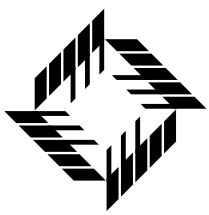
The Distributed Job Manager Administration Guide

AHPCRC

Minnesota **Supercomputer** Center, Inc.



Second Edition.

Copyright © 1993, Research Equipment Incorporated, d/b/a Minnesota Supercomputer Center, Inc. This material may be reproduced by or for the U.S. Government pursuant to the copylight license under the clause at DFARS 252.227-7013.

Connection Machine® is a registered trademark of Thinking Machines Corporation.

This package includes software developed by the University of California, Berkeley and its contributors.

Contents

1 Introduction **1**

1.1 Features . . . 1

1.2 Supported Machines . . . 1

1.3 How to Obtain DJM . . . 1

2 Theory of Operation **2**

2.1 Functional Layout . . . 2

2.2 The Life Cycle of a Job . . . 3

3 Administrative Duties **4**

3.1 Compile and Install the DJM programs . . . 4

3.2 Starting, Stopping, and Recovering DJM . . . 4

3.2.1 Starting DJM . . . 4

3.2.2 Stopping DJM . . . 5

3.2.3 Recovering DJM . . . 5

3.3 Partitions . . . 6

3.3.1 Creating a Partition Schedule . . . 6

3.3.2 Partition Modes . . . 7

3.3.3 Realization of Partitions . . . 7

3.3.4 The -kill Parameter . . . 8

3.3.5 Dedicated Partitions . . . 8

3.4 Daily Housekeeping . . . 9

3.5 How to Take a Snapshot of the Trace Buffer . . . 10

3.6 The Log File . . . 11

4 Local Customization **12**

4.1 Scores . . . 12

4.1.1 The Score Equation . . . 12

4.1.2 The Score Equation for Queued Jobs . . . 13

4.1.3 The Score Equation for Running Jobs . . . 13

4.1.4 How to Change the Score Factors . . . 15

4.2 The Concept of Soft Limits . . . 15

4.3 Limiting Foreign Access to the System . . . 15

4.4 Queues . . . 16

4.4.1 How to Create a New Queue . . . 16

4.4.2 How DJM Selects a Queue for a Job . . . 16

4.5 Gates . . . 17

5 The Distributed Job Manager Reference Manual 18

5.1 The djm.conf File . . . 18

5.1.1 The sitename Statement . . . 18

5.1.2 The machine Statement . . . 19

5.1.3 The master Statement . . . 19

5.1.4 The server Statement . . . 19

5.1.5 The client Statement . . . 20

5.1.6 The tracelog Statement . . . 20

5.1.7 The spool_dir Statement . . . 20

5.1.8 The cookie_file Statement . . . 21

5.1.9 The default_path Statement . . . 21

5.1.10 The default_umask Statement . . . 21

5.1.11 The djm_shepherd Statement . . . 21

5.1.12 The cknfs Statement . . . 21

5.1.13 The fixpath Statement . . . 22
5.1.14 The first_gid and last_gid Statements . . . 22
5.1.15 The sendmail Statement . . . 23
5.1.16 The contact Statement . . . 23
5.1.17 The cmattach Statement . . . 23
5.1.18 The cmts_shutdown Statement . . . 23
5.1.19 The start_ts Statement . . . 23
5.2 The Job File . . . 25
5.2.1 An example of the Job File . . . 25
5.2.2 Explanation of the Parameters . . . 26
5.3 The Partitions File . . . 30
5.3.1 An Example of the Partitions File . . . 30
5.3.2 Explanation of the Parameters . . . 31
5.4 The Gates File . . . 33
5.4.1 An example of the Gates File . . . 33
5.4.2 Explanation of the Parameters . . . 34
5.5 The Queue File . . . 35
5.5.1 An Example of the Queue File . . . 35
5.5.2 Explanation of the Parameters . . . 36
5.6 The Global File . . . 38
5.6.1 An Example of the Global File . . . 38
5.6.2 Explanation of the Parameters . . . 38

6 Other Sources of Information **40**

6.1 The Thinking Machines Corporation DJM Reference Manual . . . 40
6.2 Internet Mailing List: djm@msc.edu . . . 40

1 Introduction

The Distributed Job Manager[†] is a job control system that manages the execution of jobs on a massively parallel processor (MPP). It provides a comprehensive set of tools for administering an MPP system. It was originally developed at the Minnesota Supercomputer Center.

This document is intended to be read by system managers who need to know how to configure and operate the Distributed Job Manager.

1.1 Features

The Distributed Job Manager is upward compatible with the Network Queuing System (NQS), at the user level. All of the user-visible commands of NQS, e.g., qsub, qstat, and qdel, are supported. Support is also provided for many of the user-visible extensions to NQS, as proposed in the Posix Batch Queuing draft standard, IEEE Std 1003.15.

The unique features of DJM are interactivity and robustness. Interactive access is integral to the system. DJM makes no distinction between a "batch" job and an "interactive" job. A job is a login session, and jobs can be detached from the terminal and then later re-attached, possibly to a different terminal. If a partition crashes, DJM will reroute queued jobs to other Control Processors, balancing the load across the available partitions. The criteria for selecting where to run jobs can be adjusted by site policy. All of the database information is stored in human-readable form, in simple ASCII text files, and can be viewed or modified with a text editor.

1.2 Supported Machines

DJM supports the Thinking Machines Corporation (TMC) Connection Machine 2 (CM-2) and the Connection Machine 5 (CM-5). You are encouraged to donate your extensions for support of other architectures for incorporation into future DJM releases. DJM also supports the Silicon Graphics IRIS, and Sun workstations running Solaris 2, as clients.

1.3 How to Obtain DJM

The DJM package can be obtained from the CM LIGHTNING User Group FTP archives. Use anonymous ftp to access host ec.msc.edu, and retrieve the file /pub/LIGHTNING/djm_*ver*.tar.Z. See the INSTALL file for installation instructions. You need an ANSI C compiler, preferrably gcc version 2 or later.

DJM is free software under the auspices of the GNU copyright. However, it has NO WARRANTIES, EXPRESS OR IMPLIED.

[†]DJM was funded, in part, by the United States Army Research Office, under contract #DAAL03-89-C-0038.

Thinking Machines Corporation bundles their own version of DJM with the CM Operating System. That version is *not* covered by the GNU copyright; it is proprietary software and may not be redistributed without the permission of TMC.

2 Theory of Operation

2.1 Functional Layout

DJM consists of four daemons: djm_master, djm_fingerd, djm_execd, and djm_shepherd, and one user interface program, job (see Figure 1).

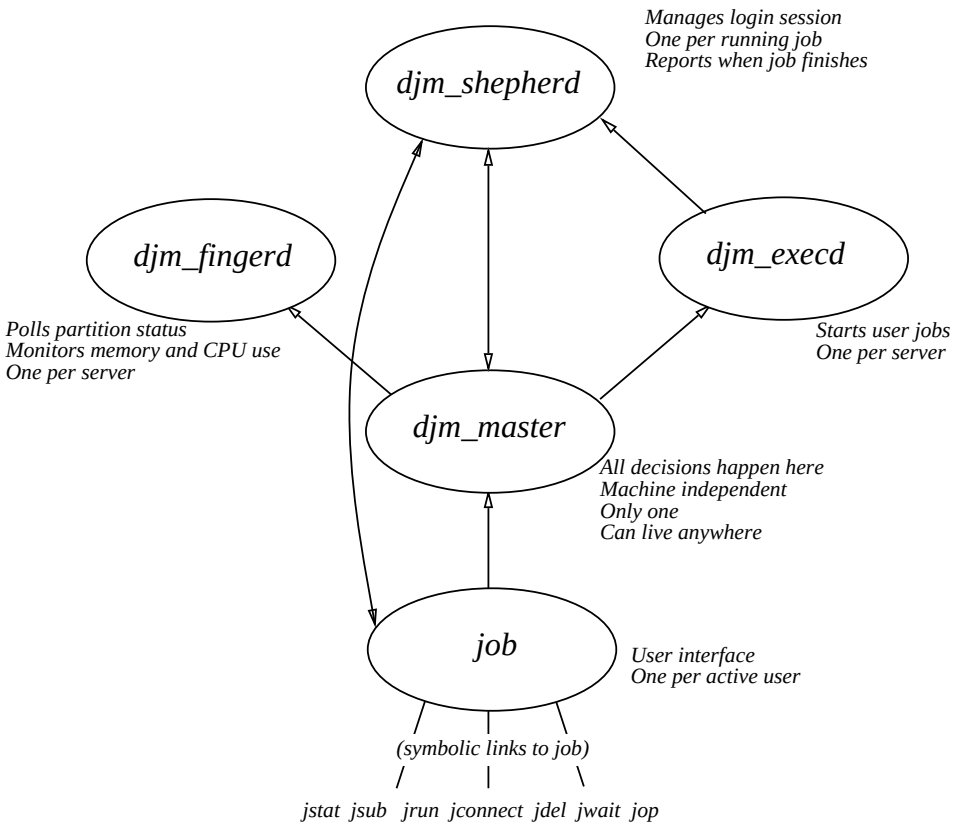


Figure 1: DJM consists of four daemons and a user interface program.

Djm_master is the primary control daemon. It does the brain work of the system, including all job scheduling. All other programs interface to this daemon. It can run anywhere and does not need to be located on a CM Control Processor[†].

[†] A "Control Processor" is TMC lingo for a Sun front-end workstation that has a hardware interface to the Connection Machine. It is also sometimes called a Scalar Processsor, because the serial portion of a CM program runs on it. It is also sometimes called a Partition Manager, because it executes the timesharing daemon that divides up the CM partition into time slices, one for each program.

Djm_fingerd represents the eyes of ears of DJM. It runs on each Control Processor, periodically scanning the CP to see what physical partitions are available and what jobs are running. It reports its findings back to djm_master.

Djm_execd represents the arms and legs of DJM. Its purpose is to start new jobs, to kill old jobs, and to send send messages to jobs, e.g., warning them of imminent shutdown.

One copy of djm_shepherd is run for each job. Its purpose is to launch the user's program, and then to report back to djm_master when the user's program finishes. It also manages the pseudo-tty (pty) for interactive users, in a manner similar to the Berkeley rlogin daemon.

