are defined.

```
--# ##############################################################
--# Step 3: Define host list for first cluster (required)
--#   Note: To enable reading S.M.A.R.T data from disk drives,
--#         add the next two lines to the file /etc/sudoers
--#         (on each node):
--#         ------
--#         Defaults:pscd   passwd_tries=0, !syslog
--#         pscd    ALL=NOPASSWD: /usr/sbin/smartctl \
--#           -[iAH] /dev/[sh]d[a-z], /usr/sbin/smartctl \
--#           -d ata -[iAH] /dev/[sh]d[a-z]
--#         ------
--#         Using S.M.A.R.T data is optional
--#   Warning: With some systems/controllers/disks, reading
--#         S.M.A.R.T data may hang your system.
--#         Test it!!!
--#         Define host list, one by one:
host("localhost")          -- replace with real hostname           (1)
host("master")                                                     (2)
host("node01")
host("node02")
host("node03")

--# Or use a "lua for loop"?
--# (defining nodes 'cnode1' up to 'cnode16')
for n = 1, 16 do                                                   (3)
    host("cnode"..n)
end

--# Use node names with leading '0'
# (defining nodes 'node-01' up to 'node-16')
for n = 1, 16 do
    host("node-"..string.format("%0.2d",n))                        (4)
end
```

Example 4.4. Configuring host list

The entry `host` (1) announces a new host to the cluster. All hosts announced using this function show up as cluster nodes within the graphical user interface. The name `localhost` should be replaced by the actual node name, e.g. `master`.

It's generally not a good idea to list a node called `localhost`, as this name is ambiguous within a cluster. Use the cluster-internal real name of the node, like `master` or `node01`.

To enable the **pscollect** command to remotely run the required agent script, the **ssh** auto-login for the user `pscd` must be configured on the newly announced host, see Section 4.3.2, “Configuring the *collector* – enable `pscd` auto-login” for more information.

To announce a bunch of nodes at once, a LUA loop may be used (3). Quit often, the numbering part of the node names include leading zeros, like `node01`. Line (4) shows how to generate appropriate node names.

At least, one `host` entry is required.

4.2.5. Configuring the *collector* – step 4

Within this section, the nodes providing *ParaStation* process management information are defined.

```
--# ##############################################################
--# Step 4.1: Hostname of ParaStation master node  (opt.)
ps4_host("localhost")                                            (1)
```

Example 4.5. Configuring *ParaStation* host name

The entry `ps4_host` (1) tells the *collector* to connect to the *ParaStation* daemon **psid** on `localhost` to gather *ParaStation* process management information, e.g. which jobs are currently active. Using `localhost` is ok, as this host name will not show up in the graphical user interface. This entry is optional and is set by defaults to `localhost`.

```
--# ##############################################################
--# Step 4.2: Hostname of ParaStation accounting node (opt.)
--#     Note: To provide accouting information, grant the
--#           user 'pscd' read access to all ParaStation
--#           accouting files on the accounting host:
--#           chmod g+rx /var/account;
--#           chmod g+r /var/account/*;
--#           chgrp -R pscd /var/account
ps4_acc_host("localhost")                                        (1)
```

Example 4.6. Configuring *ParaStation* accounting host name

The entry `ps4_acc_host` (1) tells the *collector* to connect to the host `localhost` to read *ParaStation* job accounting information. Using `localhost` is ok, as this host name will not show up in the graphical user interface. This entry is optional and is disabled by default.

The user `pscd` must be able to read the *ParaStation* accounting files located in the directory `/var/account`.

4.2.6. Configuring the *collector* – step 5

Within this section, the nodes providing batch system information are defined.

Currently, only **Torque** is supported.

```
--# ##########################################################
--# Step 5.1: Hostname of TORQUE server node (optional)
pbs_host("localhost")                                          (1)
```

Example 4.7. Configuring batch server name

The entry `pbs_host` (1) tells the *collector* to connect to the Torque server on `localhost` to gather batch job information. Using `localhost` is ok, as this host name will not show up in the graphical user interface. This entry is optional and is disabled by default.

```
--# ##########################################################
--# Step 5.2: Hostname of TORQUE accounting node (optional)
pbs_acc_host("localhost")                                      (1)
```

Example 4.8. Configuring batch accounting host name

The entry `pbs_acc_host` (1) tells the *collector* to read job accounting information collected by Torque from host `localhost`. Using `localhost` is ok, as this host name will not show up in the graphical user interface. This entry is optional and is disabled by default.

4.2.7. Configuring the *collector* – step 6

Within this section, the configuration of the virtual sensors subsystem will be defined. These virtual sensors read real sensors by either using IPMI or using lmsensors package on a node. At least one of these two sensor sources must be configured.

```
--# ##########################################################
--# Step 6: Define hardware sensor sources (required)
--#         Either IPMI (6.1) or lmsensors (6.2) must be used!

--# ##########################################################
--# Step 6.1: Define IPMI sensor sources (optional)

--# ##########################################################
--# Step 6.1.1: Define IPMI user access information in file
--#             /etc/pscollect/ipmiuser (file is required)
--#             File format: 'user:password'                   (1)
--#       Note: This file should be readable only for user
--#             'pscd'.
```

Example 4.9. Setting up the IPMI authenication

Within this step, the authentification information used to connect to a baseboard management controller (BMC) via IPMI is defined. This information is stored within a separate file `/etc/pscollect/ipmiuser`. The file contains only a single line with username and password, separated by a colon.

This file should only be readable by the user `pscd`.

```
--# ##############################################################
--# Step 6.1.2: Map IPMI hosts (BMCs) to hosts (nodes)
--#             (required, if IPMI), e.g.
--#             ipmi_host("hostname","ipmi_host_addr") or
--#             ipmi_host_p("hostname","ipmi_host_name",
--#                         "username","password")
--#             If no username/password is provided, the file
--#             /etc/pscollect/ipmiuser will be consulted
--#             (see 6.1.1)
--#       Note: If your local BMC does not respond to requests
--#             from the local host, e.g. ping from master to
--#             master-bmc does not resolve the BMC address,
--#             use the special IPMI host name "localhost".
--#             Using this name, the ipmitool uses the 'open'
--#             interface, which requires proper kernel
--#             module support. Try
--#               chkconfig -a ipmi
--#               /etc/init.d/ipmi start
--#             In addition, the user 'pscd' must be able to
--#             run ipmitool as user root. Add the next two
--#             lines to the file /etc/sudoers:
--#             ------
--#             Defaults:pscd   passwd_tries=0, !syslog
--#             pscd    ALL=NOPASSWD: /usr/bin/ipmitool -A none \
--#               -I open -H localhost [a-zA-Z/]*, \
--#               /usr/bin/ipmitool -A none -I open \
--#               -H localhost -S [a-zA-Z /]*
--#             ------

--# Single IPMI host
--ipmi_host("node01","192.168.44.1")                                  (1)
--ipmi_host("node02","node02-ipmi")
--ipmi_host("master","localhost")      -- see Note above

--# 50 IMPI hosts in one loop
for n = 1, 50 do                                                     (2)
  ipmi_host("node"..n,"192.168.44."..n)
end
```

Example 4.10. Configuring IPMI host mapping

The `ipmi_host` entry (1) tells the *collector* to read IPMI information for host `node01` from the BMC with address `192.168.44.1`. Names like `node01-bmc` may also be used instead of the BMC IP address.

The entry (2) shows an example how to map a number of nodes (`node1` up to `node50` using a LUA loop.

```
--# ##############################################################
--# Step 6.1.3: Define IPMI chassis (optional), e.g.
--#             ipmi_chassis("hostname","ipmi_chassis_addr"
ipmi_chassis("chassis1","192.168.20.1")                              (1)
```

Example 4.11. Configuring IPMI chassis mapping

The `ipmi_chassis` entry (1) maps a Chassis BMC controller managing multiple blade servers to a chassis name.

This is currently not supported in the *GridMonitor* GUI!

```
--# ##############################################################
--# Step 6.1.4: Map IPMI server sensor data (required, if IPMI),
--#             e.g. map_ipmihost("hostname","virtualsensor"
--#                                "realsensor")
--#             Required virtual sensors are:
--#               TempCPU1,
--#               TempCPU2,
--#               TempNode,
--#               FAN1,
--#               FAN2,
--#               FAN3,
--#               FAN4

--# To map 16 nodes called node01 up to node16 at once, use:
for n = 1, 16 do
    host = "node"..string.format("%0.2d",n)
    map_ipmihost(host,"TempNode","Ambient_Temp")                    (1)
    map_ipmihost(host,"TempCPU1","Temp1")
    map_ipmihost(host,"TempCPU1","Temp2")
    map_ipmihost(host,"FAN"..n, "fan"..n)
    map_ipmihost(host,"FAN"..n, "fan"..n)
    map_ipmihost(host,"FAN"..n, "fan"..n)
    map_ipmihost(host,"FAN"..n, "fan"..n)
end

--# Mapping suitable for 16 Dell server SC1435 called node01
--# up to node16:                                                  (2)
--for n = 1, 16 do
--    host = "node"..string.format("%0.2d",n)
--    map_ipmihost(host,"TempNode","Ambient_Temp")
--    map_ipmihost(host,"TempCPU1","Temp1")
--    map_ipmihost(host,"TempCPU1","Temp2")
--    for i = 1, 2 do
--        map_ipmihost(host,"FAN"..i,     "FAN_MOD_"..i.."A_RPM")
--        map_ipmihost(host,"FAN"..i +2, "FAN_MOD_"..i.."B_RPM")
--        map_ipmihost(host,"FAN"..i +4, "FAN_MOD_"..i.."C_RPM")
--        map_ipmihost(host,"FAN"..i +6, "FAN_MOD_"..i.."D_RPM")
--    end
--end

--# Mapping suitable for a Dell server PE1950:                     (3)
--map_ipmihost("node01","TempNode","Ambient_Temp")
--# Note: Temp3 and Temp4 seems to be constant (40), so
--#       ignore it for now!
--for i = 1, 2 do
--   map_ipmihost("node01","TempCPU"..i,"Temp"..i)
--end
--for i = 1, 4 do
--    map_ipmihost("node01","FAN"..i,      "FAN_MOD_"..i.."A_RPM")
--    map_ipmihost("node01","FAN"..i +4 , "FAN_MOD_"..i.."B_RPM")
--    map_ipmihost("node01","FAN"..i +8 , "FAN_MOD_"..i.."C_RPM")
--    map_ipmihost("node01","FAN"..i +12, "FAN_MOD_"..i.."D_RPM")
--end
```

Example 4.12. Configuring IPMI host sensor mappings

Within this section, the mappings from IPMI sensor values to virtual sensor values are defined. For example, entry (1) maps the IPMI sensor called `Ambient_Temp` to the virtual sensor name `TempNode`. Using LUA loops is very convenient to map a group of BMCs at once.

Use the parameter browser (`ipmi->sdr->list`) to list all available sensor names and values.

