Unified Resource Manager (URM) [8]

The Unified Resource Manager (URM) is a job scheduler that balances the demands of both batch and interactive sessions. URM provides a high-level method of controlling the allocation of system resources to run jobs that originated either in batch mode or in an interactive session.

> **Note:** In this section, the term *job* is used to identify the scope of the object being managed by URM. For purposes of this discussion, an interactive session is considered a job.

Controlled system resources include CPU time, memory, tape devices, number of jobs, and Secondary Data Segment (SDS) allocation. The URM feature is common among all Cray Research architectures, but the managed resources are tailored to the hardware configuration on which URM is installed.

Prior to the introduction of URM, the UNICOS operating system included separate facilities for scheduling batch jobs and interactive sessions. Batch jobs were scheduled by the Network Queuing System (NQS). Interactive sessions were scheduled by transient session initiators, such as `login` and `rsh`. However, URM takes input from all session initiators, including NQS and the transient session initiators.

URM combines the management of machine resources under a single umbrella, making it possible to provide consistent treatment of resource demands arriving from these various sources. In addition to monitoring resource demands, URM also monitors the following:

- History of system usage. By monitoring the UNICOS fair-share scheduler, URM monitors past use of important system resources by each user.
- Current system load. URM monitors current use of all important system resources.
- Future work backlog. URM predicts future demands on all important system resources, as indicated by the job backlog and the amount of work in process and waiting to be initiated.
- Target loads. URM manages memory oversubscription, number of active processes, SDS oversubscription, tape usage, and the number of active batch and interactive jobs.

Using this information, URM evaluates requests to initiate jobs. Requests arrive from such UNICOS service providers as NQS (for batch) and `login` (for

interactive). Following the selection process, URM sends to each service provider a list of jobs that URM recommends for initiation.

URM does not actually initiate jobs. The service providers retain full responsibility for and control of jobs within their scope. Jobs are not required to be processed through the selection server portion of URM. This is allowed in order to retain command execution capability if the system is behaving improperly. Under normal operation, service providers use the URM selection server and follow the job initiation recommendations of URM.

The Unified Resource Manager is described in detail in the following sections:

- URM features
- Summary of URM commands
- Installing URM
- Configuring URM
- URM administrator tasks
- Troubleshooting URM
- URM architecture
- URM resources
- URM checkpointing
- Tuning URM

8.1 URM Features

The Unified Resource Manager provides the following functionality:

- Provides uniform services for all types of jobs (batch and interactive, for example) and eliminates non-essential differences among them.
- Uses `sdsmgr` to schedule use of SDS space for both batch and interactive sessions.
- Maintains job resource predictions (as offered by the service providers) as well as consumption information to assist the algorithms selecting jobs to recommend for initiation.
- Provides job initiation recommendations to NQS and other service providers.

- Supports SDS oversubscription through job preemption. (The preemption mechanism is suspend/resume.)
- Supports system scheduling on CRAY T3D systems by monitoring the load information and job backlog, and by recommending the best job candidates for initiation.
- Controls the number of active jobs. This includes control over the number of jobs of each type as well as the total number of all jobs.
- Uses the information provided by the fair-share scheduler to evaluate a user's potential priority among those competing to have a job initiated.
- Provides a way to report an assessment of machine loading, a list of jobs currently recommended for initiation, and other information upon request.

8.2 Summary of URM Commands

The following UNICOS commands support URM:

Command	Description
`rmgr`(1)	Provides an interface to the URM daemon
`urmd`(8)	Starts the URM daemon
`urmsnap`(8)	Captures the current URM configuration information
`usetjob`(8)	Changes the minimum rank of batch jobs
`ustat`(1)	Displays URM job information

For details of the `rmgr`(1) and `ustat`(1) commands, see the UNICOS User Commands Reference Manual, publication SR-2011. For details of the `urmd`(8), `urmsnap`(8), and `usetjob`(8) commands, see the UNICOS Administrator Commands Reference Manual, publication SR-2022.

8.3 Installing URM

To install URM, use the UNICOS Installation / Configuration Menu System (the *menu system*). For details about the menu system and how to use it, see the following publications:

- *UNICOS System Configuration Using ICMS*, publication SG–2412
- *UNICOS Installation Menu System Reference Card*, publication SQ–2411

Perform these basic steps to install URM:

1. Verify the automatic startup of the URM daemon
2. Configure the initial URM values
3. Verify the security parameters
4. Enable the service providers

The following sections discuss these steps.

Warning: The following information on configuring URM is not written for a site running a Cray ML-Safe configuration of the UNICOS system. For information on configuring URM for a Cray ML-Safe configuration, see the description of the UNICOS MLS feature in General UNICOS System Administration, publication SG-2301.

8.3.1 Verifying Automatic Startup of URM Daemon

To ensure that the URM daemon (`urmd`) is invoked automatically, verify that the `/etc/config/daemons` file includes a line for `urmd`. If the `/etc/config/daemons` file does not contain a line for `urmd`, you should create one by using the menu system. Traverse the menus by using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    System daemons configuration ->
      System daemons table ->
```

The final menu selection displays a list of all active system daemons and their attributes, as shown in the following example:

```
     Group  Name       Start Opts   Kill                 Program    >
     -----  ---------  -----------  -------------------  ----------
     SYS1   cron       5=NO:*=YES   *                    /etc/cron
     SYS1   fsdaemon   5=NO:*=YES   *                    /etc/fsmon
     TCP    gated      NO           /etc/gated.pid       /etc/gated
E->  SYS2   urmd       YES          /usr/lib/urm/urmend  /etc/urmd
```

If this table does not include such a line for `urmd`, create a new entry having the following attributes:

```
S->  Group                  SYS2
     Name                   urmd
```

```
Startup at boot time?  YES
Kill action            /usr/lib/urm/urmend
Executable pathname    /etc/urmd
```

The meaning of these attributes is as follows:

Attribute	Description
SYS2	Places `urmd` in the group of daemons started last.
urmd	Names the URM daemon.
YES	The URM daemon executes automatically at system startup. If this attribute were `NO`, the daemon would not execute automatically at system startup; however, the daemon could be started manually by using the command `sdaemon urmd`.
/usr/lib/urm/urmend	Specifies the command to shut down `urmd` and request interactive checkpointing (if configured).
/etc/urmd	Identifies the full path name of the executable for the URM daemon.

To activate this configuration change, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    System daemons configuration ->
      Activate the daemon configuration ...
```

Activation causes the menu system to add a `urmd` line to the `/etc/config/daemons` file. You can then stop `urmd` manually by using the `sdaemon -k urmd` command or restart `urmd` manually by using the `sdaemon urmd` command.

8.3.2 Configuring Initial URM Values

As-shipped default parameters for URM configuration are designed to minimize the impact of URM installation on a running system. This allows you to learn about URM and control its effects as you activate the configuration changes that allow URM to monitor various system limits.

The default configuration should have no effect on a running system until you explicitly change certain of these URM configuration values. The following section discusses these changes.

8.3.2.1 Individual Session Initiator Configuration Changes

The as-shipped default URM configuration overrides all URM limits and recommends initiation for all jobs started by all individual session initiators except batch. (In the case of batch jobs, this default never applies, since the NQS configuration parameters override these values.) To change these defaults and permit URM to make recommendations about whether or not to initiate jobs that started from individual session initiators, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Individual session initiator defaults ->
```

Initially, every entry in the default table of values is 0:

```
     Name   CPU Time  Memory Usage
     -----  --------  ------------
E->  batch  0         0
     cron   0         0
     ftp    0         0
     login  0         0
     null   0         0
     rexec  0         0
     rsh    0         0
     site1  0         0
     site2  0         0
     site3  0         0
```

For a given session initiator (`login`, for example), if both the `CPU Time` and the `Memory Usage` entries are 0, then you override URM limits and allow all jobs started by that session initiator. For example, if the `CPU Time` and `Memory Usage` entries for `login` are 0, URM favorably recommends all `login` requests, regardless of system load. Only when either entry for `login` is a nonzero value are `login` requests subject to URM limits.

Therefore, to enable communication between each session initiator and URM, change each 0 for that session initiator to a nonzero value. For example, for `login` and `rsh`, you might use a minimum of the estimated size of a shell process (in clicks).

Note: Configuration parameters in the NQS override these values for `batch`.

The `rsh` settings also affect `rcp` (remote copy).

For any changes to these configuration parameters to take effect, you must activate the changes by using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Activate urm configuration ...
```

For details of the activation process, see Section 8.4.12, page 371.

8.3.3 Verifying Security Parameters

As part of any URM installation, you should verify that the URM configuration parameters establish only the authorized administrators and authorized hosts that you want. To learn about these parameters and how to change them, see Section 8.4.1, page 360.

8.3.4 Enabling Service Providers

With one exception, all service providers (such as `ftp`, `login`, and `rsh`) automatically use URM. The one exception is NQS. You must explicitly enable URM services in NQS.

To change the NQS configuration parameters to enable URM services, use the menu system. To enable communication between NQS and URM, the NQS configuration must include a statement to set URM on.

To ensure that this communication continues after a restart, the NQS configuration also must include a statement to set URM restart on. For details of how to enable URM services and how to enable URM restart in NQS, see the UNICOS NQS and NQE Administrator's Guide, publication SG-2305.

8.4 Configuring URM

After URM has been installed and all initial values have been configured, you can change the URM configuration values as needed. The following sections describe the URM configuration parameters, how to use the menu system to implement changes in URM configuration, and how to activate changes.

Warning: The following information on configuring URM is not written for a site running a Cray ML-Safe configuration of the UNICOS system. For information on configuring URM for a Cray ML-safe configuration, see the description of the UNICOS MLS feature in General UNICOS System Administration, publication SG-2301.

To make changes to the URM configuration, use the menu system. Traverse the menus using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
```

This menu selection displays the following menu:

```
M->  Authorized administrators ->
     Authorized hosts ->
     Machine load evaluation rates ->
     Machine target values ->
     Individual session initiator targets ->
     Individual session initiator defaults ->
     URM control settings ->
     Weighting factors for the selector ->
     Auto-configuration settings ->

     Reset DEFAULT urm configuration ...

     Import urm configuration ...
     Activate urm configuration ...
```

The following sections discuss each line of this URM configuration menu. In addition, online help is accessible on each menu in the menu system.

8.4.1 Authorized Administrators

To control access to URM, use the menu system. Traverse the menus using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Authorized administrators ->
```

The as-shipped defaults for these parameters are as follows:

```
     Login name  Type
     ----------  ----------
E->  root        privileged
     *           public
     *           anonymous
```

These parameters have the following meanings:

Parameter	Description
`privileged`	Can read any internal information. Can stop the URM daemon (using `rmgr -c 'stopdaemon'`).
`public`	Can read only global information and information owned by this user.
`anonymous`	Can read only global information.

If you want to add an authorized administrator, create a line for the login of the new authorized administrator (`admin2`, for example):

```
     Login name  Type
     ----------  ----------
     root        privileged
     *           public
     *           anonymous
E->  admin2      privileged
```

If, for security reasons, you want to limit who can use `rmgr` to access URM, delete the two asterisked lines:

```
     Login name  Type
     ----------  ----------
     root        privileged
E->  admin2      privileged
```

This limits access privileges to `root` and `admin2`.

Note: For proper URM functioning, you must retain the `root privileged` entry.

The login name * is not recognized in the privileged node. Therefore, the entry `* privileged` is not allowed. This restriction also applies to the local URM configuration file; an entry of the form `/admin/privileged/*` is purposely illegal.

8.4.2 Authorized Hosts

To control access to URM from remote hosts, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Authorized hosts ->
```

The as-shipped default for this parameter is as follows:

```
     Host name
     ---------
E->  *
```

This parameter has the following meaning:

`*` (all) Allows any client from a remote host to connect to `urmd`, assuming the administrator configuration also allows the connection.

If, for security reasons, you want to disallow connections from all remote hosts, remove the asterisk (`*`). You can then add a line for each host from which you want to allow connections (`system1` and `system2`, for example):

```
     Host name
     ---------
     system1
E->  system2
```

8.4.3 Machine Load Evaluation Rates

The as-shipped URM configuration includes a set of smoothing factors that are designed to reduce the impact of sudden changes in resource usage. These factors reduce the rate at which changes are factored into the URM decision-making process. This, in turn, protects URM users by making URM results more predictable.

To see or change the smoothing factors, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Machine load evaluation rates ->
```

The as-shipped default rates are as follows:

```
S->  MEMORY rate                          0.8
     SDS rate                             0.8
     TAPE rate                            1.0
     MPP BARRIERS rate                    1.0
     MPP PROCESSING ELEMENTS (pe)         1.0
```

These rates should not be changed, except by an analyst experienced in tuning systems using URM.

8.4.4 Machine Target Values

To properly perform its scheduling functions, URM must be aware of target usage limits for important system resources. These resources include memory, SDS, job count, tapes, and MPP (massively parallel processing systems).

During installation, a URM automatic configuration command changes many of the URM default targets. Automatic configuration determines the actual configuration of your system and replaces the as-shipped default values with values derived from the actual values for your system.

