

HP AlphaServer SC RMS Reference Manual

Quadrics Ltd. **Document Version 11 - Nov 14th 2003 - AA-RLAZE-TE**

The information supplied in this document is believed to be correct at the time of publication. No liability is assumed for its use or for the infringements of the rights of others resulting from its use. No license or other rights are granted in respect of any rights owned by any of the organizations mentioned herein.

This document may not be copied, in whole or in part, without the prior written consent of Quadrics Ltd.

Copyright 1998,1999,2000,2001,2002,2003 Quadrics Ltd.

The specifications listed in this document are subject to change without notice.

Intel is the registered trademark of Intel Corporation.

AMD is the trademark of Advanced Micro Devices, Inc.

UNIX® is a registered trademark of The Open Group.

TotalView and Etnus are registered trademarks of Etnus LLC.

All other product names mentioned herein may be trademarks of their respective companies.

The Quadrics Ltd. (Quadrics) web site can be found at:

```
http://www.quadrics.com/
```

Quadrics' address is:

One Bridewell Street
Bristol
BS1 2AA
UK

Tel: +44-(0)117-9075375
Fax: +44-(0)117-9075395

Circulation Control: None

Document Revision History

Revision	Date	Author	Remarks
4	Jun 2000	RMC	AlphaServer SC V1 Product Release
7	June 2001	DR	AlphaServer SC V2 Product Release
8	April 2002	DR	AlphaServer SC V2.4a Product Release
9	Sept 2002	DR, RMC	HP AlphaServer SC V2.5 Product Release
11	Nov 2003	DR	HP AlphaServer SC V2.6 Product Release

Contents

1 Introduction **1-1**

1.1 Scope of Manual 1-1
1.2 Audience 1-1
1.3 Using this Manual 1-1
1.4 Related Information 1-3
1.5 Location of Online Documentation 1-3
1.6 Reader's Comments 1-3
1.7 Conventions 1-3

2 Overview of RMS **2-1**

2.1 Introduction 2-1
2.2 The System Architecture 2-1
2.2.1 Nodes 2-1
2.3 The Role of the RMS 2-3
2.3.1 The Structure of the RMS 2-4
2.3.2 The RMS Daemons 2-4
2.3.3 The RMS Commands 2-5
2.3.4 The RMS Database 2-6
2.4 RMS Management Functions 2-7
2.4.1 Allocating Resources 2-7
2.4.2 Scheduling 2-8
2.4.3 Access Control and Accounting 2-10

2.4.4 RMS Configuration 2-10

3 Parallel Programs Under RMS **3-1**

3.1 Introduction 3-1
3.2 Resource Requests 3-2
3.3 Loading and Running Programs 3-3

4 RMS Daemons **4-1**

4.1 Introduction 4-1
4.1.1 Startup 4-2
4.1.2 Log Files 4-2
4.1.3 Status 4-2
4.2 The Database Server 4-2
4.3 The Machine Manager 4-3
4.4 The Partition Manager 4-3
4.5 The Switch Manager 4-5
4.6 The Transaction Log Manager 4-6
4.7 The Event Manager 4-6
4.8 The Process Manager 4-7
4.9 The RMS Daemon 4-7

5 RMS Commands **5-1**

5.1 Introduction 5-1
allocate(1) 5-3
msqladmin(1) 5-8
nodestatus(1) 5-11
prun(1) 5-12
rcontrol(1) 5-27
rinfo(1) 5-41
rmsarchive(1) 5-49
rmsbackup(1) 5-52
rmsbuild(1) 5-54

rmsctl(1) 5-57
rmsexec(1) 5-59
rmshost(1) 5-61
rmspost(1) 5-62
rmsquery(1) 5-63
rmstbladm(1) 5-66
swctrl(1) 5-69

6 Access Control, Usage Limits and Accounting 6-1

6.1 Introduction 6-1
6.2 Users and Projects 6-1
6.3 Access Controls 6-2
6.3.1 Access Controls Example 6-3
6.4 How Access Controls are Applied 6-4
6.4.1 Memory Limit Rules 6-4
6.4.2 Priority Rules 6-6
6.4.3 CPU Usage Limit Rules 6-7
6.5 Accounting 6-7

7 RMS Scheduling 7-1

7.1 Introduction 7-1
7.2 Scheduling Policies 7-1
7.3 Scheduling Constraints 7-2
7.4 What Happens When a Request is Received 7-3
7.4.1 Memory Limits 7-5
7.4.2 Swap Space 7-6
7.4.3 Draining Jobs 7-6
7.4.4 Time Slicing 7-7
7.4.5 Suspend and Resume 7-7
7.4.6 Idle Time 7-7
7.4.7 Complex Resource Requests 7-7

8 Event Handling **8-1**

8.1 Introduction 8-1
8.1.1 Posting Events 8-2
8.1.2 Waiting on Events 8-2
8.2 Event Handling 8-3
8.3 List of Events Generated 8-4
8.4 Extending the RMS Event Handling Mechanism 8-10

9 Setting up RMS **9-1**

9.1 Introduction 9-1
9.2 Installation Planning 9-1
9.2.1 Node Names 9-2
9.3 Setting up RMS 9-2
9.3.1 Starting RMS 9-2
9.3.2 Initial Setup with One Partition 9-3
9.4 Inter-Process Communication in RMS 9-4
9.5 Local Customization of RMS 9-4
9.5.1 Partition Startup 9-5
9.5.2 Prologue and Epilogue Scripts 9-5
9.5.3 Processor Set Support 9-6
9.5.4 Core File Handling 9-6
9.5.5 Event Handling 9-7
9.5.6 Switch Manager Configuration 9-7
9.5.7 Controlling Allocatable Memory 9-8
9.5.8 Rate of RMS Status Updates 9-9

10 Day-to-Day RMS Operation **10-1**

10.1 Introduction 10-1
10.2 Periodic Shift Changes 10-1
10.3 Summarizing Accounting Data 10-2
10.4 Archiving Data from the RMS database 10-3
10.5 Backing Up the RMS Database 10-5

10.6 Restoring the Database 10-7
10.7 Troubleshooting 10-7
10.8 Configuring Nodes Out 10-8
10.9 Cleanup 10-9
10.10 Draining Jobs From Nodes 10-10
10.11 Log Files 10-11
10.11.1 Creating new log files 10-12

11 The RMS Database 11-1
11.1 Introduction 11-1
11.2 General Information about the Tables 11-1
11.3 Table Definition Files 11-3
11.3.1 Table Definition 11-3
11.3.2 Field Definition 11-4
11.3.3 Index Definition 11-5
11.3.4 Sequence Definition 11-5
11.3.5 Default Definition 11-6
11.4 Categories of Table 11-6
11.4.1 Configuration of Nodes 11-6
11.4.2 Configuration of the Network 11-7
11.4.3 User Details 11-7
11.4.4 Operational State 11-8
11.4.5 Internal Tables 11-8
11.5 Listing of Tables 11-8
11.5.1 The Access Controls Table 11-8
11.5.2 The Accounting Statistics Table 11-9
11.5.3 The Archive Tables Table 11-11
11.5.4 The Attributes Table 11-12
11.5.5 The Backups Table 11-19
11.5.6 The Elans Table 11-20
11.5.7 The Elites Table 11-21

11.5.8 The Event Handlers Table 11-21
11.5.9 The Events Table 11-22
11.5.10 The Fields Table 11-23
11.5.11 The Installed Components Table 11-24
11.5.12 The Jobs Table 11-25
11.5.13 The Link Errors Table 11-26
11.5.14 The Module Types Table 11-27
11.5.15 The Nodes Table 11-28
11.5.16 The Node Statistics Table 11-31
11.5.17 The Partitions Table 11-32
11.5.18 The Projects Table 11-34
11.5.19 The Resources Table 11-35
11.5.20 The Scheduler Log Table 11-36
11.5.21 The Servers Table 11-37
11.5.22 The Services Table 11-38
11.5.23 The Software Products Table 11-39
11.5.24 The Status Table 11-40
11.5.25 The Switch Boards Table 11-41
11.5.26 The Switch Modules Table 11-42
11.5.27 The Terminal Servers Table 11-43
11.5.28 The Terminal Server Types Table 11-43
11.5.29 The Transactions Table 11-44
11.5.30 The Tree Status Table 11-45
11.5.31 The Unidentified MACs Table 11-45
11.5.32 The Users Table 11-45

A HP AlphaServer SC Interconnect Terms A-1

A.1 Introduction A-1
A.2 Capabilities A-1
A.3 Network Topology A-1
A.3.1 Broadcast Trees A-4

A.3.2 Clock Source A-5
A.4 Database Representation A-5
A.5 Control Interface A-5
A.6 Link States A-5
A.7 Link Errors A-6

B RMS Status Values **B-1**
B.1 Overview B-1
B.2 Generic Status Values B-2
B.3 Job Status Values B-2
B.4 Link Status Values B-3
B.5 Switch Network Status Values B-3
B.6 Environment Status Values B-4
B.7 Node Status Values B-5
B.8 Partition Status Values B-6
B.9 Resource Status Values B-6
B.10 Transaction Status Values B-7

C RMS Kernel Module **C-1**
C.1 Introduction C-1
C.2 Capabilities C-1
C.3 System Call Interface C-2
rms_setcorepath(3) C-3
rms_getcorepath(3) C-3
rms_prgcreate(3) C-4
rms_prgdestroy(3) C-4
rms_prgids(3) C-6
rms_prginfo(3) C-6
rms_getprgid(3) C-6
rms_prgsuspend(3) C-8
rms_prgresume(3) C-8
rms_prgsignal(3) C-8

rms_prgaddcap(3) C-10
rms_setcap(3) C-10
rms_ncaps(3) C-12
rms_getcap(3) C-12
rms_prggetstats(3) C-13
rms_setpset(3) C-15
rms_getpset(3) C-15
rms_elaninitdone(3) C-16

D RMS Application Interface **D-1**

D.1 Introduction D-1
D.1.1 Compilation D-2
D.1.2 Error handling D-3
D.2 Manual pages D-3
rms_allocateCpus(3) D-4
rms_allocateHosts(3) D-7
rms_allocateResource(3) D-9
rms_closeDatabase(3) D-12
rms_createResource(3) D-13
rms_deallocateResource(3) D-15
rms_defaultPartition(3) D-17
rms_numCpus(3) D-17
rms_numNodes(3) D-17
rms_machineName(3) D-17
rms_partitionIndex(3) D-17
rms_whichPartition(3) D-17
rms_errorString(3) D-19
rms_freeCpus(3) D-20
rms_freeCpusPerNode(3) D-20
rms_getNodeStats(3) D-21
rms_jobIds(3) D-23

rms_killJob(3) D-24
rms_nodeInfo(3) D-26
rms_partitionInfo(3) D-29
rms_resourceId(3) D-31
rms_resourceStats(3) D-32
rms_run(3) D-34
rms_suspendResource(3) D-37
rms_resumeResource(3) D-37
rms_killResource(3) D-37
rms_setResourcePriority(3) D-37
rms_version(3) D-39
rms_checkVersion(3) D-39

E Accounting Summary Script **E-1**

E.1 Introduction E-1
E.2 Command Line Interface E-1
E.3 Example Output E-2
E.4 Listing of the Script E-3

Glossary **Glossary-1**

Index **Index-1**

List of Figures

2.1 A Network of Nodes 2-2
2.2 High Availability RMS Configuration 2-3
2.3 The Database 2-6
2.4 Partitioning a System 2-7
2.5 Distribution of Processes 2-9
2.6 Preemption of Low Priority Jobs 2-10
2.7 Two Configurations 2-11

3.1 Distribution of Parallel Processes 3-2
3.2 Loading and Running a Parallel Program 3-3

A.1 A 2-Stage, 16-Node, Switch Network A-2
A.2 A 3-Stage, 64-Node, Switch Network A-2
A.3 A 3-Stage, 128-Node, Switch Network A-3

List of Tables

11.1 Access Controls Table 11-9
11.2 Accounting Statistics Table 11-9
11.3 Archive Tables Table 11-11
11.4 Default Entries in the Archive Tables Table 11-11
11.5 Attributes Table 11-12
11.6 Machine Description Attributes 11-12
11.7 System Specific Attributes 11-13
11.8 Performance Statistics Attributes 11-16
11.9 Server Attributes 11-16
11.10 Scheduling Attributes 11-17
11.11 Backups Table 11-20
11.12 Elans Table 11-20
11.13 Elites Table 11-21
11.14 Event Handlers Table 11-22
11.15 Default Entries in the Event Handlers Table 11-22
11.16 Events Table 11-22
11.17 Example of Status Changes 11-23
11.18 Fields Table 11-23
11.19 Type Values 11-24
11.20 Installed Components Table 11-24
11.21 Jobs Table 11-25

11.22 Link Errors Table 11-26
11.23 Module Types Table 11-27
11.24 Valid Module Types 11-27
11.25 Nodes Table 11-28
11.26 Node Statistics Table 11-31
11.27 Partitions Table 11-33
11.28 Projects Table 11-35
11.29 Resources Table 11-35
11.30 Scheduler Log Table 11-36
11.31 Servers Table 11-37
11.32 Default Entries in the Servers Table 11-38
11.33 Services Table 11-39
11.34 Default Entries in the Services Table 11-39
11.35 Software Products Table 11-40
11.36 Component Attribute Values 11-40
11.37 Status Table 11-40
11.38 Default Entries in the Status Table 11-41
11.39 Switch Boards Table 11-41
11.40 Switch Modules Table 11-42
11.41 Terminal Servers Table 11-43
11.42 Terminal Server Types Table 11-43
11.43 Default Entries in the Terminal Server Types Table 11-43
11.44 Transaction Log Table 11-44
11.45 Entry in the Transactions Table 11-44
11.46 Tree Status Table 11-45
11.47 Unidentified MACs Table 11-45
11.48 Users Table 11-46

A.1 Switch Network Parameters A-3

B.1 Job Status Values B-2
B.2 Link Status Values B-3

B.3 Switch Board and Module Status Values B-3
B.4 Environment Status Values B-4
B.5 Node Status Values B-5
B.6 Run Level Status Values B-5
B.7 Partition Status Values B-6
B.8 Resource Status Values B-6
B.9 Transaction Status Values B-7

1

Introduction

1.1 Scope of Manual

This manual describes the Resource Management System (RMS). The manual's purpose is to provide a technical overview of the RMS system, its functionality and programmable interfaces. It covers the RMS daemons, client applications, the RMS database, the system call interface to the RMS kernel module and the application program interface to the RMS database.

1.2 Audience

This manual is intended for system administrators and developers. It provides a detailed technical description of the operation and features of RMS and describes the programming interface between RMS and third-party systems.

The manual assumes that the reader is familiar with the following:

- UNIX® operating system including shell scripts
- C programming language

1.3 Using this Manual

This manual contains ten chapters and five appendices. The contents of these are as follows:

Chapter 1 (*Introduction*)

Explains the layout of the manual and the conventions used to present information.

Chapter 2 (*Overview of RMS*)

Overviews the functions of the RMS and introduces its components.

Chapter 3 (*Parallel Programs Under RMS*)

Shows how parallel programs are executed under RMS.

Chapter 4 (*RMS Daemons*)

Describes the functionality of the RMS daemons.

Chapter 5 (*RMS Commands*)

Describes the RMS commands.

Chapter 6 (*Access Control, Usage Limits and Accounting*)

Explains RMS access controls, usage limits and accounting.

Chapter 7 (*RMS Scheduling*)

Describes how RMS schedules parallel jobs.

Chapter 8 (*Event Handling*)

Describes RMS event handling.

Chapter 9 (*Setting up RMS*)

Explains how to set up RMS.

Chapter 10 (*Day-to-Day RMS Operation*)

Describes day-to-day operations and establishing archive and backup procedures.

Chapter 11 (*The RMS Database*)

Presents the structure of tables in the RMS database.

Appendix A (*HP AlphaServer SC Interconnect Terms*)

Defines terms relating to support for QsNet in RMS.

Appendix B (*RMS Status Values*)

Lists the status values of RMS objects.

Appendix C (*RMS Kernel Module*)

Describes the RMS kernel module and its system call interface.

Appendix D (*RMS Application Interface*)

Describes the RMS application interface.

Appendix E (*Accounting Summary Script*)

Contains an example of producing accounting information.

1.4 Related Information

The following manuals provide additional information about the RMS from the point of view of either the system administrator or the user:

- *HP AlphaServer SC User Guide*
- *HP AlphaServer SC System Administration Guide*

1.5 Location of Online Documentation

New versions of this and other Quadrics documentation can be found on the Quadrics web site `http://www.quadrics.com`.

1.6 Reader's Comments

HP welcomes any comments and suggestions that you have on this document. End users should send all comments and suggestions to their HP support engineer.

1.7 Conventions

The following typographical conventions have been used in this document:

`monospace type`

Monospace type denotes literal text. This is used for command descriptions, file names and examples of output.

`bold monospace type`

Bold monospace type indicates text that the user enters in contrast to on-screen computer output.

`italic monospace type`

Italic (slanted) monospace type denotes some meta text. This is used most often in command or parameter descriptions to show where a textual value is to be substituted.

Conventions

italic type	Italic (slanted) proportional type is used in the text to introduce new terms. It is also used when referring to labels on graphical elements such as buttons.
`Ctrl/x`	To engage this symbol depress the `Ctrl` key while depressing another key or mouse button (shown here by `x`).
TLA	Small capital letters indicate an abbreviation (see Glossary).
`ls(1)`	A cross-reference to a reference page includes the appropriate section number in parentheses.
`#`	A number sign represents the superuser prompt.
`%, $`	A percent sign represents the C shell system prompt. A dollar sign represents the system prompt for the Bourne, Korn and POSIX shells.

2

Overview of RMS

2.1 Introduction

This chapter describes the role of the Resource Management System (RMS). The RMS provides tools for the management and use of an HP AlphaServer SC system. To put into context the functions that RMS performs, a brief overview of the system architecture is given first in Section 2.2. Section 2.3 outlines the main functions of the RMS and introduces the major components of the RMS: a set of UNIX daemons, a suite of command line utilities and an SQL database. Finally, Section 2.4 describes the resource management facilities from the system administrator's point of view.

2.2 The System Architecture

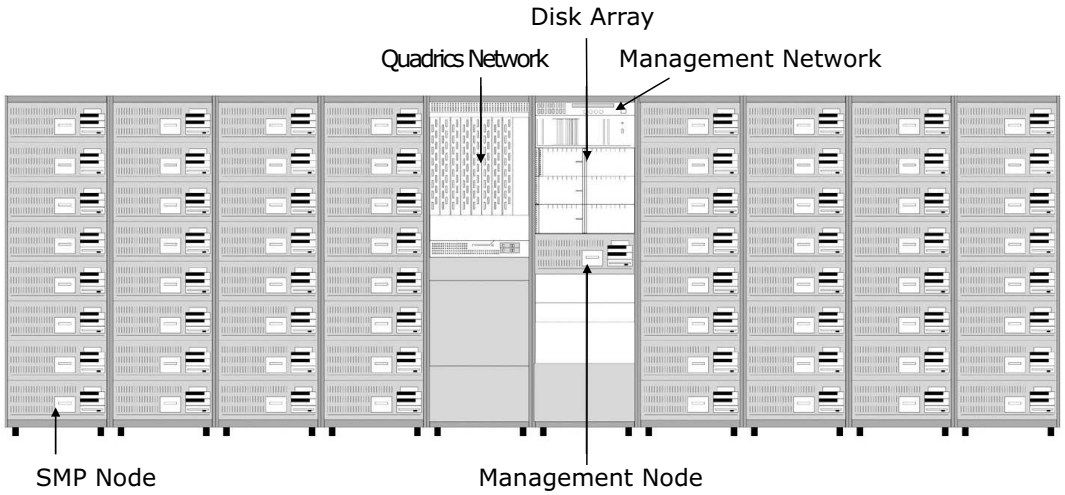
An RMS system looks like a standard UNIX system: it has the familiar command shells, editors, compilers, linkers and libraries; it runs the same applications. The RMS system differs from the conventional UNIX one in that it can run parallel applications as well as sequential ones. The processes that execute on the system, particularly the parallel programs, are controlled by the RMS.

2.2.1 Nodes

An RMS system comprises a network of computers (referred to as *nodes*) as shown in Figure 2.1. Each node may have single or multiple processors (such as a SMP server); each node runs a single copy of UNIX. Nodes used interactively to login to the RMS system are also connected to an external LAN. The application nodes, used for running

parallel programs, are accessed through the RMS.

Figure 2.1: A Network of Nodes



All of the nodes are connected to a *management network* (normally, a 100 BaseT Ethernet). They are also connected to the *HP AlphaServer SC Interconnect*, providing high-performance user-space communications between application processes.

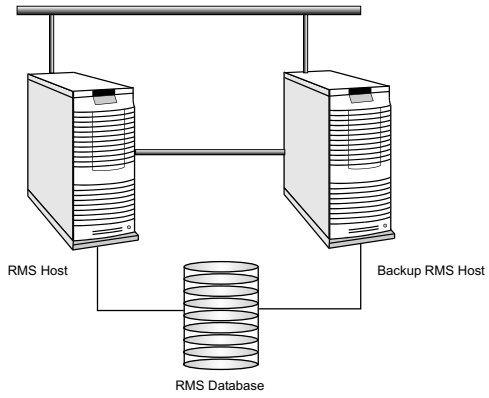
By default, RMS processes communicate using IP over the management network. RMS processes can also communicate using IP over the HP AlphaServer SC Interconnect (see Section 9.4 for more details).

The RMS processes that manage the system reside either on an interactive node or on a separate management server. This node, known as `rmshost`, holds the RMS database, which stores all state for the RMS system. [1]

For high-availability installations, the `rmshost` node should be an interactive node rather than a management server. This will allow you to configure the system for failover, as shown in Figure 2.2 (see Chapter 5 of the *HP AlphaServer SC System Administration Guide* for details).

[1]This database is refered to as the SC database in recent versions of the HP AlphaServer SC documentation.

Figure 2.2: High Availability RMS Configuration



The RMS processes run on the node with the name `rmshost`, which migrates to the backup on fail-over. The database is held on a shared disk, accessible to both the primary and backup node.

2.3 The Role of the RMS

The RMS provides a single point interface to the system for resource management. This interface enables a system administrator to manage the system resources (CPUs, memory, disks, and so on) effectively and easily. The RMS includes facilities for the following administrative functions:

Monitoring controlling and monitoring the nodes in the network to ensure the correct operation of the hardware

Fault diagnosis diagnosing faults and isolating errors; instigating fault recovery and escalation procedures

Data collection recording statistics on system performance

Allocating CPUs allocating system resources to applications

Access control controlling user access to resources

Accounting single point for collecting accounting data

Parallel jobs providing the system support required to run parallel programs

Scheduling	deciding when and where to run parallel jobs
Audit	maintaining an audit trail of system state changes

From the user's point of view, RMS provides tools for:

Information	querying the resources of the system
Execution	loading and running parallel programs on a given set of resources
Monitoring	monitoring the execution of parallel programs

2.3.1 The Structure of the RMS

RMS is implemented as a set of UNIX commands and daemons, programmed in C and C++, using sockets for communications. All of the details of the system (its configuration, its current state, usage statistics) are maintained in a SQL database, as shown in Figure 2.3. See Section 2.3.4 for an overview and Chapter 11 (*The RMS Database*) for details of the database.

2.3.2 The RMS Daemons

A set of daemons provide the services required for managing the resources of the system. To do this, the daemons both query and update the database (see Section 2.3.4).

- The *Database Manager*, `msql3d`, provides SQL database services.
- The *Machine Manager*, `mmanager`, monitors the status of nodes in an RMS system.
- The *Partition Manager*, `pmanager`, controls the allocation of resources to users and the scheduling of parallel programs.
- The *Switch Network Manager*, `swmgr`, supervises the operation of the HP AlphaServer SC Interconnect, monitoring it for errors and collecting performance data.
- The *Event Manager*, `eventmgr`, runs handlers in response to system incidents and notifies clients who have registered an interest in them.
- The *Transaction Log Manager*, `tlogmgr`, instigates database transactions that have been requested in the *Transaction Log*. All client transactions are made through this mechanism. This ensures that changes to the database are serialized and an audit trail is kept.
- The *Process Manager*, `rmsmhd`, runs on each node in the system. It starts the other RMS daemons.

- The *RMS Daemon*, `rmsd`, runs on each node in the system. It loads and runs user processes and monitors resource usage and system performance.

The RMS daemons are described in more detail in Chapter 4 (*RMS Daemons*).

2.3.3 The RMS Commands

RMS commands call on the RMS daemons to get information about the system, to distribute work across the system, to monitor the state of programs and, in the case of administrators, to configure the system and back it up. A suite of these RMS client applications is supplied. There are commands for users and commands for system administrators.

The user commands for gaining access to the system and running parallel programs are as follows:

- `allocate` reserves resources for a user.
- `prun` loads and runs parallel programs.
- `rinfo` gets information about the resources in the system.
- `rmsexec` performs load balancing for the efficient execution of sequential programs.
- `rmsquery` queries the database. Administrators can also use `rmsquery` to update the database.

The system administration commands for managing the system are as follows:

- `msqladmin` performs database server administration.
- `nodestatus` gets and sets node status information.
- `rcontrol` starts, stops and reconfigures services.
- `rmsarchive` archives redundant records and deletes them from the database.
- `rmsbackup` backs up the database.
- `rmsbuild` populates the RMS database with information on a given system.
- `rmsctl` starts and stops RMS and shows the system status.
- `rmshost` reports the name of the node hosting the RMS database.
- `rmstbladm` builds and maintains the database.

The services available to the different types of user (application programmer, operator, system administrator) are subject to access control. Access control restrictions are embedded in the SQL database, based on standard UNIX group IDs (see Section 11.5.22). Users have read access to all tables but no write access. Operator and administrative applications are granted limited write access. Password-protected administrative applications and RMS itself have full read/write access.

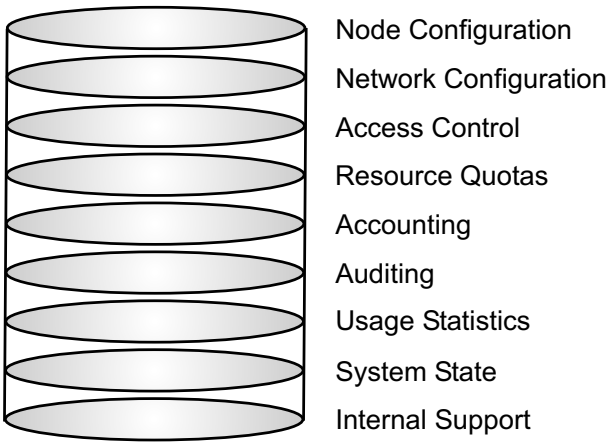
The RMS commands are described in more detail in Chapter 5 (*RMS Commands*).

2.3.4 The RMS Database

The database provides a platform-independent interface to the RMS system. Users and administrators can interact with the database using standard SQL queries. For example, the following query displays details about the nodes in the machine. It selects fields from the table called `nodes` (see Section 11.5.15). The query is submitted through the RMS client `rmsquery`.

```
$ rmsquery "select name,status from nodes"
atlasms  running
atlas0   running
atlas1   running
atlas2   running
atlas3   running
```

Figure 2.3: The Database



RMS uses the mSQL database engine from Hughes Technologies (for details see `http://www.Hughes.com.au`). Client applications may use C, C++, Java, HTML or UNIX script interfaces to generate SQL queries. See the Quadrics support page

http://www.quadrics.com for details of the SQL language subset supported by mSQL.

2.4 RMS Management Functions

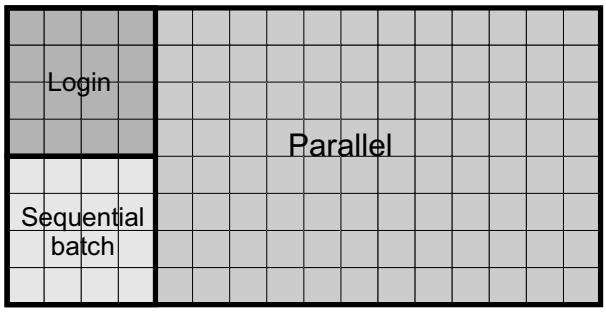
The RMS gives the system administrator control over how the resources of a system are assigned to the tasks it must perform. This includes the allocation of resources (Section 2.4.1), scheduling policies (Section 2.4.2), access controls and accounting (Section 2.4.3) and system configuration (Section 2.4.4).

2.4.1 Allocating Resources

The nodes in an RMS system can be configured into mutually exclusive sets known as *partitions* as shown in Figure 2.4. The administrator can create partitions with different mixes of resources to support a range of uses. For example, a system may have to cater for a variety of processing loads, including the following:

- Interactive login sessions for conventional UNIX processes
- Parallel program development
- Production execution of parallel programs
- Distributed system services, such as database or file system servers, used by conventional UNIX processes
- Sequential batch streams

Figure 2.4: Partitioning a System



The system administrator can allocate a partition with appropriate resources for each of these tasks. Furthermore, the administrator can control who accesses the partitions (by user or by project) and how much of the resource they can consume. This ensures that

resources intended for a particular purpose, for example, running production parallel codes, are not diverted to other uses, for example, running user shells.

A further partition, the `root` partition, is always present. It includes all nodes. It does not have a scheduler. The `root` partition can only be used by administrative users (`root` and `rms` by default).

2.4.2 Scheduling

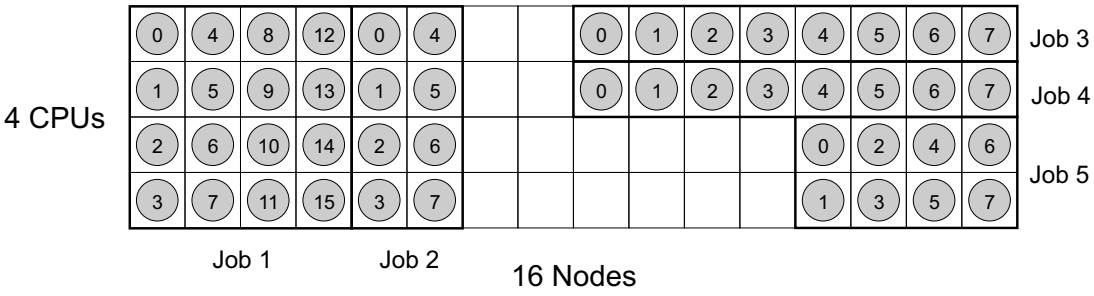
Partitions enable different scheduling policies to be put into action. On each partition, one or more of four scheduling policies can be deployed to suit the intended usage:

1. Gang scheduling of parallel programs, where all processes in a program are scheduled and de-scheduled together. This is the default scheduling policy for parallel partitions.
2. Regular UNIX scheduling with the addition of load balancing, whereby the user can run a sequential program on a lightly loaded node. The load may be judged in terms of free CPU time, free memory or number of users.
3. Batch scheduling, where the use of resources is controlled by a batch system.
4. External scheduling, where an external scheduler uses the services provided by the Partition Manager to implement a site-specific scheduling policy.

Scheduling parameters such as time limits, time slice interval and minimum request size are applied on an individual partition basis. Default priorities, memory limits and CPU usage limits can be applied to users or projects to tune the partition's workload. For details see Chapter 6 (*Access Control, Usage Limits and Accounting*) and Chapter 7 (*RMS Scheduling*).

The partition shown in Figure 2.5 has its CPUs allocated to five parallel jobs. The jobs have been allocated CPUs in two different ways: jobs 1 and 2 use all of the CPUs on each node; jobs 3, 4 and 5 are running with only one or two CPUs per node. RMS allows the user to specify how their job will be laid out, trading off the competing benefits of increased locality on the one hand against increased total memory size on the other. With this allocation of resources, all five parallel programs can run concurrently on the partition.

Figure 2.5: Distribution of Processes

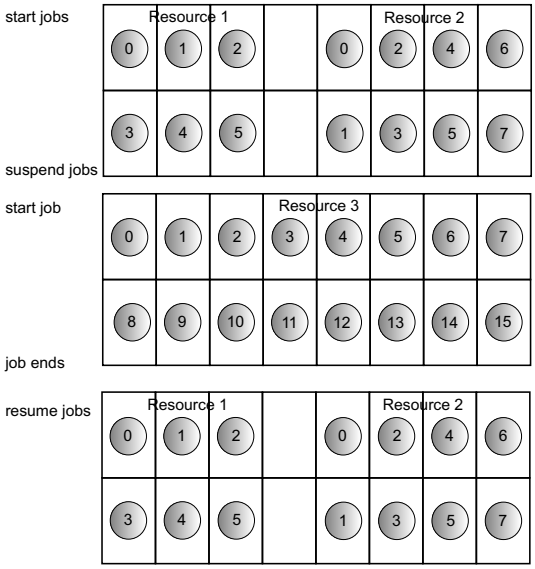


The RMS scheduler allocates contiguous ranges of nodes with a given number of CPUs per node [2]. Where possible, each *resource request* is met by allocating a single range of nodes. If this is not possible, an unconstrained request (one that only specifies the number of CPUs required) may be satisfied by allocating CPUs on disjoint nodes. This ensures that an unconstrained resource request can utilize all of the available CPUs.

The scheduler attempts to find free CPUs for each request. If this is not possible, the request blocks until CPUs are available. RMS preempts programs when a higher priority job is submitted, as shown in Figure 2.6. Initially, CPUs have been allocated for resource requests 1 and 2. When the higher priority resource request 3 is submitted, 1 and 2 are suspended; 3 runs to completion after which 1 and 2 are restarted.

[2]The scheduler allocates contiguous ranges of nodes so that processes may take advantage of the HP AlphaServer SC Interconnect hardware support for broadcast and barrier operations applied to a contiguous range of network addresses.

Figure 2.6: Preemption of Low Priority Jobs



2.4.3 Access Control and Accounting

Users are allocated resources on a per-partition basis. Resources in this context include both CPUs and memory. The system administrator can control access to resources both at the individual user level and at the project level (where a project is a list of users). This means that default access controls can be set up at the project level and overridden on an individual user basis as required. The access control mechanism is described in detail in Chapter 6 (*Access Control, Usage Limits and Accounting*).

Each partition, except the `root` partition, is managed by a Partition Manager (see Section 4.4), which mediates user requests, checking access permissions and usage limits before scheduling CPUs and starting user jobs.

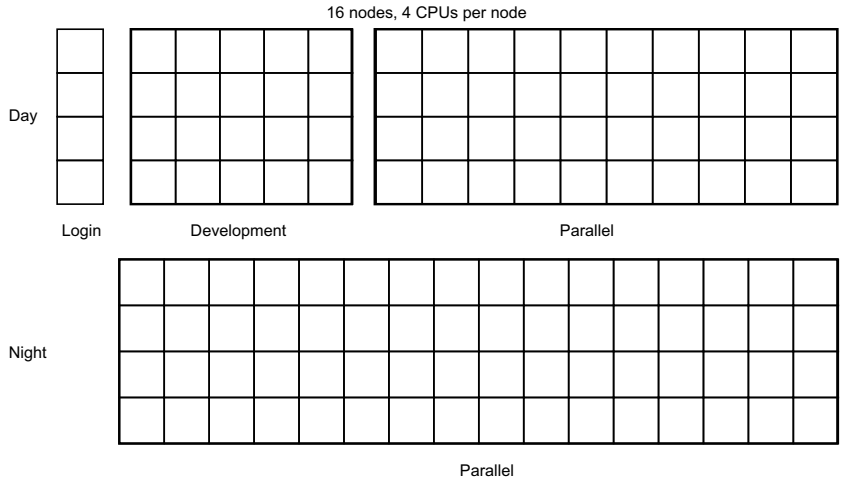
An accounting record is created as CPUs are allocated to each request. It is updated periodically until the resources are freed. The accounting record itemizes CPU and memory usage, indexed by job, by user and by project.

2.4.4 RMS Configuration

The set of partitions active at any time is known as a *configuration*. A system will normally have a number of configurations, each appropriate to a particular operating pattern. For example, there may be one configuration for normal working hours and another for night time and weekend operation.

The CPUs allocated to a partition may vary between configurations. For example, a login partition (nodes allocated for interactive use) may have more nodes allocated during working hours than at night – it may even be absent from the night time configuration. A pair of configurations are shown in Figure 2.7.

Figure 2.7: Two Configurations



RMS supports automated reconfiguration at shift changes as well as dynamic reconfiguration in response to a request from an operator or administrator. The RMS client `rcontrol` (Page 5-27) manages the switch-over from one configuration to another. For automatic reconfiguration, `rcontrol` can be invoked from a `cron(8)` job.

3

Parallel Programs Under RMS

3.1 Introduction

RMS provides users with tools for running parallel programs and monitoring their execution, as described in Chapter 5 (*RMS Commands*). Users can determine what resources are available to them and request an allocation of resources with the CPUs and memory required to run their programs. This chapter describes the structure of parallel programs under RMS and how they are run.

A parallel program consists of a controlling process, `prun`, and a number of application processes distributed over one or more nodes. Each process may have multiple threads running on one or more CPUs. `prun` can run on any node in the system but it normally runs in a login partition or on an interactive node.

In a system with SMP nodes, RMS can allocate CPUs so as to use all of the CPUs on the minimum number of nodes (a *block* distribution); alternatively, it can allocate a specified number of CPUs on each node (a *cyclic* distribution). This flexibility allows users to choose between the competing benefits of increased CPU count and memory size on each node (generally good for multithreaded applications) and increased numbers of nodes (generally best for applications requiring increased total memory size, memory bandwidth and I/O bandwidth).

Parallel programs can be written so that they will run with varying numbers of CPUs and varying numbers of CPUs per node. They can, for example, query the number of processors allocated and determine their data distributions and communications patterns accordingly (see Appendix C (*RMS Kernel Module*) for details).

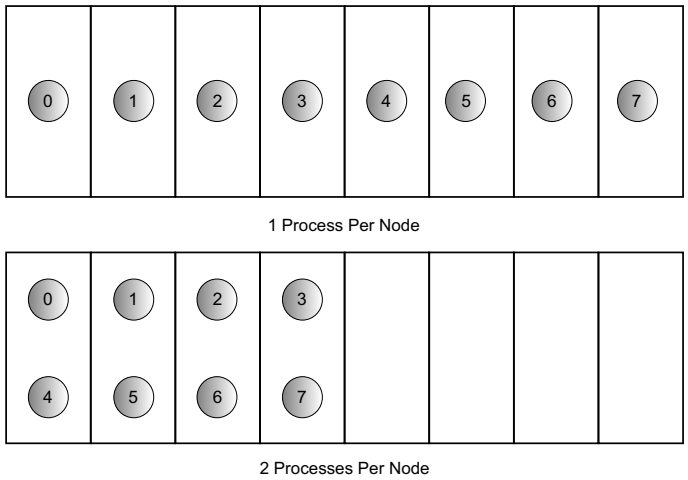
3.2 Resource Requests

Having logged into the system, a user makes a request for the resources needed to run a parallel program by using the RMS commands `prun` (see Page 5-12) or `allocate` (see Page 5-3). When using the `prun` command, the request can specify details such as the following:

- The partition on which to run the program (the `-p` option)
- The number of processes to run (the `-n` option)
- The number of nodes required (the `-N` option)
- The number of CPUs required per process (the `-c` option)
- The memory required per process (the `RMS_MEMLIMIT` environment variable)
- The distribution of processes over the nodes (the `-m`, `-B` and `-R` options)
- How standard input, output and error streams should be handled (the `-i`, `-o` and `-e` options)
- The project to which the program belongs for accounting and scheduling purposes (the `-P` option)

Two variants of a program with eight processes are shown in Figure 3.1: first, with one process per node; and then, with two processes per node.

Figure 3.1: Distribution of Parallel Processes



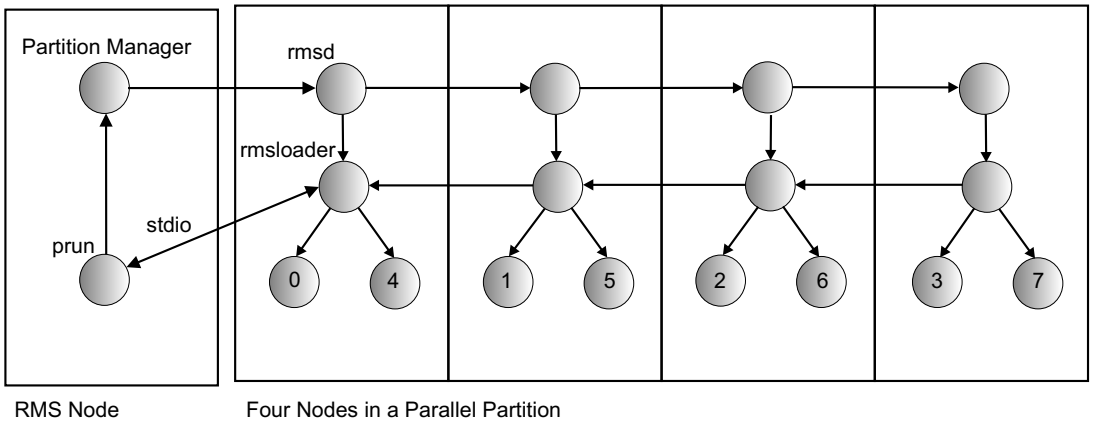
The resource request is sent to the Partition Manager, `pmanager` (described in Section 4.4). The Partition Manager performs access checks (described in Chapter 6 (*Access Control, Usage Limits and Accounting*)) and then allocates CPUs according to the policies established for the partition (see Chapter 7 (*RMS Scheduling*)).

RMS makes a distinction between allocating resources and starting jobs on them. Before the Partition Manager schedules a parallel program, it will ensure that the required CPUs and memory are allocated. Note that this may cause requests to block for longer than you might expect – especially when the job has not specified how much memory it requires. Once CPUs have been allocated, jobs can be started on them immediately.

3.3 Loading and Running Programs

A simple parallel program is shown in Figure 3.2. It has eight application processes, distributed over four nodes, two processes per node.

Figure 3.2: Loading and Running a Parallel Program



Once the CPUs have been allocated, `prun` asks the `pmanager` to start the application processes on the allocated CPUs. The `pmanager` does this by instructing the daemons running on each of the allocated nodes to start the loader process `rmsloader` on the user's behalf.

The `rmsloader` process starts the application processes executing, forwarding their `stdout` and `stderr` streams to `prun` (unless otherwise directed). Meanwhile, `prun` supplies information on the application processes as requested by `rmsloader` and forwards `stdout` and `stderr` to the controlling terminal or output files.

`prun` forwards `stdin` and certain signals (`QUIT`, `USR1`, `USR2`, `WINCH`) to the application processes. If `prun` is killed, RMS cleans up the parallel program, killing the application

processes, removing any core files if requested (see Page 5-12) and then deallocating the CPUs.

The application processes are run from the user's current working directory with the current limits and group rights. The data and stack size limits may be reduced if RMS has applied a memory limit to the program (see Page 5-16 for details).

During execution, the processes may be suspended at any time by the scheduler to allow a program with higher priority to run. All of the processes in a parallel program are suspended together under the gang-scheduling policy used by RMS for parallel programs (see Chapter 7 (*RMS Scheduling*) for details). They are restarted together when the higher priority program has completed.

A parallel program exits when all of its processes have exited. When this happens, the `rmsloader` processes reduce the exit status back to the controlling process by performing a global `OR` of the exit status of each of the processes. If `prun` is run with verbose reporting enabled, a non-zero exit status is accompanied by a message, as shown in the following example:

```
$ prun -v myprog
...
myprog: process 0 exited with status 1
```

If the level of reporting is increased with the `-vv` option, `prun` provides a commentary on the resource request. With the `-vvv` option, `rmsloader` also outputs information identifying the activity on each node running the program, as shown in the following example.

```
$ prun -vvv myprog
prun: running /home/duncan/myprog
prun: requesting 2 CPUs
prun: starting 2 processes on 2 cpus default memlimit no timelimit
prun: stdio server running
prun: loader 1 starting on atlas1 (10.128.0.7)
prun: loader 0 starting on atlas0 (10.128.0.8)
loader[atlas1]: program description complete
loader[atlas1]: nodes 2 contexts 1 capability type 0xffff8002 entries 2
loader[atlas1]: run process 1 node=5 cntx=244
prun: process 1 is pid 1265674 on atlas1
loader[atlas0]: program description complete
loader[atlas0]: nodes 2 contexts 1 capability type 0xffff8002 entries 2
loader[atlas0]: run process 0 node=4 cntx=244
prun: process 0 is pid 525636 on atlas0
...
```

When the program has exited, the CPUs are deallocated and the scheduler is called to service the queue of waiting jobs.

Sometimes, it is desirable for a user to be granted more control over the use of a resource. For instance, the user may want to run several jobs concurrently or use the same nodes for a sequence of jobs. This functionality is supported by the command `allocate` (see Page 5-3) which allows a user to allocate CPUs in a parallel partition to a UNIX shell. These CPUs are used for subsequent parallel jobs started from this shell. The CPUs remain allocated until the shell exits or a time limit expires (see Section 7.3 and Section 7.4.6).

4

RMS Daemons

4.1 Introduction

This chapter describes the role of the RMS daemons. There are daemons that run on the `rmshost` node providing services for the system as a whole:

`msql3d`	Manages the database (see Section 4.2).
`mmanager`	Monitors the state of the node (see Section 4.3).
`pmanager`	Controls the execution of parallel jobs (see Section 4.4).
`swmgr`	Monitors the state of the HP AlphaServer SC Interconnect, in cooperation with `swmserver` processes running on the embedded controllers in each of the switch modules (see Section 4.5).
`tlogmgr`	Executes change of state commands on behalf of RMS clients, typically the SC Viewer GUI (see Section 4.6).
`eventmgr`	Provides a system-wide event-handling service (see Section 4.7).

There are daemons that run on each node, providing support for RMS functionality on that node:

`rmsmhd`	The RMS Process Manager, starts and stops the other RMS daemons (see Section 4.8).
`rmsd`	Carries out instructions from `pmanager` to run users' programs (see Section 4.9).

4.1.1 Startup

RMS is started as each node executes the script `/sbin/init.d/rms` with the `start` argument on startup. This starts the `rmsmhd` daemon which, in turn, starts the other daemons on that node.

RMS can be started and stopped on all nodes using the command `rmsctl` (see Page 5-57 for details). Individual daemons can also be started, stopped and reloaded individually by `rcontrol` once RMS is running. See Page 5-27 for details.

4.1.2 Log Files

Output from the management daemons is logged to the directory `/var/rms/adm/log`. The log files are called *daemon*`.log`, where *daemon* gives the name of the RMS daemon, such as `swmgr`. The Partition Managers are distinguished by suffixing `pmanager` with a hyphen and then the name of the partition. For example, the Partition Manager for the partition `par1` is known is `pmanager-par1`. The (per-node) log files for `rmsmhd` and `rmsd` include the name of node the daemon is running on `rmsd.atlas0.log`, for example. The RMS daemons log system call failures and other errors to the file `/var/rms/adm/log/error.log`.

4.1.3 Status

The `servers` table contains information on the status of each daemon: the time it was started, its process ID and the name of its host node (see Section 11.5.21 for details of the table structure).

Note that the `status` field in the `servers` table is set to `error` if an error occurs when starting an RMS daemon. The corresponding entry in the `events` table describes what went wrong (see Chapter 8 (*Event Handling*) for details).

The command `rinfo` can be used to get reports on the status of each daemon. See Page 5-41 for details.

4.2 The Database Server

The Database Server, `msql3d`, manages the RMS database, providing an SQL interface for its clients. Client applications may use C, C++, Java or UNIX scripts to generate SQL queries for `msql3d`.

The database holds all state information for RMS. This information is initially created by the RMS client application `rmsbuild` (see Page 5-54). The information is updated by the other RMS daemons as RMS operates. The information can be archived and backed up using `rmsarchive` and `rmsbackup`. The database can be generally maintained

using the database administration program, `rmstbladm` (see Page 5-66).

4.3 The Machine Manager

The Machine Manager, `mmanager`, is responsible for detecting and reporting changes in the state of each node in the system. It records the current state of each node and any changes in state in the database.

When a node is functioning correctly, `rmsd`, a daemon which runs on each node, periodically sends performance statistics and status information to the the `mmanager` which updates the database. If the node crashes, RMS is stopped, or IP traffic to and from the node stops, then these updates stop. The `mmanager` monitors the update times for each node. If the no update occurs within the *inactivity timeout* the mmanager generates an event and updates the status of the node.

The Machine Manager records the current status of nodes in the `nodes` table (see Section 11.5.15) and the `status` table (see Section 11.5.24). Changes in node status are entered in the `events` table (see Section 11.5.9). Each node is normally in one of three states, `running` (UNIX is running and RMS is generating regular status updates), `active` (UNIX is running but RMS is not generating updates) and `not responding` (UNIX is not running). In a AlphaServer SC system the `not responding` state indicates that the node has dropped out of its TruCluster domain.

A fourth state `configured out` indicates that nodes are no longer being monitored. RMS ignores such nodes until they are configured back in again by `rcontrol` (see Page 5-33 for details).

