HP-UX Workload Manager User's Guide

Version A.03.00

Manufacturing Part Number: B8844-90008

May 2005

Publication history

Seventh edition
May 2005
B8844-90008
HP-UX 11i v1 and HP-UX 11i v2

Sixth edition
March 2004
B8844-90006
HP-UX 11.0, HP-UX 11i v1, and HP-UX 11i v2

Fifth edition
June 2003
B8844-90005
HP-UX 11.0, HP-UX 11i v1, and HP-UX 11i v2

Fourth edition
June 2002
B8844-90003
HP-UX 11.0 and HP-UX 11i v1

Third edition
June 2001
B8844-90002
HP-UX 11.0 and HP-UX 11i v1

Second edition
December 2000
B8843-90004
HP-UX 11.0

First edition
April 2000
B8844-90001
HP-UX 11.0

Notice

Contents

Contents

Contents

Contents

Contents

Contents

Tables

Tables

Figures

Figures

Preface

This book describes the Version A.03.00 release of HP-UX Workload Manager (HP-UX WLM).

The intended audience for this manual is system administrators.

System platform

HP-UX WLM runs under the HP-UX 11i v1 (B.11.11) operating systems on HP 9000 servers. Also, it runs under HP-UX 11i v2 (B.11.23) on HP 9000 servers and HP Integrity servers.

Associated software

Installing HP-UX Workload Manager ensures that the following software is also installed:

- Process Resource Manager (PRM) C.03.00

  HP-UX WLM manages workloads by using PRM controls.

- HP-UX WLM Toolkits (WLMTK) A.01.08 or later

  WLMTK facilitates integrating HP-UX WLM with various third-party products.

If the system already has the correct versions installed, they will not be modified.

HP-UX WLM provides a method for collecting system and application data from GlancePlus. You can use WLM with any HP-UX 11i version of GlancePlus. On HP-UX 11i v1, be sure to install either the 11i Enterprise Operating Environment or 11i Mission-critical Operating Environment to ensure you have a GlancePlus version that is fully compatible with WLM.

New in this edition

HP-UX WLM A.03.00 provides the following new or changed functionality:

- Integration with Security Containment

  The HP-UX feature Security Containment provides file and process isolation. WLM's integration with this feature allows you to create Secure Resource Partitions, which combine the isolation of Security Containment and the resource allocation controls of WLM.

  For more information, see the `scomp` keyword in the wlmconf(4) man page.

- Secure communications

  WLM's communications can now be secured. Use the new `wlmcert` command to manage security certificates, then use the `-s` option with `wlmd`, `wlmpard`, and `wlmcomd` as needed. You can start the daemons in secure mode at boot time by editing the file /etc/rc.config.d/wlm. You can also secure communications in `wlmgui` via the Preferences menu.

  For more information, see the wlmcert(1M), wlmd(1M), wlmpard(1M), and wlmcomd(1M) man pages and the /etc/rc.config.d/wlm file.

- A WLM configuration is now host-based unless it includes a `prm` structure

  WLM now assumes a configuration is intended for migrating CPUs between virtual partitions or nPartitions unless the configuration includes a `prm` structure.

- New keywords for configurations without `prm` structures

  The `wlmd` configuration file now supports the keywords `hmincpu`, `hmaxcpu`, and `hweight`.

  Use these keywords when managing virtual partitions or nPartitions to specify minimum and maximum CPU shares for a host as well as a CPU weight for the host. (These keywords cannot be used in configuration files that include a `prm` structure.)

  For more information, see the wlmconf(4) man page.

- The `vpar` and `npar_icod` structures in a `wlmpard` configuration are silently converted to `par` structures

- `wlmpard` is no longer required to be running in a partition on the managed complex

  Previously, if WLM was managing the partitions in a complex, `wlmpard` had to be running in one of those partitions. Now, `wlmpard` simply has to be running on a supported platform with network connectivity to the managed partitions.

- Terminology change: The Instant Capacity on Demand (iCOD) product is now known as Instant Capacity, or iCAP. Given WLM's integration with this product, Instant Capacity is mentioned in a number of places in the WLM product and documentation.

- Configuration file for `wlmpard` now has a `utilitypri` keyword

  The `utilitypri` keyword allows you to specify the lowest priority at which WLM is allowed to use Temporary Instant Capacity or Pay Per Use resources to meet service-level objectives for your workloads.

  The `utilitypri` keyword allows WLM—when Temporary Instant Capacity is available—to adjust the CPU total to meet demand.

  Using this keyword also ensures WLM maintains compliance with your license for Temporary Instant Capacity: When the license is exhausted, WLM will no longer attempt to use the temporary resources.

  For more information, see the wlmparconf(4) man page.

- Configuration file for `wlmpard` no longer supports the `partitions` or `weight` keywords

  If you use either the `partitions` or `weight` keywords in your `wlmpard` configuration file, WLM issues warnings and ignores the keywords' values.

  You can still assign weights to workload groups in the `wlmd` configuration file. You can also assign a weight on a host as a whole using the new `hweight` keyword in the `wlmd` configuration.

  For more information, see the wlmparconf(4) man page.

- The WLM GUI now has a Reconnect button that allows you to re-establish a connection—without re-entering all the data—to a system you activated a configuration on. After reconnecting, you can monitor your workloads.

- The recommended value for `wlm_interval` is 5 when using usage goals

  Typically, you should set `wlm_interval` to 5 when your configuration includes a usage goal.

- `cntl_base_previous_req` now set to 1 by default

  In previous releases, the default value for `cntl_base_previous_req` was 0. The default value is now 1.

  For more information, see the wlmconf(4) man page.

- `wlminfo` changes

  — New option: `-q`

  Quiets (suppresses) display of messages from /var/opt/wlm/msglog in the `wlminfo` output.

  — New option to `group` command: `-S`

  Shows data for the reserved workload group `PRM_SYS` in the output.

  — New command: `proc`

  Displays data about the most active processes.

  — `Hostname` field in output has been extended

  In output of `wlminfo host` and `wlminfo par`, the `Hostname` field has been extended from 16 characters to 24.

  — Field size issues in output have been addressed

  Values for the `Req` and `Shares` columns can no longer extend beyond their fields, disrupting data in surrounding fields.

  — `wlminfo group` produces output only when the configuration data includes workload groups

  There is no output when the configuration data is only for hosts.

  — Several fields now show a '-'

  In the Group field of `wlminfo slo` output and the Workload Group field of the `wlminfo group` output, you will see a '-' character if the configuration data is for a host (rather than a workload group).

— New field in `wlminfo host` output

A field named CPUs Used has been added to display the CPU utilization for the host.

- `mincpu` / `maxcpu` are now optional in all cases

  You are no longer required to specify `mincpu` / `maxcpu` statements when creating certain types of `slo` structures. For SLOs without a `mincpu`, `goal`, or `cpushares` statement, WLM requests 0 CPU shares for the associated workload group—unless a `gmincpu` or `hmincpu` value is being used.

- New tunable

  The tunable `cntl_avg` allows you to control averaging in usage controllers. By default, averaging is used.

  For more information, see the wlmconf(4) man page.

- New behavior when the sum of the `gmincpu` values exceed the total CPU

  Previously, when this sum exceeded the total CPU, the values were scaled down proportionately so that the sum equals the total CPU. Now, those values are treated as CPU requests to be met before any other requests are considered. Any `weight` values assigned to the workload groups apply.

- Pay Per Use Toolkit being deprecated

  The toolkit, including its `utilitydc` command, is being deprecated. Support for the command will be removed in a future release. In the meantime, it is still documented in the utilitydc(1M) man page.

  Please begin upgrading to the simpler and more robust solution provided by `wlmpard` and the new keyword `utilitypri`. For information on the new solution and keyword, see the wlmpard(1M) and wlmparconf(4) man pages.

- Deprecated memory record syntax no longer supported

  The following syntax was deprecated in WLM A.02.00:

  `mem =` *group* `:` *ent* `[`*cap*`] [`*sup*`] [`*method*`] [, ...];`

  As of this release, it is no longer supported.

  If you would like information on how to duplicate the behavior provided, email wlmfeedback@rsn.hp.com.

- Deprecated syntax for PSET-based groups no longer supported

  The `numcpus` and `cpus` portions of the following syntax were deprecated in WLM A.02.01:

  `groups =` *group* `: PSET [ { [numcpus =` *integer*`] [cpus =` *ID1*`,` *ID2*`, ...] } ];`

  As of this release, those portions are no longer supported.

  If you would like information on how to duplicate the behavior provided, email wlmfeedback@rsn.hp.com.

- Deprecated usage threshold tunables have been removed

  The following tunables were deprecated in WLM A.02.00:

  — `usage_goal_dflt_low`

  — `usage_goal_dflt_high`

  As of this release, these tunables are no longer supported.

- A `wlmd` configuration is rejected if `absolute_cpu_units` is set explicitly to 0 when PSETs or the `primary_host` keyword are present
- Multiple `prm` structures are no longer allowed

Feedback to the HP-UX WLM team

If you would like to comment on the current HP-UX WLM functionality or make suggestions for future releases, please send email to:

wlmfeedback@rsn.hp.com

For a forum with other HP-UX WLM users, visit the IT Resource Center's forum for HP-UX Workload/Resource Management:

http://forums.itrc.hp.com/cm/

Support and patch policies

The http://www.hp.com/go/wlm site provides information on WLM's support policy and patch policy. These policies indicate the time periods for which this version of WLM is supported and patched.

Training

HP offers a course in HP-UX resource management using WLM. For information, including a course outline, visit:

http://www.hp.com/education/courses/u5447s.html

Notational conventions

This section describes notational conventions used in this book.

`bold monospace`	In command examples, **`bold monospace`** identifies input that must be typed exactly as shown.
`monospace`	In paragraph text, `monospace` identifies command names, system calls, and data structures and types. In command examples, `monospace` identifies command output, including error messages.
italic	In paragraph text, *italic* identifies titles of documents.
`italic`	In command syntax diagrams, *`italic`* identifies variables that you must provide.
Brackets ([ ])	In command examples, square brackets designate optional entries. The following command example uses brackets to indicate that the variable *`output_file`* is optional: `command` *`input_file`* [*`output_file`*]
Curly brackets ({}), Pipe (|)	In command syntax diagrams, text surrounded by curly brackets indicates a choice. The choices available are shown inside the curly brackets, separated by the pipe sign (|). The following command example indicates that you can enter either `a` or `b`: `command {a | b}`

Horizontal ellipses (...)	In command examples, horizontal ellipses show repetition of the preceding items.
NOTE:	A note highlights important supplemental information.