The formula used to change these values can be modified by using the menu system. Values that can be changed globally include memory and SDS oversubscription, and limits for job count, tapes, and MPP. In addition to these values, values for CPU time and memory usage can be set for each individual session initiator (see "Installing URM," Section 8.3, page 355).

The following sections describe the URM configuration values that can be set globally.

To verify or change the value (either multiplier or limit) for each important system resource, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Machine target values ->
```

The as-shipped URM configuration has the following values:

```
S->  Memory oversubscription multiplier  2.0
     SDS oversubscription multiplier     1.5
     Target session maximum              MAX
     Target tape limit                   MAX
     Target MPP barriers limit           MAX
     Target MPP PE limit                 MAX
```

The auto-configure script uses each of these values as follows:

Value	Description
`2.0`	Multiplies the value of your system's actual physical user memory (`user_memory`) by this value (`2.0`) to calculate a memory oversubscription value (`memory`). URM strives to hold memory usage below this memory oversubscription value. To see the value being used by URM, view the object `/machine/target/memory`. You cannot alter this object directly; you can only change the multiplier.
`1.5`	Multiplies the value of your system's actual SDS user space (`user_sds`) by this value (`1.5`) to calculate an SDS oversubscription value (`sds`). URM strives to hold SDS usage below this SDS oversubscription value. To see the value being used by URM, view the object `/machine/target/sds`. You cannot alter this object directly; you can only change the multiplier.
`MAX`	The following paragraphs describe, for each parameter, the effects of a `MAX` value. For `Target session maximum`, the `MAX` value sets the maximum number of jobs (both interactive and batch combined) that URM allows to be running at any time (`jobcount`). The upper limit for this range is defined by limits in the kernel configuration. That is, if this field contains a number that is higher than the maximum established in the kernel configuration parameters, this higher number is ignored, and the kernel configuration value is used. However, if this field contains a lower number, the lower number takes effect for URM. For `Target tape limit`, the `MAX` value compiles a list of all tape systems actually available on the system and puts this information in the field `tape`. For MPP limits, the `MAX` value allows URM to determine whether or not the system configuration includes an MPP system and, if so, allows URM to factor MPP limits into its scheduling algorithm. **Note:** To allow proper URM autoconfiguration to occur, do not change the keyword `MAX` for this resource.

Using the menu system, you can reset these factors to any value you decide is reasonable.

For changes to these configuration parameters to take effect, you must activate the changes by using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Activate urm configuration ...
```

For details of the activation process, see Section 8.4.12, page 371.

8.4.5 Individual Session Initiator Targets

To verify or change the target values for each individual session initiator, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Individual session initiator targets ->
```

This menu selection displays a table of target values, as in the following example:

```
     Name   bb   cputime  jobcount  memory  pe   petime   sds  tape
     -----  ---  -------  --------  ------  ---  -------  ---  ----
E->  batch  MAX  9999999  MAX       MAX     MAX  9999999  MAX  MAX
```

8.4.6 Individual Session Initiator Defaults

To change the default values for each individual session initiator, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Individual session initiator defaults ->
```

This menu selection displays the table of default values. For example:

```
     Name   CPU Time  Memory Usage
     -----  --------  ------------
E->  batch  0         0
     cron   0         0
     ftp    0         0
     login  40        250
```

```
null   0         0
rexec  0         0
rsh    0         0
site1  0         0
site2  0         0
site3  0         0
```

Entries for `CPU Time` are given in seconds. Entries for `Memory Usage` are given in clicks. You can raise or lower these limits by changing the entries in this table.

Lines for `site1`, `site2`, and `site3` are for local session initiators defined by the site.

For a given session initiator (except `batch`), if both entries are 0, that session initiator is not subject to URM limits.

Note: Configuration parameters in the NQS override these values for `batch`.

The `rsh` settings also affect `rcp` (remote copy).

For changes to these configuration parameters to take effect, you must activate the changes by using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Activate urm configuration ...
```

For details of the activation process, see "Activating URM configuration changes," Section 8.4.12, page 371.

8.4.7 URM Control Settings

The URM configuration includes a variety of control factors that affect URM performance. To see a list of these control factors, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM control settings ->
```

This menu selection displays a table of URM control factors and current settings. For example:

```
     Seconds between load info access             10
     Seconds to wait for job initiation           1800
     Seconds between scheduling cycles            10
     Main loop sleep period                       10
     Old Caller timeout in seconds                600
     Share evaluation period in seconds           900
     Name of the SDS suspend command              UPATH/sdsspnd
S->  SDS residency time in secs (0=SDS mgmt off)  900
     Full path name of local URM config file      /etc/config/urm
     Full path name of URM chkpnt directory       /PPATH/chkpnt
     Checkpoint policy                            Shutdown
     Share policy                                 Standard
     Minimum # of seconds between checkpoints     1800
     Checkpoint interval measurement base         Clock
     Maximum # of seconds to keep chkpnt file     432000
     Name of the interactive chkpnt command       /UPATH/intchkpt
     Restart flag policy                          Force
     Name of the interactive restart command      /UPATH/irstart
     Rank boost to previously running batch jobs  6.0
```

The following sections discuss three of these factors that may be particularly useful to administrators: `SDS residency time in secs`, `Full path name of local URM config file`, and `Rank boost to previously running batch jobs`.

8.4.7.1 SDS Residency Time

This menu selection determines the minimum amount of time that a job remains in SDS before being swapped out (900 seconds, in this example). This applies to both batch and interactive. Prior to UNICOS 8.0, `qfdaemon` performed this function, but only for batch jobs.

If SDS residency time is set to 0, then URM does not perform SDS management. If SDS residency time is set to a nonzero number, then URM performs SDS management on interactive jobs. For URM to perform SDS management on both interactive and batch jobs, SDS residency time must be set to a nonzero number and the NQS `set job scheduling` *option* parameter must be set to the correct option. For details of enabling URM services in NQS, see the UNICOS NQS and NQE Administrator's Guide, publication SG-2305.

8.4.7.2 Local URM Configuration File

This menu selection is used to identify a site-defined configuration file. When activated, changing this value to *filename* adds an `Include` *filename* statement to the end of the `/etc/config/urm/configuration` file.

You can change the URM configuration by editing this site-defined configuration file. Or you could change the URM configuration by writing a `cron` job that accesses this file.

8.4.7.3 Rank Boost to Previously Running Batch Jobs

This menu selection specifies an additional value given to checkpointed batch jobs to improve their rank. This value is applied to batch jobs that were previously running and were checkpointed due to a shutdown or hold. The rank boost is added after the job's rank has been calculated using the weighting factors (described in the following section).

8.4.8 Weighting Factors for the Batch Selector

The method by which URM selects the next batch job to recommend for initiation depends upon the interaction between the NQS and URM. This interaction is influenced by a set of weighting factors established in the URM configuration. To see or change these weighting factors, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Weighting factors for the selector ->
```

The as-shipped default settings for these factors are as follows:

```
S->  Age in queue weight                    0.5
     MPP Barrier bits weight                0.5
     MPP Processor elements weight          0.5
     MPP Requested PE time limit weight     0.5
     Requested CPU time weight              0.5
     Requested Memory weight                0.5
     Requested SDS weight                   0.5
     Requested Tape weight                  0.5
     Service provider priority weight       0.5
     Share priority weight                  0.5
     Share entitlement weight               0.5
     Usage weight                           0.5
```

Possible values are in the range of 0.0 to 1.0. For each batch job, URM assigns a ranking based upon place in each queue. For example, using the `Age in queue weight` factor, the oldest job is assigned a ranking of 0.5, while the youngest job is assigned a ranking of 0.0. By comparing the resulting weighting factor for each batch job, URM determines the highest priority job.

These default settings should be changed only by a system analyst experienced at tuning systems running URM.

Note: Share priority is the result of the following calculation:

$Usage/entitlement^2$

Although the share priority, usage, and share entitlement weighting factors all have a nonzero value by default, you must disable one or two of these values. Configure only one of the following choices:

- Share priority weight
- Share entitlement weight
- Usage weight
- Share entitlement weight and usage weight

8.4.9 Auto-configuration Settings

To enable auto-configuration of various subsystems, URM includes a tool in the menu system. To see or change the auto-configuration settings, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Auto-configuration settings ->
```

The as-shipped default setting for each subsystem is as follows:

```
     Resource  Enabled  Tries  Wait time
     --------  -------  -----  ---------
E->  memory    YES      10     20
     mpp       YES      10     20
     serial    YES      10     20
     session   YES      10     20
     sds       YES      10     20
     tape      YES      10     20
```

The URM has the ability to determine the available system resources. This menu allows you to modify the retry parameters of the auto-configuration mechanism.

The `Resource` field names the resource to be auto-configured. URM can automatically determine the system's `memory`, `mpp`, `serial`, `session`, `sds`, and `tape` resources. The `serial` resource configures the machine's serial number and the information available through `uname`(3). The `session` resource corresponds to the system's defined maximum number of sessions (`NSESS`). The `Enabled` (`Perform auto-configuration`) field is used to enable or disable the auto-configuration of the specified resource. CRI recommends that these fields remain set to `YES`, to allow auto-configuration.

Note that disabling auto-configuration prevents URM from using the `MAX` settings in the `URM Machine target values` and `URM Individual session initiator targets` menus.

The `Tries` (`Number of tries`) field indicates the number of times URM should attempt to auto-configure the specified resource. It is necessary to have a retry value, because some resources may not be available when URM is first initialized. For example, if URM is brought up before the tape daemon is running, URM's first attempt at auto-configuring tape resources fails.

The `Wait time` field represents the time in seconds to wait before retrying the auto-configuration of the specified resource.

8.4.10 Resetting to Default URM Configuration

The menu system allows you to reset all URM configuration parameters to the as-shipped default state. This could be useful if you find that configuration changes you have made have created unexpected problems and you want to start over with a known-working URM configuration. To remove all local changes, and return to the as-shipped default configuration parameters, use the menu system. Traverse the menus using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Reset DEFAULT urm configuration ...
```

8.4.11 Importing URM Configuration

For systems on which a URM configuration has not yet been imported, the menu system provides a tool that reads in certain defaults. To read in all values contained in the `/etc/config/urm/configuration` file, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Import urm configuration ...
```

Warning: This menu item should be selected only on systems that do **not** have a running URM, as it overwrites any previously existing values for these URM parameters.

The values read in are for the following parameters: `init`, `machine`, `res`, `structure`, `urminfo`, and `val`.

8.4.12 Activating URM Configuration Changes

After implementing any URM configuration changes, you must activate the changes for them to take effect. To activate the changes from within the menu system, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Activate urm configuration ...
```

The next time `urmd` is started, the menu system updates the contents of the `/etc/config/urm/configuration` file.

Caution: The `/etc/config/urm/configuration` file should never be edited directly; any changes should be made through the menu system.

Changes activated from within the menu system do not affect the currently executing `urmd` process; you can make the changes known to the currently executing `urmd` process by typing the following:

```
rmgr /etc/config/urm/configuration
```

8.4.13 Using URM with NQS

By default, URM does not control NQS jobs. To enable full URM control over the initiation of NQS jobs, enter the following `qmgr` subcommand:

```
set job scheduling urm unlimited
```

With full URM control, NQS queue limits are no longer honored; instead, URM's resource targets and job ranking priorities determine which jobs are recommended to NQS for initiation. For information about NQS parameters, see the `qmgr`(8) man page. For information on changing NQS configuration parameters, see the UNICOS NQS and NQE Administrator's Guide, publication SG-2305.

8.5 URM Administrator Tasks

To administer URM, you should understand the following procedures:

- Using URM log files
- Viewing URM results
- Viewing machine load
- Viewing all users
- Viewing jobs of a given user
- Changing a job's minimum rank
- Changing URM configuration based on time of day (`cron`)
- Customizing URM (user exits)

These procedures are described in the following sections.

8.5.1 Using URM Log Files

The `urmd` process maintains a log file to record its actions. Each time the daemon initializes, it creates a log file if one does not exist in the URM log directory. The log file is placed in the `/usr/adm/urm/` directory and is named `Urm`. *yymmdd*. The log file is appended to throughout the day (*yymmdd*), recording all directives, errors, and other important actions when the daemon is running. The next day, when a log message occurs, the existing log file is closed, the date is incremented, and a new log file is created.

An example of log entries is as follows:

```
11:16:20    153 U D Set Basenode /
11:16:20    153 U D Set Kernelaccess
11:16:20    153 U D Set Initialized
```

The format of the log messages is as follows:

hh:mm:ss 000000 x y message

The message is prefixed by the time the message was logged (*hh:mm:ss*), the UID of the current user when the message occurred (*000000*), the access privilege (*x*), and the severity code (*y*). UID `999999` means that the message comes from `urmd` and is not associated with a user.

The access privilege labels (*x*) are as follows:

`L`	Public socket access
`N`	No user defined (internal message)
`U`	Unrestricted privilege socket access

The severity labels (*y*) are as follows:

`D`	Logging a directive
`F`	Fatal
`I`	Informative
`L`	Log message
`N`	No error
`R`	Error type out of range
`W`	Warning

X Exit message

8.5.1.1 Monitoring URM Log Files

Because `urmd` creates a new log file each day (`/usr/adm/urm/Urm`. *yymmdd*), the number of log files grows rapidly. To conserve disk space, you should monitor this growth and implement a strategy to minimize the effects. For example, you could archive all but the most recent log files. You could use `cron` to accomplish this on a scheduled basis, such as running an archiving script at the end of each week.

8.5.1.2 Turning Off URM Logging

The log file can be turned off, using the local configuration file. To turn off logging the activities of URM commands, enter the following:

```
# rmgr
rmgr-> set nolog
```

8.5.1.3 Moving the URM Log Files

By default, the URM log files are in the `/usr/adm/urm` directory. You can change the default directory with either of the following methods.

When invoking the the URM daemon, `/etc/urmd`, use the `-l` option to specify an alternate log directory. The log file names cannot be changed, only the path. See the `urmd`(8) man page for more information about the `-l` option.

When URM is running, an authorized URM administrator can change the location of the log using the following `rmgr`(1) subcommand:

`set log` *directory*

If *directory* specifies a valid directory, the current log is terminated with a message and closed, and a new log is opened in *directory*. Error messages are issued if the new log file cannot be opened, and logging remains with the old log file.

The `set log` *directory* directive can also be specified in the local configuration file. However, when this method is used, the URM daemon writes the initialization log in the default directory (`/usr/adm/urm`). To change the location of the initialization log, use the `-l` option when starting `/etc/urmd`. See the `rmgr`(1) man page for more information about the `set log` directive.

8.5.2 Viewing URM Results

At any time while URM is running, you can generate a list of the jobs that URM currently recommends for initiation. To generate this list, first invoke the `rmgr`(1) command:

```
# rmgr
```

Then choose the `view jselect` (view job select) option:

```
rmgr-> view jselect
```

This yields a report of jobs currently recommended for initiation. You can use this report to help determine configuration changes needed to tune URM for your site.

8.5.3 Viewing Machine Load

To assess machine load, first invoke the `rmgr`(1) command:

```
# rmgr
```

Then choose the `view /machine/load` (view machine load) option:

```
rmgr-> view /machine/load
```

This yields a report of the resource usage of all jobs currently running, showing the load on all subsystems monitored by URM. This report can be useful in troubleshooting URM, to determine whether some parameter has been set too low and has been exceeded.

8.5.4 Viewing All Users

To generate a list of all URM users, first invoke the `rmgr`(1) command:

```
# rmgr
```

Then choose the `view users` (view all users) option:

```
rmgr-> view users
```

This yields a report of all users currently connected to the URM daemon and their respective privileges (privileged, public, or anonymous).

8.5.5 Viewing Jobs of a Given User

To monitor the activities of a single user, first invoke the `rmgr`(1) command:

```
# rmgr
```

Then choose the `view jobs` (view one user) option:

```
rmgr-> view jobs login | UID
```

This yields a report of all jobs currently recommended for initiation that are owned by the specified user (*login* or *UID*). This might be useful in predicting system load or in troubleshooting URM.

8.5.6 Changing a Job's Minimum Rank

By default, NQS sets the minimum rank of jobs in the job backlog. The `ustat`(1) command allows the URM administrator to change the minimum rank of a specific job or jobs to affect the URM rank of those jobs.

Use `ustat`(1) to display the minimum rank of batch jobs, as in the following example:

```
ustat -a -m
```

The `view` directive of `rmgr`(1) also displays minimum rank values.

Use the `usetjob`(8) command to change the minimum rank of batch jobs. The following example changes the minimum rank for NQS job ID 12345:

```
usetjob -r 7 12345
```

If multiple machines submit jobs, the machine name is necessary to prevent possible misinterpretation of the request ID. If the job in this example originated from the machine `fred`, you would use the NQS request ID `12345.fred`. The host ID (for example, 999) could be used instead of the machine name, giving a job identification field of `12345.999`.

8.5.7 Changing URM Configuration Based on Time of Day (`cron`)

You can use the `cron` command to change the URM configuration at certain times of day. This would be useful, for example, if you wanted one configuration for daytime loads that include more interactive sessions and a different configuration for night-time loads consisting primarily of batch jobs. To implement such a configuration change:

1. Use the menu system to create a configuration file for daytime use (for example, `config.day`) and activate the change.

   ```
   cp /etc/config/urm/configuration /etc/config/urm/config.day
   ```

2. Use the menu system to create a configuration file for night-time use (for example, `config.night`) and activate the change.

   ```
   cp /etc/config/urm/configuration /etc/config/urm/config.night
   ```

3. Specify a local configuration file to be included when `urmd` is started (for example, `/etc/config/urm/local`).

4. Edit the `config.day` and `config.night` files to remove from each file the following line:

   ```
   Include "/etc/config/urm/local"
   ```

   This allows the auto-configuration to work properly in the event of a restart.

5. Create a `crontab` file that includes the following:

   ```
   cp /etc/config/urm/config.day /etc/config/urm/local
   rmgr /etc/config/urm/local
   ```

   The correct configuration file for the time of day must be copied to the `local` file, so that, in the event of a restart, the correct configuration file is used automatically.

8.5.8 Customizing URM (User Exits)

The URM contains code that allows you to customize it for the specific needs of your site by creating a site-written job selector. In URM, this site-written selector code is called last.

As shipped, the selector code is empty; it defaults to exiting without doing anything. However, you can write your selector based on any ranking algorithm you choose, and insert your code into URM. Then you could turn off all URM selectors and allow your site-written selector to make all job-initiation recommendations based on your ranking algorithm.

Warning: Use of any user exit is not permitted on a Cray ML-Safe configuration of the UNICOS system. Use of any user exit may result in loss of the evaluated rating.

8.5.8.1 URM User Exits

URM includes two user exits. The URM daemon (`/etc/urmd`) contains two routines that can be replaced with a set of user-defined versions by re-linking `urmd` with them. These user exits are defined in `site_rank.c`.

User exit	Description
`site_adjust_merit`	Allows the user to modify the ranking factor. A higher number increases the chances that a job will be recommended for execution. A lower number makes it more difficult for the job to be recommended.
`site_target_check`	Allows the user to set and reset limit flags for a job entry in URM. Jobs that have flags set will not be recommended for execution.

The order in which these ranking functions are invoked affects how the user exits can be used.

To understand URM user exits, you must understand certain functions and data structures in URM that are involved with the URM user exits. These are described in the following two sections.

8.5.8.2 URM Job-ranking Functions

URM includes three job-ranking functions (defined in `rank.c`) that call the user exits. The job-ranking functions are as follows:

Function	Description
`rank_jobs`()	Forms the jobs list. To qualify, the job must pass `raw_select` () and `maximum` filters. Next, the relative priorities and resource requirements are used for the final ranking in the job list. The other two functions are called from this one.
`adjust_merit`()	Adjusts the merit parameters of a job. The user's share is evaluated and used to adjust the relative merit of a job. The relative ranking value is stored in `urm_rank`. Called with 0 to preset the statics. This function calls the user exit `site_adjust_merit`().

`target_check()`	Determines whether or not the job exceeds the target usage limits. If the job will not exceed the target, returns `UR_NONE`. If the job will exceed the target, returns one of the other `UR_`*xxx* reason codes. This function calls the user exit `site_target_check()`.

The pseudo-code for `rank_jobs()` is as follows.

```
rank_jobs {                              /* The real ranking function.   */

   initialize_the_first_list             /* Initialize firstlist of jobs.*/
   initialize_the_job_list               /* Initialize job list.         */
   initialize_the_rec_list               /* Initialize recommend list.   */
   initialize_the_pre_list               /* Initialize prempt list.      */
   initialize_the_rest_list              /* Initialize restore list.     */
   initialize_the_chkpnt_list            /* Initialize checkpoint list.  */

   get_loadinfo(objects)                 /* Get system load information. */

   adjust_merit(initialize variables)    /* Initialize ranking weights.  */
      site_adjust_merit(initialize)      /* Allow user to initial other  */
                                         /*   user specific variables.   */

   target_check(0, TC_INIT)              /* Initialize usage limits.     */
      site_target_check(initialize)      /* Allow user to initial other  */
                                         /*   user specific limits.      */

   for (job_objects) {                   /* Add job objects to lists.    */
      update_prelist(job_object)         /* Add to prempt list.          */
      update_restlist(job_object)        /* Add to restore list.         */
      update_chkpntlist(job_object)      /* Add to checkpoint list.      */
      if (already_ranked_job)
         target_check(job, TC_PRELOAD)   /* Update potential usage and   */
            site_target_check(initialize) /*  initialize again.          */
      if (raw_select(job_object))        /* if job passes raw limits then*/
         update_firstlist(job_object)    /*    add to the first list.    */
   }

   for (first_list) {
      apply_jobmax_constraints           /* Begin to weed out jobs.      */
         create_job_list                 /* Create actual job list.      */
   }
```

```
   for (job_list) {                    /* Loop through the job list.   */
      adjust_merit(job)                /* Calculate this jobs ranking  */
                                       /*   factor.                    */
         site_adjust_merit(job)        /* And user can make changes to */
                                       /*   the ranking factor.        */
   }

   sort_job_list                       /* Sort job list.               */

   for (job_list) {                    /* Loop through the job list.   */
      target_check(job)                /* Will the job exceed limits?  */
         site_target_check(job)        /* Allow user to do more checks */
                                       /*   and even reset limit flags.*/
      if (no_reason_code)
         add_job_to_rec_list           /* Add to recommend list.       */
   }
}
        /* The rec_list now contains the recommended list of jobs that  */
        /* the session initiator should try and start.                  */
```

8.5.8.3 URM Data Structures

The structures and typedefs that URM uses to contain the job list information are defined in the following header files:

```
#include <urm.h>
#include <errnum.h>
#include "urmdefs.h"
#include "object.h"
#include "job.h"
#include "rank.h"
```

Two URM data structures in particular are useful to the user exits. These structures are defined in the following two header files, which are found in the directory `/usr/src/prod/admin/urm/urmdaemon`:

`Typedef Object;`	Defined in `object.h`
`Typedef Job;`	Defined in `job.h`

The `Job` data structure contains a field that allows URM to manage site-specific job information. When the `JOBHIST_UXIT` flag in the `history` field is set, URM interprets the `uxit` union in the `Job` data structure as a pointer to allocated memory and frees the specified space. However, if the user exit does

not use the `uxit` field or uses it as a number, the `JOBHIST_UXIT` flag must not be set.

It is vital to URM that a user exit correctly modifies data in these structures. Whoever is working with user exits should understand and be familiar with the structures present in the header files, in order to use them effectively.

There are also three predefined site flags and reason codes. When URM does a target check, it returns a flag and a reason code for what caused the job to not be recommended. The flags are defined as macros in `rank.h` and the reason codes are defined by the typedef `URM_code` in `job.h`. The predefined site flags and reason codes are as follows:

Flag	Reason code
`_SITE1_TAR`	`UR_SITE1_TAR`
`_SITE2_TAR`	`UR_SITE2_TAR`
`_SITE3_TAR`	`UR_SITE3_TAR`

Caution: The user exits can give a site almost total control over the ranking algorithm used by URM. In theory, a site can totally modify the recommended job list (and other internal URM structures).

When writing user exits, be careful when using any of the following.

These can have an adverse affect on URM and could result in an unreliable or unusable URM daemon. The user exit should, if possible, avoid any of these conditions or uses.

- System calls or functions that would block (such as trying to connect to a socket).
- `fork` and `exec` calls.
- Creating pipes or opening files.
- Use of signals.
- Code that significantly slows down the ranking.

8.5.8.4 URM User Exit Example 1

In this example, a user is submitting a weather model batch job that should be given a high priority to run, and should not have to wait in an NQS queue for

very long. The URM user exits can be used to ensure that user's job is set to run before most other jobs, if not first.

One solution is to use the `site_adjust_merit`() function. When `adjust_merit`() calls `site_adjust_merit`() with `obj` set to 0, this indicates that the user exit should initialize any variables. When an actual `obj` is passed to `site_adjust_merit`(), it contains an associated job and the ranking factor. At this point this user exit returns an increased ranking factor. The amount of the increase determines where in the job list URM puts this job. The closer to the top, the more likely this job is to be recommended by URM before others (if no targets are exceeded in `target_check`()).

The following example code (which should be placed in `site_rank.c`) would do this:

```
/*
 *      Variable hold the static value for site_adjust_merit
 */
static float site_rank_adjust;

/*
 *      float site_adjust_merit() - Adjust the ranking of a job
 *                                  depending on specific needs of the site.
 *
 *      Return the rank this job should have.
 */
float
site_adjust_merit(float   rank,
                  const   Object *obj)
{
    if (!obj) {                         /* initialize values */
        site_rank_adjust = 10.0;        /* set adjustment value */
                                        /* This value may need to be */
                                        /* increased at your site. */