"Job" is the user interface for DJM. It allows the user to issue commands, such as to submit a job. It communicates with djm_master to submit the new job. Djm_master responds with an access token that allows the user to connect to djm_shepherd, in order to establish an interactive session.

Job has several commands, such as, connect, delete, hold, kill, oper, query, release, rerun, show, signal, status, submit, and wait. These commands can be abbreviated, for example, "job stat." They can be also abbreviated by creating a symbolic link, for example, "jstat." The first letter of the symbolic link is ignored, and the remaining letters used to form the command, for example, "qstat." This is how NQS aliases are created for DJM commands.

2.2 The Life Cycle of a Job

The life cycle of a job begins when a user submits a job request using jsub or jrun. When the request is submitted, the job program sends a request packet to djm_master. Djm_master examines the request. If approved, it assigns a job ID number and creates a job file in the directory /usr/spool/djm, using the ID number as the file name. The job file contains information such as the path name of the user's program, and the amount of CPU and memory resources requested.

Djm_master periodically scans the MPP system, looking for a partition that has available resources to run the job. It polls all of the djm_fingerd daemons, one per Control Processor, at at intervals of between 15 seconds and 60 seconds. The djm_fingerd daemons go out and query the MPP hardware, and return a list of currently running jobs and the amount of free memory remaining.

When djm_master locates a partition that has sufficient free memory, it sends a copy of the job file to the corresponding djm_execd daemon and instructs it to launch the job. (Note that only djm_master needs access to the files in the /usr/spool/djm directory; the other programs get all of their information directly from djm_master.) Djm_execd in turn forks and execs a djm_shepherd daemon in order to monitor the execution of the job. Djm_shepherd locates an available pseudo-tty (pty), opens it, and prepares it for interactive use. It then forks and execs the user's program, and then waits for the program to exit. When the program exits, djm_shepherd reports back to djm_master.

During job execution, djm_master continues to monitor the process tables of the Control Processors. If a job uses too much resources, it sends a KILL signal to the job's processes, via djm_execd. When all of the processes associated with the job have exited, djm_master retires the job file, moving it to the directory /usr/spool/djm/old.

3 Administrative Duties

The two major phases in the installation of the Distributed Job Manager on a new machine are initial setup and local customization. Initial setup is where you create necessary shell scripts and operator commands that are required to operate DJM on a daily basis. Local customization is where you adapt DJM to your local environment and site policies for controlling and monitoring access to your MPP system.

To initially set up the Distributed Job Manager, you need to perform the following steps:

- Compile and install the DJM programs
- Learn how to start, stop, and recover DJM
- Create an initial set of partitions
- Write a cron script for daily housekeeping

In addition, you should learn how to get a trace dump, for debugging purposes.

3.1 Compile and Install the DJM programs

Read the INSTALL file in the DJM distribution kit for instructions on how to compile and install the DJM programs on your Control Processors and on your master host.

3.2 Starting, Stopping, and Recovering DJM

3.2.1 Starting DJM

You need to edit your boot script, /etc/rc.local, on each machine, to start the daemon programs. On the master host, you need to start djm_master. You provide djm_master with the name of the MPP system that it is supposed to manage. A list of MPP systems is found in the djm.conf file. The name is passed to the program as a single argument on the command line. For example, if the name of your CM-5 is "godzilla," then you would add the following to your /etc/rc.local file on your master host:

```
/usr/local/etc/djm_master godzilla ; (echo -n ' djm_master') > /dev/console
```

On each Control Processor, you need to start djm_execd and djm_fingerd.

```
/usr/local/etc/djm_execd ; (echo -n ' djm_execd') > /dev/console
/usr/local/etc/djm_fingerd ; (echo -n ' djm_fingerd') > /dev/console
```

It doesn't matter in what order the daemons are started.

To create a partition, first make sure that the scheduler is frozen. Type **jop freeze**. This ensures that no jobs are accidentally started on the partition while it is coming up. Then type **jop create partition -mode normal -range** *m-n* **-unlock**, where *m-n* is a range of physical processing nodes that mimics a list of processing nodes reported by the cmpartition command. On the CM-2, this will start the timesharing daemon, unless you specify **-notimeshare**. On the CM-5, DJM does not yet actually run the cmpartition command to create physical partitions, nor does it actually start the timesharing daemon, so you will need to do both of these by hand.

Once the partition is created, you can verify it by typing **jstat partitions**. If you make a mistake, retype the command (when you create a new partition, it automatically deletes any overlapping ones). Finally, type **jop unfreeze**. This will unfreeze the scheduler, and DJM will resume the scheduling of jobs on the partition.

3.2.2 Stopping DJM

To prevent new jobs from being started, type **jop freeze**. Jobs that are already running will be allowed to finish. To kill all of the jobs running on a partition, type **jop modify partition -mode shutdown -kill -range** *m-n* **-lock**[†]. The partition will be permanently halted — the shutdown persists across reboots. On the CM-2, this will terminate the timesharing daemon. DJM does not yet actually terminate the timesharing daemon on the CM-5, nor does it actually run cmpartition to update the physical partition information in /etc/cm/partitions.current, so you will need to do both of these by hand, if necessary.

3.2.3 Recovering DJM

First of all, before you worry about recovering DJM, make sure that it really needs to be recovered. If you type **jstat** and get no immediate response, it does not necessarily mean that DJM is hung. Djm_master can sometimes wait for up to 2 minutes for a response from a slow Control Processor. Long delays can also occur when the network connection is bad. After 2 minutes with no answer from a Control Processor, djm_master will mark it offline and thereafter wait only 5 seconds per poll until the server is back online. DJM can also run slower if the primary Domain Name Server (DNS) is down, or if the Network Information Server (NIS) is down.

You can investigate why DJM is slow by taking a snapshot of the trace buffer (explained later). Rarely, you may need to kill djm_master and restart it by hand. Djm_master checkpoints itself to the spool directory, /usr/spool/djm, every 5 minutes. It also checkpoints itself whenever you freeze the scheduler (**jop freeze**). Therefore, to ensure that all of the state information is current, you should try to freeze the scheduler before you kill djm_master, whenever possible.

Djm_master reads the checkpoint files when you start the daemon, or when you unfreeze the scheduler (**jop unfreeze**). If you need to edit the files with a text editor, you should only do so when djm_master is not running, or when the scheduler is frozen. Otherwise, your changes may be accidentally overwritten by the next checkpoint.

[†] Actually, **modify** is a synonym for **create**. When you "modify" a partition you are really creating a new partition that replaces the old one.

3.3 Partitions

A *partition* is a contiguous set of processors that have a common set of attributes. On a CM-5, a partition is associated with a Control Processor. On a CM-2, a partition is associated with a Front End Bus Interface (FEBI). A partition is created with the command **jop create partition**.

On the CM-2, DJM also provides a special type of partition, called a non-timeshared partition. A non-timeshared partition can float between Control Processors. DJM supports *dynamic reconfiguration* of non-timeshared partitions: the job scheduler will automatically redivide the processors as required by job load. Multiple sequencers will automatically be combined into a single large partition to run a big job, and then be split up into smaller partitions in order to run small jobs. Dynamic reconfiguration is not yet supported on the CM-5 (TMC needs to provide job checkpointing first).

3.3.1 Creating a Partition Schedule

You can use *cron* to run a nightly shell script to divide up the processors into partitions, based on a schedule. An example of a such a shell script is given below.

```
#!/bin/sh
export PATH
PATH=/bin:/usr/bin:/usr/ucb:/usr/lbin
LOG=/tmp/djm.reconfig.log
rm -f $LOG
date > $LOG
job -noprompt <<EOF > $LOG 2>&1
op freeze
op delete partitions   # erase the old image of the schedule
#
# The 32 node partition is the same every day.
#
op creat part -at '8am' -range 256-287 -mode normal -nokill \
   -soft_memcpu '16MB*1hr'
op creat part -at '10pm' -range 256-287 -mode normal -nokill
#
# Other partitions vary weekly.
#
op creat part -at 'Sun 3pm' -range 512-1023 -mode ded -kill
# Down for preventative maintenance
op creat part -at 'Mon 7am' -range 0-1023 -mode shutdown -kill
op creat part -at 'Mon noon' -range 256-287 -mode normal -nokill \
   -soft_memcpu '16MB*1hr'
op creat part -at 'Mon noon' -range 512-639 -mode normal -nokill
op creat part -at 'Mon noon' -range 640-767 -mode normal -nokill
op creat part -at 'Mon noon' -range 768-1023 -mode normal -nokill
op creat part -at 'Wed 10pm' -range 768-1023 -mode ded -kill
# Down for preventative maintenance
op creat part -at 'Thu 7am' -range 0-1023 -mode shutdown -kill
```

```
op creat part -at 'Thu noon' -range 256-287 -mode normal -nokill \
   -soft_memcpu '16MB*1hr'
op creat part -at 'Thu noon' -range 512-1023 -mode normal -nokill
#
# Put the new partitions into effect.
#
op unfreeze
EOF
```

This example reflects the actual weekly schedule for a 544 node CM-5, managed in Minnesota. The 32 node partition has a schedule that is the same every day. During the day, jobs are limited to 16 megabyte-hours (i.e., the product of a job's requested memory and requested CPU time cannot exceed 16 MB*hr). The limit is soft, meaning that if a job should accidentally exceed the limit, it will not be killed unless another job is queued waiting to run.

The remaining 512 nodes undergo a weekly cycle. They are combined into a single 512 node partition from Thursday, 7am, through Monday, 7am. They are split into 128, 128, and 256 nodes respectively during the remaining time. 256 nodes are run in dedicated mode from Wednesday, 10pm, through Thursday, 7am. 512 nodes are run in dedicated mode from Sunday, 3pm, through Monday, 7am. The entire machine goes down for preventative maintenance twice a week, on 7am, Monday, and again on 7am, Thursday.

Once again, note that DJM does not actually modify the physical partitions. You need to write a separate script for doing the actual adjustments. Such a script should bracket its activity with **jop freeze** and **jop unfreeze** commands.

3.3.2 Partition Modes

A partition can be set in one of three modes.

- *Normal* — inform the scheduler to run all jobs that meet the limits of the partition. The scheduler runs as many jobs as will fit in memory. Jobs that specify the -dedicated option are not selected.
- *Dedicated* — inform the scheduler to run only one job at a time. Each job has access to all of the resources of the partition, without competition. This allows large production jobs to be run with guaranteed access to resources. Only jobs that specify the -dedicated option are selected.
- *Shutdown* (also *halted*) — inform the scheduler not to run any jobs on the partition.