Associated documents

Associated documents include:

- *HP-UX Workload Manager A.03.00 Release Notes for HP-UX 11i v1 and HP-UX 11i v2* (/opt/wlm/share/doc/)
- Workload Management home page
  http://www.hp.com/go/wlm (white papers and case studies)
- wlm(5) man page

  See this man page for a list of all the other WLM man pages
- wlmoradc(1M) man page
- wlmtk(5) man page
- ARM Working Group Home Page
  (http://www.cmg.org/regions/cmgarmw)
- *Using the Event Monitoring Service* manual
- *Process Resource Manager User's Guide* manual
  (available at /opt/prm/doc/)
- *HP-UX Workload Manager Toolkits User's Guide* manual (available at /opt/wlm/toolkits/doc/WLMTKug.pdf)
- *Managing MC / ServiceGuard* manual
- *GlancePlus User's Guide* manual (available through the Help in the `gpm` interface to GlancePlus)
- *Managing Systems and Workgroups: A Guide for HP-UX System Administrators* manual

The manuals listed above, along with many other Hewlett-Packard manuals, are available on the web at http://docs.hp.com.

1 Introduction

This chapter introduces the basic concepts of performance management, workload management and HP-UX Workload Manager (WLM). It covers:

- Performance management overview
- What is workload management?
- What is HP-UX Workload Manager?
- Why use Workload Manager?
- WLM and partitions
- Solutions WLM provides
- What is the ideal environment for WLM?
- Using WLM on multiple servers
- WLM and PRM
- WLM product directory structure

WLM runs on any HP 9000 server with HP-UX 11i v1 (B.11.11). It also runs on HP 9000 servers and HP Integrity servers with HP-UX 11i v2 (B.11.23).

Performance management overview

Performance management is necessary to keep users satisfied and ensure that business-critical applications and transactions have the resources they need. The management of a system's performance requires:

- Monitoring the system to understand how the current combination of system resources, applications, and users affects performance
- Controlling the performance by managing the system's resources, applications, and number of users

The methods to monitor and control performance can vary greatly.

Table 1-1 explores advantages and disadvantages of several monitoring methods.

Table 1-1 Performance monitoring methods

Method	Advantages	Disadvantages
Wait for users' complaints about performance	• User-focused • System configuration is kept simple to reduce number of complaints and speed resolution	• Re-active (not pro-active) • Inexact • Requires one application/server model to keep number of calls low
Predict performance by monitoring an application's resource usage	• Detects faulty applications that are over-consuming resources • Usage trends can be used to predict future loads	• End-to-end performance is not always tied to resource consumption • Few options if problems are detected (separate applications; buy more servers)

Table 1-1 Performance monitoring methods (Continued)

Method	Advantages	Disadvantages
Directly monitor through metrics (response time, throughput, ...) obtained from the application	• Exact performance metrics • Can be used pro-actively to guarantee user's satisfaction with performance • Performance management is designed into the application	• Limited number of applications come instrumented for tracking performance • Application source code may not be available for instrumentation • Instrumenting source code can be time-intensive

After determining how to monitor performance, you must decide how to control performance. Table 1-2 examines the advantages and disadvantages of various control methods.

Table 1-2 Performance controlling methods

Method	Advantages	Disadvantages
One workload per server	• Workloads do not compete for resources	• Server resources sit idle a large percentage of the time
Multiple workloads: No resource management	• Increased resource usage • Fewer servers needed	• Workloads compete for resources • A single workload may take over the server, preventing other workloads from getting resources

Table 1-2 Performance controlling methods (Continued)

Method	Advantages	Disadvantages
Multiple workloads: Fixed resource allocation Example: Process Resource Manager (PRM) with capping enabled	• Workloads' resource use cannot exceed resource allocations • A workload cannot take over a server to the detriment of other workloads • Consistent performance when work stays constant in a workload • Excess resources can easily be tracked with PRM and GlancePlus tools	• Performance may degrade when workloads cannot exceed resource allocations that are set too low • Unused capacity cannot be shared optimally if resource usage is capped
Multiple workloads: Variable resource allocations based on usage Example: PRM without capping enabled	• Workloads are guaranteed resource minimums, but can borrow unused resources from other workloads • Increased level of work is handled automatically • Excess resources can easily be tracked with PRM and GlancePlus tools	• Workloads get resources, but performance can still be less than desired • Over-performance followed by typical performance can cause complaints

Table 1-2 Performance controlling methods (Continued)

Method	Advantages	Disadvantages
Multiple workloads: Variable resource allocations based on actual, reported performance Example: HP-UX WLM	• Consistent performance levels are maintained automatically • Workloads can be prioritized • Excess resources can easily be tracked with PRM and GlancePlus tools	• Getting performance metrics for workloads can be difficult; however, WLM has simplified this task with a built-in data collector

The remainder of this book focuses on the last row of Table 1-2: Automatically managing multiple, prioritized workloads on a single server—possibly across partitions—based on reported performance levels. This strategy is commonly referred to as goal-based workload management.

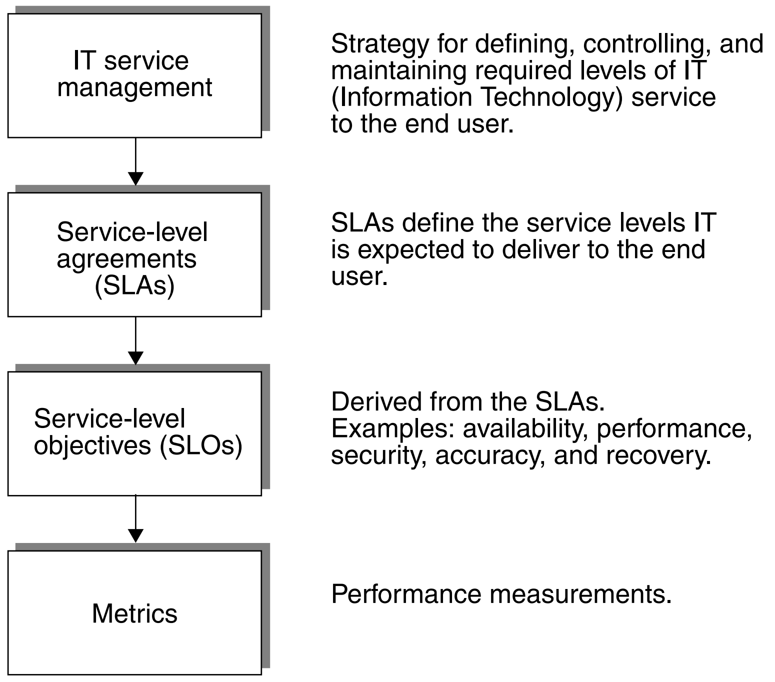
WLM implements this type of workload management allowing you to:

- Optimize your return on investment in servers
- Maintain multiple applications on a single server without increasing performance problems
- Improve your ability to forecast capacity needs

What is workload management?

System management has typically focused on monitoring the availability of systems. While system availability is certainly important, it neglects the importance of the applications themselves. However, even application availability is not the final objective. To best serve end users, an application must be available and provide its services in a timely manner, with expected and consistent levels of performance. For example, if a task is generally expected to complete in less than 2 seconds, but takes 15, the service to the end user is not available. Thus, system and application availability do not guarantee the availability of a service.

Workload management ensures a service's availability through service-level management. This type of management is based on the following components:


IT service management
Strategy for defining, controlling, and maintaining required levels of IT (Information Technology) service to the end user.
Service-level agreements (SLAs)
SLAs define the service levels IT is expected to deliver to the end user.
Service-level objectives (SLOs)
Derived from the SLAs. Examples: availability, performance, security, accuracy, and recovery.
Metrics
Performance measurements.

NOTE WLM manages performance SLOs.

Now to explore these components in more detail:

- IT service management

  A strategy driven by business requirements to provide, and in many cases guarantee, levels of service from IT (Information Technology) to end users. The needs of the end users are the primary driver for the development of the IT infrastructure. The benefits of defining this strategy are higher return on investment in IT expenditures and also the proper setting of expectations for all organizations involved.

- Service-level agreements

  Documents defining various service levels IT is expected to deliver to the end user. These documents focus on individual applications and the service levels required by the application users. SLAs can also be used for budgeting, planning, and control purposes.

  The following steps outline how to define a service-level agreement:

Step 1. Establish an inventory of system resources (people, CPUs, disk space, and so forth) that serve the end users.

Step 2. Determine how much of the inventory of system resources is currently being consumed to support the present set of applications.

Step 3. Determine what the end users require to maintain the status quo if they are already receiving acceptable service.

Step 4. Determine what the end users require to improve their current service if they are not receiving acceptable service.

- Service-level objectives

  Objectives, derived from SLAs, explicitly describe the expected availability, performance, security, accuracy, or recovery. They can also describe expected response time, job duration, or throughput.

  SLOs lay the foundation for developing the actual metrics that IT management must collect, monitor, and report on to determine whether the agreed-upon service levels are being met.

- Metrics

  This data is used to determine whether an SLO is being met.

What is HP-UX Workload Manager?

HP-UX Workload Manager provides automatic resource allocation and application performance management through the use of prioritized service-level objectives (SLOs).

WLM manages workload groups you define in the WLM configuration file. You assign applications and users to workload groups. You can also assign secure compartments, which you create with the HP-UX feature Security Containment, to workload groups. WLM then manages each workload group's CPU, real memory, and disk bandwidth resources according to its SLOs in the current configuration. If multiple users or applications within a workload group are competing for resources, standard HP-UX resource management determines the resource allocation within the group.

WLM automatically allocates CPU resources in order to achieve the workload groups' desired SLOs. These CPU resources can be allocated either in shares (portions) of multiple CPUs or in whole CPUs when using WLM's partition management or processor set (PSET) management. With real memory, WLM allows you to specify lower and upper limits on the amount of memory a workload group receives. Disk bandwidth shares can be statically assigned in the configuration file.

NOTE The WLM daemon is designed to use minimal system's resources. All of the daemon's processes run in the `PRM_SYS` (ID 0) workload group, as explained in the section "Reserved workload groups" on page 151.

WLM can manage the following resources for your workload groups:

CPU

Arbitrates CPU requests to ensure high-priority SLOs meet their objectives. SLOs make CPU requests for workload groups. CPU can be allocated in:

- Time slices on several CPUs
- Whole CPUs used by PSET-based workload groups
- Whole CPUs used by vPar-based workload groups
- Whole CPUs used by nPar-based workload groups (with each nPar using Instant Capacity software)

A workload group may not achieve its CPU request if CPU is oversubscribed and the workload group's SLOs are low priority.

Disk bandwidth

Ensures that each workload group is granted at least its share of disk bandwidth.

Memory

Ensures that each workload group is granted at least its minimum, but (optionally) no more than its capped amount of real memory.

In addition, WLM has an application manager that ensures specified:

- applications and their child processes,
- user processes, and
- secure compartment processes

run in the appropriate workload groups.

Why use Workload Manager?

WLM allows you to combine multiple applications on a single system—potentially using partitions, while ensuring that each application runs at the performance level required to meet your business goals. You create workload groups, place your applications in the workload groups, then define the necessary performance levels for each workload group. WLM helps the workload groups meet the desired performance levels by dynamically allocating resources based on workload performance, varying workloads, and changing system conditions. Thus, WLM ensures that performance levels are acceptable and business goals are achieved.