The entry (2) shows an example how to map the IPMI sensors of a Dell server SC1435. Similar, the entry (3) shows the mapping for a Dell server PE1950.

```
--# ##########################################################
--# Step 6.1.5: Map IPMI chassis sensor data (optional), e.g.
--#             map_ipmichassis("hostname","virtualsensor",
--#                             "realsensor")
map_ipmichassis("chassis1","TempChassis","Temp1")                    (1)
```

Example 4.13. Configuring IPMI chassis sensor mappings

The `map_ipmichassis` entry (1) maps a Chassis BMC controller sensor value called `Temp1` to the virtual sensor `TempChassis`.

This is currently not supported in the *GridMonitor* GUI!

```
--# ##########################################################
--# Step 6.2: Define lmsensors sources (optional), e.g.
--#           map_lmhost("hostname","Virtualsensor",
--#                      "Realsensor")
--#           Required virtual sensors are:
--#               TempCPU1,
--#               TempCPU2,
--#               TempNode,
--#               FAN1,
--#               FAN2,
--#               FAN3,
--#               FAN4
--#           To list your available sensors, use sensors
--for n = 1, 16 do
--    host = "node"..string.format("%0.2d",n)
--    map_lmhost(host,"TempCPU1","temp1")                            (1)
--    map_lmhost(host,"TempCPU2","temp2")
--    map_lmhost(host,"TempNode","temp3")
--    map_lmhost(host,"FAN1","fan1")
--    map_lmhost(host,"FAN2","fan2")
--    map_lmhost(host,"FAN3","fan3")
--    map_lmhost(host,"FAN4","fan4")
--end
```

Example 4.14. Configuring lmsensors sensor mappings

The `map_lmhost` entry (1) maps the sensor called `temp1` read using `lmsensors` on node `node01` to the virtual sensor called `TempCPU1` for node `node01`.

Like mapping IPMI values, using a LUA for-loop is handy to map a group of identical nodes at once.

Use the parameter browser (`hosts->sensors`) to list all available sensor names.

4.2.8. Configuring the *collector* – step 7

This section describes how to configure SNMP managed network switches.

```
--# ##############################################################
--# Step 7: Define SNMP managed switches (optional), e.g.
--#         snmp("addr")
snmp("switch1")                                                    (1)

--# Switch with non-default arguments:
snmp("switch2", { host="sw4", version = "1" })                     (2)

--# Switch with all arguments:
snmp("switch3",                                                    (3)
    {
        host = "s",            -- snmp source (defaults to name)
        community = "public",  -- snmp community
        version = "2c",        -- version ("1","2c" or "3")
        table_expire = 15,     -- cache expire in s
        value_expire = 5,      -- cache expire in s

        timeout = 500,         -- connection timeout in ms
        retries = 3            -- connection retries
    }
)
```

Example 4.15. Setting up SNMP devices

The entry `snmp` (1) announces a SNMP managable device using default values to connect. Likewise, entry (2) announces a snmp device called `switch2` using the address `sw4` and the SNMP protocol version 1. Entry (3) shows all available options to the `snmp` mapping call.

4.2.9. Configuring the *collector* – step 8

This section describes how to configure an additional cluster within the *collector*.

```
--# ##############################################################
--# Step 8: Define second cluster (optional)
--#         Repeat steps 2 to 7
--cluster "Cluster2"
--host "front2"
--host "c2node1"
--host "c2node2"
--host "c2node3"
--ps4_host "front2"
```

Example 4.16. Configuring an additional cluster

These entries show how to configure an additional cluster, managed by this *collector*. Just define a new `cluster` entry and repeat all required and optional configuration steps from step 2 up to step 7.

4.2.10. Configuring the *collector* – step 9

This section describes how to configure monitoring of parameter limits and saving parameters into the database.

```
--# ##########################################################
--# Step 9: Define monitoring limits and parameters stored
--#         into database (required)


...

--# ##########################################################
--# Step 9.1: Save load1 values to DB (required) and define
--#           monitor limit (optional)
--#           Monitor all clusters and all hosts
parameter("cluster/*/hosts/*",                                  (1)
    {
        monitor = { intern = true, group = "crit" },
        poll = 30,                                              (2)
        load1 = {
            save_history = compress_load, -- required           (3)
--#         Enable overload warnings: (max > nbr of cores)
--          monitor = { max = 2.1, group = "warn" },            (4)
            poll = 300                                          (5)
        },
        memfree = {                                             (6)
            save_history = compress_min, -- required
--#         Enable memory shortage warnings:
--          monitor = { min = 20000, group = "warn" },
            poll = 600
        },
        swapfree = {                                            (7)
            save_history = compress_min, -- required
--#         Enable swap shortage warnings:
--          monitor = { min = 20000, group = "warn" },
            poll = 600
        }
    }
)
```

Example 4.17. Defining general monitors

The entry (1) defines a monitor for the parameters `load1`, `memfree` and `swapfree` for all hosts on all clusters. The connection to this host is checked every 30 secs (2). Lost connections will be reported using the critical level ("`crit`").

The parameter `load1` is stored to the database (3) and may be compared to an upper limit of 2.1 (4). Exceeding the maximum value would be reported using the event group ("`warn`"). Monitoring the upper limit is currently disabled. This parameter is read, compared and stored every 120 secs (5). The entry for parameter `load1` is required.

Analogous to `load1`, monitors for the parameters `memfree` (6) and `swapfree` (7) are pre-defined. Every 10 minutes, both values are stored to the database. Monitoring of the minimum values is disabled within this example.

```
--# ##############################################################
--# Step 9.2: Monitor and save all required sensor limits
--#           (required)
--#     Note: This configures all nodes identically
--#           using '.../hosts/*'
parameter("cluster/*/sensors/hosts/*",
    {
        monitor = { intern = true, group = "crit" },
        poll = 30,
        TempCPU1 = {
--#         save parameters to DB (for diagrams)
            save_history = compress_max_hi,
--#         warn if temperature exceeds 60 (Celsius?)
            monitor = { max = 60, group = "warn" },
            poll = 120
        },
        TempCPU2 = {
            save_history = compress_max_hi,
            monitor = { max = 60, group = "warn" },
            poll = 120
        },
        TempNode = {
            save_history = compress_max_hi,
            monitor = { max = 60, group = "warn" },
            poll = 120
        },
        FAN1 = {
            save_history = compress_max,
--#         warn if fan speed drops below 3000 rpms
            monitor = { min = 3000, group = "warn" },
            poll = 300
        },
        FAN2 = {
            save_history = compress_max,
            monitor = { min = 3000, group = "warn" },
            poll = 300
        },
        FAN3 = {
            save_history = compress_max,
            monitor = { min = 3000, group = "warn" },
            poll = 300
        },
        FAN4 = {
            save_history = compress_max,
            monitor = { min = 3000, group = "warn" },
            poll = 300
        }
    }
)
```

Example 4.18. Defining virtual sensor monitors

Example 4.18, “Defining virtual sensor monitors” shows how to monitor and store the virtual sensor parameters. This entry is highly recommended. If configured, it will inform the administrator about fan or thermal problems.

Virtual sensor parameters may be configured by using either IPMI or lmsensors data. For details how to map these entries, refer to the previous section.

```
--# ##############################################################
--# Step 9.3: Define required load status (required)
parameter("cluster/*/stat",
    {
        load1 = {
            max = {
               save_history = compress_load,
               poll = 60
            },
            min = {
               save_history = compress_load,
               poll = 60
            },
            avg = {
               save_history = compress_load,
               poll = 60
            }
        }
    }
)
```

Example 4.19. Defining load1 minimum, maximum and average

This monitor calculates every 60 secs the minimum, maximum and average of the `load1` value of all hosts and saves it to the database. This monitor is required by the *GridMonitor* GUI and must not be modified!

4.2.11. Configuring the *collector* – step 10

This section describes how to configure the event notification system of the *ParaStation GridMonitor*.

Each parameter within the *collector* holds an internal state, e.g. `unavailable` or `high`. When transitioning from one state to another, events will be generated, which may be added to event groups.

Currently, only two event groups (`warn`, `crit`) are used. Refer to the previous section how to configure monitors and assign them to event groups.

```
--# ##############################################################
--# Step 10: Define event notification (required)

--# ##############################################################
--# Step 10.1: Define event notification for critical events
--#            (required)
lput("parameter/event/crit",                                        (1)
    {
        collect_time = 60,      -- collect events for 60 sec        (2)
        exec_time = 30*60,      -- max. 1 mail per 30 min           (3)

        exclude_states = {                                          (4)
            "ok"                -- dont send mails for state "ok"
        },

        unavailable = 3,        -- Warn after 3 read failures       (5)
--        exec = event_system_call( \                               (6)
--            "cat >> /tmp/pscollect.events",  \
--             "Warnings:", "")
--        exec = event_system_call( \                               (7)
--            "env DISPLAY=:0 xmessage -file -",  \
--            "Warnings:", "")
        exec = event_system_call( \                                 (8)
            "mail root -s \"Cluster Cluster1 Critical Events\"",  \
            "Critical events:", "")
    }
)

--# ##############################################################
--# Step 10.2: Define event notification for warning events
--#            (required)
lput("parameter/event/warn",                                        (9)
    {
        collect_time = 120,     -- collect events for 120 sec
        exec_time = 60*60,      -- max. 1 mail per 60 min

        exclude_states = {
            "ok"                -- dont send mails for state "ok"
        },

        unavailable = 3,        -- Warn after 3 read failures
--        exec = event_system_call( \
--            "cat >> /tmp/pscollect.events",  \
--            "Warnings:", "")
--        exec = event_system_call( \
--            "env DISPLAY=:0 xmessage -file -",  \
--            "Warnings:", "")
        exec = event_system_call( \
            "mail root -s \"Cluster Cluster1 Warnings\"",  \
            "Warnings:", "")
    }
)
```

Example 4.20. Defining event notification

The entry (1) defines the configuration for events within the group `crit`. Initially, they will be collected for 60 secs (2) before an event handling call will be executed. After this initial collect time, more events of this type will be collected for 1800 secs (30 min) (3) before the next event handling call will be issued. This insures that the email system (see below) and the administrator will not be flooded in case of catastrophic errors.

The `exclude_states` list (4) defines a list of states which will not be reported, e.g. all `ok` states. The next entry (5) defines how many consecutive read failures may occure, before a connection is declared dead. The entries (6) and (7) give examples on how to act on `crit` events. The entry (8) define the actual action taken in case a `crit` event handling call is issued. In this example, an email will be sent notifying the administrator. The command will be executed as user `pscd`.

Similar to the entry in line (1), the entry (9) defines the timeout and action taken for the event group `warn`.

4.3. Configuring the *collector* – additional steps

Beside setting up the *collector* configuration file, more things have to be configured to enable all functions of the *ParaStation GridMonitor*.

4.3.1. License

The *collector* requires a valid license provided in the file `/etc/pscollect/license`. Licenses may be obtained from `<support@par-tec.com>`.

4.3.2. Configuring the *collector* – enable pscd auto-login

All remote agents, launched by the *collector*, will by default be started using **ssh**. Therefore, the user `pscd` on the collector node must be able to login to each cluster node without providing a password or pass phrase.

To enable this "auto-login", a `ssh` key, e.g. `~/.ssh/id_dsa.pub`, of user `pscd` on the collector node must be added to its own `~/.ssh/authorized_keys` file and the `authorized_keys` of this user on all other nodes.