The maximum time allowed between statistics and status updates is set in the `attributes` table (see Section 11.5.4) with the `inactivity-timeout` attribute.

In addition to monitoring node status the Machine Manager monitors the state of each TruCluster domain. If no updates are received within the inactivity timeout from voting members of a given domain the Machine Manager will post an event indicating that the whole domain is not responding.

4.4 The Partition Manager

The nodes in the RMS machine are configured into mutually exclusive sets known as *partitions* (see Section 2.4). By restricting access to partitions, the system administrator can reserve particular partitions for specific types of task or specific users. In this way, the system administrator can ensure that resources are used most effectively; for example, that resources intended for running parallel programs are not consumed running user shells. The access restrictions are set up in the `access_controls` table (see Section 11.5.1) of the RMS database.

Each partition is controlled by a Partition Manager, `pmanager`. The Partition Manager mediates each user's requests for resources (CPUs and memory) to run jobs in the partition. It checks the user's access permissions and resource limits before adding the request to its scheduling queue. The request blocks until the resources are allocated for the job.

When the resources requested by the user become available, the Partition Manager instructs `rmsd`, a daemon that runs on each node in the partition (see Section 4.9), to create an Elan capability for the user's jobs. Finally, the Partition Manager replies to the user's request and the user's job can start.

The Partition Manager makes new scheduling decisions periodically and in response to changes in resource requests (see Chapter 7 (*RMS Scheduling*) for details). These decisions may result in jobs being suspended or resumed. Such scheduling operations, together with those performed as jobs are killed, are performed by the Partition Manager sending scheduling or signal delivery requests to the `rmsd`s.

The Partition Manager is connected to its `rmsd`s by a tree of sockets. Commands are routed down this tree; they complete when an acknowledgement is returned. For example, jobs are only marked as finished when the Partition Manager has confirmed that all of their processes have exited.

If the tree of sockets is broken by a node crash (or by RMS being stopped on a node), the Partition Manager marks the node's partition as `blocked` and generates an event. The node can then be rebooted or configured out of the machine. Job scheduling and signal delivery stops while the partition is blocked. Jobs that don't use the failied node will continue to run (and may complete) while the partition is in this state. If the node is rebooted quickly, the `rmsd`s reconnect and the Partition Manager continues as before. If the node does not reboot quickly the Partition Manager will configure the node out. At this point the partition will return to the `running` state and job scheduling will resume. Jobs that spanned the failing node are cleaned up at this point. The other jobs run on unless explicitly killed.

The Partition Manager is started by the `rmsmhd` daemon, running on the `rmshost` node, on instruction from `rcontrol` (see Page 5-27). Once the partition is running, a startup script `/usr/opt/rms/etc/pstartup` is executed. This script performs site-specific and OS-specific actions depending upon the partition type.

Configuration information about each partition is held in the `partitions` table (see Section 11.5.17). The information is indexed by the name of the partition together with the name of the active configuration. The Partition Manager makes updates to the `partitions` table (see Section 11.5.17) when it starts and as CPUs are allocated and freed.

The Partition Manager creates an entry in the `resources` table (see Section 11.5.19) each time a user makes a request for resources to run a job. This entry is updated each

time CPUs are allocated or deallocated. The Partition Manager adds an entry to the `jobs` table (see Section 11.5.12) as each job starts, updating it if the job is suspended or resumed and when the job completes.

Scheduling decisions are recorded in the `scheduler_log` table (see Table 11.30) if resource request debugging is enabled by `rcontrol` (see Page 5-40)

The Partition Manager creates an entry in the accounting statistics (`acctstats`) table (see Section 11.5.2) when CPUs are allocated. The entry is updated periodically until the request completes.

The Partition Manager consults the `users` table (see Section 11.5.32), the `projects` table (see Section 11.5.18) and the `access_controls` table (see Section 11.5.1) to verify users' access permissions and usage limits.

The Partition Manager performs periodic housekeeping functions at a rate controlled by the `pmanager-`*`name`*`-poll-interval` attribute (see Section 9.5.8) where *`name`* is the name of the partition. This interval also controls the time allowed for commands issued by the partition manager to complete.

4.5 The Switch Manager

The Switch Network Manager, `swmgr`, controls and monitors the HP AlphaServer SC Interconnect (see Appendix A (*HP AlphaServer SC Interconnect Terms*)). It does this *via* the `swmserver` processes associated with each switch module. `swmgr` collects fan, power supply and temperature data from `swmserver` processes running on the embedded controllers in each of the HP AlphaServer SC Interconnect switch modules. It updates status information in the database and generates events if components fail or the switch modules need maintenance. Events are generated under the following circumstances:

- Switch module exceeds temperature threshold.
- Power supply failure in switch module.
- Fan failure in switch module.
- Clock failure in switch module.
- Switch module has not generated a recent heartbeat.
- Board removed from switch module.
- Failure of the control cable connecting the switch to the embedded controller or node.

The following additional events are generated by the `swmgr` on receipt of a link error report from an `swmserver`:

- Link error data available.
- Link in reset data available.

The rate at which these events are generated can be controlled by the the `linkerror-event-interval` attribute (see Section 11.5.4).

The Switch Network Manager creates and maintains the entries in the `elites` table (see Section 11.5.7), the `switch_boards` table (see Section 11.5.25) and the `switch_modules` table (see Section 11.5.26). In the event of errors, it creates entries in the `link_errors` table (see Section 11.5.13).

4.6 The Transaction Log Manager

The Transaction Log Manager, `tlogmgr`, executes change of state requests that have been entered in the `transactions` table (see Section 11.5.29) by RMS administrative clients, typically the SC Viewer GUI. This mechanism is employed to serialize changes to the database and to provide an audit trail of such changes.

The entry in the `transactions` table records who requested the change. It names the service required together with any arguments to pass to the process on startup. A transaction handle (a unique ID) is generated for the entry and passed to both the client and the RMS daemon that provides the service.

The RMS daemon uses the transaction handle to label any results it produces, such as an entry in the `transaction_outputs` table (see Section 11.4.5). The client uses the handle to select the result from the relevant table. Output from the service is appended to an output log. The name of this log is entered in the `transactions` table together with the status of the transaction.

The services that are available are listed in the `services` table (see Section 11.5.22). Each entry in the `services` table specifies which command to run, who can run it and on which host.

The Transaction Log Manager maintains the `transactions` table (see Section 11.5.29). It consults the `services` table (see Section 11.5.22) in order to execute transactions on behalf of its clients.

4.7 The Event Manager

When an RMS daemon detects an anomaly (such as a node crash or a high temperature reading), it writes an event description to the `events` table (see Section 11.5.9). It is the job of the Event Manager, `eventmgr`, to execute recovery scripts that either correct the fault or report it to the operators if manual intervention is required.

On receiving an event notification, the Event Manager looks for a matching entry in the `event_handlers` table (see Section 11.5.8), executing the handler script if it finds a match (see Section 8.2 for details). If no match is found, it runs the default event handler script; this script is site-specific, but it would typically run a command to escalate the event through SNMP or email.

The Event Manager also implements the event-waiting mechanism that enables client applications both to generate and to wait efficiently on a specified event. Typical events include the following:

- Nodes changing state
- Partitions starting
- Transaction log entries being executed

The details that describe the event are held in the `events` table (see Section 11.5.9). The Event Manager's job is to notify interested clients that the event has occurred. This frees the clients from having to poll for the information. For more information on RMS event handling, see Chapter 8 (*Event Handling*).

The Event Manager consults the `events` table (see Section 11.5.9) and the `event_handlers` table (see Section 8.5).

4.8 The Process Manager

The Process Manager, `rmsmhd` (also known as the RMS mother hen daemon), is responsible for starting and stopping the other RMS daemons. It runs on each node and is responsible for managing the other daemons that run on its node. It starts them as the node boots, stops them as the node halts and starts or stops them in response to requests from the RMS client application `rcontrol` (see Page 5-27).

RMS stores information regarding which daemons run on which nodes; this information is stored centrally in the RMS database, rather than in node-specific configuration files. On startup, the Process Manager checks the `servers` table (see Section 11.5.21) for entries matching its node. This information is used to start the other daemons. If its child processes (the other daemons) are killed, it checks the table to see whether they should be restarted. The Process Manager creates its own entry in the `servers` table.

4.9 The RMS Daemon

The RMS daemon `rmsd` runs on each node in the machine. Its purpose is as follows:

- To start application processes

- To implement scheduling decisions made by the Partition Manager
- To clean up after parallel programs that have finished
- To execute RMS remote procedure calls on behalf of clients elsewhere in the network
- To collect accounting data and performance statistics

`rmsd` carries out the following tasks on behalf of the Partition Manager to run a user's parallel program:

- Creating and destroying communication contexts (see Section C.2)
- Starting the application loader, `rmsloader`.
- Delivering signals
- Suspending and resuming processes
- Collecting accounting data from the kernel

The `rmsd`s communicate with each other and with the Partition Manager that controls their node over a tree of sockets. Requests (for example, to deliver a signal to all processes in a parallel program) are passed down this tree to the appropriate range of nodes. The results of each request are combined as they pass back up the tree.

In each TruCluster domain, the `rmsd` daemons on the root nodes monitor the state of all of the nodes in the domain, updating the Machine Manager when a node boots, is halted or crashes.

`rmsd` is started by the RMS daemon `rmsmhd`. If the `rmsd` exits or is killed then `rmsmhd` will restart it.

`rmsd` records configuration information about each node (number of CPUs, amount of memory and so on) in the `nodes` table (see Section 11.5.15) as it starts. `rmsd` records details of the node's HP AlphaServer SC Interconnect configuration in the `elans` table as it starts (see Section 11.5.6 and Appendix A (*HP AlphaServer SC Interconnect Terms*)).

`rmsd` periodically sends status information and performance statistics to the Machine Manager. The interval at which these statistics are sampled is set in the `attributes` table with the `cpu-stats-poll-interval` attribute.

5

RMS Commands

5.1 Introduction

This chapter describes the RMS commands. RMS includes utilities that enable system administrators to configure and manage the system, in addition to those that enable users to run their programs.

RMS includes the following commands intended for use by system administrators:

`rcontrol`	The `rcontrol` command is used to control the system resources.
`rmsarchive`	The `rmsarchive` command archives redundant records and then deletes them from the database. It is typically run as a `cron(8)` job.
`rmsbackup`	The `rmsbackup` command automates the backup of the RMS database. It is typically run as a `cron(8)` job.
`rmsbuild`	The `rmsbuild` command creates and populates an RMS database for a given machine.
`rmsctl`	The `rmsctl` script is used to stop and start the RMS system and to report its status.
`rmsquery`	The `rmsquery` command is used to select data from the database and, in the case of system administrators, to update it.
`rmstbladm`	The table administration `rmstbladm` program is used to create a database, to back it up and to restore it.

`swctrl`	The `swctrl` command is used to query the state of the switch network.

The following utilities are used internally by RMS and may also be used by system administrators:

`msqladmin`	The `msqladmin` command is used for creating and deleting databases and stopping the mSQL server.
`nodestatus`	The `nodestatus` command is used to get or set the status or run level of a node.
`rmshost`	The `rmshost` command reports the name of the node running the RMS management daemons.
`rmspost`	The `rmspost` command is used to post events from scripts.

RMS includes the following commands for all users of the system:

`allocate`	The `allocate` command is used to reserve access to a set of CPUs either for running multiple tasks in parallel or for running a sequence of commands on the same CPUs.
`prun`	The `prun` command is used to run a parallel program or to run multiple copies of a sequential program.
`rinfo`	The `rinfo` command is used to determine what resources are available and which jobs are running.
`rmsexec`	The `rmsexec` command is used to run a sequential program on a lightly loaded node.

The application loaders `rmsloader` and `pfloader` are used by `prun` to execute parallel jobs.

The following sections describe the commands in more detail, listing them in alphabetical order.

NAME

allocate – Reserves access to CPUs

SYNOPSIS

```
allocate [-hIv] [-B base] [-C CPUs] [-N nodes | all] [-n CPUs]
         [-P project] [-p partition] [-R request]
         [script [args ...]]
```

OPTIONS

`-B` *base*
: Specifies the number (node id) of the base node (the first node to use). By default, the base node is unassigned, leaving the scheduler free to select nodes that are not in use.

`-C` *CPUs*
: Specifies the number of CPUs required per node (default 1).

`-h`
: Displays the list of options.

`-I`
: Allocate CPUs immediately or fail. By default, `allocate` blocks until resources become available.

`-N` *nodes*`|all`
: Specifies the number of nodes to allocate (default 1). To allocate one CPU on each node in the partition, use the argument `all` as follows: `allocate -N all`. Either the `-C` option or the `-n` option can be combined with `-N` but not both.

`-n` *CPUs*
: Specifies the total number of CPUs required.

`-P` *project*
: Specifies the name of the project with which the job should be associated for scheduling and accounting purposes.

`-p` *partition*
: Specifies the target partition from which the resources are to be allocated.

`-R` *request*
: Specify particular resource requests flags. The types of *request* currently supported are as follows:

  `hwbcast=0|1`
  : With a value of 1, this specifies a contiguous range of nodes and constrains the scheduler to queue the request until a contiguous range becomes available.

`immediate=0|1` With a value of 1, this specifies that the request should fail if it cannot be met immediately (this is the same as the `-I` option).

`rails=`*n* In a multirail system, this specifies the number of rails required. For example:

```
$ allocate -N4 -Rrails=2
...
```

`railmask=`*mask* In a multirail system, this specifies which rails are required using a bitmask to select them. Set bit `1«r` in the mask to select rail `r`. Valid values of the rail identifier `r` run from `0` to `R-1` where `R` is the number of rails in the system. For example to select rail one:

```
$ allocate -N2 -Rrailmask=2
...
```

`samecpus=0|1` With a value of 1, this specifies the same set of CPUs on each node, for example, the first two CPUs, 0 and 1, on each node.

Multiple requests can be entered as a comma-separated list, for example, `-R hwbcast=1,immediate=1`.

`-v` Specifies verbose operation.

DESCRIPTION

The `allocate` program allocates resources for subsequent use by the `prun(1)` command. `allocate` is intended for use where a user wants to run a sequence of commands or several programs concurrently on the same set of CPUs.

The `-p`, `-N`, `-C`, `-B` and `-n` options control which CPUs are allocated. The `-N` option specifies how many nodes are to be allocated. When this option is specified the user is allocated a constant number of CPUs per node (default 1). The `-C` option specifies the number of CPUs required per node. The alternative `-n` option specifies the total number of CPUs to allocate. It does not force the allocation of a constant number of CPUs per node.

The `-B` option specifies the base of a range of nodes. The `-N` option specifies its extent. So for example, `-B0 -N4` specifies four nodes from base node 0. Note that nodes that have been configured out are excluded. The `-B` option should be used to gain access to a specific file system or device that is not available on all nodes. If the `-B` option is used, the scheduler allocates the same number of CPUs on each node. Using this option causes

a request to block until the base node and any additional nodes required to run the program are free (unless the `-I` has been given).

The `-p` option specifies the partition from which CPUs can be allocated. CPUs cannot be allocated from the `root` partition.

The Partition Manager, `pmanager`, allocates processing resources to users as and when the resources are requested and become available (see Section 4.4). By default, a contiguous range of nodes is allocated to the request where possible. This enables programs to take advantage of the system's hardware broadcast facilities. The `-R` option can be used with `hwbcast` set to 1 to ensure that the range of nodes allocated is contiguous.

Before allocating resources, the Partition Manager checks the resource limits imposed on the current project. The project can be specified explicitly with the `-P` option. This overrides the value of the environment variable `RMS_PROJECT` or any default setting in the `users` table (see Section 11.5.32).

The *`script`* argument (with optional arguments) can be used in two different ways, as follows:

1. *`script`* is not specified, in which case an interactive command shell is spawned with the resources allocated to it. The user can confirm that resources have been allocated to an interactive shell by using the `rinfo` command (see Page 5-41).

   The resources are reserved until the shell exits or until a time limit expires, whichever happens first. The system administrator can set a limit on the time for which CPUs may be allocated (see Section 11.5.17) or on the time for which allocated CPUs are idle (see `pmanager-idletimeout` attribute Section 11.5.4).

   Parallel programs, executed from this interactive shell, run on the CPUs allocated with `allocate`. They can run sequentially or concurrently; once CPUs have been allocated the user can choose how to use them.

2. *`script`* specifies a shell script, in which case the resources are allocated to the named subshell and freed when execution of the script completes.

ENVIRONMENT VARIABLES

The following environment variables may be used to identify resource requirements and modes of operation to `allocate`. They are used where no equivalent command line options are given.

`RMS_DEBUG` Specifies whether to execute in verbose mode and display diagnostic messages. The value is a mask taking values between `0` (no debug) and `2047`. Setting this variable will generate additional information

that may be useful in diagnosing problems (see Section 10.11). If this environment variable is not set, the `-v` option enables reporting of resource request debugging information.

`RMS_IMMEDIATE` Controls whether to exit (value `1`) rather than block (value `0`) if resources are not immediately available. The `-I` option overrides the value of this environment variable. By default, `allocate` blocks until resources become available. `root` resource requests are always met.

`RMS_MEMLIMIT` Specifies the maximum amount of memory required in megabytes. This must be less than or equal to the limit set by the system administrator.

`RMS_PARTITION` Specifies the name of a partition. The `-p` option overrides the value of this environment variable.

`RMS_PROJECT` Specifies the name of the project with which the request should be associated for accounting purposes. The `-P` option overrides the value of this environment variable.

`RMS_NRAILS` Specifies the number of rails (layers of switch network) required.

`RMS_TIMELIMIT` Specifies the execution time limit in seconds. The program will be signaled either after this time has elapsed or after any time limit imposed by the system has elapsed. The shorter of the two time limits is used.

`SHELL` Specifies the user shell to execute when no command script is given.

`allocate` passes all existing environment variables through to the shell that it executes. In addition, it sets the following environment variable:

`RMS_RESOURCEID` The identifier of the allocated resource.

`RMS_MACHINE` The RMS machine name, for example `atlas`.

EXAMPLES

To run a sequence of jobs on the same CPUs:

```
$ allocate -n 16 jobscript
```

where `jobscript` is a shell script such as the following:

```
#!/bin/sh
# simple job script
prun -n 16 program1
prun -n 16 program2
```

If the script was run directly then each resource request would block until resources became available and there would be no guarantee of both requests using the same CPUs. By running the script under `allocate`, there is only one resource request and both jobs are run on the same CPUs.

To run two programs on the same CPUs at the same time:

```
$ allocate -N 16 -C 2 << EOF
prun program1 &
prun program2 &
rinfo
wait
EOF
```

SEE ALSO

`prun`, `rinfo`

NAME

msqladmin – Perform administrative operations on the mSQL database server

SYNOPSIS

msqladmin [-q] [-f *confFile*] [-h *host*] *command*

OPTIONS

`-f` *confFile*	Specify a non-default configuration file to be loaded.
`-h` *host*	Specify a remote hostname or IP address on which the mSQL server (`msql3d`) is running. The default is to connect to a server on the local host using a UNIX domain socket rather than TCP/IP.
`-q`	Put `msqladmin` into quiet mode. If this flag is specified, `msqladmin` will not prompt the user to verify dangerous actions (such as dropping a database).

DESCRIPTION

`msqladmin` is used to perform administrative operations on the mSQL database used by RMS. Such tasks include the creation of databases, performing server shutdowns and so on. The available commands for `msqladmin` are:

`create` *db_name*	Creates a new database called *db_name*.
`drop` *db_name*	Removes the database called *db_name* from the server. This will also delete all data contained in the database specified.
`shutdown`	Shuts the mSQL server down.
`reload`	Forces the server to reload ACL (access control list) information.
`version`	Displays version and configuration information about the currently running server.
`stats`	Displays server statistics.

Most administrative functions can only be executed by the user specified in the run-time configuration as the admin user (`rms`). They can also only be executed from the host on which the server process is running (for example, you cannot shut down a remote server process).

EXAMPLES

When used with the `version` command `msqladmin` displays version and configuration information:

```
# msqladmin version

Version Details :-

        msqladmin version       3.4
        mSQL server version     3.4
        mSQL protocol version   23
        mSQL connection         Localhost via UNIX socket
        Target platform         OSF1

Configuration Details :-

        Default config file     /var/rms/msql.conf
        TCP socket              1114
        UNIX socket             /var/rms/adm/msql/msql3d.sock
        mSQL user               rms
        Admin user              rms
        Install directory       /var/rms
        PID file location       /var/rms/adm/msql/msql3d.pid
        Memory Sync Timer       60
        Hostname Lookup         False
        Backend Processes       4
```

When used with the `stats` command `msqladmin` displays statistics for the server and the connected clients:

```
# msqladmin stats
atlas0: msqladmin stats

Server Statistics
-----------------

Mini SQL Version 3.4
Copyright (c) 1993-94 David J. Hughes
Copyright (c) 1995-2002 Hughes Technologies Pty Ltd.
All rights reserved.

Config file      : /var/rms/msql.conf
```

```
Max connections  : 3844
Cur connections  : 7
Peak connections : 12
Connectns aborted: 0
Query count      : 7248
Backend processes: 0
Running as user  : rms
Server uptime    : 0 days, 0 hours, 44 mins, 57 secs
Connection count : 1102

Connection table :
  Sock  Username  Hostname     Procname             Database      Connect   Idle   Queries
 +-----+---------+------------+--------------------+-------------+---------+------+---------+
 |   5 | rms     | UNIX Sock  | eventmgr           | rms_atlas   |  0H 44M |   34 |      48 |
 |  31 | rms     | UNIX Sock  | tlogmgr            | rms_atlas   |  0H 44M |    0 |     114 |
 |  61 | rms     | UNIX Sock  | pid: 529184        | rms_atlas   |  0H 44M |    0 |      46 |
 |  62 | rms     | UNIX Sock  | mmanager           | rms_atlas   |  0H 44M |    0 |     363 |
 |  65 | rms     | UNIX Sock  | swmgr              | rms_atlas   |  0H 44M |   34 |      55 |
 |  72 | rms     | UNIX Sock  | pmanager[parallel] | rms_atlas   |  0H 44M |    0 |     716 |
 |  86 | rms     | UNIX Sock  | (unknown)          | No DB       |  0H  0M |    0 |       1 |
 +-----+---------+------------+--------------------+-------------+---------+------+---------+
```

NAME

nodestatus – Gets or sets the RMS status or run level of a node

SYNOPSIS

nodestatus [-bhr] [*status*]

OPTIONS

-b	Background the process.
-h	Display the list of options.
-r	Get/set run level.

DESCRIPTION

The nodestatus command is used to update status information in the RMS database as nodes are booted or halted. When run without arguments, nodestatus gets the status of the node on which it is running from the Machine Manager. When run with the -r flag, nodestatus gets the current run level.

When nodestatus is run with the *status* argument, it updates the node's status or, if the -r flag is set, it updates the node's run level. The change is reflected in the nodes table for the node on which the command is running (see Section 11.5.15). This mechanism is used to track the progress of booting a node. Administrative privileges are required to update the status or run level of a node.

The *status* can be one of these values: unknown, not responding, active, running or configured out.

Status updates may be delayed if the node running the database server is down. If background operation is specified with the -b option, nodestatus runs in the background and keeps trying until the database server is up and running.

NAME

prun – Runs a parallel program

SYNOPSIS

```
prun [-IOstv] [-B base] [-c cpus] [-e mode] [-i mode] [-l tag]
     [-m block | cyclic] [-N nodes | all] [-n procs] [-o mode]
     [-P project] [-p partition] [-R request] program [args ...]
prun [-dtv] -p root | -r [-B base] [-H hostnamelist] [-l tag]
     [-N nodes] [-R request] program [args ...]
prun -f procfile [-EItv] [-e mode] [-i mode] [-l tag] [-o mode]
     [-P project] [-p partition]
prun [-h]
```

OPTIONS

`-B` *base*	Specifies the number (node id) of the base node (the first node to use). By default, the base node is unassigned, leaving the scheduler free to select nodes that are not in use.
`-c` *cpus*	Specifies the number of CPUs required per process (default 1).
`-d`	Distribute `rmsloader` to `/tmp` on each node prior to execution of the program. This option allows RMS to be used to install itself across a machine. This option can only used with the `-r` or `-p root` options and as such is restricted to admin users.
`-E`	*Explain* a procfile. This option can only be used with the `-f` option.
`-e` *mode*	Specifies how standard error output is redirected. Valid values for *mode* and their meanings are described below.
`-f` *procfile*	Generate a resource request from instructions in the *procfile*, allocate CPUs and start sufficient instances of `pfloader` to load the processes specified in the file. The format of a procfile is described below.
`-H` *hostlist*	Run processes on listed hosts. This option can only used with the `-r` or `-p root` options and as such is restricted to admin users.

`-h` Display the list of options.

`-I` Allocate CPUs immediately or fail. By default, `prun` blocks until resources become available.

`-i` *mode* Specifies how standard input is redirected. Valid values for *mode* and their meanings are described below.

`-l` *tag* Specify a tag to prefix each line of output.

`-m block|cyclic`
Specifies whether to use block (the default) or cyclic distribution of processes over nodes.

`-N` *nodes*`|all` Specifies the number of nodes required. To use all nodes in a partition, select the `all` argument as follows: `prun -N all`. If the number of nodes is not specified then the RMS scheduler will allocate one CPU per process.

`-n` *procs* Specifies the number of processes required. The `-n` and `-N` options can be combined to control how processes are distributed over nodes. If neither is specified, `prun` starts one process.

`-O` Allows resources to be over-committed. Set this flag to run more than one process per CPU.

`-o` *mode* Specifies how standard output is redirected. Valid values for *mode* and their meanings are described below.

`-P` *project* Specifies the name of the project with which the job should be associated for scheduling and accounting purposes.

`-p` *partition* Specifies the partition on which to run the program. By default, the partition specified in the `default-partition` attribute in the `attributes` table is used. The default is `parallel` (see Section 11.5.4).

`-R` *request* Requests a particular configuration of resources. The types of *request* currently supported are as follows:

`cluster=`*cluster* `|` `cluster-member=`*cluster-member*
This requests that all processes be run on the named *cluster* or on all nodes with the specified *cluster-member* ID.

`force=0|1` This is used only with the `-r` or `-p root` options. With a value of 1, this specifies that processes should be run on configured out nodes as well as configured in ones.

`hwbcast=0|1` With a value of 1, this specifies a contiguous range of nodes and constrains the scheduler to queue the request until a contiguous range of nodes becomes available.

`immediate=0|1` With a value of 1, this specifies that the request should fail if it cannot be met immediately (the same as the `-I` option).

`maxcpus=`*n* Specifies the minimum number of CPUs to be requested when `prun` is used with the `-O` option.

`mincpus=`*n* Specifies the minimum number of CPUs to be requested when `prun` is used with the `-O` option.

`pty=0|1` With a value of 1, this specifies that RMS should create a `pty` for process `0` of the parallel program. Input to `prun` is passed raw to this process. Output from this process is returned raw. All signals caught by `prun` are forwarded. For example to allocate a single CPU and run a remote shell:

```
$ prun -n1 -B4 -Rpty=1 csh
$ hostname
atlas4
```

`rails=`*n* In a multirail system, this specifies the number of rails required. For example:

```
$ prun -N2 -Rrails=1 tping -fdping 64k
  0:    65536 bytes    501.35 uSec   261.44 MB/s
$ prun -N2 -Rrails=2 tping -fdping 64k
  0:    65536 bytes    317.64 uSec   412.64 MB/s
```

`railmask=`*mask* In a multirail system, this specifies which rails are required using a bitmask to select them. Set bit `1«r` in the mask to select rail `r`. Valid values of the rail identifier `r` run from `0` to `R-1` where `R` is the number of rails in the system. For example to select rail one:

```
$ prun -N2 -Rrailmask=2 myprog
...
```

`samecpus=0|1` With a value of 1, this specifies the same set of CPUs on each node, for example, the first two CPUs, 0 and 1, on each node.

Multiple requests can be entered as a comma-separated list, for example, `-R hwbcast=1,immediate=1`.

`-r` Run processes using `rsh` (or `ssh` if the Used for administrative operations such as starting and stopping RMS. Always uses the `root` partition and runs one process per node. If the `-d` option is used with `-r` then `prun` with use `rcp` (or `scp`) to distribute copies of `rmsloader` prior to use. This allows `prun` to be used to install RMS.

`-s` Print statistics as the job exits.

`-t` Prefix output with the process number (a special case of the `-l` option).

`-T` *rid* TotalView support, run one process per node on a named resource.

`-v` Specifies verbose operation. Repeated use of the `-v` option increases the level of output: `-vv` shows each stage in running a program; `-vvv` enables logging of debugging output from the `rmsloader` processes on each node.

DESCRIPTION

The `prun` program executes multiple copies of the specified *program* on a partition. `prun` automatically requests resources for the program unless it is executed from a shell that already has resources allocated to it (see Page 5-3).

The way in which processes are allocated to CPUs is controlled by the `-c`, `-n`, `-p`, `-B` and `-N` options or by the `-f` option. The `-n` option specifies the total number of processes to run. The `-c` option specifies the number of CPUs required per process, this defaults to 1. The `-N` option specifies how many nodes are to be used.

If the `-N` option is not used then the scheduler selects CPUs for the program from any of the available nodes. By default, RMS will allocate a contiguous range of nodes where possible. This enables programs to take advantage of the system's hardware broadcast facilities. To constrain RMS to make the range contiguous, use the `-R hwbcast=1` option. If the `-N` option is used, the scheduler allocates the specified number of nodes (in a contiguous range, if possible) and the same number of CPUs on each node.

The `-B` option specifies the base of a range of nodes. The `-N` option specifies its extent. So for example, `-B0 -N4` specifies four nodes from base node 0. Note that nodes that have been configured out are excluded. The `-B` option should be used to gain access to a specific file system or device that is not available on all nodes. If the `-B` option is used, the scheduler allocates the same number of CPUs on each node. Using this option causes

a request to block until the base node and any additional nodes required to run the program are free.

The `-I` option specifies that resource requests should fail if they cannot be met immediately. The default is to block until CPUs are available.

The `-m` option specifies how processes are to be distributed over nodes. The choice is between `block` (the default) and `cyclic`. If a program has `n` processes with identifiers `0,1,...n-1` distributed over `N` nodes then, in a block distribution, the first `n/N` processes are allocated to the first node and so on. If the distribution is cyclic, process 0 runs on the first node, process 1 on the second and so on until process `N-1` is placed on the last node, at which stage the distribution wraps around with process `N` running on the first node and so on. If `n` is not divisible by `N` then one extra process runs on each of the first `n%N` nodes.

The `-p` option specifies the partition to use. If no partition is specified then the default partition is used. The default partition is stored in the `default-partition` attribute in the `attributes` table (see Section 11.5.4). Note that use of the `root` partition (all nodes in the machine) is restricted to administrative users and is equivalent to using the `-r` option.

The `-r` or `-p root` options are used for running administrative operations on all the nodes in a machine. Processes will be run using `rsh`. By default, processes will be run on all nodes which are configured in. To run on configured out nodes as well (to start RMS on a node before configuring it in, for example) use the `-Rforce=1` option. Use the `-H` *`hostnamelist`* option or the `-B` and `-N` options to run on particular nodes. On large machines, `rsh` or `ssh` processes will be spawned so that there will be a maximum of 25 running concurrently.

The `-d` option is used only with the `-r` or the `-p root` options. If the `-d` option is used, a copy of `rmsloader` will be copied to `/tmp` on each node and executed in preference to the default. This allows `prun -r` to operate before RMS has been installed on a node. `rcp` is used to distribute the files.

The processes in a parallel program inherit their `ulimits` from `prun`. If the system administrator has set a memory limit then this may result in reduced values of the data size, stack size and virtual memory size limits.

Before allocating resources, `prun` checks the resource limits imposed on the current project. The project can be specified explicitly with the `-P` option. This overrides the value of the environment variable `RMS_PROJECT` or any default setting in the `users` table (see Section 11.5.32).

The `-f` option instructs `prun` to generate a resource request from the information in a *`procfile`*. This file specifies the processes required and their distribution. When resources have been allocated, `prun` starts the requisite number of instances of the

pfloader application loader to load and run the processes.

Each line in the file may take one of the two forms:

number hostname[:path] executable [args]
number index[:path] executable [args]

The meaning of these lines is as follows:

number This indicates the number of processes required. It can be specified in two ways:

1. A triplet of numbers in the form: *lowest*;*highest*;*step* is used to specify explicitly the ranks required for processes. The first process is assigned the rank *lowest*, the next has the rank *lowest* + *step*, the next *lowest* + (*step**2) and so on until the rank exceeds *highest*.
2. A single number that specifies how many processes are required with no indication of their ranks.

RMS assigns all of the ranks specified explicitly first. The remaining ranks are then allocated to processes specified in the second way. The assignments in each category are carried out in the order in which the lines appear in the file. prun reports an error if any of the ranks specified overlap or if a rank is missing.

hostname This specifies which node will host the processes.

index The node id of the host on which the processes will be executed.

path The working directory for the processes is set to *path*.

executable This specifies the executable to run.

arguments The *arguments* are passed to the processes.

The -f option can be combined with the options shown in the synopsis. It can also be used with the -R option and the following requests: immediate and rails. If other options are specified, prun silently ignores them. An additional -E option can be used in conjunction with the -f option. This instructs prun to explain how the processes will be distributed without actually launching any processes.

The -T option to prun is provided to support the TotalView bulk startup mechanism. When used with the id of a resource for which CPUs have been allocated prun will start one copy of the application process on each of the allocated nodes. Use of the -T option is not restricted to TotalView, you might use it, for example, to run the ps command on the same nodes as an application is using.

By default, when running a parallel program, `prun` forwards standard input to process `0`. The `-i` option requests a different mode of operation. Valid values for *mode* and their meanings are as follows:

rank — Forward standard input to the process *rank* where `0` ≤ *rank* < `n` and `n` is the number of processes in the program.

`all` — Broadcast standard input to all of the processes.

`none` — Do not forward standard input.

file — `prun` opens the named *file* and associates it with the standard input stream so that each process reads standard input from the file. If the file does not exist, a read returns `EOF`.

file.% — `prun` expands the % character to generate and open a separate file name for each process: process 0 reads standard input from *file*`.0`, process 1 reads standard input from *file*`.1` and so on. If the file does not exist, a read returns `EOF`.

If the *mode* is *rank* or `all`, `prun` polls its standard input and forwards the data to the `rmsloader` of the application process (or processes if the *mode* is `all`). `rmsloader` writes the data to the standard input pipe for the process. This write may fail if the pipe is full (the application has not read the data). If this happens, `rmsloader` will periodically attempt to resend the data to the pipe. `prun` will not poll for further standard input until it has received an acknowledgement from the process (or all processes in the case of broadcast input) to say that this operation has completed.

The `-o` and `-e` options control the redirection and filtering of standard output and standard error respectively. Valid values for *mode* and their meanings for these options are as follows:

rank — Redirect to `prun` standard output (or standard error) from the process identified by *rank* where `0` ≤ *rank* < `n` and `n` is the number of processes in the program.

`all` — Redirect standard output (or standard error) from all processes to `prun`. This is the default.

`none` — Do not redirect standard output (or standard error) from any process.

file — `prun` opens the named *file* for output and associates it with the standard output (standard error) stream so that each process writes standard output (standard error) to the file.

file.% prun expands the % character to generate and open for output a separate file name for each process: process 0 writes standard output (standard error) to *file*.0, process 1 writes to *file*.1 and so on.

Standard output from a parallel program is line-buffered and redirected to prun when a newline character is received. Output that does not end in a newline is buffered by rmsloader.

Standard error is unbuffered and forwarded to prun as soon as it is received by rmsloader.

There is no global synchronization of output from a parallel program. If multiple processes output data, the order in which the data is output will not necessarily be the same each time the program is run. The *elanrte* functions described in the *Elan Programming Manual* provide for ordered output.

prun provides support for prefixing each line of output from a parallel program with a process dependent tag. The -t option generates a tag identifying the process by its rank. The -l option allows a more general character string prefix to be specified. The following special characters in the string are expanded:

%g	the global rank of the process
%G	the number of processes
%h	the rank of the node
%H	the number of nodes
%l	the rank of the process on this node
%L	the number of processes on the node
%m	text base of the hostname
%n	numeric part of the hostname
%p	the pid of the process
%t	the current time (hh:mm:ss)
%@	the hostname of the node (the same as "%m%-n")
%%	a single %

The -t option to prun is equivalent to a label of "%_g ". By default prun pads numeric fields with leading zeros to aid tools that parse this output. This behaviour can be

adjusted by adding a modifier after the '%' as follows: hyphen ('-') do not pad, underscore ('_') pad with spaces.

prun forwards certain signals to the processes in a parallel application (SIGUSR1, SIGUSR2, SIGQUIT and SIGWINCH by default). If the pty request flag is set all catchable signals are forwarded. Signals can also be sent using rcontrol or the RMS calls rms_killJob() and rms_killResource(). Note however that signals sent in this way will be delivered to all of the processes in the parallel program including rmsloader, which may cause rmsloader to be killed and the parallel program to be terminated. The following signals can be sent by rcontrol without (necessarily) terminating the parallel program: SIGTERM, SIGINT, SIGHUP, SIGUSR1, SIGUSR2, SIGXCPU, SIGWINCH and SIGQUIT.

prun exits when all of the processes in the parallel program have exited or when one process has been killed. If all processes exit cleanly then the exit status of prun is the global OR of their individual exit status values. If one of the processes is killed, prun will exit with a status value of 128 plus the signal number. prun can also exit with the following codes:

0 An application process exited with the code 127 (GLOBAL EXIT), which indicates success, causing all of the processes to exit.

123 An application process exited with the code 123 (GLOBAL ERROR) causing all of the processes to exit.

124 The job was killed when one or more of the nodes it was using crashed or was configured out.

125 One or more processes were still running when the exit timeout expired.

126 prun was run with the -I option and resources were not available.

127 prun was run with invalid arguments.

Note

Users of the Tru64 UNIX shell csh should be aware that its treatment of exit codes of 128 or greater is different from that of other shells. The shell variable $status is set to signal number minus 128. For example if a parallel program is killed with signal 11 the prun exit code will be -117 rather than 139.

If an application process started by prun is killed, RMS will run a post mortem core analysis script that generates a backtrace if it can find a core file for the process.

Core files are saved in the directory *local-corepath/resource-id*. The value of *local-corepath* is defined in the attributes table. The *resource-id* can be listed

by `rinfo` (see Page 5-41). `prun` also sets the environment variable `RMS_RESOURCE_ID` to the value of the resource identifier.

The attribute `rms-keep-core` in the `attributes` table determines whether core files are saved (see Section 11.5.4). The environment variable `RMS_KEEP_CORE` can be set to override the value in the `attributes` table.

If the system has been set up to run `prologue` or `epilogue` scripts (see Section 9.5.2) then they will run before or after your job respectively. If the `prologue` script fails, your program will not run and the following message will be displayed:

```
prun: Error: allocate failed: atlas0 /usr/local/rms/prologue status 1
```

The node will be configured out if this happens (failure to run the prologue script correctly indicates a serious problem with a node).

ENVIRONMENT VARIABLES

The following environment variables may be used to identify resource requirements and modes of operation to `prun`. These environment variables are used where no equivalent command line options are given:

`RMS_DEBUG`	Whether to execute in verbose mode and display diagnostic messages. Setting a value of `1` or more generates additional information that may be useful in diagnosing problems (see Section 10.11).
`RMS_EXITTIMEOUT`	Specifies the time allowed in seconds between the first process exit and the last. This option can be useful in parallel programs where one process can exit leaving the others blocked in interprocess communication. It should be used in conjunction with an exit barrier at the end of correct execution of the program. A permanent exit timeout can be set by setting the `exit-timeout` system attribute.
`RMS_IMMEDIATE`	Controls whether to exit rather than block if resources are not immediately available. The `-I` option overrides the value of this environment variable. By default, `prun` blocks until resources become available. Requests to use the `root` partition do not block, but they are restricted to system admin users.
`RMS_IOLABEL`	Specifies a process dependent prefix for each line of output.

RMS_KEEP_CORE	Controls whether core files are saved. This overrides the default behaviour set by the system administrator (in the `rms-keep-core` attribute in the `attributes` table.
RMS_MEMLIMIT	The maximum amount of memory required per process in megabytes. This must be less than or equal to any limit set by the system administrator. See Section 6.4 for more information on memory limits.
RMS_NRAILS	Specifies the number of HP AlphaServer SC Interconnect rails required.
RMS_PARTITION	Specifies the name of a partition. The `-p` option overrides the value of this environment variable.
RMS_PROJECT	The name of the project with which the job should be associated for scheduling and accounting purposes. The `-P` option overrides the value of this environment variable.
RMS_RSH_LIMIT	Specifies the number of concurrent `rsh` processes started when using the `-r` or `-p root` options only. The `rsh` processes exit as soon as `rmsloader` is running, but having large numbers of them running at the same time may degrade system performance. The default value (25) allows `prun -r` to execute reasonably quickly on a large machine.
RMS_SSH	Specifies that the `-r` or `-p root` options will start processes with `ssh` instead of `rsh`.
RMS_STATINTERVAL	Specifies the rate at which the `statsig` signal is delivered to the application processes. Delivery of this signal will prompt `libelan` to dump performance statistics. Set the environment variable to the time in seconds between samples. The signal to be delivered is specified using the `libelan` environment variable `LIBELAN_STATSIG`.
RMS_STDINMODE	Specifies the *mode* for forwarding standard input to a parallel program. The `-i` option overrides the value of this environment variable. Values for *mode* are the same as those used with the `-i` option.
RMS_STDOUTMODE	Specifies the *mode* for redirecting standard output from a parallel program. The `-o` option overrides the value of this environment variable. Values for *mode* are the same as those used with the `-o` option.

RMS_STDERRMODE	Specifies the *mode* for redirecting standard error from a parallel program. The -e option overrides the value of this environment variable. Values for *mode* are the same as those used with the -e option.
RMS_TIMELIMIT	Specifies the CPU allocation time limit in seconds. The program will be signaled either after this time has elapsed or after any time limit imposed by the system has elapsed. The shorter of the two time limits is used. Note that this timelimit does not apply where prun is used on preallocated CPUs.

prun passes all existing environment variables through to the processes that it executes. In addition, it sets the following environment variables:

RMS_JOBID	The identifier for the job.
RMS_NNODES	The number of nodes used by the application.
RMS_NODEID	The logical identifier of the node within the set allocated to the application.
RMS_NPROCS	The total number of processes in the application.
RMS_PROCFILERANK	The procfile rank of the process in the application when the -f *procfile* option has been used.
RMS_RANK	The rank of the process in the application. The rank ranges from 0 to n-1, where n is the number of processes in the program.
RMS_RESOURCEID	The identifier of the allocated resource.

EXAMPLES

In the following example, prun is used to run a four-process program with no specification of where the processes should run.

```
$ prun -n 4 hostname
atlas0
atlas0
atlas0
atlas0
```

The machine atlas has four CPUs per node and so, by default, the scheduler allocates all four CPUs on one node to run the program. Add the -N option, as follows, to control how the processes are distributed over nodes.

```
$ prun -n 4 -N 2 hostname
atlas0
atlas0
atlas1
atlas1
$ prun -n 4 -N 4 hostname
atlas1
atlas3
atlas0
atlas2
```

The `-m` option controls how processes are distributed over nodes. It is used in the following example in conjunction with the `-t` option which tags each line of output with the identifier of the process that wrote it.

```
$ prun -t -n 4 -N 2 -m block hostname
0 atlas0
1 atlas0
2 atlas1
3 atlas1
$ prun -t -n 4 -N 2 -m cyclic hostname
0 atlas0
2 atlas0
1 atlas1
3 atlas1
```

The following examples illustrate the use of the `-l` option to prefix each line of output with a process dependent tag:

```
$  prun -N4 -l "%g: " hostname
1: atlas1
3: atlas3
0: atlas0
2: atlas2

$ prun -N4 -n16 -l "%g (%n.%l) " hostname | sort
00 (0.0) atlas0
01 (0.1) atlas0
02 (0.2) atlas0
03 (0.3) atlas0
04 (1.0) atlas1
05 (1.1) atlas1
06 (1.2) atlas1
07 (1.3) atlas1
08 (2.0) atlas2
09 (2.1) atlas2
10 (2.2) atlas2
11 (2.3) atlas2
12 (3.0) atlas3
13 (3.1) atlas3
```

```
14 (3.2) atlas3
15 (3.3) atlas3

$ prun -N4 -l "%t " hostname
11:50:42 atlas1
11:50:42 atlas0
11:50:42 atlas2
11:50:42 atlas3
```

The examples so far have used simple UNIX utilities to illustrate where processes are run. Parallel programs are run in just the same way. The following example measures DMA performance between a pair of processes on different nodes.

```
$ prun -N 2 dping 0 1k
  0:        0 bytes      1.88 uSec     0.00 MB/s
  0:        1 bytes      2.74 uSec     0.36 MB/s
  0:        2 bytes      2.64 uSec     0.76 MB/s
  0:        4 bytes      1.98 uSec     2.02 MB/s
  0:        8 bytes      1.98 uSec     4.04 MB/s
  0:       16 bytes      2.04 uSec     7.85 MB/s
  0:       32 bytes      2.09 uSec    15.31 MB/s
  0:       64 bytes      2.97 uSec    21.57 MB/s
  0:      128 bytes      3.08 uSec    41.55 MB/s
  0:      256 bytes      3.37 uSec    75.96 MB/s
  0:      512 bytes      4.32 uSec   118.59 MB/s
  0:     1024 bytes      5.89 uSec   173.96 MB/s
```

The `-s` option instructs `prun` to print a summary of the resources used by the job when it finishes.

```
$ prun -s -N 2 dping 0 32
  0:        0 bytes      1.87 uSec     0.00 MB/s
  0:        1 bytes      2.76 uSec     0.36 MB/s
  0:        2 bytes      2.66 uSec     0.75 MB/s
  0:        4 bytes      1.98 uSec     2.02 MB/s
  0:        8 bytes      1.97 uSec     4.05 MB/s
  0:       16 bytes      2.03 uSec     7.88 MB/s
  0:       32 bytes      2.07 uSec    15.43 MB/s
Elapsed time         0.97 secs    Allocated time       1.94 secs
User time            0.68 secs    System time          0.21 secs
Cpus used               2
```

Note that the allocated time (in CPU seconds) is twice the elapsed time (in seconds) because two CPUs were allocated.

The `-f` option is used in the following examples to show how a file can be used to specify different distributions of processes.

A file with the following lines specifies an uneven block distribution.

```
3 atlas0 a.out -x 1
4 atlas1 a.out -x 1
2 atlas2 a.out -x 1
3 atlas3 a.out -x 1
```

Three processes (with ranks 0-2) will run on `atlas0`, four processes (ranks 3-6) will run on `atlas1`, two processes (ranks 7-8) will run on `atlas2` and three processes (ranks 9-11) will run on `atlas3`. All of the processes will execute `a.out` with the arguments `-x 1`.

The following lines specify a master-slave configuration of processes.

```
1 atlas0 master
3 atlas0 slaves
```

Process 0 will be an instance of `master`; processes 1-3 will be instances of `slaves`.

A cyclic distribution is achieved by the following lines:

```
0;15;4 atlas5 a.out -y 0
1;15;4 atlas6 a.out -y 0
2;15;4 atlas7 a.out -y 0
3;15;4 atlas8 a.out -y 0
```

Processes 0, 4, 8 and 12 will be assigned to `atlas5`; processes 1, 5, 9 and 13 will be assigned to `atlas6`; processes 2, 6, 10 and 14 will be assigned to `atlas7` and processes 3, 7, 11 and 15 will be assigned to `atlas8`.

The final example sets some of the ranks explicitly:

```
0;0;1 atlas6 source
5;5;1 atlas7 sink
4     atlas8 worker
```

Process 0 will be an instance of `source`, process 5 will be an instance of `sink` and processes 1-4 will be instances of `worker`.

SEE ALSO

`allocate`, `rinfo`

NAME

rcontrol – Controls use of system resources

SYNOPSIS

rcontrol [-ehs] [-m *machine*] [-r *level*] [*command args ...*]

OPTIONS

`-e`	Exit on the first error.
`-h`	Display the list of options.
`-m` *machine*	Select database by machine name.
`-r` *level*	Set reporting level.
`-s`	Stop and print warning on error.

command is specified as follows:

`create` *object* [=] *name* [configuration=*val*] [partition=*val*] [attr=*val*]

object may be one of: `access_control`, `attribute`, `configuration`, `node`, `partition`, `project`, `user`. If an `access_control` is specified, a `partition` must also be named to identify the object uniquely. Similarly, if a `partition` is specified, a `configuration` must also be named together with a list of nodes.

`remove` *object* [=] *name* [configuration=*val*] [partition=*val*]

object may be one of: `access_control`, `attribute`, `configuration`, `node`, `partition`, `project`, `user`. If an `access_control` is specified, a `partition` must also be named to identify the object uniquely. If a `partition` is specified, a `configuration` must also be named to identify the object uniquely.

`drain nodes[=]` *list*

list specifies a quoted list of nodes (such as `'atlas[1-3,6,8]'`) to drain.