In general, WLM provides the ability to:

- Prioritize workloads on a single system, adjusting the CPU resources for their associated workload groups based on goals
- Manage by service-level objectives within and across virtual partitions and nPartitions
- Adjust resource allocations by automatically enabling or disabling SLOs based on time of day, system events, or application metrics
- Enable SLOs associated with a Serviceguard package failover
- Ensure the workload groups of critical workloads have sufficient resources to perform at desired levels
- Set minimum and maximum amounts of CPU available to a workload group's applications
- Set minimum and maximum amounts of memory available to a workload group's applications
- Grant a workload group dedicated memory resources and CPU resources (in the form of a processor set)
- Adjust the number of CPUs in a partition or a processor set to meet SLOs
- Grant a workload group CPU shares in direct proportion to a metric, such as number of processes in the workload group

- Set and manage user expectations for performance
- Run multiple workloads on a single system and maintain performance of each workload
- Monitor resource consumption by applications or users through HP GlancePlus, WLM utilities, or PRM utilities

Service-level objectives (SLOs)

A key reason for using WLM is its ability to manage service-level objectives. After defining a workload group, the system administrator can specify one or more SLOs for each workload group. WLM allocates CPU to workload groups based on whether the group's application is underperforming, meeting, or overperforming its SLOs.

SLOs can be shares-based or goal-based. With a shares-based SLO, WLM simply tries to grant the associated workload group a certain amount of the CPU by allocating it CPU shares. (A CPU share is 1/100 of a single CPU or 1/100 of each CPU on the system, depending on WLM's mode of operation.) With goal-based SLOs, WLM actively changes the associated workload group's CPU allocation to best meet the SLO. These SLOs are based on one of two goal types:

- Metric goals

  Goals based on a metric, such as processing at least *x* transactions per minute or a response time under *y* seconds.

- Usage Goals

  Goals based on utilization of allocated CPU for processes in workload groups. If a workload group's processes are not using a certain amount of the group's allocation, the allocation is decreased; similarly, if the processes are using a high percentage of the group's allocation, the allocation is increased.

Prioritized SLOs

Another great reason for using WLM is that it allows the system administrator to prioritize the SLOs. Valid priorities start at 1, with 1 being the highest priority.

SLO priorities do not have to be uniquely assigned—multiple SLOs can be granted the same priority, allowing more than one workload group's objective to be top priority. This can be beneficial when the workloads in multiple workload groups are equally important. Typically, all the SLOs in a given configuration should not be assigned the same priority; otherwise, under heavy system load, WLM may not be able to allocate CPU optimally when there is not enough to meet all SLOs.

A single workload group can have multiple SLOs, each with a different priority. One SLO would be the high priority, "must meet" goal and the remaining SLOs would be lower priority, "meet if possible" goals. For example, a priority 1 goal might be for the workload group's application to keep response time under three seconds. The priority 2 goal might then be to keep response time under one second. This lower priority goal might be met, but only after the priority 1 SLOs of its associated workload group and all other workload groups are met.

For information on how WLM uses priorities, see "Allocating CPU: The rising tide model" on page 112.

WLM and partitions

The HP Partitioning Continuum offers three layers of partitioning. The layers are:

- Hard partitions

  These partitions are through hardware. One such partition is a complete server, which can be clustered in a HyperPlex configuration. The other type, called nPartition, is available on Superdome and Keystone servers, which can support up to 16 nPartitions on one server node with cell-based hardware isolation and complete software isolation (each node runs its own copy of HP-UX). The isolation guarantees that an application running in one nPartition is not affected by an application in another nPartition.

- Virtual partitions

  HP's virtual partitions, which are implemented in software, offer a unique granularity of partitioning for servers. You can create virtual partitions consisting of one or more CPUs. Each virtual partition runs its own instance of the HP-UX operating system. Complete software isolation is provided between virtual partitions. In addition, virtual partitions provide the ability to dynamically adjust the partition size by the dynamic addition and deletion of CPUs. Virtual partitions can be used within hard partitions.

- Resource partitions

  PRM combined with processor sets provides resource partitions. These partitions run within a single instance of HP-UX. PRM can be used within, but not across, hard partitions and virtual partitions.

WLM can be used both within and across hard partitions and virtual partitions.

For information on using WLM across partitions, see Chapter 9, “Managing SLOs across partitions,” on page 283.

Solutions WLM provides

The following sections illustrate how WLM provides various business solutions. For information on the configuration file syntax needed to produce these solutions, see “Configuring WLM” on page 127.

Reserving a portion of the server all of the time

In this first example, the SLO requests a fixed allocation for the workload contained in the group Marketing, reserving a portion of the server’s CPU resources. This SLO is priority 1 and is in effect at all times.

Group	Marketing
Priority	1
CPU shares	30 shares (30% of one CPU) (Absolute CPU units enabled)

Reserving a portion of the server at specified times

The next SLO also requests a fixed allocation, reserving 600 CPU shares in this case. However, the associated workload group contains a payroll application that only runs twice a month. Consequently, the SLO is enabled only twice a month, on the 6th and 21st.

Group	Payroll
Priority	1
CPU shares	600 shares (6 CPUs) (Absolute CPU units enabled)
Condition	6th or 21st

Reserving a portion of the server based on an event

The following example shows another SLO that is enabled only some of the time. Rather than a date or time though, the condition is a metric. The SLO is enabled only when a system accounting program is running. Once the SLO is active, it works to ensure the accounting program completes quickly by reserving 60 CPU shares for the associated workload group. When the program is completed, the SLO is disabled. At that point, the SysAcct group gets 1% of the total CPU, as explained in the section "Specifying when the SLO is active (optional)" on page 192.

Group	SysAcct
Priority	1
CPU shares	60 shares (60% of one CPU) (Absolute CPU units enabled)
Condition	System accounting program is running

Ensuring resources in a Serviceguard failover

Similarly, the next SLO is enabled based on a metric. This example illustrates an SLO that is only active if the Serviceguard package pkgA is active on the current server. WLM provides a utility that generates a metric indicating whether a package is active. When the metric has value 1, the package is active, thus enabling the SLO. The SLO then causes WLM to attempt to allocate 250 CPU shares to the associated workload group.

Group	Failover_pkgA
Priority	1
CPU shares	250 shares (2.5 CPUs) (Absolute CPU units enabled)
Condition	pkgA is active on the current server

Allocating CPU as used

Now consider an SLO with a usage goal. Usage goals do not require a metric value; WLM tracks the metric itself. In this example, WLM adjusts the group’s CPU allocation based on the CPU utilization of applications executing in the Orders group. If the utilization is low (due perhaps to fewer of the group’s applications running), then the CPU allocation for the Orders group is reduced, making more resources available to other groups. If utilization is high, the group receives a larger CPU allocation. Regardless, the allocation should be at least 200 shares but no more than 800 shares.

Group	Orders
Priority	1
Goal	Match CPU allocation to consumption
Min CPU	200 shares (2 CPUs) (Absolute CPU units enabled)
Max CPU	800 shares (8 CPUs)

Allocating CPU per metric

WLM allows you to grant a workload group a certain amount of CPU per unit of a metric. In our example below, the workload group gets 5 CPU shares for each active process in the workload group. (This “number of active processes” metric is easily retrieved through WLM’s interface to GlancePlus.) However, the group’s CPU shares request is not allowed to fall below 10 shares or exceed 90 shares. 5 active processes would result in 25 CPU shares.

Group	webserver
Priority	1
Goal	5 CPU shares for each active process in the workload group
Min CPU	10 shares (10% of one CPU) (Absolute CPU units enabled)
Max CPU	90 shares (90% of one CPU) (Absolute CPU units enabled)

Controlling how workloads share and borrow excess cycles

Here, the workload group Development has multiple SLOs, where each SLO has a usage goal. The owner of the group funded 30% of the server. Consequently, the group's owner expects to get 30% of the server when needed. (Turning off "absolute CPU units," 30% of the server is 30 CPU shares.) However, when the workload is not busy, some of the excess resources are available for sharing, as long as the Development group always gets at least 15 shares. This is represented by the following SLO:

Group	Development
Priority	1
Goal	Match CPU allocation to consumption
Min CPU	15 shares (15% of total CPU with absolute CPU units disabled, which is the default)
Max CPU	30 shares (30% of total CPU with absolute CPU units disabled, which is the default)

When the workload becomes very busy, the Development group would like to borrow from other groups that may have excess resources. Based on usage patterns, the system administrator has agreed to let the group borrow up to an additional 30 shares if available. This allows the Development group to access up to 60 CPU shares. The associated SLO is shown below:

Group	Development
Priority	2
Goal	Match CPU allocation to consumption
Min CPU	30 shares (30% of total CPU with absolute CPU units disabled, which is the default)
Max CPU	60 shares (60% of total CPU with absolute CPU units disabled, which is the default)

Because this SLO is priority 2, it is met only after all priority 1 SLOs have been met.

Meeting peak demand

Now consider an SLO with a usage goal. The SLO’s associated workload group receives more, or less, CPU depending on how much it is actually using.

Group	Orders
Priority	1
Goal	Match CPU allocation to consumption
Min CPU	20 shares (20% of one CPU) (Absolute CPU units enabled)
Max CPU	80 shares (80% of one CPU) (Absolute CPU units enabled)

In this next example, the SLO has a metric goal instead of a usage goal. The SLO becomes active, based on the metric number_of_active_jobs, once the server is sufficiently busy. It attempts to keep the average job completion time under ten seconds.

Group	Jobs
Priority	1
Goal	Average completion time < 10 seconds
Min CPU	40 shares (40% of one CPU) (Absolute CPU units enabled)
Max CPU	90 shares (90% of one CPU) (Absolute CPU units enabled)
Condition	number_of_active_jobs > 100

Automatically resizing processor sets (PSETs)

With multi-processor systems, you can group processors together to form processor sets, also known as PSETs. By creating PSETs, you isolate CPU resources for users or applications.

WLM allows you to define workload groups based on PSETs. If you then specify SLOs for these workload groups, WLM automatically adjusts the number of processors in the PSETs based on progress toward the SLOs. Here is a PSET workload group Batch that gets five processors, but only between 10pm and 4am. At other times, Batch gets a minimum of one processor.

Group	Batch
Priority	1
CPU shares	500 shares (5 CPUs)
Condition	Time is between 10pm and 4am

Automatically resizing virtual partitions

HP’s virtual partitions allow you to partition a server, with each partition consisting of one or more processors. Each virtual partition runs its own instance of the HP-UX operating system. You can adjust the partition size by the dynamic addition and deletion of processors. With WLM, you can automate the resizing of partitions.

Consider a system with two virtual partitions. The SLO for the Apps workload group on vPar 1 has a higher priority than the SLO for vPar 0’s workload group.