4.3.3. Configuring the *collector* – enable local IPMI access

Typically, BMCs share one of the network interfaces with the server. Quite often, the BMC is only reachable from other nodes, not from the local host itself. To circumvent this problem, the *collector* may be advised to use the local kernel interface to read data from the local BMC. To do so, the special name `localhost` must be used as BMC name when calling the `impi_host()` mapping function.

```
ipmi_host("master","localhost")
```

Example 4.21. Configure access to local IPMI parameters

Refer to Section 4.2.7, “Configuring the *collector* – step 6” for details.

To enable the *collector* to read data for the local node using the command **ipmitool**, the *collector* must be able to access the IPMI device `/dev/ipmi0`. In order to do so, the following lines must be added to the file `/etc/sudoers`:

```
Defaults:pscd   passwd_tries=0, !syslog
pscd    ALL=NOPASSWD: /usr/bin/ipmitool -A none \
  -I open -H localhost [a-zA-Z/]*, /usr/bin/ipmitool -A none -I open \
  -H localhost -S [a-zA-Z /]*
```

Example 4.22. Enabling access to local IPMI data

4.3.4. Configuring the *collector* – enable disk monitoring

To enable the *collector* to read and monitor disk parameters using S.M.A.R.T, the command **smartctl**, launched by the agent process, must be able to access the raw disk devices, like `/dev/hda`. To do so, the following lines must be added to the file `/etc/sudoers`:

```
Defaults:pscd   passwd_tries=0, !syslog
pscd    ALL=NOPASSWD: /usr/sbin/smartctl \
  -[iAH] /dev/[sh]d[a-z], /usr/sbin/smartctl \
  -d ata -[iAH] /dev/[sh]d[a-z]
```

Example 4.23. Enabling access to disk parameters

This has to be done on all nodes of the cluster.

For particular disk controllers or drives, the **smartctl** may hang your system. Please test carefully! Not all drives or controllers support S.M.A.R.T.

4.4. Configuring the graphical user interface (*GridMonitor* GUI)

The PHP scripts providing the web-based *GridMonitor* GUI are configured using the configuration file `/opt/parastation/www/gridmon/config/cluster-psmp.inc.php`. This file holds an array in PHP syntax, defining all configurable *GridMonitor* GUI parameters.

Comments are marked, as usual in PHP, with `//` (until end of line) or with `/* … */`.

4.4.1. Configuring basic *GridMonitor* GUI parameters

The basic configuration section defines information about known *collectors* and therefore known clusters:

```
$Cluster = array (                                                  (1)
  'Cluster1' => array (                                             (2)
    'pscollector' => array(
      'server' => 'localhost',                                      (3)
      'port' => 4000                                                (4)
    ),
    'PhView' => &$PhView_Cluster1,                                  (5)
    …
  ),
//'Cluster2' => array (                                             (6)
//  'pscollector' => array(
//    'server' => 'frontend2',                                      (7)
//    'port' => 4000                                                (8)
//   ),
//  'PhView' => &$PhView_Cluster2,                                  (9)
//    …
//),
);
```

Example 4.24. Basic cluster configuration for the *GridMonitor* GUI

This defines a list of known Clusters `$Cluster` (1), having an entry for cluster `Cluster1` (2). How to connect to the related *collector* is defined by `pscollector`, using the node `localhost` (3) and TCP port `4000` (4).

Most of the parameter info is read from the *collector* at startup, utilizing the parameter type subsystem of the *collector*. Therefore, only parameters local to the PHP scripts have to be defined at this place.

To add a second cluster, e.g. `Cluster2` (6), uncomment or copy the appropriate lines and modify the cluster name ('`Cluster2`') (6) and server name ('`frontend2`') (7). The TCP port ('`4000`') (8) will most probably be the same.

the cluster name has to match the configured cluster name in the *collector* configuration file!

The array `PhView` (5) defines the physical outline of the cluster, specifying how many racks, how many nodes per rack and how many blades per blade chassis are installed. This may be adjusted to match the actual cluster layout. For more information, refer to Section 4.4.2, “Configuring *GridMonitor* GUI physical view”.

4.4.2. Configuring *GridMonitor* GUI physical view

As mentioned in the previous section, the variable `PhView` (5) within the cluster configuration (see Section 4.4, “Configuring the graphical user interface (*GridMonitor* GUI)”) refers to an array describing the physical outline of the cluster.

```
$PhView_Cluster1 = array(
  'Cluster1' => array(                                                  (1)
    'row' => 1,
    'uw' => 130,                                                        (2)
    'uh' => 14,                                                         (3)
    'free'       => array('h' => 1, 't' => 'skip'),                     (4)
    'blade01'    => array(                                              (5)
      'h' => 6, 't' => 'Blade',                                         (6)
      'n' => array(                                                     (7)
        'w' => 10,                                                      (8)
        'skip'      => array('h' => 1, 't' => 'skip'),                  (9)
        'bnode-001' => array('h' => 1, 't' => 'Node'),                  (10)
        'bnode-002' => array('h' => 1, 't' => 'Node'),
        …
      ),
    ),
    'node-01'   => array('h' => 2, 't' => 'skip'),                      (11)
    'node-02'   => array('h' => 2, 't' => 'skip'),
    …
    'localhost' => array('h' => 2, 't' => 'Node'),                      (12)
    'Switch-1'  => array('h' => 1, 't' => 'Switch'),                    (13)
  ),
);
```

Example 4.25. Configuring the physical cluster view for the *GridMonitor* GUI

The array variable `$PhView_Cluster1` defines a single rack called `Cluster1` (1). If more than a single rack is used, the array `Cluster1` must be duplicated and more reasonable names like `Rack1`, `Rack2`, etc. should be used. These names will also show up in the '`Rack Details`' pull-down menu of the left hand navigation area of the physical view page. See also Section 8.2, "*GridMonitor* GUI: Cluster physical view page".

Each basic entry in the rack is 130 pixel wide (2) and 14 pixel high (3). The entry `skip` (4) is unused ('skip').

The next entry (5) of type 'blade' represents a blade chassis (holding vertical slots) with a height of 6 units (6). This array (7) defines the slots within the blade chassis. Each slot is 10 pixels wide (8). The first slot (9) is empty ('skip'). The following slots (10) are filled with nodes `node-001` to `node-002` ('Node').

Unused entries ('skip') of height 2 ('h') are defined in (11). Whereas (12) defines a node entry `localhost` ('Node') of height 2 ('h'). The last entry (13) defines a switch entry `switch-1` ('Switch') of height 1 ('h').

If an entry is marked as type node (`'Node'`) or as switch (`'Switch'`), the entry name equals the node or switch name. Both types are animated showing their current state using different colors. Entries with type skip (`'skip'`) are drawn, but not animated.

The height of each rack is computed automatically. Empty space may be inserted using entries of type `'skip'`. Multiple racks may be defined within the array variable `Cluster1`.

To minimize typing, the scripts `makeracks.sh` and `makebladeracks.sh` are provided, located in `/opt/parastation/www/psgridmon/config`. These scripts generate a prototype physical configuration for typical server or blade racks by using information on how many nodes are installed, how many nodes per rack are installed, what size these nodes are, etc. The output of these scripts may be saved into a file and modified to suit the current physical layout.

4.4.3. Configuring *GridMonitor* GUI default values

A number of default values for the *GridMonitor* GUI can be configured for each cluster independently using the array variable `.defaults` within the global array `$Cluster`, located in the file `cluster-psmp.inc.php`.

```
$Cluster = array(
  'Cluster1' => array(
    …
    '.defaults' => array(
        /* default icon state */
        'Match' => 'All',                                          (1)
        /* default time frame */
        'Last'  => 'hour',                                         (2)
        /* default sort order */
        'Sort'  => 'desc',                                         (3)
        /* event groups ignored in cluster overview */
        'IgnoredEvents' => array(                                  (4)
            'default' => true,
            'ok' => true,
        ),
        /* number of icons per line in cluster overview */
        'NbrOfIconsPerRow' => 10,                                  (5)
        /* list of pre-defined rack views, */
        /*   e.g. 'Left' => 'rack0:rack1' */
        'PhView' => array(                                         (6)
        ),
        /* default diagram size */
        'DiagramSize' => 'medium',                                 (7)
        /* list of known PBS queues, */
        /*   e.g. 'large' => 'large' */
        'QueueList' => array(                                      (8)
        ),
    ),
    …

  ),
);
```

Example 4.26. Configuring default values for *GridMonitor* GUI

The entry `Match` (1) defines the default icons state shown when displaying the cluster overview page. Valid entries are none to suppress icons entirely, `all` to show all icons, `idle` to show icons for idle nodes only, `in use` to show icons for nodes in use only, `ev.pnd` to show icons for nodes having pending events, `dead` to show icons for unavailable nodes and `ev+de` to show icons for nodes having pending events and also unavailable nodes.

The parameter `Last` (2) defines the default time frame for history charts and event lists. Valid entries are `10min`, `hour`, `4hour`, `12hour`, `day`, `week` or `month`.

The `Sort` entry (3) indicates whether lists are sorted ascending (`asc`) or descending (`desc`) by default.

The array `IgnoreEvents` (4) lists all events which should not show up in the cluster overview event list.

With the variable `NbrOfIconsPerRow` (5) the number of icons per line within the cluster overview icon area is defined.

The array `PhView` (6) defines lists of racks selectable as groups within the `Physical view` cluster page. Each array entry consists of a label and a colon-separated list of rack names, which are configured in the `PhysView` section. See Section 4.4.2, “Configuring *GridMonitor* GUI physical view”, for details.

All particular rack entries configured within the physical view configuration (see Section 4.4.2, “Configuring *GridMonitor* GUI physical view” are selectable by default and should not be configured within this array.

The variable `DiagramSize` (7) specifies the default size of diagrams within the Diagram page. Valid entries are `small`, `medium` or `large`.

The array variable `QueueList` (8) lists all queues of a PBS batch queuing system. See also Section 4.2, “Configuring the *collector* – step by step” for information on how to configure a PBS server.

4.4.4. Configuring cluster pictures within the *GridMonitor* GUI

The *GridMonitor* GUI displays small pictures of all configured clusters within the left hand navigation area. To show actual pictures of the systems, copy image files named like the configured clusters to the default installation directory `/opt/parastation/www/gridmon/images`. The pictures are automatically scaled by the browser, but for best results the picture size should be 180x120 pixels. The pictures may be encoded as GIF using the filename suffix `.gif`, as JPEG with suffix `.jpg` or as PNG with suffix `.png`.

Do not overwrite the default file `Cluster.jpg`, otherwise the picture will not link to the particular cluster, but to this section of the documentation.

Chapter 5. Maintenance

The *GridMonitor* is designed as a tool to monitor many nodes for a long period of time. Therefore, special care has been taken to ensure the long term stability of the components.

In spite of this design principle, a few issues must be observed while running the *GridMonitor* for a long period of time.

5.1. Parameter database

By nature, the *GridMonitor* is constantly inserting new parameter (+ timestamp) data into the parameter database. Using a special round-robin database, the new data overwrites the oldest ones recorded in the database. For each particular parameter, multiple stages of historical data may be and by default will be configured. Moving data from one stage to another will take place automatically and will compress a set of historical data within the current stage to provide a new single date within the next stage. The *GridMonitor* will handle this compressed data transparently.