    } else {                            /* else make adjustment */
        if (strncmp(obj->ovalue.job->owner,"user1",5) == 0) {
            rank += site_rank_adjust;
        }                               /* If 'user1' is the owner */
    }                                   /* give a high rank value. */
    return (rank);
}
```

In this example code, `obj->ovalue.job->owner` contains the name of the owner of that job. This example compares name to `user1`, which is assumed to be running the weather model batch job. If true, the rank is adjusted and the new value is returned. By giving a high rank to this job, we are guaranteed that this job will be ordered first (or almost first) on the job list.

Note that if the weather model job does not pass the target checks, it still will not be recommended for execution.

8.5.8.5 URM User Exit Example 2

In this example, a user should be prevented from running batch jobs from 8:00 A.M. to 5:00 P.M. (during peak interactive loads). The URM user exits can be used to prevent this particular user from running batch jobs during this time.

One solution would be to change the ranking factor as in example 1, but instead of adding to the ranking factor, decrease it by a significant amount (make sure the ranking is a positive number). Although this user could still possibly run jobs, those jobs will be very difficult for URM to recommend for initiation.

Another way to prevent the user's batch jobs from running during this time is to use the `site_target_check`() user exit. The following example code (which should be placed in `site_rank.c`) would do this:

```
/*
 *      int peak_inter_time() - determine if between 8:00am and 5:00pm
 */
static int
peak_inter_time()
{
     int rc=0;                          /* 0 = off hours, 1 = peak time */
     time_t timval;                     /* see time(2) man pages */
     struct tm *tmptr;                  /* see time(2) man pages */

     timval = TOTIME(_rtc());           /* TOTIME is a URM macro that acts */
                                        /* like time() system call but uses */
                                        /* the real time clock instead and */
                                        /* does not cause a context switch. */

     tmptr = localtime(&timval);        /* break into components */

     if ((tmptr->tm_hour >= 8)&&(tmptr->tm_hour < 17)) rc = 1;
                                        /* If greater than 8:00am and less */
                                        /* than 5:00pm set the return code. */
```

```
     return (rc);
}

/*
 *      int site_target_check() - Extend the target checking function
 *                                according to needs of the site.
 *
 *      Return appropriate flags to the calling routine.
 *
 *      If init_flag is true, this is an initialization call which
 *      may or may not be useful depending on the nature of the
 *      code.
 */
int
site_target_check(int     flags,
                  Object *obj,
                  struct  _indirect_vals *target,
                  struct  _indirect_vals *load,
                  struct  _total_vals    *total,
                  struct  _need_vals     *need,
                  const   int init_flag)
{
    if (init_flag) {                    /* initialize values */
        /* do initialization */         /* set values */

    } else {                            /* else make adjustment */

        if (strncmp(obj->ovalue.job->owner,"user1",5) == 0) {
                                        /* If 'user1' is the owner, */

            if (peak_inter_time()) {  /* and this is peak interactive time, */
                flags |= _SITE1_TAR;  /* prevent the job from running by */
            }                         /* setting the flag.  target_check() */
                                      /* will set the reason code later. */
        }
    }
    return (flags);
}
```

The `site_target_check`() uses `peak_inter_time`() only to say if the system is currently running in peak interactive time. `site_target_check`() checks to see if this job is owned by `user1`, and, if so, it calls `peak_inter_time`() to find out if this is peak interactive time. If all true, set

the flag value for _SITE1_TAR, meaning that a site-defined target has been exceeded by this job (in this case, the job will **not** be recommended). After returning the flag, target_check() knows that this flag means to set the reason code to UR_SITE1_TAR and the job will not be recommended to run. If user1 were to queue up batch jobs at 4:45 P.M., they will sit in the NQS queue until 5:00 P.M., at which time the user exit no longer flags these jobs and URM begins to recommend that they be run.

8.5.8.6 URM User Exit Example 3

In this example, an NQS queue has been designated as a high priority queue called weather, into which weather model batch jobs are submitted. The URM user exits can be used to ensure that this queue gets high priority.

This example is similar to example 1, but instead of basing priority on the user ID, priority is established by service priority. An assumption is made that NQS queues have unique service priorities and that the weather queue service priority is known inside the user exit. Using site_adjust_merit(), the ranking factor for jobs having the service priority for the weather queue can be increased. The amount of the increase determines where in the job list URM puts this job. The closer to the top, the better chance this job has of being recommended by URM before others (if no targets are exceeded in target_check()).

The following example code (which should be placed in site_rank.c) would do this:

```
/*
 *      Variable site_weather_svcpri holds the static value for svcpri.
 *      Variable site_rank_adjust holds the static value for rank adjustment.
 */
static int site_weather_svcpri;
static float site_rank_adjust;

/*
 *      float site_adjust_merit() - Adjust the ranking of a job
 *                                  depending on specific needs of the site.
 *
 *      Return the rank this job should have.
 */
float
site_adjust_merit(float   rank,
                  const   Object *obj)
{
```

```
    if (!obj) {                          /* initialize values */
        site_weather_svcpri = 40;        /* set 'weather' queue service pri */
        site_rank_adjust = 10.0;         /* set adjustment value */

    } else {                             /* else make adjustment */
        if (obj->ovalue.job->svcpri == site_weather_svcpri) {
            rank += site_rank_adjust;
        }                                /* If svcpri of 'weather' queue, */
    }                                    /* then give a high rank value. */
    return (rank);
}
```

In the example code, `obj->ovalue.job->svcpri` contains the `svcpri` of the batch queue of that job. This example compares an `svcpri` of 40, which is assumed to be the service priority of the `weather` NQS queue, to what is in `obj->ovalue.job->svcpri` for the job. If they are the same, the rank is adjusted and the new value is returned. By giving a high rank to this job, we are guaranteed that this job will be ordered first (or almost first) on the job list.

If the weather model job does not pass the target checks, it still will not be recommended for execution.

8.5.8.7 URM User Exit Example 4

This example shows how to modify the `site_target_check`() function so that NQS jobs from queues with a specific service priority will always be recommended for initiation.

```
#define EXPRESS_PRI 15

/*
 * This module is reserved for local use.
 *
 * Please see rank.c for the calls to these function.
 */

/*
 * int site_target_check() - Extend the target checking function
 * according to needs of the site.
 *
 * Return appropriate flags to the calling routine.
 * Zero means no targets exceeded.
 * If init_flag is true, this is an initialization call which
```

```
 * may or may not be useful depending on the nature of the code.
 */

int
site_target_check(int flags,
                  Object *obj,
                  struct _indirect_vals *target,
                  struct _indirect_vals *load,
                  struct _total_vals *total,
                  struct _need_vals *need,
                  struct _max_vals *max,
                  const int init_flag)
{
    Job     *jp;
    char    msg[132];

    if (init_flag) {
        /*
         *      Do nothing for initialization.
         */
        return(flags);
    }
    jp = obj->ovalue.job;     /* Get the job object pointer */

    if (jp->svcpri == EXPRESS_PRI) {
        /*
         *      If the job's svcpri is EXPRESS_PRI, format and write
         *      a log message.
         *      Returning zero makes sure this job is not removed
         *      from consideration by the target checking rules.
         */

        sprintf(msg, "Batch job %d in NQS express queue (pri %d) cleared",
                jp->svcid, EXPRESS_PRI);
        write_urm_log(ESTATE_INFO, msg);
        return(0);

    } else {
```

```
        /*
         *      Jobs that are not EXPRESS_PRI get no special treatment;
         *      'flags' is returned without change.
         */
        return (flags);
    }
}
```

8.6 Troubleshooting URM

The URM is part of the Cray Message System and provides explicit indications of the cause of many error conditions and statuses. Both the `urmd`(8) daemon process and the `rmgr`(1) user interface command access the message system.

8.6.1 URM Daemon Failures

If the `urmd` process fails to start or aborts during execution, check the log file for an indication of the reason. Reasons for the `urmd` process to fail include:

- The `urm` service name does not exist in the `/etc/services` file. Use the menu system to create the `urm` service port.
- The `/bin/rmgr` command does not exist.
- The `/etc/urmd` process was not initiated by a privileged process.
- Configuration files either have not been installed in `/etc/config/urm` or contain invalid commands.
- Files with the same names as the configuration files exist in `/usr/adm/urm`. To open a configuration file, `rmgr` first checks its current directory (during initialization, `/usr/adm/urm`) before the configuration directory (`/etc/config/urm`). The critical file names are `configuration`, `init`, `local`, `machine`, `res`, `structure`, `urminfo`, and `val`.

The `urmd` process can be restarted on an active system. It reads the session table to determine the current state of the machine and continue from there. It is important to reconnect the NQS daemon to the restarted `urmd` process. To do so, use `qmgr` directives to set URM scheduling on and to set URM restart on. For details of enabling URM services in NQS, see the UNICOS NQS and NQE Administrator's Guide, publication SG-2305.

If the `rmgr` command fails to connect to a `urmd` process on a remote machine, it is likely that the URM configuration on the remote machine does not have this machine in its `/hosts` list.

Note: To execute the following procedure and get the expected results, you must be in the URM authorized administrator list (`/admin/privileged`).

To view which machines are allowed to connect remotely, use the `rmgr` command, followed by the `view /hosts` option:

```
# rmgr
rmgr-> view /hosts
-Vrw--  0    0 Jan  4 13:22 <*          >  "null"
```

In this case, the * shows that remote connections are accepted from all hosts (as long as user validation also passes). If the desired host is not in the hosts list, use the menu system to add the host(s) to the URM configuration.

If a non-superuser process fails to obtain a privileged connection to the `urmd` process even though their name is in the URM `/admin/privileged/` list, ensure that the `/bin/rmgr` executable has been installed properly. This program must be installed as `setuid-root` in order to use a privileged socket connection with the `urmd` process.

8.6.2 URM and NQS

The URM and NQS work closely together to maintain the desired system loads. For NQS to work with URM, the NQS daemon must be configured to communicate with the `urmd` process. To verify this communication, first make sure that the `urmd` process is executing (and if not, restart it). Then check the NQS configuration parameters to verify that URM scheduling is turned on (the NQS `set job scheduling` *option* parameter must be set to the correct *option*). For information on displaying the NQS configuration parameters, see the `qmgr`(8) man page. For information on changing NQS configuration parameters, see the UNICOS NQS and NQE Administrator's Guide, publication SG-2305.

To manually establish (or re-establish) communication between NQS and URM on a running system, you can use the `qmgr`(8) command, specifying the appropriate subcommand.

To return NQS to its default mode (disable URM control), enter the following `qmgr` subcommand:

```
set job scheduling nqs normal
```

For more information on the `qmgr` command and subcommands, see the `qmgr`(8) man page.

8.6.3 URM and the Fair-share Scheduler

The URM considers fair-share usage information from the user data base (UDB) when prioritizing batch jobs. If your system has the fair-share scheduler turned on, it must be functioning properly, that is updating the usage information in the UDB.

This section explains the three fair-share components used by URM: share priority weight, share entitlement weight, and usage weight. For more information on fair-share, see "Fair-share Scheduler", Chapter 4, page 191.

8.6.3.1 Share Priority Weight

In a system using the fair-share scheduler, URM evaluates the effective share of each resource group and stores the information in a table named `/share` in the URM object tree. Each of the table entries holds the relative share of its resource group. The groups are converted from the hierarchy into a flat organization for ease of searching and evaluation. Relative shares are not adjusted for usage in the object tree.

URM evaluates the share organization at a rate determined by `/urm/share_eval` (the time to go before the next evaluation is found in `/urm/share_to_go`). Normally, the resource group hierarchy and the relative priorities among the resource groups do not change frequently. If the administrator changes the organization of the resource hierarchy or the relative shares within the hierarchy, URM can be forced to regenerate its internal view of the resource groups by setting `share_to_go` to 0.

When a job is being evaluated, the user's resource group is determined as input to the share evaluation function. The entitlement of the user is determined as described in the following section. Next, usage is determined as described in the "Usage weight" section, Section 8.6.3.3, page 391 (the method depends on the setting of `share_policy`), and the priority is determined from the following expression:

$$priority = usage / entitlement^2$$

This is the same expression that is used in the kernel to compute `kl_usage`, which is the main component of `p_sharepri`. Numerically smaller values are better. If the fair-share scheduler is not active, the value of this component is 0.

Each job's share priority for ranking is determined from the following expression:

share_wt`*(1.0-(`*job_priority/max_priority*`))`

Share priority ranking is a combination of usage and entitlement ranking available separately. It is expected that either share priority ranking or a combination of usage and entitlement ranking would be chosen, but not some combination of share ranking and usage or entitlement ranking. This capability offers additional flexibility in ranking choices.

8.6.3.2 Share Entitlement Weight

The share entitlement at each level in the fair-share hierarchy is determined from the following expression:

entitlement=node_share/group_share

The value *group_share* is the total number of shares allocated in the resource group. This means that a node's entitlement is the fraction of the total shares of the group assigned to that node, without regard to how many nodes in the group have activity. This is not how the fair-share scheduler evaluates a user's entitlement, but an exact method is very difficult to implement when some of the users or resource groups waiting to have jobs initiated may not be active. The intent is to determine the relative entitlements of a number of users competing for initiation.

Each level of the hierarchy is evaluated as described in the preceding paragraph, and the product of the entitlements of each level in the hierarchy becomes the user's entitlement. This value is returned to the share priority calculation and to the ranking evaluator separately to be normalized and weighted with the `entitle_wt` factor.

Numerically larger values are better. If the fair-share scheduler is not active, the value of this component is 0. Each job's entitlement for ranking is determined from the following expression:

entitle_wt`*(`*job_entitlement/max_entitlement*`)`

8.6.3.3 Usage Weight

Usage is derived from the decayed usage as maintained by the fair-share scheduler in either the lnode or the UDB. When URM examines a job, it checks for current active usage by using the `limits`(2) system call. If the user is active (has an lnode), current usage is returned. If the user is not active, the user's

record is read from the UDB, and the current usage value is calculated from the recorded usage decayed over the time period since the user last used the machine.

If `share_policy` is `Standard`, only the terminal node's usage is returned. If `share_policy` is `Fair_ratio`, usage of the subject node, proportional to the sum of the usage in each level of the hierarchy, is multiplied together to form the usage value. Usage supplied to the share priority component is the value determined by `share_policy`; this value is also returned to the ranking evaluator separately to be normalized and weighted with the `usage_wt` factor.

Numerically smaller values are better. If the fair-share scheduler is not active, the value of this component is 0. Each job's usage for ranking is determined from the following expression:

usage_wt`*(1.0-(`*job_usage/max_usage*`))`

8.7 URM Architecture

The URM includes three servers: the `urmd` selection server, the `rmgr` query and command server, and the `sdsmgr` job server. The following sections describe these URM servers.

8.7.1 Selection Server (`urmd`)

The `urmd` selection server (the URM daemon) accepts job selection requests and responds with recommendations. This is the server that NQS, `login`, and other session initiators use to get initiation recommendations from URM. This server uses information about the state of the system and the backlog to recommend which job or jobs should be initiated. If any of the resources required by the job are not available, a recommendation is not issued and the job should not be started, because it may block or fail during its device reservation process.

The initiating service makes requests of the selection server by presenting a single job to add to the current backlog or by presenting its entire backlog of jobs otherwise eligible to initiate. There is also the capability of removing a job from the URM backlog. These functions are intended to provide enough information internal to URM to know the backlog, while giving flexibility to the service to add, delete, or rearrange jobs as required within the scope of the service policy. Services such as `login` that do not have a backlog simply present a job for evaluation. URM assumes that when a single job is presented for evaluation, a positive recommendation implies job initiation.

Device availability checks must succeed for URM to recommend initiation of a job. Resource requirements known to the selection server do not affect the execution of the job, because nothing is really reserved until the job initiates and makes its own reservation requests. This approach is taken to prevent incomplete or inaccurate resource information from affecting the actual execution of jobs.

The selection server expects that all relevant information about the resources needed by a job will be presented to it when a recommendation is requested. Only in this way are useful recommendations possible. All of the URM attributes associated with the job should be presented so they can also be evaluated. URM evaluates these attributes and recommends initiation only when all of the requirements are satisfied. Other information needed is the identification of the service (NQS, interactive, and so on) and, for each job, the following information:

- User name and/or user ID
- Memory and SDS size limits
- CPU time limit and nice value
- Tape usage information and other resource requirements of the job (batch only)
- Fair-share resource group ID
- Job priority (minimum rank)

8.7.2 Query and Command Server (`rmgr`)

The `rmgr` query and command server accepts requests for status information and replies to such requests as appropriate. This server is intended to support the needs of network-level scheduling and local administrators. This server also provides the configuration capability to URM.

For more information about the query and command server, see the `rmgr`(1) man page.

8.7.3 SDS Management (`sdsmgr`)

URM includes the `sdsmgr` job server, which monitors any interactive or batch session that uses secondary data space (SDS) in the SSD. When the `sdsmgr` is enabled, SDS space can be oversubscribed in the same manner as memory can be oversubscribed, by swapping SDS to `swapdev`. SDS in use and SDS changes

requested by all sessions are determined from the session table by the `kerninfo.c` module of URM, which gathers load information. `sdsmgr` reads the `sdsmap` information to obtain the amount of SDS that is available but not in use at that time. If the amount of SDS in use at a particular point in time is greater than the physical amount of SDS space available after `ldcache` allocation, SDS is considered to be oversubscribed.

Changing the state of a session from active use of SDS to waiting for a turn at using this resource is termed *preemption*. This releases the SDS space for use by another session. Preemption of SDS space is done via the `suspend` () system call, which causes both process memory and SDS space for the session to be written to the swap device. Restoration of a session to a running state is accomplished by using the `resume` () system call, which removes the `PC_SSPND` flag from each process. When `sched` makes each process eligible for running and `swapper` swaps it in, the SDS space is reallocated. If the required SDS space is not available, the process is not swapped in.

To manage SDS, `sdsmgr` first makes a list of sessions using SDS; this list is ordered by nice value and time-in-queue. Preference is given to lower nice values and older jobs in order to provide fastest wall-clock turnaround. The amount of SDS in use (`s_sdsuse` in the session table) is further broken down into SDS in core and SDS swapped out (`S_SUSPNDED` flag set). The amount of SDS requested (`s_sdsreqsz` in the session table) is further broken down into additional SDS needed for `swapin` and SDS needed to grow in core. As long as no session needs more SDS, and no session needs to be swapped in to continue running, `sdsmgr` takes no actions.

If, after accounting for swapped sessions, there is a session trying to `ssbreak` for more SDS space, `sdsmgr` looks for a preemption candidate. Sessions with SDS are guaranteed not to be preempted for a specific length of wall-clock time (known as a *residence interval*), as long as they do not change their SDS allocation. Sessions in core that are waiting on `ssbreak` (that is, the kernel cannot allocate a larger SDS area) are preempted first. Exceptions to this rule are sessions that have just been initiated (`sp->s_sdsuse == 0` and `sp->s_sdsreqsz > 0`). These sessions are given a chance to acquire SDS and start running before being preempted. Because NQS jobs acquire all their SDS specified on the `qsub` on the first `ssbreak`, NQS jobs should not be preempted when they first start running, as happened with `qfdaemon`. This behavior also assumes that users have not changed the environment variables `SDSLIMIT`, `SDSINCR`, and `SDSMAXFR`.

If no sessions trying to grow their SDS space are found, sessions that have had SDS allocated longer than the configured SDS residence interval are considered for preemption next. Sessions are preempted until enough SDS space has been

freed up to satisfy the in-core SDS requested. If SDS space is available, and there are swapped out sessions, the job that has been out the longest, has the most favorable (lowest) nice value, and fits in the available SDS is swapped in. As long as as there are no sessions needing SDS to be swapped in, sessions currently using SDS are not preempted, even if they have exceeded the SDS residence interval. The `sdsmgr` does only one preempt for residence time exceeded and one restore per URM cycle.

8.8 URM Resources

This section describes the resources that URM controls. These are the resources usually associated with physical objects and with such notions as available devices or capacity.

URM resources include the following:

Resources	Description
CPU time	Like memory, this resource can be used to control job initiation but is more likely to be used to group jobs of similar duration for selection decisions.
Memory	Memory is not strictly a controlled resource, but URM manages oversubscription (swap control) and recommends against initiating jobs that require memory usage above a specified maximum. This is mostly needed for administrative controls, such as preventing jobs needing more than a specified amount of memory from being initiated during periods of heavy interactive use. Another use of this resource is to group jobs of similar usage, so that jobs with large memory requirements need not compete against one another during execution.
MPP	The MPP resources can be configured to allow multiple user applications to run concurrently (space sharing). The MPP hardware resources that must be shared are the barrier bits and the processing elements (PEs). URM monitors the MPP barrier synchronization mechanism (barrier pools for user-designated bits and for operating-system-designated bits). URM monitors the MPP administrative resource pools (sets of PEs designated for use by batch, interactive, or both job types).
SDS	Allocation of SDS space is evaluated both to deliver service to deserving jobs in the proper order and to manage resource conflicts to avoid oversubscription beyond manageable limits.

	User limits on SDS space are controlled through the UDB limits on batch and interactive.
Share	Although shares, as used in the fair-share scheduler, are not resources in the sense of tapes, each user has potential to do work based on a priority derived from the share. Because this priority depends on the user's history, as well as the set of users active on the machine at a given time, URM must consider the effective priority of each request in the initiation recommendation.
Tape	URM supports tape resources as defined by the UNICOS tape subsystem. Eight different user limits are available to control tape resources.

8.9 URM Checkpointing

URM can be configured to checkpoint interactive sessions at shutdown, if requested by the owner of the session. The resulting restart images are kept in the URM `chkpnt` directory.

URM can also be configured to do periodic checkpoints of interactive or batch sessions, if requested by the owner of the session. URM sends checkpoint notices to NQS when a batch session requires periodic checkpointing; the resultant restart files are kept in the NQS `chkpnt` directory. URM manages restart files for interactive sessions in a special URM `chkpnt` directory.

8.9.1 Configuring URM Checkpointing

Checkpointing is done only on an individual session basis. The `chkptint`(1) command provides the mechanism by which any user is able to specify that checkpointing be done at shutdown for an interactive session or that periodic checkpointing is desired for an interactive or a batch session.

This command has one required option, of the form `-s` *sec*, which specifies the requested checkpoint frequency in seconds. When URM is running in checkpoint only at shutdown mode, all that is required is that the `chkptint -s #` be nonzero; `chkptint -s 0` can be used to turn off the automatic or periodic checkpoint request for that session. The `chkptint` command can be part of a script or placed in a user's `.cshrc` or `.profile` file.

The following URM objects (as taken from a `rmgr-> view /urm` display) are used to configure URM checkpoint features:

```
LVrwr-  0    0 Dec 28 08:13 <restart_switch>  "Force"
LVrwr-  0    0 Dec 28 08:13 <chkpnt_switch >  "Auto"
LVrwr-  0    0 Dec 28 08:13 <min_interval  >   Int, 1800
LVrwr-  0    0 Dec 28 08:13 <interval_type >  "CPU, Clock"
LVrwr-  0    0 Dec 28 08:13 <retain_chkpnt >   Int, 432000
LVr-r-  0    0 Dec 28 08:13 <shutdown      >   Int, 0
LVr-r-  0    0 Dec 28 08:13 <shut_done     >   Int, 0
LVrwr-  0    0 Dec 28 08:13 <restart_cmd   >  "UPATH/irstart"
LVrwr-  0    0 Dec 28 08:13 <chkpnt_path   >  "PPATH/chkpnt"
LVrwr-  0    0 Dec 28 08:13 <chkpnt_cmd    >  "UPATH/intchkpt"
```

The `restart_switch` object indicates whether or not to use the `RESTART_FORCE` flag on the restart system call when restarting an interactive session.

The `chkpnt_switch` object defines the type of checkpointing desired. `No` turns both features off; `Shutdown` turns on checkpointing for interactive sessions at shutdown; and `Auto` turns on periodic checkpointing of batch and interactive sessions and specifies checkpointing for interactive sessions at shutdown as well.

The `min_interval` object defines the minimum interval (in seconds) between checkpoints with which URM will comply. If a user requests a checkpoint interval less than this minimum, the minimum will be used. This constraint applies only to periodic checkpoint behavior.

The `interval_type` object defines the checkpoint criteria. `CPU` means user-plus-system CPU time; `Clock` means elapsed wall-clock time. This constraint applies only to periodic checkpoint behavior.

The `retain_chkpnt` object defines the length of time (in seconds) URM will keep a restart image before deleting it. Deleting these images prevents taking up disk space with unwanted restart files.

The `shutdown` and `shut_done` flags indicate whether or not shutdown is in progress or completed. When `shutdown` and `shut_done` are both 0, no shutdown is in progress. If `shutdown` is equal to 1, URM goes through the process of looking for sessions to checkpoint, but does not terminate when completed. If `shutdown` is equal to 2, a checkpoint and terminate sequence has been requested. When all shutdown checkpointing is complete, `shut_done` is set to 1.

The `restart_cmd` object defines the name of the command `rmgr` executes to restart a selected interactive session.

The `chkpnt_path` object defines the path to the URM checkpoint directory. The default is `/usr/adm/urm/chkpnt`.

The `chkpnt_cmd` object defines the name of the command URM spawns as a separate process to checkpoint an interactive session.

For periodic checkpointing of interactive sessions, URM scans the session table, notes when a checkpoint interval has been set by using the `chkptint` command, and sets a timer for that session. When the requested CPU or wall-clock interval has elapsed, URM will fork and `exec` a process to do the checkpoint for the interactive session. The timer is reset after a successful checkpoint.

When URM determines that it is time to do a periodic checkpoint of a batch job, URM sends a `chkpnt` request, defined as part of the interface between NQS and URM, along with job sequence number, MID, and SID. NQS creates the restart file in the NQS checkpoint directory, as if a `qchkpnt` command had been issued by the user.

8.9.2 Managing Restart Images

Several `rmgr` subcommands aid in the management of restart images. Upon logging in, an interactive user can query URM, using the `rmgr` utility `view restart` subcommand, to see if the user has any checkpointed sessions available.