When CPU usage for the Apps group reaches a certain point, WLM automatically migrates a CPU from vPar 0 to vPar 1 to satisfy the higher priority SLO.

Group	Apps
Priority	1
Goal	Match CPU allocation to consumption

Automatically resizing nPartitions via Instant Capacity CPUs

HP’s nPartitions (nPars) allow you to partition a server. Each partition provides hardware isolation as well as operating-system isolation. If you have the Instant Capacity (formerly known as iCOD) software configured on each partition, you can configure WLM to “move” CPUs to the partitions where they are most needed. (Given the hardware isolation, the CPUs are not literally moved: WLM deactivates a CPU on one partition, then activates a CPU on another partition—giving the appearance of moving CPUs without incurring a charge for an additional CPU.)

Consider a system with two nPars. nPar 0 is running the production, customer-accessible version of a shopping web site. nPar 1 runs the test version of this web site. The SLO for the Production workload group on nPar 0 has a higher priority than the Test SLO for nPar 1’s workload group.

When CPU usage, or utilization, for the Production group reaches a certain point, WLM automatically migrates a CPU from nPar 1 to nPar 0 to satisfy the higher priority SLO.

Group	Production
Priority	1
Goal	Match CPU allocation to consumption

What is the ideal environment for WLM?

You will benefit most from WLM if your environment meets one or more of the following conditions:

- Run more than one workload concurrently on a server—regardless of whether the workloads all run under one instance of HP-UX or in separate partitions, each with its own instance of HP-UX. These workloads could be multiple database servers, a database server and an applications server, or any other combination of workloads, provided that they are using HP 9000 servers running HP-UX 11i v1 or HP 9000 servers or HP Integrity servers running HP-UX 11i v2.
- Have workloads that can be prioritized.
- Have an important workload with end-user performance requirements.
- Desire consistent performance from applications under varying application and system loads.
- Run Serviceguard and need to ensure proper prioritization of workloads after a failover.
- Want more control over resource allocation than PRM provides.

Using WLM on multiple servers

WLM manages workloads on individual servers. To manage workloads on multiple servers, install and configure WLM on each of the servers.

WLM and PRM

The Workload Manager (WLM) and Process Resource Manager (PRM) products both work by modifying and enabling a PRM configuration. Only use one of these products at a time to configure resources on a system.

WLM product directory structure

The WLM directories and files included in the product installation are described in Table 1-3. The table is not a full listing, but presents a majority of the files.

NOTE In addition to the WLM-specific files, the WLM installation ensures that PRM C.03.00 and HP-UX WLM Toolkits (WLMTK) A.01.08 or later are installed. For information on the WLMTK files, see the *HP-UX Workload Manager Toolkits User's Guide* (/opt/wlm/toolkits/doc/WLMTKug.pdf).

Table 1-3 WLM's directories and files

Directory/file	Description
/opt/wlm/wlm.quickref.txt	A WLM quick reference
/opt/wlm/bin/wlmaudit	Command for viewing audit data files
/opt/wlm/bin/wlminfo	Monitoring utility
/opt/wlm/bin/wlmgui	Graphical interface for specifying a WLM configuration and monitoring SLOs
/opt/wlm/bin/wlmcert	Utility for managing security certificates

Table 1-3 WLM's directories and files (Continued)

Directory/file	Description
/opt/wlm/bin/wlmcw	The WLM configuration wizard
/opt/wlm/bin/wlmd	The WLM daemon
/opt/wlm/bin/wlmpard	The WLM global arbiter daemon
/opt/wlm/bin/wlmprmconf	Script for converting PRM configuration files to WLM configuration files
/opt/wlm/lbin/wlmrcvdc	Built-in WLM data collector
/opt/wlm/bin/wlmsend	Command-line/scripting interface for sending metric data to WLM
/opt/wlm/bin/wlmcomd	Communications daemon
/opt/wlm/lbin/coll/glance_app	Command used with `wlmrcvdc` to extract GlancePlus application metrics
/opt/wlm/lbin/coll/glance_gbl	Command used with `wlmrcvdc` to extract GlancePlus global system metrics
/opt/wlm/lbin/coll/glance_prm	Command used with `wlmrcvdc` to extract GlancePlus PRM-specific application metrics
/opt/wlm/lbin/coll/glance_prm_byvg	Command used with `wlmrcvdc` to extract GlancePlus metrics on PRM-managed volume groups

Table 1-3 WLM's directories and files (Continued)

Directory/file	Description
/opt/wlm/bin/wlmd	The WLM daemon
/opt/wlm/bin/wlmpard	The WLM global arbiter daemon
/opt/wlm/bin/wlmprmconf	Script for converting PRM configuration files to WLM configuration files
/opt/wlm/lbin/wlmrcvdc	Built-in WLM data collector
/opt/wlm/bin/wlmsend	Command-line/scripting interface for sending metric data to WLM
/opt/wlm/bin/wlmcomd	Communications daemon
/opt/wlm/lbin/coll/glance_app	Command used with `wlmrcvdc` to extract GlancePlus application metrics
/opt/wlm/lbin/coll/glance_gbl	Command used with `wlmrcvdc` to extract GlancePlus global system metrics
/opt/wlm/lbin/coll/glance_prm	Command used with `wlmrcvdc` to extract GlancePlus PRM-specific application metrics
/opt/wlm/lbin/coll/glance_prm_byvg	Command used with `wlmrcvdc` to extract GlancePlus metrics on PRM-managed volume groups

Table 1-3 WLM's directories and files (Continued)

Directory/file	Description
/opt/wlm/lbin/coll/glance_tt	Command used with `wlmrcvdc` to extract GlancePlus application transactions metrics
/opt/wlm/lbin/coll/glance_tt+	Command used with `wlmrcvdc` to extract GlancePlus application transaction metrics for use in generating new metrics
/opt/wlm/lbin/coll/sg_pkg_active	Command used with `wlmrcvdc` to determine the status of a Serviceguard package
/opt/wlm/lbin/scm/	Directory for Servicecontrol Manager (SCM) tool definitions and subtools
/opt/wlm/lbin/wlmemsmon	WLM's EMS monitor
/opt/wlm/newconfig/etc/opt/resmon/dictionary/wlm.dict	EMS dictionary file
/sbin/init.d/wlm	Start/stop script
/etc/rc.config.d/wlm	File to set environment variables used by the start/stop script
/opt/wlm/lib/libwlm.sl	WLM library that provides the API for data collectors to send data to WLM
/opt/wlm/include/wlm.h	Header file for data collectors

Table 1-3 WLM's directories and files (Continued)

Directory/file	Description
/opt/wlm/share/man/man1m.Z/	Contains man pages on the WLM daemon (`wlmd`), the EMS monitor (`wlmemsmon`), `wlmaudit`, `wlminfo`, `wlmgui`, `wlmcert`, `wlmcomd`, `wlmsend`, `wlmrcvdc`, `wlmpard`, `wlmprmconf`, and the `glance_*` and `sg_pkg_active` scripts
/opt/wlm/share/man/man3.Z/libwlm.3	Man page for the WLM API
/opt/wlm/share/man/man4.Z/wlmconf.4	Man page that provides syntax information for the WLM configuration file
/opt/wlm/share/man/man4.Z/wlmparconf.4	Man page that provides syntax information for the WLM global arbiter configuration file
/opt/wlm/share/man/man5.Z/wlm.5	Man page that gives an overview of WLM
/opt/wlm/share/doc/	Contains the WLM user's guide and release notes
/opt/wlm/share/doc/howto/	Contains white papers on how to perform WLM tasks
/var/opt/wlm/msglog	WLM log file—created when WLM runs
/var/opt/wlm/wlmdstats	Optional statistics log file—created by activating a WLM configuration with the `-l` option to `wlmd`

Table 1-3 WLM's directories and files (Continued)

Directory/file	Description
/var/opt/wlm/wlmpardstats	Optional global arbiter statistics log file—created by activating a WLM global arbiter configuration with the `-l` option to `wlmpard`
/opt/wlm/config/tunables	Master tunables file (read-only: do not edit)
/opt/wlm/examples/wlmconf/README	Information text file for the wlmconf directory
/opt/wlm/examples/wlmconf/	Example WLM configuration files
/opt/wlm/examples/dsi/wlmdstats/README	Information text file for the DSI-wlmdstats example directory
/opt/wlm/examples/perl/scripts/simple_perfmon.pl	Example perl-based data collector
/opt/wlm/toolkits/	Various WLM Toolkit products for integrating WLM with third-party applications

NOTE Read the file /opt/wlm/examples/DISCLAIMER regarding items in the /opt/wlm/examples directory.

The /opt/wlm/examples/wlmconf directory provides example configuration files. The /opt/wlm/examples/dsi/wlmdstats directory provides information on how to collect statistics from `wlmd` to feed into HP MeasureWare. The /opt/wlm/examples/perl/scripts directory includes a script that shows how to collect data using perl.

The /opt/wlm/share/doc/howto directory contains the following white papers:

- perfmon.html

  Writing a Better WLM data collector
- config.html

  Configuring HP-UX Workload Manager Version A.02.00
- tuning.html

  Tuning HP-UX Workload Manager
- flexcap.html

  Using HP-UX Workload Manager to Achieve Flexible Capping

2 Learning WLM by example

This chapter introduces WLM by example and is for those who would like to quickly see and understand what WLM can do. It touches on several of the features of the configuration file, as well as the `wlmd` command, which activates configuration files. This chapter does not cover the entire range of WLM functionality, nor does it explain every detail of the configuration file syntax. References to more detailed information are provided throughout the chapter.

The example configuration files and perl scripts are available in /opt/wlm/examples/userguide/ so that you can try them easily and get a sense of how WLM works. Your results will vary depending on your hardware.

You may also be interested in the Chapter 3, “WLM quick start (the essentials to using WLM),” on page 53.

Getting familiar with the configuration file

Here is a sample configuration file,
/opt/wlm/examples/userguide/initial.wlm:

```
#
# Name:
#       initial.wlm
#
# Version information:
#
#       $Revision: 1.8 $
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.01.02
#       or later.  It uses the cpushares keyword introduced in A.01.02
#       and is, consequently, incompatible with earlier versions of
#       HP-UX WLM.
#
# Requirements:
#       To ensure WLM places the perl scripts below in their assigned
#       workload groups, add "/opt/perl/bin/perl" (without the quotes) to
#       the file /opt/prm/shells.


prm {
    groups = g2 : 2;    # Define a workload group

                        # Assign loop.pl to that workload group
    apps = g2 : /opt/perl/bin/perl loop.pl;

                        # To ensure WLM places the loop.pl script in its
                        # assigned workload group, add "/opt/perl/bin/perl"
                        # (without the quotes) to the file /opt/prm/shells

    gmincpu = g2 : 5;   # Ensure the workload group gets at least 5 CPU
                        # shares, even if its SLO is not active

    gmaxcpu = g2 : 30;  # Ensure the workload group gets no more than 30 CPU
                        # shares
}
```

```
slo test {  # Define an SLO for the workload group

    pri = 1; # Assign the SLO a priority of 1 (top priority)

         # Ensure the SLO requests 15 CPU shares
    cpushares = 15 total;
         # An SLO’s request may not be met--depending on its priority and
         # the amount of available CPU. Of course in this configuration with
         # only one SLO, this SLO will receive its requested CPU shares.

    entity = PRM group g2; # Associate the SLO with the workload group
                           # defined for it
}
```

This file has only one workload group, `g2`, defined. Coincidentally, this group is given the group ID 2 as well. For each FSS workload group you define, you must also specify an SLO, using an `slo` structure. (FSS workload groups are explained in the section “Specifying workload groups (optional)” on page 149.)