3.3.3 Realization of Partitions

A partition is *realized* when its activation time has arrived, and the scheduler is unfrozen. When a partition is initially created, it is unrealized. Such a partition is called a *future* partition, to distinguish it from a realized, or *current*, partition. The job scheduler looks only at current partitions in deciding where to run a job. Future partitions are held in a queue, sorted by their activation times. You can delete a future

partition with the command **jop delete partition** *id*, where *id* is the ID number of the future partition as shown by the jstat command[†].

DJM does not (yet) change the physical partitions in the /etc/cm/partitions.current file, called *run* partitions, to conform with its view of the current partitions. Therefore it is possible to have a discrepancy between the range of nodes shown for a run partition and that shown for the current partitions. When this happens, the scheduler applies the following rule: the resource limitations of *all* partially overlapping current partitions are applied to a run partition in the most *restrictive* manner.

Consider the following example. A run partition contains 512 nodes (say, cmpartition -pn_range 256-767). Two current partitions overlap, one with range 0-511 and a second one with range 512-1023. The first partition has a hard CPU limit of 4 hours (-hard_cpu 4hr), while the second partition has a mem*cpu limit of 3 megabyte-hours (-hard_memcpu 3mb*hr). In this case, the scheduler applies both the 4 hour CPU limit *and* applies the 3mb-hr mem*cpu limit, to the run partition.

3.3.4 The -kill Parameter

Before a new partition is realized, the -kill parameter determines if DJM will first terminate all of the running jobs that occupy the processor range. This is useful, for example, to clear the machine in preparation for dedicated mode, or to shut down the machine in preparation for preventative maintenance. The -kill parameter influences the scheduler in two ways. First, a warning message is broadcast to the jobs at T minus 30, 5, and 1 minutes, to warn of the impending kill. Second, queued jobs that ask for more CPU time than the time remaining before the kill are blocked from starting. This prevents jobs from being killed before they can finish. Users can inform the scheduler that don't care if their jobs finish, by specifying the -interruptible option. Users who do their own checkpointing will benefit by specifying this option, because as the hour of reckoning approaches, the machine will be gradually drained of other jobs. Conversely, a job that requests a long stretch of uninterrupted CPU time will be at a disadvantage. A user who submits an "impossible" job, one that requests more uninterrupted CPU time than will be available in any scheduled future partition, will never be able to run at all. In such cases, the user will receive a warning message when the job is submitted.

You should always specify -kill if you plan to bring down the machine. Otherwise, uninterruptible jobs will be killed accidentally, and without warning. You should also create future partitions far in advance, with at least as much lead time as your published "guaranteed" time for uninterrupted machine availability. Typically this at least 12 hours, and usually much longer.

Some day, when TMC provides checkpoint capability, DJM will checkpoint jobs instead of killing them. Until then, this is the best it can do.

3.3.5 Dedicated Partitions

A dedicated partition allows you to run only one job at a time. All such jobs must specify the -dedicate option, indicating that the want exclusive access to the resources of the partition. Typically you will create

[†]You cannot delete a current partition; you can only replace it by realizing a new future partition on top of it. When a future partition is realized, and it partially overlaps with one or more current partitions, it replaces *all* of the partially overlapping current partitions.

a single large partition that spans the entire machine. Dedicated jobs are placed into a special dedicated queue (see *Queues*). A dedicated partition has special scheduling rules.

- Only one job at a time may run.
- The job's -memory option is ignored. The job has access to all of the memory in the partition.
- The job must include the -dedicated option.
- Any foreign jobs started in the partition are killed immediately with an explanatory message.
- If no jobs are running, the partition is automatically converted back to normal mode, after a grace period of typically 10 minutes (see the parameter dedicated_grace_period in the global file).
- On a CM-2, the dedicated partition is continually subdivided to reduce the size of the partition to match the size of the remaining dedicated jobs. The cleaved portions are converted to normal mode partitions.
- Dedicated jobs must be submitted at least 4 hours in advance (see the parameter dedicated_submit_by_limit). Otherwise, the dedicated period is cancelled. The dedicated partition is converted into a normal partition. The -kill option is ignored if the formerly dedicated partition has the same attributes as the corresponding current partition.

The above rules mean that dedicated partitions are volatile, in that they can spontaneously revert back to normal partitions. You can suppress the volatility by adding the **-lock** option. This allows you to provide "guaranteed" dedicated time, at the risk of leaving the machine idle if the dedicated jobs finish early.

Because of the volatility of dedicated partitions, you should not set the **-kill** option on a normal partition that follows it. This is because the dedicated partition may have spontaneously reverted to normal mode. Instead, the scheduler provides a special term in the score equation to penalize dedicated jobs that run "late," running_dedicated_late_factor. If the machine is idle, the dedicated job will continue to run until the first user submits a normal job. When the first normal job is submitted, the negative factor ensures that the dedicated job will be killed.

3.4 Daily Housekeeping

You need to perform some simple daily tasks in order to prevent old files from clogging the spool directory, and to update the internal list of future partitions. These tasks can be accomplished with a cron script. You should run the cron script once each day. Your script needs to do the following.

- Purge old job files from /usr/spool/djm/old.
- Trim the log file, /usr/spool/djm/log.
- Refresh the in-memory image of partition schedule.

An example of such a cron script is given below.

```
#!/bin/sh
export PATH
PATH=/usr/ucb:/bin:/usr/bin:/etc:/usr/etc:/usr/local/etc
DIR=/usr/spool/djm
# maximum number of lines to retain in the log file
NLINES=3000
# number of days to keep old job files
NDAYS=4

djm_reconfig.sh  # refresh the partition schedule

cd ${DIR}

# Trim the log file down to NLINES
OLINES=`wc log | awk ' { print $1}'`
if [ ${OLINES} -gt ${NLINES} ]; then
        # watch for no space on /tmp
        if tail -${NLINES} log > /tmp/trim$$
        then
            cat < /tmp/trim$$ > log
        else
            echo "unable to trim $i."
        fi
        rm -f /tmp/trim$$
fi

# Purge stale job files
find old -mtime +${NDAYS} -print | xargs rm

exit 0
```

In this example, we first call a secondary script, djm_reconfig.sh, to update the internal image of the partition schedule. This is the script presented earlier, in the section on Partitions. At our site, we make it a separate script, so that we can run it by hand when we've changed in the partition schedule.

The script trims the log file, /usr/spool/djm/log, to keep it from getting too large. Change the NLINES parameter to the number of lines that you want to keep. Finally, the script purges old job files from the directory /usr/spool/djm/old, that are older than 4 days. Change the NDAYS parameter if you want to retain the old job files for a different time period. The old job files contain the information that is necessary to permit you to rerun a previously completed job (**job rerun** *id*).

3.5 How to Take a Snapshot of the Trace Buffer

Every daemon, as well as the user interface program (**job**), keeps an internal log of it's most recent activity. The log is called a *trace buffer*. Sometimes it is useful to get a snapshot of the trace buffer. Examination of the snapshot, with the source code in hand, can help you to get a better understanding of the internal operation of DJM. It will also help you in trouble-shooting configuration problems.

Snapshots of the trace buffer are written to a file in the /tmp directory. The name of the trace file is /tmp/*name*.trace, e.g., /tmp/djm_fingerd.trace. To get a snapshot of a program, send it a SIGQUIT signal. You can do this with the command **kill -3** *pid*, where *pid* is the process ID number of the program. The program will intercept the signal, and write out a copy of its trace buffer. The program will not exit.

There are two other ways to get a trace buffer. First, the command **jop dump_trace** will cause all the daemons to dump their trace buffers simultaneously. Second, if a program encounters a fatal internal error, it will *panic*: it will dump its trace buffer, print a warning message on the console, and then exit.

Below is an excerpt of a trace buffer snapshot from the djm_master daemon.

```
18:50:01 MSG recv_tcp_msg(sock=15,host=c1.msc.edu)
18:50:01 MSG data=f7fffc88, dlen=348
18:50:01 EXEC do_notify_request jid=44951, exitcode=9
18:50:01 MSG send_tcp_msg(sock=15,host=c1.msc.edu)
18:50:01 MSG data=f7fffc0c, dlen=4
18:50:01 EXEC setting exitcode 9 jid 44951
18:50:01 JOB sync_job_state
18:50:01 SCHED update_resource_use
18:50:01 SCHED no pids left, job 44951
18:50:01 LOG log: FINISHED: job 44951 finished, exitcode 0009
18:50:01 LOG log: FINISHED: user mxy0303, file job32, project unknown.
18:50:01 LOG send_to_job 44951
18:50:01 USER annotate job 44951 host c1.msc.edu
18:50:01 MSG send_tcp_msg(sock=6,host=c1.msc.edu)
18:50:01 MSG data=f7fff918, dlen=37
18:50:01 MSG recv_tcp_msg(sock=6,host=c1.msc.edu)
18:50:01 MSG data=f7fff608, dlen=516
18:50:01 LOG send_to_job 44951
18:50:01 JOB delete_job 44951 (674a8)
18:50:01 LOG log: DELETE: job 44951 deleted.
18:50:01 JOB remove_job 44951 (674a8)
```

3.6 The Log File

DJM records all information in a log file, /usr/spool/djm/log. The log file contains a record of each major event in the life cycle of a job. An example is shown below.

```
Nov 24 18:43:30 SUBMIT: user mxm0305, cdir /wrk2/mxm0305, file runmesh
Nov 24 18:43:30 SUBMIT: requested_nproc 32, requested_cputime 600,
    requested_memory 81920
Nov 24 18:43:30 SUBMIT: jid 19226, state Que, pri 50, dedicated no,
    exact_mem no, exact_proc no
Nov 24 18:43:30 SUBMIT: queue default
Nov 24 18:43:32 RUN: job 19226, user mxm0305, server e1.msc.edu,
    interface 0, nproc 32, baseproc 256
Nov 24 18:43:32 RUN: score 92792, cputime 600, mem 81920, project unknown
```

```
Nov 24 18:59:20 FINISHED: job 19226 finished, exitcode 0000, cputime 393,
    max_memory 79034.
Nov 24 18:59:20 FINISHED: user mxm0305, file runmesh, project unknown,
    state Exiting.
Nov 24 18:59:20 DELETE: job 19226 deleted.
```

In this example, the user mxm0305 submitted a job that requested 32 processors, 600 seconds of CPU time, and 81,920 kilobytes of memory. The current directory for the job is /wrk2/mxm0305, and the name of the file to execute is runmesh. The job ID number is 19226, and the job priority is 50. The job is not dedicated, and it does not ask for an exact number of processors[†]. The job runs two seconds later, on the CP named e1.msc.edu. It has 32 processors, and the first physical processor is at NI address 256. The job's score is 92,792. The job finished at 18:59, consuming 393 seconds of CPU time and 79,034 kilobytes of memory.