```
rmgr-> view restart kcz
Restart images belonging to User <kcz>, UID 343 on path /ptmp/urm/chkpnt/kcz
        <1> 12281123.2640
        <2> 12281153.2640
        <3> 12281223.2640
User <kcz> has 3 restart images.
```

To decide which of the saved restart images to try to restart, a user can use the `rmgr` utility `view restart` subcommand to obtain a detailed description of the restart image.

```
rmgr-> view restart kcz 2
Restart file /tmp60/kcz/chkpnt/kcz/12281153.2640 belongs to uid 343
Restart file 12281153.2640 for session 2640 created  Dec 28 11:53

  Restart file session characteristics:
  #procs = 5;  #tasks = 5;  pgrp inheritance = 69747;  nice value = 20.
  PID of session parent = 69746;  PID of foreground process group = 76223.
```

```
  User cputime = 118212137; system cputime = 196492829
        cpulimit[0] = 4611686018427387903
        cpulimit[1] = 4611686018427387903
        cpulimit[2] = 4611686018427387903
  Memory in use = 868; memhiwat = 2047; memlimit = 35184372088831
  SDS in use = 0; sdshiwat = 0; sdslimit = 900000
  Max #of procs allowed = 100

  Mtask    1: name: csh pid: 69747  #sibs:    1.
        process group pid = 102032, parent pid = 69746
        Memory size= 124 klics, swap_image size= 0 klics.
        SDS allocated= 0 klics, SDS requested= 0 klics.

  Mtask    2: name: zup pid: 70005  #sibs:    1.
        process group pid = 102032, parent pid = 69747
        Memory size= 156 klics, swap_image size= 0 klics.
        SDS allocated= 0 klics, SDS requested= 0 klics.

  Mtask    3: name: csh pid: 70030  #sibs:    1.
        process group pid = 102032, parent pid = 70005
        Memory size= 96 klics, swap_image size= 0 klics.
        SDS allocated= 0 klics, SDS requested= 0 klics.

  Mtask    4: name: man pid: 76223  #sibs:    1.
        process group pid = 102032, parent pid = 70030
        Memory size= 148 klics, swap_image size= 0 klics.
        SDS allocated= 0 klics, SDS requested= 0 klics.

  Mtask    5: name: more pid: 76224  #sibs:    1.
        process group pid = 102032, parent pid = 76223
        Memory size= 112 klics, swap_image size= 0 klics.
        SDS allocated= 0 klics, SDS requested= 0 klics.
User <kcz> has 2 restart images.
```

The `rmgr` utility `delete restart` subcommand allows users to delete unwanted saved restart images. For example:

```
rmgr-> delete restart kcz 1
Deleted restart image <1> for user kcz
rmgr->
```

To request that a previously saved session be restarted, a user first views the saved images and then specifies that a particular image be restarted. The existing session with `rmgr` is eliminated, and the terminal connected to the login shell is saved in the restart session.

The following example restarts a saved session that was in the middle of a `man ps` command. Output resumes where it left off when the job was saved.

```
rmgr-> view restart kcz
Restart images belonging to User <kcz>, UID 343 on path /tmp60/kcz/chkpnt/kcz
        <1> 12281153.2640
        <2> 12281223.2640
User <kcz> has 2 restart images.

rmgr-> restart kcz 1
     When a process has exited and has a parent, but the parent has not
     waited for it, the process is marked <defunct>.

   Format Specification
     The -o option allows the output format to be specified under user
     control.  The format specification consists of a list of field names
--More--
```

8.9.3 Checkpointing at Shutdown

The `rmgr` subcommands used in a controlled shutdown are `set shutdown stopdaemon`, or `set shutdown` and `stopdaemon` used separately and sequentially. When `set shutdown stopdaemon` is used, `urmd` terminates after all checkpointing has completed. When `set shutdown` is used, any requested checkpoints are performed, but `urmd` continues. When `stopdaemon` is used, `urmd` terminates without performing any checkpointing.

8.10 Tuning URM

This section provides information specific to tuning the Unified Resource Manager (URM). It covers the following topics:

- Tuning URM control settings
- Tuning URM job selection criteria
- Using URM with NQS

The examples in this section demonstrate changing URM settings with the `rmgr`(1) command. Any changes made with `rmgr` affect the running version of URM only; when the URM daemon is stopped, these changes are lost. To make permanent changes to URM settings, use the menu system (`install`(8) command), as described in Section 8.4, page 359.

8.10.1 Tuning URM Control Settings

URM is released with a set of default control settings, including the following:

- Machine target values (`/machine/target`)
- Monitoring cycles (`/urm`)
- Group scheduling control (`/machine/target/group_sched`)
- Load smoothing factors (`/machine/rate`)

The remainder of this section describes how to change each of these control settings.

8.10.1.1 Machine Target Values

The URM machine target values set system-wide resource usage boundaries. URM does not recommend initiation of jobs specifying resources that exceed these targets.

When determining the recommendation status of each job, URM compares system resource loads against the following targets:

- Memory oversubscription target
- SDS oversubscription target
- Maximum session target
- Tape limit target
- MPP limit targets

The values for the tape and MPP limit targets are automatically updated by URM through its automatic configuration capability and should not be changed manually. However, if your site does not have tapes, the automatic configuration of tape usage targets should be disabled.

Use the `rmgr` command to view the current machine target values, as shown in the following example:

```
rmgr => view /machine/target
        .
        .
        .
 LVrwr- 0 0 Sep 16 06:21 <sds_os > Float, 2.000000
 LVrwr- 0 0 Sep 16 06:21 <memory_os > Float, 2.000000
        .
        .
        .
 LVrwr- 0 0 Sep 16 06:21 <tape > Int, 2
 LVrwr- 0 0 Sep 16 06:21 <jobcount > Int, 300
 LVrwr- 0 0 Sep 16 06:21 <pe > Int, 0
 LVrwr- 0 0 Sep 16 06:21 <bb > Int, 0
```

The `/machine/target` display also shows other URM values, such as the group scheduling controls; these values are discussed in following sections.

To change a machine target value, use the following `rmgr` directive:

`/machine/target/`*resource*`=`*value*

To permanently change the configuration, use the following menu in the menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Machine target values
```

The remainder of this section describes the configurable machine targets for the memory, SDS space, and active jobs.

8.10.1.1.1 Memory Oversubscription Target

The amount of memory in use is called the *memory load*. URM computes the memory load by adding the size of all running jobs and recommended jobs. The size of a running job is calculated from the actual size in the kernel session table, *smoothed* by the requested size of the job (see Section 8.10.1.4, page 410). The size of a recommended job is the amount of memory the job requested. If a job did not request memory requirements (that is, interactive jobs), the default memory usage is used; see Section 8.10.2.2.2, page 421, for more information.

The memory oversubscription target, `/machine/target/memory_os`, sets an upper limit on the total amount of user memory (including swap space) that can be in use at the same time. If the actual memory load matches or exceeds the

target value, no jobs are initiated until the load drops. This limitation includes only jobs from session initiators that have been configured to allow URM control of job initiation; for more information, see Section 8.10.2.2.2, page 421.

Estimating memory use in this manner allows URM to plan ahead for future memory use, if smoothing factors are used, because most jobs do not use the full amount of declared memory until they have been running for a period of time.

The memory oversubscription target is a multiplier, or percentage value, instead of an absolute value. The default multiplier, 2.0, sets the memory target to twice the total amount of user memory configured.

If a high amount of swapping is affecting system performance, consider decreasing the memory oversubscription multiplier. For example, to change the memory oversubscription target to 1.5, enter the following `rmgr` directive:

```
/machine/target/memory_os = 1.5
```

To permanently change the configuration, use the following menu in the menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Machine target values ->
        Memory oversubscription multiplier
```

Note: The maximum value for the memory oversubscription target is limited by the amount of swap space configured on the system. Do not set this target to a value larger than the amount of available swap space. (If SDS space is configured, the sum of the memory oversubscription target and the SDS oversubscription target should be less than or equal to the amount of available swap space.)

The following example demonstrates how to calculate a value for memory oversubscription that corresponds to previous usage of the system by using the data from the system `sar`(1) logs. For example, assume that the `sar` log for a typical day shows the following average values for memory use:

- 53730 clicks of total user memory available (*umemtot*)
- 38144 clicks of user memory in use (*umemuse*)
- 46769 clicks of swap space in use (*swapuse*)

Use the following equation to calculate the memory oversubscription target value for URM:

memory_os = (*umemuse* + *swapuse*) / *umemtot*

= (38144 + 46769) / 53730

= 1.6

Setting `/machine/target/memory_os` to 1.6 will result in memory use with URM that is similar to previous system behavior.

8.10.1.1.2 SDS Oversubscription Target

The SDS oversubscription target, `/machine/target/sds_os`, sets an upper limit on the total amount of SDS space that can be in use at the same time. If the actual SDS load matches or exceeds the target value, no jobs requesting SDS space are initiated until the load drops.

The SDS oversubscription target is a multiplier, or percentage value, instead of an absolute value. The default multiplier, 1.5, sets the SDS target to one and one-half times the amount of available SDS space. This target could be increased to allow URM to recommend more jobs using SDS space (which might increase swapping as well), as in the following example:

```
/machine/target/sds_os = 2.0
```

To permanently change the configuration, use the following menu in the menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Machine target values ->
        SDS oversubscription multiplier
```

Note: The maximum value for the SDS oversubscription target is limited by the amount of swap space configured. Do not set this target to a value larger than the amount of available swap space. (If SDS space is configured, the sum of the memory oversubscription target and the SDS oversubscription target should be less than or equal to the amount of available swap space.)

In addition to monitoring SDS load, URM manages SDS space by preempting the jobs using SDS to allow other jobs a chance to use SDS space. (Both batch and interactive jobs are affected by this feature.) For more information, refer to Section 8.10.1.2.5, page 407.

8.10.1.1.3 Maximum Session Target

The maximum session target, `/machine/target/jobcount`, determines the boundary at which URM stops recommending the initiation of any more jobs. By default, this target is set to the size of the kernel session table (set by the `NSESS` parameter).

> **Note:** Setting the active job target to too small a value could result in wasted CPU resources, because URM would not recommend jobs even when adequate resources are available. To control job count and prevent the system from becoming overcommitted, use the job targets for the individual session initiators. Refer to Section 8.10.2, page 414, for more information on these targets.

8.10.1.2 Monitoring Cycles

The URM monitoring cycles control the frequency of load monitoring, batch job ranking, and SDS preemption. These cycles define the minimum possible delay for initiation of batch jobs. (Interactive jobs are always handled immediately.) The controlling monitoring cycle is called the *main loop*; this cycle calls all other monitoring cycles. Therefore, all other cycles cannot occur more frequently than the main loop cycle.

The monitoring cycles can be changed to increase or decrease URM's sensitivity to fluctuations in job load. This section describes changing the interval, or delay, for the main loop and the subsidiary cycles.

8.10.1.2.1 URM Main Loop

The URM main loop consists of the following operations:

Operation	Description
Kernel information check	Checks system load and configuration information, monitors SDS usage
Job scheduling check	Ranks and recommends any batch jobs that are waiting in its backlog
Share evaluation check	Reevaluates the fair-share hierarchy

SDS residence management Manages SDS usage

The main loop is controlled by the main loop delay, `/urm/sleep_time`; it is set to 10 seconds by default. This delay specifies the amount of time URM waits before performing the subsidiary cycles. Each subsidiary cycle in the main loop has its own delay; these should be set to a multiple of the main loop delay. If they are not, the main loop delay will take precedence.

To increase the main loop delay to 20 seconds, for example, enter the following `rmgr` directive:

```
/urm/sleep_time = 20
```

To permanently change the configuration, use the following menu in the menu system. However, it is recommended that you **not** change this value.