`configure in nodes[=] list`

list specifies a quoted list of nodes (such as `'atlas[1-3,6,8]'`) to configure in.

`configure out nodes[=] list [reason[=]text]`

list specifies a quoted list of nodes (such as `'atlas[1-3,6,8]'`) to configure out.

`start object [=] name [option [=] kill | wait]`

object may be one of: `configuration`, `partition`, `server`. An `option` is only allowed if *object* is `configuration`.

`stop object [=] name [option [=] kill | wait]`

object may be one of: `configuration`, `partition`, `server`. If `server` is specified as the *object*, no `option` should be given.

`restart object [=] name [option [=] kill | wait]`

object may be one of: `partition`, `server`. If `server` is specified as the *object*, no `option` should be given.

`reload object [=] [host:]name [debug [=] value]`

object may be one of: `partition`, `server`. *host* is only relevant for servers which run on more than one node.

`suspend resource [identifier] [=] name [name ...]`

identifier may be one of: `name` or `batchid`. Where *identifier* is omitted it defaults to `name`.

`suspend resource attribute [=] value [attribute [=] value ...]`

Attributes of the same name are `OR`ed together. Attributes with different names are `AND`ed together. The result of the logical expression identifies a resource or set of resources as the target of the command.

`resume resource [identifier] [=] name [name ...]`

identifier may be one of: `name`, `batchid`. Where *identifier* is omitted it defaults to `name`.

`resume resource attribute [=] value [attribute [=] value ...]`

Attributes of the same name are `OR`ed together. Attributes with different names are `AND`ed together. The result of the logical expression identifies a resource or set of resources as the target of the command.

`kill resource [`*`identifier`*`] [=] `*`name`*` [`*`name ...`*`]  [signal [=] `*`sig`*`]`

`identifier` may be one of: `name`, `batchid`. Where *`identifier`* is omitted it defaults to `name`.

`kill resource `*`attribute`*`[=]`*`value`*` [`*`attr`*`[=]`*`value ...`*`]  [signal[=]`*`sig`*`]`

Attributes of the same name are `OR`ed together. Attributes with different names are `AND`ed together. The result of the logical expression identifies a resource or set of resources as the target of the command.

`kill job [name] [=] `*`name`*` [`*`name ...`*`]  [signal [=] `*`sig`*`]`

`set resource `*`identifier`*` [=] `*`name`*` priority [=] `*`value`*

`identifier` may be one of: `name`, `batchid`.

`set `*`object`*` [=] `*`name attribute`*` [=] `*`value`*` [`*`attribute`*` [=] `*`value ...`*`]`

`object` may be one of: `access_control`, `configuration`, `node`, `partition`, `project`, `user`.

`set `*`attribute`*` [=] `*`name`*` val [=] `*`value`*

`exit`

`quit`

`help [all | `*`command`*`]`

`show `*`object`*` [=] `*`name`*

`object` may be one of: `nodes`, `configuration`, `partition`.

DESCRIPTION

`rcontrol` is used to manage the following: nodes, partitions and configurations; servers; users and their resource requests, projects and access controls; system attributes.

`rcontrol` can create, start, stop and remove a configuration or partition. It can create, remove and set the attributes of nodes and configure them in and out of the machine. Operations on nodes may specify a single host name, such as `atlas4`, or a *`list`* of host names, such as `'atlas[4-7]'`. Lists of host names must always be quoted.

`rcontrol` can start or stop an RMS server. It can also instruct a running server to reload access control information or change its reporting level.

`rcontrol` can be used to suspend or resume the allocation of CPUs to a resource request, alter its scheduling priority or send a signal to its jobs. Operations on resource

requests may specify a request by resource name or by using the batch system identifier. Alternatively, requests can be identified by attributes such as user name, partition, project or status. With the `kill` command, jobs may also be identified by their name.

`rcontrol` can be used to create or remove or to set the attributes of users, projects and access controls. Details of which attributes can be modified in this way are specified in the `fields` table in the RMS database. System attributes can also be created, removed or have their value set.

The `help` command prints information on all of the commands and their arguments. When used with the name of a command as an argument, it prints more information on the specified command.

When used without arguments, `rcontrol` runs interactively. A sequence of commands can be entered. Use the `exit` or `quit` commands or `Ctrl/d` to exit.

Most `rcontrol` commands are restricted to administrative users (`root` and `rms` users, by default). The job control commands (`suspend`, `resume`, `kill` and `set priority`) may also be issued by the user running the job in question.

In all of the `rcontrol` commands, the use of the equals sign is optional. The following two examples – using `rcontrol` to configure into the system three nodes named `atlas1`, `atlas2` and `atlas3` – are equivalent.

```
# rcontrol configure in nodes = 'atlas[1-3]'
# rcontrol configure in nodes 'atlas[1-3]'
```

Creating and Removing Nodes

To create a new node description, use `rcontrol` with the `create` command and the argument `node` followed by the hostname of the node. Additional attribute-value pairs specify properties of the node, such as its type and position. Supported node types are listed in the `module_types` table (see Table 11.23). The attributes `rack` and `unit` specify the position of the node in the system.

```
# rcontrol create node = atlas1 type = ES40 rack = 0 unit = 3
```

To remove a node description from the RMS database, use `rcontrol` with the `remove` command and the argument `node` followed by the name of the node.

```
# rcontrol remove node = atlas1
```

To create or remove multiple nodes use `rcontrol` with a hostname list rather than a single hostname. For example:

```
# rcontrol create nodes 'atlas[2-31]'
# rcontrol remove nodes 'atlas[28-31]'
```

Creating and Removing Partitions

RMS scheduling policy and access controls are based on partitions. Partitions are non-overlapping sets of nodes. The set of partitions in operation at any time is called the *active* configuration. RMS provides for several operational configurations and includes mechanisms for switching between them with `rcontrol`.

To create a new partition description, use `rcontrol` with the `create` command and the argument `partition` followed by the name of the partition. In addition, you must specify the configuration to which the partition belongs. Additional attribute-value pairs specify properties of the partition: a list of its nodes, its scheduling type, time limit, time slice interval, memory limit or minimum number of CPUs that may be allocated. The `nodes` attribute must be specified. Default values will be selected for the other attributes if none are given.

```
# rcontrol create partition = p1 configuration = day nodes = 'atlas[1-4]' \
          type = parallel
```

The scheduling `type` attribute of the partition may be one of the following:

`parallel`	The partition is for the exclusive use of gang-scheduled parallel programs.
`login`	The partition runs interactive user logins and load-balanced sequential jobs.
`general`	The partition runs all classes of job. This is the default partition type.
`batch`	The partition is for the exclusive use of a batch system.
`restricted`	Access to the partition is restricted, normally because it is controlled by an external scheduler.

The `timelimit` attribute specifies the maximum time in seconds for which CPUs in the partition can be allocated. On expiry of the time limit, jobs will be sent the signal specified by `warning-signal`, as defined in the `attributes` table. The default setting is `SIGTERM`. If the jobs have not exited within a grace period, they will be killed. The grace period for a site is defined in the `attributes` table (attribute name `grace-period`). Its default value is 60 seconds (see Table 11.10).

The `timeslice` attribute specifies the period in seconds for which jobs are allocated CPUs before the CPUs may be reallocated to another job of equal priority. The default value for `timeslice` is `NULL`, disabling time-slicing.

The `memlimit` attribute defines the default memory limit per CPU for applications running on this partition. It can be overridden on a per-user or per-project basis. The

default value of memlimit is NULL, disabling memory limits unless they are set for specific users or projects.

The mincpus attribute controls the minimum number of CPUs that may be allocated to a job running on this partition. The default value of mincpus is 0. The maximum number of CPUs that can be allocated is controlled on a per-user or per-project basis.

To remove a partition description from the RMS database, use rcontrol with the remove command and the argument partition followed by the name of the partition. You must also specify the name of the configuration since the same partition name may appear in a number of configurations. To remove an entire configuration from the RMS database, use rcontrol with the remove command and the argument configuration followed by the name of the configuration.

```
# rcontrol remove partition = par1 configuration = night
# rcontrol remove configuration = night
```

Note that partitions cannot be removed while they are in use. Similarly, the nodes and type of a partition cannot be changed while the partition is running. If the other attributes of a partition are changed while the partition is running, the Partition Manager is reloaded automatically so that it uses the new information for subsequent jobs. Jobs that are already running are not affected.

Starting and Stopping Partitions

To start a partition in the active configuration, use rcontrol with the start command and the partition argument followed by the name of the partition. To start all of the partitions in a configuration, use rcontrol with the start command and the configuration argument followed by the name of the configuration. A configuration is made active by starting it in this way.

```
# rcontrol start partition = par1
# rcontrol start configuration = day
```

To stop a partition in the active configuration, use rcontrol with the stop command and the partition argument followed by the name of the partition. To stop all of the partitions in the active configuration, use rcontrol with the stop command and the configuration argument followed by the name of the configuration.

When stopping partitions, you can optionally specify what should happen to the running jobs. The options are to leave them running, to wait for them to exit or to kill them. The default is to leave them running.

```
# rcontrol stop partition = par1 option = kill
# rcontrol stop configuration = day option = wait
```

If starting a configuration when a different configuration is already running there is an issue about what should happen to resources which are already allocated. If there is a partition with the same name in the new configuration and the allocated resource will fit inside it the resource will remain allocated and any running jobs will be resumed. If the resource will not fit in the new configuration (an *incompatible allocated resource*) the jobs must either be killed or waited for and this can be given as an option.

```
# rcontrol start configuration = par1 option = kill
# rcontrol start configuration = par1 option = wait
```

An error will be printed if `rcontrol` is used with the `start` command on a partition that is already running. For example:

```
# rinfo
MACHINE     CONFIGURATION
atlas       day

PARTITION       CPUS    STATUS        TIME   TIMELIMIT  NODES
root              16                                    atlas[0-3]
atlas           0/16   running       02:28              atlas[0-3]
# rcontrol start partition parallel
rcontrol: Error: partition parallel is already running
```

Use `rcontrol` with the `restart` command to restart a running partition.

Configuring Nodes In or Out

To configure a node in or out, use `rcontrol` with the `configure in` or `configure out` commands. Use the `nodes` argument to specify the list of nodes being configured in or out.

```
# rcontrol configure in nodes = 'atlas[2-4]'
# rcontrol configure out nodes = 'atlas[2,5-7]'
# rcontrol configure out nodes = 'atlas[12] reason = 'maintenance'
```

Use the `reason` field to give a description of why the node has been configured out. The `rinfo` command will show this field when used with the `-nl` options.

Note that a partition must be stopped before nodes can be configured in or out of it. Jobs may be left running but any jobs running on a node while it is being configured out will be killed. When stopping a partition, it is advisable to wait until jobs have exited or kill them.

When run from a terminal `rcontrol` will seek confirmation before configuring nodes out if doing so will cause jobs to be killed.

```
# configure out atlas1
rcontrol: will kill user jobs. Continue (y/n): y
```

Draining Jobs From Nodes

Configuring a node out will kill the jobs running on it. Where this is not necessary (because, for example, the node is to be taken down for preventative maintenance) the jobs can be *drained* first. To do this use `rcontrol` with the `drain` command as follows:

```
# rcontrol drain nodes = 'nodelist'
```

where *`nodelist`* is a list of nodes in the same partition. The nodes will be configured out when the current CPU allocations are completed. The nodes will not be allocated to any new or queued resource requests. For example:

```
# rcontrol drain node atlas4
```

The cancel option allows you to cancel a drain operation on one or more nodes. For example

```
# rcontrol drain node atlas4 option cancel
```

See Section 10.10 for more details on draining jobs from nodes.

Reloading Database Information

To instruct a Partition Manager to reload its `access_controls`, `users`, and `projects` tables, use `rcontrol` with the `reload` command and the `partition` argument followed by the name of the partition.

```
# rcontrol reload partition = par1
```

To instruct a Partition Manager to change its reporting level, use `rcontrol` with the `reload` command and the `partition` argument followed by the name of the partition. In addition, you should specify the attribute `debug` and a value. The Partition Manager writes its reports to a log file in the directory `/var/rms/adm/log` (see Section 4.1.2 and Section 10.11).

```
# rcontrol reload partition = par1 debug = 1
```

Managing Servers

To stop an RMS server, use `rcontrol` with the `stop` command and the `server` argument followed by the name of the server. To start it again, use `rcontrol` with the `start` command, the `server` argument and the name of the server. The command `rinfo` (with the `-s` flag) can be used to show the status of the RMS servers.

To instruct an RMS server to change its reporting level, use the `reload` command and the `server` argument with the name of the server. In addition, you should specify the

attribute debug and a value. RMS servers write their log files to the directory /var/rms/adm/log on the rmshost (see Section 10.11).

```
# rcontrol stop server = mmanager
# rcontrol start server = mmanager
# rcontrol reload server = mmanager debug = 1
```

For servers which run on more than one node an optional *host* argument can be given to reload a server running on a particular node. If no host is given all of the appropriate servers are reloaded.

```
# rcontrol reload server = atlas1:rmsd
# rcontrol reload server = rmsd
```

An error will be printed if rcontrol is used with the start command on a server that is already running. For example:

```
# plague0: rcontrol start server swmgr
rcontrol: Warning: server swmgr:plague0 already running
```

Use rcontrol with the restart command to restart a running server.

Managing Resources

All of the rcontrol commands used to manage individual resources and jobs use an *identifier* to identify a particular resource or job. For resources, the *identifier* can be either name or batchid. For jobs the identifier is always name. Where name is used the name is the resource name or job name as shown in the RESOURCE (or JOB for jobs) column of the output from rinfo. The name entered can be just the numeric id part of the identifier or the full *partition.id* string. For example, to suspend the resource shown in the rinfo output shown below the resource name would be given as parallel.22 or just 22. Similarly, the job name would be either parallel.21 or 21.

```
RESOURCE        CPUS    STATUS        TIME    USERNAME  NODES
parallel.22        8   running       00:32       tom     atlas[0-3]

JOB             CPUS    STATUS        TIME    USERNAME  NODES
parallel.21        8   running       00:32       tom     atlas[0-3]
```

Where a resource was allocated by a batch system which set its own batch id in the allocation request then this *batchid* can be used as a resource identifier.

To instruct the scheduler to suspend the allocation of CPUs to a resource request, use rcontrol with the suspend command followed by either the name of the resource or the batch system's identifier for the request. This suspends jobs running on the allocated CPUs and decrements the user's CPU usage count.

```
# rcontrol suspend resource name = 2234
# rcontrol suspend resource batchid = 14
```

Note that a resource request that has been suspended by an administrative user cannot be resumed by its owner.

To instruct the scheduler to resume the allocation of CPUs to a resource request, use `rcontrol` with the `resume` command followed by either the name of the resource or the batch system's identifier for the request. This reschedules jobs that were running on the allocated CPUs, unless doing so would cause the user's CPU usage limit to be exceeded.

```
# rcontrol resume resource name = 2267
# rcontrol resume resource batchid = 384
```

To instruct RMS to send a signal to the jobs running on an allocated resource request, use `rcontrol` with the `kill` command followed by the name of the resource, the job identifier or the batch system's identifier for the request. This kills the jobs running on the allocated CPUs (by sending the signal `SIGKILL` to each process). The optional attribute `signal` can be used to send a specific signal. The signal can be either the symbolic name of a signal (see signal(4) for a list of signal names) or its number. For example, to send the signal `SIGTERM`:

```
# rcontrol kill resource name = 9835 signal = 15
# rcontrol kill resource batchid = 396 signal = SIGTERM
# rcontrol kill job name = 4821 signal = 15
```

To instruct the scheduler to change the priority of a resource request, use `rcontrol` with the `set` command and the `resource` argument followed by either the name of the resource or the batch system's identifier for the request. In addition, you should specify the attribute `priority` and the new value. Priority values range from 0 to 100 (default 50).

```
# rcontrol set resource name = 32 priority = 25
# rcontrol set resource batchid = 48 priority = 40
```

`rcontrol` can also be used to suspend, kill or resume jobs identified by their attributes. The attributes that can be specified are: `partition`, `project`, `status` and `user`. Attributes of the same name are `OR`ed together, attributes with different names are `AND`ed.

For example, to kill a job run by a user called `tom` on the partition `par1` whether its status is blocked or queued:

```
# rcontrol kill resource user = tom status = blocked status = queued partition = par1
```

To suspend all of the jobs belonging to the project called `science`:

```
# rcontrol suspend resource project = science
```

Managing Users, Projects and Access Controls

In addition to managing partitions and nodes, `rcontrol` can be used to create, remove and set the attributes of users, projects and access controls. The `fields` table contains details of which objects and attributes may be modified (see Section 11.5.10).

The table has seven fields: the `tablename` field specifies the table that will be modified; the `name` field specifies which entry in the named table will be modified; the `type` field determines the range of valid values; the `min` field gives the minimum for values of `type` integer while the `max` field gives the maximum; the `textattr` field either gives a comma-separated list of valid values or a *`table-name.table-field`* pair. In the case of the *`table-name.table-field`* pair, the value in the `name` field of the `fields` table must also be present in the table named `table-name` in the field called `table-field`. The `access` field specifies whether this field can be updated by the system administrator.

To create a user, use the `rcontrol create` command to specify the object type (in this case, `user`) and the object name (for example, `frank`).

```
# rcontrol create user = frank
```

To update an existing user record, use the `rcontrol set` command. For example, to change the projects to which a user belongs, use `rcontrol set` followed by the object type (in this case, `user`), the object name (in this example, `frank`), the attribute to be changed (`projects`) and its new value (in this example, `parallax`); the new value must already have been defined as a project.

```
# rcontrol set user = frank projects = parallax
```

Note that a user can be in more than one project in which case the value would be a comma-separated list:

```
# rcontrol set user = frank projects = parallax,aero
```

To create an access control called, for example, `science`, in the `par1` partition, use `rcontrol` with the `create` command followed by the type of the object, its name and the name of the partition. Additional attribute-value pairs specify attributes of the access control, for example, its class.

```
# rcontrol create access_control = science partition = par1 class = project
```

Just as partitions require a configuration name to identify them uniquely, access controls require a partition name.

To set the attributes of an object, use `rcontrol` with the `set` command followed by the name of the object. Specify the name of the attribute and the required value. An attribute's value can be set to null by entering `NULL`, `Null` or `null` as the value.

```
# rcontrol set access_control = std partition = par1 priority=75 memlimit=NULL
```

To remove an object, use `rcontrol` with the `remove` command and the name of the object.

```
# rcontrol remove user = frank
# rcontrol remove access_control = science partition = par1
```

After changing user, project or access control information, the Partition Managers must be reloaded so that they use the new information.

```
# rcontrol reload partition = par1
```

Jobs that were already running will not be affected by any change to resource limits, except that they may be suspended if the new CPU usage limits are lower than before.

Setting System Attributes

System attributes can be created, removed or set using `rcontrol` `create`, `remove` and `set`.

```
# rcontrol create attribute = name val=value
# rcontrol remove attribute = name
# rcontrol set attribute = name val=value
```

Any system attributes can be modified in this way but there are some, mentioned below, whose values are checked if they are created or set (see Section 11.5.4).

The attribute `pmanager-queuedepth` limits the number of resource requests that a Partition Manager will handle at any time. If the attribute is undefined or set to `NULL` or `0`, no limit is imposed. By default, it is set to `0`.

If a limit is set and reached, subsequent resource requests by `prun` will block or, if the immediate option to `prun` is set, fail. The blocked requests will not appear in the RMS database.

To set the `pmanager-queuedepth` attribute, use `rcontrol` with the `set` command. Specify `attribute`, give the attribute name and set the `val` argument to the required value.

```
# rcontrol set attribute = pmanager-queuedepth val = 20
```

If you set a limit while the partition is running, reload the partition to make the limit take effect.

```
# rcontrol reload partition = par1
```

The attribute `pmanager-idletimeout` limits the amount of time an allocated resource may remain idle. If a resource request exceeds this limit, it will time out with an exit status of 125 and `allocate` will exit with the following message:

```
allocate: Error: idle timeout expired for resource allocation
```

If the attribute is undefined or set to `NULL`, no limit is imposed. By default, it is not set. To set a limit, use `rcontrol` with the `set` argument. Specify `attribute`, give the attribute name and set the `val` argument to the required timeout value in seconds.

```
# rcontrol set attribute = pmanager-idletimeout val = 5
```

If you set a time limit while the partition is running, you should also reload the partition to make the limit take effect.

```
# rcontrol reload partition = par1
```

The attribute `default-priority` determines the default priority given to resource requests. Priorities may range from 0 to 100. The default is 50.

To set the `default-priority` attribute, use `rcontrol` with the `set` command. Specify `attribute`, give the attribute name and set the `val` argument to the required value.

```
# rcontrol set attribute = default-priority val = 75
```

The attribute `grace-period` specifies the amount of time in seconds that jobs are given to exit after they have exceeded their time limit and received a signal to quit. It may be set to any value between 0 and 3600, the default being 60.

The attribute `cpu-stats-poll-interval` specifies the interval between successive polls for gathering node statistics. The interval is specified in seconds and must be in the range 0 to 86400 (1 day).

The attribute `rms-keep-core` determines whether core files are deleted or saved. By default, it is set to `1` so that core files are saved. Change this to `0` to delete core files. The attribute `local-corepath` specifies the directory in which core files are saved. By default, it is set to `/local/core/rms`.

EXAMPLES

The following command line creates a partition called `par1` with eight nodes called `atlas1`, `atlas2` and so on in the configuration called `day`.

```
# rcontrol create partition=par1 configuration=day nodes='atlas[1-8]'
```

The partition is started and stopped as follows:

```
# rcontrol start partition = par1
# rcontrol stop partition = par1
```

Stopping the partition in this way will leave the jobs running. Alternatives are to wait for them to exit or to kill them.

```
# rcontrol stop partition = par1 option = wait
# rcontrol stop partition = par1 option = kill
```

If the system has several operating configurations, for example, one for the prime shift (called `day`) and another for evening and weekends (called `night`) then the set of partitions making up a configuration can be started and stopped together:

```
# rcontrol stop configuration = day
# rcontrol start configuration = night
```

To suspend or resume the jobs running on a specified resource:

```
# rcontrol suspend resource name = 2212
# rcontrol resume resource name = 2212
```

To set the priority of a resource:

```
# rcontrol set name resource = 2212 priority = 4
```

To set the type of a node (or nodes):

```
# rcontrol set node = atlas22 type = DS20L
# rcontrol set nodes = 'atlas[23-31]' type = DS20L
```

To kill the jobs running on some specified resources:

```
# rcontrol kill resource name = 2212 2213 2214
# rcontrol kill resource batchid = 44 45
# rcontrol kill job name = 254
```

To instruct a Partition Manager to reread the `user`, `projects` and `access_controls` tables:

```
# rcontrol reload partition = par1
```

To enable debugging reports from the RMS scheduler for the partition called `par1`:

```
# rcontrol reload partition = par1 debug = 41
```

Enabling scheduler debug will cause scheduling decisions to be logged in the `scheduler_log` table (see Table 11.30 for details).

NAME

rinfo – Displays resource usage and availability information for parallel jobs

SYNOPSIS

```
rinfo [-chlmNnpqr] [-E class | type] [-e all |
      class:name:type [-H hours]] [-j [choice="opt">jid]]
      [-L [partition] [statistic]] [-LM] [-o orderByList] [-s daemon
      [hostname | all]] [-t node | name] [-u [userList]]
```

OPTIONS

`-c` List the configuration names. With the `-l` option, shows detailed information about configurations and partitions.

`-E class | type` Show event classes or types.

`-e all | class:name:type [-H hours]`

Show all events or events matching *`class:name:type`*. Use `rinfo` with the `-E class` option or the `-E type` option to list possible event classes and types. If `*` is used for *`class`*, *`name`* or *`type`* then all events will be matched. If the `-H` *`hours`* option is used, only events in the preceding *`hours`* hours will be shown. If the `-l` option is used, a banner is printed.

`-h` Display the list of options.

`-l` Give more detailed information.

`-m` Show the machine name.

`-N` Show the status of the HP AlphaServer SC Interconnect network error service on each node.

`-n` Show the status of each node. This can be combined with `-l`.

`-p` Identify each active partition by name and indicate the number of CPUs in each partition.

`-q` Print information on the user's quotas and projects.

`-j [jid]` With no argument, list current jobs. With a job identifier argument, display full details about the specified job. When used with the `-l` option and a job identifier argument, display the full hostnames and process IDs for the job.

`-L [partition] [statistic]`

Print the hostname of a lightly loaded node in the machine or the specified *partition*. RMS provides a load-balancing service, accessible through `rmsexec`, that enables users to run their processes on lightly loaded nodes, where loading is evaluated according to a given *statistic* (see Page 5-59) .

`-LM` Show license management information.

`-o orderByList` Display resource information selected by and in the order given by *orderByList*, which is a comma-separated list of field names from the resources table (see Table 11.29). If the `-j` option is used, job information is displayed and the list must contain fields from the jobs table (see Table 11.21). An error message is generated if the field names are invalid.

`-r` Show the allocated resources.

`-s daemon [hostname] | all`

Show the status of the *daemon*. Used with the argument `all`, `rinfo` shows the status of all daemons running on the `rmshost` node. For daemons that run on multiple nodes, such as `rmsd`, the optional *hostname* argument specifies the hostname of the node on which the daemon is running.

`-t node | name` Where *node* is the QsNet network identifier of a node, `rinfo` translates it into the hostname; where *name* is a hostname, `rinfo` translates it into the network identifier (see Section A.3 for more information on network identifiers).

`-u userList` With no argument, only display information about the user's resources. With a list of user names, only display information about the listed users' resources.

DESCRIPTION

The `rinfo` program displays information about resource usage and availability. Its default output is in four parts that identify: the machine, the active configuration,

resource requests and the current jobs. Note that the latter sections are only displayed if jobs are active.

```
$ rinfo
MACHINE        CONFIGURATION
atlas          day

PARTITION      CPUS    STATUS        TIME   TIMELIMIT  NODES
root              6                                   atlas[0-2]
parallel        2/4   running    01:02:29             atlas[0-1]

RESOURCE       CPUS    STATUS        TIME    USERNAME  NODES
parallel.996      2 allocated       00:05        user  atlas0

JOB            CPUS    STATUS        TIME    USERNAME  NODES
parallel.1115     2   running       00:04        user  atlas0
```

The machine section gives the name of the machine and the active configuration.

For each partition in the active configuration, `rinfo` shows the number of CPUs in use, the total number of CPUs, the partition status, the time since the partition was started, any CPU time limits imposed on jobs, and the node names. This information is extracted from the `partitions` table (see Section 11.5.17). The description of the `root` partition shows the resources of the whole machine.

The resource section identifies the resource allocated to the user, the number of CPUs that the resource includes, the user name, the node names and the status of the resource. The `time` field specifies how long the resource has been held in hours, minutes and seconds.

The job section gives the job identifier, the user name, the number of CPUs the job is using, on which nodes and the status of the job. The `time` field specifies how long the job has been running in hours, minutes and seconds.

The `-u` option restricts the output to that relating to a specific user. The `-r` and `-j` options restrict the information output to that relating to specified resources and jobs but generate additional detail including the process ID of the `prun` process that started the job, resource usage, the number of processes running and their distribution, process IDs and hostnames.

The fields displayed for jobs and resources and the ordering of these fields can be specified with the `-o` option (see Table 11.21 and Table 11.29 for the field names).

EXAMPLES

When used with the `-cl` flags, `rinfo` prints information on the available configurations and shows current status.

```
$ rinfo -cl
CONFIGURATION   PARTITION   STATUS    ALLC  SUSP  QUED  BLKD  NODES
night           parallel    -                                 atlas[0-15]
day             test        running      1     0     0     0  atlas[8-15]
day             parallel    running      0     0     0     0  atlas[0-7]
```

When used with the `-q` flag, `rinfo` prints information on the user's projects, CPU usage limits, memory limits and priorities.

```
$ rinfo -q
PARTITION       CLASS         NAME     CPUS   MEMLIMIT   PRIORITY
parallel      project      default      0/8        100          0
parallel      project      divisionA  16/64       none          1
```

In this example, the access controls allow any user to run jobs on up to 8 CPUs with a memory limit of 100MB. Jobs submitted for the `divisionA` project run at priority 1, have no memory limit and can use up to 64 CPUs. 16 of these 64 CPUs are in use.

When used with the `-s` option, `rinfo` prints information on the status of the RMS servers. Addition of the `-l` option generates banners in the output and a summary of the status of the per-node daemons.

```
$ rinfo -s all
tlogmgr              rmshost      running        239241
eventmgr             rmshost      running        239246
mmanager             rmshost      running        239260
swmgr                rmshost      running        239252
pmanager-parallel    rmshost      running        239175

$ rinfo -l -s rmsd
SERVER               HOSTNAME     STATUS         PID
rmsd                 atlas0       running        740600
rmsd                 atlas1       running        1054968
rmsd                 atlas2       running        1580438
rmsd                 atlas3       running        2143669
rmsd                 atlasms      running        239212

STATUS               NODES
running              atlas[0-3] atlasms
```

In the above example, the system is functioning correctly. In the following example, one of the nodes has crashed.

```
$ rinfo -l -s rmsd
SERVER               HOSTNAME     STATUS         PID
rmsd                 atlas0       running        740600
rmsd                 atlas1       running        1054968
rmsd                 atlas2       not responding
rmsd                 atlas3       running        2143669
rmsd                 atlasms      running        239212
```

The next example restricts the output to resources allocated to two users and requests more detailed output.

```
$ rinfo -rul dick harry
RESOURCE       CPUS  STATUS         TIME   USERNAME  NODES
parallel.41965 2     allocated  01:31:30   dick     atlas3
parallel.41964 2     allocated  01:32:08   harry    atlas2
parallel.41963 1     allocated  01:32:14   harry    atlas1
```

The following example uses the `-o` option to select which fields from the `resources` and `jobs` tables are displayed and their order.

```
$ rinfo -o username,name,status -u dick
USERNAME       NAME        STATUS
dick           41965       allocated

$ rinfo -j -o name,resource,status -u dick
NAME       RESOURCE        STATUS
40091      41965           running
```

The following examples show how `rinfo` can be used to list events. First, to show all events, listed in time order, with a banner:

```
$ rinfo -l -e all
Date     Time             Class       Name    Type Description
 ...
12/20/02 16:48:26          node      atlas0   status running
12/20/02 16:48:26          node      atlas2   status running
12/20/02 16:48:26          node      atlas3   status running
12/20/02 16:49:10     partition    parallel   status blocked
12/20/02 16:54:26          node      atlas1   status active
12/20/02 16:54:42          node      atlas1   status configured out
12/20/02 16:54:51     partition    parallel   status running
12/20/02 17:13:26     partition    parallel   status closing
12/20/02 17:13:26     partition    parallel   status down
12/20/02 17:15:25     partition    parallel   status starting
12/20/02 17:15:26     partition    parallel   status running
12/20/02 17:17:58 switch_module     QR1T00   control power/control problem
```

To restrict the output to node status events in the last hour use:

```
$ rinfo -e node:*:status -H 1
12/20/02 16:48:26          node      atlas0   status running
12/20/02 16:48:26          node      atlas2   status running
12/20/02 16:48:26          node      atlas3   status running
12/20/02 16:54:26          node      atlas1   status active
12/20/02 16:54:42          node      atlas1   status configured out
```

The following example uses the `-LM` option to show license management information.

```
$ rinfo -LM
RMS License Management Information

Feature        Tokens   Expires
RMS/partition  32       permanent

Feature        Tokens   In use since
Total          32
RMS/partition  16       Fri Dec 20 16:17:09 2002
```

The `-N` causes `rinfo` to poll the HP AlphaServer SC Interconnect network error service on each node and display a summary of the results. For example:

```
$ rinfo -N
rinfo: neterr service running on atlas[0-3]
```

When used with the `-jl` options `rinfo` will display information on the running jobs. The `-j` option with of one or more job ids will show more detailed information on specific jobs. If the `-l` option is used as well then `rinfo` will print the process ids on each node. For example:

```
$ rinfo -jl
JOB             CPUS    STATUS        TIME    USERNAME  NODES
parallel.31023     4   running       00:11      duncan  atlas[0-3]

$ rinfo -j 31023
JOB          STATUS       COMMAND                  HOSTNAMES
31023        unknown      sleep 1000               atlas[0-3]

RESOURCE     PARTITION    CPUS   PRI   USERNAME     PID          PROJECT
38558        parallel     4      50    duncan       864376       default

ETIME        ATIME        UTIME        STIME        PAGEFLTS     MEM
03:06        12:24        00:00        00:00        0            0

PROCESSES PER NODE
2 atlas[0-3]

$ rinfo -jl 31023
JOB          STATUS       COMMAND                  HOSTNAMES
31023        unknown      sleep 1000               atlas[0-3]

RESOURCE     PARTITION    CPUS   PRI   USERNAME     PID          PROJECT
38558        parallel     4      50    duncan       864376       default

ETIME        ATIME        UTIME        STIME        PAGEFLTS     MEM
03:36        14:24        00:00        00:00        0            0

PROCESSES
atlas0       865507   864461
```

```
atlas1      1429295  1428940
atlas2      1590783  1590735
atlas3      2550854  2550825
```

When used with the `-rl` options `rinfo` will display information on the resource requests. The `-r` option with of one or more resource ids will show more detailed information on specific requests. If the `-l` option is used as well then `rinfo` will print the process ids on each node. For example:

```
$ rinfo -rl
RESOURCE        CPUS    STATUS        TIME    USERNAME  NODES
parallel.38558     4 allocated       11:23      duncan  atlas[0-3]

$ rinfo -r 38558
RESOURCE     PARTITION    CPUS   PRI   USERNAME     PID          PROJECT
38558        parallel     4      50    duncan       864376       default

ETIME        ATIME        UTIME        STIME        PAGEFLTS     MEM
11:36        46:24        00:00        00:00        0            0

STATUS       HOSTNAMES
allocated    atlas[0-3]

JOBS
31023

PROCESSES PER NODE
2 atlas[0-3]

$ rinfo -rl 38558
RESOURCE     PARTITION    CPUS   PRI   USERNAME     PID          PROJECT
38558        parallel     4      50    duncan       864376       default

ETIME        ATIME        UTIME        STIME        PAGEFLTS     MEM
11:36        46:24        00:00        00:00        0            0

STATUS       HOSTNAMES
allocated    atlas[0-3]

JOBS
31023

HOSTNAME     JOB       PROCESSES
atlas0       31023       865507   864461
atlas1       31023      1429295  1428940
atlas2       31023      1590783  1590735
atlas3       31023      2550854  2550825
```

SEE ALSO

`allocate`, `prun`

NAME

rmsarchive – Archives a RMS database

SYNOPSIS

rmsarchive [-h] [-m *machine*]

OPTIONS

`-h`	Display the list of options.
`-m` *machine*	Specifies which machine's database to archive. By default, the database of the host machine is archived.

DESCRIPTION

Certain tables in the RMS database which hold operational or transactional records are constantly being updated and grow in size. Periodically, redundant records should be removed from these tables to ensure the efficient operation of the database. Any operational records required for subsequent analysis should be archived first. For example, the records in the `resources` table can be analyzed to monitor usage patterns. Transaction records, on the other hand, are of little interest once the transaction has completed, and can simply be deleted. `rmsarchive` automates the archive process.

Use of `rmsarchive` is restricted to members of the RMS admin group.

The criteria for archiving and deleting records are held in the `archive_tables` table in the database (see Table 11.3). Each record in this table has four fields:

`name`	The name of a table to archive.
`lifetime`	The maximum time in hours for the data to remain in the table.
`timefield`	The name of the field in the table `name` that records the data's lifetime.
`selectstr`	An SQL select string, based on fields in the table `name`, that determines which records to archive.

For more detailed information, see Chapter 11 (*The RMS Database*) and Section 11.5.3.

The records are archived as a sequence of SQL statements in a file called *machine_date*.sql, which is compressed using gzip(1) and stored in /var/rms/archive. /var/rms/archive will be created the first time that rmsarchive is run. The *date* format is: yyyy-mm-dd-HH:MM.

On completion, rmsarchive sends mail to the users listed by the users-to-mail attribute in the attributes table. The mail message takes the following form:

```
rmsarchive: Nov 10 01:05:01

rmsarchive: '/usr/opt/rms/bin/rmstbladm  -c -x 73 -v' started Nov 10
01:05:00
rmsarchive: finished Nov 10 01:05:01 total records 2918 file is
/var/rms/archive/atlas_2001-11-10-01:05.sql

rmsarchive: rmstbladm output was ...
rmstbladm: archiving to /var/rms/archive/atlas_2001-11-10-01:05.sql
rmstbladm: archiving table 'rms_atlas:rms:acctstats': 4 records dumped
rmstbladm: archiving table 'rms_atlas:rms:disk_stats': 0 records dumped
rmstbladm: archiving table 'rms_atlas:rms:events': 4 records dumped
rmstbladm: archiving table 'rms_atlas:rms:jobs': 4 records dumped
rmstbladm: archiving table 'rms_atlas:rms:link_errors': 2902 records dumped
rmstbladm: archiving table 'rms_atlas:rms:node_stats': 0 records dumped
rmstbladm: archiving table 'rms_atlas:rms:resources': 4 records dumped
rmstbladm: gzipping '/var/rms/archive/atlas_2001-11-10-01:05.sql'
```

On completion, rmsarchive adds a record to the backups table. For example:

```
name     ctime             handle filename                                    archived
-------------------------------------------------------------------------------------
archive  11/10/01 01:05:01 49     /var/rms/archive/atlas_2001-11-10-01:05.sql 2918
```

rmsarchive invokes rmstbladm to manipulate the database.

EXAMPLES

To archive the records in the database for a specified machine:

```
$ rmsarchive -m atlas
```

To run rmsarchive daily at 1:05 a.m. using cron(8), add the following line to the root crontab file on the RMS host machine, specifying the site-specific path for rmsarchive, shown here as /usr/bin.

```
5 1 * * * /usr/bin/rmsarchive
```

SEE ALSO

`rmsbackup`, `rmstbladm`

NAME

rmsbackup – Backs up a RMS database

SYNOPSIS

rmsbackup [-b | -c] [-h] [-m *machine*]

OPTIONS

-b	Back up the database without archiving.
-c	Archive the database without backing it up.
-h	Display the list of options.
-m *machine*	Specifies which machine's database to back up. By default, the database of the host machine is backed up.

DESCRIPTION

rmsbackup backs up the RMS database and stores it as a sequence of SQL statements that can be restored (using rmstbladm) to recreate the data (see Section 10.6 for details of restoring a database). The SQL statements are written to a file called *machine_date*.sql, which is compressed using gzip(1) and stored in /var/rms/backup. /var/rms/backup will be created the first time that rmsbackup is run. The *date* format is: yyyy-mm-dd-HH:MM.

Use of rmsbackup is restricted to members of the RMS admin group.

On completion, rmsbackup sends mail to the users listed by the users-to-mail attribute in the attributes table. The mail message takes the following form:

```
rmsbackup: Nov  7 01:05:04

rmsbackup: '/usr/opt/rms/bin/rmstbladm  -b -x 62 -v' started Nov  7 01:05:02
rmsbackup: finished Nov  7 01:05:04 total records 23548 file is
/var/rms/backup/atlas_2001-11-07-01:05.sql

rmsbackup: rmstbladm output was ...
rmstbladm: backing up to /var/rms/backup/atlas_2001-11-07-01:05.sql
rmstbladm: backing up table 'rms_atlas:rms:access_controls': 1 records dumped
rmstbladm: backing up table 'rms_atlas:rms:acctstats': 322 records dumped
```

```
rmstbladm: backing up table 'rms_atlas:rms:archive_tables': 7 records
dumped
rmstbladm: backing up table 'rms_atlas:rms:attributes': 29 records dumped
rmstbladm: backing up table 'rms_atlas:rms:backups': 57 records dumped
...
rmstbladm: backing up table 'rms_atlas:rms:users': 10 records dumped
rmstbladm: gzipping '/var/rms/backup/atlas_2001-11-07-01:05.sql'
```

On completion, `rmsbackup` adds a record to the `backups` table. For example:

```
name     ctime            handle filename                                    archived
-------------------------------------------------------------------------------------
backup   11/07/01 01:05:01 38    /var/rms/archive/atlas_2001-11-07-01:05.sql 23548
```

Before backing up a database, `rmsarchive` should be run to archive and delete redundant records. When run with no arguments, `rmsbackup` automatically runs `rmsarchive` before performing the backup. It uses `rmstbladm` to manipulate the database.

EXAMPLES

To archive and back up the records in the database for a specified machine:

```
$ rmsbackup -m atlas
```

To run `rmsbackup` daily at 1:10 a.m. using `cron(8)`, add the following line to the `root crontab` file on the RMS host machine, specifying the site-specific path for `rmsbackup`, shown here as `/usr/bin`.

```
10 1 * * * /usr/bin/rmsbackup
```

SEE ALSO

`rmsarchive`, `rmstbladm`

NAME

rmsbuild – Creates and populates an RMS database

SYNOPSIS

```
rmsbuild [-dhv] [-I list] [-m machine] [-n nodes | -N list]
         [-R rails] [-t type]
```

OPTIONS

`-d`	Create a demonstration database.
`-h`	Display the list of options.
`-I` *list*	Specifies the names of any interactive nodes.
`-m` *machine*	Specifies a name for the machine.
`-n` *nodes*	Specifies the number of nodes in the machine.
`-N` *list*	Specifies the nodes in the machine by name.
`-R` *rails*	Specifies the number of directly connected network rails (default 0).
`-t` *type*	Specifies the node type.
`-v`	Specifies verbose operation.

Nodes can be specified by number (`-n`) or by name (`-N`) but not both. Lists of node names should be quoted, for example, `'atlas[0-15]'`.

DESCRIPTION

`rmsbuild` creates a database for a machine of a given size, adding default entries to the `nodes` table. For detailed information on this table see Section 11.5.15.

`rmsbuild` is used during the initial installation of a machine. It should be run on the `rmshost` node. `rmsbuild` runs `rmstbladm` to create a new database or update an existing one (see Page 5-66).

Detailed information about each node (number of CPUs, amount of memory and so on) is added by `rmsd` as it starts on each node. Additional entries can be added later if further nodes are installed.

The machine name is specified with the `-m` option. Machines should be given a short name that does not end a digit. Node names are generated by appending a number to the machine name. If no machine name is given `rmsbuild` will derive it by stripping any trailing digits from the hostname of the local host. A warning message will be printed.

The `rmsbuild` command can be rerun to add nodes to an existing machine description or to check its consistancy. To recreate an RMS database from scratch (deleting any data held in it at present) use the `msqladmin` command to drop it first and then run `rmsbuild`. The data can then be restored from a backup file using `rmstbladm` (see Section 10.6 for details).

Database entries for the nodes are generated by the `-n` or `-N` options. Use `-n` with a number to generate entries for nodes `0` to `n-1`. Use `-N` to generate entries for a named list of nodes such as `atlas[4-8]`.

Some systems include a management server. Use the `-I` option to specify the management server name and create a description of the management server in the RMS database. To generate the management server name, append the letters `ms` to the machine name; for example, `atlasms`. `rmsbuild` will set the network id to the special value `RMS_NOELAN` (-100) for management server nodes, instructing other RMS processes not to look for Elan adapters on these nodes.

Some systems have more than one type of node (a mix of ES40s and ES45s or a mix of ES45s and DS20Ls for example). To create a machine description for such a machine use `rmsbuild` with the `-t` and the majority node type and then use `rcontrol` to change the type of the minority nodes.

The `-R` option allows you to specify the number of directly connected network rails (default 0), those with parallel ports connected to nodes. This option is for older HP AlphaServer SC Interconnect switches only, it is not needed with the newer switches which connect to the management Ethernet.

If the demonstration mode is selected with the `-d` option, `rmsbuild` constructs the entries for a demonstration database; that is to say, a database that does not necessarily correspond to the physical resources of the system. Attributes of the nodes that would normally be set by `rmsd` are set to representative values and a default partition is created. The `-d` option is primarily for testing purposes but can be useful when demonstrating RMS. When creating such a database, you should take care to give it a different name from that of your system.

EXAMPLES

To create a description of a 64-node system called `atlas` with one management server, use `rmsbuild` as follows:

```
# rmsbuild -m atlas -I 'atlasms' -N 'atlas[0-63]'
```

To create a machine description for a 128-node system called `demo`, use `rmsbuild` as follows:

```
# rmsbuild -d -m demo -n 128
```

To create a machine description for a system with four ES45 I/O servers and 32 DS20L compute nodes, use `rmsbuild` and `rcontrol` as follows:

```
# rmsbuild -m atlas -t DS20L -N 'atlas[0-35]'
# rcontrol set node 'atlas[0-3]' type ES45
```

SEE ALSO

, `msqladmin` `rmsbackup`, `rmstbladm`

NAME

rmsctl – Stops, starts or shows the status of the RMS system.

SYNOPSIS

rmsctl [-aehv] [*start* | *stop* | *restart* | *show*]

OPTIONS

-a	Show all servers, when used with the show command.
-e	Only show errors, when used with the show command.
-h	Display the list of options.
-v	Verbose operation

DESCRIPTION

The rmsctl script is used to start, stop or restart the RMS system on all nodes in a machine. It can also be used to show status information.

rmsctl starts and stops RMS by executing on each node the script /sbin/init.d/rms. Note that rsh must be enabled for root users in order for this to function correctly.

rmsctl start starts all of the partitions in the active configuration and sets their autostart fields in the servers table to 1. rmsctl stop stops all of the partitions and sets the autostart fields to 0 (see Section 11.5.21).

This contrasts with the behavior of the script /sbin/init.d/rms, run from the rmshost node, which preserves the current state of the active configuration over a stop/start cycle (see Section 9.3.1).

When used with the command show, rmsctl shows the current status of the system.

EXAMPLES

To stop the RMS system, use rmsctl as follows:

```
# rmsctl stop
RMS service stopped on atlas1
```

```
RMS service stopped on atlas0
RMS service stopped on atlas3
RMS service stopped on atlas2
RMS service stopped on atlasms
```

To start the RMS system, use `rmsctl` as follows:

```
# rmsctl start
RMS service started on atlas0
RMS service started on atlas1
RMS service started on atlasms
RMS service started on atlas2
RMS service started on atlas3
pmanager-parallel: cpus=16 (4 per node) maxfree=4096MB swap=5171MB no memory limits
pstartup.OSF1: general partition parallel starting
pstartup.OSF1: enabling login on partition parallel
Enabling login on node atlas1.quadrics.com
Enabling login on node atlas3.quadrics.com
Enabling login on node atlas0.quadrics.com
Enabling login on node atlas2.quadrics.com
```

To show the status of the RMS system, use `rmsctl` as follows:

```
# rmsctl show
SERVER               HOSTNAME     STATUS         PID
tlogmgr              rmshost      running        778
eventmgr             rmshost      running        780
mmanager             rmshost      running        789
swmgr                rmshost      running        799
pmanager-parallel    rmshost      running        33357

STATUS          NODES
running         atlas[0-3] atlasms

CPUS            NODES
4               atlas[0-3] atlasms

MEMORY          NODES
4096            atlas[0-3]
1024            atlasms

SWAP SPACE      NODES
5171            atlas0[0-3] atlasms

TMP SPACE       NODES
6032            atlas[0-3]
5703            atlasms
```

SEE ALSO

rcontrol

NAME

rmsexec – Runs a sequential program on a lightly loaded node

SYNOPSIS

rmsexec [-hv] [-p *partition*] [-s *stat*] [*hostname*] *program* [*args ...*]

OPTIONS

`-h`	Display the list of options.
`-v`	Specifies verbose operation.
`-p` *partition*	Specifies the target partition. The request will fail if load-balancing is not enabled on the partition (see Section 11.5.17).
`-s` *stat*	Specifies the statistic on which to base the load-balancing calculation (see below).

DESCRIPTION

The `rmsexec` program provides a mechanism for running sequential programs on lightly loaded nodes – nodes, for example, with free memory or low CPU usage. It locates a suitable node and then runs the *program* on it.

The user can select a node from a specific partition (of type `login` or `general`) with the `-p` option. Without the `-p` option, `rmsexec` uses the default load-balancing partition (specified with the `lbal-partition` attribute in the `attributes` table). In addition, the *hostname* of the node can be specified explicitly. The request will fail if this node is not available to the user. System administrators may select any node.

The `-s` option can be used to specify a statistic on which to base the loading calculation. Available statistics are:

`usercpu`	Percentage of CPU time spent in the user state.
`syscpu`	Percentage of CPU time spent in the system state - a measure of the I/O load on a node.
`idlecpu`	Percentage of CPU time spent in the idle state.

`freemem`	Free memory in megabytes.
`users`	Lowest number of users.

By default, `usercpu` is used as the statistic. Statistics can be used on their own, in which case a node is chosen that is lightly loaded according to this statistic, or you can specify a threshold using *`statistic < value`* or *`statistic > value`*

EXAMPLES

Some examples follow:

```
$ rmsexec -s usercpu myprog
$ rmsexec -s "usercpu < 50" myprog
$ rmsexec -s "freemem > 256" myprog
```

SEE ALSO

`prun`, `rinfo`

NAME

rmshost – Prints the name of the node running the RMS management daemons

SYNOPSIS

rmshost [-hl]

OPTIONS

-h	Display the list of options.
-l	Prints the fully qualified domain name.