If you want to try this configuration file, be sure to update the path for where perl is installed on your system. If you do not have perl on your system, use a shell, such as /usr/bin/sh.

To check the syntax of this file, use the following `wlmd` command:

```
# /opt/wlm/bin/wlmd -c initial.wlm
```

The lack of output indicates that no syntax problems were found.

For more information on setting up workload groups and their SLOs, see “Defining the PRM components” on page 139 and “Defining SLOs” on page 169.

Setting up the workload

Now that we have the configuration file ready, we need a workload. We will use a perl script, named loop.pl, as our workload. The first line of the script is:

```
#!/opt/perl/bin/perl -w
```

You may need to change this line, depending on where the perl you use is installed on your system. The -w option warns about possible errors in the perl script.

To ensure WLM places perl scripts in their assigned workload groups, add the following line (or a line corresponding to where the perl you use is installed on your system) to the file /opt/prm/shells:

```
/opt/perl/bin/perl
```

If you do not have perl installed, you can write a shell script to perform a similar task. Be sure the full path of the shell you use is listed in the file /opt/prm/shells.

The loop.pl script (available in /opt/wlm/examples/userguide/) runs an infinite outer loop, counts in the inner loop, and shows the time spent counting:

```
#!/opt/perl/bin/perl -w
#
# Name:
#       loop.pl
#
# Version information:
#
#       $Revision: 1.4 $

$maxcount = 3000000;

# Here is an infinite loop
while (1) {
    $count = 1;
    $start = time();
    while ($count <= $maxcount) {
        $count++;
    }
    $end = time();
    printf (“loop.pl: Elapsed time=%.2f seconds\n”, $end-$start);
}
```

Invoking WLM with the `wlmd` command

We have our configuration and our workload. Now we are ready to start the workload and run WLM.

Given the amount of output our script generates, you may want to start it in a separate window:

```
# /opt/wlm/examples/userguide/loop.pl
```

If you are interested in seeing which workload group the script is running in, try the following command:

```
# ps -P
```

At this point, the script does not belong to any workload group, so it does not have an entry in the `PRMID` column. For information on other HP-UX commands that support WLM, see “HP-UX command/system call support” on page 405.

Log in as root and use the `wlmd` command to start WLM and enable the configuration:

```
# /opt/wlm/bin/wlmd -a initial.wlm
```

This starts the `wlmd` process. Use the `wlminfo slo` command to determine which SLOs are active and what their CPU allocations (shown in the Shares column) are:

```
# /opt/wlm/bin/wlminfo slo

Tue Aug 26 11:49:52 2003

SLO Name                 Group         Pri     Req Shares  State Concern
test                     g2              1      15      15  PASS
```

Assuming at least 60 seconds have passed, if you run `ps -P` now, you will see that loop.pl belongs to the workload group `g2`.

Also, the script’s output should be slower because WLM is running now. Previously, the script could use as much CPU as it wanted. With WLM enabled, the script’s CPU usage is limited. Thus, the inner loop is taking more time to complete now.

Use `ps` to get the PID of loop.pl and stop it before continuing:

```
# kill PID_of_loop.pl
```

Next we are going to change the configuration file, to add another group. We will also make two copies of the loop.pl script, loop2.pl and loop3.pl, and assign them to groups g2 and g3 respectively. We will remove the loop.pl script. The new configuration, available in the file /opt/wlm/examples/userguide/multiple_groups.wlm, looks like:

```
#
# Name:
#       multiple_groups.wlm
#
# Version information:
#
#       $Revision: 1.8 $
#
# This file is based on the file initial.wlm. Comments below indicate
# changes in this file from the initial.wlm file.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.01.02
#       or later.  It uses the cpushares keyword introduced in A.01.02
#       and is, consequently, incompatible with earlier versions of
#       HP-UX WLM.
#
# Requirements:
#       To ensure WLM places the perl scripts below in their assigned
#       workload groups, add "/opt/perl/bin/perl" (without the quotes) to
#       the file /opt/prm/shells.


prm {
    groups = g2 : 2,
             g3 : 3; # Added group g3

                     # Changed loop.pl to loop2.pl
                     # Placed loop3.pl in the new group
    apps = g2 : /opt/perl/bin/perl loop2.pl,
           g3 : /opt/perl/bin/perl loop3.pl;

                     # Added gmincpu/gmaxcpu values for group g3
    gmincpu = g2 : 5, g3 : 5;
    gmaxcpu = g2 : 30, g3 : 60;
}
```

```
slo test2 { # Changed name of SLO
    pri = 1;
    cpushares = 15 total;
    entity = PRM group g2;
}

slo test3 { # Defined an SLO for the new workload group
    pri = 1;
    cpushares = 20 total;
    entity = PRM group g3;
}
```

Start the scripts loop2.pl and loop3.pl before starting WLM:

```
# /opt/wlm/examples/userguide/loop2.pl
# /opt/wlm/examples/userguide/loop3.pl
```

Notice that the inner loop for both scripts is taking about the same time to complete the counting. Now start WLM with the following command:

```
# /opt/wlm/bin/wlmd -a multiple_groups.wlm -l slo
```

You can use this command even if WLM is already running. WLM will simply activate the new configuration, replacing the existing one.

If you look at the `wlminfo slo` output after the end of the current WLM interval, you will probably find that the test2 SLO is getting 15 CPU shares and the test3 SLO is getting 20 CPU shares.

```
# /opt/wlm/bin/wlminfo slo

Tue Aug 26 11:54:26 2003

SLO Name                 Group        Pri    Req  Shares  State Concern
test2                    g2             1     15      15  PASS
test3                    g3             1     20      20  PASS
```

In this case, the workload group `g3` gets more CPU, and its workload loop3.pl should take less time to complete the inner loop. However, you can alter the outcome by adding another SLO for group `g2` at a lower priority. Here is `g2`’s second SLO. After all priority 1 SLOs are met, WLM attempts to meet this SLO:

```
slo test2_stretch {
        pri = 2;
        cpushares = 30 total;
        entity = PRM group g2;
}
```

Add this slo structure to our configuration file (resulting in the file /opt/wlm/examples/userguide/stretch.wlm) and restart wlmd:

```
# /opt/wlm/bin/wlmd -a stretch.wlm
```

WLM will now try to grant 30 CPU shares to group g2 if additional CPU shares are left after satisfying all the priority 1 SLOs. With the additional CPU, loop2.pl should run faster than loop3.pl.

For more information on wlmd, see “WLM command reference” on page 373.

Using goal-based SLOs

So far, our configurations have provided static CPU allocations. The next progression is to include goal-based SLOs. With this type of SLO, you set bounds on the amount of CPU that WLM can grant and specify a goal for some metric. WLM then adjusts the CPU within those bounds to best meet the goal. Before setting up such a goal, we will run WLM with the configuration below to get a baseline for how much time is spent in the inner loops of loop2.pl and loop3.pl. The configuration below is at /opt/wlm/examples/userguide/baseline.wlm.

```
#
# Name:
#       baseline.wlm
#
# Version information:
#
#       (C) Copyright 2002-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.8 $
#
# This file is based on the file multiple_groups.wlm. Comments below indicate
# either changes in this file from the multiple_groups.wlm file or new
# syntax elements that will be uncommented in a later configuration.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.01.02
#       or later.  It uses the cpushares keyword introduced in A.01.02
#       and is, consequently, incompatible with earlier versions of
#       HP-UX WLM.
```

```
#
# Requirements:
#       To ensure WLM places the perl scripts below in their assigned
#       workload groups, add "/opt/perl/bin/perl" (without the quotes) to
#       the file /opt/prm/shells.

prm {
    groups = g2 : 2,
             g3 : 3;
    apps = g2 : /opt/perl/bin/perl loop2.pl,
           g3 : /opt/perl/bin/perl loop3.pl;
    gmincpu = g2 : 5, g3 : 5;
    gmaxcpu = g2 : 30, g3 : 60;
}

slo test2 {
    pri = 1;
    cpushares = 15 total;
    entity = PRM group g2;
}

slo test3 {
    pri = 1;
    mincpu = 20; # Replaced cpushares statement with mincpu/maxcpu
    maxcpu = 50; # statements because cpushares keyword cannot be used
                 # with the goal keyword, which is introduced below and
                 # will be uncommented in metric_goal.wlm.
#
# (mincpu and maxcpu statements only serve to bound requests for CPU
# shares--they do not make CPU shares requests. When an SLO has a mincpu
# and maxcpu, but no goal or cpushares statement, WLM simply gives the
# SLO's associated workload group the number of CPU shares specified in the
# mincpu statement.)

    entity = PRM group g3;

# The remaining elements will be uncommented in a later configuration.
#       goal = metric count_time3 < 20.0;
}

#  tune count_time3 {
#       coll_argv = wlmrcvdc;
#  }
```

We have added a few lines in anticipation of setting up the goal-based SLO; however, these lines are commented out. We will explore those lines shortly.

Now, start the scripts (if they are not already running) and activate the WLM configuration:

```
# /opt/wlm/examples/userguide/loop2.pl
# /opt/wlm/examples/userguide/loop3.pl
# /opt/wlm/bin/wlmd -a baseline.wlm
```

With this configuration, we get the following statistics:

```
loop2.pl: Elapsed time = 37.00 seconds
loop3.pl: Elapsed time = 29.00 seconds
```

Also, according to the `wlminfo slo` output, loop2.pl is getting 15 CPU shares and loop3.pl is getting 20 shares. We expected these values because of the `cpushares` statement in the test2 SLO and the `mincpu` value in the test3 SLO. With no `goal` or `cpushares` statement in the test3 SLO, WLM simply gives the associated workload group its `mincpu` based on the SLO's priority and the amount of available CPU.

```
# /opt/wlm/bin/wlminfo slo

Tue Aug 26 11:57:09 2003

SLO Name                 Group         Pri     Req Shares  State Concern
test2                    g2              1      15     15  PASS
test3                    g3              1      20     20  PASS
```

These numbers give us a good idea of the elapsed times we can expect to see for a given level of CPU.