4 Local Customization

By default, DJM is configured to provide a simple FIFO (first in, first out) queuing policy. This section describes how you can change the behavior of the job scheduler, so you can adapt DJM to meet your site's administrative needs.

4.1 Scores

Job scheduling is based on the concept of *scores*. Each job is assigned a numeric score. The job with the highest score gets to run first.

Scores are recalculated at periodic intervals, about once per minute. Many factors go into the calculation of a job score. For example, the longer a job is queued, the higher its score becomes (*queued_wait_factor*). Running jobs also have scores. If the score of a queued job rises to the point where it exceeds the score of a running job, the running job is killed to make room for the queued job. (For this reason, running jobs should usually have high scores.) When a running job exceeds its memory request or its CPU request, its score is greatly reduced, increasing the likelyhood that it will be killed to make room for a queued job.

4.1.1 The Score Equation

A score is computed by summing several terms. Each term consists of a variable, multiplied by a constant factor. By tuning the constant factors, you can change the sum, thus affecting the score. For example, one variable is *number_of_seconds_queued*. It is multipled by the constant factor *queued_wait_factor*. By changing the factor, you can change how number_of_seconds_queued will influence the score.

[†] "Exact_proc" means that DJM will not run the job on a partition that has more processors than the number requested, should a partition with the exact number not be available.

There are two different sums, one computed for queued jobs, and another one computed for running jobs. Each sum has a different set of variables and constant factors. The constant factors are double-precision floating point numbers.

4.1.2 The Score Equation for Queued Jobs

The score of a queued job is the sum of the following terms:

- *queued_base_factor*. This is a constant added to the scores of all queued jobs.
- *queued_wait_factor* * number_of_seconds_queued. This changes the score in proportion to how long the job has been waiting to run. Usually set to 1.
- *queued_nproc_factor* * number_of_processors. This changes the score in proportion to how many processors are requested. Usually zero or slightly negative.
- *queued_cputime_factor** number_of_cpu_seconds_requested. This changes the score in proportion to how much CPU time the job requested. Usually zero or negative.
- *queued_memory_factor* * number_of_kilobytes_of_memory. This changes the score in proportion to how much memory the job requested. Usually zero or slightly negative.
- *queued_mem_cpu_product_factor** number_of_kilobytes_of_memory * number_of_cpu_seconds_requested. This changes the score in proportion to the product of the requested memory and CPU time. Usually zero or slightly negative.
- *queued_pri_factor** user_job_priority. This changes the job score in proportion to the -priority option indicated on the job command line. The default priority is 100. Minimum and maximum priority limits can be set for each queue.
- *queued_already_queued_factor* * njobs_already_queued. This changes the job score in proportion to the number of other jobs already queued by the user. Usually negative, the default value is -3600. The purpose of this factor is to prevent a single user from hogging the machine by submitting a large number of jobs.
- *queued_already_queued_cputime_factor** sum_of_cpu_seconds. This changes the score in proportion to the sum of the requested CPU time of all other jobs already queued by the user.
- *queued_already_queued_memory_factor* * number_of_kilobytes_of_memory. This changes the score in proportion to the sum of the requested memory of all jobs already queued by the user.
- *queued_already_queued_mem_cpu_product_factor** number_of_kilobytes_of_memory * number_of_cpu_seconds. This changes the score in proportion to the sum of the product of the requested memory and the CPU time of all other jobs already queued by the user.

4.1.3 The Score Equation for Running Jobs

The score of a running job is the sum of the following terms:

- *running_base_factor*. This is a constant added to the scores of all running jobs. This should be a large value, bigger than the score of any queued job under normal conditions.
- *running_nproc_factor* * number_of_processors. This changes the score in proportion to how many processors are in use. Usually zero.
- *running_memory_factor* * number_of_kilobytes_of_memory. This changes the score in proportion to how much memory is in use. Usually zero.
- *running_cputime_exceeded_factor*. This is a constant added to the score if the job exceeded the job's requested CPU time or the partition's soft CPU limit. Usually a large negative value.
- *running_memory_exceeded_factor*. This is a constant added to the score if the job exceeded the job's requested memory limit or the partition's soft memory limit. Usually a large negative value.
- *running_mem_cpu_product_exceeded*. This is a constant added to the score if the job exceeds partition's soft limit on the product of the memory times the cpu time (kilobyte-seconds). Usually a large negative value.
- *running_dedicated_late_factor*. This is a constant added to the score if the job is dedicated but has overrun its nightly time dedicated time slot. This allows dedicated jobs to continue to run into the morning if the system is idle. Usually a large negative value.

The final sum of a score is constrained to the range 1 through 999,999,999. Use the command **jstat scores** to see the instantaneous score calculation for each job. Look for jobs that have unexpectedly high or low scores. This may be an indication that one of your constant factors is out of tune. Badly turned constant factors will not be apparent to your users until the machine becomes heavily loaded.

An example of the actual score factors for the *default* queue, on the CM-5 located in Minnesota, is given below.

```
queued_base_factor                          100000
queued_wait_factor                          1
queued_nproc_factor                         -0.25
queued_pri_factor                           0
queued_cputime_factor                       0
queued_memory_factor                        0
queued_mem_cpu_product_factor               0
queued_already_queued_factor                -3600
queued_already_queued_cputime_factor        0
queued_already_queued_memory_factor         0
queued_already_queued_mem_cpu_product_factor 0
running_base_factor                         1e+07
running_nproc_factor                        0
running_memory_factor                       0
running_cputime_exceeded_factor             -2e+07
running_mem_cpu_product_exceeded_factor     -2e+07
running_memory_exceeded_factor              -2e+07
running_interactive_factor                  1000
running_dedicated_late_factor               -2e+07
```

4.1.4 How to Change the Score Factors

Each queue has its own private set of constant score factors. The queue's score factors determine the priority of the jobs within the queue, and also determine the priority of the jobs with respect the jobs located in other queues[†]. The constant factors used in score calculations are kept in the queue's data file, /usr/spool/queue_*name*, where *name* is the name of the queue. To change the score factors, first freeze the job scheduler (**jop freeze**). Then edit the queue data files with a text editor. When you are done, unfreeze the job scheduler (**jop unfreeze**). The job scheduler will then recalculate all of the job scores based on the new constants. Type **jstat scores** to see the results of your handiwork.

4.2 The Concept of Soft Limits

The scheduling rules of Network Queuing System (NQS) are more restrictive than those of DJM. NQS imposes a system of *hard* limits. When a job, running under NQS, consumes more CPU time than it had originally requested, NQS will immediately kill it. It will do so even if the machine would be otherwise idle. DJM, on the other hand, provides a system of *soft* limits. DJM will kill a job only if it is necessary to allow jobs with higher scores to run.

When you submit a job to DJM, you are creating a contract between yourself and the system. You promise not to use more than n units of resources, and in exchange the system promises to run your job to completion. If you break your promise, the system has the right to kill your job, if necessary, in order to free up resources for other people.

Users who are accustomed to running under NQS may be surprised by soft limits, if the machine is lightly loaded. For example, a user might accidentally submit a runaway job, one that never finishes. Under NQS the hard limit would have prevented the user from accidentally exhausting his/her account allocation (over a holiday weekend, perhaps). Under DJM, the job will run forever, or at least until the machine becomes more heavily loaded[‡].

4.3 Limiting Foreign Access to the System

DJM cannot prevent people from starting CM processes outside of DJM. But it will notice such processes after they start. A process that runs outside of the control of DJM is called a *foreign job*. It is important for you to limit the amount of CPU and memory that foreign jobs can consume. Otherwise, DJM will not be able to schedule jobs effectively. Foreign jobs tend to rudely "butt ahead" of queued jobs, penalizing polite users who have waited, possibly a long time, to run. If they consume too much memory, foreign jobs can cause other jobs to crash, by overflowing the available physical memory or the available disk swap space.

The global file, /usr/spool/djm/global, contains parameters that let you control foreign jobs. You can set soft limits on foreign jobs with the parameters foreign_requested_cputime and foreign_requested_mem_per_proc.

[†] The term "queue" is actually somewhat of a misnomer, because jobs can be selected out of order. Jobs are selected solely on the basis of their scores.

[‡] To avoid this problem, DJM allows a user to specify explicit hard limits (**-hard_cputime** and **-hard_memory**).

You can set hard limits with foreign_hard_cputime and foreign_hard_mem_per_proc. The recommended values for these parameters on a CM-5 are given below.

`foreign_requested_cputime`	0	(unlimited)
`foreign_requested_mem_per_proc`	8192	(8 megabytes per processor)
`foreign_requested_total_mem`	0	(unlimited)
`foreign_hard_cputime`	300	(5 minutes)
`foreign_hard_mem_per_proc`	8192	(8 megabytes per processor)
`foreign_hard_total_mem`	0	(unlimited)

The key parameter is hard_cpu_time. This allows users to bypass DJM to run short benchmarks or to debug a program without having to use DJM. In no case should this value be less than 120 (2 minutes), because DJM cannot reliably identify the foreign-ness of a new job in less than that amount of time. A value of 0 means unlimited.

Use of soft limits for foreign jobs is not recommended. Experience has shown that new users who are ignorant of DJM often are surprised after they become accustomed, over many days, to running long foreign jobs, only to find that suddenly their jobs no longer complete successfully. This tends to generate more complaints than if a hard limit was enforced from the first day.

4.4 Queues

As mentioned earlier, a *queue* groups together jobs that have the same constant score factors. When you install DJM, two queues are automatically created: the default queue and the foreign queue. The default queue receives all jobs submitted to DJM, and the foreign queue receives all jobs run outside of DJM. Both are initially set up to provide a simple FIFO (first in, first out) queuing policy, and both have equal score factors. The factors that penalize jobs are set to zero, so jobs are never killed if they exceed their CPU or memory limits. You will want to change these factors to large negative numbers in a production environment.