Due to the design of the round-robin database, no special care must be taken of this database. The space required to store all data will be allocated on the file system during startup of the *collector*.

More information how to configure the compression of parameter data may be found in Section 4.2.10, “Configuring the *collector* – step 9”.

5.2. Event database

All events generated by the *collector* will be recorded into the event database. No events are ever deleted, therefore this database will grow.

Data can be purged manually. Use the command **sqlite** to manipulate data in the event database file directly. For more infos on **sqlite** refer to sqlite(1).

5.3. Logfile

The *collector* stores all internal messages and error messages received from agent scripts into the logfile `/var/log/pscollect.log`.

Part II. Using the *GridMonitor* graphical user interface

The sections in Part II describe how to navigate and read information using the graphical user interface of the *GridMonitor*.

Chapter 6. *GridMonitor* GUI: Navigation

6.1. General hints

The Graphical User Interface (*GridMonitor* GUI) uses HTML pages, sent by a webserver upon request. To use the *GridMonitor* GUI, any graphical browser may be used. It is tested with recent versions of Mozilla, Firefox, Konqueror and Internet Explorer 6. The window size should be at least 640x480 pixels, all available window space will be used.

All navigation information is sent using URL parameters, e.g. …`&view=node&node=node-01`. Therefore, each page can be easily bookmarked for future references. No cookies are used!

As mentioned before, JavaScript is required for proper navigation with the *GridMonitor*. To automatically re-display a page without pressing the reload button of the browser, add the parameter `&refresh=nn` to the URL. This will send a redirect header with the current address to the browser, with a delay of `nn` secs. If redirection is enabled within the browser, the page will automatically be reloaded after this period of time. The timeout `nn` must be greater than 20 secs!

This works for all pages within the *GridMonitor* GUI. Be careful when refreshing pages showing a large amount of data within small intervals. Depending on the data type(s) and source, it will be reloaded by the *collector*, therefore, large amounts of data may be transfered each time the current page is refreshed.

6.2. Topbar and left hand navigation area

The left hand navigation area of each page allows you to navigate to different views of the current topic, e.g. showing the 'physical view' instead of the 'overview' of a cluster. In addition, links to sub-components are available, like cluster nodes within the cluster overview page.

Figure 6.1. Example left hand navigation area

The current navigation depth is shown in the top bar, therefore, going to the superior level is as easy as clicking to the respective entry. In addition, links to the online documentation ('`Documentation`') and the About page ('`About`') are always available.

Figure 6.2. Example topbar

Using the example shown in Figure 6.2, “Example topbar”, clicking on '`Home`' will bring you to the *GridMonitor* home page, '`Overview`' will bring you to the overview page of all known clusters, whereas '`JULI`' will bring you to the overview page of cluster `JULI`.

The online documentation is only available, if the *GridMonitor* documentation packages is installed. See Section 3.4, “Installing the documentation” for details.

6.3. View configuration

Within the page header, most of the available pages have a view configuration section, where general view parameters for this page may be selected by using pull-down menus.

Figure 6.3. Example view configuration area

'`Time range`'
: This pull-down menu defines the time range for historical data displayed on the current page. This time range is used for history charts, event lists and similar information, where a period of time beginning now and going back in time is required.

'`Sort by`'
: Within the parameter browser, this pull-down menu selects the column used for sorting.

'`Sort`'
: This pull-down menu defines the sorting order for all types of lists displayed. Ordering can be ascending or descending.

'`Show icons`'
: This pull-down menu defines which node icons are shown in the icon area. Node icons are selected by the current node state. Available selections are:

Entry	Description
all	all node icons are shown
idle	only icons for idle nodes are shown
in use	only icons for used nodes are shown (load1 > 0.1)
ev.pnd	only icons for nodes having pending events are shown
dead	only icons for dead nodes are shown
ev+de	only icons for nodes which are dead or have pending events are shown

Table 6.1. Icon selection

The default icon state can be configured using the variable `Match` within the *GridMonitor* GUI configuration. See Section 4.4.3, “Configuring *GridMonitor* GUI default values” for details. All icons are shown in a color representing the current node state. See glossary section for details about node states and colors.

Figure 6.4, “Example icon area” shows an example icon area indicating different kinds of node states.

Figure 6.4. Example icon area

In this example, node `juli-cn026-c` is currently unavailable, all other nodes are either idle or in use.

For a short-hand for the icon pull-down menue, click on the appropriate section of the loadbar to get a list of icons with the selected state.

To save display space in the browser window, select `'ev+de'` to only show icons for nodes requiring the administrators attention. Nodes running under normal conditions are typically not of interest. This is especially true for large clusters, as the icon area would fill up the browser window.

Other view configuration pull-down menus not shown in the above example include:

`'Sort by'`
: This pull-down menu defines available columns within tables, which might be used for sorting. This menu is available within the parameter browser.

`'Select queue'`, `'Select job state'`
: This pull-down menus allows you to narrow the list by the selected queue and job state.

6.4. Diagrams

Each diagram within the *GridMonitor* GUI can be clicked on to bring up a detailed diagram page. Within the left hand navigation area of this diagram page, the size of the chart can be selected (`small`, `medium`, `large`). The default diagram size within this page is configurable, see Section 4.4.3, “Configuring *GridMonitor* GUI default values” for details.

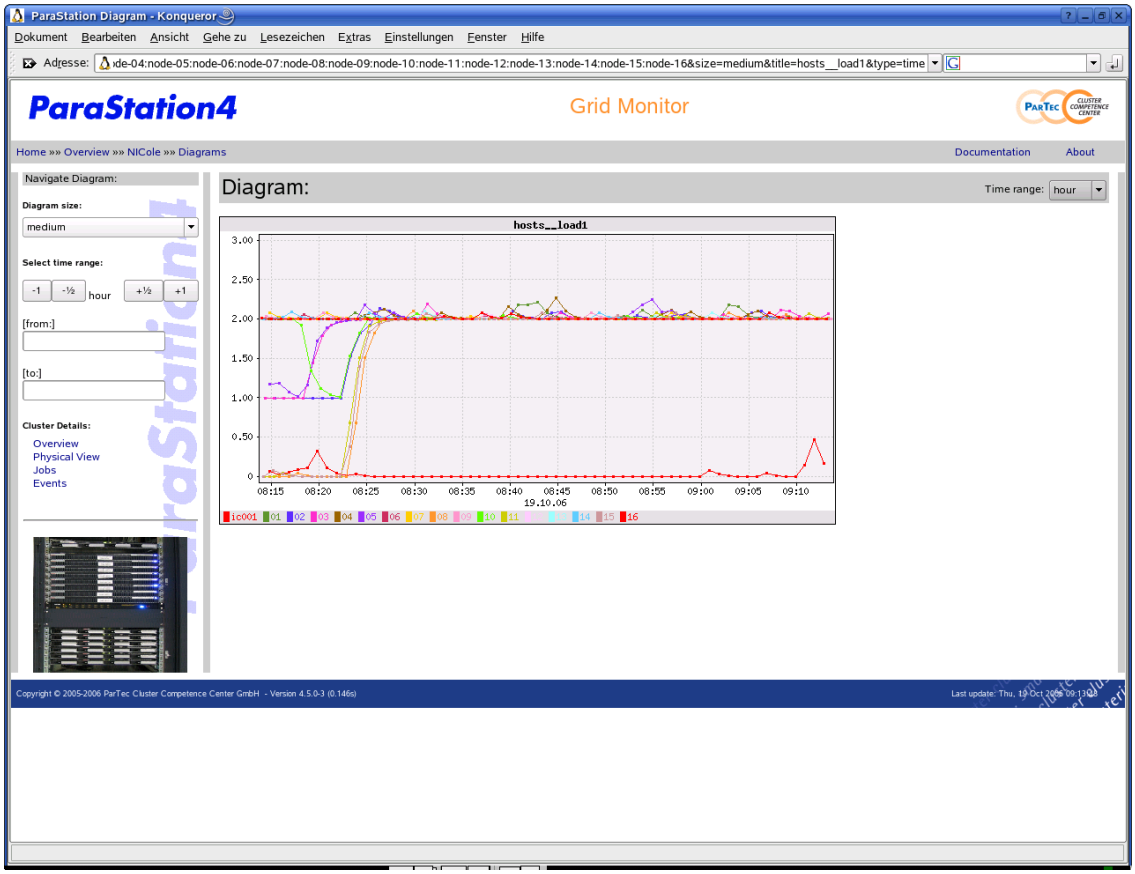

Figure 6.5. Example diagram

For history charts, the displayed time frame can be redefined, by selecting the appropriate entry from the '`Time range`' pull-down menu. Alternatively, a dedicated time frame can be selected using the left hand navigation area entries '`[from:]`' and '`[to:]`'. Time entries are in the format dd.mm.yyyy hh:mm:ss. In addition, the current time frame can be moved back and forth, if already in the past, on the time axis using the '`-1`', '`-1/2`', '`+1/2`' and '`+1`' buttons within the 'Select time range' area of the left hand navigation area.

6.5. Links to more details

Many pages contain names of clusters, nodes, switches or other devices. Most of these names are links to device-specific pages providing more details about this particular device. For example, going from the cluster page to a dedicated node page is as easy as clicking on the particular node icon.

Chapter 7. *GridMonitor* GUI: Overview page

The overview page shows the overall state of all configured clusters. For each cluster, a brief summary is displayed:

- Basic cluster parameters like total cluster load, how many nodes and CPUs are available, how many nodes are down and an average node temperature.
- The current overall load - loadbar diagram.
- A load history diagram.
- How many nodes are available/down/idle/in use.
- How many nodes and which ones have events pending.
- How many jobs are currently being executed.

Information about currently being executed jobs is only available, if these jobs are started up using *ParaStation* and `ps4_host` entry within the *collector* configuration is set up.

Chapter 8. *GridMonitor* GUI: Cluster pages

The cluster pages show different aspects of cluster-related information for a dedicated cluster system. The pages may be selected using the links within the `Node Details` navigation area.

8.1. *GridMonitor* GUI: Cluster overview page

The cluster overview page shows the overall state of a particular cluster. A brief information summary is displayed:

- A summary of the current load of all nodes using a loadbar. See loadbar for details.
- Details of the current node states. Shown as colored icons.
- An overall load history diagram providing average and maximum load of all nodes.
- A current node load diagram giving the actual load value of each cluster node. This diagram uses bar charts for small clusters and radar charts for larger systems.
- An event list for the specified period of time.
- A job list providing information about how many and which *ParaStation* jobs are currently executing.

The picture shown in Figure 8.1, “Cluster overview” is an example for a typical cluster overview page.