DESCRIPTION

The rmshost command prints the name of the node that is running (or should run) the RMS management daemons. It is used by the RMS system.

NAME

rmspost – Posts an RMS event

SYNOPSIS

rmspost [-hp] [-n *events*] [*class:name:type:description*]

OPTIONS

`-n events`	Number of events to post.
`-p`	Notify the RMS GUI SC Viewer

DESCRIPTION

The `rmspost` command posts an RMS event. The event is described by its `class` (the class of object to which it refers), `name` (the name of the object to which the event refers), `type` (the type of event) and `description` (a description of the event) (see Chapter 8 (*Event Handling*) for details).

The RMS `eventmgr` will run any handlers that match the class, name and type of the event, passing them the description.

EXAMPLES

To post an event warning that a node is being shut down use `rmspost` as follows:

```
# rmspost node atlas4 status "Shutting down nodeatlas4"
```

SEE ALSO

`eventmgr`

NAME

rmsquery – Submits SQL queries to the RMS database

SYNOPSIS

rmsquery [-ehsuv] [-d *database*] [-m *machine*] [*SQLquery*]

OPTIONS

-d *database*	Select database by name.
-e	Exit quickly with an error if the RMS database server is not running.
-h	Display the list of options.
-m *machine*	Select database by machine name.
-s	Shows supported SQL syntax.
-u	Print dates as seconds since January 1st 1970. The default is to print dates as a string created with `localtime(3)`.
-v	Verbosely prints field names above each column of output.

DESCRIPTION

`rmsquery` is used to submit SQL queries to the RMS database. Users are restricted to using the `select` statement to extract information from the database. System administrators may also submit SQL statements that update the database: `create`, `delete`, `drop`, `insert` and `update`. Note that queries modifying the database are logged. When used without arguments, `rmsquery` operates interactively and a sequence of commands can be issued.

The source for `rmsquery` is available from the software download pages of the Quadrics website http://www.quadrics.com. When used interactively, this version of `rmsquery` supports GNU readline and history mechanisms. Type `history` to see recent commands, use `Ctrl/p` and `Ctrl/n` to step back and forward through them. The `tables` command lists the tables in the selected database. The command `fields` followed by the name of a table lists the fields in a table. The command `verbose` toggles printing of field names. To quit interactive mode, type `Ctrl/d` or `exit` or `quit`.

When used in interactive mode the following commands can be submitted in the place of a query:

`exit (or ^D)`	Exits and saves history file.
`fields` *`table`*	Prints the names of the fields in a table.
`help`	Prints the list of available commands.
`history`	Prints query history.
`syntax`	Prints a brief description of supported SQL syntax (the equivalent of the `-s` option).
`tables`	Prints the names of all of the tables in the database.
`verbose`	Prints field names above the columns (the equivalent of the `-v` option).

The sources for `rmsquery` are distributed under the terms of the GNU General Public License. See `http://www.gnu.org` for details and more information on GNU readline and history. The sources can be downloaded from the Quadrics website. Details of the SQL language can be found on the Quadrics support page `http://www.quadrics.com`.

EXAMPLES

An example follows of a `select` statement that results in a list of the names of all of the nodes in the machine. Note that the *`query`* must be quoted. This is because `rmsquery` expects a single argument.

```
$ rmsquery "select name from nodes"
atlas0
atlas1
atlas2
atlas3
```

In the following example, `rmsquery` is used to print information on all jobs run by a user:

```
$ rmsquery "select name,status,hostnames,cpus,startTime,endTime from \
                    resources where username='user'"
7     finished  atlas[0-3]   4      12/21/99 11:16:44  12/21/99 11:16:46
8     finished  atlas0       2      12/21/99 11:54:23  12/21/99 11:54:29
9     finished  atlas[0-3]   4      12/21/99 11:54:35  12/21/99 11:54:39
```

The `-v` option prints field names. In the following example, `rmsquery` is used to print resource usage statistics:

```
$ rmsquery -v "select * from acctstats"
name  uid   project  started            etime   atime   utime   stime  ...
-------------------------------------------------------------------------
7     1507  1        12/21/99 11:16:44    2.00    8.00    0.10    0.22  ...
8     1507  1        12/21/99 11:54:23    6.65   13.30   10.62    0.10  ...
9     1507  1        12/21/99 11:54:35    4.27   16.63   12.28    0.44  ...
```

When used without arguments, `rmsquery` operates interactively and a sequence of commands can be issued.

```
$ rmsquery -v
sql> select name, status from partitions
name       status
------------------
login      running
parallel   running
sql>
```

NAME

rmstbladm – Database administration

SYNOPSIS

```
rmstbladm [-BbcDdfhuv] [-a file] [-C component] [-m machine]
          [-r file] [-t table]
```

OPTIONS

`-a` *file*	Append data to the database from the named file of SQL statements.
`-B`	Dump the first five rows of each table to `stdout` as a sequence of SQL statements. A specific table can be dumped if the `-t` option is used. A specific component can be dumped if the `-C` option is used.
`-b`	Back up the database. The data is written as a sequence of SQL statements to a file named *machine_date*`.sql`, which is compressed using `gzip(1)` and stored in `/var/rms/backup`. A specific table can be backed up if the `-t` option is used. A specific component can be backed up if the `-C` option is used.
`-C` *component*	Specifies a single component for `rmstbladm` operations.
`-c`	Clean old records from the database according to the criteria specified in the `archive_tables` table (see Chapter 11 (*The RMS Database*)). The records are archived as a sequence of SQL statements in a file called *machine_date*`.sql`, which is compressed using `gzip(1)` and stored in `/var/rms/archive`.
`-D`	Dump the contents of the database to `stdout` as plain text. A specific table can be dumped if the `-t` option is used. A specific component can be dumped if the `-C` option is used.
`-d`	Dump the contents of the database to `stdout` as a sequence of SQL statements. A specific table can be dumped if the `-t` option is used. A specific component can be dumped if the `-C` option is used.
`-f`	Recreate the database from scratch. A specific table can be recreated if the `-t` option is used. A specific component can be recreated if the `-C` option is used.

`-h`	Displays the list of options.
`-m` *`machine`*	Specifies a machine for `rmstbladm` to administer.
`-r` *`file`*	Restore the database tables from the named file of SQL statements. If the file is compressed (see the `-b` option), it is uncompressed into `/tmp` before the restore operation takes place.
`-t` *`table`*	Specifies a single table to be worked on.
`-u`	By default, `rmstbladm` checks the consistency of the database. If the `-u` flag is specified, the database is updated to the current revision level. A specific table can be updated if the `-t` option is used. A specific component can be updated if the `-C` option is used.
`-v`	Specifies verbose operation.

DESCRIPTION

The command `rmstbladm` is used to administer the RMS database. Unless a specific machine is specified with the `-m` option, `rmstbladm` operates on the database of the host machine.

`rmstbladm` creates the database tables and their default entries according to a set of table definition files. The table definition files for RMS are held in `/usr/opt/rms/msql/rms`. The table definition files for other components are held in component directories filed under `/usr/opt/sra/msql`. By default, RMS parses all of the component directories. The `-C` option can be used to restrict this to a single component. The `-t` option can be used to restrict this to a single table (see Chapter 11 (*The RMS Database*) for further details).

When installing or upgrading a system, `rmstbladm` is used to check the consistency of the database, to change the structure of the tables and to add default entries. Once the system is installed and working correctly, the database should be backed up using `rmsbackup` (which, in turn, uses `rmstbladm`) so that it can be restored later should this prove necessary.

`rmstbladm` can be used to back up individual tables (or the whole database), to restore tables from file or to force the recreation of tables. Backups are written as text files containing the SQL statements required to recreate the data. The files are automatically compressed using `gzip(1)` and stored in `/var/rms/backup` as *`machine_date`*`.sql.gz`, where *`machine`* denotes the machine which holds the database and *`date`* gives the date and time of the backup in the format `yyyy-mm-dd-HH:MM`.

Before restoring the database from a backup using the `-r` option, all jobs should be killed using `rcontrol` and RMS should be halted using `rmsctl`. All updates to the database made since the backup was taken will be lost.

Certain tables in the RMS database (in particular, the `resources`, `jobs`, `events` and `acctstats` tables) grow over time and as each job is run. To remove old entries from the database and archive them, use `rmsarchive` (which, in turn, uses `rmstbladm` with the `-c` option). This deletes and archives records according to the criteria specified in the `archive_tables` table (see Chapter 11 (*The RMS Database*)). The records are archived as a sequence of SQL statements in a file called *`machine_date`*`.sql`, which is compressed using `gzip(1)` and stored in `/var/rms/archive`.

Access to `rmstbladm` options that update the database is restricted to administrative users.

EXAMPLES

To archive old accounting records in the database of a selected machine:

```
$ rmstbladm -c -t acctstats -m atlas
```

To back up the database:

```
$ rmstbladm -b
```

To back up a selected table to a specified text file as a sequence of SQL statements:

```
$ rmstbladm -d -t nodes > backup_nodes.sql
```

To back up a selected component to a specified text file as a sequence of SQL statements:

```
$ rmstbladm -d -C sra > backup_sra.sql
```

To update the database on installing a new version of RMS:

```
$ rmstbladm -u
```

To restore the database from a backup file:

```
$ rmstbladm -r /var/rms/backup/atlas_2001-11-30-01:05
```

SEE ALSO

`rmsarchive`, `rmsbackup`

NAME

swctrl – Switch network management utility used by SC Viewer or as a command-line interface.

SYNOPSIS

swctrl *command* [*args ...*]

OPTIONS

`-h`	Display the list of commands.
`-m` *machine*	Specify the machine name (defaults to rmshost name).
`-v`	Verbosely prints switch module reports.

DESCRIPTION

The `swctrl` command provides a command line interface to managing the state of the HP AlphaServer SC Interconnect. The `swctrl` command queries/modifies module settings in the switch module server `swmserver`. For federated networks, `swmserver` resides on the embedded controllers (QM410s in a HP AlphaServer SC Interconnect system or QM503s in a QsNet^{II} system) located within the switch module chassis. Non-federated networks have a single `swmserver` per rail running on the nodes connected to the switch module control port. In both cases, `swmserver` provides switch management and environment reporting capabilities directly to the `swctrl` client.

When used without arguments, `swctrl` operates interactively and a sequence of commands can be issued. A single command can be executed from the command line

```
# swctrl command[args] ...
```

where *command arg* pairs take the following form:

`check [` *modules* `[=]` *name ...* `]`

Provides cross module consistency checking. Checks are performed for clock-source and client/server software versions. Additionally, environment checks are performed on module power supply units, fans, board temperatures and board population (checking for correct board combinations).

`check [tree [`*`id`*`]] [ rail `*`r`*`]] [ hours `*`h`*`]]`

Check for errors on tree *id* (see below for details of valid tree ids). Where no tree is specified `swctrl` defaults to the broadcast tree on each rail. The optional *rail* argument allows a single rail to be selected. The optional *hours* arg controls the time period over which errors are selected (the default is to select all current errors).

`clear [ `*`modules`*` [=] `*`name ...`*`  ]`

The `clear` command clears errors from the named modules, reseting the error counters to 0.

`configure { in | out } tree `*`t`*` [ rail `*`r`*` ]`

Configure a whole tree in or out. The tree is identified by its top switch *t* and rail *r*. If no rail is given, rail 0 is assumed. All links down from the top switch are held in reset when configured out.

Note that when a whole tree is configured out the adaptive routing hardware in the Elite will avoid the configured out links.

`configure { in | out } link `*`m:b:c:l`*

Configure an individual link in or out. The link is identified by m:b:c:l where m is the name of the module, b is the board slot, c is the chip id and l is the link number. The link will be held in reset when configured out.

`exit` Exit `swctrl`.

`get `*`param`*` [ `*`modules`*` [=] `*`name ...`*`  ]`

Get switch module parameters.

`help` `[`*`all | command`*`]`

Context sensitive help is available for each *command*.

`info [ `*`modules`*` [=] `*`name ...`*`  ]`

Reports environmental information such as module name, type, clock frequency, clock source, maximum board temperature, software version, `IP`, `MAC` and gateway address. Additionally, the boot protocol, upgrade filename and `tftp` server `IP` address are provided.

`quit`

`reboot [`*`upgrade`*`] [ `*`modules`*` [=] `*`name ...`*`  ]`

Reboots the embedded controller. The absence of the *upgrade* argument will cause the module to reboot only. If no *modules* argument is supplied, the command will apply to all communicable modules within the machine.

`scan internal|external [ `*`modules`*` [=] `*`name ...`*`  ]`

Perform a boundary scan on links within a switch module or links between switch modules. Internal boundary scans test link connectivity within a single module. External boundary scans test inter-module connectivity and therefore must have two modules specified in it's argument list.

`set attr val [ `*`option`*`  [=] `*`value ...`*`  ]`

The following `attr val` pairs apply:

`clock-source [ A | B | local ]`

The clock source must be consistent across modules within a federated network.

`clock-frequency `*`value`*

Clock frequencies must be between 380 and 420MHz.

`broadcast-tree id [rail `*`r`*`]`

Set broadcast tree.

`network-type `*`value`*`]`

Set network type.

`show [ `*`modules`*` [=] `*`name ...`*`  ]`

Show details of the switch network configuration. See `info`.

`verbose` Set verbose operation.

Supported network types and top switch id ranges are as follows:

Network Type	Tree Ids
QMS16	0 - 3
QMS128	0 - 15
QMF256	0 - 63

(continued on next page)

(continued from previous page)

Network Type	Tree Ids
QMF512	0 - 63
QMF1024	0 - 63
QMF1536	0 - 31
QMF3072	0 - 255

When selecting a broadcast tree or selecting a tree to configure out the tree id must be consistent with the network type.

EXAMPLES

In the following example, `swctrl` is used to report basic information from the embedded controllers.

```
$ swctrl info
Modules         : 0 (QR1T00)  1 (QR1N00)

QNX Module: QR1T00 Type: QMS16X8  Clock 399.9 src A  Temperature: 29
Software version: 3-02052799  ip: 192.168.84.200  mac: 00:40:53:06:00:3b
Gateway: 0.0.0.0  Protocol: bootp  Upgrade file: /upgrade.tar  tftp server: 192.168.84.1
QM401           : 0 1 2 3 4 5 6 7
QM408           : 61

QNX Module: QR1N00 Type: QMS32U96D  Clock 399.9 src A  Temperature: 44
Software version: 3-02052799  ip: 192.168.84.202  mac: 00:40:53:06:00:4a
Gateway: 0.0.0.0  Protocol: bootp  Upgrade file: /upgrade.tar  tftp server: 192.168.84.1
QM401           : 0 1 2 3 4 5
QM402           : 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
QM407           : 6 7
QM408           : 61
```

Module clock sources must be consistent within a federated network. For cases where multiple clock sources are available (and the module chassis has a clock card of type QM408), the clock source can be set as follows:

```
$ swctrl set clock-source B
Confirm new clock source B for modules: QR1T00 QR1N00
(yes/no):
```

Note

The absence of the [`modules` *`name`* `...`] argument causes the clock source assignment to be applied to all modules.

The next example illustrates how `swctrl` is used to upgrade selected module controllers (federated networks with QNX embedded controllers only). Prior to this, ensure that the module `tftp` server `IP` address is set and that the `upgrade.tar` filename in the `/tftpboot` directory matches that shown by `swctrl info [ modules` *`name`* `... ]`. Omission of the upgrade argument will simply cause the module controller to reboot.

```
$ swctrl reboot upgrade module QR1N00

Modules         : 0 (QR1N00)

swctrl: upgrading module QR1N00 - allow 2 minutes for completion.
```

6

Access Control, Usage Limits and Accounting

6.1 Introduction

RMS access controls and usage limits operate on a per-user or per-project basis (a project is a list of named users). Each partition may have its own controls. This mechanism allows system administrators to control the way in which the resources of a machine are allocated amongst the user community.

RMS accounts for resource usage by user and by project. As each request is allocated CPUs, an accounting record is created, containing the `uid` of the user, the project name, the resource identifier and information on resource usage (see Section 6.5). This record is updated periodically while the CPUs remain allocated.

6.2 Users and Projects

When a system is first installed, there is only one project, called the *default* project. All users are members of this project and anyone who has logged into the system can request all of the CPUs. This simple setup is intended for a single class of cooperating users.

To account for resource usage by user or by project, the administrator must create additional user and project records in the RMS database. To control the resource usage of individuals or groups of users, the administrator must, in addition, create access

control records.

When submitting requests for CPUs, users can select any project of which they are a member (by setting the `RMS_PROJECT` environment variable or by using the `-P` flag when executing `prun` or `allocate`). RMS rejects requests to use projects that do not exist or requests to use projects of which the user is not a member. Users without an RMS user record are subject to the constraints on the default project.

In general, each user is a member of several projects, while projects may have many users. Membership of a project is specified in the `users` table with the `projects` field (see Section 11.5.32). The value of `projects` may be either a single name or list of project names, separated by commas or space. The wildcard character, `*`, may be entered as a project name, denoting that the user is a member of all projects. The ordering of the names in the list is significant: the first project specified becomes the user's default project.

User and project records are created by the system administrator and stored in the `users` and `projects` tables (see Section 11.5.32 and Section 11.5.18).

6.3 Access Controls

Access control records specify the maximum resource usage of a user or project on a given partition. They are created by the system administrator using `rcontrol` or `rmsquery` and stored in the `access_controls` table (see Section 11.5.1).

Each entry specifies the following attributes:

name The name of the user or project.

class Whether the entry refers to a user or a project.

partition The partition to which the access control applies.

priority The default priority of requests submitted by this user or project. Priorities range from 0, the lowest priority, to 100. The default is 50 (the value of the `default-priority` attribute).

maxcpus The total number of CPUs that this user or project can have allocated at any time.

memlimit The maximum amount of memory in megabytes per CPU that can be allocated.

A suspended request does not count against a user's or project's maximum number of CPUs. However, when the request is resumed, a usage check is performed and the request is blocked if starting it would take the user or project over their usage limit.

The access controls for individual users must set lower limits than those of the projects of which they are a member. That is to say, they must have a lower priority, smaller number of CPUs, smaller memory limit and so on than the access control record for the project. Where a memory limit exists for a user or project, it takes precedence over any default limit set on the partition (see Section 11.5.17).

When the system is installed, there are no access control records. If there is no default access control record in the database when a Partition Manager starts, it creates one using information from the partition. The memory limit is set to that of the partition, the priority is 0 and the CPU usage limit is equal to the number of CPUs in the partition. If the partition has no memory limit then all jobs run with memory limits disabled until access control records are created.

6.3.1 Access Controls Example

To illustrate how the RMS access controls mechanism works, we consider an example in which a system is primarily intended for use by Jim, Mary and John, members of the project called design. When they are not using the system, anyone else can submit small jobs.

First, create a project record for design:

```
rcontrol create project = design description = "System Design Team"
```

name	id	description
design	1	System Design Team

Now create user records for Jim, Mary and John:

```
rcontrol create user = jim project = design
rcontrol create user = mary project = design
rcontrol create user = john project = design
```

name	projects
jim	design
mary	design
john	design

Now create access controls for the design project and for the default project (all other users):

```
rcontrol create access_control = design class = project partition = \
```

```
    parallel priority = 5
rcontrol create access_control = default class = project partition = \
    parallel priority = 0 memlimit = 256
```

name	class	partition	priority	maxcpus	memlimit
design	project	parallel	5	Null	Null
default	project	parallel	0	Null	256

Requests submitted by Jim, Mary and John run at priority 5, causing other users' jobs to be suspended if running. These requests are not subject to CPU or memory limits.

Requests submitted by other users run at priority 0 and are subject to a memory limit of 256MB per CPU. Note that on a system with 4 CPUs and 4GB of memory per node, it would be necessary for each node to have at least 5GB of swap space to ensure that jobs submitted by the design group were not blocked by other users (see Section 7.4.2 for details).

In this example, we have not set the `maxcpus` limit as we do not mind how many CPUs the users allocate. This limit could be set if there were two groups of users of equal priority and you wanted to bound the number of CPUs that each could allocate.

6.4 How Access Controls are Applied

The rules governing memory limits, priority values and CPU usage limits are described in more detail in the following sections.

6.4.1 Memory Limit Rules

Memory limits for a resource request are derived by applying the following rules in sequence until an access control record with a memory limit is found.

1. The root user has no memory limits.
2. If the user has an access control record for the partition, the memory limit in the access control record applies.
3. The access control record for the user's current project determines the memory limit.
4. The access control record for the default project determines the memory limit.

Having selected an access control record, the memory limit for the program is set by the value of its `memlimit` field. A null value disables memory limits. Other values are interpreted as the memory limit in megabytes for each CPU. A process with one CPU

allocated has its memory limits set to this value. A process with more than one CPU allocated has proportionately higher memory limits. Data segment size, maximum memory limit, stack size and virtual memory are set but the virtual memory limit has a minimum value of 300MB to allow for the default MPI system buffer and the Elan SDRAM.

The `RMS_MEMLIMIT` environment variable can be used as an override to reduce the memory limit set by the system but not to raise it.

By default, the memory limit is capped by the minimum value for any node in the partition of the smaller of these two amounts:

1. The amount of memory on the node.
2. The amount of swap space.

If lazy swap allocation is enabled (see Section 7.4.2), the memory limit is capped by the minimum value for any node in the partition of the amount of memory per node.

Unix Memory Limits

Where RMS memory limits are in force the processes in a parallel application will be subject to per-process Unix memory limits that reflect the RMS values. For example with no memory limits

```
$ prun -n1 ulimit -a
time(seconds)        unlimited
file(blocks)         unlimited
data(kbytes)         4194304
stack(kbytes)        4194304
memory(kbytes)       2273456
coredump(blocks)     unlimited
nofiles(descriptors) 4096
vmemory(kbytes)      4194304
```

and with an RMS memory limit of 400 megabytes.

```
$ prun -n1 ulimit -a
time(seconds)        unlimited
file(blocks)         unlimited
data(kbytes)         409600
stack(kbytes)        409600
memory(kbytes)       409600
coredump(blocks)     unlimited
nofiles(descriptors) 4096
vmemory(kbytes)      409600
```

Where no RMS memory limit is in force the `inherit-user-limits` attribute controls whether the memory limits are inherited from the node on which `prun` is executed

(value 1) or from the application node (value 0, the default). This attribute controls the setting of the data, stack, memory and virtual memory limits.

The set-vm-limit attribute controls whether the UNIX virtual memory limit is set. The default is to set this limit as described above. This behaviour can be disabled by (leaving the virtual memory limit at its default value for the application node) by creating the set-vm-limit attribute and setting its value to 0.

The combined effect of these settings is summarized below:

RMS	RMS attributes		UNIX memory limits	
memlimit	**inherit-user**	**set-vm-limit**	**data,stack,memory**	**vm**
Not set	0	0	Max for node	Max for node
Not set	0	1	Max for node	Max for node
Not set	1	0	Inherit from prun	Max for node
Not set	1	1	Inherit from prun	Inherit from prun
Set	0	0	RMS memlimit	Max for node
Set	0	1	RMS memlimit	RMS memlimit
Set	1	0	RMS memlimit	Max for node
Set	1	1	RMS memlimit	RMS memlimit

where the UNIX memory limit is set to a value which is either that set by RMS, that inherited from from prun or the maximum value for the application node.

6.4.2 Priority Rules

The priority of a resource request is derived by applying the following rules in sequence until an access control record with a priority is found.

1. The root user has priority over all other users.
2. If the user has an access control record for the partition then this record determines the priority.
3. The access control record for the user's current project determines the priority.
4. The access control record for the default project determines the priority.

Having selected an access control record, the priority of the resource request is set by the value of its priority field. A null value sets the priority to 50, the default. Higher priority jobs are scheduled first. The user can instruct rcontrol to lower the initial priority but not to raise it. An administrator can raise or lower priorities.

6.4.3 CPU Usage Limit Rules

RMS keeps track of the number of CPUs in use by each user and each project. A request to allocate additional CPUs is blocked if it would cause the usage limit for the user or the usage limit for the user's current project to be exceeded. The request remains blocked until the user or other users in the user's current project free enough CPUs to allow the request to be granted. The CPUs can be freed either because the resources are deallocated or because the user suspends the resource using `rcontrol`.

The CPU usage limit is derived by applying the following rules in sequence until an access control record with a CPU usage limit is found.

1. No CPU usage limits are set on jobs run by the root user.
2. If the user has an access control record for the partition, the CPU usage limit is determined by the `maxcpus` field in this record.
3. The access control record for the user's current project determines the CPU usage limit.
4. The access control record for the default project determines the CPU usage limit.

CPU usage limits can be set to a higher value than the actual number of CPUs available in the partition. This is useful if gang scheduling and time slicing are in operation on the partition. For example, if a partition has 16 CPUs and the usage limit for a given user is 32 then RMS will allow two 16 CPU jobs to run (see Section 7.4.4 for details).

6.5 Accounting

As each request is allocated CPUs, an entry is added to the accounting statistics (`acctstats`) table (see Section 11.5.2) specifying the following details about the job:

name Resource name (see Section 11.5.19).

uid Identifier of the user.

project Name of the user's current project.

started Time at which resources were allocated (UTC).

ctime Time at which accounting record was last updated (UTC).

etime Elapsed time (in seconds) since CPUs were allocated.

atime Time (in CPU seconds) for which CPUs have been allocated. Note that `atime` stops ticking while a request is suspended.

utime	Time (in seconds) for which processes were executing in user state.
stime	Time (in seconds) for which processes were executing in system state.
cpus	Maximum number of CPUs used.
mem	Maximum memory extent of the program in megabytes.
pagefIts	Number of page faults requiring I/O summed over processes.
memint	Memory integral for the program in megabyte hours.
running	Set to show that the CPUs are in use.

Accounting records are updated periodically until the CPUs are deallocated. The `running` flag is set to 0 at this point.

The `atime` statistic is summed over all CPUs allocated to the resource request. The `utime` and `stime` statistics are accumulated over all processes in all jobs running on the allocated CPUs.

Note

The `memint` statistics are not implemented in the current release. All values for this fields are 0.

7

RMS Scheduling

7.1 Introduction

The Partition Manager (see Section 4.4) is responsible for scheduling resource requests and enforcing usage limits. This chapter describes the RMS scheduling policies and explains how the Partition Manager responds to resource requests.

7.2 Scheduling Policies

The scheduling policy in use on a partition is controlled by the `type` attribute of the partition. The `type` attribute can take one of five values:

login Normal UNIX time-sharing applies. This scheduling policy is used for partitions that do not run parallel programs, such as interactive login partitions.

In addition, RMS supports load-balanced sequential processing, whereby users can request to have sequential programs executed on a lightly loaded node. Load balancing is enabled on `login` and `general` partitions. The default partition used for load balancing is specified by the attribute `lbal-partition` in the `attributes` table (see Table 11.5). `rmsexec` (see Page 5-59) can be used to run a program with load balancing.

parallel A *gang scheduling* policy is used. This is for partitions intended for production runs of parallel programs. With gang scheduling, the

scheduling decisions apply to all processes in a parallel program together. That is to say, all of the processes in a program are either running or suspended at the same time.

Gang scheduling is required for tightly-coupled parallel programs which communicate frequently. It becomes increasingly important as the rate of interprocess communication increases. For example, if a program is executing a *barrier* synchronization, all processes must be scheduled before the barrier completes.

Effective scheduling of parallel programs requires that user access through commands such as `rsh`, `rlogin` and `telnet` is disabled. This is carried out by the partition start-up script (see Section 4.4).

general The scheduling policy supports UNIX time-sharing with load balancing and gang scheduling. It is appropriate for a login partition that is used for developing and debugging parallel programs.

batch The scheduling policy is determined by a batch system. It is appropriate for partitions that are for the exclusive use of a batch system. The batch system may run sequential or parallel programs as it wishes but interactive use is prohibited.

restricted The scheduling policy is determined by an external scheduler. The external scheduler may run sequential or parallel programs as it wishes but interactive use is prohibited.

7.3 Scheduling Constraints

The scheduling decisions made while gang scheduling are controlled by a number of parameters. These parameters can be specified for individual users and for projects (groups of users) in the `access_controls` table (see Section 11.5.1). Restrictions on the partition itself are specified in the `partitions` table (see Section 11.5.17). The parameters are as follows:

Priority

Each resource request is assigned a priority taken from the `priority` field of the `access_controls` table. The Partition Manager schedules resource requests in order of priority. Where a number of requests are queued with the same priority, they are scheduled by order of submission time. The submission into the queue of a high priority request may cause existing low priority jobs to be suspended. Changing the priority of a request requires administrator privileges.

Maximum Number of CPUs

An upper limit can be set on the number of CPUs that may be allocated to a user or project at any point in time. Requests that take the usage count for the user or project above this limit are blocked. Requests for more CPUs than the limit on a user or project are rejected.

Time Limit

Jobs are normally run to completion or until they are preempted by a higher priority request. Each partition may have a time limit associated with it which restricts the amount of time the Partition Manager may allow for a parallel job. On expiry of this time limit, the job is sent a `SIGXCPU` signal. A period of grace is allowed following this signal for the job to clean up and exit. After this period, the job is killed and the resource deallocated. The duration of the grace period is specified in the `attributes` table (see Section 11.5.4) and can be set using `rcontrol`.

Memory Size

The Partition Manager can enforce memory limits that restrict the size of a job. The default memory limits are designed to prevent memory starvation (a node having free CPUs but no memory) and to control whether parallel jobs page or not. The system administrator can control the maximum amount of allocatable memory on each node through use of the `max-alloc` attributes.

7.4 What Happens When a Request is Received

A user's request for resources, made through the RMS commands `prun` or `allocate`, specifies the following parameters:

cpus The total number of CPUs to be allocated.

nodes The number of nodes across which the CPUs are to be allocated. This parameter is optional.

base node The identifier of the first node to be allocated. This parameter is optional.

hwbcast A contiguous range of nodes. This parameter is optional. When a contiguous range of nodes is allocated to a job, messages can be broadcast in hardware. This offers advantages of speed over a software implementation if the job makes use of broadcast operations.

rails The number of rails (layers in the switch network) required in a multirail system. A `railmask` can be specified to select the rails required.

cluster The cluster on which to allocate nodes. This parameter is optional.

cluster-member The cluster member IDs of nodes to be allocated. This parameter is optional.

memory The amount of memory required per CPU. This parameter is optional (set through the environment variable `RMS_MEMLIMIT`) but jobs with low memory requirements may be scheduled sooner if they make these requirements explicit.

time limit The length of time for which the CPUs are required. This parameter is optional (set through the environment variable `RMS_TIMELIMIT`).

immediate The request should fail rather than block if resources are not available immediately.

Note

The RMS scheduler attempts to allocate CPUs on a contiguous range of nodes. If a contiguous range of nodes is not available then requests that explicitly specify a contiguous range with the `hwbcast` parameter will block if the requested CPUs cannot be allocated. Requests that do not set the `hwbcast` parameter will be met using multiple node ranges to a limit of 6.

When the Partition Manager receives a request, it first checks to see if the partition has sufficient resources. If the resources are available, the next check is on the resource limits applied to the user and the project. If these checks fail, the request is rejected.

If the checks succeed, the scheduler attempts to allocate CPUs from those that are currently free. If sufficient CPUs are free but allocating them would exceed the user's CPU usage limit, the request is marked as `blocked` (or, if the immediate parameter is set, the request fails). If CPUs can be allocated, the resource request is marked as `allocated` and job(s) may use the CPUs. If the request cannot be met, it is added to the list of active requests and marked as `queued`. The scheduler then re-evaluates the allocation of CPUs to all of the requests in the list.

The list of resource requests is sorted in priority order. Requests of the same priority are sorted by submission time. When evaluating the list, the scheduler works down the requests trying to allocate CPUs to them. The highest priority request is allocated CPUs first, except when doing so would cause the system to run out of swap space (see Section 7.4.2).

In considering each request, the scheduler first looks at whether it has already been allocated CPUs (a *bound* request). CPUs remain allocated to a request unless they are preempted by a higher priority request, in which case the request of lower priority is suspended together with any jobs that were running on it. If the request is not yet bound then CPUs are allocated, if sufficient are free.

The list of requests is re-evaluated when the following events take place:

- A new request is made
- A request completes
- A request is suspended
- A request is resumed
- A request's priority changes
- The partition's time-slice interval expires (see Section 7.4.4)

The scheduler logs the decisions it takes in the `scheduler_log` table (see Table 11.30).

Consider what happens when a high priority request is submitted to a partition that is already running jobs. If there are sufficient CPUs free (matching the constraints of the request) then the job(s) start. If there are not enough free CPUs, the list of requests is re-evaluated. CPUs are allocated to the high priority request and its job(s) are allowed to start. The jobs of the lower priority requests, whose CPUs were taken for the high priority request, are suspended. Any of the low priority jobs for which CPUs are available continue.

7.4.1 Memory Limits

If memory limits are enabled (by setting the `memlimit` attribute of a partition or access control) then a request is only allocated CPUs on nodes that have sufficient memory available. RMS enforces memory limits by setting the virtual memory, data and stack size limits on a process. If the process exceeds the allowed size, it is killed (and the parallel program terminated).

Users, whose programs consume a known amount of memory each time they run, can set their own memory limit with the environment variable `RMS_MEMLIMIT`. Setting this variable (especially if the new value is much smaller than their default memory limit) may cause their jobs to be scheduled sooner than would otherwise be the case. Users cannot raise their memory limits above the level set by the system administrator. They can only lower them.

The default memory limit is calculated by dividing the memory available by the number of CPUs per node. For example, if a node has 4GB of allocatable memory and 4 CPUs

then each CPU that is allocated comes with 1GB of memory. Larger memory limits can be set but this risks having CPUs idle through memory starvation, unless there is a plentiful supply of jobs requesting small amounts of memory.

If memory limits are enabled, the RMS scheduler keeps track of the maximum memory usage per node. The ratio of memory limit to memory size determines how many requests can be present (allocated or suspended) before jobs start to page.

7.4.2 Swap Space

By default, the operating system reserves swap space as a program allocates memory. Hence, a process requiring 1GB of memory must also have 1GB of swap space. If memory limits are enabled, RMS does not allocate CPUs to new requests if the addition of their maximum memory usage to that already allocated would cause the total for the node to exceed the maximum allocatable memory for the node – controlled *via* the `max-alloc` attributes (see Section 11.5.4). The system administrator must ensure that all nodes have at least this amount of swap space.

Each node normally has significantly more swap space than memory. The ratio of memory limit to swap space determines how many requests (allocated or suspended) can be present on each node.

Tru64 UNIX supports a `lazy` swap allocation policy in which swap space is only allocated when required. This enables large memory jobs to run on nodes with relatively little swap space.

Warning

If `lazy` swap allocation is enabled, there must be sufficient swap space for the UNIX daemons and any other processes running on such nodes. Without this, processes (including those belonging to the system) will be killed if the system runs out of swap space.

7.4.3 Draining Jobs

The RMS scheduler can be instructed to drain jobs from a node (see Section 10.10). Where this process has started the CPUs will no longer be available for running parallel programs. The node will remain configured in until jobs that are using the node have completed or have been killed. At that point the node will be configured out. The `rinfo` command shows nodes in this state.

7.4.4 Time Slicing

Time slicing is enabled on a partition by setting its `timeslice` attribute; values of 15–120 seconds are recommended. If a `timeslice` is set, the Partition Manager evaluates the list of requests periodically. The list of requests is still sorted by priority but requests of the same priority are sorted on the number of time slices since they were last scheduled (rather than the submission time). When the system has requests for more CPUs than are available, the scheduler suspends requests at the end of each time slice so that others can use the CPUs.

When setting up a system for time slicing, it is important to set memory limits that ensure that all jobs remain resident in memory. System performance will be poor if time slicing between large jobs causes paging. The ratio of memory limit to memory size controls how many requests can progress concurrently. For example, on a node with 4 CPUs and 4GB of memory, setting a memory limit of 512MB will allow two jobs to be time sliced without paging.

The scheduler processes resource requests as it receives them. It tries to fit new requests to free CPUs. If no CPUs are available the request blocks, at least until the next time slice.

7.4.5 Suspend and Resume

The allocation of CPUs to a request can be suspended using `rcontrol`. Doing this reduces the CPU usage counts for the user and project, enabling other jobs to start. Either the user or administrator can resume the allocation at a later time. To resume the jobs, the CPUs are reallocated unless doing so would exceed a CPU usage limit. In this case, the request is marked as `blocked` and CPUs will only be allocated and the jobs restarted when sufficient CPUs become available. Note that requests that are suspended by the administrator cannot be resumed by their owner.

7.4.6 Idle Time

The amount of time that the resources allocated to a request can remain idle can be constrained by using `rcontrol` to set an idle timer (see also Section 11.5.4). By default, no timer is set. If an idle timer is set, it starts timing as soon as the resource has been allocated. It is stopped if the request is suspended and restarted when the request is resumed. If the idle timeout expires, the CPUs are deallocated.

7.4.7 Complex Resource Requests

A resource request made using the `prun` command's `procfile` mechanism can allocate CPUs on specific hosts (see Page 5-12). It is not restricted to a simple range of nodes.

Such requests can also be made through the RMS API. Once a resource request has been created with `rms_createResource()`, multiple calls to `rms_allocateCpus()` can be used to allocate arbitrary sets of CPUs (see Appendix D (*RMS Application Interface*) for more details).

8

Event Handling

8.1 Introduction

RMS includes a general mechanism for posting, waiting on and handling events. This functionality is provided by the Event Manager, `eventmgr` (see Section 4.7), which runs on the `rmshost` node.

Events are specified by RMS class, name, type and description:

class The class of object generating the event

name The instance of the object

type The event type

description A text description of the event

Generally, the description is either a single word that an event handler script can act on or a full description of some problem.

Events have a string representation as follows:

```
class:name:type:description
```

For example, the following event signifies that the new status of the node `atlas0` is `running`.

```
node:atlas0:status:running
```

The Event Manager writes events to the `events` table (see Table 11.16) in the RMS database. The following query prints the contents of the `events` table in time order:

```
$ rmsquery -v "select * from events order by ctime"
id  name    class  type    ctime             handled description
-------------------------------------------------------------
20  atlas0  node   status  05/04/01 15:53:02  1       running
21  atlas0  node   status  05/05/01 11:27:29  1       not responding
```

8.1.1 Posting Events

Events are normally posted by RMS servers but they can also be generated by the command line utility `rmspost`. This is useful for testing the response of the system to rare events. It can be run with a single argument as follows:

```
rmspost "class:name:type:description"
```

Alternatively, it can be run with 4 arguments as follows:

```
rmspost class name type "long description"
```

Note that the multiple word description, given as the fourth argument, must be quoted.

8.1.2 Waiting on Events

The command line utility `rmswait` waits on events. It can be run with a single argument as follows:

```
rmswait "class:name:type"
rmswait ":name:"
```

Alternatively, flags can be used to specify the class, name and type. The following example specifies the class with the `-c` option and the name with the `-n` option. The `-t` flag is used to specify the type of the event.

```
$ rmswait -c node -n atlas0
```

`rmswait` completes when a matching event is posted, after printing the event details on `stdout`.

Two events match if their class, name and type are same. They also match if one or more of the class, name and type is null. For example:

`node:atlas0:status` matches `node:atlas0:status`

`node::status` matches `node:atlas0:status`

`node::` matches `node:atlas0:status`

`:::` matches `node:atlas0:status`

Note that the class, name, type and description must all be specified when posting events but one or more of the class, name and type can be null when waiting on events.

8.2 Event Handling

Event handler scripts are specified in the `event_handlers` table. The default handlers installed by RMS are as follows:

```
rmsquery -v "select * from event_handlers"
name  class      type        timeout  handler
-------------------------------------------------------------------------------
      node       status      600      /usr/opt/rms/etc/rmsevent_node
                 temphigh    300      /usr/opt/rms/etc/rmsevent_env
                 tempwarn    300      /usr/opt/rms/etc/rmsevent_env
                 fan         300      /usr/opt/rms/etc/rmsevent_env
                 psu         300      /usr/opt/rms/etc/rmsevent_env
      event      escalation   -1      /usr/opt/rms/etc/rmsevent_escalate
```

The script `rmsevent_node` is run for all node status events. `rmsevent_env` is run for all environment events (temperature warnings, fan failures and PSU failures).

A timeout can be associated with each event handler. If the timeout is exceeded, the handler is killed and an *event escalation* event posted. `rmsevent_escalate` is run when one of the other handlers does not complete in the time allowed.

The event handler scripts can send mail to users warning them of events. To enable this, use `rcontrol` to set the `users-to-mail` attribute in the machine `attributes` table (see Table 11.5) as shown in the following example:

```
$ rcontrol create attribute=users-to-mail val='tom@company.com dick admin'
```

By default, event handler scripts are filed in `/usr/opt/rms/etc`. Local scripts should be filed in `/usr/local/rms/etc` as the contents of the `/usr/opt/rms` directory may change when a new release is installed.

The default event handlers use the `event-handler-path` attribute, specified in the `attributes` table (see Table 11.5), to locate and execute OS-specific and site-specific handler scripts in sequence before performing the default action.

The event handler path is a colon-separated list as shown in the following example:

`"/usr/local/rms/etc:/usr/local/lsf/etc"`

To set the `event-handler-path` attribute, use `rcontrol` as shown in the following example:

```
$ rcontrol create attribute=event-handler-path \
   val='/usr/local/rms/etc:/usr/local/lsf/etc'
```

Event handlers are executed in the order in which the `eventmgr` receives events. For example, if the status of a node changes from `running` to `active` (indicating that RMS has stopped running or the node has stopped responding) and then changes back to `running`, the event handler script `rmsevent_node` will be executed twice: first with a status of `active` and then with a status of `running`.

Event handler scripts are called with five arguments: the event identifier, the class, name and type of the event and the event description. A template `rmsevent_template` for writing event handler scripts is included in the release in `/usr/opt/rms/etc` and reproduced below. It illustrates how event handlers are called.

```
#!/bin/sh
#
# template default handler
#
# args:   id class name type description ...
#
program=‘basename $0‘
id=$1
class=$2
name=$3
type=$4
shift 4
description=$@

echo "handle event $id class $class name $name type $type: $description"

exit 0
```

8.3 List of Events Generated

The following events are generated by RMS:

`domain:`*`name`*`:status:`*`description`*

This event is generated by the Machine Manager when it detects problems with a TruCluster domain. The *`description`* contains one of the following:

Description	Meaning
`not responding`	The Machine Manager has not received recent updates from any node in this domain. The `inactivity-timeout` attribute defines the maximum period allowed between updates.
`inconsistent nodesets`	The Machine Manager has received inconsistent reports of the state of this domain

`elan:`*`name`*`:status:`*`description`*

This event is generated by the `rmsd` when a node drops out of the Elan nodeset but does not panic. This happens if its link cable is disconnected. The *`description`* contains `disconnected`.

`event:`*`name`*`:escalation:"did not complete"`

This event is generated by RMS when an event handler did not complete. The *`name`* contains the ID of the event being escalated. If a handler is registered, it is called with a description of the event. The handler should pass the event to an external management agent.

`node:`*`name`*`:status:`*`description`*

This event is generated by RMS when the status of a node changes. The *`name`* field contains the name of the node. The *`description`* contains one of the following:

Description	Meaning
`unknown`	node has been configured in
`active`	node is booted but RMS is not running
`running`	RMS is running
`not responding`	node does not respond to `ping`
`configured out`	node has been configured out

`node:`*`name`*`:temperature:`*`description`*

This event is generated by RMS when the temperature of a node changes by more than 2 degrees Centigrade. The *`name`* field contains the name of the node. The *`description`* shows the ambient temperature as follows:

`ambient=30`

`node:`*`name`*`:temphigh:`*`description`*

This event is generated by RMS when the temperature of a node (measured in degrees Centigrade) exceeds a certain threshold. The *`name`* field contains

the name of the node. The *description* shows the ambient temperature as follows:

```
ambient=45 threshold exceeded
```

`node:`*name*`:fan:"fan failure"`

This event is generated by RMS when a fan fails in a node. The *name* field contains the name of the node.

In the event of multiple failures, the reports are concatenated.

`node:`*name*`:psu:"psu failure"`

This event is generated by RMS when a PSU fails in a node. The *name* field contains the name of the node.

In the event of multiple failures, the reports are concatenated.

`partition:`*name*`:status:`*description*

This event is generated by RMS when the status of a partition changes. The *name* field contains the name of the partition. The *description* contains one of the following:

Description	Meaning
`starting`	partition is starting
`running`	partition is running
`blocked`	partition is blocked
`closing`	partition is closing down
`down`	partition has been shut down

`server:`*name*`:software:`*description*

This event is generated by RMS when a server fails to start. The *name* field contains the name of the server (for example, `rmsd`). The *description* takes the following form:

`Failed to start` *server*`:  `*message*

server gives the name of the server that has failed. *message* is forwarded from the failed server.

`server:`*name*`:elan:`*description*

These events are generated by RMS when the `rmsd` detects a problem with HP AlphaServer SC Interconnect. The *name* field indicates the name of the node. The *description* contains one of the following

Description	Meaning
`invalid node id`	A node has multiple Elan adapters at different network positions
`invalid node level`	A node has multiple Elan adapters connected to rails of different sizes
`cannot determine position`	The Elan device driver cannot determine the position of a node.
`elan3_neterr_svc failed`	The RMS daemon failed to initialise the network error fixup service

`switch_module:`*`name`*`:control:"power/control problem"`

This event is generated by RMS when the switch network monitoring system cannot access a switch module either because the switch module is powered down or because the control cable is disconnected. The *`name`* field contains the name of the switch module.

`switch_module:`*`name`*`:control:"power/control ok"`

This event is generated by RMS when the switch network monitoring system gains access to a switch module either when the switch module is powered up or when the control cable is connected. The *`name`* field contains the name of the switch module.

`switch_module:`*`name`*`:board:`*`description`*

This event is generated by RMS when a switch board is inserted into or removed from a switch module. The *`name`* field contains the name of the switch module. The *`description`* contains one of the following, where *`n`* gives the slot number:

`board removed from slot` *`n`*
`board inserted in slot` *`n`*

`switch_module:`*`name`*`:psu:`*`description`*

This event is generated by RMS when a PSU in a switch module changes state. The *`name`* field contains the name of the switch module. The *`description`* shows the state of the PSUs, listing them in numerical order, as shown in the following examples:

Description	Meaning
`PSU status (off, on)`	psu 1 has failed, psu 2 is ok
`PSU status (on, off)`	psu 1 is ok, psu 2 has failed

(continued on next page)

(continued from previous page)

Description	Meaning
`PSU status (on, on)`	psu 1 is ok, psu 2 is ok

`switch_module:`*`name`*`:fan:`*`description`*

This event is generated by RMS when a fan in a switch module fails. The *`name`* field contains the name of the switch module. The *`description`* shows the fan speeds, listing them in numerical order, as shown in the following:

```
Fan speeds (3325 3350 2825 3325 3350 3325)
Fan speeds (3325 **** 3350 3325 3350 3325)
```

Normal fan speed readings are in the range `3000` to `3500` rpm. Values outside this range should be investigated. The value `****` indicates that reading the fan speed failed. The fan should be replaced.

`switch_module:`*`name`*`:temperature:`*`description`*

This event is generated by RMS when the temperature of a switch module changes. The *`name`* field contains the name of the switch module. The *`description`* shows the temperature as follows:

```
Temperature ( 30C )
```

`switch_module:`*`name`*`:temphigh:`*`description`*

This event is generated by RMS when the temperature of a switch module exceeds the warning temperature threshold (40 degrees celsius) or the critical temperature threshold (50 degrees celsius). The *`name`* field contains the name of the switch module. The *`description`* shows the problem as follows:

```
Temperature (40C) exceeds warning threshold
Temperature (52C) exceeds critical threshold
```

`switch_module:`*`name`*`:clock:`*`description`*

This event is generated by RMS if the `swmgr` detects a problem with the network clock in one or more of the switch modules. The *`name`* field contains the name of the switch module. The *`description`* field shows the problem as follows:

```
Clock failure (126.5)
Inconsistant clock source
```

`switch_module:`*`name`*`:firmware:`*`description`*

This event is generated by RMS if the `swmgr` detects a problem with the firmware on one of the switch modules. The *`name`* field contains the name of the switch module. The *`description`* field shows the problem as follows:

```
Inconsistant firmware revision
```

`switch_module:`*`name`*`:error:`*`description`*

This event is generated by RMS if the `swmgr` detects a new link error in one of the switch modules. The *`name`* field contains the name of the switch module. The *`description`* field shows the problem as follows:

```
Link in reset
Link error
```

A full description of the error is written to the `link_errors` table (see Section 11.5.13). The handler for this type of event should be used to post a message to the maintainers of the system. The attribute `linkerror-event-interval` controls the rate at which these events are generated.

`switch_module:`*`name`*`:bcast-error:`*`description`*

This event is generated by RMS if the `swmgr` detects an error on the broadcast tree of any HP AlphaServer SC Interconnect rail. The *`description`* shows the position of the error.