4.4.1 How to Create a New Queue

The easiest way to create a new queue is to copy the data file of an existing queue and then modify it. First, freeze the scheduler (**jop freeze**). Then copy a queue data file, for example, /usr/spool/djm/queue_default, to a new file, /usr/spool/djm/queue_*name*, where *name* is the name of your new queue. You can then edit the parameters in the new data file. When you are finished, type **jop unfreeze**. Djm_master will see the new data file and read the parameters. The new queue is now ready for use.

4.4.2 How DJM Selects a Queue for a Job

When a job is submitted, DJM scans the queues, looking for one that will accept the job. It scans the queue with the lowest search_order first, and then scans queues with progressively higher values until one is found that meets all of the user-specified criteria (memory, nproc, cputime, and PN range) and that meets all of the access restrictions (user id, group id, per-user submit limit, and the total submit limit). If no queue is found that is willing to accept the job, the job is rejected.

You should assign the lowest search_order to the most restrictive queues. The *default* queue should have the highest search_order and have the least restrictive memory and CPU limits. Most DJM jobs will find a suitable queue before being placed in the *default* queue, and DJM will be able to manage the job mix accordingly.

4.5 Gates

A *gate* allows you to modify the behavior of the scheduler at fixed points during the life cycle of a job. When a job reaches a gate, it triggers the execution of an external program, written by you. Your program can examine the job and make one of three choices.

1. Your program can accept the job, allowing it to pass through the gate.
2. Your program can reject the job.
3. Your program can adjust the score of the job.

There are three types gates. The type determines under what circumstances the gate is checked. A *required* gate will reject the job if the job does not explicitly request the gate by specifying -need *name* (or its synonym, -gate *name*). An *optional* gate will ignore the job unless the job explicitly requests the gate, again with -need or -gate. An *implicit* gate will check the job regardless of the -need or -gate option.

You can use the parameters -queue, -range, and -server to select which jobs are checked by the gate. Gates are created and realized just like partitions, at scheduled times. For more information on the syntax for creating a gate, see the man page for **jop create gate**.

DJM inspects the last line of the output of your program. If the line begins with "yes" or "ok," the job is accepted. If the line consists of an integer, the score of the job is adjusted upward (if positive) or downward (if negative). Any other output is construed by DJM as a rejection of the job, and the output is sent to the user as an explanatory message.

You can trigger a gate at any of six locations along the life cycle of a job. These locations are called *events*. You assign a list of events that will trigger a gate, using the -events parameter. An event is one of the following.

- *submit* — Trigger the gate when the job is submitted. If you reject the job, the output from your program will be sent to the user as an explanatory message. An integer value will bias the job's base queue score.
- *queued* — Check the gate periodically while the job is queued. The gate is checked no more frequently than -min_interval (default 5 minutes). If you reject the job, it is placed in BLOCKED state and not allowed to run. An integer value will bias the job's base queue score.
- *foreign* — Trigger the gate when a new foreign job is detected. If you reject the job, the foreign process is killed, and the output from your program is sent as an explanatory message. An integer value will bias the job's base running score.
- *start* — Trigger the gate just before the job begins execution. If you reject the job, it goes back on the queue. (This can be very expensive, because a start-event has no minimum interval. You can

also deadlock the scheduler if it is draining the partition to make room for the job.) An integer value will bias the job's base running score.

- *running* — Check the gate periodically while the job is running. The gate is checked no more frequently than -min_interval (default 5 minutes). If you reject the job, it is killed and the output from your program is sent as an explanatory message. An integer value will bias the job's base running score.
- *finished* — Trigger the gate when the job finishes. This allows you to collect accounting statistics. Obviously, it makes no sense for you to reject the job at this point!

When a gate is triggered, it will execute a program, written by you. (The program can also be a shell script, if you set the executable permission bit and insert #!/bin/sh in the first line of the file.) Your program receives two arguments. The first argument is the path name of the data file containing the job's state information. This is usually /usr/spool/djm/*nnnnn*, where *nnnnn* is the job ID number. The second argument is one of *submit, queued, start, foreign, running,* or *finished*, representing the event that triggered the gate.

Your program should return a zero exit code, indicating that it executed successfully. A non-zero exit code indicates a system error. In this case, the output, if any, is flagged and printed on the console as a system error message. The job will then be rejected if the -reject_if_error flag is set.

Clearly, gates are more flexible than constant score factors. However, they are more complicated to set up, and they incur more overhead.

5 The Distributed Job Manager Reference Manual

This chapter catalogs all of the approximately 200 configurable parameters that are available to you for customizing DJM to meet the needs of your local site.

5.1 The djm.conf File

The DJM Configuration file, /usr/local/etc/djm.conf, contains the fundamental information on the nature of your system. It lists, for example, the number of nodes on your MPP. This file needs to be installed on all hosts, including all client hosts.

5.1.1 The sitename Statement

The sitename Statement contains the name of your site. It is printed in a banner on the first line of the output of each job. You should specify the site name here, 32 characters maximum.

```
sitename  AHPCRC / Minn. Supercomputer Ctr
```

5.1.2 The machine Statement

A single site can support multiple MPP systems. Each MPP is defined in a `machine` statement.

```
#         name       CM_name   type    kbmem/pn   nproc   min_nproc
machine   godzilla   none      cm5        32768    1024          32
machine   octacube   CM-MSC    cm200        128   32768        8192
```

The first argument is the name of the MPP, as DJM knows it. This is sometimes referred to as the DJM Machine Name. Users can specify this name on the -machine option of the **jsub** command line, to select which MPP system will receive the job. The second argument is the name of the Connection Machine, as specified in the configuration.lisp file, on CM-2 systems. It is ignored on other MPP systems. The third argument is the machine type, one of cm2, cm200, or cm5. The fourth argument is the memory size of your machine, measured in kilobytes of memory per processing node. On a CM-2, note that 1024 kilobits/pn is represented as 128 kbytes/pn. The fifth argument is the number of processors. On a CM-2, it represents the number of single bit processors. A "full gallon" CM-2 has 65,536. On a CM-5, it represents the number of address spaces in your Data Network. A For example, a Scale 3 CM-5 system with 32 PNs has 64 address spaces. The final argument is the size of the smallest partition that can be created. It represents a single sequencer on a CM-2, or the smallest possible partition on a CM-5.

5.1.3 The master Statement

The master statement declares which host will run djm_master, the master daemon.

```
master  godzilla  cm5-compile.msc.edu
master  octacube  cm2fe-1.msc.edu
```

Each MPP system must have exactly one master host. The first argument is the DJM Machine Name. The second argument is the fully qualified domain name of the master host. In this example, cm5-compile is a compile server, and is not itself a Control Processor for the CM-5, although you can have a CP also be the master host if you wish.

5.1.4 The server Statement

The server statements list each of the front ends, or Control Processors. These are the hosts that have the actual hardware interfaces that connect to the MPP system.

```
server  godzilla  cm5-1.msc.edu
server  godzilla  cm5-2.msc.edu
server  godzilla  cm5-3.msc.edu
server  godzilla  cm5-4.msc.edu
server  godzilla  cm5-5.msc.edu
server  octacube  cm2fe-1.msc.edu   0-1
server  octacube  cm2fe-2.msc.edu   0-1
```

The first argument is the DJM Machine Name. The second argument is the fully qualified domain name of the host. In this example, the CM-5 has five servers, and the CM-2 has two. The third argument is required only if your server has more than one hardware interface; it specifies a range of hardware interfaces that are available for use by DJM. In this example, the CM-2 servers each have two interfaces, 0 and 1.

5.1.5 The client Statement

The client statements list each of the hosts that are permitted to submit jobs to DJM. The hosts listed here will be able to run the commands **job**, **jop**, **jsub**, **jstat**, **jrun**, etc. You should make sure that the User ID numbers listed in the password entries in /etc/passwd on each client host are identical to those on the server hosts. If a client can submit jobs to multiple MPP systems, you should list the default MPP system first.

```
client  godzilla  cm5-compile.msc.edu
client  godzilla  cm5-1.msc.edu
client  godzilla  cm5-2.msc.edu
client  godzilla  cm5-3.msc.edu
client  godzilla  cm5-4.msc.edu
client  godzilla  cm5-5.msc.edu
client  octacube  cm2fe-1.msc.edu
client  octacube  cm2fe-2.msc.edu
client  godzilla  bashful.msc.edu
client  godzilla  happy.msc.edu
client  godzilla  grumpy.msc.edu
client  godzilla  sleepy.msc.edu
client  octacube  bashful.msc.edu
client  octacube  happy.msc.edu
client  octacube  grumpy.msc.edu
client  octacube  sleepy.msc.edu
```

5.1.6 The tracelog Statement

The tracelog statement specifies the path name for the file to receive a copy of trace log. Trace logs are written whenever a daemon crashes or when a daemon receives a SIGQUIT signal.

```
tracelog    /tmp/%s.trace
```

The %s is substituted with the name of the daemon.

5.1.7 The spool_dir Statement

The spool_dir statement specifies the name of the spool directory. Job state data is saved in this directory on the master host. Core files are written into this directory on the server hosts. Client hosts do not need a spool directory.

```
spool_dir  /usr/spool/djm
```

5.1.8 The cookie_file Statement

The cookie_file statement specifies the name of the cookie file. To verify the authenticity of a TCP/IP message, each daemon requires that it receive a correct authentication key from the remost host. An authentication key is constructed by merging an internal key, compiled into each DJM program, with an external key, located in the cookie file. The file should have root-only read permission. The file should contain a single ASCII decimal number of approximately 8 to 10 digits, terminated by a newline character. The cookie file needs to be installed on all hosts, including all client hosts.

```
cookie_file   /usr/local/etc/djm.cookie
```

5.1.9 The default_path Statement

The default_path statement specifies the initial shell executable search path when a job is started. This parameter goes into the PATH environment variable.

```
default_path  /usr/ucb:/bin:/usr/bin:/usr/lbin:
```

5.1.10 The default_umask Statement

The default_umask statement specifies the initial umask. The umask controls the permission bits for newly created files.

```
default_umask     027
```

5.1.11 The djm_shepherd Statement

The djm_shepherd statement specifies is used by the djm_execd daemon to locate the djm_shepherd program on the server host.

```
djm_shepherd  /usr/local/etc/djm_shepherd
```

5.1.12 The cknfs Statement

Sun's Network File System (NFS) has a design flaw when run on NFS client machines that mount filesystems with the **-o soft** option. A soft mount point is supposed to allow the caller to cancel a disk I/O operation if the NFS server is offline. However, the chdir() system call, if used to access a dead NFS server, will

cause the caller to hang forever, and cannot be cancelled. This misfeature is detrimental to the reliability of DJM, as the master daemon, for example, cannot afford to be hung for long periods of time.