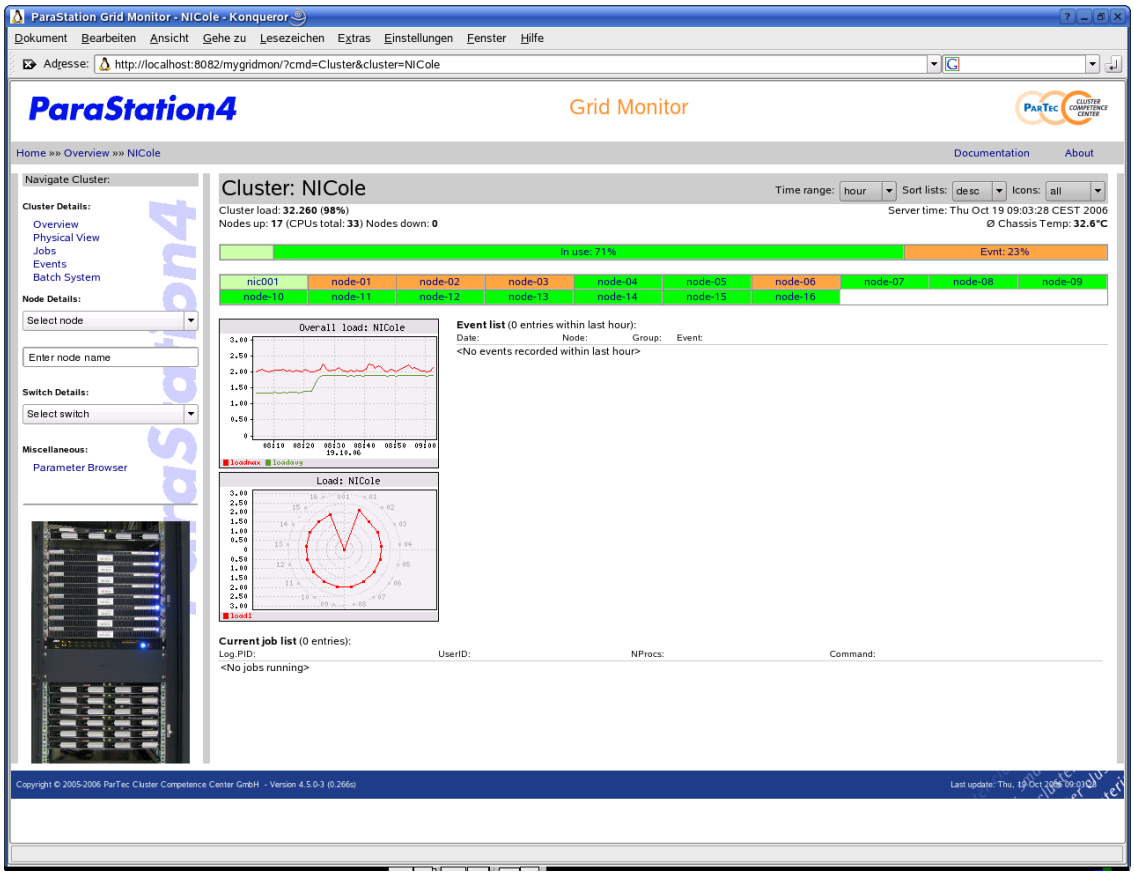

Figure 8.1. Cluster overview

The example shows a cluster with four nodes marked as having pending events. However, the event list is empty. This is ok, as the events still pending on the nodes happened not within the timeframe selected for the event list.

8.2. *GridMonitor* GUI: Cluster physical view page

This page gives information about a particular cluster like the cluster overview page, but uses a view oriented on the physical layout of the system. As shown in Figure 8.2, “Cluster physical overview”, the nodes are shown as they appear in real life within their enclosures. In addition, each node and switch shows its state using the corresponding color.

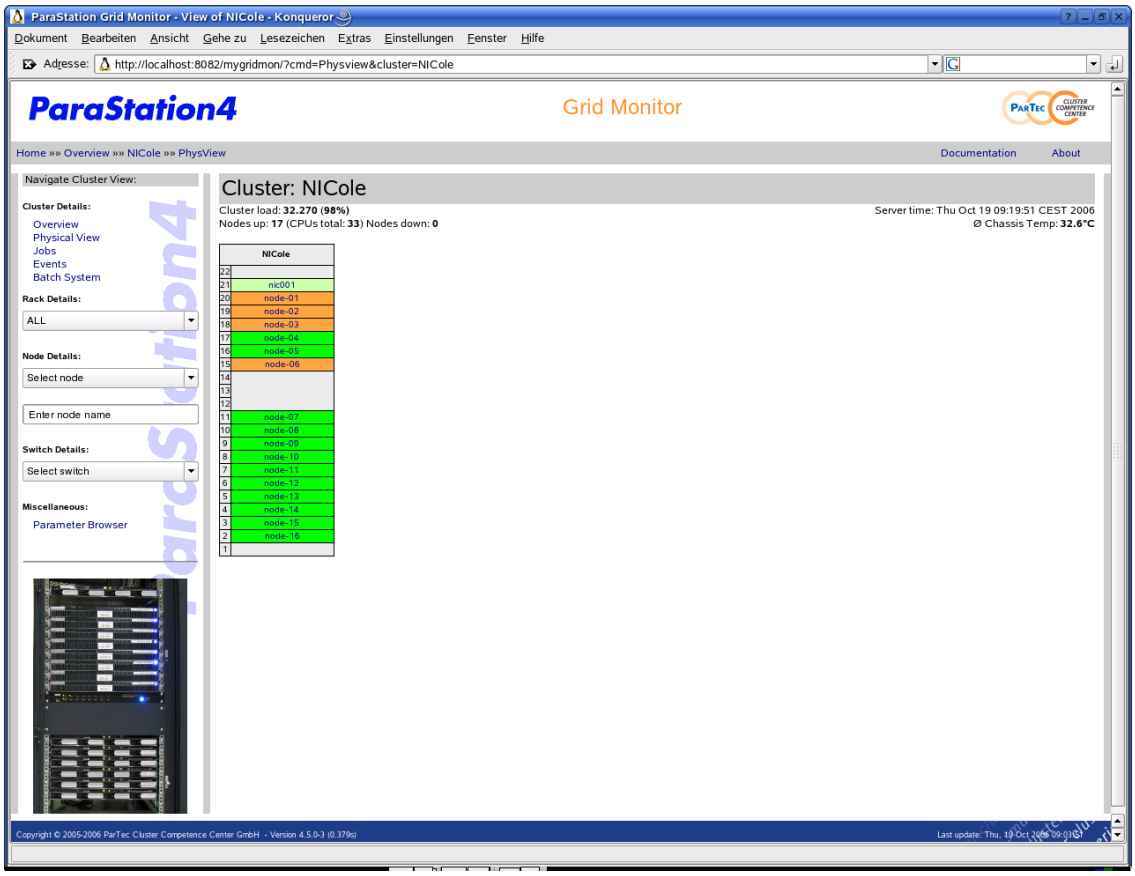

Figure 8.2. Cluster physical overview

Using the '`Rack Details`' pull-down menu of the left hand navigation area, each rack of a cluster can be selected, provided more than a single rack is configured. In addition, pre-configured groups of racks can be selected. For details on how to configure rack groups, refer to Section 4.4.2, “Configuring *GridMonitor* GUI physical view”.

8.3. *GridMonitor* GUI: Cluster events page

The cluster events page lists all or all selected events for a particular cluster and a selected period of time.

To narrow the number of events displayed, a pattern can be entered within the left hand navigation area. This pattern is searched for in the `Node`, `Group` and `Event` column.

In addition, the start and end time for the list of shown events may be selected using the `from:` and `to:` input boxes of the left hand navigation area. Its also possible to define a time range ending now by using the '`Time range`' pull-down menu.

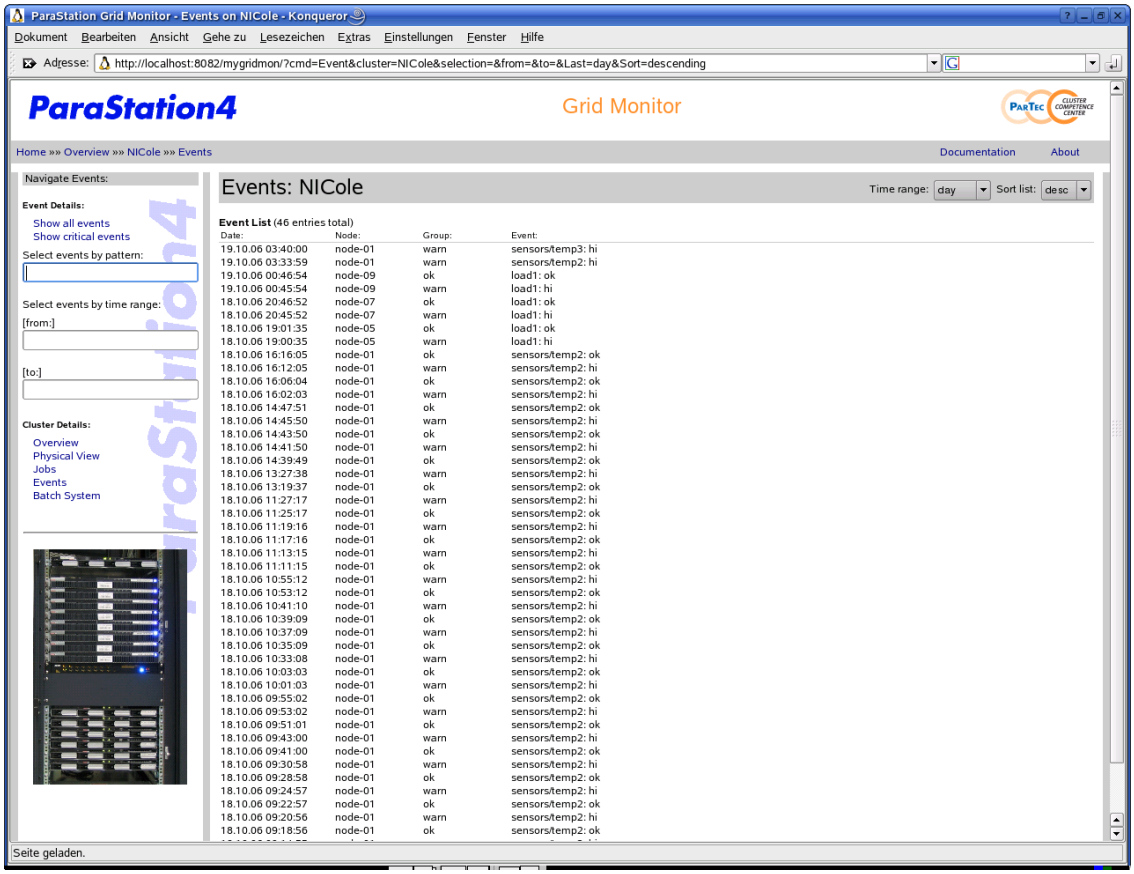

Figure 8.3. Cluster events list

8.4. *GridMonitor* GUI: *ParaStation* jobs page

Using the *ParaStation* page, more detailed information about currently active *ParaStation* jobs can be displayed.

Information about jobs being currently executed on this cluster is only available for those jobs started up using *ParaStation*. The `ps4_host` entry within the *collector* configuration must be set up, too.

8.5. *GridMonitor* GUI: Batch queuing system information

Using the batch system information pages, more detailed information about known queues and jobs to the batch queuing system can be displayed.

The Queues page shows an overview of all known queues within the configured batch queuing system, as well as the number of currently known (= queued, running, held, waiting and exited) jobs per queue.