`transaction:`*`name`*`:status:`*`description`*

This event is generated by RMS when a transaction is submitted and when a transaction is complete. The *`name`* field contains the handle of the transaction. This a unique identifier for the transaction (see Table 11.44). The *`description`* takes the following form:

`Transaction` *`handle`* `failed`

The `logfile` field of the `transactions` table holds the name of the log file that records the reasons for the failure.

8.4 Extending the RMS Event Handling Mechanism

The RMS event handling mechanism is open and extensible. New event types and handlers for them can be added on a site-by-site basis. For example, you might run a periodic file system status check on each node, execute a local clean-up script and post a file system event to RMS if the free space in the file system dropped below a prescribed level. The handler script could perform partition-wide or machine-wide cleanup and post a notification of the problem using email or a SNMP message.

9

Setting up RMS

9.1 Introduction

This chapter describes how to set up and customize RMS operations. The information is organized as follows:

- Planning the installation (see Section 9.2).
- Starting RMS and configuring the system (see Section 9.3).
- Customizing RMS (see Section 9.5).
- The RMS log files (see Section 10.11).

9.2 Installation Planning

Before you install RMS, think about how the resources of the system will be used and who is going to use them. Ask yourself the following questions:

- Will the system be open to anyone to use or is it for a specific group of users?
- Will the machine run a constant workload or do you expect cyclical patterns in usage, for example, a prime shift versus evenings and weekends?
- Is the machine primarily for running parallel jobs or do you expect a significant workload from sequential jobs?

- Will some of your users have jobs that consume all of the resources of the system for extended periods of time? If so, are you happy for other users to wait until the machine is available or do they need access to resources of their own?
- How do you wish to process the accounting data?

The answers to these questions should help you to determine how to configure the system. It may be that you cannot answer these questions, in which case you should start with one of the basic configurations described below.

9.2.1 Node Names

While planning the machine installation, give some thought to its name and the names of its nodes. We recommend selection of a short name for the machine (for example, `atlas`). Node names should use the name of the machine as a base and their network port number as a suffix (for example, `atlas0`, `atlas1`, `...`, `atlas63`). Names in this form can be referenced easily in RMS as a list (for example, `atlas[0-63]`).

Some machines have a management server node that is connected to the management network but is not connected to the HP AlphaServer SC Interconnect. By convention, this node is given the suffix `ms` (for example, `atlasms`).

9.3 Setting up RMS

RMS should be installed according to the instructions in the *HP AlphaServer SC Installation Guide*.

For the purposes of this section, we assume a machine with 64 nodes, where each node has 4 CPUs, 4GB of memory and an 18GB disk. You should make adjustments for the actual number of nodes in your system. If RMS is already running on the machine, skip to Section 9.3.2.

9.3.1 Starting RMS

The RMS initialization script, `/sbin/init.d/rms`, is run on each node with the argument `start` as the node boots. Conversely, when the node halts, the script is run with the `stop` argument.

To start or stop RMS manually on all of the nodes at once, run `rmsctl` on the `rmshost` node with the appropriate argument (`start` or `stop`). This command runs `/sbin/init.d/rms` on each of the nodes in turn. `rsh` must be enabled for `root` users for this command to work correctly. It is normally enabled as part of the installation.

```
# rmsctl start
```

Configure all of the nodes into the machine using `rcontrol`.

```
# rcontrol configure in 'atlas[0-63]'
```

Use `rinfo` with the `-n` option to check the status of the nodes. The output should show that all of the nodes are `running`.

```
# rinfo -n
running atlas[0-63], atlasms
```

If any of the nodes show a status other than `running`, restart them by running `/sbin/init.d/rms` on the nodes in question. For example, to restart RMS on `atlas3`, enter the following:

```
# /sbin/init.d/rms stop
# /sbin/init.d/rms start
```

or alternatively:

```
# /sbin/init.d/rms restart
```

If necessary, configure out any nodes that fail:

```
# rcontrol configure out atlas3
```

Restarting RMS

RMS daemons such as the Machine Manager and the Partition Manager can be stopped and started by executing the `/sbin/init.d/rms` script on the `rmshost` node. When run on the `rmshost` node, the script checks the status of each of the partitions in the active configuration. If a partition is in the `running` state or `blocked` state, the partition is stopped and its `autostart` field in the `servers` table is set to 1, otherwise the field is set to 0. When the node boots, only those partitions that have their `autostart` field set to 1, are restarted. This means that the state of the configuration is preserved.

By contrast, if `rmsctl` is used to start and stop the machine, all of the partitions in the active configuration are started: when `rmsctl` stops RMS, it sets all of the `autostart` fields to 0; when it starts RMS, it sets them to 1.

9.3.2 Initial Setup with One Partition

This example describes the simplest possible setup. All nodes are in a single partition and there are no memory limits, time limits or access controls. Any user can run a job using all of the CPUs.

Once RMS is running on all of the nodes, you set up a single partition as follows:

```
# rcontrol create partition=parallel configuration=day nodes='atlas[0-63]'
# rcontrol start partition=parallel
```

You should now be able to run a parallel program across all 64 nodes, for example:

```
# prun -N64 hostname
...
# prun -N64 dping 0 32
...
```

9.4 Inter-Process Communication in RMS

RMS processes communicate with each other using TCP/IP sockets. By default, a server will `bind` and a client will `connect` to the primary hostname of a node. By default, this will map to the management network IP address for the node. RMS can perform its inter-process communications either over the management network or the HP AlphaServer SC Interconnect (using *IP over Elan*). To enable IP over Elan, set the `rms-hostname-suffix` attribute to `-eip0` in the `attributes` table (see Table 11.5). RMS will then append `-eip0` to the hostnames when binding and connecting IP sockets. For example, RMS will use `atlas0-eip0` rather than `atlas0`. For RMS to operate with IP over Elan, all nodes must be connected to the HP AlphaServer SC Interconnect. `rmshost` is used as an alias in either mode of operation: it should be set to a management network address or an Elan IP address as appropriate.

9.5 Local Customization of RMS

RMS can be customized to suit local operating patterns in a variety of ways. Customization is done through site-specific scripts in `/usr/local/rms/etc`. Note that this is the default value of the `local-rmsdir` attribute, change this attribute to use another directory. The following site-specific customizations are supported:

- Partition startup
- Prologue and Epilogue scripts
- Core file analysis
- Event handling
- Switch manager configuration
- Controlling allocatable memory
- Rate of RMS status updates

If site-specific scripts exist then they override the defaults supplied with RMS. You might start by creating /usr/local/rms/etc, copying some of the default RMS scripts and modifying them to suit.

9.5.1 Partition Startup

The default partition startup script, /usr/opt/rms/etc/pstartup enables or disables logging in to a node according to the partition type. Site-specific variants might check whether users are logged in to the node and warn them of changes. They might also check on the availability of space in local temporary file systems.

To create a site-specific partition startup script, copy the default script /usr/opt/rms/etc/pstartup to /usr/local/rms/etc and modify it as required. If you wish to use a different directory for the site specific script use rcontrol to set the local-rmsdir attribute to the desired path.

9.5.2 Prologue and Epilogue Scripts

In the process of allocating CPUs to a user RMS can optionally execute a prologue script. In the process of deallocating CPUs RMS can optionally execute an epilogue script. The scripts execute as root.

RMS will check for prologue/epilogue scripts in the default site specific directory as specified by the attribute local-rmsdir. The default value for this attribute will be /usr/local/rms/etc. The existence of executable scripts prologue and/or epilogue (or symbolic links with the same names) in this directory will be sufficient to cause them to be run.

If such scripts exist CPU allocation and deallocation will not complete until after the scripts have run. Note that this may introduce delays. For example, the user's prun will have completed but the CPUs will not be deallocated until after the epilogue scripts have completed on all nodes.

Where multiple parallel jobs are run on the same CPUs as part of a single resource request prologue will be executed prior to the first job and the epilogue will be executed after the last job.

prologue/epilogue scripts should return 0 on success. A non-zero exit status on any node will be interpreted as an error and will cause the parallel job to fail with the message:

```
prun: Error: allocate failed: atlas0 /usr/local/rms/prologue status 1
```

and the exit status 127. Failure of an epilogue script will not be reported to the user; in general it will be executed after the prun has finished.

If a prologue/epilogue script returns a non-zero exit status the RMS daemon will

automatically configure the node out. In doing so it will set the `reason` field for the node to `prologue failed` or `epilogue failed` as appropriate.

The implementor of prologue/epilogue scripts needs to be aware that there may be other parallel jobs running on the node at the same time as the script. Failure of the script, followed by automatic configure out of the node will cause such jobs to be killed.

The uid and gid of the owner of the program together with the RMS resource id will be passed to the prologue/epilogue scripts as environment variables.

- RMS_UID=1507
- RMS_GID=200
- RMS_RESOURCE=222

9.5.3 Processor Set Support

RMS includes support for processor sets where they are supported in the base OS. Processor set support ensures that applications are restricted to the allocated CPUs. This is important when a node is being shared between multiple users, especially when they are running multi-threaded applications. If the `processor-sets` attribute is set to 1 RMS will create processor sets when a node is shared between two or more resource requests. The CPUs allocated by RMS will be assigned to these processor sets and jobs will be bound to them. Having changed this attribute it is necessary to restart RMS before the change takes effect.

9.5.4 Core File Handling

By default, RMS instructs the operating system to dump core files to local temporary file space under `/local/core/rms`. Change the attribute `local-corepath` in the `attributes` table to select an alternative default directory for core files. Subdirectories are created in *`local_corepath/resource-id`* for each resource request. Change the attribute `rms-keep-core` to disable the keeping of core files.

If dumping is disabled, a core file analysis script is run on at least one node before the core files are deleted. The default script prints a backtrace showing where the program crashed. A site-specific variant might copy core files from the local temporary directory to a global file system for subsequent analysis.

To create a site-specific core file analysis script, copy the default script, `/usr/opt/rms/etc/core_analysis`, to `/usr/local/rms/etc` (or the directory specified by the `local-rmsdir` attribute) and modify it as required. You might, for example, move the first core file to be created to a system wide directory. Take care not to copy large core files to a slow filesystem.

Note

RMS must be restarted in order for changes to the core file control attributes to take effect.

9.5.5 Event Handling

The default event handlers check for the existence of site-specific handlers of the same name in the path specified by the attribute `event-handler-path`, as described in Section 8.2. If such a script exists, it will be executed in preference to the default handler. To make site-specific changes, copy the default scripts to this directory and amend them to your needs. Use `rmspost` (see Section 8.1.1) to test their correct operation.

9.5.6 Switch Manager Configuration

RMS provides switch network error monitoring services for HP AlphaServer SC Interconnect switches with direct control cable connections to system nodes and the newer switches with an embedded controller connected to the management network.

In a system with directly connected switches the `swmserver` monitoring daemon(s) must run on the node(s) with the control cables. To install such a system from scratch use the `-R` to `rmsbuild` passsing it the number of rails.

```
# rmsbuild -m atlas -N 'atlas[0-63]' -R 1
```

This will create an `swmserver` entry for `atlas0` in the `servers` table. For a dual rail system `swmserver` entries will be created for nodes `atlas0` and `atlas1`. These entries will be created automatically by `rmstbladm` when upgrading an existing machine description.

If the switch control cable connections are not on the default nodes (node 0 for rail 0, node 1 for rail 1) or if you are adding a second directly connected switch to an existing system then it is necessary to create the `swmserver` entry by hand. To remove an existing entry use `rcontrol` as follows:

```
# rcontrol remove server=swmserver hostname=atlas0
```

and to create a new entry use `rcontrol` as follows:

```
# rcontrol create server=swmserver hostname=atlas1 port=7200 rms=0 args=1 autostart=1
```

Setting `rms=0` is required as the swmserver does not update the RMS database. Setting `autostart=1` instructs the `rmsmhd` to restart `swmserver` if it exits or is killed. Setting `args=1` specifies that this is the `swmserver` entry for rail 1.

At this point you should restart RMS with `rmsctl restart`.

The `swmserver` daemon consumes CPU time while sampling the network for errors. Therefore, it should ideally be run on a lightly loaded node that is not used to run parallel jobs; for example, a management server. To set the system up in this way connect the control cable to the management server and then use `rmsquery` to change the name of the host used to run `swmserver` and `rmsctl` to restart RMS as follows:

```
# rmsquery "update servers set hostname='atlasms' where name = 'swmserver'"
# rmsctl restart
```

If your system does not have a suitable management server you can reduce the rate at which the `swmserver` polls for errors by changing the system attribute `swmgr-poll-interval` interval from the default value (30 seconds) to 5 minutes. Use `rcontrol` to do this, as follows:

```
# rcontrol create attribute name=swmgr-poll-interval val=300
```

The change in polling frequency will take effect next time the `swmgr` is started. To force the change to occur immediately, use `rcontrol` to restart the server.

No special setup is required to install a system where the HP AlphaServer SC Interconnect switches are connected to the management network.

9.5.7 Controlling Allocatable Memory

The `max-alloc` attributes control the maximum amount of allocatable memory per node. When memory limits are enabled the `pmanager` will not schedule new jobs on a node if doing so would cause the total amount of memory allocated to exceed this value. For example, on a machine with 4GB nodes and memory limits enabled `max-alloc` should be set to a value of approximately 3400 MB to prevent jobs from paging (the precise value will vary between systems). A lower value would reserve memory for system processes. A higher value would allow jobs to be paged out.

In a system where all of the nodes have the same amount of memory you need only use the `max-alloc` attribute itself. If there is a mix of memory sizes then you should set the attributes `max-alloc8` for 8GB nodes and `max-alloc16` for 16GB nodes. These attributes over-ride the default `max-alloc` on such nodes.

Note

It is necessary to restart RMS before changes to the `max-alloc` attributes take effect.

9.5.8 Rate of RMS Status Updates

The rate at which RMS performs status updates can be controlled by setting the following attributes:

Attribute	Value	Description
`rms-poll-interval`	30	Controls the rate at which RMS servers perform their periodic housekeeping operations
`pmanager-`*`name`*`-poll-interval`	30	Controls the rate at which the Partition Manager performs its housekeeping operations and hence the time in which nodes must respond to a command before the partition is marked as `blocked`
`cpu-stats-poll-interval`	30	Controls the rate at which the RMS daemons send statistics and status updates to the Machine Manager

The default poll intervals are appropriate for systems of upto 256 nodes They should be increased to `60` or `90` seconds for 1024 node systems.

The `inactivity-timeout` attribute needs to be set to a value at least 3 times that of `rms-poll-interval`. Its default value (300 seconds) is compatible with the recommendations above.

The `stats-direct` attribute controls whether statistics and status updates are sent direct to the Machine Manager (value 1) or routed via the first node in each cluster (the default, value 0). Setting `stats-direct` to 1 is recommended on large systems with a separate management server, especially when they are running tightly coupled parallel programs.

10

Day-to-Day RMS Operation

10.1 Introduction

This chapter describes how to automate some routine or day-to-day RMS operations including establishing archive and backup procedures. The information is organized as follows:

- Periodic shift changes (see Section 10.2)
- Summarizing accounting data (see Section 10.3)
- Archiving data (see Section 10.4)
- Backing up the database (see Section 10.5)
- Restoring the database (see Section 10.6)
- Troubleshooting (see Section 10.7)
- Draining jobs from nodes (see Section 10.10)
- Configuring nodes in or out for maintenance or repair (see Section 10.8)

10.2 Periodic Shift Changes

RMS supports periodic shift change through a `cron(8)` job that changes the active configuration. For example, a system can be set up with two operating configurations:

one called day and the other called night. During the day, the resources are split into two partitions: a login partition (called login) for program development and a parallel partition (called parallel) for execution of short parallel programs. At night, all of the nodes are assigned to a single partition (again called parallel) with a longer time limit for running parallel jobs.

Use the following commands to create this pair of configurations:

```
# rcontrol create partition=login configuration=day type=login nodes='atlas[0-7]'
# rcontrol create partition=parallel configuration=day type=parallel \
                  timelimit=600 nodes='atlas[8-63]'
# rcontrol create partition=parallel configuration=night type=parallel \
                  timelimit=3600 nodes='atlas[0-63]'
```

To start the day configuration, enter the following:

```
# rcontrol start configuration=day
...
```

To switch to the night configuration, use this command:

```
# rcontrol start configuration=night
...
```

Note that, after the switch, any jobs running on the parallel partition will continue to run as the parallel partition in the configuration night has more nodes. However, when changing back from night to day, you must decide what to do with any jobs that are running on nodes 'atlas[0-7]'. The options are to wait for them to finish or to kill them. To wait for them to finish, start the configuration with the wait option:

```
# rcontrol start configuration=day option=wait
...
```

To kill them, start the configuration with the kill option:

```
# rcontrol start configuration=day option=kill
...
```

When you are satisfied with the shift changes, install a cron(8) job to perform them automatically.

10.3 Summarizing Accounting Data

Accounting records accumulate in the RMS database as each job is run. By default, a record is included in the archive_tables table to archive them (see Section 10.4) but

you may prefer to delete them only after a report has been generated. Each site has its own requirements in this respect. A simple example script to produce a summary of resource usage is included in the release in `/usr/opt/rms/examples/scripts/accounting_summary`. See Appendix E (*Accounting Summary Script*) for a listing. The script produces the following output.

```
Accounting Summary of Machine atlas at 16:01 Wed 21 Feb 2001
Usage by Project/User For Previous Day

Project    User               CPU          User           Sys  Number
Name       Name              Secs          Secs          Secs Sessions
-------------------------------------------------------------------------
default    addy               596           533             6      8
           duncan              58            37             2      6
           johnt              540           227            51     15
           root             29272             2             8     37
           stephen            286            87           134     56
-------------------------------------------------------------------------
Total default               30752           886           201    122
-------------------------------------------------------------------------
Grand Total                 30752           886           201    122
-------------------------------------------------------------------------
```

When the accounts have been processed, the script can optionally delete the accounting records for resource requests that have completed.

This script (or one based on it) can be run nightly with a `cron(8)` job, as shown in the following example.

```
0 0 * * * /usr/opt/rms/examples/scripts/accounting_summary
```

10.4 Archiving Data from the RMS database

A number of tables (see Section 11.4.4) in the database grow over time or as jobs are submitted, for example, the `events` table and the `jobs` table. These tables can be kept to a reasonable size by periodically running `rmsarchive`. This command archives selected records and then deletes them from the database.

`rmsarchive` uses criteria specified in the `archive_tables` table (see Table 11.3) to determine which records to archive. Each entry in the `archive_tables` has four fields:

`name` The name of a table to archive.

`lifetime` The maximum time in hours for the data to remain in the table.

`timefield` The name of the field in `name` that records the data's lifetime.

`selectstr`	A SQL select statement, based on fields in `name`, that determines which records to archive. If not null it is used in addition to the default selection based on `lifetime` and `timefield`.

Use `rmsquery` (see Page 5-63) to create or update entries in the `archive_tables` table.

For example, to add an entry that causes `rmsarchive` to delete all records in the `transactions` table that are more than three days old:

```
$ rmsquery "insert into archive_tables (name,lifetime,timefield) \
            values ('transactions',72,'mtime')"
```

The following example causes `rmsarchive` to delete all records from the `transactions` table that denote completed transactions:

```
$ rmsquery "insert into archive_tables (name,lifetime,timefield,selectstr) \
            values ('transactions',72,'mtime','status = \'complete\'')"
```

Note that it is necessary to escape the single quotes in the `selectstr`.

`rmsarchive` archives the records as a sequence of SQL statements in a file called *`machine_date`*`.sql`, where *`machine`* is the name of the RMS machine and the *`date`* format is: `yyyy-mm-dd-HH:MM`. The archive file is compressed using `gzip(1)` and stored in `/var/rms/archive`.

If you want to archive the database of a machine which is not the database of the host machine, give the name of the machine as a command line argument to `rmsarchive`. For example, to archive the database of a machine called `juno`:

```
$ rmsarchive -m juno
```

When the archive process is complete, `rmsarchive` sends mail to any users listed by the `users-to-mail` attribute in the `attributes` table. This can be set using `rcontrol` (see Page 5-27).

The mail message includes the name of the archive file, which tables were archived and how many records were deleted, as shown in the following excerpt:

```
rmsarchive: Nov 10 01:05:01

rmsarchive: '/usr/bin/rmstbladm  -c -x 73 -v' started Nov 10
01:05:00
rmsarchive: finished Nov 10 01:05:01 total records 2918 file is
/var/rms/archive/atlas_2001-11-10-01:05.sql

rmsarchive: rmstbladm output was ...
rmstbladm: archiving to /var/rms/archive/atlas_2001-11-10-01:05.sql
rmstbladm: archiving table 'rms_atlas:rms:acctstats': 4 records dumped
```

```
rmstbladm: archiving table 'rms_atlas:rms:disk_stats': 0 records dumped
rmstbladm: archiving table 'rms_atlas:rms:events': 4 records dumped
rmstbladm: archiving table 'rms_atlas:rms:jobs': 4 records dumped
rmstbladm: archiving table 'rms_atlas:rms:link_errors': 2902 records dumped
rmstbladm: archiving table 'rms_atlas:rms:resources': 4 records dumped
rmstbladm: gzipping '/var/rms/archive/atlas_2001-11-10-01:05.sql'
```

`rmsarchive` also adds a record to the `backups` table to show the status of the archive procedure. For example:

```
name     ctime             handle  filename                                      archived
-------------------------------------------------------------------------------------------
archive  11/10/01 01:05:01 49      /var/rms/archive/atlas_2001-11-10-01:05.sql 2918
```

The archive procedure can be automated by adding an entry to the `crontab` file for `cron(8)` to execute. For example, to run `rmsarchive` daily at 1:05 a.m. using `cron`, add the following line to the `root crontab` file on the RMS host machine:

```
5 1 * * * /usr/bin/rmsarchive
```

`rmsarchive` invokes `rmstbladm` to manipulate the database. `rmstbladm` can be used directly to archive specific tables. The following example archives (without deleting) data from the `events` table (see Section 11.5.9):

```
$ rmstbladm -d -t events > events.sql
```

Alternatively, you can execute a SQL query to extract (but not delete) data, as shown in the following examples. The second example selects data collected 2 days or more ago.

```
rmsquery -v "select * from jobs where endTime <> 0 and \
 endTime < $old order by startTime" > jobs.dat

now=‘rmsgettime‘
old=‘expr $now - 172800‘
rmsquery -v "select * from backups where ctime <= $old \
  order by ctime" > link_errors.sql
```

Failure to clear old entries can cause problems as described in Section 10.7. Note that the default `archive_tables` table contains an entry for the accounting statistics table. You may prefer to remove it from the `archive_tables` table and use a script as described in Section 10.3 to delete accounting statistics records only after a report has been generated.

10.5 Backing Up the RMS Database

To back up the database, use `rmsbackup`. `rmsbackup` backs up the tables as a set of SQL statements that can be restored using `rmstbladm`. The SQL statements are written

to a file called *machine_date*.sql, where *machine* is the name of the RMS machine and the *date* format is: yyyy-mm-dd-HH:MM. The back-up file is compressed using gzip(1) and stored in /var/rms/backup.

By default, rmsbackup archives the database first to remove any redundant entries. To prevent the archiving, specify the -b flag on the command line.

If you want to back up the database of a machine which is not the database of the host machine, give the name of the machine as a command line argument to rmsbackup. For example, to back up the database of a machine called called juno:

```
$ rmsbackup -m juno
```

When the backup is complete, rmsbackup sends mail to any users listed by the users-to-mail attribute in the attributes table. This can be set using rcontrol (see Page 5-27).

The mail message includes the name of the backup file, which tables were backed up how many records each table contained, as shown in the following excerpt:

```
rmsbackup: Nov  7 01:05:04

rmsbackup: '/usr/opt/rms/bin/rmstbladm  -b -x 62 -v' started Nov  7 01:05:02
rmsbackup: finished Nov  7 01:05:04 total records 23548 file is
/var/rms/backup/atlas_2001-11-07-01:05.sql

rmsbackup: rmstbladm output was ...
rmstbladm: backing up to /var/rms/backup/atlas_2001-11-07-01:05.sql
rmstbladm: backing up table 'rms_atlas:rms:access_controls': 1 records
dumped
rmstbladm: backing up table 'rms_atlas:rms:acctstats': 322 records dumped
rmstbladm: backing up table 'rms_atlas:rms:archive_tables': 7 records
dumped
rmstbladm: backing up table 'rms_atlas:rms:attributes': 29 records dumped
rmstbladm: backing up table 'rms_atlas:rms:backups': 57 records dumped
...
rmstbladm: backing up table 'rms_atlas:rms:users': 10 records dumped
rmstbladm: gzipping '/var/rms/backup/atlas_2001-11-07-01:05.sql'
```

rmsbackup also adds a record to the backups table to show the status of the back-up procedure. For example:

```
name     ctime            handle filename                                    archived
-------------------------------------------------------------------------------------
backup   11/07/01 01:05:01 49    /var/rms/archive/atlas_2001-11-07-01:05.sql 23548
```

The back-up procedure can be automated by adding an entry to the crontab file for cron(8) to execute. For example, to run rmsbackup daily at 1:05 a.m. using cron, add the following line to the root crontab file on the RMS host machine:

```
5 1 * * * /usr/bin/rmsbackup
```

10.6 Restoring the Database

Before restoring the database from a backup file, all jobs should be killed using `rcontrol` (see Page 5-27) and RMS should be halted using `rmsctl` (see Page 5-57). All updates to the database made since the backup was taken will be lost.

To restore the data, use `rmstbladm` as follows:

```
# rcontrol kill resource partition=parallel
# rmsctl stop
# rmstbladm -r machine_date.sql.gz
# rmsctl start
```

`rmstbladm` automatically unzips the file into `/tmp` before starting to restore the data.

10.7 Troubleshooting

The RMS database is stored in `/var/rms/msqldb-3` on the `rmshost` node. The database server will exit if this file system fills up. RMS will not operate until sufficient space has been created in this file system. Ensure that there is at least 100MB free to allow for updates. The database server can be restarted using the script `/sbin/init.d/msqld`.

If the database tables are not cleared out on a regular basis, the database continues to grow until the performance of RMS is affected. Indications that this is happening include the following:

- The database server, `msql3d`, uses more memory.
- The table join operations performed by `rinfo` take longer.
- Queries acting on large tables may exceed normal user memory limits.
- `rmsarchive` (or `rmstbladm`) takes a long time to clear out old entries or may fail, although insert operations succeed and the tables continue to grow.

The point at which memory limits are exceeded varies with the number of nodes in the machine and the amount of memory on the `rmshost` node. To check that the size of the database is within operating limits, enter the following query:

```
$ rmsquery; "select * from events" > /tmp/stats.sql
```

If this fails, follow these steps to recover from the problem:

1. Log in to the rmshost node as root and stop the database server, as follows:

```
# /sbin/init.d/msqld stop
MSQL: service stopped
```

2. Change to the directory that contains the database, for example:

```
# cd /var/rms/msqldb-3/rms_atlas
```

   Delete the following files: events.dat, events.def, events.idx and events.ofl.

```
# rm events.*
```

3. Restart the database server, as follows:

```
# msql3d start
MSQL: daemon started
```

4. Create a new events table, as follows:

```
# rmstbladm -u
```

After this, rmsarchive (or rmstbladm) should be run to clean out old entries and it should now succeed.

10.8 Configuring Nodes Out

If a node fails and cannot be rebooted, it must be configured out while it is being repaired. The procedure for this is as follows:

Configure out the node (atlas2 in this case). Any jobs that are running on the failed node will be killed. Other jobs will continue to run.

```
# rcontrol configure out node=atlas2 reason="maintenance"
```

The reason field should be entered to distinguish the node from one which has been automatically configured out by the partition manager because it was not responding. Nodes which are automatically configured out will have the reason field set to automatically configured out. The rinfo command will show the reason field against configured out nodes when used with the -nl flags.

```
# rinfo -nl
...
REASON CONFIGURED OUT           NODES
node is down                    atlas[642,740,764,767]
Elan position not found         atlas[662,697,735]
automatically configured out    atlas[720,741]
```

If jobs are running on a node you will be prompted for whether to kill them or not before configuring out the node. For example:

```
# rinfo
MACHINE     CONFIGURATION
atlas       day

PARTITION       CPUS    STATUS        TIME   TIMELIMIT  NODES
root              16                                    atlas[0-3]
parallel        4/16   running       49:12              atlas[0-3]

RESOURCE        CPUS    STATUS        TIME    USERNAME  NODES
parallel.49386     4 allocated       00:02      duncan  atlas[0-3]

JOB             CPUS    STATUS        TIME    USERNAME  NODES
parallel.39809     4   running       00:02      duncan  atlas[0-3]

# rcontrol configure out atlas2
rcontrol: will kill user jobs. Continue (y/n): y
```

The configure out operation will fail if you elect not to kill the jobs. See Section 10.10 for details of how to drain jobs from a node and have it configured out automatically when the jobs complete.

After this procedure, the partition runs without the node. This reduces temporarily the maximum size of job that can run.

When the node has been repaired, configure the node back in as follows:

```
# rcontrol configure in node=atlas2
```

Configuring the node back in will clear the `reason` field and brings the partition back up to its full complement of nodes.

10.9 Cleanup

When a parallel job completes RMS will execute its *cleanup* process before allocating the CPUs to another resource request. The cleanup process consists of instructing the `rmsd` on each node that was in use to kill any remaining processes, destroy Elan capabilities, remove shared memory segments, (optionally) delete core files and execute the epilogue script. This process normally completes quickly, but can be delayed.

Note

The cleanup process will be delayed if any of the system calls required blocks. This may also make the `rmsd` unresponsive.

A long delay in cleaning up after a parallel job will cause the partition to block. This is indicative of an error on the node. RMS will configure such nodes out if the time taken to cleanup after a parallel job exceeds the value specified in the `cleanup-timeout` attribute. The node will also be configured out if a blocked system call causes the partition as a whole to be blocked for more than the time specified by the `partition-blocked-timeout`. Both attributes are specified in seconds.

10.10 Draining Jobs From Nodes

Configuring a node out will kill the jobs running on it. Where this is not necessary (because, for example, the node is to be taken down for preventative maintenance) the jobs can be *drained* first. To do this use `rcontrol` with the `drain` command as follows:

```
# rcontrol drain nodes = 'nodelist'
```

where *`nodelist`* is a list of nodes in the same partition. The nodes will be configured out when the current CPU allocations are completed. The nodes will not be allocated to any new or queued resource requests. Requests for specific sets of nodes that span a node being drained will fail if one of the requested nodes is being drained. For example

```
# prun -B0 -N6 -R hwbcast=1 hostname
prun: Error: can't allocate 6 cpus on nodes 0-5: atlas4 is being drained
```

Similar requests that did not specify a precise set of nodes will succeed, avoiding nodes that are being drained. Requests to configure out a node that is being drained will result in jobs running on that node being killed and the node being configured out immediately.

Information on nodes being drained will be maintained in the nodes table. It will be reported by rinfo as follows:

```
# rinfo -nl
STATUS          NODES
running         atlas[0-4,6-7]
drain           atlas4
configured out  atlas5

REASON CONFIGURED OUT         NODES
drain complete                atlas5
```

State on nodes being drained will be preserved over a partition restart. The nodes will be configured out when the jobs complete. You should wait until nodes have been drained before changing configuration (an error will be generated if you try and change configuration with jobs running if making the change would split a job across two or more partitions).

The cancel option allows you to cancel a drain operation on one or more nodes. For example

```
# rcontrol drain node atlas4 option cancel
```

10.11 Log Files

The RMS daemons output reports to log files in the directory `/var/rms/adm/log`. The amount of detail is controlled for each daemon by setting a reporting level. By default, the reporting level is set to 0.

The reporting level is a bitmap that turns on different reports. Values for the reports are as follows:

Symbolic Name	Value	Description
INIT_DEBUGGING	1	Server initialization and shutdown messages
REQ_DEBUGGING	2	Requests made to servers
JOB_DEBUGGING	8	Job startup and change of state
RESOURCE_DEBUGGING	32	Resource allocation and change of state
EDIT_DEBUGGING	64	SQL queries
MALLOC_DEBUGGING	256	Monitor server memory allocation

If resource debugging is enabled, the Partition Manager logs scheduling decisions to the `scheduler_log` table (see Table 11.30). It adds an entry to the table when a new request is queued, when a resource is allocated for the request, each time a job is suspended or resumed and when a job completes.

The level of reporting can be controlled in three ways.

1. On an individual user basis, by setting the environment variable `RMS_DEBUG`.
2. Using `rcontrol` to reload the daemon with a specified debug value. For example, the following command reloads the Machine Manager with a reporting level of 1:

   ```
   # rcontrol reload server=mmanager debug=1
   ```

   The following example reloads the Partition Manager for the `par1` partition with a reporting level of 41 (initialization, job and resource information)

   ```
   # rcontrol reload partition=par1 debug=41
   ```

3. Using `rmsquery` to set the `args` field of the daemon's entry in the `servers` table (see Section 11.5.21) to `-r` *value*, where *value* is the required reporting level.

   The following example gives the Partition Manager for the `par1` partition a reporting level of 33.

```
# rmsquery "update servers set args='-r 33' where name='pmanager-par1'"
```

Then restart the Partition Manager. This change remains in place each time the partition is restarted. The output files in `/var/rms/adm/log` can grow in size rapidly when debug options are enabled. Take care not to fill the file system.

10.11.1 Creating new log files

The RMS daemons will detach from their current log file, move it to a dated subdirectory of `/var/rms/adm/log.dated` and create a new logfile in `/var/rms/adm/log` when sent a `SIGHUP`.

11

The RMS Database

11.1 Introduction

This chapter describes the tables which make up the RMS database. Each machine has its own database, called `rms_`*`machine`*, where *`machine`* is the name of the machine. This allows a single database server to support multiple machines.

The information in this chapter is organized as follows:

- Section 11.2 explains how data is stored in the tables and represented in this chapter. It also describes the different levels of access to the data.
- Section 11.3 describes how table definition files are used to generate the tables and their contents.
- Section 11.4 groups the tables into functional categories.
- Section 11.5 specifies the fields in each of the tables, listing them in alphabetical order.

11.2 General Information about the Tables

- All of the field names in the database are case sensitive.
- Fields are of these types:

  `char(`*`length`*`)` This denotes a character string of the specified *`length`*.

`int`	This denotes an integer value.
`real`	This denotes a real numeric value.
`%`	This denotes a percentage value stored in an integer field.
`UTC`	This denotes a UTC time value stored in an integer field. Client programs should convert time to local time and output the result as a string.
`x-y`	This denotes a range of possible integer values.
`text(`*`length`*`)`	This denotes a character string of arbitrary length. The average length is expected to be *`length`*. Fields of type `text` can be selected by the field name but the text entry cannot be matched. If the text is a list of items, for example, a list of node names, the items in the list may be separated by white space. A list of names, all of which share a common base, for example, `atlas0 atlas1 atlas2`, may also be represented by a `glob`-like expression, in this example, `atlas[0-2]`. The normal `glob(7)` expression syntax is relaxed to include multiple digit numbers. For example, `atlas[0-10,14]` represents the nodes numbered from `atlas0` to `atlas10` inclusive plus `atlas14`.

- Default values may be assigned to fields in the table definition files as described in Section 11.3. Symbolic names may be used to specify these values. In the listing of tables in Section 11.5, symbolic names are represented as follows:

Symbolic Name	Representation
`EMPTY_STRING`	`""`
`RMS_UNASSIGNED`	`-`
`TRUE`	`T`
`FALSE`	`F`

  Values defined as `not_null` (see Section 11.3.2) are indicated by a superscript (^{NN}) next to their default value.

- There are three levels of access to the database:

  1. Users can extract information from all of the tables but cannot update them.
  2. Operators and administrators can extract information from all of the tables and, in addition, update a limited selection of fields in some tables.
  3. RMS itself can extract and update information in all fields of all tables. The description of the tables in Section 11.5 includes information about which RMS

programs create and update each field.

11.3 Table Definition Files

RMS supports the definition of database tables through text files. When `rmstbladm`, the database administration program, creates or updates the database, it checks for the presence of table definition files. If a table definition file called *`tablename`* is present, `rmstbladm` parses it to generate a table called *`tablename`*.

`rmstbladm` looks in `/usr/opt/rms/msql/rms` for table definitions for RMS and in `/usr/opt/sra/msql` for table definitions for other components. The components directory has a subdirectory for each component. The subdirectory has the same name as the component.

By default, `rmstbladm` parses all of the table definition files in these directories. The `-C` and `-t` options can be specified to restrict this to a specified component or table. See Page 5-66 for more details of using `rmstbladm`.

A table definition file is a plain text file. Each line in the file begins with a tag that introduces a definition. Lines that start with a hash character, #, are treated as comments.

The tags are as follows:

- `table` (see Section 11.3.1)
- `field` (see Section 11.3.2)
- `index` (see Section 11.3.3)
- `sequence` (see Section 11.3.4)
- `default` (see Section 11.3.5)

The definitions that may follow each tag are described in the following sections.

11.3.1 Table Definition

The `table` definition has the following format:

`table` *`name`*

This definition specifies a *`name`* for the table. This must be the same as the name of the definition file.

The definition file may only contain one line with the `table` tag.

For example, to define a table called `users`, the table definition file would be called `users` and would include the following line:

```
table users
```

11.3.2 Field Definition

The `field` definition has the following format:

`field` *`name type default`* `[not_null]`

This definition specifies a field called *`name`*.

The *`type`* takes one of the following forms:

`INT`	This specifies an integer type.
`REAL`	This specifies a real numeric type.
`CHAR` *`length`*	This specifies a character string with a maximum *`length`*. If the string contains white space characters, it must be enclosed in double quote marks.
`TEXT` *`length`*	This specifies a string of words delimited by double quotation marks. The average length of the text is estimated as *`length`* but length is unrestricted. If the string contains white space characters, it must be enclosed in double quote marks.

A *`default`* value may be specified for the field. This value will be used (using the `default` keyword) if the field is not explicitly set. A number of symbolic names can be entered as defaults. They are as follows:

Symbolic Name	Value
`NULL`, `Null` or `null`	Null
`EMPTY_STRING`	An empty string (`CHAR` and `TEXT` types only)
`RMS_UNASSIGNED`	No value assigned (`INT` and `REAL` types only)
`FALSE`	0 (`INT` types only)
`TRUE`	1 (`INT` types only)

The optional `not_null` entry prevents the database server from assigning a null value to the field. You cannot specify `not_null` if the `default` value has been set to null. If `not_null` has been specified and a default entry field has been assigned a null value, `rmstbladm` will error with a message similar to the following:

```
rmstbladm: Error: rms_atlas:rms:partitions: field 'nodes' can't be Null at line 17
```

The definition file may contain multiple lines with the `field` tag.

For example, to add a field called `username` which holds a maximum of 16 characters

with an empty string as the default value and which is guaranteed never to have a null value:

```
field username CHAR 16 EMPTY_STRING not_null
```

To add a field called `userID` which has an integer value initialized to 0:

```
field userID INT 0
```

To add a field called `projects` which can hold any number of characters (but the average length is estimated as 64) and has a default value of `p1 p2`:

```
field projects TEXT 64 "p1 p2"
```

11.3.3 Index Definition

The `index` definition has the following format:

`index` *`name`* `[unique]` *`field`* `[`*`field`* `...]`

This definition nominates one or more *`field`*s for use as an index (called *`name`*) into the table. Multiple fields are combined to form a compound index. The use of indices improves performance where the indexed fields are used in the `where` clause of a SQL query. Note that fields of type `TEXT` cannot be used as index fields.

The optional `unique` tag is used to assert that the index formed by the *`field`*s must produce a unique value. The database server will reject any attempt to insert a record that duplicates the index.

The definition file may contain multiple lines with the `index` tag.

For example, to create a unique index called `index1` from the fields `username` and `userID`:

```
index index1 unique username userID
```

11.3.4 Sequence Definition

The `sequence` definition has the following format:

`sequence [`*`value`*`] [`*`step`*`]`

This definition associates a sequence with the table. The sequence is maintained by the database server. It facilitates atomic operations (such as getting the next sequence value) for database clients. Each time the value of the sequence is selected by a client, the database server returns the current value and then increments it.

The optional *`value`* specifies the initial value of the sequence. The optional *`step`* defines the amount to increment or decrement the value. By default, the initial value is 1 and the step is 1.

The definition file may contain only line with the `sequence` tag.

For example, to set up a sequence that starts at 5 and increments in steps of 2:

```
sequence 5 2
```

11.3.5 Default Definition

The `default` definition has the following format:

`default` *`field=value`* [*`field=value ...`*]

This definition declares that a record should be inserted in the table with each *`field`* set to the corresponding *`value`*. Values for fields of type `CHAR` and `TEXT` should be entered in double quotation marks.

The definition file may contain multiple lines with the `default` tag.

For example, to insert a record into the table used in the examples:

```
default username="guest" userID=43 projects="demo p1"
```

11.4 Categories of Table

The database contains tables storing information on the following:

- The configuration of the machine and its nodes (see Section 11.4.1)
- The configuration of the HP AlphaServer SC Interconnect (see Section 11.4.2)
- The configuration of the user population: the access controls and resource quotas applied to them (see Section 11.4.3)
- The operation of the machine, including its current state, the jobs in progress, the resource requests, the resources in use, current transactions and accounting records (see Section 11.4.4)
- The internal workings of the machine, largely transparent to administrators and users Section 11.4.5

11.4.1 Configuration of Nodes

The following tables contain information about the individual nodes and about the machine as a whole.

nodes	describes the attributes of each node
node statistics	contains performance data for each node
module types	describes the characteristics of each supported module
software products	describes the components of each software product
installed components	describes the components installed on each node
servers	holds details on each daemon (hostname, port number, pid)
services	describes the services available and who can use them
event handlers	lists the handlers used to act on events
partitions	defines each partition and its scheduling parameters
scheduler log	logs scheduling decisions
attributes	holds site-specific attribute-value pairs
status	contains a summary of the state of objects (currently only nodes)
fields	specifies how objects and attributes may be modified
archive tables	defines which tables and which records to archive
backups	logs the status of backup and archive procedures

11.4.2 Configuration of the Network

The following tables describe the network components. Definitions of terms used in describing the HP AlphaServer SC Interconnect can be found in Appendix A (*HP AlphaServer SC Interconnect Terms*).

elans	records the position and state of the Elan network adapters
elites	records the position and state of the Elite switches
switch boards	records the position and state of each switch board
switch modules	records the configuration and state of each switch module
unidentified MACs	lists the network addresses of unassigned switch modules
terminal servers	holds details of each terminal server connected to the machine
terminal server types	defines the types of terminal server in use
tree status	describes a tree of nodes connected to a Partition Manager

11.4.3 User Details

The following tables contain information about the users of the RMS: their privileges and priorities.

(continued on next page)

(continued from previous page)

users	lists the projects to which users belong
projects	lists the projects
access controls	describes limits on user access to resources

11.4.4 Operational State

The following tables hold details of the current state of the machine. The tables are constantly being updated as RMS uses them in the course of providing its services to users. Once a particular service has been provided, the information in the tables is of no further relevance to RMS. These tables should be archived regularly to keep their size to a minimum (see Section 10.4).

resources	describes the allocation of resources to users
jobs	describes the jobs currently being processed
events	records changes to the state of the machine
transactions	records requests to change the machine configuration
link errors	logs network errors
accounting statistics	contains an accounting record for each resource

11.4.5 Internal Tables

The RMS database includes a number of tables that are mainly used internally. These are noted in this chapter as being of internal use but are not described in any detail.

transaction outputs	contains output from requests posted to the transaction log
request types	describes output formats in the transaction outputs table

11.5 Listing of Tables

This section lists the tables in alphabetical order.

11.5.1 The Access Controls Table

The `access_controls` table, shown in Table 11.1, contains access control and usage limit descriptions for users and projects.

Table 11.1: Access Controls Table

Field	Type	Default	Description
name	char(16)	""NN	name of the user or project
class	char(8)	""NN	class of control: user or project
partition	char(16)	""NN	partition to which access controls apply
priority	int	50	default scheduling priority (0-100)
maxcpus	int	0	maximum number of CPUs
memlimit	int	0	maximum memory per CPU in megabytes

The name and partition fields are combined to provide a unique index into the table.

An entry for the reserved partition name default specifies the priority, maxcpus and maxmem that should apply for any partition names not explicitly specified.

The priority field stores the default scheduling priority of jobs submitted by a user or project. The higher the value, the more likely the job is to run. Priority values range from 0 to 100.

The maxcpus field stores the maximum number of CPUs that a user or project may have allocated at once. Requests for more than this number of CPUs fail. Once this number of CPUs is allocated, additional requests block until some CPUs are freed.

The memlimit field stores the default memory limit per CPU for jobs submitted by the named user or project.

11.5.2 The Accounting Statistics Table

Each time CPUs are allocated to a request, a record is created in the accounting statistics (acctstats) table shown in Table 11.2. Records are updated periodically and at the end of each job by the Partition Manager, pmanager (see Section 4.4).

Table 11.2: Accounting Statistics Table

Field	Type	Default	Description
name	char(16)	""NN	name of the allocated resource
uid	int	0NN	user ID
project	char(16)	""NN	project name
started	UTC	0NN	time when CPUs were allocated
ctime	int	0NN	time last modified
etime	char(16)	""NN	elapsed time in seconds
atime	char(16)	""NN	CPU seconds allocated

(continued on next page)

Table 11.2: Accounting Statistics Table (cont.)

Field	Type	Default	Description
utime	char(16)	""NN	user time in seconds
stime	char(16)	""NN	system time in seconds
cpus	int	0NN	maximum number of CPUs used
mem	int	0NN	maximum memory extent in megabytes
pageflts	int	0NN	number of page faults requiring I/O
memint	int	0NN	memory integral in megabytes per hour
running	int	0NN	CPUs allocated and running jobs (0 | 1)

The name field provides a unique index into the table.

The etime field stores the elapsed time (in seconds) since CPUs were first allocated to the resource, including any time during which the resource was suspended.

If a partition is stopped while a job is running and then restarted before the job completes, the etime field will correctly show the total elapsed time of the running job including the time when the partition was down. If a partition is stopped while a job is running and then restarted after the job completes, the acctstats table entry will reflect only the time when the partition was running. Any additional time that the job was running while the partition was down is not included in the acctstats table entry. If a job is terminated with rcontrol or because it exceeds its time limit, the etime field reflects the time for which CPUs were allocated.

The atime field stores the total elapsed time (in seconds) that CPUs have been allocated – this excludes time during which the resource was suspended but includes any time when the partition was down while jobs were running. The value stored is the total for all CPUs used by the resource.

The utime and stime fields are summed over all processes in the program.

The pageflts field shows the number of page faults requiring I/O summed over all of the processes in the parallel program. It is normally 0. A non-zero and growing value indicates that the program is paging on some or all nodes.

The memint field is set to 0 in HP AlphaServer SC Version 2.5.

The running field is set to 1 while CPUs are allocated. It is set to 0 when the CPUs are deallocated, at which point no further updates take place.

The number of entries in the accounting statistics table can grow rapidly. The table should be cleared periodically of old entries as described in Section 10.3.

11.5.3 The Archive Tables Table

The `archive_tables` table, shown in Table 11.3, is used to specify how records in the database are selected by `rmstbladm` (see Page 5-66) for archiving. For details of the archive and backup commands, see Page 5-49 and Page 5-52.

Table 11.3: Archive Tables Table

Field	Type	Default	Description
`name`	char(32)	""NN	name of table to archive
`lifetime`	int	48NN	lifetime in hours for records
`timefield`	char(16)	""NN	name of the field which records the lifetime
`selectstr`	text(32)	""	SQL `select` string

The `name` field provides a unique index into the table. It specifies a table to be archived.

The records in this table are selected for archive if the field named by `timefield` is older than `lifetime`. A further selection can be applied by inserting an SQL query into `selectstr`. The fields used in the `selectstr` must be in the named table.

The `lifetime` field specifies in hours the amount of time the selected records are to remain in the table before any archiving takes place.

The `timefield` selects the field to use for calculating the age of the records. See Section 10.4 for more details about how to use this table.

By default, the `archive_tables` table is created with the entries shown in Table 11.4.

Table 11.4: Default Entries in the Archive Tables Table

name	lifetime	timefield	selectstr
`jobs`	48	`endTime`	`"status <> 'running' and status <> 'reconnect' and status <> 'suspended' and status <> 'blocked'"`
`resources`	48	`endTime`	`"status <> 'allocated' and status <> 'reconnect' and status <> 'suspended' and status <> 'blocked' and status <> 'queued'"`
`acctstats`	48	`started`	`"running=0"`
`link_errors`	168	`mtime`	`null`
`events`	48	`ctime`	`"handled=1"`

11.5.4 The Attributes Table

The `attributes` table stores information specific to the site or the release. This information is stored as attribute-value pairs as shown in Table 11.5.

Table 11.5: Attributes Table

Field	Type	Default	Description
name	char(32)	""NN	name of the attribute
val	char(128)	""NN	value of the attribute

The `name` field provides a unique index into the table.