Stop WLM and loop3.pl:

```
# /opt/wlm/bin/wlmd -k
```

```
# kill PID_of_loop3.pl
```

Remove the comment characters (# symbols) from the file baseline.wlm to get the file /opt/wlm/examples/userguide/metric_goal.wlm. The SLO named test3 in metric_goal.wlm has a goal of keeping the metric count_time3 below 20.0. Based on the `mincpu` and `maxcpu` values, the SLO can use between 20 and 50 CPU shares to meet this goal. The count_time3 metric is going to be the time that the workload spends in its inner loop. The workload, which is based on loop3.pl, will be represented by the script loop3_wlmsend.pl and is described below.

The other previously commented lines include a new structure. This is a `tune` structure. It starts the command `wlmrcvdc`, which reads in values for a metric. Because the structure is named for the metric count_time3, the received values are used in place of count_time3 in the `goal` statement. These values are sent to `wlmrcvdc` by the `wlmsend` command, shown in the perl script below.

As mentioned before, we use the `wlmsend` command to send values for the count_time3 metric. Because this metric is a measure of the time spent in loop3.pl's inner loop, the script itself is a great place for the `wlmsend` command. Adding the `wlmsend` command, loop3.pl becomes loop3_wlmsend.pl (available in /opt/wlm/examples/userguide/). This is how the new script looks:

```
#!/opt/perl/bin/perl -w
#
# Name:
#       loop3_wlmsend.pl
#
# Version information:
#
#       (C) Copyright 2002-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.5 $

$maxcount = 3000000;

# Here is an infinite loop
while (1) {
    $count = 1;
    $start = time();
    while ($count <= $maxcount) {
        $count++;
    }
    $end = time();
    $time_spent = $end - $start;
    printf ("loop3_wlmsend.pl: Elapsed time=%.2f seconds \n", $time_spent);

    # Here is the wlmsend line:
    system "/opt/wlm/bin/wlmsend count_time3 $time_spent";
    # The wlmsend command sets the value of count_time3 to $time_spent
    # and informs WLM of this value by means of the tune structure
    # named for the count_time3 metric, starting in the file
    # metric_goal.wlm.
}
```

The `system` keyword executes the command

```
/opt/wlm/bin/wlmsend count_time3 $time_spent
```

as a UNIX command, with `$time_spent` replaced by its actual value. We will not collect metrics on the loop2.pl script, so it is not going to have a `wlmsend` line.

Now, activate WLM and start the scripts:

```
# /opt/wlm/bin/wlmd -a metric_goal.wlm
# /opt/wlm/examples/userguide/loop2.pl
# /opt/wlm/examples/userguide/loop3_wlmsend.pl
```

NOTE We started WLM first in this example because the `wlmsend` invocation in loop3_wlmsend.pl expects WLM to prepare for data to arrive. If WLM is not ready, you will receive the following message:

```
wlmsend: Wait time expired: Rendezvous point does not exist
```

After running the perl scripts under the above configuration for a few minutes, loop3_wlmsend.pl's loop time settles at 18 seconds. The time for loop2.pl continues to be approximately 43 seconds. As for CPU allocation, loop2.pl receives 15 CPU shares, while loop3_wlmsend.pl receives 31. The group to which loop3_wlmsend.pl belongs, namely `g3`, does not approach its `maxcpu` value of 50 in this case because its goal is easily met with 31 shares.

With the loop time at 18 seconds, the goal is satisfied and you will see the from the `wlminfo` output that the SLO is passing:

```
# /opt/wlm/bin/wlminfo slo -v

Tue Aug 26 12:00:47 2003

SLO Name         Group    Pri Metric   Goal    Req Shares  State Concern
test2            g2         1      -      -     15     15   PASS
test3            g3         1     18     20     31     31   PASS
```

However, if the goal were not being satisfied with the current resources, you could either:

- Increase the workload group's `gmaxcpu` value and/or the SLO's `maxcpu` value

or

- Change the `goal` statement to compare count_time3 against a higher value

Rather than make the goal easier, let's see what happens when we make the goal harder—if not impossible. First, change

```
goal = metric count_time3 < 20.0;
```

to

```
goal = metric count_time3 < 5;
```

WLM gathers workload performance data and adjusts CPU allocation based on that data once per WLM interval, which is 60 seconds by default. Thus, it may take a few minutes to satisfy the goal. If you want WLM to gather data and adjust CPU allocations more frequently, assign a smaller value to `wlm_interval`. You assign this value inside a global `tune` structure (an unnamed `tune` structure) in the configuration file. For information on the `wlm_interval`, see "Specifying the WLM interval (optional)" on page 203.

In addition to making the goal harder, decrease the WLM interval by adding the following `tune` structure:

```
tune {
   wlm_interval = 30;
}
```

This new configuration is available as the file /opt/wlm/examples/userguide/harder_goal.wlm. Activate the new configuration as follows:

```
# /opt/wlm/bin/wlmd -a harder_goal.wlm
```

The output from the scripts is now:

```
loop2.pl: Elapsed time = 38.00 seconds
loop3_wlmsend.pl: Elapsed time = 13.00 seconds
```

The goal has not been met. In the `wlminfo slo -v` output, the test3 SLO shows a FAIL entry in the State column. The metric value is 13. It also shows the CPU allocations, or shares, for loop2.pl and loop3_wlmsend.pl are 15 and 45 respectively.

```
# /opt/wlm/bin/wlminfo slo -v

Tue Aug 26 12:04:43 2003

SLO Name          Group     Pri Metric    Goal     Req Shares  State Concern
test2             g2          1      -       -      15     15  PASS
test3             g3          1     13       5    44.5     45  FAIL
```

That leaves 35 CPU shares on the system. By default, those excess shares go to the default user workload group `OTHERS`. We can distribute the excess to our defined workload groups by placing a tunable in the global `tune` structure. The new tunable is `distribute_excess`. We specify it in the `tune` structure with a value of 1:

```
tune {
   wlm_interval = 30;
   distribute_excess = 1;
}
```

giving the new file /opt/wlm/examples/userguide/distribute_excess.wlm. Activate this new configuration:

```
# /opt/wlm/bin/wlmd -a distribute_excess.wlm
```

After a few WLM intervals pass, the output from the scripts is:

```
loop2.pl: Elapsed time = 21.00 seconds
loop3_wlmsend.pl: Elapsed time = 12.00 seconds
```

So the execution time of the inner loops has decreased more than it did under the previous configuration. This is because the CPU shares for loop2.pl and loop3_wlmsend.pl have increased to 30 and 60 respectively. The workload groups are getting the excess CPU, as expected.

Stop the two scripts:

```
# kill PID_of_loop2.pl
```

```
# kill PID_of_loop3_wlmsend.pl
```

For more information on using goals, see “Specifying a goal (optional)” on page 183. For information on `distribute_excess`, see “Distributing excess CPU to your workloads (optional)” on page 206.

Conditional activation of SLOs

You can activate SLOs depending on the time of the day, day of the week/month/year, system events, or application metrics. Use the `condition` keyword in an `slo` structure to enable SLOs conditionally.

Up to now, we have usually run two scripts at the same time. This time, we will run them one at a time.

Start WLM with the distribute_excess.wlm configuration and loop2.pl:

```
# /opt/wlm/bin/wlmd -a distribute_excess.wlm
# /opt/wlm/examples/userguide/loop2.pl
```

Running only loop2.pl, we see from the `wlminfo group` output

```
# /opt/wlm/bin/wlminfo group

Tue Aug 26 12:07:56 2003

Workload Group   PRMID  CPU Shares  CPU Util  Mem Shares  State
OTHERS               1       10.00      0.00        0.00  ON
g2                   2       30.00      0.00        0.00  ON
g3                   3       60.00      2.23        0.00  ON
```

that the workload group `g2` gets 30 CPU shares from its test2 SLO. Also, we see that the test3 SLO gets 60 shares for its workload group `g3`—despite the fact that there are no active applications in `g3`. In this case, the SLO does not need to be active. Similarly, if we stop loop2.pl and start loop3_wlmsend.pl, we find that the test2 SLO still gets 30 CPU shares for its workload group, even though the group has no active applications. WLM's `condition` keyword solves this problem.

First, we write a small script that periodically reports the status of the applications to WLM. The following script, reporter.pl, reports information every minute to WLM on whether the loop2.pl and loop3.pl scripts are running. (We're changing from loop3_wlmsend.pl back to loop3.pl.) The reporter.pl script uses two metrics: loop2_active and loop3_active. If a script is active, its metric is set to 1; otherwise, it is set to 0.

Here is /opt/wlm/examples/userguide/reporter.pl:

```
#!/opt/perl/bin/perl -w
#
# Name:
#       reporter.pl
#
# Version information:
#
#       (C) Copyright 2002-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.5 $

# This script uses the ps command to see if the scripts loop2.pl and
# loop3.pl are running. If a script is running, the wlmsend command sends a
# value of 1 for the script; otherwise, it sends a value of 0.

# Here is an infinite loop

while (1) {
    $loop2_running = `env UNIX95=  ps -C loop2.pl | wc -l`;
    --$loop2_running; # Line count is one more due to header

    $loop3_running = `env UNIX95=  ps -C loop3.pl | wc -l`;
    --$loop3_running; # Line count is one more due to header

    # The following wlmsend commands set the values for the metrics
    # loop2_active and loop3_active to $loop2_running and
    # $loop3_running respectively. WLM picks up these values because of
    # the tune structures named for the metrics in the file
    # condition.wlm.

    system "/opt/wlm/bin/wlmsend loop2_active $loop2_running";
    system "/opt/wlm/bin/wlmsend loop3_active $loop3_running";

    sleep(60); # Wait for 60 seconds
}
```

We must modify the distribute_excess.wlm file so that WLM expects the loop2_active and loop3_active metrics. We will also add a condition to each SLO so it is active only when the metric value inside the condition is nonzero. In addition, we remove the global `tune` structure and the count_time3 `tune` structure. Lastly, we change loop3_wlmsend.pl to loop3.pl and replace the `goal` statement in the test3 SLO with a `cpushares` statement. The altered file, /opt/wlm/examples/userguide/condition.wlm, is below:

```
#
# Name:
#       condition.wlm
#
# Version information:
#
#       (C) Copyright 2002-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.9 $
#
# This file is based on the file distribute_excess.wlm. Comments below
# either indicate changes in this file from the distribute_excess.wlm file
# or explain elements of this file.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.01.02
#       or later.  It uses the cpushares keyword introduced in A.01.02
#       and is, consequently, incompatible with earlier versions of
#       HP-UX WLM.
#
# Requirements:
#       To ensure WLM places the perl scripts below in their assigned
#       workload groups, add "/opt/perl/bin/perl" (without the quotes) to
#       the file /opt/prm/shells.


# Removed tune structure that changed WLM interval and set
# distribute_excess

prm {
    groups = g2 : 2,
             g3 : 3;
    apps = g2 : /opt/perl/bin/perl loop2.pl,
           g3 : /opt/perl/bin/perl loop3.pl; # Changed loop3_wlmsend.pl
                                             # to loop3.pl
    gmincpu = g2 : 5, g3 : 5;
    gmaxcpu = g2 : 30, g3 : 60;
}
```

```
slo test2 {
    pri = 1;
    cpushares = 15 total;
    entity = PRM group g2;
    condition = metric loop2_active; # Added condition statement so
}                                    # that SLO is active only when the
                                     # metric loop2_active is nonzero,
                                     # which indicates the loop2.pl
                                     # workload is running

slo test3 {       # Removed the mincpu, maxcpu, and goal statements
    pri = 1;
    cpushares = 50 total;       # Added a cpushares statement
    entity = PRM group g3;
    condition = metric loop3_active; # Added condition statement so
}                                    # that SLO is active only when the
                                     # metric loop3_active is nonzero,
                                     # which indicates the loop3.pl
                                     # workload is running

# Added tune structure to get value for loop2_active metric;
# value for metric is set by using wlmsend in reporter.pl script
tune loop2_active {
    coll_argv = wlmrcvdc;
}

# Added tune structure to get value for loop3_active metric;
# value for metric is set by using wlmsend in reporter.pl script
tune loop3_active {
    coll_argv = wlmrcvdc;
}

# Removed count_time3 tune structure
```