Information can be sorted by queue name, ascending or descending.

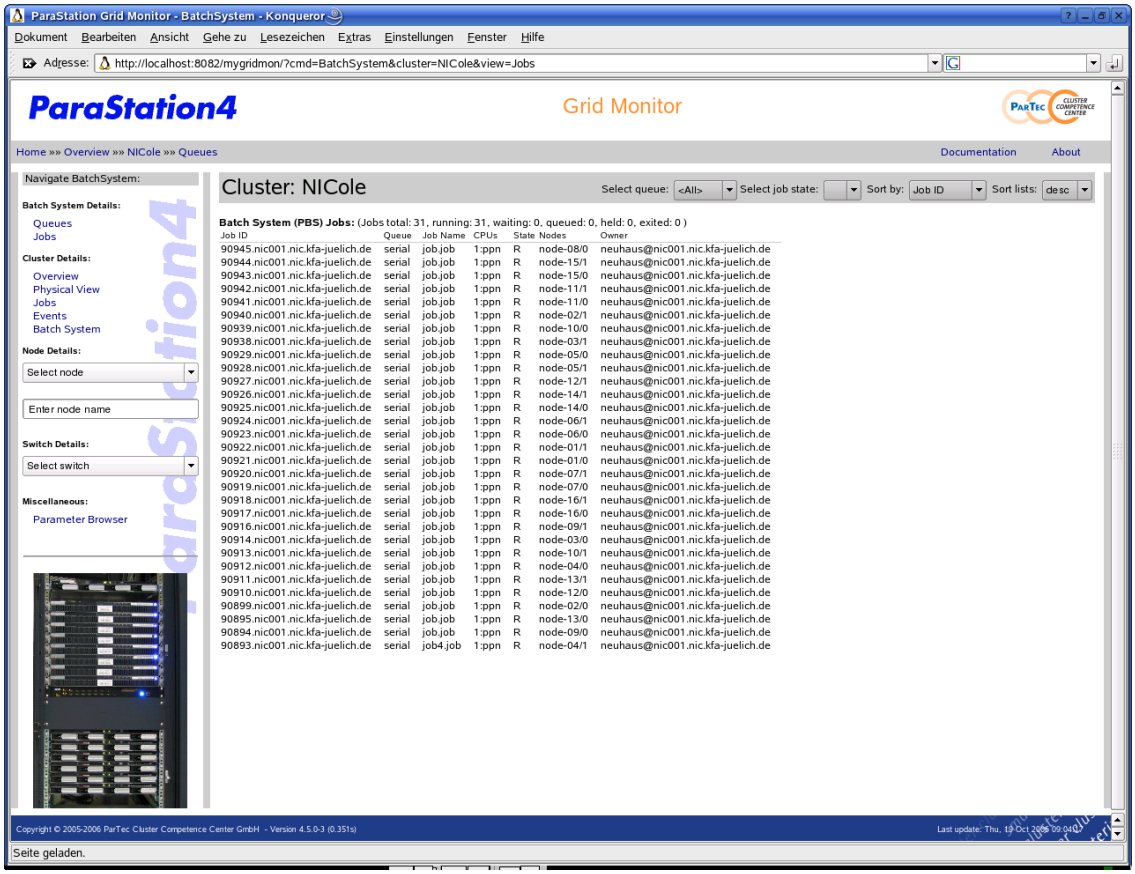

Figure 8.4. Batch system jobs list

The Jobs page lists all currently known jobs in detail, as shown in Figure 8.4, “Batch system jobs list”.

The list can be narrowed by selecting a dedicated queue name or a job state using the appropriate pull-down menu in the header area. In addition, the list can be sorted by all available columns, like `job ID`, `queue`, `job name`, `state`, `nodes` or `owner`. The list may also be sorted ascending or descending using the '`Sort`' pull-down menu.

Chapter 9. *GridMonitor* GUI: Node pages

The Node pages show different aspects of node-related information. These pages can be selected using the links within the '`Node Details`' navigation area of the left hand navigation area.

9.1. *GridMonitor* GUI: Node overview page

The node overview page shows the overall state of a particular node. A brief information summary is displayed:

- The basic node parameters like current load, available and used memory, current node time, fan speed and temperature.
- Diagrams for load history, current temperatures and fan speeds.
- How many and which events are pending for this particular node.
- Events list for the specified period of time.

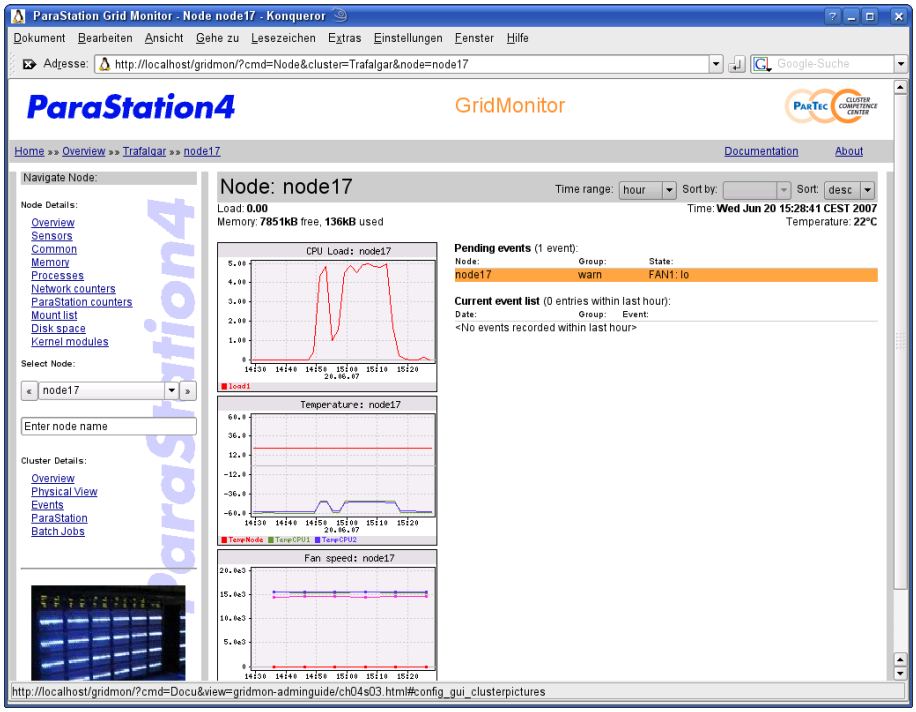

Figure 9.1. Node overview

9.2. *GridMonitor* GUI: Node sensors page

The node sensors page displays information about the physical environment of a node, typically:

- Three different temperatures called `TempCPU1`, `TempCPU2` and `TempNode`.
- Four different fan speeds called `FAN1` up to `FAN4`.

More parameters may be available depending on the configuration of the virtual sensors. Refer to Section 4.2.7, “Configuring the *collector* – step 6” for details.

In addition, history charts for `TempCPU1`, `TempCPU1` and `TempNode` as well as for all fans (`FAN1`, `FAN2`, …) are shown.

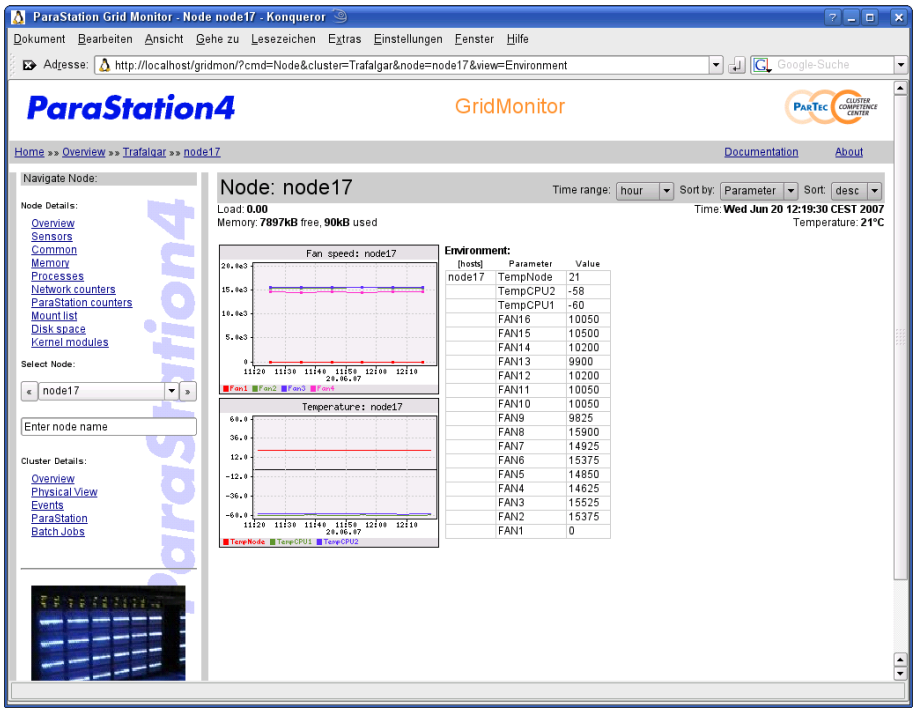

Figure 9.2. Node sensors

Availability of the history charts depends on the configuration of the virtual sensors parameters, like `TempCPU1` or `FAN1`, within the *collector* configuration. Refer to Section 4.2.7, “Configuring the *collector* – step 6” for details.

All available sensors may be shown using the parameter browser. Go to `ipmi>sdr>list` or `hosts>sensors`. See section Chapter 11, *GridMonitor GUI: Parameter browser page* for details.

9.3. *GridMonitor* GUI: Node common page

Within the node common page, basic information about a particular node is shown:

- The official node name.
- The system type, hostname, kernel version, build date and architecture, as reported by **uname**.
- The number of CPUs.
- The amount of physical memory installed (reported in kB).
- The amount of swap space configured (reported in kB).
- The host's uptime, reported in seconds.

9.4. *GridMonitor* GUI: Node memory page

The node memory page displays two history charts for `free memory` as well as `free swap space`.

Availability of this information depends on the proper configuration of the *collector*. See Section 4.2.10, “Configuring the *collector* – step 9” for details. By default, these parameters are not stored into the database.

9.5. *GridMonitor* GUI: Node process list page

This node page shows the actual process list of the particular node.

9.6. *GridMonitor* GUI: Node network page

The node network page shows the current network device table of the node, including various counters.

9.7. *GridMonitor* GUI: Node mount page

The mount page shows the current mount table of the node.

9.8. *GridMonitor* GUI: Node disk space page

This page displays the disk usage counters of locally available file systems.

9.9. *GridMonitor* GUI: Node ParaStation counters page

Within this page, various *ParaStation* counters are shown.

9.10. *GridMonitor* GUI: Node Infinipath counters and statistics page

Within this page, various Infinipath counters are shown.

This page may only be available if Infinipath is installed on the system.

9.11. *GridMonitor* GUI: Node kernel modules page

The node kernel modules page lists all loaded kernel modules for the particular node, including size, use count and used by information.

Chapter 10. *GridMonitor* GUI: SNMP (switch) pages

The SNMP pages show information about devices managed using the SNMP protocol, e.g. network switches.