The table is created by the table administration program, `rmstbladm` (see Page 5-66), which inserts any default entries specified in the table definition file. Further attributes are inserted by RMS daemons. Values can also be inserted or modified by an administrator using `rcontrol` (see Page 5-27).

The entries in the `attributes` table can be grouped into four sections:

1. Machine description attributes
2. Site specific system attributes
3. Statistics collection attributes
4. Server attributes
5. Parallel processing attributes

Machine Description Attributes

The machine description attributes shown in Table 11.6 are supported:

Table 11.6: Machine Description Attributes

Attribute	Default	Description
`network-type`	`QMS16`	data network type
`network-levels`	`2`	number of levels of switch network
`network-rails`		number of rails present in switch network
`network-railmask`		bitmask of rails present in switch network
`racks`	`1`	number of 19" racks in the machine
`units-per-rack`	`40`	height of a 19" rack in units

Minimum and maximum values for various fields are defined in `rms/rmsdefs.h` which is in `/usr/opt/rms/include`.

The `network-type`, `network-levels` and `network-rails` attributes are all created by SC Viewer. Valid network types are shown in Table 11.24.

The `racks` and `units-per-rack` are created by `rmstbladm` from default entries in the table definition file but they can be modified by an administrator.

System Specific Attributes

The system specific attributes shown in Table 11.7 are supported:

Table 11.7: System Specific Attributes

Attribute	Default	Description
`rmshost`		node running the RMS daemons
`version`	`2.71`	RMS version number
`event-handler-path`		path to event handler scripts
`altzone`	`0`	shift in seconds to apply to UTC time to get local time on `rmshost`
`elan-disconnected-sif`	`""`	The signal to be sent to the processes in a parallel application if the node they are running on becomes disconnected from one or more network rails. No signal is sent by default.
`inherit-user-limits`	`""`	Where memory limits are disabled RMS will set UNIX per-process memory limits based on the defaults for the application node unless this variable is created and its value set to 1, in which case memory limits will be inherited from the `prun`
`memsplit-enabled`	`""`	If this attribute is created and its value set to 1 the RMS scheduler will subdivide blocks of free nodes into subsets where all nodes have the same amount of memory. This behaviour is not enabled by default. It can be useful when running a system with memory limits enabled.

(continued on next page)

Table 11.7: System Specific Attributes (cont.)

Attribute	Default	Description
modify-time		internal attribute used by SC Viewer
osarch	OSF1	operating system
rms-hostname-suffix	""	hostname suffix
broadcast-tree-*rail*	""	The broadcast tree for a given rail.
cleanup-timeout	""	The time (in seconds) that RMS will spending trying to cleanup after a parallel job. The RMS daemon will configure its node out if this timelimit is exceeded. This attribute is used to protect the system as a whole from errant behaviour on one node, for example a process that does not exit when killed with SIGKILL. The minimum value of this attribute is 900 seconds.
processor-sets	0	Enable support for processor sets by setting the value of this attribute to 1 and restarting RMS
linkerror-lifetime	3600	The time (in seconds) that must elapse without a error report on a link before the swmgr closes a record in the link-errors table.
partition-blocked-timeout	""	The time in secs that the Partition Manager will allow the partition to be blocked before polling the responsiveness of each RMS daemon and configuring out the nodes that do not respond. This behaviour is not enabled by default as it is generally only required when a kernel bug causes system calls to block. This attribute should be set to at least 900 seconds (15 minutes).

(continued on next page)

Table 11.7: System Specific Attributes (cont.)

Attribute	Default	Description
set-vm-limit	""	Where memory limits are enabled RMS will set the UNIX per-process virtual memory limit unless this attribute is created and its value set to 0.
swmgr-errorfree-interval	3600	The time (in seconds) that a link must be free of errors before a link error record is closed by the swmgr.
swmgr-resend-interval	300	The time (in seconds) after which the switch network monitor swmserver should resend a report that does not receive acknowledgement. The swmgr will report an error if it does not receive heartbeat messages from the switch monitors in twice this period.
swmgr-reporting-threshold	15	The number of errors logged on a link before further error reports are generated by the switch monitor swmserver.
use-ssh	""	if set prun will use ssh rather than rsh to execute administrative commands.
ssh-args	""	args to be passed to ssh when used by prun with the -r option.

The event-handler-path attribute is created by rmstbladm from default entries in the table definition file but can be modified by an administrator. The rmshost, version, modify-time and osarch attributes are created by the rmsbuild program. The altzone attribute should be created by an administrator using rcontrol if it is required.

The default value of the event-handler-path attribute is /usr/local/rms/etc:/usr/opt/lsf/etc.

By default, RMS processes connect or bind to the IP address of the primary hostname of each node; for example, the IP address of atlas0. The rms-hostname-suffix attribute can be used to modify the hostname used, selecting a different IP address. For example, if the rms-hostname-suffix is set to -eip0 then RMS will use the IP

address of `atlas0-eip0`.

Performance Statistics Attributes

The performance statistics attributes shown in Table 11.8 control the collection of performance statistics. The statistics are collected by `rmsd` at the intervals given in this table. In the current release, only CPU statistics are gathered.

Table 11.8: Performance Statistics Attributes

Attribute	Default	Description
`cpu-stats-poll-interval`	`30`	time in seconds between CPU samples
`fsys-stats-multiplier`	`10`	Controls the number of CPU statistics samples between filesystem statistics samples. The default is to collect filesystem statistics one time in 10.
`stats-direct`	`0`	Performance statistics and status updates are routed direct to the Machine Manager if this attribute is set to 1.

The `cpu-stats-poll-interval` attribute is created by `rmstbladm` from a default entry in the table definition file.

Server Attributes

The server attributes in Table 11.9 control the behavior of the RMS daemons. The modification times are in UTC. Client applications should convert this to local time and print it as a string.

Table 11.9: Server Attributes

Attribute	Default	Description
`rms-poll-interval`	`30`	polling interval for all RMS daemons that have not been allocated a value of their own. Values in the range 30 seconds (the default) to 120 seconds are supported.

(continued on next page)

Table 11.9: Server Attributes (cont.)

Attribute	Default	Description
`swmgr-poll-interval`	`30`	polling interval for the RMS switch manager. Values in the range 30 seconds (the default) to 120 seconds are supported.
name`-poll-interval`	`30`	polling interval for an individual RMS server.
`status-modify-time`	`0`	last time the status changed
`resource-modify-time`	`0`	last time a resource was modified
`linkerror-event-interval`	`1800`	the time in seconds between link error events generated for a switch module.

The server attributes are all created by `rmstbladm` from default entries in the table definition file. The attribute `linkerror-event-interval` controls the rate at which link error events are posted by the `swmgr`

The `rms-poll-interval` attribute controls the default rate at which RMS servers perform their periodic housekeeping tasks. This rate can modified for individual servers by creating an attribute *name*`-poll-interval` where *name*is replaced by the name of the server. For example the attribute `pmanager-parallel-poll-interval` controls the rate at which the Partition Manager for the partition `parallel` calls its housekeeping function.

Note

As a guide the RMS poll intervals should be set to values in the range `30` to `120` seconds. It should not be necessary to set higher values.

Parallel Processing Attributes

The attributes in Table 11.10 control the scheduling of parallel programs.

Table 11.10: Scheduling Attributes

Attribute	Default	Description
`default-partition`	`parallel`	the partition used by default for parallel programs
`default-priority`	`50`	the default scheduling priority

(continued on next page)

Table 11.10: Scheduling Attributes (cont.)

Attribute	Default	Description
grace-period	60	the time allowed in seconds for a parallel program to exit after a CPU time signal
lbal-partition	login	the default partition for load balancing requests
exit-timeout		default exit timeout (absent by default)
pmanager-queuedepth	0	maximum number of queued requests
pmanager-idletimeout	0	number of seconds an allocated resource may remain idle
warning-signal	SIGTERM	signal to tell jobs to quit
rms-keep-core	1	keep (1) or remove (0) core files
local-corepath	/local/core/rms	directory path for core files
local-filesystems	/tmp /local /tmp1 local1	local file systems
local-rmsdir	/usr/local/rms/etc	directory path for site specific scripts
auto-configure-out	1	automatically configure out nodes which are not responding (1) or not (0)
configure-out-active	1	automatically configure out nodes which are active (1) or not (0)
max-alloc	3400	maximum amount of memory which can be allocated (per node) in MBytes
max-alloc8	7200	maximum amount of memory which can be allocated (per node) in MBytes on a node with 8GB of memory
default-railmask	3	The default rail mask for parallel programs that do not specify the number of rails they require.

The auto-configure-out, configure-out-active, default-partition, default-priority, default-railmask, grace-period, pmanager-idletimeout, pmanager-queuedepth, rms-keep-core, and warning-signal attributes are created by rmstbladm from default entries in the table definition file.

The values of the auto-configure-out and configure-out-active attributes control the behaviour of the pmanager when nodes become active or not responding, causing a partition to become blocked. If both attributes are set the pmanager will automatically configure out active nodes. If auto-configure-out is set but configure-out-active is not set only nodes which are not responding will be configured out automatically. If neither attribute is set the partition will remain blocked until the problem is resolved.

See Chapter 6 (*Access Control, Usage Limits and Accounting*)) for more information on default-priority and Page 5-27 for more information on grace-period and warning-signal.

If the number of resource requests reaches pmanager-queuedepth, subsequent requests either block or fail immediately (if the immediate option to prun has been selected). The blocked requests do not appear in the database.

If a resource request is idle for pmanager-idletimeout secs then resource allocation times out (see Page 5-3) with an exit status of 125. This behaviour is disabled if the attribute's value is 0 (the default).

An adminstrator can create or set the max-alloc, max-alloc8, exit-timeout, lbal-partition, rms-keep-core, local-corepath, local-filesystems and local-rmsdir attributes using rcontrol (see Page 5-27).

See section Page 5-12 for more information about exit-timeout, rms-keep-core and local-corepath, and Page 5-59 for more information about lbal-partition.

RMS will look for site specific partition startup, core file analysis, prologue and epilogue scripts in local-rmsdir. See Section 9.5 for more information about these scripts.

The pmanager creates the max-alloc attribute if it is not present (see Section 4.4).

In a multirail system the pmanager uses the default-railmask attribute to select the rails to use for a program that does not specify its own requirements. Set bit 1«r in the mask to select rail r. Valid values of the rail identifier r run from 0 to R-1 where R is the number of rails in the system.

11.5.5 The Backups Table

The backups table, shown in Table 11.11, contains one entry for each archive or backup file created by rmsarchive (see Page 5-49) or rmsbackup (see Page 5-52). The records are created by rmstbladm (see Page 5-66) when used with either the -a or -b option.

Table 11.11: Backups Table

Field	Type	Default	Description
name	char(16)	""NN	type of operation (archive or backup)
status	char(16)	""	status of the operation
ctime	UTC	0NN	time the archive (or backup) file was created
handle	int	0NN	unique ID for the archive (or backup)
filename	char(256)	""	name of the archive or backup file
archived	int	0	number of records output to file

The handle field provides a unique index into the table. It is generated automatically.

The status field is set to either complete or error.

11.5.6 The Elans Table

The elans table, shown in Table 11.12, contains one entry for each Elan network adapter connected to the HP AlphaServer SC Interconnect. Entries are created by SC Viewer and maintained by the rmsd running on the node that contains the Elan.

Table 11.12: Elans Table

Field	Type	Default	Description
name	char(8)	""NN	unique identifier for the adapter
hostname	char(16)	""NN	name of node containing the adapter
rail	int	0NN	rail number (0-31)
type	int	3NN	The type of Elan, 3 for an Elan3 or 4 for an Elan4
netid	int	0NN	network address within the rail
level	int	null	Position.NodeLevel from Elan devinfo struct
revision	int	-	chip revision level
ecount	int	0NN	error count for the last sample
ecount10	int	0NN	error count for the last 10 samples
status	char(8)	unknownNN	Elan status
linkstate	char(2)	U	state of the link
linkerrors	text(32)	" "	description of errors in last 10 samples

The name and rail fields are combined to provide a unique index into the table.

The status field holds one of the values described in Section B.2. The linkstate field holds one of the values in Table B.2.

The level field records the Position.NodeLevel from the Elan devinfo structure. Supported values are null (no value can be obtained), 2 (16-way switch), 3 (128-way

switch), 4 (256-way switch), 5 (512-way and 1024-way) and 6 (4096-way switch).

Entries in the `linkerrors` field give the ID of the link and then, in brackets, a vector of counts for each of the supported error types (see Section B.4).

11.5.7 The Elites Table

The `elites` table, shown in Table 11.13, contains one entry for each switch in the network. Its entries are created by SC Viewer and maintained by the Switch Network Manager, `swmgr` (see Section 4.5).

Table 11.13: Elites Table

Field	Type	Default	Description
`name`	char(16)	`""`^{NN}	a unique ID for each switch
`rail`	int	`0`^{NN}	rail number (0-31)
`level`	int	`0`^{NN}	level number (0-3)
`netid`	int	`0`^{NN}	network address in rail (0-1023)
`routeUp`	int	`""`^{NN}	link (up) used for broadcast tree (4-7).
`bcastTop`	int	`3`^{NN}	broadcasts are sent (down) on links 0 - `bcastTop`
`plane`	int	`0`^{NN}	plane number (0-63)
`board`	char(8)	`""`^{NN}	name of board containing the switch
`chip`	int	`0`^{NN}	chip number on the board (0-7)
`revision`	int	`1`	chip revision number
`status`	char(8)	`unknown`^{NN}	Elite status
`linkstate`	char(8)	`UUUUUUUU`	state of each link
`linkerrors`	text(32)	`" "`	description of errors in last 10 samples

The `name` and `rail` fields are combined to provide a unique index into the table.

The status field holds one of the values in Section B.2.

The `linkstate` field contains a character for each of the 8 links. Each link can be in one of the states shown in Table B.2.

Entries in the `linkerrors` field give the ID of the link and then, in brackets, a vector of counts for each of the supported error types (for more details on HP AlphaServer SC Interconnect terms see Appendix A (*HP AlphaServer SC Interconnect Terms*)).

11.5.8 The Event Handlers Table

The `event_handlers` table, shown in Table 11.14, defines which handler scripts are run in response to events. Event handling is discussed in detail in Chapter 8 (*Event Handling*).

Table 11.14: Event Handlers Table

Field	Type	Default	Description
id	int	0 NN	unique identifier for handler
name	char(16)	"" NN	name of object that has changed state
class	char(16)	"" NN	class of the object, such as node or partition
type	char(16)	"" NN	type of event
timeout	int	-1 NN	time limit in seconds for the handler to complete
handler	text(32)	default_handler NN	name of handler script

The id field provides a unique index into the table and is generated automatically.

By default, the event_handlers table is created with the entries shown in Table 11.15.

Table 11.15: Default Entries in the Event Handlers Table

id	name	class	type	timeout	handler
1	null	node	status	600	/usr/opt/rms/etc/rmsevent_node
2	null	node	temphigh	300	/usr/opt/rms/etc/rmsevent_env
3	null	node	tempwarn	300	/usr/opt/rms/etc/rmsevent_env
4	null	node	fan	300	/usr/opt/rms/etc/rmsevent_env
5	null	node	psu	300	/usr/opt/rms/etc/rmsevent_env
6	null	event	escalation	-1	/usr/opt/rms/etc/rmsevent_escalate
7	null	switch...	null	300	/usr/opt/rms/etc/rmsevent_qsnet

11.5.9 The Events Table

Entries are added to the events table, shown in Table 11.16, each time an object managed by the RMS changes state, for example, when a node status changes, a partition starts or a component fails.

Table 11.16: Events Table

Field	Type	Default	Description
id	int	0 NN	unique identifier for each event

(continued on next page)

Table 11.16: Events Table (cont.)

Field	Type	Default	Description
name	char(32)	""NN	name of object that has changed state
class	char(16)	""NN	class of object, such as node or partition
type	char(32)	""NN	type of event
ctime	UTC	0NN	time at which the event occurred
handled	int	0NN	whether event was handled (1) or not (0)
description	text (32)	""	description of the event

The id provides a unique index into the table. It is generated automatically.

Table 11.17 shows three typical events. The first shows a node status change as RMS starts on node cfs1, the second shows a temperature change on module mod2 and the third shows the partition parallel starting.

Table 11.17: Example of Status Changes

name	class	ctime	type	description
cfs1	node	893427468	status	running
mod2	module	894991521	temperature	ambient=15
parallel	partition	894991490	status	running

The events table can grow rapidly. Run the archive utility, rmsarchive, to remove old entries. This should be done periodically using a cron(8) script. See Page 5-49 for details. The lifetime attribute in the archive_tables table (see Section 11.5.3) determines how old the entries must be before they are removed.

Events are discussed in detail in Chapter 8 (*Event Handling*).

11.5.10 The Fields Table

The fields table, shown in Table 11.18, defines which RMS objects and attributes can be created and modified using rcontrol (see Page 5-27), identifying them by a table name and field name within that table.

Table 11.18: Fields Table

Field	Type	Default	Description
name	char(16)	""NN	name of the field
tablename	char(16)	""NN	name of the table

(continued on next page)

Table 11.18: Fields Table (cont.)

Field	Type	Default	Description
access	char(8)	""NN	currently unused; always set to admin
type	char(16)	""	defines the type of value
rangemin	int	-	minimum value
rangemax	int	-	maximum value
textattr	text(32)	""	specifies how values are validated

The name and tablename fields are combined to provide a unique index into the table.

The value of the type field determines how rcontrol checks the validity of values entered by an administrator. The type field may hold one of the values shown in Table 11.19.

Table 11.19: Type Values

Value	Description
null	no checking
selectedtext	textattr gives a comma-separated list of valid values
integer	entry must be in range bounded by rangemin and rangemax
relation	textattr gives a tablename.fieldname pair; entry must be a value of fieldname in tablename

Values in the attributes table are not checked using this method; the valid values for attributes are built into rcontrol. Minimum and maximum values for various fields are defined in rms/rmsdefs.h in /usr/opt/rms/include.

11.5.11 The Installed Components Table

The installed_components table, shown in Table 11.20, contains information about software components installed on each node.

Table 11.20: Installed Components Table

Field	Type	Default	Description
hostname	char(16)	""NN	hostname of the node on which the component is installed

(continued on next page)

Table 11.20: Installed Components Table (cont.)

Field	Type	Default	Description
product	char(16)	""NN	name of the software product to which the component belongs
prodversion	char(32)	""NN	version of the software product to which the component belongs
component	char(32)	""NN	name of the component
compversion	char(32)	""NN	version of the component
ctime	UTC	0NN	time the component was installed

11.5.12 The Jobs Table

The jobs table, shown in Table 11.21, contains one entry for each parallel job. The entries are maintained by the Partition Manager, pmanager (see Section 4.4).

The jobs table can grow rapidly. Run the archive utility, rmsarchive to remove old entries. This should be done periodically using a cron(8) script. See Page 5-49 for details. The lifetime attribute in the archive_tables table (see Section 11.5.3) determines how old the entries must be before they are removed.

Table 11.21: Jobs Table

Field	Type	Default	Description
name	char(16)	""NN	unique identifier for each job
resource	char(16)	""NN	name of resource on which job is running
status	char(16)	""NN	status of the job
cpus	text(32)	""	list of CPUs allocated to this job
nodes	text(32)	""	list of nodes allocated to this job
hostnames	text(32)	""	list of hostnames allocated to this job
startTime	UTC	0	time the job started
endTime	UTC	0	time the job completed
contexts	char(16)	""	range of Elan contexts allocated to the job
exitStatus	int	0	exit status of the job
railmask	int	1	mask identifying the rails in use by a job
flags	int	""	the partition manager's flags for a job
session	int	-	UNIX session ID of the allocating process
cmd	text(128)	""NN	command being executed

The job name provides a unique index into the table. It is generated automatically.

The status field holds one of the values shown in Table B.1.

While the job is running, endTime is set to the time by which the job must end if there is a timelimit on the partition or the job. If there is no time limit, endTime is set to 0. Finally, endTime is updated to show the time at which the job completed.

The nodes and cpus fields contain lists of node and CPU numbers in use by a job. Each pair of values defines a cpus x nodes box allocated to the job. The total number of CPUs allocated is the sum of the area of these boxes. See also Section 2.4.2.

A command name, cmd, passed to prun, may be up to MAXPATHLEN characters in length. In the jobs table, the command name is truncated to 32 characters, including three dots (...) appended to the name to show that it has been truncated.

11.5.13 The Link Errors Table

The link_errors table, shown in Table 11.22, contains one entry for each link error detected by the Switch Network Manager, swmgr (see Section 4.5).

Table 11.22: Link Errors Table

Field	Type	Default	Description
id	int	0	unique identifier for each error
name	char(16)	""NN	name of the chip detecting the fault
class	char(16)	""NN	type of chip detecting fault (elan, elite)
ctime	UTC	0NN	time at which the error was detected
mtime	UTC	0NN	last modification time of record
current	int	0NN	link error has occurred recently
description	text(32)	""NN	description of the error

The id provides a unique index into the table. It is generated automatically. The link_errors table is also indexed using the name field.

Entries in this table are updated by the swmgr. High error counts in the description field indicate that an error is persistent. Increasing counts indicate that it is current.

If the swmgr has received a recent report of a link error then the current field is set to 1. This field is set to 0 if no further errors are reported within the *link error lifetime* (one hour).

Entries in the description field give the ID of the link and then, in brackets, a vector of counts for each of the supported error types. HP AlphaServer SC Interconnect link errors are described in the *HP AlphaServer SC Interconnect Installation and Diagnostics Manual*.

11.5.14 The Module Types Table

The module_types table, shown in Table 11.23, contains descriptions of each of the module types supported in a given release of the RMS. It is updated by the table administration program, rmstbladm (see Page 5-66), when a new release is installed.

Table 11.23: Module Types Table

Field	Type	Default	Description
name	char(16)	""NN	name of the module type
class	char(16)	""NN	class of module (node, network, storage)
units	int	0NN	height of the module in units
cpus	int	0NN	number of CPUs in the module
psus	int	0NN	number of PSUs in the module
fans	int	0NN	number of fans in the module
thermistors	int	0NN	number of thermistors in the module
vendor	char(16)	"";	vendor
description	text(32)	""	description of the module

The module types supported in the current release are shown in Table 11.24. The thermistors field name has been abbreviated to therm.

Table 11.24: Valid Module Types

name	class	units	cpus	fans	psus	therm	description
AS4100	node	6	4	1	1	1	HP AlphaServer 4100
DS20	node	12	2	1	1	1	HP AlphaServer DS20
ES40	node	8	4	6	3	7	HP AlphaServer ES40 - rack mounted
ES45	node	8	4	6	3	7	HP AlphaServer ES45 - rack mounted
DS20L	node	1	2	11	1	5	HP AlphaServer DS20L - rack mounted
UP2000	node	3	2	1	1	1	Alpha Processor UP2000
PC	node	2	2	0	0	0	Dual Processor PC
IA32	node	0	1	0	1	0	Generic IA32 node
IA64	node	0	1	0	1	0	Generic IA64 node
Alpha	node	0	1	0	1	0	Generic Alpha node

(continued on next page)

Table 11.24: Valid Module Types (cont.)

name	class	units	cpus	fans	psus	therm	description
RX2600	node	0	2	4	2	0	Dual Processor Itanium 2 (McKinley
QMS16	network	4	0	0	1	1	QM-S16 switch network
QMS128	network	17	0	6	2	24	QM-S128 switch network
QMS64U64D	network	17	0	6	2	24	64-way federated node switch
QMS32U96D	network	17	0	6	2	24	96-way federated node switch
QMS16X8	network	17	0	6	2	24	128-way federated top switch
QMS16U16DX4	network	17	0	6	2	24	64-way federated mid switch
DEC900	tserver	2	0	0	0	0	DecServer 900 Terminal Server

See Appendix A (*HP AlphaServer SC Interconnect Terms*) for more details of the network modules.

11.5.15 The Nodes Table

The `nodes` table, shown in Table 11.25, contains configuration information on each node in a machine. The entries are created by the RMS clients `rmsbuild` and `rcontrol`. Fields are updated by `rmsd` when the node is booted or RMS is restarted and by the Machine Manager, `mmanager`, when the node's status or run level changes.

Table 11.25: Nodes Table

Field	Type	Default	Description
name	char(16)	""NN	the name of the node
type	char(8)	ES40	node type, such as ES40
maxmem	int	0	maximum MBytes memory available
maxfree	int	0	maximum MBytes free memory available
pagesize	int	8192	the page size of the node

(continued on next page)

Table 11.25: Nodes Table (cont.)

Field	Type	Default	Description
level	int	""	the network level of the node
drain	int	0	job draining in progress
maxswap	int	0	maximum MBytes swap space available
maxtmp	int	0	temporary file system space in MBytes
cpus	int	1	number of CPUs
cpus_reserved	int	0	number of CPUs reserved for OS services
elans	int	1	bitmap of Elan devices present
netid	int	-	physical network ID (if applicable)
configured	int	0	node is configured in (1) or out (0)
status	char(16)	unknown	current node status
runlevel	char(16)	unknown	UNIX run level
boot_time	UTC	-	time when node was last booted
swap_eager	int	1	swap allocation is lazy(0) or eager(1)
console	char(32)	""	command line to connect to console
ipaddr	char(20)	""	IP address
macaddr	char(20)	""	Ethernet address
elanipaddr	char(20)	""	Elan IP address
cluster	char(20)	""	cluster name
member	int	-	cluster member ID
votes	int	0	does cluster member vote (1) or not (0)
rack	int	-	ID of rack that contains the node
unit	int	-	location of the node in the rack
temp	int	0	bitmap of the functioning thermistors
psus	int	0	bitmap of the functioning power supply units
fans	int	0	bitmap of the functioning fans
estatus	char(16)	""	environmental status of the node
environment	text(32)	""	description of the environmental status
reason	text (32)	""	reason why node is configured out

The name field provides a unique index into the table.

The type field takes a value from the module_types table (see Section 11.5.14). It is set by default to DEFAULT_NODETYPE as defined in rms/rmsdefs.h in /usr/opt/rms/include.

The cpus_reserved field specifies the number of CPUs that are unavailable for running parallel programs. These CPUs are reserved for running system services.

The valid range of values for the cpus, cpus_reserved and netid fields are given in rmsdefs.h in /usr/opt/rms/include.

The configured field indicates whether a node is active (1) or configured out for repair or upgrade (0).

The status field indicates the service level being provided by a node. Valid values are shown in Table B.5. State changes are logged in the events table (see Section 11.5.9); entries are keyed by class=node.

The level field records the Position.NodeLevel from the Elan devinfo structure. Supported values are null (no value can be obtained), 2 (16-way switch), 3 (128-way switch), 4 (256-way switch), 5 (512-way and 1024-way) and 6 (4096-way switch). All nodes should report the same level.

The runlevel may have one of the values shown in Table B.6.

The elans field is a bitmap of the Elan devices present in the node. It has one bit set for each device. In previous releases, only device 0 was supported.

The psus and fans fields are bitmaps; their width is controlled by the corresponding values in the module_types table (see Section 11.5.14). Similarly, the temp field maps onto the thermistors field of the module_types table.

rmsd collects environmental data from the kernel on each node. The operational status of the cooling fans and power supplies is logged along with the temperature status of vital system components. This is used to generate an environment status, estatus, and an environment string, environment. The environment status can take one of the values shown in Table B.4.

If the environment status is recorded as ok, the environment string contains temperature readings from the CPU, power supply unit and enclosure. If a node has more than one instance of each type of temperature sensor, the maximum of their values is recorded.

Temperature information is recorded as a list of attribute-value pairs, for example:

```
ambient=15 cpu=40 psu=20
```

If an error occurs, the environment string contains details of what has failed, for example, the following string indicates that the CPU fan number 1 has failed on the node.

```
cpu fan 1 failure
```

11.5.16 The Node Statistics Table

The node statistics (`node_stats`) table, shown in Table 11.26, contains performance statistics collected periodically by the `rmsd` daemon running on each node.

The interval at which the nodes are sampled for CPU statistics is controlled by the attribute `cpu-stats-poll-interval` in the `attributes` table; the default is to sample every 2 minutes.

Table 11.26: Node Statistics Table

Name	Type	Default	Description
`name`	char(16)	""NN	name of the node
`ctime`	UTC	0	when sample was collected (see notes)
`usercpu`	%	0	user mode CPU time since last sample
`syscpu`	%	0	kernel mode CPU time since last sample
`idlecpu`	%	0	idle CPU time since last sample
`freemem`	int	48	free memory in megabytes
`ubc`	int	0	size of the unified buffer cache in megabytes
`wired`	int	0	memory in use in megabytes
`freeswap`	int	48	free swap space in megabytes
`pages`	int	0	page fault rate
`interrupts`	int	0	interrupts rate (except clock)
`syscalls`	int	0	system call rate per second
`users`	int	0	number of users logged in
`freetmp`	int	0	free temporary file space in megabytes
`cpu0`	%	0	time spent in user and system mode on CPU 0
`cpu1`	%	0	time spent in user and system mode on CPU 1
`cpu2`	%	0	time spent in user and system mode on CPU 2
`cpu3`	%	0	time spent in user and system mode on CPU 3
`s_inuse`	int	0	swap space in use in megabytes
`s_reserved`	int	0	swap space in use in megabytes
`p_in`	int	0	number of pages per second swapped in
`p_out`	int	0	number of pages per second swapped out
`m_active`	int	0	memory (in MB) in use
`m_inactive`	int	0	memory (in MB) allocated
`rql5`	real	0	length of 5-second run queue
`rql30`	real	0	length of 30-second run queue
`rql60`	real	0	length of 60-second run queue

(continued on next page)

Table 11.26: Node Statistics Table (cont.)

Name	Type	Default	Description
tmp_usage	%	0	percentage of /tmp in use
local_usage	int	0	percentage of local file system used

The name field provides a unique index into the table.

The ctime field indicates when the record was last updated. Two values have special meanings:

0	Indicates that for test purposes the data should be processed as current.
-1	Indicates that the record has never been updated.

The usercpu and syscpu statistics are percentages of CPU time spent in the given mode calculated over the period since the last sample and averaged over all CPUs.

The freemem field shows the amount of free memory in clean virtual memory pages that is available for use. The freeswap field shows the amount of swap space actually usable for swapping. The wired field shows the amount of memory that is in use and not available for paging.

The interrupts, pages and syscalls statistics are rates per second averaged over the interval since the last sample.

The four CPU fields show the percentage of time spent executing instructions in user and kernel mode on each of four CPUs. Valid values range between 0 and 100. If a given CPU is not present, the value of the field is set to Null.

s_inuse shows the amount of megabytes of swap space currently in use. s_reserved shows the amount reserved for use as swap space.

m_active shows the total amount in megabytes of virtual memory pages that are currently in use but that can be used for paging. m_inactive shows the total amount in megabytes of virtual memory pages that are allocated but are most likely to be used for paging.

If the tmp_usage field reaches 100, this indicates that the file system is full. The local_usage field relates to the fullest of the local file systems. The local file systems are specified using the local-filesystems attribute (Table 11.10).

11.5.17 The Partitions Table

The nodes in a machine are grouped into partitions according to their function. For example, there may be an administrative partition, a login partition and a parallel

programming partition. A set of partitions spanning the machine is called a *configuration*. Different configurations may be appropriate to different times of the day or week. For example, one for daytime running and another for nights and weekends. Only one configuration, the *active configuration*, can be running at a time.

The `partitions` table, shown in Table 11.27, describes how nodes are allocated to partitions in each of the configurations. It also contains scheduling parameters (see also Section 7.3) for each partition.

The entries in the `partitions` table are created by `rcontrol`. The information is updated by the Partition Manager, `pmanager`, as it starts.

Table 11.27: Partitions Table

Field	Type	Default	Description
`name`	char(16)	""NN	name of the partition, such as `par`
`configuration`	char(16)	""NN	name of the configuration, such as `day`
`nodes`	text(32)	""NN	list of nodes in the partition
`configured_nodes`	text(32)	""	list of nodes configured into the partition
`cpus`	int	`0`	number of CPUs configured in
`freecpus`	int	`0`	number of free CPUs
`active`	int	`0`	whether partition is active (1) or not (0)
`startTime`	UTC	`0`	time partition was last started
`status`	char(16)	`down`	status of the partition
`timelimit`	int	`0`	time limit in seconds for a parallel job
`type`	char(16)	`general`	partition type (`parallel`, `login`, `general`, `batch`, `restricted`)
`timeslice`	int	`0`	time slice interval in seconds
`mincpus`	int	`0`	minimum number of CPUs that can be allocated
`memlimit`	int	`-`	default memory limit in megabytes
`rms_alloc`	int	`0`	restrict resource allocation to administrative users only
`use_elan`	int	`1`	use Elans in this partition.

Partition names do not have to be unique but the combination of a partition and a configuration name must provide a unique index into the table. For example, there may be a partition named `login` in two different configurations, each with a different set of values in the `partitions` table.

Valid values for the `status` of the partition are shown in Table B.7.

The `type` field controls how jobs are scheduled on the partition (see also Section 7.2). If the partition type is `parallel` then it is exclusively for gang-scheduled parallel programs. Partitions of type `login` support interactive logins and load-balanced sequential program execution. Partitions of type `general` support login shells, load-balanced sequential program execution and parallel programs. Partitions of type `batch` are under the exclusive control of a batch system. The batch system can use them for sequential or parallel jobs but interactive use is prohibited. Partitions of type `restricted` are scheduled by an external scheduler rather than the RMS Partition Manager and jobs can only be run on them by the `root` user.

The `freecpus` field stores the count of the number of CPUs available in the partition. It is updated by the `pmanager` each time CPUs are allocated or freed.

The `configured_nodes` field stores the subset of nodes that were configured in when the partition was started.

The `timeslice` field stores the interval in seconds between periodic rescheduling of parallel jobs. Time slicing is disabled when this field is 0, the default.

The `timelimit` field stores the maximum interval in seconds for which CPUs in a partition may remain allocated. Time limits are disabled when this field is 0, the default.

The `memlimit` field stores the default memory limit in megabytes per CPU for jobs running on this partition.

If the `rms-alloc` attribute is set to `1` then only members of the `rms` group can allocate CPUs. The default value is `0`. This feature is intended for batch systems requiring exclusive use of a partition. The batch scheduler should (a) be run under the user id of a member of the `rms` and (b) make calls to rms_allocateResource or rms_createResource passing the uid of the user on whose behalf the CPUs are being allocated.

11.5.18 The Projects Table

The `projects` table, shown in Table 11.28, lists all of the projects that have been defined. A project is a list of users. Membership of the project is specified in the `projects` field of the `users` table (see Section 11.5.32). All accounting records include the project to which a user's job is being billed (see Table 11.2).

Table 11.28: Projects Table

Field	Type	Default	Description
name	char(16)	""NN	project name
description	text(32)	""	label describing the project

The name field provides a unique index into the table.

11.5.19 The Resources Table

The resources table, shown in Table 11.29, contains one entry for each current resource request. The entries in this table are maintained by the Partition Manager, pmanager (see Section 4.4).

The resources table can grow rapidly. Run the archive utility, rmsarchive, to remove old entries. This should be done periodically using a cron(8) script. See Page 5-49 for details. The lifetime attribute in the archive_tables table (see Section 11.5.3) determines how old the entries must be before they are removed.

Table 11.29: Resources Table

Field	Type	Default	Description
name	char(16)	""NN	resource name
partition	char(16)	""NN	partition name
username	char(16)	unknownNN	name of the user
hostnames	text(32)	""	list of hostnames allocated
status	char(16)	""NN	status of the resource
cpus	text(32)	""	list of CPUs allocated
nodes	text(32)	""	list of nodes allocated
startTime	UTC	0	time resources were allocated
endTime	UTC	0	time resources were deallocated
priority	int	0	current priority of the request
flags	int	0	scheduler flags for the resource
ncpus	int	-	number of cpus allocated
batchid	char(16)	""	batch system identifier for the request
memlimit	int	-	memory allocated per CPU in megabytes
project	char(16)	""NN	name of project associated with resource

(continued on next page)

Table 11.29: Resources Table (cont.)

Field	Type	Default	Description
pid	int	-	pid of allocating process
allocated	int	-	CPUs allocated (0 | 1)

The name provides a unique index into the table. It is generated automatically. The resources table is also indexed on the batchId field.

The hostnames field lists the names of the nodes allocated to this request.

Valid values for the status field are given in Table B.8.

The cpus and nodes fields contain lists of CPU and node numbers in use by a job. Each pair of values defines a cpus x nodes box allocated to the job. The total number of CPUs allocated is the sum of the area of these boxes.

While the resource is allocated endTime is set to the time at which the resource will be deallocated if there is a timelimit on the partition or the resource. If there is no time limit, endTime is set to 0. Finally, endTime is updated to show the time at which the resource was deallocated.

The batchid field contains the batch system identifier for this request. If the request was made by LSF® then the field contains LSB_JOBID. If the request was made by DPCS then this field contains PSUB_JOBID.

The allocating process whose pid is recorded is either prun or allocate.

11.5.20 The Scheduler Log Table

The scheduler_log table, shown in Table 11.30, records scheduling decisions made by the Partition Manager. It is only used when the logging of resource debugging output is enabled (see Section 10.11). The Partition Manager writes a record to the table when logging is enabled and again when it is disabled.

Table 11.30: Scheduler Log Table

Field	Type	Default	Description
id	int	0^{NN}	a unique ID
resource	char(16)	""^{NN}	name of the resource
parent	char(16)	""	parent of the resource
status	char(16)	""	status of the resource
ctime	UTC	0^{NN}	time the record was added

The id field provides a unique index into the table. It is generated automatically.

The `parent` field is only used when an existing resource has requested (and been granted) an allocation of additional resources. It records the name of the existing (parent) resource (see Appendix D (*RMS Application Interface*)).

Valid values for the `status` field are described in Section B.9.

11.5.21 The Servers Table

The `servers` table, shown in Table 11.31, contains one entry for each RMS daemon. The table administration program, `rmstbladm` (see Page 5-66), creates some default entries in the table and the others are created or updated by the RMS daemons when they start up.

Table 11.31: Servers Table

Field	Type	Default	Description
`name`	char(32)	`""`NN	server (daemon) name
`hostname`	char(16)	`rmshost`	host on which the daemon is running
`port`	int	`-1`	IP port number to bind to for this server
`pid`	int	`-1`	process ID of the server
`rms`	int	`1`	is daemon an RMS server (1) or not (0)
`startTime`	int	`0`	time at which the daemon was started
`autostart`	int	`1`	daemon starts automatically (1) or not (0)
`status`	char(8)	`ok`NN	server status
`args`	char(32)	`""`	site-specific arguments for the server

The `name` and `hostname` fields are combined to provide a unique index into the table.

The `hostname` field contains the name of the node on which the daemon should run. It is set to `rmshost` if the daemon should run on the `rmshost` node.

The `rms` field specifies whether the server is an RMS daemon or a conventional UNIX daemon. This controls the method used to determine whether or not the process is running.

The `autostart` field determines whether a daemon should be restarted automatically if it exits or is killed by a signal.

By default, the `servers` table is created with the entries shown in Table 11.32.

(continued on next page)

Table 11.32: Default Entries in the Servers Table (cont.)

name	hostname	port	rms	args

Table 11.32: Default Entries in the Servers Table

name	hostname	port	rms	args
tlogmgr	rmshost	6200	1	""
eventmgr	rmshost	6201	1	""
mmanager	rmshost	6202	1	""
swmgr	rmshost	6203	1	""
peventproxy	rmshost	6205	0	""

11.5.22 The Services Table

The `services` table, shown in Table 11.33, is an internal table used by RMS to determine the command to execute for each service, the names of the hosts that support the command and which users have permission to use the service. It contains one entry for each of the RMS clients that provides a configuration management service (for example, `rmsquery` and `rcontrol`). The entries are created by the table admininstration program, `rmstbladm`. See Chapter 5 (*RMS Commands*) for details of these services.

Table 11.33: Services Table

Field	Type	Default	Description
name	char(16)	""NN	name of the service
hostname	char(16)	rmshost	host of service, such as rmshost
group	char(8)	rms	group(s) with access to the service
sequential	int	1	wait for cmd to finish (1) or not (0)
cmd	char(64)	""NN	command to execute

The name field provides a unique index into the table.

The hostname field contains the name of the host on which the service should run.

The group field holds the name of the UNIX group which is allowed to run this service. Currently, only rms is valid.

Some services, such as rcontrol, must have exclusive access to the database, requiring that other transactions wait until they complete. The sequential field should be set to 1 for these services. Others such as swctrl may run for long periods of time and should not block the execution of other transactions. sequential should be set to 0 for these services.

By default, the services table is created with the entries shown in Table 11.34.

Table 11.34: Default Entries in the Services Table

name	hostname	group	sequential	cmd
rcontrol	rmshost	rms	1	rcontrol
swctrl	rmshost	rms	1	swctrl
tlogmgr	rmshost	rms	1	
sql	rmshost	rms	1	rmsquery
rundiag	rmshost	rms	0	rundiag
bootpup	rmshost	rms	1	bootpup

11.5.23 The Software Products Table

The software_products table, shown in Table 11.35, contains information about the components that make up a software product. The table contains entries for each subset in each of the installed products defining the components required for a given release.

Table 11.35: Software Products Table

Field	Type	Default	Description
name	char(16)	""NN	name of the product
version	char(32)	""NN	version of the product
component	char(32)	""NN	name of the component
comptype	char(16)	subsetNN	type of component
compversion	char(32)	""NN	version of the component
compattr	text(16)	""	component attributes

The only valid value for the comptype field is subset.

The compattr field currently contains one value which dictates where a software component will be installed. The possible values are shown in Table 11.36.

Table 11.36: Component Attribute Values

Value	Description
opt	Component should only be installed on rmshost
root	Component should be installed on rmshost and the cluster root node(s)

11.5.24 The Status Table

The status table, shown in Table 11.37, provides a summary of the state of a group of objects of a given class (currently, restricted to nodes). There is an entry in the table for each state of the class of objects. See Appendix B (*RMS Status Values*) for details of valid states. The entry contains a string that lists the name of all objects in that state.

Table 11.37: Status Table

Field	Type	Default	Description
name	char(16)	""NN	name of the state
class	char(16)	""NN	class of the object
ctime	UTC	0NN	time when record last updated
names	text(64)	""	list of objects with this status

By default, the status table is created with the entries shown in Table 11.38.

Table 11.38: Default Entries in the Status Table

name	class
STATUS_UNKNOWN	node
STATUS_NOTRESPONDING	node
STATUS_ACTIVE	node
STATUS_RUNNING	node
STATUS_CONFIGUREDOUT	node

11.5.25 The Switch Boards Table

The switch_boards table, shown in Table 11.39, contains one entry for each switch board in the HP AlphaServer SC Interconnect. It is created and maintained by the Switch Network Manager, swmgr (see Section 4.5), on receipt of messages from the swmserver processes responsible for monitoring each switch module.

Table 11.39: Switch Boards Table

Field	Type	Default	Description
name	char(16)	""NN	board name
module	char(16)	""NN	name of module containing the board
rail	int	0	rail number
slot	int	0	slot number in the module
type	char(16)	""NN	board type (QM401 or QM402)
status	char(8)	""NN	board status
temp	int	0	temperature data from thermistors on board
PN	char(8)	""	
FN	char(8)	""	
EC	char(8)	""	
SN	char(8)	""	
MN	char(8)	""	

The name and rail fields are combined to provide a unique index into the table.

Valid values for the status of a switch board are shown in Section B.5.

The valid range of values for the rail field is given in rmsdefs.h in /usr/opt/rms/include.

The slot field can take the values 0–7, 16–31 or 61.

11.5.26 The Switch Modules Table

The switch_modules table, shown in Table 11.39, contains one entry for each switch module in the HP AlphaServer SC Interconnect. It is created by SC Viewer when the network is set up. It is maintained by SC Viewer and the Switch Network Manager, swmgr (see Section 4.5) on receipt of messages from the swmserver processes responsible for monitoring each module.

Table 11.40: Switch Modules Table

Field	Type	Default	Description
name	char(32)	""NN	module name, such as QR0N01
type	char(16)	""NN	type of module, such as QMS6464D or QMS16X8
rail	int	0	rail number
level	int	0	level in the switch network
rack	int	0	rack number
unit	int	0	location in rack
psus	int	1	bitmap showing status of PSUs
fans	int	0	bitmap showing status of fans
temp	int	0	maximum board temperature
status	char(16)	""	module status
version	int	0	version of swmserver protocol
qnx	int	0	does module have QNX controller (1) or not (0)
qnxstats	char(32)	""	performance statistics from QNX controller
ipaddr	char(20)	""	IP address of module
macaddr	char(20)	""	Ethernet address of QNX controller
started	int	0	time swmserver started
ctime	int	0	time record was last updated
build	int	0	software build number of QNX controller
clksrc	char(8)	local	clock source for module (A, B or local)
clkfreq	int	400	clock frequency for module
location	text(32)	""	description of location

The name field is a unique index on the switch_modules table.

The psus and fans fields are bitmaps with one bit for each PSU or fan. The width of the bitmaps is determined by the corresponding entry in the module_types table (see Table 11.23).

Valid values for the status of a switch module are shown in Section B.5.

11.5.27 The Terminal Servers Table

The tservers table, shown in Table 11.41, contains information on each terminal server in the machine.

Table 11.41: Terminal Servers Table

Field	Type	Default	Description
name	char(16)	""NN	name of the terminal server
type	char(16)	""NN	type of the terminal server
baseport	int	0NN	base port number used
numports	int	0NN	number of ports used
ipaddr	char(20)	""	IP address of the terminal server
macaddr	char(20)	""	Ethernet address of the terminal server
basenetid	int	0NN	network ID of node attached to base port

Each type of terminal server must have an entry in the tserver_types table (see Table 11.42).

11.5.28 The Terminal Server Types Table

The tserver_types table, shown in Table 11.42, contains details about each type of terminal server.

Table 11.42: Terminal Server Types Table

Field	Type	Default	Description
name	char(16)	""NN	name of the terminal server type
adminport	int	0NN	administrative port number
baseport	int	0NN	base port number
numports	int	0NN	number of ports
description	text(32)	""	description of the terminal server

By default, the tserver_types table is created with the entry shown in Table 11.43.

Table 11.43: Default Entries in the Terminal Server Types Table

name	adminport	baseport	numports	description
DEC900	1024	2001	32	DecServer 900

11.5.29 The Transactions Table

Changes to the state of the machine are made through a request entered in the `transactions` table shown in Table 11.44. This table records who made each change, when it was made and whether or not the operation was successful.

The Transaction Log Manager, `tlogmgr` (see Section 4.6), actions requests in the `transactions` table, running commands on the user's behalf (in practice, the user here is a system administrator). This mechanism provides an audit trail and sequential ordering of changes in state.

Table 11.44: Transaction Log Table

Field	Type	Default	Description
`name`	char(16)	`""`NN	name of the service
`status`	char(16)	`""`NN	transaction status
`ctime`	UTC	`0`NN	creation time
`mtime`	UTC	`0`	last modification time
`handle`	int	`0`NN	unique identifier for the transaction
`logfile`	char(32)	`""`	`stdout` or `stderr` log for the transaction
`username`	char(16)	`nobody`NN	user issuing the command
`args`	char(256)	`""`	arguments for the command

Valid values for the transaction `status` field are given in Table B.9.

An example of the transaction to add a partition is shown below in Table 11.45.

The `handle` provides a unique index into the table. It is generated automatically and passed to both the service and the client. The service uses the handle to label any output resulting from the transaction; the client uses the handle to select the resulting entries.

If the service fails, the output log (conventionally in the directory `/var/rms/adm/log`) may contain useful diagnostics. Client applications wait for the transaction to complete and then `cat` the logfile.

Table 11.45: Entry in the Transactions Table

name	status	ctime	mtime	handle	logfile
`rcontrol`	`complete`	`1014678011`	`1014678011`	`44`	`tr44.log`

username	args
`rms`	`create partition=login nodes='atlas[0-3]'`

11.5.30 The Tree Status Table

The tree_status table, shown in Table 11.46, describes the state of the tree of sockets connecting the Partition Manager(s) to the RMS daemons on each node (see Section 4.4). It is created and maintained by the Partition Manager and updated by the rmsd.

Table 11.46: Tree Status Table

Field	Type	Default	Description
name	char(24)	""NN	name of the node
partition	char(16)	""NN	name of the partition
status	char(16)	""	status of the node
parent	char(16)	""	parent node in the tree
level	int	0	level in the tree
ctime	UTC	0	time entry was last updated
children	text(32)	""	list of children of node
bmask	char(16)	""	bitmap of connected children
bstatus	text(32)	""	status of each child in bitmap

The name and level fields are combined to provide a unique index into the table.

Valid values for the status field are: new, connecting, ok and error.

11.5.31 The Unidentified MACs Table

The unidentified_macs table, shown in Table 11.47, lists the Ethernet addresses and IP addresses (if assigned) of switch modules that have not been assigned locations in the network. The entries in the table are created by the swmgr and then removed by SC Viewer as modules are bound to their physical locations.