To avoid this problem, DJM can make use of a special program, called **cknfs**, that will ping the NFS server to verify that it is alive. The cknfs statement specifies the location of the cknfs utility. The cknfs utility is available on all major Sun source archive FTP sites, such as ftp.uu.net. It was posted as free software on Usenet, to the comp.sources.sun newsgroup, in 1989, under the catalog number V01i033. If you do not have cknfs, do not include this statement.

Note: cknfs is ineffectual when dealing with an automounter.

```
cknfs   /usr/local/bin/cknfs   # optional
```

5.1.13 The fixpath Statement

Sun's Automounter has a design flaw. When a directory is accessed via the automounter, the automounter prepends a temporary mount point to the front of the directory name. This causes the **pwd** command to return an incorrect name for the current working directory. To avoid this problem, you need to provide a shell script that will strip the garbage prefix off the front of the path name. The prefix will vary depending on the type of automounter you use, and on where the automounter decides to place it's temporary mount points.

```
fixpath   /usr/local/bin/fixpath   # optional
```

A sample script would look something like this:

```
#!/bin/sh
sed 's/\/tmp_mnt//' << EOF
$1
EOF
```

5.1.14 The first_gid and last_gid Statements

DJM identifies a job by adding a unique group ID number to each process in the job. To do this, DJM needs a pool of reserved group ID numbers that it can allocate from. These two statement specify a range of group IDs that are reserved for use by DJM. Make sure that no group ID in this range is listed anywhere in the /etc/group file on any of your hosts.

```
first_gid  31000
last_gid   32000
```

5.1.15 The sendmail Statement

The sendmail statement gives the path name for the sendmail utility. DJM uses the sendmail utility to send e-mail messages to users concerning their jobs.

```
sendmail    /usr/lib/sendmail
```

5.1.16 The contact Statement

If DJM encounters a problem running a user's job, it will instruct the user to notify the person listed in in the contact statement.

```
contact    djm-help@mysite.edu
```

5.1.17 The cmattach Statement

On a CM-2, DJM needs to run a utility, cmattach, to attach the user's process to the Connection Machine. DJM provides a replacement utility for cmattach, called jattach. The jattach utility implements the preemption option (-P) which was removed from cmattach in CMSS 6.1.2. Preemption is necessary so that DJM can override the kernel wait queue. Otherwise, DJM can be blocked by foreign jobs that use cmattach -w. The jattach utility also avoids a bug in cmattach so that a CM-2 job can checkpoint itself on receipt of a signal during system shutdown. If, however, you insist on using the stock cmattach utility, place it below and also edit config.h to remove the USE_PREEMPT symbol.

```
cmattach   /usr/local/bin/jattach
```

5.1.18 The cmts_shutdown Statement

On a CM-2, DJM needs to run a shell script to shut down the timesharing system. The cmts_shutdown statement specifies the path to the script.

```
cmts_shutdown  /usr/local/etc/cmts-shutdown
```

The script is included in the CM-2 CMSS software distribtion.

5.1.19 The start_ts Statement

On a CM-2, DJM needs to run a shell script to start the timesharing system. The start_ts statement specifies the path to the script. A sample script would look something like this:

```
#!/bin/sh
/usr/local/etc/run-ts-daemon -w dvwd dv1: -maxprocs 16 -procsize 1021 \
   -clientnice 0 -checksum 2 -suspendtime 180 $*
```

The script must not specify an interface or a sequencer (-i or -S), because these arguments will be passed to the script by djm_execd. If the symbol USE_PREEMPT is defined in config.h, DJM will also pass the -P (preempt) option to the script.

5.2 The Job File

The job file contains all of the information that is associated with a particular job. There is one job file for each job in the system. The job file has the name /usr/spool/djm/*nnnnn*, where *nnnnn* is the job's unique ID number.

5.2.1 An example of the Job File

```
jid                     19568
uid                     2014
sid                     10333
state                   Starting
queue_name              default
uname                   kevin
file                    myscript
jobname                 myscript
cdir                    /u1/kevin/mydir
stdout_name             myscript.o19568
shell_name              /bin/csh
project_name            myproject
term_name               xterm
ttyname                 /dev/ttyp1
pri                     50
when_queued             754369878
when_run                754369880
gid                     31435
groups                  4 :  200 2 25 150
active_pid              10333
active_cputime          3
active_memory           29696
active_when_started     754369889
accum_cputime           0
max_memory              29696
is_timeshared           yes
actual_nproc            32
actual_baseproc         256
requested_cputime       3600
requested_memory        32768
requested_nproc         32
requested_baseproc      -1
requested_server_name   c1.msc.edu
hard_cputime            7200
hard_memory             0
want_timeshared         yes
mail_user               kevin@msc.edu
mail_begin              no
mail_end                yes
dedicated               no
rerun                   0
```

```
safety                10
sp_disk               0
sp_memory             0
ucode                 0
server_name           c1.msc.edu
server_interface      0
argv                  myscript
argv                  myarg1
argv                  myarg2
need_argv             hippi
need_argv             sda
hastty                yes
jrun                  yes
capture               no
echo                  yes
onlcr                 yes
linelimit             9999
inet_port             2448
credentials           675061094
exitcode              0
exact_mem             no
exact_proc            yes
interruptible         yes
njobs_running         1
njobs_queued_ahead    0
cputime_queued_ahead  0
memory_queude_ahead   0
submit_gate_factor    0
queued_gate_factor    0
foreign_gate_factor   0
start_gate_factor     0
running_gate_factor   0
```

5.2.2 Explanation of the Parameters

The meaning of the parameters is described below.

- *jid* is the job's ID number, assigned by DJM.
- *uid* is the user's ID number, as found in the password file.
- *sid* is the job's session ID number, as reported by **ps -j**.
- *state* is one of Queued, Held, Blocked, Running, Starting, Suspended, Signaled, Exiting, Dying, or Dead, respectively.
- *queue_name* is the name of the queue that owns the job.
- *uname* is the name of the user.

- *file* is the name of the program or script to be executed.
- *jobname* is the job's name. This is the same as the file, unless changed with the -jobname option.
- *cdir* is the current working directory.
- *stdout_name* is the name of the file that will receive the job's output.
- *shell_name* is the name of the shell program that will execute the user's script.
- *project_name* is used for accounting purposes. It is passed to the MPP accounting system. It sets the environment variable DJM_PROJECT.
- *term_name* is the name of the terminal. It sets the environment variable TERM.
- *ttyname* is the name terminal device.
- *pri* is the job's priority, as set by the -priority option.
- *when_queued* is the time when the job was queued (submitted), measured in seconds since 00:00 GMT, January 1, 1970,
- *when_run* is the time when the job was run.
- *gid* is the unique group ID number, assigned by DJM. DJM uses the group ID number to locate all of the processes associated with the job.
- *groups* is the list of group IDs numbers to which the user belongs. The number before the colon (:) indicates how many group IDs follow.
- *active_pid* is the process ID number of the active process — the process that is currently running on the MPP.
- *active_cputime* is the CPU time consumed by the active process, measured in seconds.
- *active_memory* is the current amount of MPP memory being used by the active process, measured in kilobytes.
- *active_when_started* is the time when the active process started.
- *accum_cputime* is the cumulative CPU time consumed by all other processes prior to the current active process.
- *max_memory* measures the peak amount of MPP memory used during the execution of the job.
- *is_timeshared* is "yes" if the job is running on a timesharing daemon (CM-2 only).
- *actual_nproc* is the number of processing nodes actually in use.
- *actual_baseproc* is the ordinal of the first processing node in use. Together, actual_nproc and actual_baseproc identify a range of processing nodes.
- *requested_cputime* is the amount of CPU time requested by the user.
- *requested_memory* is the total amount of MPP memory requested by the user.
- *requested_nproc* is the number of processing nodes requested by the user.

- *requested_baseproc* is the ordinal of the first processing node requested by the user. This allows the user to force the job to run on a particular partition. -1 means "don't care."
- *requested_server_name* is the name of the server host. This allows the user to force the job to run on a particular host.
- *hard_cputime* is the hard limit for CPU time.
- *hard_memory* is the hard limit for total MPP memory use.
- *want_timeshared* is "yes" if the user wants to restrict the choice of partitions to those that have a timesharing daemon (CM-2 only).
- *want_notimeshared* is "yes" if the user wants to restrict the choice of partitions to those that do not have a timesharing daemon (CM-2 only).
- *mail_user* is the address to receive e-mail messages.
- *mail_begin* is "yes" if DJM should send an e-mail message to the user when the job begins.
- *mail_end* is "yes" if DJM should send an e-mail message to the user when the job ends.
- *dedicated* is "yes" if the job requires dedicated use of the partition.
- *rerun* is the maximum number of times to rerun the job after an abnormal termination.
- *safety* is the level of safety checking that the OS should perform (0 to 10). It sets the CMOS_SAFETY environment variable.
- *sp_disk* is the amount of serial disk space requested by the user. Serial disk space is provided by the Control Processor.
- *sp_memory* is amount of serial memory requested by the user. Serial memory is provided by the Control Processor.
- *ucode* is the requested microcode level. It sets the -ucode parameter on the cmattach command line (CM-2 only).
- *server_name* is the name of the server host that is running the job.
- *server_interface* is the ordinal of the hardware interface on the server host.-
- *argv* is a list of command arguments, one per line. These are passed to the user's program or script.
- *need_argv* is a list of needed gates, as set by the -need or the -gate option.
- *hastty* is "yes" if the job has a pseudo-tty.
- *jrun* is "yes" if the job was launched from a **jrun** command.
- *capture* is "yes" if the output stream should be copied to the output file while the job is connected to the terminal.
- *echo* is "yes" if keystrokes are echoed back to the user.
- *onlcr* is "yes" if a newline character is mapped to a carriage return / newline pair on the output stream.
- *linelimit* is the maximum number of lines to send to the output file.