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM control settings ->
        Main loop sleep period
```

8.10.1.2.2 Kernel Information Check

The kernel information delay, `/urm/info_delay`, controls the frequency of checks for resource loads and configuration information; it is set to 10 seconds by default. Changing this value is **not** recommended. Refer to Section 8.10.1.4, page 410, for more information on the kernel information delay.

8.10.1.2.3 Job Scheduling Check

The job scheduling check controls the frequency of batch job ranking and batch job recommendation. During the job scheduling check, URM ranks all jobs in the backlog and makes any possible recommendations.

The job scheduling delay, `/urm/sched_delay`, is set to 10 seconds by default. This delay, along with the main loop delay, defines the actual minimum delay for batch jobs to receive an initiation recommendation after they are registered with URM.

8.10.1.2.4 Share Evaluation Check

If the fair-share scheduler is enabled, URM periodically evaluates the machine share (normalized share) of each resource group or shareholder (account ID) for use in the batch job ranking calculation.

The share evaluation cycle is controlled by the share evaluation delay, `/urm/share_eval`, which is set to 900 seconds (15 minutes) by default. Resource group hierarchies and relative priorities among the resource groups tend to change infrequently, so this evaluation is not performed as often as the other cycles.

The countdown timer for the fair-share evaluation period, `/urm/share_to_go`, contains the amount of time remaining until the next evaluation. Use the following `rmgr` directive to view this value:

```
view /urm/share_to_go
```

If relative fair-share priorities change on a one-time basis (for example, when the share allocations are changed in the UDB), set `/urm/share_to_go` to 0; this ensures that share evaluation will be performed during the next main loop cycle.

For more information on the fair-share scheduler, see "Fair-share scheduler," Chapter 4, page 191.

8.10.1.2.5 SDS Residence Management

URM controls SDS oversubscription by preempting jobs, both batch and interactive, to ensure that other jobs waiting for SDS space get a chance to use it. Once during each main loop, URM checks to see if SDS space is in use, then calls the `sdsmgr` program to handle the actual preemption.

The SDS residence interval is controlled by `/urm/sds_residence`; this value is set to 900 seconds (15 minutes) by default. You can disable URM management of SDS space by setting `/urm/sds_residence` to 0.

To permanently change the configuration, use the following menu in the menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM control settings ->
        SDS residency time in secs
```

The following example shows the `view sds` directive to the `rmgr` command on a system with SDS oversubscription:

```
rmgr-> view sds

SDSwork: SDS residence interval set to 900
Number of sessions using SDS  = 2
Number of preempted SDS jobs = 0

SDS jobs:
         preempted   = 0
         restoring   = 0
         swapped     = 1
         in core     = 1

SDS units:
         physical size            12288
         allocated                23296
         requested                0
         available                128
         in memory                12160
         swapped out              11136
         needed by running   jobs 0
         needed by suspended jobs 0
         being preempted out      0
         being restored to memory 0
SDS state is OVERSUBSCRIBED by 11008
```

8.10.1.3 Group Scheduling Control

During the batch job ranking phase (see Section 8.10.2.1.1, page 415), the default behavior for URM is to select the jobs with the best (highest) rank to recommend for initiation. However, some jobs might receive a consistently low rating because of atypical resource usage. Using rank alone as a selection criterion can prevent these jobs from being recommended for initiation on busy systems. To prevent this situation, URM has a *group scheduling control* feature to control job recommendations by group characteristics, or similarity of resource usage, as well as the batch job ranking calculation.

By default, group scheduling control is disabled. All jobs fall into the same default group (also called the *batch job pool*). Group scheduling control is enabled by setting `/machine/target/group_sched` to 1. When this feature is enabled, URM divides all batch jobs into four groups (defined by computing

the average rank and standard deviation) based on how similar each job is to the average job. Group 1 contains jobs closest to the average, and group 4 contains jobs farthest from the average. URM stores a count of jobs for each group in four `/machine/target/jobs_in_sg` values.

URM also monitors the recommendation history for each group, that is, the number of jobs selected, or *picked*, from each group. The group scheduling control feature allows you to change the job selection percentage to favor unusual or low ranked jobs by setting the *pick value* (selection percentage) for each group. The pick values define how often a job is selected from each group. URM stores integer values in four `/machine/target/pick_in_sg` values; these are converted to relative values, or percentages. By default, a total value of 20 is used to divide the group pick values; the pick value for group 1 is 9 (45% of jobs are selected from this group), group 2 is 7 (35%), group 3 is 3 (15%), and group 4 is 1 (5%). The recommendation history is stored in four `/machine/target/rec_from_sg` values.

The following `rmgr` directive displays sample URM values used for group scheduling control:

```
rmgr => view /machine/target
                .
                .
                .
 LVrwr- 0 0 Sep 20 07:23 <group_sched > Int, 1
                .
                .
                .
 LVr-r- 0 0 Sep 20 07:23 <jobs_in_sg1 > Int, 20
 LVr-r- 0 0 Sep 20 07:23 <jobs_in_sg2 > Int, 11
 LVr-r- 0 0 Sep 20 07:23 <jobs_in_sg3 > Int, 4
 LVr-r- 0 0 Sep 20 07:23 <jobs_in_sg4 > Int, 2
 LVr-r- 0 0 Sep 20 07:23 <rec_from_sg1 > Int, 45
 LVr-r- 0 0 Sep 20 07:23 <rec_from_sg2 > Int, 35
 LVr-r- 0 0 Sep 20 07:23 <rec_from_sg3 > Int, 15
 LVr-r- 0 0 Sep 20 07:23 <rec_from_sg4 > Int, 5
 LVrwr- 0 0 Sep 20 07:23 <pick_in_sg1 > Int, 9
 LVrwr- 0 0 Sep 20 07:23 <pick_in_sg2 > Int, 7
 LVrwr- 0 0 Sep 20 07:23 <pick_in_sg3 > Int, 3
 LVrwr- 0 0 Sep 20 07:23 <pick_in_sg4 > Int, 1
                .
                .
                .
```

The line for `group_sched` shows that group scheduling control is enabled. The number of jobs in each group are displayed in the values `jobs_in_sg1`, `jobs_in_sg2`, `jobs_in_sg3`, and `jobs_in_sg4`. The recommendation history is shown in the values `rec_from_sg1`, `rec_from_sg2`, `rec_from_sg3`, and `rec_from_sg4`. The default pick values are shown in `pick_in_sg1`, `pick_in_sg2`, `pick_in_sg3`, and `pick_in_sg4`.

To change the percentages for group selection, modify the `pick_in_sg` values. The following example enables group scheduling control and sets pick values so that jobs are selected from each group on an equal basis:

```
/machine/target/group_sched = 1
/machine/target/pick_in_sg1 = 1
/machine/target/pick_in_sg2 = 1
/machine/target/pick_in_sg3 = 1
/machine/target/pick_in_sg4 = 1
```

Note: The sum of the `pick_in_sg` values should be less than or equal to the maximum number of initiations for the session initiator (that is, the value set by `/machine/jobmax/` *initiator* `/start_max`). See Section 8.10.2.1.3, page 419, for more information.

To permanently enable group scheduling control and make permanent changes to the pick values, insert the appropriate `rmgr` directives in the local configuration file by using the following menu in the configuration menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM control settings ->
        Full pathname of local URM config file
```

8.10.1.4 Load Smoothing Factors

Each resource that URM monitors has a specific *load* associated with it (stored in `/machine/load/` *resource*) that indicates the amount of the resource in use. URM tracks the load for the following resources:

- Memory
- Total active sessions
- SDS usage
- Tape usage

- MPP resource usage
- Individual session initiators: `batch`, `login`, `rsh`, `rexec`, `null`, `ftp`, and `cron`.
- Site-specific session initiators `site1`, `site2`, and `site3`, if defined (see Section 8.4.5, page 365, for more information)

Use the `rmgr` directive `view /machine/load` to display the current load values, as in the following example:

```
rmgr => view /machine/load

 LVrwr- 0 0 Sep 17 08:22 <memory > Int, 41905
 LVrwr- 0 0 Sep 17 08:22 <shared_txt > Int, 10660
 LVrwr- 0 0 Sep 17 08:22 <sesstab_mem > Int, 54222
 LVrwr- 0 0 Sep 17 08:22 <sds > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <tape > Int, 1
 LVrwr- 0 0 Sep 17 08:22 <bb > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <pe > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <site3 > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <site2 > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <site1 > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <rsh > Int, 1
 LVrwr- 0 0 Sep 17 08:22 <rexec > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <null > Int, 31
 LVrwr- 0 0 Sep 17 08:22 <login > Int, 35
 LVrwr- 0 0 Sep 17 08:22 <ftp > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <cron > Int, 1
 LVrwr- 0 0 Sep 17 08:22 <batch > Int, 1
```

Instead of using actual load values to estimate available system resources, URM applies a *smoothing function* to each resource load before using the load in its recommendation calculations. The smoothing function prevents overreaction to fluctuations in resource use and helps protect users from an erratic system response. (The load smoothing function is also called a *moving average* or *rolling average*.) The amount of smoothing is controlled by smoothing factors.

Each resource has its own smoothing factor, which can be set to a proportional value between 0.0 and 1.0, depending on how accurately you want it to represent changes in actual resource loads. The value 1.0 specifies that URM should match load values to actual changes; the value 0.0 specifies no change to existing load values. The closer a smoothing factor is to 1.0, the more quickly load changes will be adjusted. The closer to 0.0, the less effect actual resource usage has on job selection. For example, the smoothing factors for the session

initiators (batch, `ftp`, `login`, and so on) are set to 1.0 by default; this allows URM to enforce any session boundaries (`jobcount` target values).

Note: Smoothing occurs only when resource use drops. Increases in resource use are reflected immediately.

The graph in Figure 9 shows how this adjustment takes place. This example compares the results of four different smoothing factors for the memory load:

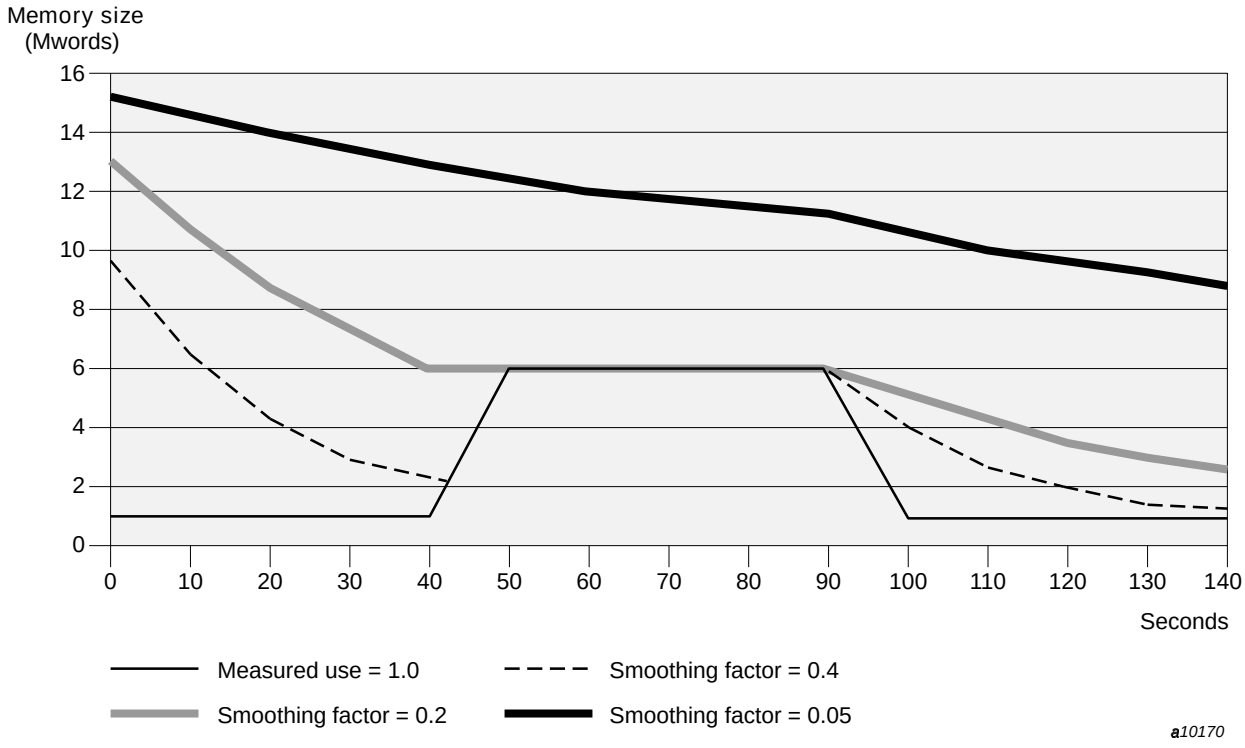

Figure 9. Example of different smoothing factors

Note: Smoothing factors of values at or near 0.0 is not recommended. Setting a smoothing factor below 0.1 would cause URM to cease updating the load values. Using a very low value for a smoothing factor can cause URM to react too slowly to changes in resource loads.

The graph in Figure 10 shows the effect of the default smoothing factor for memory (0.8) on URM calculations of memory load over time:

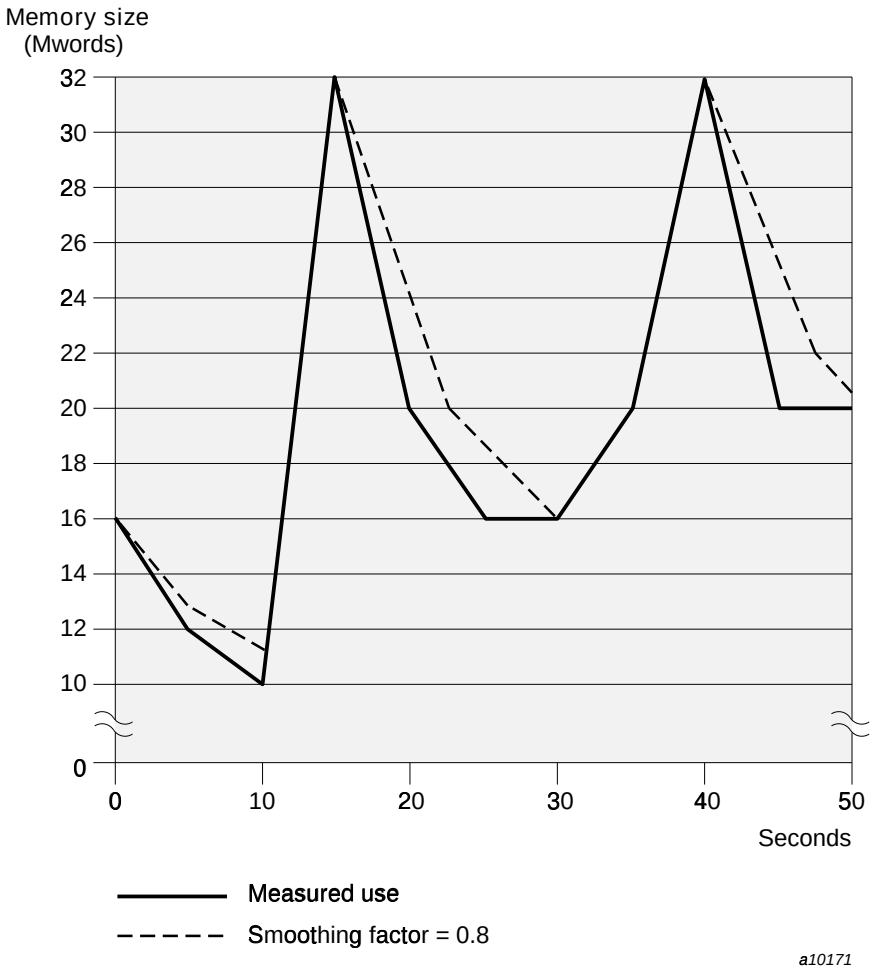

Figure 10. Default smoothing factor for memory load

Use the `rmgr` command to display the current smoothing factors, as in the following example:

```
rmgr =>view /machine/rate

    LVrwr-  0    0 Oct 25 23:59 <memory        > Float, 0.800000
    LVrwr-  0    0 Oct 25 23:59 <sds           > Float, 0.800000
    LVrwr-  0    0 Oct 25 23:59 <tape          > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <bb            > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <pe            > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <site3         > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <site2         > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <site1         > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <rsh           > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <rexec         > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <null          > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <login         > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <ftp           > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <cron          > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <batch         > Float, 1.000000
```

To change any of the load smoothing factors, enter the following `rmgr` directive:

`/machine/rate/`*resource* `=` *factor*

The value *resource* specifies the resource, such as `memory`, and *factor* specifies a value between 0.1 and 1.0.

To permanently change the configuration, use the following menu in the menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM machine load evaluation rates
```

8.10.2 Tuning URM Job Selection Criteria

In addition to controlling system resource targets, URM controls the job selection criteria for each session initiator. (The job selection criteria are called *individual session initiator targets* in the menu system.)

For the batch session initiator, URM monitors the session maximum targets (in `/machine/jobmax` values) and the resource loads (in `/machine/load` values). For nonbatch session initiators, URM monitors the defaults (in `/machine/default` values) and the resource loads (in `/machine/load` values) for each session initiator.

The following sections describe the job selection criteria used for batch jobs and for interactive jobs.

8.10.2.1 Batch Jobs

URM tracks the following job selection criteria for batch jobs:

- Job count; total number of active sessions allowed for the batch session initiator
- Maximum number of batch job initiations per scheduling cycle
- Memory request for a batch job
- CPU request for a batch job
- Tape request for a batch job
- SDS request for a batch job
- MPP requests for a batch job

In addition to the job selection criteria, URM checks the machine target values and batch job ranking before recommending a job for initiation. (Refer to Section 8.10.1.1, page 401, for more information about target values; see the following section for information on the batch job ranking calculation.)

Use the following `rmgr` directive to change any of the job selection criteria for the batch session initiator:

`/machine/jobmax/batch/`*resource* `=` *value*

To permanently change the configuration, access the following menu and select `batch` as the session initiator name:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM control settings ->
        Individual session initiator targets
```

The following sections discuss each of the batch job selection criteria.

8.10.2.1.1 Batch Job Ranking Calculation

URM assigns a rank to each job based on the job's *resource attributes*, or declared amount of resources requested by the job (see "Using URM with NQS," Section

8.4.13, page 372, for more information). During batch job ranking, each resource attribute is normalized (converted to a proportional value between 0.0 and 1.0) and multipled by a weighting factor. Each resource has an associated weighting factor that can be configured to a smaller or larger value based on the desired importance for that resource in the batch job ranking calculation.

By default, all weighting factors are equal (set to 0.5). Increasing the value of a weighting factor increases its relative importance in the ranking; setting a weighting factor to 0 effectively removes that resource from the calculation.

URM uses following equation for batch job ranking:

```
rank = (share * share_wt) +
       (usage * usage_wt) +
       (service_pri * service_wt) +
       (age * age_wt) +
       (cpu * cpu_wt) +
       (memory * mem_wt) +
       (tape * tape_wt) +
       (SDS * sds_wt) +
       (bb * bb_wt) +
       (PE * pe_wt) +
       (PE_time * petime_wt) +
       (prevrun_boost) +
       (min_rank)
```

The resources and associated weighting factors are represented in this equation as follows:

share * share_wt

Fair-share component. If the fair-share scheduler is not enabled, the value of this component is 0. If fair-share is enabled, *share* represents the normalized share value of the job owner's resource group or account; share_wt represents the fair-share weighting factor. URM evaluates the system share hierarchy at a rate determined by /urm/share_eval (default 900 seconds). This weighting factor can be adjusted by changing /urm/share_wt with the rmgr command.

usage * usage_wt

CPU usage component. The value *usage* represents the normalized usage maintained for the job's user in the shrusage field in the UDB; usage_wt represents the usage weighting

factor. If the user already has jobs active on the system, the current usage is obtained; otherwise, the decayed usage from the UDB is used. The usage weighting factor can be adjusted by changing `/urm/usage_wt` with the `rmgr` command.

service_pri * `service_wt`

NQS queue priority component. The value *service_pri* represents the interqueue priority from NQS; `service_wt` represents the service weighting factor. If `service_wt` is set to 0, the NQS interqueue priorities no longer take effect in the URM priority calculation. This weighting factor can be adjusted by changing `/urm/service_wt` with the `rmgr` command.

age * `age_wt`

Age-in-queue component. The value *age* represents the length of time the job has been waiting for initiation from the URM queue; `age_wt` represents the age weighting factor. The age of the oldest job is 1.0, and the age of the youngest job is 0.0. The age weighting factor can be adjusted by changing `/urm/age_wt` with the `rmgr` command.

cpu * `cpu_wt`

Requested CPU resource component. The value *cpu* represents the normalized CPU requirement specified with each batch job; `cpu_wt` represents the CPU weighting factor. The `cpu_wt` value can be adjusted by changing `/urm/cpu_wt` with the `rmgr` command.

memory * `mem_wt`

Requested memory resource component. The value *memory* is the normalized memory requirement specified with each batch job; the largest job is 0.0, and the smallest job is 1.0. The value `mem_wt` represents the memory weighting factor. This weighting factor can be set by changing `/urm/mem_wt` with the `rmgr` command.

tape * `tape_wt`

Requested tape resource component. The value *tape* is the normalized tape device requirement specified with each batch job; *tape* is set to 0.0 for the job requesting the most tape resources, and to 1.0 for the job requesting the least resources.

The value `tape_wt` represents the tape weighting factor. This weighting factor can be set by changing `/urm/tape_wt` with the `rmgr` command.

SDS * `sds_wt`

Requested SDS resource component. The value *SDS* represents the normalized SDS requirement specified with each batch job; *SDS* is set to 0.0 for the job requesting the most SDS space, and to 1.0 for the job requesting the least resources. The value `sds_wt` represents the SDS weighting factor. This weighting factor can be set by changing `/urm/sds_wt` with the `rmgr` command.

BB * `bb_wt`

PE * `pe_wt`

PE_time * `petime_wt`

These components define the MPP resource attributes for MPP barriers, processor elements, and requested PE time job attributes, respectively.

prevrun_boost

Value to improve the rank of a previously checkpointed job.

min_rank

Minimum rank (NQS job priority).

To make a permanent configuration change to the weighting factors, use the following menu in the menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Weighting factors for the selector
```

8.10.2.1.2 Batch Job Count

The job count, or maximum number of active batch sessions, is controlled by `/machine/jobmax/batch/jobcount`. By default, this value is set to the size of the kernel session table (set by the `NSESS` parameter) during URM's

automatic configuration process. Changing this value affects the relative proportions of batch and interactive jobs on the system.

8.10.2.1.3 Batch Maximum Initiation

The maximum number of batch initiations allowed during a single scheduling cycle is controlled by the `/machine/jobmax/batch/start_max` value. When this limit is reached, URM does not recommend any more batch jobs until its next scheduling cycle. The default for this value is 10.

If group scheduling control is enabled, the sum of the `pick_in_sg` values should be less than or equal to this value; see Section 8.10.1.3, page 408, for more information.

8.10.2.1.4 Batch Memory Request Target

The memory request target for batch jobs is controlled by `/machine/jobmax/batch/memory`. This value determines the maximum size, in clicks, of a batch job. Jobs requesting more memory than this value will not be recommended.

By default, this value is configured automatically to the maximum amount of user memory available. Consider increasing this value if your site runs batch jobs that have two or more processes whose total memory requirements exceed the amount of available user memory.

8.10.2.1.5 Batch CPU Request Target

The CPU request target is controlled by `/machine/jobmax/batch/cputime`. This value determines the maximum amount of CPU time, in seconds, for a batch job. Jobs requesting more CPU usage than this value are not recommended for initiation. By default, this value is set to 9999999 seconds.

8.10.2.1.6 Batch Tape Request Target

The batch tape request target is controlled by `/machine/jobmax/batch/tape`. This value specifies the maximum number of tape devices allowed for a batch job. Jobs requesting more devices than this value will not be recommended. This value is set to the number of online tape devices during URM's automatic configuration process.

8.10.2.1.7 Batch SDS Request Target

The SDS request target is controlled by `/machine/jobmax/batch/sds`. This value specifies the maximum amount of SDS space, in clicks, that a batch job can use. Jobs requesting a larger SDS usage than this value are not recommended for initiation. This value is set to the amount of available SDS space during URM's automatic configuration process. Consider increasing this value if your site runs jobs whose total SDS requirements exceed the amount of available space.

8.10.2.2 Interactive Jobs

Interactive jobs (jobs initiated from all session initiators except `batch`) are evaluated for initiation by URM immediately. Unlike batch jobs, no backlog queue is maintained by URM. In most cases, if URM does not recommend the job for initiation, the session initiator will terminate it. This is the case for jobs submitted by `login`.

In order for URM to recommend an interactive job for initiation, the requested resources for the job must not exceed the following target and maximum values:

- Active job target for the system (`/machine/target/jobcount`). For more information, refer to Section 8.10.1.1.3, page 405.
- Interactive job count; maximum number of active sessions for the specified interactive session initiator.
- Memory oversubscription target for the system.

8.10.2.2.1 Interactive Job Count

The job count maximum for each interactive session initiator is controlled by `/machine/jobmax/` *initiator* `/jobcount` The default values for each interactive session initiator are determined during the automatic configuration process for URM. For example, the maximum job count for `login` sessions is set to the size of the kernel session table by default. Use the `rmgr` command to change this value, as in the following example:

```
/machine/jobmax/login/jobcount = 100
```

To permanently change the configuration, access the following menu and select the desired session initiator name (for example, `login`):

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
```

```
URM configuration ->
  URM control settings ->
    Individual session initiator targets
```

8.10.2.2.2 Interactive Memory and CPU Defaults

For interactive jobs, URM does not monitor session initiator values, as is done for the `batch` service initiator, because the interactive service initiators do not supply resource requirement information to URM. Instead, URM uses the default values for memory and CPU use in `/machine/default/` *initiator* `/memory` and `/machine/default/` *initiator* `/cputime`. For example, to determine if a login session will exceed the memory oversubscription target, URM compares the memory default for the job (in `/machine/default/login/memory`) to the current system memory load.

By default, these values are set to 0; this specifies that URM should recommend all interactive jobs for initiation. To enable URM control of the number of active interactive sessions for a specific initiator, both the `memory` and `cputime` defaults must be set to a nonzero value. For example, use the following `rmgr` directives to enable URM control of login sessions:

```
/machine/default/login/memory = 200
/machine/default/login/cputime = 10
```

The value 200 specifies the average size of memory, in clicks, allowed for login sessions. The value 10 specifies the average amount of CPU time in seconds. URM does not consider the `cputime` value when recommending interactive sessions, but both `memory` and `cputime` must be set to a nonzero value to enable URM to control jobs from a session initiator.

> **Note:** The session initiator defaults are used only for URM job initiation recommendations. They do not affect the actual memory and CPU limits of the executing session.