Table 11.47: Unidentified MACs Table

Field	Type	Default	Description
name	char(32)	""NN	Ethernet address
ipaddr	char(16)	""NN	IP address

The name field provides a unique index into the table.

11.5.32 The Users Table

The users table, shown in Table 11.48, contains information on each user's projects.

Table 11.48: Users Table

Field	Type	Default	Description
name	char(16)	""NN	login name
projects	text(32)	""NN	list of the user's projects

The name field provides a unique index into the table.

The projects field may contain a single project name or a comma-separated list of project names. The wildcard, *, may be specified as a project name denoting that the user is a member of all projects.

The ordering of the names in the list is significant: the first project specified is the user's *default* project. For purposes of accounting, access control and scheduling, the default project is assumed unless the user explicitly specifies another project. A project can be specified explicitly by using the environment variable RMS_PROJECT or by using the -P option to prun or allocate.

A

HP AlphaServer SC Interconnect Terms

A.1 Introduction

RMS includes support for programs that use HP AlphaServer SC Interconnect. This appendix provides an introduction to HP AlphaServer SC Interconnect, defining terms used elsewhere in this manual.

A.2 Capabilities

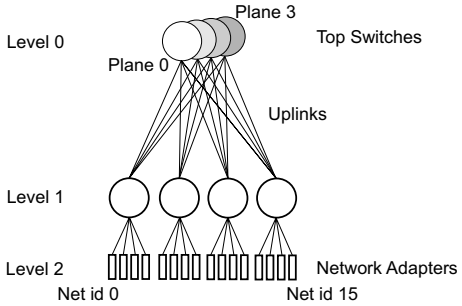
Before an application process can use HP AlphaServer SC Interconnect, it must be given an Elan capability (see Section C.2), describing the nodes and communications contexts that it is allowed to use. In general, processes present this capability to the kernel as they start.

Having granted a request for CPUs, RMS generates an appropriate capability and pushes it into the RMS kernel module on each of the allocated nodes. The capabilities together with information on the processes that make up the program can then be accessed through the `librmscall` system call library (see Section C.3 for details).

A.3 Network Topology

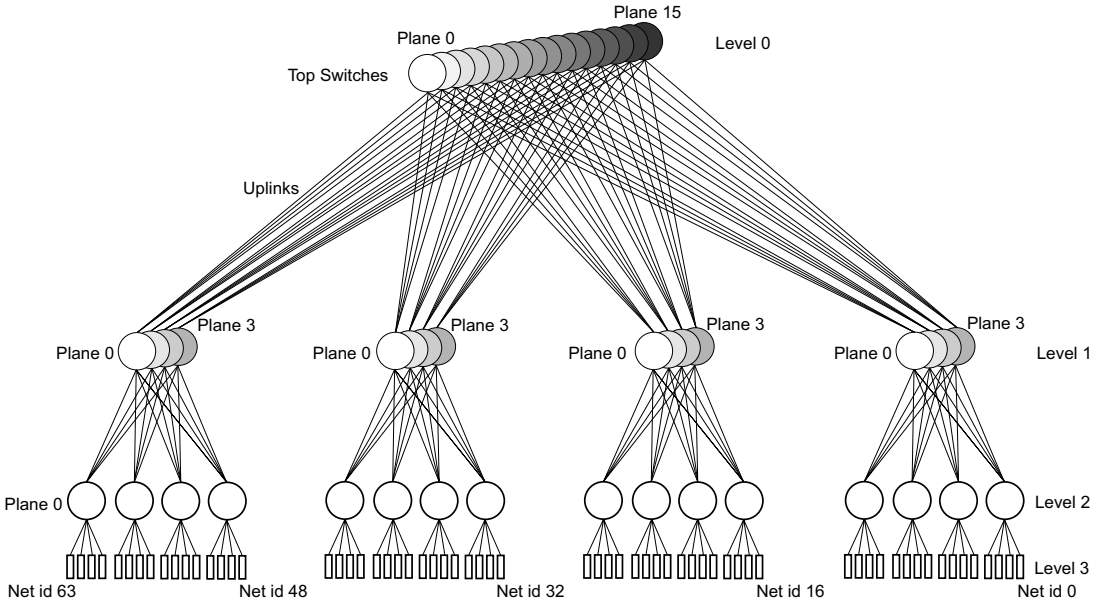
HP AlphaServer SC Interconnect is a multistage switch network, also known as a *fat tree network*. It is built from 8-way *crosspoint switches*, known as Elites. Each node is connected to the network by a network adapter, the Elan. The connection of a 2-stage (16-node) switch network is shown in Figure A.1.

Figure A.1: A 2-Stage, 16-Node, Switch Network



The *level* is the index of the stage, starting with 0 at the top. Note that in a 2-stage switch network the Elans are at level 2. Each component has a network ID that describes how to reach it from the top of the network. The *plane* is the index of switches that have the same switch network ID. The interconnection of a 3-stage (64-node) switch network is shown in Figure A.2.

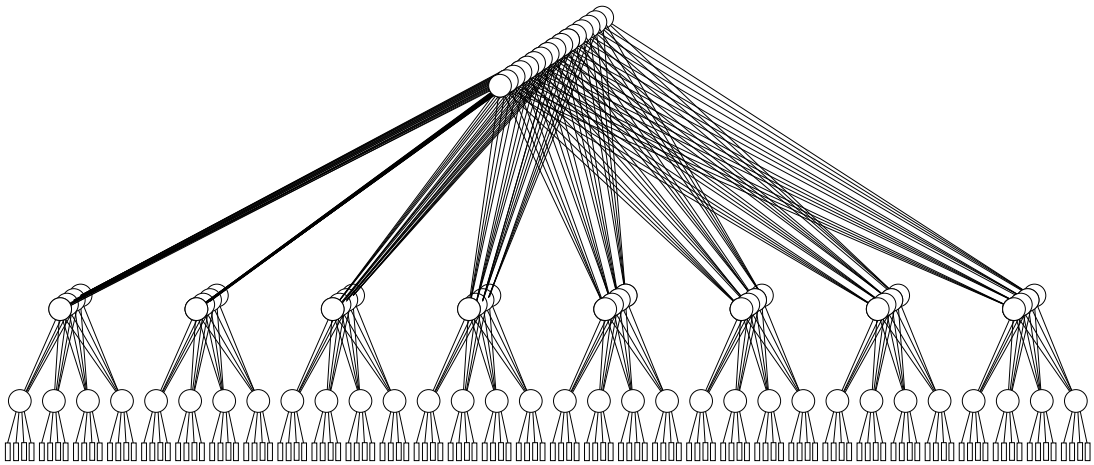
Figure A.2: A 3-Stage, 64-Node, Switch Network



The 3-stage switch network is composed from four 2-stage networks. Each 2-stage network connects sixteen nodes and has sixteen free *uplinks*. These uplinks connect the four 2-stage networks to an additional level of switches to form a 3-stage network, connecting up to 64 nodes.

Four such 64-node networks and an additional stage of switches can be used to construct a 256-way network. Alternatively, the unused uplinks can be used to double the number of nodes a switch can connect. This avoids the need to add an additional switch stage but the resulting network cannot be expanded further. This technique is used in the 128-node network, shown in Figure A.3.

Figure A.3: A 3-Stage, 128-Node, Switch Network



There are switch network modules that connect up to 16 or 128 nodes. The 16-node network is a rack-mountable module containing a single network board. The 128-node network is a rack-mountable chassis containing up to 24 network boards: 8 at the front connecting the nodes to the lower stages; and 16 at the rear providing the upper stages of the network. A central backplane joins the stages. These switch modules may be partially populated for networks containing fewer than 128 nodes.

The number of nodes and switches in these networks is shown in Table A.1. The number of switches refers to the total number of Elite ASICs required to construct the network. The number of hops refers to the maximum number of links traversed for nodes that have to communicate through a top switch. The bidirectional nature of the links means that traffic can be localized to a subtree large enough to span both nodes.

Table A.1: Switch Network Parameters

Name	Levels	Nodes	Switches	Hops
QMS16	2	16	8	4
QMS128	3	128	80	6
QMF256	4	256	256	8

(continued on next page)

Table A.1: Switch Network Parameters (cont.)

Name	Levels	Nodes	Switches	Hops
QMF512	4	512	448	8
QMF1024	5	1024	1280	10
QMF1536	4	256	1380	8

The Elan performs automatic routing and broadcast communications. Using the switch network, data can be broadcast directly to a contiguous range of processors: data is routed up to a node in the tree from which all processors can be reached; then the data is routed down to all switch outputs in the broadcast range on the way down. Data can be recombined as it travels through the network to support global *reduction* operations and *barrier synchronization*.

Multiple Elan network adapters may be installed per node, each connected to a different switch network. This replication can increase fault tolerance and bisectional bandwidth, assuming each Elan is attached to a separate PCI bus. Each separate Elan/Elite network attached to a node is known as a *layer* (or a *rail*).

A.3.1 Broadcast Trees

HP AlphaServer SC Interconnect networks consist of multiple independent trees, each identified by a top switch. One of these trees (per rail) is special in that it carries all broadcast traffic. The number of trees (and hence the range of valid top switch ids) varies with the network type as indicated below

Network Type	Tree Ids
QMS16	0 - 3
QMS128	0 - 15
QMF256	0 - 63
QMF512	0 - 63
QMF1024	0 - 63
QMF1536	0 - 31

The broadcast trees are labelled by the route to their top switch, tree `0`, being the `4,4,4` route and tree `63` being the `7,7,7` route.

The `QMS16` and `QMS128` switches are standalone networks in a single chassis. The `QMF`*n* networks are federated networks constructed from multiple 64 or 96-way node switches connected by 4 or 16-way top switches.

A.3.2 Clock Source

Standalone switches have a local clock source. In a federated network the switch modules are all clocked from one of two clock boxes connected to clock inputs A and B. It is necessary to ensure that all switch modules on a given rail have the same clock source.

A.4 Database Representation

The switch network is described by four tables in the database. The `switch_modules` table (see Section 11.5.26) provides information about each switch module configured into the machine. The `switch_boards` table (see Section 11.5.25) gives details of each board, its status and its position in the machine. The `elans` table (see Section 11.5.6) and the `elites` table (see Section 11.5.7) describe the position in the switch network of each component, its attributes and its current link state and errors.

A.5 Control Interface

RMS includes the control and monitoring daemon, `swmgr` (see Section 4.5), for managing the switch network. `swmgr` resides on the `rmshost` node. It interacts with `swmserver` processes, which reside on the nodes connected to the switch modules, to gather data on the network.

`swmgr` probes the switch network control interface for switch modules and boards to determine the size of the network. It then creates or updates the entries in the `elans` table, the `elites` table, the `switch_boards` table and the `switch_modules` table. Having done this, the `swmgr` uses the switch network control interface to extract error and performance data. This interface is also used for link continuity (boundary scan) testing. `swmgr` generates events to notify the system administrator about changes in the state of the network (see Chapter 8 (*Event Handling*)).

A.6 Link States

The state of each link in the switch network is recorded in the `linkerrors` field in the `elites` table (see Section 11.5.7). Valid values for the states are shown in Section B.4.

Links are normally in the *connected* state (`C`). Unconnected links will be in the *reset* state (`R`). Links will be in the *unknown* state (`U`) if the `swmgr` has not run or if the control cable is not attached to the switch. The states *acking* (`A`) and *nacking* (`N`) are set by the switch control software.

A.7 Link Errors

The `swmgr` logs network errors to the `link_errors` table (see Section 11.5.13). The description contains information that should be used in reporting a problem with the switch network. See *HP AlphaServer SC Interconnect Installation and Diagnostics Manual* for details of the daignostics.

B

RMS Status Values

B.1 Overview

This appendix lists the various states that RMS objects can enter. State information is stored in the `status` field of the RMS table for the object in question. For example, the current state of a partition is held in the `partitions` table (see Section 11.5.17). and the current state of a node is entered in the `nodes` table (see Section 11.5.15).

Status changes are recorded in the `events` table (see Section 11.5.9). Entries in the `events` table are identified by `class=`*X* and `name=`*N* where *X* is the class of object and *N* is its name. For most tables, the `name` field forms the primary key. In the case of a partition, both the `name` and the `configuration` fields are required to define a unique entry. A switch module is identified by its `name` and `type` fields. A switch board is identified by its `name` and `rail`.

Status values are shown for the following RMS objects:

- Jobs (see Section B.3)
- Links (see Section B.4)
- Switch network (see Section B.5)
- Environment of nodes (see Section B.6)
- Nodes (see Section B.7)
- Partitions (see Section B.8)

- Resources (see Section B.9)
- Transactions (see Section B.10)

B.2 Generic Status Values

There are three generic status values:

`ok`	This state means that an object is functioning correctly as far as the relevant RMS daemon can tell.
`error`	This state means that one or more errors have been detected. A description of the problem will be found in the event record.
`unknown`	This state means that the RMS daemon responsible for an object either has not run or is unable to determine the state of the object.

Where an object, such as a switch board, has many component status values, the `ok` state means that all component values are `ok`. If one or more of the components are in error then the status will be `error`.

B.3 Job Status Values

The status of each job is stored in the `status` field of the `jobs` table. It is updated by the Partition Manager when the job is started, suspended, resumed or completed. Valid job status strings are shown in Table B.1.

Table B.1: Job Status Values

Status	Description
`running`	All processes are scheduled
`suspended`	All processes are suspended
`finished`	All processes have exited
`hung`	One or more nodes is not responding
`aborted`	User aborted job (Ctrl/C on `prun`)
`failed`	One or more of the nodes in use by this job crashed or was configured out
`killed`	An application process was killed by a signal
`expired`	Time limit has expired
`syskill`	Job was killed by an administrative user
`unknown`	Partition is blocked or down

If a job is killed because one of the nodes it was running on has crashed or has been configured out then its final status value will be `failed`.

B.4 Link Status Values

Each switch (see Appendix A (*HP AlphaServer SC Interconnect Terms*)) has an entry in the `elites` table. Each switch has eight links and the state of each of these links is recorded in the `linkstate` field of the `elites` table. The field holds eight characters, one for each link. Valid values for the characters are as shown in Table B.2. See also Section A.6.

Table B.2: Link Status Values

State	Character	Description
Connected	`C`	Normal working state
Reset	`R`	Link is in reset
Acking	`A`	Link generates an ACK for all transactions
Nacking	`N`	Link generates an NACK for all transactions
Not connected	`_`	Link is not connected
Unknown	`U`	`swmgr` cannot determine link state.

B.5 Switch Network Status Values

Status information for switch boards is held in the `switch_boards` table; that for switch modules is held in the `switch_modules` table. The information is updated by the Switch Network Manager, `swmgr` (see Section 4.5), on receipt of messages from the `swmserver` processes responsible for monitoring each switch module. Valid status values for switch boards and switch modules are shown in Table B.3.

Table B.3: Switch Board and Module Status Values

Status	Description
ok	Normal working state
absent	Board or module is disconnected or powered off
unknown	Board or module cannot be accessed
error	Board or module is reporting errors

In addition, the `switch_module` table records status information about the fans and PSUs (see Section 11.5.26).

Changes in status are recorded in the `events` table (see Table 11.16); entries are keyed

by `class=switch_module`, `type=`*X* and `name=`*N*, where *N* is the name of the switch module, as entered in the `name` field of the `switch_modules` table, and *X* is `control`, `board`, `temperature`, `temphigh`, `psu` or `fan`. The event descriptions are listed in Section 8.3.

B.6 Environment Status Values

Status information for nodes is held in the `nodes` table. The `estatus` field of this table stores the status of the node's operating environment; specifically, it shows whether the cooling fans and power supply units (PSUs) are working correctly and whether any of the other components are overheating.

The valid values for the `estatus` field and their meanings are shown in Table B.4.

Table B.4: Environment Status Values

Status	Description
`ok`	Fans, PSUs and temperature ok
`fan failure`	A fan has failed
`psu failure`	A power supply unit has failed
`node hot`	The enclosure is too hot
`cpu hot`	A CPU is too hot
`psu hot`	A power supply unit is too hot

If the environment status, `estatus`, is recorded as `ok`, the `environment` field of the `nodes` table contains temperature readings from the CPU, PSU and enclosure. If a node has more than one instance of each type of temperature sensor, the maximum of their values is recorded.

Temperature information is recorded as a list of attribute-value pairs, for example:

```
ambient=15 cpu=40 psu=20
```

Note that not all node types support all types of thermistor reading. The `environment` field may contain only a subset of this information.

If an error occurs, the environment string contains details of what has failed. For example, the following string indicates that the CPU fan number 1 has failed on the node.

```
cpu fan 1 failure
```

Changes in environment status are recorded in the `events` table (see Table 11.16); entries are keyed by `class=node`, `type=`*X* and `name=`*N*, where *N* is the name of the

node, as entered in the name field of the nodes table, and *X* is temperature, temphigh, psu or fan. The event descriptions are listed in Section 8.3.

B.7 Node Status Values

The current state of a node is found in the status and runlevel fields of the nodes table. State changes are logged in the events table; entries in the events table are identified by class=node and name=*N*, where *N* is the name of the node as entered in the name field of the nodes table. The event descriptions are listed in Section 8.3.

The status field contains one of the values shown in Table B.5.

Table B.5: Node Status Values

Status	Description
unknown	Node has been configured in
active	Node responds to IP requests but RMS is not running
running	RMS is active on this node
not responding	Machine Manager cannot get response from node
configured out	Node has been configured out of the machine

As a node boots, its status progresses from unknown then not responding to active and on to running. A long delay in reaching the running state indicates a problem with booting that should be investigated further. If a node changes from the running state to the active state and stays there then there is a problem with RMS on that node. If a node changes from the running state to the not responding state then either the node has crashed or IP communications to the node are failing. In this case, RMS runs the rmsevent_node event handler script. This script attempts to determine what went wrong.

The current UNIX run level of a node is held in the nodes table in the runlevel field. This field is updated by the nodestatus program as the run level changes. The valid strings are shown together with their meaning in Table B.6.

Table B.6: Run Level Status Values

Status	Description
run level S	Single user mode
run level 1	UNIX run level 1
run level 2	UNIX run level 2
run level 3	UNIX run level 3

The run level of a node that is configured out will continue to be updated as the node is booted or halted. RMS is started at `run level 3`.

B.8 Partition Status Values

The current state of a partition is entered in the `status` field of the `partitions` table. Changes to a partition's state are logged in the `events` table. Entries in the `events` table are identified by `class=partition` and `name=`*P*, where *P* is the name of the partition as entered in the `name` field of the `partitions` table. All such events refer to partitions in the active configuration. The event descriptions are listed in Section 8.3.

The partition's status can take one of the values shown in Table B.7.

Table B.7: Partition Status Values

Status	Description
`starting`	The partition is starting
`running`	The partition is operational
`blocked`	One or more of the `rmsd`s in the partition is not responding
`closing`	The partition is in the process of closing down
`down`	The partition has been shut down successfully

Note that the `active` field in the `partitions` table denotes whether or not the partition is part of the active configuration. The database may contain a number of different configurations but only one is active at any time.

B.9 Resource Status Values

The status of each resource request is stored in the `status` field of the `resources` table. It is updated by the `pmanager` when CPUs are allocated and deallocated and also as jobs using the CPUs complete. The valid resource status strings are as shown in Table B.8.

Table B.8: Resource Status Values

Status	Description
`blocked`	CPUs cannot be allocated because of a usage limit
`queued`	Insufficient CPUs free for the request
`allocated`	CPUs are allocated
`suspended`	CPUs are temporarily deallocated
`finished`	CPUs have been deallocated

(continued on next page)

Table B.8: Resource Status Values (cont.)

Status	Description
`aborted`	CPUs were deallocated when `prun` was killed
`killed`	CPUs were deallocated when resource request was killed
`failed`	CPUs were deallocated because node crashed or configured out
`expired`	Time limit expired
`syskill`	Job was killed by an administrative user

The final status of a resource is that of the last job to exit (see Table B.1).

B.10 Transaction Status Values

Requests to change the state of the machine are entered in the `transactions` table. The Transaction Log Manager, `tlogmgr`, actions the requests on the users' behalf. This mechanism provides an audit trail of state changes.

Each request has a `status` field associated with it. Valid values for this field are shown in Table B.9.

Table B.9: Transaction Status Values

Status	Description
`submitted`	Client application has submitted transaction
`started`	`tlogmgr` has started to execute the transaction
`complete`	Transaction completed successfully without errors
`error`	Transaction completed with errors
`failed`	Transaction failed

C

RMS Kernel Module

C.1 Introduction

The RMS kernel module supports the operation of RMS on each node in a system. It provides functions that bind together the set of processes that make up a program on each node, allowing RMS to apply scheduling, signal delivery and statistics gathering operations to them collectively. For example, the RMS kernel module allows the `rmsd` daemon or an administrator process to send a signal to all processes in a parallel program at the same time.

The RMS kernel module stores the Elan capabilities assigned to a program, making them available to the processes of that program and their children. The capabilities are only accessible to processes that belong to this one parallel program; they are not available to other processes. The RMS kernel module keeps track of processes through handlers called whenever a process belonging to a parallel program forks or exits.

Note that a parallel program under RMS may consist of one or more UNIX process groups or sessions. It is the ability of a UNIX process to call `setpgrp` or `setsid` at any time that makes the program structure provided by the RMS kernel module necessary; without it, suspend, resume, signal delivery and program cleanup operations on changing sets of processes are unreliable.

C.2 Capabilities

An Elan capability describes a parallel program's rights to use one rail of HP AlphaServer SC Interconnect. It specifies the range of nodes that have been allocated

and the Elan hardware context numbers to be used.

```
typedef struct elan_capability
{
    ELAN_USERKEY cap_userkey;                /* User defined protection      */
    int          cap_version;                /* Version number               */
    short        cap_type;                   /* Type                         */
    short        cap_spare;                  /* spare was cap_elan_type */
    int          cap_lowcontext;             /* low context number in block  */
    int          cap_highcontext;            /* high context number in block */
    int          cap_myontext;               /* my context number            */
    int          cap_lownode;                /* low elan id of group         */
    int          cap_highnode;               /* high elan id of group        */
    unsigned int cap_railmask;               /* rails this capability is valid for */
    bitmap_t     cap_bitmap[ELAN_BITMAPSIZE]; /* Bitmap of process to node translation  */
} ELAN_CAPABILITY;
```

Elan capabilities are created on each node allocated to a parallel program and passed to its processes through the RMS kernel module.

C.3 System Call Interface

The RMS kernel module is accessed through its system call interface. This interface allows processes with administrator privileges to create program descriptions, add Elan capabilities to them, collect resource utilization statistics from them and destroy them when their processes have exited. It also allows them to suspend or resume the processes and deliver signals to them.

The RMS system call interface allows user processes to determine how many nodes, CPUs, rails and contexts they have been allocated. This information is primarily (but not exclusively) for use by parallel programming libraries.

NAME

rms_setcorepath, rms_getcorepath – Set, get the path for application core files

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrmscall [ library ...  ]
#include <rms/rmscall.h>

int rms_setcorepath(caddr_t path);
int rms_getcorepath(pid_t pid, caddr_t path, int maxlen);
```

PARAMETERS

`path`	Array containing the path name.
`maxlen`	Size of the array pointed to by `path`.
`pid`	Process identifier.

DESCRIPTION

The function `rms_setcorepath()` sets the core file path for the current process. The core file path is set by `rmsd` as it starts a new parallel program and inherited by any child processes. Administrator privileges are required to set a core path.

The function `rms_getcorepath()` returns the core file path of a process. If *`pid`* is negative, it returns the core file path of the current process.

RETURN VALUES

Upon successful completion, `rms_setcorepath()` and `rms_getcorepath()` return `0`. Otherwise, they return `-1` and set `errno` to indicate the error.

`EACCESS`	Caller is not permitted to perform this operation.
`ENOMEM`	Insufficient memory to perform this operation.
`ESRCH`	Process does not exist.
`EEXIST`	The core path has not been set.

NAME

rms_prgcreate, rms_prgdestroy – Create, destroy program descriptions

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrmscall [ library ...  ]
#include <rms/rmscall.h>

int rms_prgcreate(int id, uid_t uid, int cpus);
int rms_prgdestroy(int id);
```

PARAMETERS

`id`	Program identifier.
`uid`	Owner of the program.
`cpus`	Number of CPUs allocated.

DESCRIPTION

`rms_prgcreate()` creates a new program description with the current process as its root process. Any children of this process will belong to this program. `rms_prgdestroy()` destroys an existing program description. Calls to `rms_prgdestroy()` will fail if processes belonging to this program are still running. Creating or destroying program descriptions requires administrator privileges. The `rmsd` creates a new program description for each parallel program. The program identifier is the unique identifier for the parallel job, that is to say, it is common across nodes.

RETURN VALUES

Upon successful completion, `rms_prgcreate()` and `rms_prgdestroy()` return `0`. Otherwise, they return `-1` and set `errno` to indicate the error.

`EACCESS`	Caller is not permitted to perform this operation.
`ENOMEM`	Insufficient memory to perform this operation.

EINVAL	Program identifier is in use or the number of CPUs is invalid.
ECHILD	Processes belonging to this program are still running.
EEXIST	Program identifier does not exist.

SEE ALSO

rms_getprgid(3)

NAME

rms_prgids, rms_prginfo, rms_getprgid – Get information on a program or programs

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrmscall [ library ...  ]
#include <rms/rmscall.h>

int rms_prgids(int maxids, int *ids, int *nids);
int rms_prginfo(int id, int maxids, pid_t *pids, int nids);
int rms_getprgid(int pid, int *id);
```

PARAMETERS

`id`	Program identifier.
`pid`	Process identifier.
`maxids`	Maximum number of identifiers to be returned.
`ids`	Array of program identifiers.
`pids`	Array of process identifiers.
`nids`	Number of program or process identifiers returned.

DESCRIPTION

`rms_prgids()` returns the identifiers of each active program. `rms_prginfo()` returns the identifiers for each process belonging to a particular parallel program – the current program if *`id`* is negative. `rms_getprgid()` returns the program identifier (if any) for a particular process – the current process if *`pid`* is negative.

RETURN VALUES

Upon successful completion, `rms_prgids()`, `rms_prginfo()` and `rms_getprgid()` return `0`. Otherwise, they return `-1` and set `errno` to indicate the error.

`EACCESS`	Caller is not permitted to perform this operation.

EINVAL	Count of number of array elements is invalid.
EFAULT	Array address is invalid.
ENOMEM	Insufficient kernel memory to perform this operation.
ESRCH	Process or program does not exist.

SEE ALSO

rms_prgcreate(3)

NAME

rms_prgsuspend, rms_prgresume, rms_prgsignal – Suspend or resume the processes in a program, deliver a signal to all processes in a program

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrmscall [ library ...  ]
#include <rms/rmscall.h>

int rms_prgsuspend(int id);
int rms_prgresume(int id);
int rms_prgsignal(int id, int signo);
```

PARAMETERS

`id`	Program identifier.
`signo`	Signal number.

DESCRIPTION

`rms_prgsuspend()` suspends all of the processes in a program. The RMS suspends a parallel program by calling `rms_prgsuspend()` on each node that it is using. `rms_prgsuspend()` requires administrator privileges. `rms_prgresume()` resumes all of the processes in a program. The RMS resumes a parallel program by calling `rms_prgresume()` on each node that it is using. `rms_prgresume()` requires administrator privileges.

`rms_prgsignal()` sends a signal to all of the processes in a program. The RMS delivers signals to a parallel program by calling `rms_prgsignal()` on each node that it is using. The function is also used to confirm that all processes belonging to a program have exited. `rms_prgsignal()` can be called by the owner of the program or a process with administrator privileges.

RETURN VALUES

Upon successful completion, `rms_prgsuspend()`, `rms_prgresume()` and `rms_prgsignal()` return `0`. Otherwise, they return `-1` and set `errno` to indicate the error.

EACCESS	Caller is not permitted to perform this operation.
ESRCH	No such program identifier.
EINVAL	Invalid signal number.

SEE ALSO

rms_prgcreate(3)

NAME

rms_prgaddcap, rms_setcap – Associate Elan capabilities with a program or process

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrmscall [ library ...  ]
#include <rms/rmscall.h>

int rms_prgaddcap(int id, int index, ELAN_CAPABILITY *cap);
int rms_setcap(int index, int context);
```

PARAMETERS

`id`	Program identifier.
`index`	Index of the capability for this program.
`cap`	Pointer to a capability.
`context`	Context number for this process.

DESCRIPTION

`rms_prgaddcap()` and `rms_setcap()` associate Elan capabilities with a program and its processes. The function `rms_prgaddcap()` adds a new capability to a program. It is called once for each rail in use by the program. Each capability defines the range of node numbers and Elan hardware contexts available to a parallel program. Capabilities are indexed from `0` to *`ncaps`*`-1` where *`ncaps`* is the number of capabilities allocated.

`rms_prgaddcap()` requires administrator privileges. It is called by `rmsd` as it creates a parallel program.

The function `rms_setcap()` assigns Elan hardware context numbers to the current process. It is called by the RMS application loader, `rmsloader`, as it creates each new application process. The contexts assigned must lie within a previously assigned capability for the program.

RETURN VALUES

Upon successful completion, `rms_prgaddcap()` and `rms_setcap()` return `0`. Otherwise, they return `-1` and set `errno` to indicate the error.

EACCESS	Caller is not permitted to perform this operation.
ENOMEM	There was insufficient memory to perform this operation.
ESRCH	Program does not exist.
EFAULT	Capability has invalid address.
EINVAL	Invalid context number (`rms_setcap()` only).

SEE ALSO

rms_ncaps(3)

NAME

rms_ncaps, rms_getcap – Return information on the Elan capabilities allocated to a process in a parallel program

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrmscall [ library ...  ]
#include <rms/rmscall.h>

int rms_ncaps(int *ncaps);
int rms_getcap(int index, ELAN_CAPABILITY *cap);
```

PARAMETERS

`ncaps`	Number of capabilities allocated.
`index`	Index of a capability to be returned.
`cap`	Pointer to a capability.

DESCRIPTION

`rms_ncaps()` returns the number of Elan capabilities allocated to a program. `rms_getcap()` returns a specified Elan capability. Capabilities are indexed from `0` to *`ncaps`*`-1`.

RETURN VALUES

Upon successful completion, `rms_ncaps()` and `rms_getcap()` return `0`. Otherwise, they return `-1` and set `errno` to indicate the error.

`EFAULT`	Invalid address.
`EINVAL`	Invalid capability identifier.
`EEXIST`	Calling process is not part of a parallel program.

SEE ALSO

`rms_prgaddcap``(3)`

NAME

rms_prggetstats – Return resource usage information for a program

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrmscall [ library ...  ]
#include <rms/rmscall.h>
int rms_prggetstats(int id, prgstats_t *stats);
```

PARAMETERS

`id`	Program identifier.
`stats`	Pointer to a program statistics structure.

DESCRIPTION

`rms_prggetstats()` returns resource usage information for the processes of a parallel program on the calling node. The RMS kernel module sums resource usage over the processes in a program. The statistics returned by `rms_prggetstats()` are the sum over all processes belonging to program *id* on this node, including those that have already exited.

Setting *id* to `-1` instructs the RMS kernel module to return values for the caller's program. Resource utilization statistics are available to the owner of the program and to any process with administrator privileges.

```
/*
 * program statistics
 */
typedef struct {
    uint64_t etime;             /* elapsed cpu time (millisecs)       */
    uint64_t atime;             /* allocated cpu time (millisecs)     */
    uint64_t utime;             /* user cpu time (millisecs)          */
    uint64_t stime;             /* system cpu time (millisecs)        */
    int ncpus;                  /* number of cpus allocated           */
    int flags;                  /* program status flags               */
    int mem;                    /* max memory size in megabytes       */
    int pageflts                /* number of page faults              */
    uint64_t memint;            /* memory integral                    */
} prgstats_t;
```

The elapsed time statistic `etime` is the time in millisecs since the program was created. The allocated time statistic `atime` is the time in millisecs for which CPUs have been allocated multiplied by the number of CPUs allocated. The `utime` and `etime` statistics are summed over the processes that make up the program (on this node).

If one or more processes belonging to the program is still running, the `flags` field will contain the value `PRG_RUNNING`. This will be replaced by `PRG_ZOMBIE` when the last process has exited. The program description should be destroyed when this value is seen.

The Partition Manager periodically sums these statistics over the nodes used to run a parallel program, updating its entry in the accounting statistics (`acctstats`) table.

RETURN VALUES

Upon successful completion, `rms_prggetstats()` returns `0`. Otherwise, it returns `-1` and sets `errno` to indicate the error.

`EACCESS`	Caller is not permitted to perform this operation.
`EFAULT`	Invalid address for statistics array.
`ESRCH`	No such program.

SEE ALSO

`rms_prginfo``(3)`

NAME

rms_setpset, rms_getpset – Associate a processor set with a parallel job

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrmscall [ library ...  ]
#include <rms/rmscall.h>
int rms_pset(int psid);
int rms_getpset(int id, int * psid);
```

PARAMETERS

psid	Processor set id.
id	Program id.

DESCRIPTION

The function rms_setpsetid() associates a processor set with a parallel program, that of the calling process.

The function rms_getpsetid() returns the processor set id for a parallel job given its program id.

RETURN VALUES

Upon successful completion, rms_setpsetid() and rms_getpsetid() return 0. Otherwise, they return -1 and set errno to indicate the error.

EACCESS	Caller is not permitted to perform this operation.
EFAULT	Invalid address passed to rms_getpsetid
ESRCH	The parallel program specified does not exist.

NAME

rms_elaninitdone – Informs the RMS kernel module that a process has completed its Elan initialization

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrmscall [ library ...  ]
#include <rms/rmscall.h>
int rms_elaninitdone(int vp);
```

PARAMETERS

`vp`	Virtual process id of the process, its rank in a parallel program.

DESCRIPTION

The function `rms_elaninitdone()` informs the RMS kernel module that a process has completed its Elan initialization. It is used to distinguish parallel programs that have successfully execute the startup barrier in the Elan library from those that either never call, or never complete this operation.

RETURN VALUES

Upon successful completion, `rms_elaninitdone()` returns `0`. Otherwise, it returns `-1` and set `errno` to indicate the error.

`ESRCH`	The process is not part of a parallel program.

D

RMS Application Interface

D.1 Introduction

The RMS application interface is provided so that external batch systems and workload schedulers can make inquiries about the availability of resources, allocate and deallocate CPUs and perform job control operations.

The scheduler should query the size of a partition using `rms_numNodes()` and `rms_numCpus()` or the partition description function `rms_partitionInfo()`. Use the functions `rms_freeCpus()` or `rms_freeCpusPerNode()` to get information on the number of free CPUs and their distribution over nodes. A scheduler can then allocate CPUs to users using `rms_allocateResource()`. A successfull allocation request returns a unique identifier for the allocated resource.

Alternatively, to build up a complex resource request which requires more than one set (or box) of CPUs, the scheduler should call `rms_createResource()`. This allocates a resource for a specified user with details about the user's project, priority and time limit but no information about the processing requirements. To add the processing information, the scheduler can make one or more calls to `rms_allocateCpus()`. The scheduler can choose whether to block waiting for additional CPUs to become available or to back off and make use of the CPUs already allocated.

Having allocated CPUs, a batch system can start the user's job script. The resource ID is passed to this script in the `RMS_RESOURCEID` environment variable. Invocations of `prun` in the job script will pick up this environment variable and run on the allocated resource. The function `rms_resourceId()` returns the RMS resource ID for a given

batch system ID.

A workload scheduler can call `rms_run()` directly to generate its own parallel jobs. This avoids the need to `fork` a process and `exec prun`.

The partition type controls whether allocation requests made by users who are not privileged will succeed. If its value is `batch` then only the batch scheduler can allocate CPUs; interactive `prun` requests will fail. Similarly, if the partition type is `external`, only the external scheduler can allocate CPUs. If the partition type is `general` or `parallel` then RMS will treat the batch system as a source of allocation requests but not as the exclusive source.

An external scheduler can use the `rms_suspendResource()` and `rms_resumeResource()` calls to suspend the allocation of CPUs to a user. This will cause all processes running on the allocated CPUs to be suspended. The `rms_killResource()` call delivers a signal to all processes running on an allocated resource. Users can execute multiple jobs on a given resource. The function `rms_jobIds()` returns the ids of the current jobs. The `rms_killJob()` call delivers a signal to all processes in a specified job. These job control operations can be performed by a member of the `rms` group or the owner of the resource.

The function `rms_nodeInfo()` returns information about one or more nodes while `rms_getNodeStats()` returns performance statistics for each node. The function `rms_resourceStats()` returns accounting data on the resources consumed by a parallel job.

D.1.1 Compilation

The RMS application interface is provided as a dynamic library `librmsapi.so`. Function prototypes are defined in the header file `<rms/rmsapi.h>`.

Programs using the RMS application interface should be compiled as follows:

```
$ cc -o myprog myprog.c -lrms -lrmsapi
```

The RMS header file `rms/version.h` includes version information on RMS. The function `rms_version()` reports the version of RMS that a program is linked with. The function `rms_checkVersion()` checks that the version of RMS a program was compiled against is compatibile with the installed version. The `RMS_VERSION` macro allows a program to be conditionally compiled against specific versions. For example:

```
#if RMS_VERSION_CODE < RMS_VERSION(2,81)
     printf("drain not supported\n");
#else
     printf("drain %d\n", nodeinfo.drain);
#endif
```

D.1.2 Error handling

RMS application interface functions return `0` on success and a negative error code on error. The error codes are defined in `<rms/ecodes.h>`. The function `rms_errorString()` converts an RMS error code into a character string suitable for printing by the caller as part of an error message.

If the database server is down when an RMS application interface function is called the following message will be printed on `stderr`

```
Warning: Can't connect to MSQL server on rmshost: retrying ...
```

the function will block for a period of approximately 60 seconds before failing with the error code `RMS_RC_ERRORSTRING`.

D.2 Manual pages

Manual pages for the RMS application interface functions follow:

NAME

rms_allocateCpus – Allocate CPUs to a resource

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
int rms_allocateCpus(int rid, int cpus, char *requestFlags);
```

PARAMETERS

`rid`	Resource identifier returned from a previous call to `rms_createResource()`.	
`cpus`	Total number of CPUs to allocate.	
`requestFlags`	The request flags currently supported are as follows:	
	`nodes=`*`n`*	This specifies the number of nodes required.
	`baseCpu=`*`n`*	This specifies the base of a range of CPUs to allocate on a given node or nodes.
	`baseHost=`*`hostname`*	This specifies the first node by name.
	`baseNode=`*`index`*	This specifies the first node by its number in the partition.
	`cpusPerNode=`*`n`*	This specifies the number of cpus required on each node.
	`hwbcast=0 | 1`	With a value of 1, this specifies a contiguous range of nodes and constrains the scheduler to queue the request until a contiguous range becomes available. The default is 0.
	`immediate=0 | 1`	With a value of 1, this specifies that the request should fail if resources are not available immediately.
	`memlimit=`*`value`*	This specifies how many megabytes of memory are required.

`rails=`*n*	In a multirail system, this specifies the number of rails required, where $1 \leq n \leq 32$.
`railmask=`*mask*	In a multirail system, this specifies which rails are required, using a bitmask to select them.
`topCpu=`*n*	This specifies the top of a range of CPUs to allocate on a given node or nodes.
`topHost=`*hostname*	This specifies the last node by name.

Multiple request flags can be given as a comma-separated list, for example, `immediate=1,hwbcast=1`.

DESCRIPTION

`rms_allocateCpus()` allocates CPUs for a resource that has previously been created by a call to `rms_createResource()`.

The function may be called one or more times with a given resource ID to allocate multiple sets (or, in scheduler terms, boxes) of CPUs for the resource.

The *`requestFlags`* place constraints on the set of CPUs allocated. If a base node and the number of nodes are used to specify a range of nodes, the `hwbcast` request flag is redundant. This flag is intended for use when only the number of nodes is specified but a contiguous range is required.

By setting the `immediate` flag, the caller can choose whether to block waiting for additional CPUs to become available. Blocking and non-blocking requests are handled differently: if a request blocks, all CPUs previously allocated to it are deallocated but the request (including previous requests) remains queued; if a request fails rather than blocking, all CPUs previously allocated to it remain allocated. This behavior prevents deadlock. It also supports requests that specify a minimum number of CPUs.

The request flag `topHost` can be set to guard against nodes being configured in or out after the resource request has been made but before the CPUs are allocated. The following consistancy check is applied:

```
baseHost + nNodes - 1 = topHost - nConfiguredOut
```

where `nNodes` is the number of nodes requested and `nConfiguredOut` is the number of nodes configured out in the range `baseHost` to `topHost`. The call to `rms_allocateCpus()` will fail if this consistancy check fails. For example if you wish to allocate CPUs on nodes `atlas[4-7]` you should set the following request flags:

```
baseHost=atlas4,nNodes=4,topHost=atlas7
```

Specifying the `topHost` request flag is optional. It is not needed if the intention is simply to allocate a range of nodes.

The `baseCpu` and `topCpu` request flags allow specific CPUs to be allocated within each node. They would normally be used in conjunction with `baseHost` and `topHost` to request particular CPUs on a particular node or range of nodes. The value for `baseCpu` must be less than or equal to that for `topCpu`, both must be present and both must be less than the number of CPUs on the node.

If the partition defined for the request by `rms_createResource()` is of type `batch` or `external` (under the control of either a batch or external scheduler), the caller must be a privileged user. In other cases, the caller must match the *`uid`* field of the resource.

Before executing `rms_run()` to run a program on the resource, the environment variable `RMS_RESOURCEID` must be set to the value *`partition.rid`* (where *`partition`* is the name of the partition and *`rid`* is the resource ID returned by `rms_createResource()`).

Once a program has run on the resource, no further calls may be made to `rms_allocateCpus()` to add sets of CPUs to the resource.

Each call to `rms_allocateCpus()` allocates CPUs on a range of nodes (called a segment). RMS imposes a limit (of 150) on the maximum number of segments for a given resource request. Allocating large numbers of segments is inefficient and increases CPU usage in the partition manager. Where possible, an application should minimise the number of segments requested. For example, rather than calling `rms_allocateCpus()` once for each node, sort the nodes being requested by their id and make one request for each contiguous range. The function `rms_allocateHosts()` performs this optimization.

RETURN VALUES

On successful completion, `rms_allocateCpus()` returns the ID of the resource allocated. This value should be passed to subsequent calls. A negative integer is returned on error. This value can be passed to `rms_errorString()` to return a printable description of the error.

SEE ALSO

`rms_allocateHosts``(3)`, **`rms_allocateResource`**`(3)`, **`rms_createResource`**`(3)`

NAME

rms_allocateHosts – Allocate CPUs on multiple nodes to a resource

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
int rms_allocateHosts(int rid, char *hosts, int cpusPerNode,
                      char *requestFlags);
```

PARAMETERS

`rid`	Resource identifier returned from a previous call to `rms_createResource()`.
`hosts`	Hostnames on which to allocate CPUs.
`cpusPerNode`	Number of CPUs to allocate on each node.
`requestFlags`	The request flags currently supported are as follows:

`immediate=0 | 1`	With a value of 1, this specifies that the request should fail if resources are not available immediately.
`memlimit=`*`value`*	This specifies how many megabytes of memory are required.

Multiple request flags can be given as a comma-separated list, for example, `immediate=1,hwbcast=1`.

DESCRIPTION

The function `rms_allocateHosts()` allocates CPUs on one or more nodes for a resource that has previously been created by a call to `rms_createResource()`. The names of the nodes to allocate are given in a space or comma separated list `hosts`. For example to allocate four CPUs on each of `atlas4`, `atlas5`, `atlas8`, and `atlas9` use `rms_allocateHosts()` as follows:

```
rms_allocateHosts(rid,"atlas4,atlas5,atlas8,atlas9", 4, requestFlags);
```

The *requestFlags* place constraints on the set of CPUs allocated. By setting the `immediate` flag, the caller can choose whether to block waiting for additional CPUs to become available. Blocking and non-blocking requests are handled differently: if a request blocks, all CPUs previously allocated to it are deallocated but the request (including previous requests) remains queued; if a request fails rather than blocking, all CPUs previously allocated to it remain allocated. This behavior prevents deadlock. It also supports requests that specify a minimum number of CPUs.

If the partition defined for the request by `rms_createResource()` is of type `batch` or `external` (under the control of either a batch or external scheduler), the caller must be a privileged user. In other cases, the caller must match the *uid* field of the resource.

The function `rms_allocateHosts()` may be called one or more times with a given resource ID to allocate multiple sets of CPUs for the resource, but once a program has run on the resource, no further calls may be made to `rms_allocateHosts()`.

RETURN VALUES

On successful completion, `rms_allocateHosts()` returns the ID of the resource allocated. This value should be passed to subsequent calls. A negative integer is returned on error. This value can be passed to `rms_errorString()` to return a printable description of the error.

SEE ALSO

rms_createResource(3)

NAME

rms_allocateResource – Allocate a resource

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
int rms_allocateResource(char *partition, int cpus, int baseNode,
                         int nodes, uid_t uid, char *project,
                         char *requestFlags);
```

PARAMETERS

`partition`	Partition containing the resources.
`cpus`	Total number of CPUs to allocate.
`baseNode`	ID of the first node to allocate.
`nodes`	Number of nodes to allocate.
`uid`	User on whose behalf the resource should be allocated.
`project`	User's project name.
`requestFlags`	The request flags currently supported are as follows:

`cluster=`*`cluster`* `|` *`cluster-member`*	This requests that all processes be run on the named *`cluster`* or on all nodes with the specified *`cluster-member`* ID.
`hwbcast=0 | 1`	With a value of 1, this specifies a contiguous range of nodes and constrains the scheduler to queue the request until a contiguous range becomes available. The default is 0.
`immediate=0 | 1`	With a value of 1, this specifies that the request should fail if resources are not available immediately.

`rails=`*`n`*	In a multirail system, this specifies the number of rails required, where $1 \leq n \leq 32$.
`railmask=`*`mask`*	In a multirail system, this specifies which rails are required, using a bitmask to select them.
`signal=`*`val`*	*`val`* specifies how RMS should handle signals. By default, `SIGUSR1`, `SIGUSR2` and `SIGWINCH` are forwarded to the application. *`val`* may be set as follows:

`none`	Do not forward signals
`current`	Forward default signals
`all`	Forward all signals that are caught

Multiple request flags can be given as a comma-separated list, for example, `immediate=1,hwbcast=1`. An error will be generated if the flags are inconsistant.

DESCRIPTION

`rms_allocateResource()` allocates CPUs from a named partition. If *`partition`* is `NULL`, the default partition is used, otherwise the named partition must exist. You can optionally specify the base node and the number of nodes as with the `allocate` (see Page 5-3) and `prun` (see Page 5-3) commands. Alternatively, this can be left to the scheduler by passing the value `RMS_UNASSIGNED`. If the requested CPUs are not available, the request will block unless the `immediate` flag has been entered, in which case the request will fail.

If the caller belongs to the `rms` group then `rms_allocateResource()` can be used to allocate CPUs on behalf of another user identified by *`uid`*. In this case, the CPUs will be available to this user only. If *`project`* is not null, the request is subject to the usage restrictions of, and is accounted to, the specified project, which must exist. If *`project`* is null, the user's default project applies.

Before running a program on the specified resource, the environment variable `RMS_RESOURCEID` must be set to the value *`partition.rid`* (where *`partition`* is the name of the partition and *`rid`* is the resource ID returned by `rms_allocateResource()`).

The function `rms_allocateResource()` will block if the partition to it refers is `blocked`. and the `immediate` flag is not set to 1. The call completes when the partition returns to the `running` state.

A process calling `rms_allocateResource()` must call `rms_deallocateResource()` and have it return success if it is to avoid leaking memory.

RETURN VALUES

Upon successful completion, `rms_allocateResource()` returns the ID of the resource allocated. This value should be passed to subsequent calls. A negative integer is returned on error. This value can be passed to `rms_errorString()` to return a printable description of the error.

SEE ALSO

`rms_deallocateResource``(3)`, **`rms_suspendResource`**`(3)`, **`rms_defaultPartition`**`(3)`

NAME

rms_closeDatabase – Close connection to the RMS database

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
void rms_closeDatabase();
```

DESCRIPTION

Calls to RMS API functions open a socket connection to the RMS database. This connection can be closed using the function `rms_closeDatabase()`. A new connection will be opened if required by the next API call.

The function `rms_closeDatabase()` should be used where a persistent process (a daemon or a long running application) is making RMS API calls from large numbers of nodes. The call should be made at the end of each sequence of API calls so as to minimise the number of database connections held open at any given point in time.

NAME

rms_createResource – Create a resource

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
int rms_createResource(char *partition, uid_t uid,
                       char *requestFlags);
```

PARAMETERS

`partition` Partition containing the resources.

`uid` User on whose behalf the resource should be allocated.

`requestFlags` The request flags currently supported are as follows:

`batchid=`*`val`*	Set the batch system's identifier for the resource to *`val`*.
`priority=`*`val`*	Set the priority of the resource to *`val`*.
`project=`*`name`*	Specifies which project to associate with the resource.
`rails=`*`n`*	In a multirail system, this specifies the number of rails required, where $1 \leq n \leq 32$.
`railmask=`*`mask`*	In a multirail system, this specifies which rails are required, using a bitmask to select them.
`timelimit=`*`secs`*	Set the resource's time limit in seconds.