- *inet_port* is the Internet port number that the jconnect command uses to reestablish an interactive session.
- *credentials* are the encrypted authentication credentials for the job. The jconnect command decrypts the credentials and passes them to djm_shepherd in order to reestablish an interactive session.
- *exitcode* is the number returned from the exit() system call by the last process in the job. A zero indicates success.
- *exact_mem* is "yes" if the job demands access to all of the requested MPP memory, even if it is appears that it is using far less actual memory. This option is for jobs that grow slowly in size over a long period of time. Because of the potential for harming the system, a user must be specially validated to use the -exact_mem option.
- *exact_proc* is "yes" if the job demands access to a partition with exactly the number of requested processors. Normally, DJM will attempt to run the job on a larger partition, if a partition with the exact number of processors is not available.
- *interruptible* is "no" if the job demands uninterrupted access to the machine. "Yes" informs the scheduler that it is safe to start the job despite the fact it would be killed later by a scheduled shutdown.
- *njobs_running* is the number of other jobs owned by the same user that are also running. This parameter is not updated after the job begins execution.
- *njobs_queued_ahead* is the number of other jobs owned by the same user that are also queued, at the time the job was submitted.
- *cputime_queued_ahead* is the sum of the requested CPU time of all other jobs owned by the same user that are also queued, at the time the job was submitted.
- *memory_queude_ahead* is the sum of the requested memory of all other jobs owned by the same user that are also queued, at the time the job was submitted.
- *submit_gate_factor* is a constant bias factor that is added to the job's queued score. It is obtained from the execution of a gate program at the time the job was submitted.
- *queued_gate_factor* is a constant bias factor that is added to the job's queued score. It is obtained from the most recent execution of a gate program while the job is queued.
- *foreign_gate_factor* is a constant bias factor that is added to the job's running score. It is obtained from the execution of a gate program at the time the foreign job was started.
- *start_gate_factor* is a constant bias factor that is added to the job's running score. It is obtained from the execution of a gate program at the time the job was started.
- *running_gate_factor* is a constant bias factor that is added to the job's running score. It is obtained from the most recent execution of a gate program while the job is running.

5.3 The Partitions File

The partitions file, /usr/spool/djm/partitions, contains information of the configuration of the realized (current) and the unrealized (future) partitions. The information comes from the command **jop create partition**.

5.3.1 An Example of the Partitions File

```
current_partition:
current_mode                        normal
current_nproc                       32
current_baseproc                    256
current_reconfig_time               754372800
current_want_timeshared             yes
current_lock                        no
current_server_name                 none
current_hard_cputime                0
current_hard_mem_per_proc           0
current_hard_foreign_cputime        0
current_hard_cpu_times_mem_per_proc 0
current_soft_cputime                0
current_soft_mem_per_proc           0
current_soft_cpu_times_mem_per_proc 58982400
current_total_run_limit             0

current_partition:
current_mode                        normal
current_nproc                       512
current_baseproc                    512
current_reconfig_time               754177936
current_want_timeshared             yes
current_lock                        no
current_server_name                 none
current_hard_cputime                0
current_hard_foreign_cputime        0
current_hard_mem_per_proc           0
current_hard_cpu_times_mem_per_proc 0
current_soft_cputime                0
current_soft_mem_per_proc           0
current_soft_cpu_times_mem_per_proc 0
current_total_run_limit             0
```

```
future_partition:
future_fid                          3
future_mode                         dedicated
future_nproc                        512
future_baseproc                     512
future_want_timeshared              yes
future_lock                         no
future_unlock                       no
future_server_name                  none
future_reconfig_time                754520400
future_reconfig_kill                yes
future_nwarnings                    0
future_hard_cputime                 0
future_hard_mem_per_proc            0
current_hard_foreign_cputime        0
future_hard_cpu_times_mem_per_proc  0
future_soft_cputime                 0
future_soft_mem_per_proc            0
future_soft_cpu_times_mem_per_proc  0
future_total_run_limit              0
```

5.3.2 Explanation of the Parameters

The meaning of the parameters is described below. For most parameters, zero means "unlimited."

- *current_mode* is one of Normal, Dedicated, or Shutdown.
- *current_nproc* is the number of processing nodes.
- *current_baseproc* is the ordinal of the base processing node.
- *current_reconfig_time* is the time at which the partition was realized, measured in seconds since 00:00 GMT, January 1, 1970.
- *current_want_timeshared* is "yes" if a timesharing daemon should be started (CM-2 only).
- *current_lock* is "yes" if overlapping future partitions should be not be realized, or if a dedicated partition should not revert to normal mode if no jobs are queued nor running.
- *current_server_name* is the name of the server host.
- *current_hard_cputime* is the hard CPU time limit, measured in seconds. Queued jobs that request more than this amount of time will not be scheduled to run in this partition. Running jobs that consume more than this amount of CPU time will be killed.
- *current_foreign_hard_cputime* is a more restrictive hard CPU time limit that is applied only to foreign jobs.
- *current_hard_mem_per_proc* is the hard per-processor memory limit for jobs, measured in kilobytes.
- *current_hard_cpu_times_mem_per_proc* is the hard limit on the product of CPU time and per-processor memory, measured in kilobyte-seconds.

- *current_soft_cputime* is the soft CPU time limit for jobs. Queued jobs requesting more than this amount of time will not be scheduled to run in this partition unless they specify the -interruptible option. Running jobs that exceed this amount of will a reduction in their score (see the description of running_cputime_exceeded_factor), making them vulnerable to being killed if other jobs are waiting to run.
- *current_soft_mem_per_proc* is the soft per-processor memory limit, measured in kilobytes.
- *current_soft_cpu_times_mem_per_proc* is the soft limit on the product of CPU time and per-processor memory, measured in kilobyte-seconds.
- *current_total_run_limit* is maximum total number of jobs that may concurrently run in this partition.
- *future_fid* is a unique ID number, assigned by DJM.
- *future_mode* is one of Normal, Dedicated, or Shutdown.
- *future_nproc* is the number of processing nodes.
- *future_baseproc* is the ordinal of the base processing node.
- *future_reconfig_time* is the time at which the partition will be realized.
- *future_want_timeshared* is "yes" if a timesharing daemon should be started (CM-2 only) when the partition is realized.
- *future_lock* is "yes" if, once the partition is realized, other subsequent overlapping partitions should be not be realized, or if a dedicated partition should not revert to normal mode if no jobs are queued nor running.
- *future_unlock* forces the partition to be realized even if the current partition is locked.
- *future_server_name* is the name of the server host.
- *future_reconfig_time* is the time at which the partition will be realized.
- *future_reconfig_kill* is "yes" if jobs should be killed when the partition is realized. A warning message is sent at T minus 30, 5, and 1 minutes.
- *future_nwarnings* is the number of warning messages sent so far.
- *future_hard_cputime* is the hard CPU time limit for jobs, measured in seconds. Queued jobs that request more than this amount of time will not be scheduled to run in this partition. Running jobs that consume more than this amount of CPU time will be killed.
- *future_foreign_hard_cputime* is a more restrictive hard CPU time limit that is applied only to foreign jobs.
- *future_hard_mem_per_proc* is the hard per-processor memory limit for jobs, measured in kilobytes.
- *future_hard_cpu_times_mem_per_proc* is the hard limit on the product of CPU time and per-processor memory, measured in kilobyte-seconds.
- *future_soft_cputime* is the soft CPU time limit for jobs. Queued jobs requesting more than this amount of time will not be scheduled to run in this partition unless they specify the -interruptible option. Running jobs that exceed this amount of will a reduction in their score (see the description of running_cputime_exceeded_factor), making them vulnerable to being killed if other jobs are waiting to run.

- *future_soft_mem_per_proc* is the soft per-processor memory limit, measured in kilobytes.
- *future_soft_cpu_times_mem_per_proc* is the soft limit on the product of CPU time and per-processor memory, measured in kilobyte-seconds.
- *future_total_run_limit* is maximum total number of jobs that may concurrently run in this partition.

5.4 The Gates File

The gates file, /usr/spool/djm/gates, contains information of the configuration of the gates. Gates allow you to write programs that monitor or control access to the machine. The information comes from the command **jop create gate**.

5.4.1 An example of the Gates File

```
gate:
id              1
name            verifyproj
type            implicit
exec            /usr/local/etc/verifyproj
event_mask      submit,start
min_interval    -1
server          c1.msc.edu
baseproc        -1
nproc           0
activate_time   754011774
reject_if_error no
queue_argv      big
queue_argv      express

gate:
id              2
name            cray
type            optional
exec            /usr/local/etc/ping_cray
event_mask      queued
min_interval    300
server          c1.msc.edu
baseproc        768
nproc           256
activate_time   754011774
reject_if_error yes
queue_argv      remote
```

5.4.2 Explanation of the Parameters

The meaning of the parameters is described below.

- *id* is a unique ID number, assigned by DJM
- *name* is the name of the gate, matched by the -need or -gate -option.
- *type* is one of Implicit, Optional, or Required.
- *exec* is the name of the program or shell script that will approve of the job. For more information, see the section on *Gates*.
- *event_mask* is a list of events that will trigger the gate. The events are Submit, Queued, Foreign, Start, Running, and Finished.
- *min_interval* is minimum interval to call the gate for the periodic events, Queued and Running.
- *server* is the name of the server host that will trigger the gate. Jobs on other server hosts are ignored. If no server is specified, all servers are included.
- *baseproc* is base ordinal of the partition that will trigger the gate. Jobs on other partitions are ignored. -1 means all partitions.
- *nproc* is used with baseproc, above, to specify a unique partition.
- *activate_time* is the time at which to activate the gate.
- *reject_if_error* is "yes" if the job will be rejected, should DJM be unable to execute your program or should your program return a non-zero exit code.
- *queue_argv* is a list of queues that will trigger the gate, one per line. Jobs in other queues are ignored. If no queue is specified, a job in any queue will trigger the gate.

5.5 The Queue File

The queue file contains all of the information that is unique to a particular partition. There is one queue file for each queue in the system. The queue file has the name /usr/spool/djm/queue_*name*, where *name* is the name of the queue.