The pages may be selected using the links within the '`Switch Details`' navigation area of the left hand navigation area.

10.1. *GridMonitor* GUI: Switch Overview page

The Overview page shows information about the input and output counters of each switch port. In addition, error and discard counters are reported.

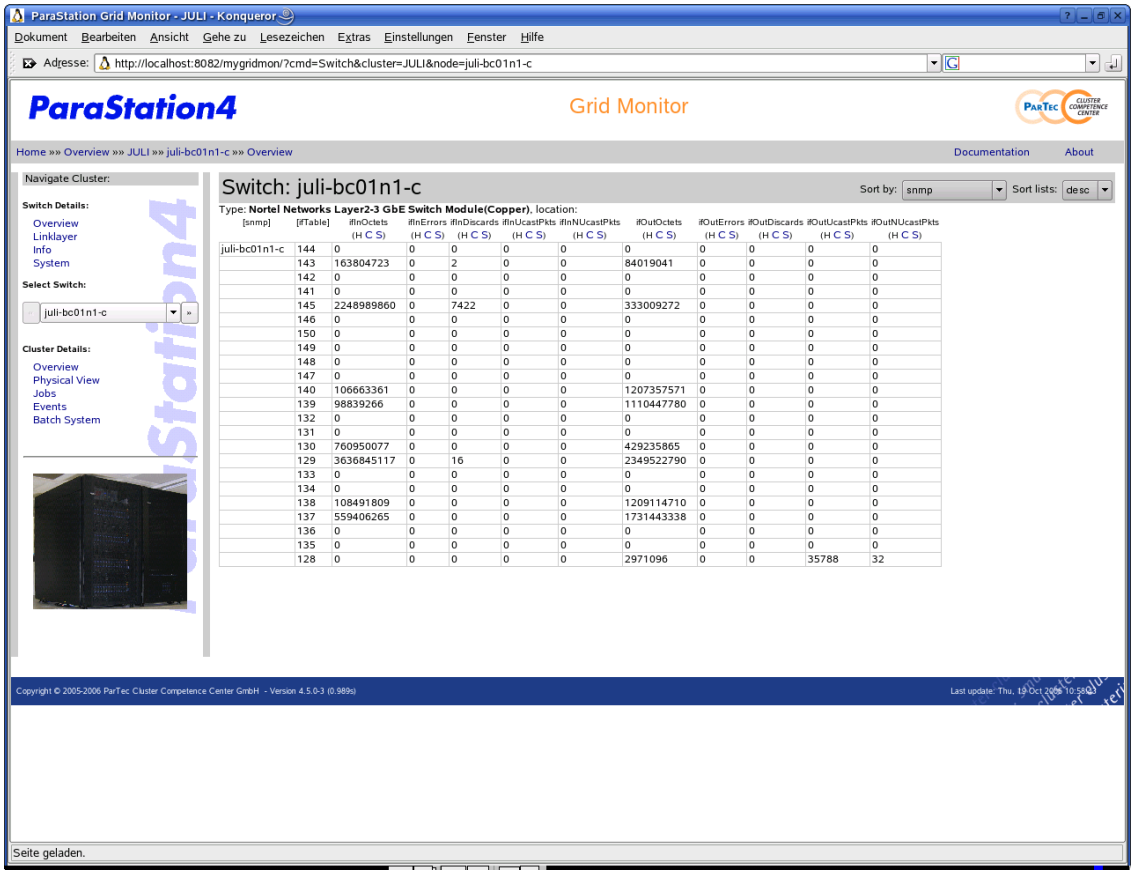

Figure 10.1. Switch overview

10.2. *GridMonitor* GUI: Switch LinkLayer page

The LinkLayer page shows information about the link layer of each port. The admin- and operater state, the physical address (MAC address) and the MTU of each port are reported.

10.3. *GridMonitor* GUI: Switch Info page

The Info page shows information about the description (=name), type and speed of each port.

10.4. *GridMonitor* GUI: Switch System page

The System page shows generic information about the switch, like type, location, name and contact information, and the uptime.

Chapter 11. *GridMonitor* GUI: Parameter browser page

The parameter browser of the *GridMonitor* is a generic tool to display each particular parameter known to the *collector*. Parameters are organized in a hierarchical way by using table and record entries. Each table provides indices and parameters, whereas each record only holds parameters. These parameters may be scalar parameters, like integers, floats or strings, or again may be tables or records. Examples for top-level indices are host names or switch names.

Parameters may be provided by each data source known to the *collector*, therefore the parameter browser is for example also a SNMP (or MIB) browser. You can go down all the way through all the MIBs known to a device or group of devices. The data source is transparent to the parameter browser, therefore the user does not have to take care if changing from one source to another.

The selected data is shown using a matrix layout, similar to a calculation sheet. It is organized in rows and columns, where the scalar parameters and indices show up in columns.

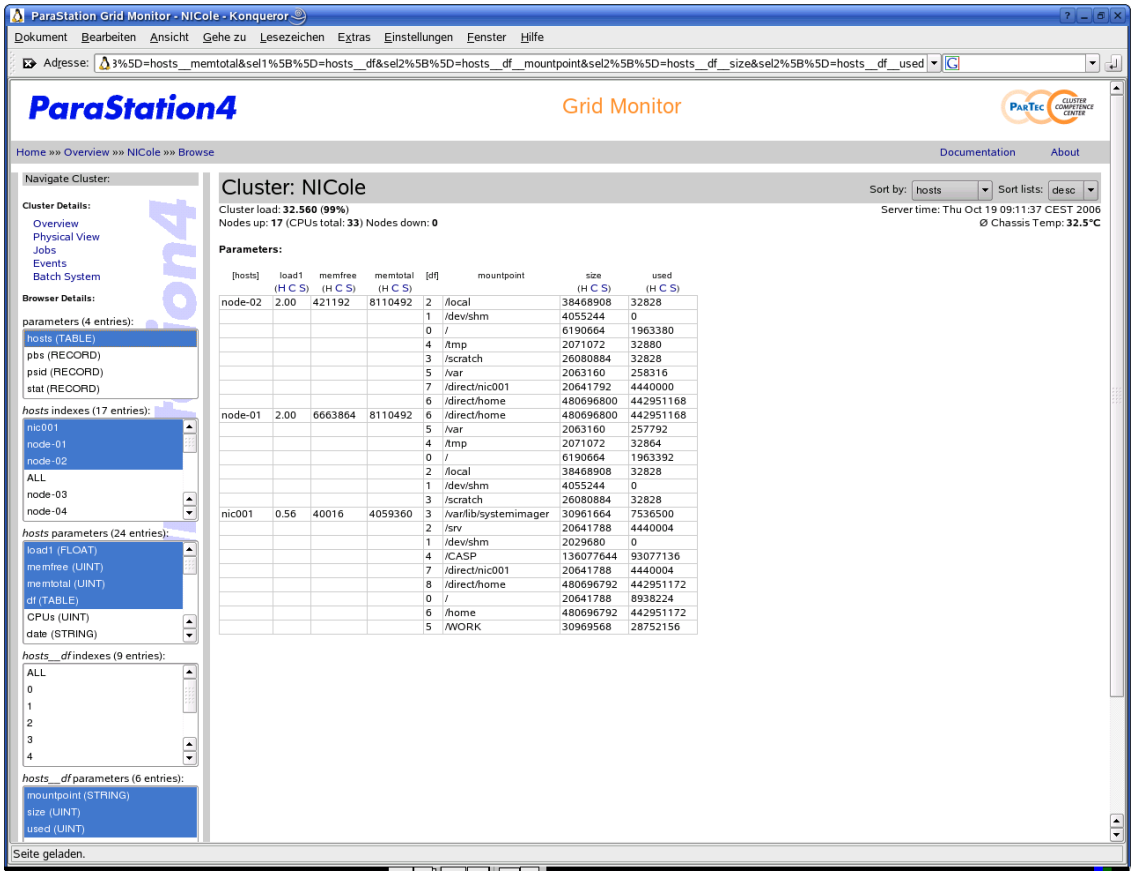

Figure 11.1. Parameter browser

11.1. *GridMonitor* GUI: Parameter browser select boxes

Using the left hand navigation area, one or more parameters may be selected by repeatedly using index and parameter select boxes as shown in Figure 11.1, “Parameter browser”. Selections are made by selecting the appropriate select box entry with the mouse. Multiple selections within a box can be made by holding down the control key.

When selecting a table (`TABLE`) or record (`RECORD`) type parameter, all available indices and parameters for this table or record are shown, after submitting the selection by pressing the `Submit` button.

When selecting scalar parameters like integers (`INT`, `UINT`) or strings (`STRING`), those parameters are shown using all selected indices, e.g. nodes.

Selections can be narrowed by selecting only interesting indices, e.g. selecting only those nodes from the node list, which are currently of interest.

11.2. *GridMonitor* GUI: Parameter browser sorting

The shown parameter browser table can be sorted by each column represeting a parameter or index, using the '`Sort by`' pull-down menu. The columns may be sorted ascending or descending using the '`Sort lists`' pull-down menu.

11.3. *GridMonitor* GUI: Parameter browser diagrams

Each numerical column may be visualized using different diagram types, like bar charts ('`C`') and radar charts ('`S`'). If the parameter listed in the particular column is also stored into the database, a history chart ('`H`') is available, too. For more information on diagrams, refer to Section 6.4, “Diagrams”.

Part III. Additional information

The sections in Part III provide more technical information on the *ParaStation GridMonitor*.

Chapter 12. Troubleshooting

This chapter provides some hints to problems seen while installing or using the *ParaStation GridMonitor*. For addition help, please contact <support@par-tec.com>.

12.1. Problem: *GridMonitor* GUI shows no values for temperatures or fan speeds

Description: when clicking on node-specific pages, no or not all values for node temperatures or fan speeds are shown.

Solution: when using the lmsensors package to read sensors data, check for the **sensors** output on each node. The `lmsensors` package must be installed and configured to return correct values for certain parameters.

The output of **sensors** should look like this:

```
VCore 1:   +1.42 V  (min =  +1.42 V, max =  +1.57 V)
VCore 2:   +3.31 V  (min =  +1.42 V, max =  +1.57 V)
+3.3V:     +3.26 V  (min =  +3.14 V, max =  +3.47 V)
+5V:       +4.95 V  (min =  +4.76 V, max =  +5.24 V)
+12V:     +12.04 V  (min = +10.82 V, max = +13.19 V)
-12V:     -11.46 V  (min = -13.18 V, max = -10.80 V)
-5V:       -2.13 V  (min =  -5.25 V, max =  -4.75 V)
V5SB:      +5.38 V  (min =  +4.76 V, max =  +5.24 V)
VBat:      +3.84 V  (min =  +2.40 V, max =  +3.60 V)
fan1:     5314 RPM  (min = 2848 RPM, div = 2)
fan2:     5075 RPM  (min = 2848 RPM, div = 2)
fan3:     3409 RPM  (min = 1424 RPM, div = 4)
temp1:       +45C  (high =   +65C, hyst =   +60C)
temp2:     +25.0C  (high =   +85C, hyst =   +80C)
temp3:     +24.5C  (high =   +85C, hyst =   +80C)
```

Example 12.1. Sample sensors output

If your system does not support at least 2 fans (fan1, fan2) and 3 temperatures (temp1, temp2, temp3), the missing values are reported as '0'. This may lead to unexpected events and alarms.