Stop the loop3_wlmsend.pl script if you have it running:

```
# kill PID_of_loop3_wlmsend.pl
```

We now start WLM, reporter.pl, and loop2.pl:

```
# /opt/wlm/bin/wlmd -a condition.wlm
# /opt/wlm/examples/userguide/reporter.pl
# /opt/wlm/examples/userguide/loop2.pl
```

NOTE For this example, we started WLM first because the `wlmsend` invocation in reporter.pl expects WLM to prepare for data to arrive. If WLM is not ready, you will receive the following message:

```
wlmsend: Wait time expired: Rendezvous point does not exist
```

Looking at the `wlminfo slo` output (after the end of the current WLM interval), we see that the test3 SLO is inactive (State is OFF) and that it is only getting the `gmincpu` shares (5 in this case). However, the test2 SLO is getting 15 shares.

```
# /opt/wlm/bin/wlminfo slo

Tue Aug 26 12:11:24 2003

SLO Name                 Group         Pri     Req Shares  State Concern
test2                    g2              1      15      15  PASS
test3                    g3              1       -       5  OFF   Disabled
```

If we stop loop2.pl and start loop3.pl, we find that the test2 SLO becomes inactive and gets only its `gmincpu` of 5 shares, whereas the test3 SLO is active and getting 50 shares.

```
# /opt/wlm/bin/wlminfo slo

Tue Aug 26 12:14:24 2003

SLO Name                 Group         Pri     Req Shares  State Concern
test2                    g2              1       -       5  OFF   Disabled
test3                    g3              1      50      50  PASS
```

Stop the two scripts:

```
# kill PID_of_loop3.pl
```

```
# kill PID_of_reporter.pl
```

For more information on `wlmsend` and `wlmrcvdc`, see “WLM command reference” on page 373. For information on the `condition` keyword, see “Specifying when the SLO is active (optional)” on page 192.

Using usage goals

Now we will see how a usage goal can be used. A usage goal is better described as a utilization goal. A workload group's utilization is the amount of CPU it uses divided by its current CPU allocation, that is, (CPU used) / (CPU allocation). This type of goal has the benefit of automatically increasing a workload group's CPU allocation when more CPU is needed, as well as freeing some of a workload group's CPU allocation when the workload is less busy.

In this example, we will work with loop3.pl to create loop3_usage_goal.pl. First, we change `$maxcount` to 1000000. Also, we remove the line with the `system` command. Next, we add a call to the `sleep` function so that the script does less CPU-intensive work. With less CPU work, loop3.pl's workload group does not need the same level of CPU as before. As a result of the usage goal, the workload group makes its unused CPU cycles available to other workload groups that may need it.

The /opt/wlm/examples/userguide/loop3_usage_goal.pl script is now:

```
#!/opt/perl/bin/perl -w
#
# Name:
#       loop3_usage_goal.pl
#
# Version information:
#
#       (C) Copyright 2002-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.4 $

$maxcount = 1000000;
```

```
# Here is an infinite loop
while (1) {
    $count = 1;
    sleep(7); # Sleep for 7 seconds
    $start = time();
    while ($count <= $maxcount) {
        $count++;
    }
    $end = time();
    $time_spent = $end - $start;
    printf "loop3_usage_goal.pl: Elapsed time=%.2f seconds \n", $time_spent;
}
```

The configuration file that we are going to use is shown below. We are not going to run loop2.pl for this example, so we do not need to worry about group `g2`. We added a usage goal for the test3 SLO. The goal is to keep the CPU utilization between 60% and 90%. The `mincpu` has been reduced to 5. The file is /opt/wlm/examples/userguide/usage_goal.wlm.

```
#
# Name:
#       usage_goal.wlm
#
# Version information:
#
#       (C) Copyright 2002-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.9 $
#
# This file is based on the file condition.wlm. Comments below indicate
# changes in this file from the condition.wlm file.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.01.02
#       or later.  It uses the cpushares keyword introduced in A.01.02
#       and is, consequently, incompatible with earlier versions of
#       HP-UX WLM.
#
# Requirements:
#       To ensure WLM places the perl scripts below in their assigned
#       workload groups, add "/opt/perl/bin/perl" (without the quotes) to
#       the file /opt/prm/shells.
```

```
prm {
    groups = g2 : 2,
             g3 : 3;
    apps = g2 : /opt/perl/bin/perl loop2.pl,
           g3 : /opt/perl/bin/perl loop3_usage_goal.pl;
    gmincpu = g2 : 5, g3 : 5;
    gmaxcpu = g2 : 30, g3 : 60;
}

slo test2 {
    pri = 1;
    cpushares = 15 total;
    entity = PRM group g2; # Removed condition statement
}

slo test3 {
    pri = 1;
    mincpu = 5;              # Changed mincpu from 20 to 5
    maxcpu = 50;
    entity = PRM group g3;
    goal = usage _CPU 60 90; # Changed from a metric goal to a
}                            # usage goal
                             #
                             # Removed condition statement

# Set a small interval now that configuration includes a usage goal
tune {
    wlm_interval = 5;
}
```

Now we start the loop3_usage_goal.pl script. We also start WLM.

```
# /opt/wlm/examples/userguide/loop3_usage_goal.pl
# /opt/wlm/bin/wlmd -a usage_goal.wlm
```

At the end of the current WLM interval, `wlminfo` has updated data available. Initially, there are 28 shares allocated for the test3 SLO.

```
# /opt/wlm/bin/wlminfo slo

Tue Aug 26 12:16:39 2003

SLO Name              Group        Pri    Req Shares  State Concern
test2                 g2             1     15     15  PASS
test3                 g3             1   27.5     28  PASS
```

Later, as the CPU usage of the workload stabilizes at about 3 shares, the allocation is set to 5 shares, resulting in a CPU utilization of 60%.

```
# /opt/wlm/bin/wlminfo slo

Tue Aug 26 12:17:39 2003

SLO Name             Group        Pri     Req Shares  State Concern
test2                g2             1      15     15  PASS
test3                g3             1       5      5  PASS
```

The behavior may be different for your system. If you find that the utilization is higher than 60% from the beginning, increase the value passed to the `sleep` function inside loop3_usage_goal.pl. If you find that initially the utilization is too low, then decrease the value passed to the `sleep` function inside loop3_usage_goal.pl.

For information on usage goals, see "Specifying a goal (optional)" on page 183.

3 WLM quick start (the essentials to using WLM)

This chapter presents an overview of the techniques and tools available for using WLM. It serves as a condensed version of this entire user's guide, exposing you to the essentials and allowing you to quickly get started using WLM. The terminology of WLM as well as WLM concepts and the syntax of the WLM configuration files are not covered here. That information is discussed in the remainder of the book.

The chapter first gives an overview of a WLM session. Then, it provides background information on various ways to use WLM, including how to complete several common WLM tasks. Lastly, it discusses how to monitor WLM and its effects on your workloads.

Show me WLM in action

This section provides a quick overview of various commands associated with using WLM. It takes advantage of some of the configuration files and scripts that are used in the chapter "Learning WLM by example" on page 29. These files are in the directory /opt/wlm/examples/userguide/ as well as on http://www.hp.com/go/wlm.

To become familiar with WLM, how it works, and some related commands, complete the following procedure:

Step 1. Log in as root

Step 2. Start WLM with an example configuration file:

```
# /opt/wlm/bin/wlmd -a /opt/wlm/examples/userguide/multiple_groups.wlm
```

The file multiple_groups.wlm is shown below. This configuration:

1. Defines two workload groups: `g2` and `g3`.
2. Assigns applications (in this case, perl programs) to the groups. (With shell/perl programs, give the full path of the shell or perl followed by the name of the program.)
3. Sets bounds on CPU usage. The number of CPU shares for the workload groups can never go below the `gmincpu` or above the `gmaxcpu` values. These values take precedence over the minimum and maximum values that you can optionally set in the `slo` structures.
4. Defines an SLO (service-level objective) for `g2`. The SLO is priority 1 and requests 15 CPU shares for `g2`.
5. Defines a priority 1 SLO for `g3` that requests 20 CPU shares.

```
#
# Name:
#       multiple_groups.wlm
#
# Version information:
#
#       $Revision: 1.8 $
#
```

```
# This file is based on the file initial.wlm. Comments below indicate
# changes in this file from the initial.wlm file.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.01.02
#       or later.  It uses the cpushares keyword introduced in A.01.02
#       and is, consequently, incompatible with earlier versions of
#       HP-UX WLM.
#
# Requirements:
#       To ensure WLM places the perl scripts below in their assigned
#       workload groups, add "/opt/perl/bin/perl" (without the quotes) to
#       the file /opt/prm/shells.


prm {
    groups = g2 : 2,
             g3 : 3; # Added group g3

                     # Changed loop.pl to loop2.pl
                     # Placed loop3.pl in the new group
    apps = g2 : /opt/perl/bin/perl loop2.pl,
           g3 : /opt/perl/bin/perl loop3.pl;

                     # Added gmincpu/gmaxcpu values for group g3
    gmincpu = g2 : 5, g3 : 5;
    gmaxcpu = g2 : 30, g3 : 60;
}

slo test2 { # Changed name of SLO
    pri = 1;
    cpushares = 15 total;
    entity = PRM group g2;
}

slo test3 { # Defined an SLO for the new workload group
    pri = 1;
    cpushares = 20 total;
    entity = PRM group g3;
}
```

Step 3. See what messages a WLM startup produces. Start another session to view the WLM message log:

```
# tail -f /var/opt/wlm/msglog
```

```
08/29/02 08:35:23 [I] (p6128) wlmd initial command line:
08/29/02 08:35:23 [I] (p6128)   argv[0]=/opt/wlm/bin/wlmd
08/29/02 08:35:23 [I] (p6128)   argv[1]=-a
08/29/02 08:35:23 [I] (p6128)   argv[2]=/opt/wlm/examples/userguide/multiple_gro
ups.wlm
08/29/02 08:35:23 [I] (p6128) what: @(#)HP-UX WLM A.02.00 (2002_03_21_17_04_11)
hpux_11.00
08/29/02 08:35:23 [I] (p6128) dir: @(#) /opt/wlm
08/29/02 08:35:23 [I] (p6128) SLO file path: /opt/wlm/examples/userguide/multipl
e_groups.wlm
08/29/02 08:35:26 [I] (p6128) wlmd 6128 starting
```

The text in the log shows when the WLM daemon `wlmd` started, as well as what arguments it was started with—including the configuration file used.