Multiple request flags can be given as a comma-separated list, for example, `project=staff,priority=80`.

DESCRIPTION

`rms_createResource()` creates a resource on the specified *`partition`* for the user specified by *`uid`*.

The resource has no CPUs allocated to it. Subsequent calls to `rms_allocateCpus()` are used to build up a complex resource. (A complex resource is one with multiple sets or boxes of CPUs). See also Chapter 7 (*RMS Scheduling*).

The *`requestFlags`* parameter is set to specify additional attributes of the resource request, such as, its priority.

The request is subject to the usage restrictions of, and is accounted to, the specified `project`, which must exist. If `project` is null, the user's default project applies.

RETURN VALUES

Upon successful completion, `rms_createResource()` returns the ID of the resource allocated. This value should be passed to subsequent calls. A negative integer is returned on error. The supported error codes are as follows:

`-1` Request cannot be met.

SEE ALSO

`rms_allocateCpus``(3)`

NAME

rms_deallocateResource – Deallocate a resource

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
#include <rms/ecodes.h>
int rms_deallocateResource(int rid);
```

PARAMETERS

`rid`	ID of the resource to deallocate.

DESCRIPTION

The function `rms_deallocateResource()` deallocates a resource killing any jobs that are using its CPUs. This call will fail rather than block if CPUs cannot be deallocated quickly, for example while a partition is in the `blocked` state or its partition manager is down. Where the intention is to block until the CPUs have been deallocated you should repeat the call to `rms_deallocateResource()` until a successfull return code is generated.

When called by a process other than the one that allocated the resource the `rms_deallocateResource()` function is equivalent to calling `rms_killResource()` (with the signal `SIGKILL`) followed by a check on the RMS database to determine whether CPUs remain allocated.

RETURN VALUES

Upon successful completion, `rms_deallocateResource()` returns 0. A negative value is returned on error. This value can be passed to `rms_errorString()` to return a printable description of the error. The function `rms_deallocateResource()` will return the following error codes

`RMS_RC_OK`	The resource has been deallocated. Its CPUs are available for reuse.

`RMS_RC_NORESOURCE`	The resource identified by `rid` is invalid, i.e. does not appear in the RMS database.
`RMS_RC_RETRY`	The resource has not been deallocated yet. Repeat the call after a delay of a few seconds to confirm that the process has been completed.
`RMS_RC_SERVERDOWN`	The partition manager is down. The process of deallocating the resource will not have started yet. Repeat the call after a delay of a few seconds.
`RMS_RC_PERMDENIED`	The caller does not have permission to deallocate the resource identified by `rid`.

It will return success if the resource has already been deallocated.

SEE ALSO

`rms_allocateResource``(3)`, **`rms_suspendResource`**`(3)`, **`rms_defaultPartition`**`(3)`

NAME

rms_defaultPartition, rms_numCpus, rms_numNodes, rms_machineName, rms_partitionIndex, rms_whichPartition – Provide information on an RMS system

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>

char *rms_defaultPartition();
char *rms_machineName();
int rms_numCpus(char *partition);
int rms_numNodes(char *partition);
int rms_partitionIndex(char *hostname);
int rms_whichPartition(char *hostname, char *partition, int maxlen);
```

PARAMETERS

`partition`	Name of an active partition.
`hostname`	Name of a node.
`maxlen`	Maximum length of partition name.

DESCRIPTION

The function `rms_defaultPartition()` returns the name of the default partition, if one exists. `rms_numCpus()` returns the total number of CPUs in the named partition. If the `partition` argument is `NULL`, it returns the number of CPUs in the machine. The function `rms_machineName()` returns the name of the system, for example, `atlas`. `rms_numNodes()` returns the total number of nodes in the named partition. If the `partition` argument is `NULL`, it returns the number of nodes in the machine. The function `rms_partitionIndex()` returns the index of a node in its partition, values range from `0` to `n-1` where `n` is the number of nodes configured into the partition.

The function `rms_whichPartition()` fills the `partition` argument with the name of the active partition containing a given node, to a maximum length of `maxlen` characters.

RETURN VALUES

`rms_defaultPartition()` returns `NULL` on error. The function `rms_partitionIndex()` returns `-1` if the node is configured out. Other functions return `0` or greater on success and negative values on error. Error codes can be converted into strings with `rms_errorString()`.

SEE ALSO

`rms_allocateResource``(3)`, **`rms_errorString`**`(3)`

NAME

rms_errorString – Returns the text string for an RMS error code.

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
#include <rms/ecodes.h>
char *rms_errorString(int code);
```

PARAMETERS

`code`	Return code from an RMS library function.

DESCRIPTION

`rms_errorString()` returns a text string suitable for printing as part of an error report when a function in the RMS library returns with a error. For example:

```
if ((rc = rms_suspendResource(rid)) != RMS_RC_OK)
{
    fprintf(stderr,"failed to suspend resource %d: %s\n", rid, rms_errorString(rc));
    ...
}
```

The error codes returned by RMS functions are enumerated in the header file `<rms/ecodes.h>`. The error code `RMS_RC_ERRORSTRING` is used for error messages that do not have a specific code. When called with this value `rms_errorString()` will return an error string held in the RMS library.

RETURN VALUES

`rms_errorString()` always returns a printable character string.

NAME

rms_freeCpus, rms_freeCpusPerNode – Provide information on the current usage of RMS partitions

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>

int rms_freeCpus(char *partition);
int rms_freeCpusPerNode(char *partition, int *freeCpus, int nodes);
```

PARAMETERS

`partition`	Name of an active partition.
`freeCpus`	The number of CPUs currently free on each node in a partition.
`nodes`	Specifies the number of nodes (normally the number of nodes in the partition).

DESCRIPTION

`rms_freeCpus()` returns the number of free CPUs in the named partition. `rms_freeCpusPerNode()` returns the number of free CPUs in the named partition, it also fills the `freeCpus` array with the number of CPUs free on each node.

The calling process must run on a node in the HP AlphaServer SC system.

RETURN VALUES

On success, `rms_freeCpus()` and `rms_freeCpusPerNode()` return the number of CPUs currently free. A negative value is returned on error.

SEE ALSO

rms_allocateResource(3)

NAME

rms_getNodeStats – Return statistics on the utilization of a node.

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
int rms_getNodeStats(char *hostname, rms_nodestats_t *stats);
```

PARAMETERS

`hostname`	Name of the node.
`stats`	Current utilization statistics.

DESCRIPTION

`rms_getNodeStats()` returns information on the utilization of a node. The following statistics are provided:

```
typedef struct {
    int usercpu;               /* percentage user cpu time */
    int syscpu;                /* percentage system cpu time */
    int maxmem;                /* maximum memory (megabytes) */
    int freemem;               /* free memory (megabytes) */
    int maxswap;               /* maximim swap space (megabytes) */
    int freeswap;              /* free swap space (megabytes) */
    int maxtmp;                /* maximum temporary file space (megabytes) */
    int freetmp;               /* free temporary file space (megabytes) */
    int pages;                 /* pages in and out */
    int interrupts;            /* non clock interrupts */
    int users;                 /* number of users */
} rms_nodestats_t;
```

The rate at which node utilization statistics are updated is controlled by the `cpu-stats-poll-interval` attribute in the `attributes` table of the RMS database. (See Table 11.5).

RETURN VALUES

Upon successful completion, `rms_getNodeStats()` returns `0`. A negative value is returned on error.

NAME

rms_jobIds – Return the ids of jobs using a resource

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
int rms_jobIds(int rid, int *jids, int maxids);
```

PARAMETERS

`rid`	Id of the resource.
`jids`	Array of job ids to be returned.
`maxids`	Maximum number of job ids to be returned.

DESCRIPTION

`rms_jobIds()` returns the ids of jobs running on a given resource. It can be used in conjunction with `rms_killJob()` to send a signal to a specific job rather than to all of the jobs on a given resource.

RETURN VALUES

Upon successful completion, `rms_jobIds()` returns `0`. A negative value is returned on error.

SEE ALSO

rms_killJob(3)

NAME

rms_killJob – Send a signal to a job

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
int rms_killJob(int jid, int signo);
```

PARAMETERS

`rid`	Id of the job.
`signo`	Signal to be sent.

DESCRIPTION

`rms_killJob()` sends a signal to all the processes in a specific job. It can be used in conjunction with `rms_jobIds()` to send a signal to a specific job rather than to all of the jobs on a given resource.

```
int jids[MAX_JOBS];
...
if ((rc = rms_jobIds(rid, jids, MAX_JOBS)) < 0)
{
    fprintf(stderr,"%s: failed to get job ids for resource %d: %s\n",
            program, rid, rms_errorString(rc));
    return(-1)
}
else if (rc == 0)
    return(0)

if ((rc = rms_killJob(jids[0], SIGTERM)) < 0)
{
    fprintf(stderr,"%s: failed to kill job %d: %s\n",
            program, jids[0], rms_errorString(rc));
    return(-1)
}
```

RETURN VALUES

Upon successful completion, `rms_killJob()` returns `0`. A negative value is returned on error.

SEE ALSO

rms_jobIds(3), **rms_killResource**(3)

NAME

rms_nodeInfo – Return a description of one or more nodes

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>

int rms_nodeInfo(char *partition, rms_nodeinfo_t *info,
                 int maxnodes, int options);
```

PARAMETERS

`partition`	Name of a partition.
`info`	Array of node descriptions.
`maxnodes`	Number of elements in the description array.
`options`	Controls level of detail provided.

DESCRIPTION

`rms_nodeInfo()` returns a description of one or more nodes. The following information is available for each node.

```
typedef struct {
    char hostname[NAME_LEN];    /* name of the node */
#ifdef DIGITAL_UNIX
    char domain[NAME_LEN];      /* TruCluster domain name of the node */
#endif
    int  nodeIndex;             /* index in the machine */
    int  partitionIndex;        /* index in the partition (-1 if configured out) */
    int  nodeStatus;            /* status of the node */
    int  ncpus;                 /* number of cpus on this node */
    int  oncpus;                /* number of online cpus on this node */
    int  fcpus;                 /* number of free cpus on this node  */
    unsigned int fcpumask;      /* mask of the free cpus on this node */
    int  maxmem;                /* maximum memory (megabytes) */
    int  maxswap;               /* maximim swap space (megabytes) */
    int  maxtmp;                /* maximum temporary file space (megabytes) */
    int  freemem;               /* free memory */
```

```
    int  freeswap;              /* free swap space */
    int  freetmp;               /* free temporary file space */
    int  railmask;              /* mask of Elan rails present on this node */
    int  drain;                 /* jobs are being drained from this node */
} rms_nodeinfo_t;
```

If `rms_nodeInfo()` is called with the name of an active partition, it will return information on each node in that partition. If the partition argument is `NULL`, it will return information on each node in the system. The `info` array should be sized using `rms_numNodes()`. On return the `hostname` contains the short form hostname of each node. The `domain` field contains the name of the TruCluster domain for each node.

The `nodeIndex` is the index of the node in the machine as a whole; values range from `0` to `N-1` where the machine has `N` nodes. The `partitionIndex` is the index of the node in its current partition; values range from `0` to `n-1` where the partition has `n` nodes. The `partitionIndex` is `-1` if the node is configured out and `-2` if it is not part of a partition.

Status values are the first character of the current node status. Supported values are `u` (unknown), `n` (not responding), `a` (active), `r` (running) and `c` (configured out).

Information on the number of CPUs on each node is provided in the `ncpus` field, `oncpus` is the maximum number available and `fcpus` is the number currently free. The number of free CPUs per node is only available if a partition is selected. If the partition argument is NULL then `fcpus` will contain the value `0` for all nodes on return. The `fcpumask` field indicates which CPUs are free. It will have one bit set for each free CPU. The expression `(fcpumask & (1«k))` is non-zero when CPU `k` is free. For example, the value 13 indicates that CPUs 0,2 and 3 are free (1«0)|(1«2)|(1«3).

The `maxmem`, `maxswap` and `maxtmp` fields describe the maximum amounts of memory, swap and temporary file space available on each node. The `freemem`, `freeswap` and `freetmp` fields describe the current amounts of memory, swap and temporary file space available on each node.

The *`options`* argument controls the level of detail provided and, hence, the number of database queries and RPCs generated by the call. The `hostname` is always provided. Fields not provided are set to `0`. The following values for *`options`* are supported:

`NODEINFO_BASIC | 0` — The machine index, the current partition index and the rail mask of each node are provided.

`NODEINFO_STATUS` — The current status of each node is provided and the `drain` field is updated.

`NODEINFO_CPU` — The number of CPUs on each node (`ncpus`), the number potentially available (`oncpus`), the number currently free (`fcpus`) and the free CPU mask `fcpumask` are provided.

NODEINFO_CONF	The parameters maxmem, maxtmp, maxswap, freemem, freetmp and freeswap are returned.
NODEINFO_ALL	All fields are provided.

The *options* values may be ORed together to generate different combinations of the fields.

In general, you should minimize the level of detail returned by this function, especially if it is being called frequently on a large machine.

The drain field is set to 1 if the node is currently being drained of jobs, 0 otherwise. CPUs cannot be allocated on a node in this state. The node will be configured out and the drain field cleared when the jobs using this node complete and their CPUs are deallocated.

RETURN VALUES

Upon successful completion, rms_nodeInfo() returns the number of nodes in the partition (or the whole machine if the partition arg is NULL) provided this is less than *maxnodes*. A negative value is returned on error.

KNOWN PROBLEMS

The current implementation of the RMS API is restricted to nodes with 32 or fewer CPUs per node. This restriction will be removed in future versions, but in resolving this problem the size of the rms_nodeinfo_t structure will increase. Users can protect against this change by allocating 128 bytes for each rms_nodeinfo_t structure. The layout of the existing structure will be preserved with new fields and padding being added to the end.

NAME

rms_partitionInfo – Returns description of a partition

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
int rms_partitionInfo(char *partition, rms_partitioninfo_t *info);
```

PARAMETERS

`partition`	Name of a partition.
`info`	Pointer to a partition description.

DESCRIPTION

The function `rms_partitionInfo()` reads a description of an active partition from the RMS database. The following information is available for each node.

```
typedef struct {
    char baseHost[NAME_LEN];    /* first node in partition */
    int  maxNodes;              /* number of nodes when everything is configured in */
    int  numNodes;              /* number of nodes configured in */
    int  numCpus;               /* number of CPUs configured in */
    int  timelimit;             /* timelimit in seconds (-ve if disabled) */
    int  timeslice;             /* scheduler timeslice in seconds (-ve if disabled) */
    int  mincpus;               /* minimum cpus allocatable (-ve if disabled) */
    int  memlimit;              /* memory limit in megabytes (-ve if disabled) */
    char status[NAME_LEN];      /* current status */
    int  railmask;              /* mask of Elan rails present on all nodes */
} rms_partitioninfo_t;
```

The partition has `maxNodes` nodes of which `numNodes` are configured in at present. The `baseHost` is the name of the first node in the partition (whether configured in or out). The field `nCpus` contains the number of CPUs currently configured in. The field `status` returns the current status of the partition manager.

The rail mask for a partition is the global AND of that for those of its nodes that are configured in. It will take the value `1` if rail `0` is present on all nodes `2` if rail `1` is present on all nodes and `3` if both rail `0` and rail `1` are present on all nodes.

The fields `timelimit`, `timeslice`, `mincpus` and `memlimit` return scheduling parameters for the partition. These fields take negative values if the parameter is not set.

RETURN VALUES

Upon successful completion, `rms_partitionInfo()` returns 0. A negative value is returned on error.

SEE ALSO

`rms_defaultPartition``(3)`

NAME

rms_resourceId – Return the RMS resource ID for a batch job.

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
int rms_resourceId(char *batchid);
```

PARAMETERS

`batchid`	Batch system ID for a parallel job.

DESCRIPTION

`rms_resourceId()` converts the batch system ID for a parallel job into the RMS ID for the resource request. It is typically used with job control operations. For example:

```
rms_suspendResource(rms_resourceId(batchId));
```

Where a resource request fails and is resubmitted with the same batch system ID, `rms_resourceId()` returns the ID of the most recent request.

RETURN VALUES

RMS resource IDs are unique positive integers. A negative return code indicates that the call has failed, generally because the batch system ID is not known to RMS

SEE ALSO

rms_suspendResource(3)

NAME

rms_resourceStats – Return accounting data for a parallel job.

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
int rms_resourceStats(int rid, rms_resourcestats_t *stats);
```

PARAMETERS

`rid`	ID of the resource.
`stats`	Current accounting data for the resource request.

DESCRIPTION

`rms_resourceStats()` returns accounting data for a resource request. The following data is provided:

```
typedef struct {
    int rid;                    /* resource id */
    int uid;                    /* user id */
    char project[16];           /* project name */
    time_t started;             /* time at which CPUs were allocated */
    double etime;               /* elapsed time (secs) */
    double atime;               /* allocated time (cpu secs) */
    double utime;               /* user time (cpu secs) */
    double stime;               /* system time (cpu secs) */
    int cpus;                   /* number of CPUs allocated */
    int mem;                    /* maximum memory allocated */
    int memint;                 /* memory integral */
    int pageflts;               /* number of major page faults */
    int running;                /* CPUs are currently allocated */
} rms_resourcestats_t;
```

The accounting data for a resource request is updated periodically by the Partition Manager. Statistics are summed over all jobs running on an allocated resource and all processes in each parallel job. The `running` field will have a non-zero value while CPUs are allocated.

RETURN VALUES

Upon successful completion, `rms_resourceStats()` returns `0`. A negative value is returned on error.

SEE ALSO

`rms_allocateResource``(3)`

NAME

rms_run – Run a program on an allocated resource

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
int rms_run(int rid, char *cmd, char **args, char *jobFlags);
```

PARAMETERS

`rid`	Resource ID.
`cmd`	Command to execute.
`args`	Arguments for the command.
`jobFlags`	The job flags currently supported are as follows:

`processes=`*n*	Set the number of processes to run. By default, one process is run on each CPU.
`contexts=`*n*	Each process requires *n* contexts.
`cpus=`*n*	Each process requires *n* CPUs.
`signals=0 | 1`	Enable(`1`) or disable(`0`) the forwarding of signals.
`in=`*mode*	Set the mode for standard input. The default is `0`. Valid values are described below.
`one-process-per-node`	Start one process on each node.
`out=`*mode*	Set the mode for standard output. The default is `all`. Valid values are described below.
`err=`*mode*	Set the mode for standard error. The default is `all`. Valid values are described below.
`tag=0 | 1`	With a value of 1, this specifies that output from each process should be tagged by the process ID.
`verbose=`*n*	Set the level of verbose output from the program. Supported values are `0` (quiet), `1` (minimal output), and `2` (full output).

Multiple request flags can be given as a comma-separated list, for example, `tag=1,verbose=1`.

DESCRIPTION

`rms_run()` starts a parallel program on a previously allocated resource. It must be called by the user to whom the resource was allocated.

By default, `rms_run()` executes *`cmd`* with the specified *`args`* on each of the resource's CPUs. The *`jobFlags`* can be used to specify a different distribution of processes.

The *`args`* is an array of character pointers to the arguments themselves, the first member of the array points to a string containing the name of the executable and the last element of the array is `NULL`.

```
char args[3]
args[0] = "uname"
args[1] = "-a"
args[2] = NULL
```

If the resource does not accommodate the distribution of processes requested with the *`jobFlags`*, `rms_run()` returns an error. For example, a call to `rms_run()` requests 2 CPUs per process, but an odd number of CPUs have been allocated on some of the nodes. This call will result in an error.

`rms_run()` inherits the `stdio` descriptors of the calling process. The `mode` job flag can be set to redirect standard input as follows (the effect on standard output and standard error is provided in parentheses):

`rank` Redirect `stdio` to (from) the process identified by *`rank`* where `0` $\leq$ *`rank`* $\leq$ `n-1` and `n` is the number of processes in the program.

`all` Redirect `stdio` to (from) all processes.

`none` Do not redirect `stdio` to (from) any process.

`file` Open the named *`file`* for input (output) and associate it with the `stdio` stream so that each process reads (writes) the file.

`file.%` Expand the `%` character to generate and open for input (output) a separate file name for each process: process 0 reads (writes) `stdio` to (from) *`file`*`.0`, process 1 reads (writes) to (from) *`file`*`.1` and so on.

If the `one-process-per-node` flag is set in `jobFlags` then `rmsloader` will start one process on each node with CPUs allocated. This option should be set for parallel applications where the first process (on each node) forks the others.

By default, `rms_run()` forwards the signals `SIGUSR1`, `SIGUSR2` and `SIGWINCH` to the application. Signal forwarding can be disabled by setting the `signals` job flag to `0`. When signal forwarding is enabled, signals are forwarded according the process's signal mask.

`rms_run()` returns when the application processes have exited or been killed. A process may make multiple calls to `rms_run()` using the same resource.

If the calling process is killed before the application processes have exited or been killed, RMS kills the latter.

RETURN VALUES

Upon successful completion, `rms_run()` returns the global OR of the exit status values of the processes in the parallel program. If one of the processes is killed, `rms_run()` returns a status value of 128 plus the signal number. It can also return the following codes:

`0`	A process exited with the code 127 (`GLOBAL EXIT`), which indicates success, causing all of the processes to exit.
`123`	A process exited with the code 123 (`GLOBAL ERROR`) causing all of the processes to exit.
`125`	One or more processes were still running when the exit timeout expired.
`126`	The resource is inadequate for the request.

SEE ALSO

`rms_allocateResource``(3)`

NAME

rms_suspendResource, rms_resumeResource, rms_killResource, rms_setResourcePriority – Job control operations on allocated resources

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>

int rms_suspendResource(int rid);
int rms_resumeResource(int rid);
int rms_killResource(int rid, int signo);
int rms_setResourcePriority(int rid, int pri);
```

PARAMETERS

`rid`	ID of the resource request.
`signo`	Signal to send.
`pri`	New priority for a resource request.

DESCRIPTION

`rms_suspendResource()` and `rms_resumeResource()` suspend and resume a resource specified by *`rid`*. The caller must be either the owner of the resource or a member of the `rms` group.

`rms_killResource()` sends a signal to all of the processes in all of the jobs running on a specified resource. The caller must be either the owner of the resource or a member of the `rms` group.

`rms_setResourcePriority()` changes the priority of a resource request. The caller must be either the owner of the resource or a member of the `rms` group.

RETURN VALUES

Upon successful completion, `rms_suspendResource()`, `rms_resumeResource()` and `rms_killResource()` return `0`. A negative value is returned on error, for example if

the resource id is invalid or the partition is down. This value can be passed to `rms_errorString()` to obtain a printable description of the error.

SEE ALSO

`rms_allocateResource`(3), **`rms_errorString`**(3)

NAME

rms_version, rms_checkVersion – RMS version information and checking

SYNOPSIS

```
cc [ flag ...  ]  file ...  -lrms -lrmsapi [ library ...  ]
#include <rms/rmsapi.h>
#include <rms/version.h>

char *rms_version();
int rms_checkVersion(char *version);
```

PARAMETERS

`version` The RMS version being compiled with (from `<rms/version.h>`).

DESCRIPTION

The function `rms_version()` returns the version string that an RMS library was built with. The function `rms_checkVersion()` checks that the RMS version an application was compiled with is compatible with that installed the machine where the application is run.

RETURN VALUES

The function `rms_checkVersion()` returns TRUE if the versions are compatible, and FALSE otherwise.

E

Accounting Summary Script

E.1 Introduction

This appendix describes the example accounting summary script included in `/usr/opt/rms/examples/scripts/accounting_summary` and referred to in Section 10.3.

- Section E.2 describes the command line interface.
- Section E.3 shows a sample of output from the script.
- Section E.4 is a listing of the script.

E.2 Command Line Interface

The script has the following command line interface:

```
accounting_summary [ -hd [-u | -p] [-M | -H] ] [days]
```

The options are as follows:

`-h`	Display help on the options.
`-d`	Delete the accounting records of all resource requests that have completed after outputting the accounting summary.
`-u`	Sort the records by user name and then by project name.

`-p`	Sort the records by project name and then by user name. This is the default.
`-M`	Show time in minutes rather than seconds.
`-H`	Show time in hours rather than seconds.
`days`	Show statistics for the specified number of days. By default, statistics are shown for the previous day only.

The script processes the arguments passed to it on the command line and generates a SQL query which it passes to `rmsquery`. The query acts on two tables in the RMS database: the accounting statistics (`acctstats`) table, and the `resources` table. The information returned by the query is formatted to produce output as shown in Section E.3.

If the `-d` option is specified on the command line, after printing the accounting summary, the script generates another SQL query to delete all accounting records that have their `running` field set to 0, denoting that the resource request has completed.

If a query fails, the script outputs an error message.

E.3 Example Output

An example of using the script, together with the output produced, follows. After running the script, all of the accounting records for resource requests that have finished are deleted.

```
# accounting_summary -d

Accounting Summary of Machine atlas at 16:01 Wed 21 Feb 2001
Usage by Project/User For Previous Day

Project    User               CPU          User           Sys  Number
Name       Name              Secs          Secs          Secs Sessions
------------------------------------------------------------------------
default    addy               596           533             6      8
           duncan              58            37             2      6
           johnt              540           227            51     15
           root             29272             2             8     37
           stephen            286            87           134     56
------------------------------------------------------------------------
Total default               30751           885           201    122
------------------------------------------------------------------------
Grand Total                 30751           885           201    122
------------------------------------------------------------------------
```

E.4 Listing of the Script

```
#!/bin/sh
##############################################################################
#
# accounting_summary
#
##############################################################################

usage() {
    echo "Usage : $sname [ -u -p -d [ -M | -H ] ] [ days ]"
}

help() {
   usage
   echo "\t-h\tThis help message"
   echo "\t-p\t'Project' is primary sort field (default)"
   echo "\t-u\t'User' is primary sort field"
   echo "\t-M\tShow time in minutes rather than seconds"
   echo "\t-H\tShow time in hours rather than seconds"
   echo "\t-d\tDelete all 'not running' accounting records after producing summary"
   exit 0
}

#
# main
#
sname="accounting_summary"
OS=`uname`
if [ "$OS" = "Linux" ]; then
    RMSPATH="/usr/bin"
else
    if [ "$OS" = "OSF1" ]; then
        RMSPATH="/usr/opt/rms/bin"
    else
        RMSPATH="/opt/rms/bin"
    fi
fi
RMSGETTIME=${RMSPATH}/rmsgettime
RMSQUERY=${RMSPATH}/rmsquery
tmpfile="/tmp/accounting_summary_$$"
if [ -x /bin/gawk ]; then
    AWK="/bin/gawk"
else
    AWK="/usr/local/bin/gawk"
fi
primary="project"
delete=""
hours=""
minutes=""
```

Listing of the Script

```
#
# parse the options
#

while [ $# -gt 0 ]; do
   option=‘echo $1 | sed "s/^-//"‘
   if [ "$option" = "$1" ]; then
      break
   fi
   if [ "$option" = "p" ]; then
   primary="project"
   elif [ "$option" = "u" ]; then
   primary="user"
   elif [ "$option" = "d" ]; then
   delete="1"
   elif [ "$option" = "M" ]; then
        if [ "$hours" = "1" ]; then
            echo "$sname: ERROR : -M and -H are mutually exclusive"
            exit 1
        fi
   minutes="1"
   elif [ "$option" = "H" ]; then
        if [ "$minutes" = "1" ]; then
            echo "$sname: ERROR : -M and -H are mutually exclusive"
            exit 1
        fi
   hours="1"
   elif [ "$option" = "h" ]; then
   help
   else
      echo "$sname: ERROR : invalid option $1"
      usage
      exit 1
   fi
   shift
done

if [ $# -gt 0 ]; then
    days=$1
    shift
else
    days=1
fi

if [ $# -gt 0 ]; then
    usage
    exit 1
fi

now=‘$RMSGETTIME‘
secsperday=‘expr 60 \* 60 \* 24‘
daysecs=‘expr $secsperday \* $days‘
```

```
starttime=‘expr $now - $daysecs‘

if [ "$primary" = "project" ]; then
    primarytitle="Project"
    secondarytitle="User"
    querystr="select \
    acctstats.project,resources.username, \
    acctstats.atime,acctstats.utime, acctstats.stime \
    from resources,acctstats \
    where acctstats.started > $starttime and resources.name=acctstats.name \
    order by acctstats.project,resources.username"
else
    primarytitle="User"
    secondarytitle="Project"
    querystr="select \
    resources.username,acctstats.project, \
    acctstats.atime,acctstats.utime,acctstats.stime \
    from resources,acctstats \
    where acctstats.started > $starttime and resources.name=acctstats.name \
    order by resources.username,acctstats.project"
fi

machine=‘rinfo -m‘

/bin/rm -f $tmpfile
$RMSQUERY $querystr > $tmpfile
if [ $? -ne 0 ]; then
    echo "$sname : ERROR : $RMSQUERY $querystr FAILED"
    exit 1
fi

cat $tmpfile | \
$AWK ’BEGIN {
    primary     = ""
    secondary   = ""
    nvalues     = 3
    for (i=1; i<=nvalues; i++) {
        values[i]       = 0
        primvalues[i]   = 0
        grandvalues[i]  = 0
    }
    recs        = 0
    primrecs    = 0
    grandrecs   = 0
    printprim   = 1
}

function printsortfields() {
    if (printprim == 1) {
    printf ("%-10.10s %-8.8s ", primary, secondary)
    printprim = 0
    } else {
```

Listing of the Script

```
            printf ("\t   %-8.8s ", secondary)
        }
    }
    function printdashes() {
        printf
       ("-------------------------------------------------------------------\
    ----\n")
    }

    function printvals(vals,   i) {
        for (i=1; i<=nvalues; i++) {
            if (hours == 1 || minutes == 1) {
                printf (" %13.2f", vals[i])
            } else {
                printf (" %13.0f", vals[i])
            }
        }
    }
    NF > 0 {
        if ($1 != primary) {
           if (primary != "") {
               printsortfields()
               printvals(values)
               printf (" %6d\n", recs)
               printdashes()
               printf ("Total %-10.10s    ", primary)
               printvals(primvalues)
               printf (" %6d\n", primrecs)
               printdashes()

            } else {
           datestr = strftime("%H:%M %a %d %b %Y")
           titlestr = sprintf ("Accounting Summary of Machine %s at %s", \
                               machine, datestr)
           print titlestr
           if (days > 1) {
               daystr = sprintf ("%d Days",days)
           } else {
               daystr = "Day"
           }

           printf ("Usage by %s/%s For Previous %s\n", primtitle, sectitle, daystr)
           printf ("\n")
           printf ("%-10.10s %-8.8s       CPU     User       Sys  Number\n", \
                   primtitle, sectitle)
           if (hours == 1) {
                   printf ("Name       Name          Hours    Hours    Hours Sessions\n")
           } else {
               if (minutes == 1) {
                   printf ("Name       Name          Mins     Mins      Mins Sessions\n")
               } else {
```

```
            printf ("Name       Name          Secs     Secs      Secs Sessions\n")
         }
      }
      printdashes()
   }
   primary = $1
   secondary = $2
   for (i=1; i<=nvalues; i++) {
       values[i]       = 0
       primvalues[i]   = 0
   }
   recs      = 0
   primrecs  = 0
   printprim = 1
   } else {
       if ($2 != secondary) {
           printsortfields()
           printvals(values)
           printf (" %6d\n", recs)
           secondary = $2
           for (i=1; i<=nvalues; i++) {
            values[i] = 0
           }
           recs = 0
       }
    }
    for (i=1; i<nvalues; i++) {
        if (hours == 1) {
            val = $(i+2) / 3600
        } else {
            if (minutes == 1) {
                val = $(i+2) / 60
            } else {
                val = $(i+2)
            }
        }
        values[i] = values[i] + val
        primvalues[i] = primvalues[i] + val
        grandvalues[i] = grandvalues[i] + val
    }
    recs++
    primrecs++
    grandrecs++
}
END {
    printsortfields()
    printvals(values)
    printf (" %6d\n", recs)
    printdashes()
    printf ("Total %-10.10s    ", primary)
    printvals(primvalues)
```

Listing of the Script

```
        printf (" %6d\n", primrecs)
        printdashes()
        printf ("Grand Total         ")
        printvals(grandvalues)
        printf (" %6d\n", grandrecs)
        printdashes()
    }' primtitle="$primarytitle" sectitle="$secondarytitle" machine=$machine \
        days=$days hours=$hours minutes=$minutes

    /bin/rm $tmpfile

    if [ "$delete" ]; then
        echo "$sname : Deleting accounting statistics records"
        querystr="delete from acctstats where running=0"
        $RMSQUERY $querystr
        if [ $? -ne 0 ]; then
            echo "$sname : ERROR : $RMSQUERY $querystr FAILED"
            exit 1
        else
            echo "$sname : Accounting statistics records deleted"
        fi
    fi

    exit 0
```

Glossary

Abbreviations

API Application Program Interface — specification of interface to software package (library).

CFS Cluster File System — the file system for Tru64 UNIX clusters.

CGI Common Gateway Interface — a standard method for generating HTML pages dynamically from an application so that a Web server and a Web browser can exchange information. A CGI script can be written in any language and can access various types of data, for example, a SQL database.

CPU Central Processing Unit — the part of the computer that executes the machine instructions that make up the various user and system programs.

CRC Cyclic Redundancy Check — a method of error detection.

CVS Concurrent Versions System — a revision control utility for managing software releases and controlling the concurrent editing of files by multiple software developers.

DIMM Dual In-Line Memory Module.

DMA Direct Memory Access — high performance I/O technique where peripherals read/write memory directly and not through a CPU.

GNU GNU's Not UNIX — A UNIX-like development effort of the Free Software Foundation, headed by Richard Stallman.

HTML HyperText Markup Language — a generic markup language, comprising a set of tags, that enables structured documents to be delivered over the World Wide Web and viewed by a browser.

HTTP HyperText Transfer Protocol — a communications protocol commonly used between a Web server and a Web browser together with a URL (Uniform Resource Locator).

LED Light-Emitting Diode.

MIMD Multiple Instruction, Multiple Data — parallel processing computer architecture characterized as having multiple processors each (potentially) executing a different instruction sequence on different data.

MMU Memory Management Unit — part of CPU that provides protection between user processes and support for virtual memory.

MPI Message Passing Interface — parallel processing API.

MPP Massively Parallel Processing — processing that involves the use of a large number of processors in a coordinated fashion.

PCI Peripheral Component Interconnect — the Elan is connected to a node through this interface.

PDF Portable Document Format — the page description language used by Adobe Acrobat, derived from PostScript, for displaying pages on the screen.

PIO Programmed input/output — I/O directed through a CPU (see also DMA).

PTE Page Table Entry — an entry in the page table which maps the base address of a page to physical memory.

RISC Reduced Instruction Set Computer — a computer whose machine instructions represent relatively simple operations that can be executed very quickly.

RMS Resource Management System — Quadrics software for managing clusters of UNIX nodes.

SDRAM Synchronous Dynamic Random Access Memory — high performance computer memory architecture.

Shmem A one-sided (put/get) inter-process communication interface used on high-performance parallel systems.

SMP Symmetric MultiProcessor — a computer whose main memory is shared by more than one processor.

SNMP Simple Network Management Protocol — a protocol used to monitor and control devices on the Internet.

SQL Structured Query Language — a database language.

TLB Translation Lookaside Buffer — part of the MMU that caches the result of virtual to physical address translations to minimize translation times in subsequent accesses to the same page.

URL Uniform Resource Locator — a standard protocol for addressing information on the World Wide Web.

UTC Coordinated Universal Time[1] — on UNIX systems it is represented as the time elapsed in seconds since January 1st, 1970 at 00:00:00.

Terms

barrier A synchronization point in a parallel computation that all of the processes must reach before they are allowed to continue.

bisectional bandwidth
The worst case bandwidth across the diameter of the network.

critical section A section of program statements that can yield incorrect results if more than one thread tries to execute the section at the same time.

Elan capability An Elan capability describes the user's rights to run a parallel program. It includes information on the distribution of processes over nodes and Elan contexts and a private (random) key .

Elan event A parallel-processing synchronization primitive implemented by the Elan card.

[1]Used to be called GMT.

Elan memory The SDRAM on the Elan card.

Flit A communications cycle unit of information.

HTTP cookies Cookies provide a general mechanism that HTTP server-side connections use to store and to retrieve information on the client side of the connection.

machine The term machine is used as a shorthand for HP AlphaServer SC System. The machine name (for example `atlas`) is the name of the HP AlphaServer SC system and its nodes are named `atlas0`, `atlas1`, ...

main memory The memory normally associated with the main processor, that is to say, memory on the CPU's high speed memory bus.

main processor The main CPU (or CPUs for a multi-processor) of a node, an Alpha™ 21264 for example.

management network

A private network used by the RMS daemons for control and diagnostics.

multirail system

A system that has more than one Elan card connected to each node, each Elan card being connected to a different switch network.

multi-threaded program

A multi-threaded program is one that is constructed such that, during its execution, multiple sequences of instructions are executed concurrently (possibly by different CPUs). Each thread of execution has a separate stack but otherwise they all share the same address space.

node A system with memory, one or more CPUs and one or more Elan cards running an instance of the operating system.

poll Loop and check on each loop whether a specified event has occurred.

rail A network connecting together one adapter in each node. A HP AlphaServer SC Interconnect system may have multiple rails.

rank An integer value that identifies a single process from a set of parallel processes.

reduce Combine the results of a parallel computation into a single value.

remote memory The memory (Elan card or main) of a node when accessed by another node over the network.

resource A set of CPUs allocated to a user to run one or more parallel jobs.

slice A local copy of a global object.

switch network The network constructed from the Elan cards and Elite cards.

thread An independent sequence of execution. Every host process has at least one thread.

virtual memory A feature provided by the operating system, in conjunction with the MMU, that provides each process with a private address space that may be larger than the amount of physical memory accessible to the CPU.

virtual process A (possibly multi-threaded) component of a parallel program executing on a node.

word A 64-bit value on Alpha, IA64 and Opteron platforms, 32-bit on IA32.

Index

A

access controls
  CPU usage, 6-7, 7-3
  memory limits, 6-4, 7-3, 7-5
  priority, 6-6, 7-2
  records, 6-2
  system services, 2-6
  table, 11-8
accounting
  record, 2-10, 6-1, 6-7
  statistics, 11-9
  summarizing data, 10-2, E-1
allocate, 5-3
application node, 2-1
archive, 10-3
attributes
  altzone, 11-13
  auto-configure-out, 11-17
  cleanup-timeout, 11-13
  configure-out-active, 11-17
  cpu-stats-poll-interval, 5-39, 11-16
  default-partition, 11-17
  default-priority, 5-39, 11-17
  event-handler-path, 8-3, 9-7, 11-13
  exit-timeout, 11-17
  fsys-stats-multiplier, 11-16
  grace-period, 5-31, 5-39, 11-17
  inactivity-timeout, 4-3
  inherit-user-limits, 6-5
  lbal-partition, 11-17
  linkerror-event-interval, 11-16
  local-corepath, 9-6, 11-17
  local-filesystems, 11-17
  local-rmsdir, 11-17
  max-alloc, 9-8, 11-17
  modify-time, 11-13
  name-poll-interval, 11-16
  network-level, 11-12
  network-railmask, 11-12
  network-rails, 11-12
  network-type, 11-12
  osarch, 11-13
  partition-blocked–timeout, 11-13
  pmanager-idletimeout, 5-39, 11-17
  pmanager-queuedepth, 5-38, 11-17
  processor-sets, 9-6
  racks, 11-12
  resource-modify-time, 11-16
  rms-alloc, 11-17
  rms-cleanup-signal, 11-17
  rms-hostname-suffix, 9-4, 11-13
  rms-keep-core, 5-39, 9-6, 11-17
  rms-poll-interval, 11-16
  rmshost, 11-13
  set-vm-limit, 6-5
  stats-direct, 9-9, 11-16
  status-modify-time, 11-16
  swmgr-poll-interval, 9-8
  table, 11-12
  units-per-rack, 11-12
  users-to-mail, 8-3

version, 11-13
warning-signal, 5-31

B

backup, 10-5
log files, 11-19
block distribution, 3-1

C

capability, A-1, C-1
commands, 2-5, 5-1
allocate, 5-3
msqladmin, 5-8
nodestatus, 5-11
prun, 5-12
rcontrol, 5-27
rinfo, 5-41
rmsarchive, 5-49
rmsbackup, 5-52
rmsbuild, 5-54
rmsctl, 5-57
rmsexec, 5-59
rmshost, 5-61
rmsloader, 3-3
rmspost, 5-62, 8-2
rmsquery, 5-63
rmstbladm, 5-66
rmswait, 8-2
swctrl, 5-69
configurations, 2-10, 11-32
shift changes, 10-1
core files, 9-6
cyclic distribution, 3-1

D

daemons, 2-4
Database Server (msql3d), 4-2
Event Manager (eventmgr), 4-6
Machine Manager (mmanager), 4-3
Partition Manager (pmanager), 4-3
Process Manager (rmsmhd), 4-7
rmsd, 4-7
Switch Manager (swmgr), 4-5
Transaction Log Manager (tlogmgr), 4-6
database, 2-2, 2-6
administration, 5-66
archiving, 5-49
backing up, 5-52
building, 5-54
field names, 11-1
name, 11-1
restoring, 5-66
SQL interface, 2-6
SQL queries, 5-63
tables, 11-6
Database Server, 4-2
documentation
feedback, 1-3
draining jobs from nodes, 5-34, 10-10

E

Elan, A-1
Elite, A-1
environment variables
RMS_DEBUG, 5-21
RMS_EXITTIMEOUT, 5-21
RMS_IMMEDIATE, 5-21
RMS_IOLABEL, 5-21
RMS_JOBID, 5-23
RMS_KEEP_CORE, 5-21, 9-6
RMS_MEMLIMIT, 5-22
RMS_NNODES, 5-23
RMS_NODEID, 5-23
RMS_NPROCS, 5-23
RMS_NRAILS, 5-22
RMS_PARTITION, 5-22
RMS_PROCFILERANK, 5-23

RMS_PROJECT, 5-22
RMS_RANK, 5-23
RMS_RESOURCEID, 5-23
RMS_RSH_LIMIT, 5-22
RMS_SSH, 5-22
RMS_STATINTERVAL, 5-22
RMS_STDERRMODE, 5-23
RMS_STDINMODE, 5-22
RMS_STDOUTMODE, 5-22
RMS_TIMELIMIT, 5-23
epilogue script, 9-5
Event Manager, 4-6
eventmgr, 4-6
events, 8-1
event handler path, 8-3
handlers, 8-3
mail alerts, 8-3
posting, 8-2
string, 8-1
table, 11-22
waiting, 8-2

G

gang scheduling, 7-1

I

interactive node, 2-1
IP, 9-4

J

jobs
status values, B-2
table, 11-25

K

kernel module, C-1

L

link
errors, 11-26
status values, B-3
load balancing, 5-59
local customization
allocatable memory, 9-8
core files, 9-6
event handling, 9-7
partition startup, 9-5
prologue/epilogue scripts, 9-5
switch manager configuration, 9-7
log files, 11-44

M

Machine Manager, 4-3
machine name, 5-55
Macine Manager
inactivity timeout, 4-3
management functions, 2-3
access control, 2-10
accounting, 2-10, 6-7, 11-9
resource allocation, 2-7
scheduling, 2-8, 7-1
management server, 2-2
memory limits
access controls, 6-4
environment variables, 5-22
scheduling, 7-5
UNIX, 6-5
msql3d, 4-2
msqladmin, 5-8

N

network
external, 2-1
management, 2-2
nodes, 2-1, 11-28

status values, B-3
switch, 2-2, 4-5
node statistics
table, 11-31
nodes
application, 2-1
configuring in and out, 5-33, 10-8
draining, 5-34
draining jobs, 10-10
interactive, 2-1
naming, 5-55
statistics, 11-31
status, 4-3
status values, B-5
table, 11-28
nodestatus, 5-11

P

parallel program, 3-1
Partition Manager, 4-3
partitions, 2-7, 4-3
root, 2-8
scheduling, 2-8
startup, 9-5
status values, B-6
table, 11-32
pfloader, 5-12
pmanager, 4-3
priority, 7-2
process file, 5-12, 5-16
Process Manager, 4-7
processor sets, 9-6
project, 2-10, 6-1
default, 6-1
membership, 6-2
specifying, 11-46
table, 11-34
prologue script, 9-5
prun, 5-12
pstartup, 9-5

R

rail, A-4
rcontrol, 5-27
resource request, 3-2
resources
allocation, 2-7, 5-3
cleanup, 10-9
status values, B-6
table, 11-35
restore, 10-7
rinfo, 5-41
RMS
installation planning, 9-1
kernel module, C-1
local customization, 9-4
node names, 9-2
system call interface, C-2
RMS API calls
rms_allocateCpus, D-4
rms_allocateHosts, D-7
rms_allocateResource, D-9
rms_checkVersion, D-39
rms_closeDatabase, D-12
rms_createResource, D-13
rms_deallocateResource, D-15
rms_defaultPartition, D-17
rms_errorString, D-19
rms_freeCpus, D-20
rms_freeCpusPerNode, D-20
rms_getNodeStats, D-21
rms_jobIds, D-23
rms_killJob, D-24
rms_killResource, D-37
rms_machineName, D-17
rms_nodeInfo, D-26
rms_numCpus, D-17
rms_numNodes, D-17
rms_partitionIndex, D-17
rms_partitionInfo, D-29
rms_resourceId, D-31

rms_resourceStats, D-32
rms_resumeResource, D-37
rms_run, D-34
rms_suspendResource, D-37
rms_version, D-39
rms_whichPartition, D-17
RMS operation
accounting data, 10-2
archiving, 10-3
backup, 10-5
configuring in and out, 10-8
log files, 10-11
restore, 10-7
shift changes, 10-1
troubleshooting, 10-7
rmsarchive, 5-49
rmsbackup, 5-52
rmsbuild, 5-54
rmsctl, 5-57
rmsd, 4-7
rmsexec, 5-59
rmshost (command), 5-61
rmshost (hostname alias), 2-2
rmsloader, 3-3, 4-8
rmsmhd, 4-7
rmspost, 5-62, 8-2
rmsquery, 5-63
rmstbladm, 5-66
rmswait, 8-2

S

scheduling
algorithm, 2-9
batch, 7-2
complex requests, 7-7
constraints, 7-2
CPU usage, 7-3
draining, 7-6
gang, 7-1
idle time, 7-7
log, 11-36
memory limits, 7-3, 7-5
parameters, 2-8
policies, 2-8, 7-1
preemptive, 2-9
priority, 7-2
queue, 7-4
restricted, 7-2
suspending jobs, 7-7
swap space, 7-6
timelimit, 7-3
timesharing, 7-1
timeslicing, 7-7
services
table, 11-38
software products
table, 11-39
status
table, 11-40
status values
environment, B-4
generic, B-2
job, B-2
link, B-3
network, B-3
node, B-5
partition, B-6
resource, B-6
transaction, B-7
swap space, 7-6
swctrl, 5-69
Switch Manager, 4-5, 9-7
switch modules, A-3
table, 11-42
switch network
adapters, A-4
barrier synchronization, A-4
boards, 11-41
control interface, 4-5, 5-69, 11-21
crosspoint switch, A-1
Elan, A-1

elans table, 11-20
Elite, A-1
elites table, 11-21
fat tree network, A-1
level, A-1
links, A-3
multistage network, A-1
plane, A-1
rail, A-4
reduction, A-4
swmserver, 9-7
top switch, A-3
uplinks, A-2
swmgr, 4-5
swmserver, 9-7
system calls
rms_elaninitdone, C-16
rms_getcap, C-12
rms_getcorepath, C-3
rms_getprgid, C-6
rms_getpset, C-15
rms_ncaps, C-12
rms_prgaddcap, C-10
rms_prgcreate, C-4
rms_prgdestroy, C-4
rms_prggetstats, C-13
rms_prgids, C-6
rms_prginfo, C-6
rms_prgresume, C-8
rms_prgsignal, C-8
rms_prgsuspend, C-8
rms_setcap, C-10
rms_setcorepath, C-3
rms_setpset, C-15

T

tables
access controls, 11-8
accounting statistics, 11-9
archive tables, 11-11
attributes, 11-12
backups, 11-19
elans, 11-20
elites, 11-21
event handlers, 11-21
events, 11-22
fields, 11-23
installed components, 11-24
jobs, 11-25
link errors, 11-26
module types, 11-27
node statistics, 11-31
nodes, 11-28
partitions, 11-32
projects, 11-34
resources, 11-35
scheduler log, 11-36
servers, 11-37
services, 11-38
software products, 11-39
status, 11-40
switch boards, 11-41
switch modules, 11-42
terminal server types, 11-43
terminal servers, 11-43
transactions, 11-44
tree status, 11-45
users, 11-45
terminal servers
table, 11-43
types table, 11-43
tlogmgr, 4-6
Transaction Log Manager, 4-6
transactions
table, 11-44
troubleshooting, 10-7

U

user commands, 2-5
users

table, 11-45