When using IPMI to read sensor data, check the command **ipmitool** for proper sensor data output:

```
master: # ipmitool -I lan -H node01-bmc -U root -P root sdr list
Temp             | -58 degrees C     | ok
Temp             | -56 degrees C     | ok
Temp             | 40 degrees C      | ok
Temp             | 40 degrees C      | ok
Ambient Temp     | 20 degrees C      | ok
CMOS Battery     | 0x00              | ok
…
```

Example 12.2. Sample IPMI output

Check for proper mapping of real to virtual sensors, see Section 4.2.7, “Configuring the *collector* – step 6” for details.

12.2. Problem: negative CPU temperatures shown

Description: CPU temperatures are reading negative values.

This is absolutely perfect for Intel® Core 2 Duo microprocessors, as these CPUs no longer provide real temperature values, but a "thermal health indicator" varying from -100 (cool) to 0 (hot).

More information on this and further links providing more detailed background information may be found on http://en.wikipedia.org/wiki/Platform_Environment_Control_Interface.

12.3. Problem: history charts report errors

Description: history diagrams within the *GridMonitor* GUI show no values for a particular parameter, but errors like NOTE: no data for 'memfree'.

Verify that the *collector* is configured to save the reported parameter to the database. For example, to save the `memfree` parameter periodically to the database, an entry like

```
parameter("cluster/*/hosts/*",
  …
  memfree = {
    save_history = true,
    poll = 600
  }
  …
```

must be present in the configuration file `/etc/pscollect/cluster.conf`. How to configure these parameters is described in Section 4.2.10, “Configuring the *collector* – step 9”.

12.4. Problem: *GridMonitor* GUI shows no batch jobs

Description: after configuring a Torque batch system server (see Section 4.2, “Configuring the *collector* – step by step” for details), the Queues page indicates running jobs within the `Run` column, but the corresponding Jobs page does not list any jobs.

Solution: within the Torque server configuration, the `query_other_jobs` parameter must be set to `true`, otherwise the *GridMonitor* agent running as user `pscd` is not allowed to read detailed job information.

See qmgr(1) and pbs_queue_attributes(7b) for details.

12.5. Problem: empty '`Select Queue`' menu

Description: within the Jobs page of the Batch System area, the '`Select queue`' pull-down menu does not list any queue names beside `All`.

Solution: currently, the names of all available queues must be configured within the `.defaults` section of the *GridMonitor* GUI configuration file. See Section 4.4, “Configuring the graphical user interface (*GridMonitor* GUI)” for details.

12.6. Problem: no *ParaStation* job list shown

Description: within the Overview page of the cluster area, no *ParaStation* job list is shown at the bottom of the page, although jobs are running.

Solution: first of all, check if the `ps4_host` entry in the configuration file `pscollect.conf` is correct. It could point to any node in the cluster running a *ParaStation* daemon.

If *ParaStation* isn't used for job management, no information about currently active jobs is available on the cluster Overview page. However, information about queues and jobs is always available using the Batch Queue pages.

Next, on 64bit architetures, check if `/opt/parastation/bin/pscollect_psi` will resolve all required libraries:

```
$ ldd /opt/parastation/bin/pscollect_psi
libpse.so => /opt/parastation/lib/libpse.so …
libpsi.so => /opt/parastation/lib/libpsi.so …
libm.so.6 => /lib64/libm.so.6 …
libnetsnmp.so.10 => /usr/lib64/libnetsnmp.so.10 …
libc.so.6 => /lib64/libc.so.6 …
/lib64/ld-linux-x86-64.so.2 …
```

If either `libpse.so` or `libpsi.so` could not be resolved, add a symbolic link from the directory `/opt/parastation/lib64` to `/opt/parastation/lib`:

```
# cd /opt/parastation
# ln -s lib64 lib
```

On 64bit environments, recent versions of *ParaStation* will store libraries in the directory `/opt/parastation/lib64`.

Reference Pages

This appendix lists all reference pages related to the *ParaStation GridMonitor*.

pscollect

pscollect — the *ParaStation GridMonitor* data collecting process.

Synopsis

pscollect [-v*level*] [-d] [-l] [-f *filename*] [-c *command*] [-p *portno*] [-h *hostname*] [-t *sec*] [-?]

Description

After reading the configuration file, **pscollect** starts an agent on every configured node. Upon request of a client, **pscollect** retrieves data from one or more agent(s) or the database and returns it to the client. Data may also be read periodically by the **pscollect**.

Any data read from an agent may also be stored in the database, depending on the configuration, by adding time stamps. It will be cached within **pscollect**, too, depending on the parameter. In addition, parameters may be monitored against minimum or maximum limits.

New values may be calculated, based on values of parameters read from agents.

Options

`-c, --command=`*`command`*
: Run initial command at startup.

`-v, --verbose`
: Be more verbose.

`-f` *`configfile`*`, --conf=`*`configfile`*
: Use configuration file *`configfile`*.

`-p` *`portumber/name`*`, --port=`*`portnumber/name`*
: Listen on TCP port specified by its *`portumber`* or service *`name`*

`-h` *`hostname`*`, --host=`*`hostname`*
: Accept connections from host *`hostname`* only (default: accept from any host).

`-t` *`sec`*`, --timeout=`*`sec`*
: Timeout for blocked TCP connections (default: 60 sec).

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

`--usage`
: Show a usage message.

Files

`/etc/pscollect/cluster.conf`
: Global configuration file for the *collector* process.

See also

psvalue(8)

psvalue

psvalue — the *ParaStation GridMonitor* default agent for compute nodes

Synopsis

psvalue

Description

The psvalue script is started by the *collector* using ssh on each configured node. It is returning local parameters upon request of the *collector*.

See also

pscollect(8)

psget

psget — retrieve data from the collecting process.

Synopsis

psget [-p portno] [-h host] [-?] [*host* [*port*]]

Description

The **psget** command connects to a *collector* (**pscollect**) and allows reading of data.

This tool is intended for debugging purposes only!

Options

`-p portno`, `--port=portno`
: Connect to port or service.

`-h host/IP address`, `--host=host/IP address`
: Connect to host.

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

`--usage`
: Show a usage message.

See also

pscollect(8)

Glossary

Agent	An agent is a process startet or at least contacted by the *collector*. It returns information upon requests made by the *collector*. An agent may run on the collector node and retrieve data using network connections or may run on a remote node to retrieve data local to this node. Currently, agents for general node parameters (`psvalue`), Torque (`psvalue_pbs`), the *ParaStation* daemon (`pscollect_psi`), Torque and *ParaStation* accounting (`psvalue_acc`) and BMCs using IPMI (`psvalue_ipmi`) are available. The SNMP agent is built-in to *collector*
BMC	A baseboard management controller (BMC) is a small controller within a server enabling system independent information retrieval and node management. BMCs typically provide a LAN interface using the IPMI protocol to allow for example reseting of the node, reading sensors parameters, FRU parameters, etc. BMC are often refered to as IPMI controllers.
Collector	A process collecting and managing all the information within one (or more) clusters. It is running on the collector node, which is typically the cluster master node. Only one instance of the *collector* is required within a cluster. The *collector* gathers information using agents. Within this document, the software package holding all necessary files to run the *collector* process is sometimes called collector, too.
Collector node	This is the system where the *collector* is running. For smaller cluster systems, this is often the frontend node, for large clusters a dedicated master node should be used. The collector node must not necessarily be part of the cluster, in many installations its a dedicated server running the *collector* and the webserver providing the *GridMonitor* GUI.
Cluster node	An independant hardware entity within a cluster. Cluster nodes may be dedicated compute nodes, frontend nodes, master nodes, fileservers, I/O-nodes or other management nodes like collector nodes.
Frontend node	A dedicated cluster node, intended for user login, compiling applications, launching jobs, etc. Within smaller clusters, this node is often used as a management node, too.
Compute node	Dedicated servers within the cluster running the compute jobs.
Management node	A system (or cluster node) within a cluster not or not only used for computing purposes, but for management tasks. This node typically runs services required clusterwide like NIS, NFS server, batch queuing system, *GridMonitor*, etc. Within larger clusters, dedicated management nodes are available, like batch queuing system servers or file servers. The management node is often refered as master node.
GridMonitor	A software bundle to monitor various kinds of activities within a cluster. The *GridMonitor* consists of a collector, agents and a *GridMonitor* GUI.

GridMonitor GUI

Graphical user interface for the *GridMonitor*. The *GridMonitor* GUI is based on web pages, generated by an Apache webeserver using PHP scripts. For more details, refer to Chapter 6, *GridMonitor GUI: Navigation*.

IPMI

The Intelligent Platform Management Interface (IPMI) defines a set of common interfaces to computer hardware and firmware which system administrators can use to monitor system health and manage the system.

Loadbar

The loadbar is a small bar showing the status of the cluster. Each section of the bar shows the percentage of the nodes in status idle (light green), in use (green), event pending (orange) and unavailable (red).

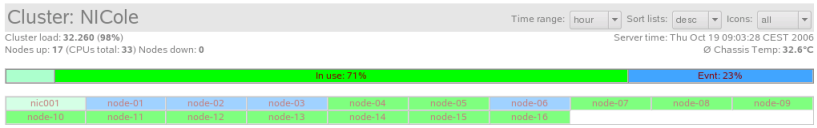

Figure 15. Loadbar

The loadbar shown in Figure 15, “Loadbar” shows a cluster having a small fraction of idle nodes, 71% of the nodes being in use and 23% of the nodes having a pending event. No dead nodes (red) are shown. Be aware that nodes with pending events may also run jobs.

Clicking on the loadbar will reload the current page providing an icon list showing all nodes with the selected status, e.g. all dead nodes. Refer also to Chapter 8, *GridMonitor GUI: Cluster pages*.

Parameter type system

Beside the actual data, the *collector* also provides information about the type of available data. This is called the `parameter type system`. Using this system, it's easy for a *GridMonitor* GUI to construct dynamic selection boxes without actually reading the data and therefore wasting network bandwidth and compute cycles. The parameter type system also enables a *GridMonitor* GUI to dynamically include new parameters without modifying the scripts or page layout.

The `parameter type system` is for internal use only and does not show up to the user or administrator.

SNMP

`Simple Network Management Protocol`, implemented for many types of network devices.

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, Fast Ethernet or Gigabit Ethernet is used for this purpose.

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, administration and data network are often combined into a single interconnect.

Node states

Each node has an associated state, which describes the node from an administrator's point of view. Within the *GridMonitor* GUI, four different states are currently defined:

State	Description
idle	The node is currently available and not used.
in use	The node is currently available and used.
event pending	The node is currently available and may be in use, but there is at least one event pending.
dead	The node is currently not available.

Table 2. Node states

Icon area

Within the cluster overview page, the icon area shows icons for all or a selected set of the cluster nodes. These icons are colored, indicating the current state of the respective node. See Node states for details about node states.

The nodes shown in the icon area can be defined by using the `icon` pull-down menu. See Section 6.3, “View configuration” for details.