5.5.1 An Example of the Queue File

```
name                        default
search_order                100
enabled                     yes
started                     yes
users                       0 :
groups                      0 :
projects                    0 :
exact_mem_users             4 :  0 5781 5884 5886
exact_mem_groups            0 :
exact_mem_projects          0 :
max_memory                  0
max_mem_per_proc            0
max_cputime                 0
max_cpu_times_mem_per_proc  0
max_nproc                   0
min_memory                  0
min_cputime                 0
min_cpu_times_mem_per_proc  0
min_nproc                   0
min_pri                     0
max_pri                     100
foreign                     no
range_baseproc              -1
range_nproc                 0
dedicated                   no
user_nsub_limit             4
user_nrun_limit             4
user_part_nrun_limit        3
queue_nrun_limit            999
queue_part_nrun_limit       999
queue_mem_run_limit         0
queue_part_mem_run_limit    0
```

```
queued_base_factor                          100000
queued_wait_factor                          1
queued_nproc_factor                         -0.25
queued_pri_factor                           0
queued_cputime_factor                       0
queued_memory_factor                        0
queued_mem_cpu_product_factor               0
queued_already_queued_factor                -3600
queued_already_queued_cputime_factor        0
queued_already_queued_memory_factor         0
queued_already_queued_mem_cpu_product_factor 0
running_base_factor                         1e+07
running_nproc_factor                        0
running_memory_factor                       0
running_cputime_exceeded_factor             -2e+07
running_mem_cpu_product_exceeded_factor     -2e+07
running_memory_exceeded_factor              -2e+07
running_interactive_factor                  1000
running_dedicated_late_factor               -2e+07
```

5.5.2 Explanation of the Parameters

The meaning of the parameters is described below.

- *name* is the name of the queue.
- *search_order* is determines the order in which DJM will search for a queue to receive a new job. DJM scans the queues from the lowest search_order value to the highest value.
- *enabled* is "yes" if the queue can receive jobs.
- *started* is "yes" if the queue can run jobs.
- *users* is a list of user ID's that are validated to submit jobs to the queue.
- *groups* is a list of group IDs that are validated to submit jobs to the queue.
- *projects* is a list of project names that are validated to submit jobs to the queue.
- *exact_mem_users* is a list of user ID's that are validated to use the -exact_memory option.
- *exact_mem_groups* is a list of group IDS that are validated to use the -exact_memory option.
- *exact_mem_projects* is a list of project names that are validated to use the -exact_memory option.
- *max_memory* is the maximum amount of total memory that a job can request, measured in kilobytes.
- *max_mem_per_proc* is the maximum amount of memory per processing node that a job can request, measured in kilobytes.
- *max_cputime* is the maximum amount of CPU time that a job can request, measured in seconds.

- *max_cpu_times_mem_per_proc* is the maximum product of CPU time and memory per processing node that a job can request, measured in kilobyte-seconds.
- *max_nproc* is maximum number of processors that a job can request.
- *min_memory* is the minimum number of processors that a job can request.
- *min_cputime* is the minimum amount of CPU time that a job can request.
- *min_cpu_times_mem_per_proc* is the minimum product of CPU time and memory per processing node that a job can request.
- *min_nproc* is the minimum number of processors that a job can request.
- *min_pri* is the minimum priority that a job can request, specified by the -priority option.
- *max_pri* is the maximum priority that a job can request, specified by the -priority option.
- *foreign* is "yes" if the queue accepts foreign jobs. The queue will reject non-foreign jobs.
- *dedicated* is "yes" if the queue accepts dedicated jobs. The queue will reject non-dedicated jobs.
- *range_baseproc*, together with *range_nproc*, specifies a unique partition. Jobs that do not request a specific partition, or that request a different partition, will be rejected. -1 means "don't care."
- *user_nsub_limit* is the maximum number of jobs that can be submitted by a single user.
- *user_nrun_limit* is the maximum number of jobs that can be run by a single user.
- *user_part_nrun_limit* is the maximum number of jobs that can be run by a single user in a single partition.
- *queue_nrun_limit* is the maximum number of jobs that can be run from this queue.
- *queue_part_nrun_limit* is the maximum number of jobs that can be run from this queue in a single partition.
- *queue_mem_run_limit* is the maximum amount of memory that can be consumed by this queue.
- *queue_part_mem_run_limit* is the maximum amount of memory that can be consumed by this queue in a single partition.

The constant score factors, e.g., queued_wait_factor, are discussed in the section on *Scores*.

5.6 The Global File

The global file, /usr/spool/djm/global, contains system-wide parameters.

5.6.1 An Example of the Global File

```
nextjid                          19595
nextgid                          31456
prompt_cputime                   300
prompt_nproc                     32
prompt_mem_per_proc              1024
global_user_nsub_limit           999
global_user_nrun_limit           999
global_part_nrun_limit           5
global_int_part_nrun_limit       6
global_user_part_nrun_limit      999
global_job_cputime_limit         86400
always_believe_requested_memory  no
default_pri                      50
default_sp_memory                0
default_sp_disk                  0
default_safety                   0
foreign_requested_cputime        0
foreign_requested_mem_per_proc   0
foreign_requested_total_mem      4194304
foreign_hard_cputime             900
foreign_hard_mem_per_proc        0
foreign_hard_total_mem           4194304
idle_timelimit                   3600
dedicated_grace_period           600
maximum_future_reconfig          259200
dedicated_submit_by_limit        14400
maximum_memory_fraction_for_draining  0.75
operators                        8 :  0 2014 2223 2185 78 2023 2230 8733
managers                         2 :  0 2014
```

5.6.2 Explanation of the Parameters

The meaning of the parameters is described below. For most parameters, 0 means "unlimited."

- *nextjid* is the job ID that will be assigned to the next submitted job.
- *nextgid* is the group ID that will be assigned to the next submitted job.
- *prompt_cputime* is the CPU time that will be used as the default if the user presses return in response to the question "Estimated CPU time?"

- *prompt_nproc* is the number of processing nodes that will be used as the default if the user presses return in response to the question "Number of processors?"
- *prompt_mem_per_proc* is the amount of memory per processing node that will be used as the default if the user presses return in response to the question "Estimated memory?"
- *global_user_nsub_limit* is the maximum number of jobs that a single user may submit concurrently.
- *global_user_nrun_limit* is the maximum number of jobs that a single user may run concurrently.
- *global_part_nrun_limit* is the maximum number of jobs that may concurrently run in a single partition. This limit is applied to non-interactive jobs (launched with jsub).
- *global_int_part_nrun_limit* is is the maximum number of jobs that may concurrently run in a single partition. This limit is applied to interactive jobs (launched with jrun). It is typically higher than global_part_nrun_limit.
- *global_user_part_nrun_limit* is the maximum number of jobs that a single user may run in one partition.
- *global_job_cputime_limit* is maximum amount of CPU time that a single job can request.
- *always_believe_requested_memory* is "yes" if the scheduler should always fully allocate the requested memory for the job, even if the job actually consumes less memory. Normally, the scheduler will return the unused memory back to the free pool two minutes after the job begins. Use of this parameter is discouraged, because most users tend to over-estimate their memory use, often by a large amount. The -exact_memory option has a less drastic effect, because it is applied on a per-job basis[†].
- *default_pri* is the default value for the -priority option.
- *default_sp_memory* is the default value for the -sp_memory limit.
- *default_sp_disk* is the default value for the -sp_disk limit.
- *default_safety* is the default value for the -safety option.
- *foreign_requested_cputime* is the implied requested CPU time for foreign jobs. A foreign job that exceeds this limit is assigned a low score, allowing it to be killed if other jobs want to run.
- *foreign_requested_mem_per_proc* is the implied requested memory per processor for foreign jobs.
- *foreign_requested_total_mem* is the implied requested
- *foreign_hard_cputime* is the hard CPU time limit for foreign jobs.
- *foreign_hard_mem_per_proc* is the hard limit on memory per processor for foreign jobs.
- *foreign_hard_total_mem* is the hard limit on total memory for foreign jobs.
- *idle_timelimit* is the maximum time interval over which a job can sit idle, not clocking any CPU time. After the interval expires, the job will be killed with an explanatory message. This applies to both normal and to foreign jobs. It does not apply to jobs on a non-timeshared CM-2 sequencer; instead, use the command **cmidletimer**.

[†] The correct solution to the problem of DJM wrongly guessing the users' declared vs. actual memory use is checkpointing. If DJM guessed wrong, it can checkpoint the job until more memory becomes available.

- *dedicated_grace_period* is the maximum time interval over which a dedicated mode partition can sit idle, before it is automatically converted to a normal mode partition.
- *maximum_future_reconfig* sets how far in advance a future partition can be scheduled. Future partitions which are scheduled more than n seconds advance are ignored. This allows you to create scripts with future partitions that are planned months in advance, but without burdening the master daemon with lots of unnecessary overhead. The default is 3 days (259,200 seconds). Scheduled future reconfiguration events are usually put into a script (djm_reconfig.sh) and executed nightly via cron.
- *dedicated_submit_by_limit* is the minimum time interval between when a dedicated job is submitted and when a dedicated mode partition begins. If there are no jobs submitted by the deadline, the dedicated mode partition is converted to a normal mode partition, effectively cancelling the dedicated time period.
- *maximum_memory_fraction_for_draining* is the maximum amount of memory that a user can request, express as a fraction of the total memory of the partition, under which draining will occur. The default is 75% (0.75). Jobs that request more than this amount of memory ought to be resubmitted with the -dedicated option. The fraction should never be larger than the total amount of free memory, taking into account the overhead of the timesharing system (roughly 90%).
- *operators* is a list of user IDs that are validated to run the **jop** command to perform operation functions. These include all commands except **jop mgr**.
- *managers* is a list of user IDs that are validated to use the command **jop mgr**.

6 Other Sources of Information

6.1 The Thinking Machines Corporation DJM Reference Manual

An excellent source of supplementary information is the *Distributed Job Manager Reference Manual*, published by Thinking Machines Corporation. Chapter 1 contains introductory material and a tutorial for new users. Chapter 2 touches on many of the administrative concepts explained in this document. Where there is a discrepancy, however, this document takes precedence, because it is based on a more recent version of the software.

6.2 Internet Mailing List: djm@msc.edu

There is an Internet mailing list, djm@msc.edu, for system administrators who maintain DJM. Many system administrators read this mailing list, and they can advise you in setting up and maintaining DJM on your system. To join the mailing list, send a message to djm-request@msc.edu.