Step 4. See that the workload groups are in effect

The `prmlist` command shows current configuration information. (This PRM, or Process Resource Manager, command is available because WLM uses PRM to provide some of the WLM functionality.)

```
# /opt/prm/bin/prmlist
```

```
PRM configured from file:  /var/opt/wlm/tmp/wmprmBAAa06128
File last modified:        Thu Aug 29 08:35:23 2002

PRM Group                       PRMID    CPU Entitlement
-------------------------------------------------------
OTHERS                              1             65.00%
g2                                  2             15.00%
g3                                  3             20.00%

PRM User                        Initial Group                 Alternate Group(s)
--------------------------------------------------------------------------------
root                            (PRM_SYS)

PRM Application                 Assigned Group                Alternate Name(s)
--------------------------------------------------------------------------------
/opt/perl/bin/perl              g2                            loop2.pl
/opt/perl/bin/perl              g3                            loop3.pl
```

Starting with WLM version A.02.00, a web interface to the `prmlist` command is available. For information, see the wlm_watch(1M) man page.

In addition, starting with WLM version A.02.01, you can use the `wlminfo` command, which shows CPU Shares and utilization (CPU Util) for each workload group.

```
# /opt/wlm/bin/wlminfo group
```

```
Thu Aug 29 08:36:38 2002

Workload Group    PRMID  CPU Shares  CPU Util  Mem Shares  State
OTHERS                1       65.00      0.00        0.00  ON
g2                    2       15.00      0.00        0.00  ON
g3                    3       20.00      0.00        0.00  ON
```

Step 5. Start the scripts referenced in the configuration file.

a. WLM checks the files /etc/shells and /opt/prm/shells to ensure one of them lists any shell or interpreter, including perl, used in a script. If the shell or interpreter is not in either of those files, WLM ignores its application record (the workload group assignment in an `apps` statement).

Add the following line to the file /opt/prm/shells so that the application manager can correctly assign the perl programs to workload groups:

```
/opt/perl/bin/perl
```

b. The following scripts produce output so you may want to start them in a new window. Start the two scripts loop2.pl and loop3.pl:

```
# /opt/wlm/examples/userguide/loop2.pl &
```

```
# /opt/wlm/examples/userguide/loop3.pl &
```

These scripts start in the `PRM_SYS` group because you started them as the root user. However, the application manager soon moves them (within 30 seconds) to their assigned groups, `g2` and `g3`. After waiting 30 seconds, run the following `ps` command to see that the processes have been moved to their assigned workload groups:

```
# ps -efP | grep loop
```

The output will include the following items (column headings are included for convenience):

```
PRMID    PID TTY        TIME COMMAND
   g3   6463 ttyp1      1:42 loop3.pl
   g2   6462 ttyp1      1:21 loop2.pl
```

Step 6. Manage workload group assignments:

a. Start a process in the group g2:

```
# /opt/prm/bin/prmrun -g g2 /opt/wlm/examples/userguide/loop.pl
```

b. Verify that loop.pl is in g2 with ps:

```
# ps -efP | grep loop
```

The output will confirm the group assignment:

```
g2   6793 ttyp1      0:02 loop.pl
```

c. Move the process to another group.

Use the PID for loop.pl from the last step to move loop.pl to the group g3:

```
# /opt/prm/bin/prmmove g3 -p loop.pl_PID
```

In this case, *loop.pl_PID* is 6793.

Step 7. Verify workload group assignments:

The ps command has two options related to WLM.

-P	Shows PRM IDs (workload group IDs) for each process
-R *workload_group*	Shows ps listing for only the processes in *workload_group*

Looking at all the processes, we get output including the items shown below (column headings are included for convenience):

```
# ps -efP | grep loop
```

```
PRMID    PID TTY        TIME COMMAND
   g3   6793 ttyp1      1:52 loop.pl
   g3   6463 ttyp1      7:02 loop3.pl
   g2   6462 ttyp1      4:34 loop2.pl
```

Focusing on the group `g3`, we get the following output:

```
# ps -R g3
```

```
    PID TTY       TIME COMMAND
    6793 ttyp1     1:29 loop.pl
    6463 ttyp1     6:41 loop3.pl
```

Step 8. Check CPU usage

The `wlminfo` command shows CPU usage (utilization) by workload group. The command's output, which may be slightly different on your system, is shown below.

```
# /opt/wlm/bin/wlminfo group
```

```
Workload Group   PRMID  CPU Shares  CPU Util  Mem Shares  State
OTHERS               1       65.00      0.00        0.00  ON
g2                   2       15.00     14.26        0.00  ON
g3                   3       20.00     19.00        0.00  ON
```

This output shows that both groups are using CPU up to their allocations. If the allocations were increased, the groups' usage would probably increase to match the new allocations.

Step 9. Stop WLM:

```
# /opt/wlm/bin/wlmd -k
```

Step 10. See what messages a WLM shutdown produces:

Running the `tail` command again

```
# tail -f /var/opt/wlm/msglog
```

you will see messages similar to the following:

```
08/29/02 09:06:55 [I] (p6128) wlmd 6128 shutting down
08/29/02 09:06:55 [I] (p7235) wlmd terminated (by request)
```

Step 11. Stop the loop.pl, loop2.pl, and loop3.pl perl programs.

Where is WLM installed?

The following table shows where WLM and some of its components are installed.

Table 3-1 **WLM installation directories**

Item	Installation path
WLM	/opt/wlm/
WLM Toolkits	/opt/wlm/toolkits/
Man pages for WLM and its Toolkits	/opt/wlm/share/man/

Installing WLM ensures the product Process Resource Manager, sometimes referred to as PRM, is also installed. This product is installed at /opt/prm/.

Can I see how WLM will perform—without actually affecting my system?

WLM provides a passive mode that allows you to see how WLM will approximately respond to a given configuration—without putting WLM in charge of your system's resources. Using this mode, enabled with the `-p` option to `wlmd`, you can gain a better understanding of how various WLM features work. In addition, you can check that your configuration behaves as expected—with minimal effect on the system.

For example, with passive mode, you can determine:

- How does a `cpushares` statement work?
- How do goals work? Is my goal set up correctly?
- How might a particular `cntl_convergence_rate` value or the values of other tunables affect allocation change?
- How does a usage goal work?
- Is my global configuration file set up as I wanted? If I used global arbitration on my production system, what might happen to the CPU layouts?
- Is a user's default workload set up as I expected?
- Can a user access a particular workload?
- When an application is run, which workload does it run in?
- Can I run an application in a particular workload?
- Are the alternate names for an application set up correctly?

For more information on how to use WLM's passive mode, as well as explanations of how passive mode does not always represent actual WLM operations, see "Trying a configuration without affecting the system" on page 224.

Activate a configuration in passive mode by logging in as root and running the command:

```
# /opt/wlm/bin/wlmd -p -a config.wlm
```

substituting your configuration file's name for *`config.wlm`*.

The WLM global arbiter, `wlmpard`, which is used in managing SLOs across virtual partitions, also provides a passive mode.

How do I start WLM?

Before starting WLM (activating a configuration), you may want to try the configuration in passive mode, discussed in the previous section. Otherwise, you can activate your configuration by logging in as root and running the command:

```
# /opt/wlm/bin/wlmd -a config.wlm
```

substituting your configuration file's name for *config.wlm*.

When you run the `wlmd -a` command, WLM starts the data collectors you have specified in the WLM configuration.

Although data collectors are not necessary in every case, be sure to monitor any data collectors you do have. Because data collection is a critical link in the effective maintenance of your configured service-level objectives, you need to be aware when a collector exits unexpectedly. One method for monitoring collectors is to use `wlminfo slo`.

For information on creating your WLM configuration, see the section "How do I create a configuration file?" on page 63.

WLM automatically logs informational messages to the file /var/opt/wlm/msglog. In addition, WLM can log data that allows you to verify WLM's management as well as to fine-tune your WLM configuration file. To log this data, use the `-l` option. This option causes WLM to log data to /var/opt/wlm/wlmdstats. The following command line starts WLM, logging data for SLOs every third WLM interval:

```
# /opt/wlm/bin/wlmd -a config.wlm -l slo=3
```

For more information on the `-l` option, see the wlmd(1M) man page.

How do I stop WLM?

With WLM running, stop it by logging in as root and running the command:

```
# /opt/wlm/bin/wlmd -k
```

How do I create a configuration file?

The WLM configuration file is simply a text file.

To create your own WLM configuration file, use one or more of the following techniques:

- Determine which example configurations can be useful in your environment and modify them appropriately. For information on example configurations, see “Where can I find example WLM configurations?” on page 65.
- Create a new configuration using:
  - `vi` or any other text editor
  - the WLM configuration wizard, which is discussed in the section “What is the easiest way to configure WLM?” on page 64
  - the WLM graphical user interface, `wlmgui`

  Using the wizard, the configuration syntax is almost entirely hidden. With `wlmgui`, you need to be somewhat familiar with the syntax.
- Enhance your configuration created from any of the above methods by modifying it based on ideas from the following sections:
  - “How do I put an application under WLM control?” on page 67
  - “What are some common WLM tasks?” on page 83

For configuration file syntax, see the wlmconf(4) man page.

What is the easiest way to configure WLM?

The easiest and quickest method to configure WLM is to use the WLM configuration wizard.

NOTE Set your `DISPLAY` environment variable before starting the wizard.

To start the wizard, run the following command:

```
# /opt/wlm/bin/wlmcw
```

The wizard provides an easy way to create initial WLM configurations. To maintain simplicity, it does not provide all the functionality available in WLM. Also, the wizard does not allow you to edit existing configurations.

Where can I find example WLM configurations?

WLM and its toolkits come with example WLM configuration files. These files are located in the directories indicated in the table below.

Table 3-2 **Example WLM configurations**

For	See example WLM configurations in the directory
Examples showing a range of WLM functionality	/opt/wlm/examples/wlmconf/
Examples used in this chapter	/opt/wlm/examples/userguide/
Using WLM with Apache web servers	/opt/wlm/toolkits/apache/config/
Using WLM to manage job duration	/opt/wlm/toolkits/duration/config/
Using WLM with Oracle databases	/opt/wlm/toolkits/oracle/config/
Using WLM with SAS software	/opt/wlm/toolkits/sas/config/
Using WLM with SNMP agents	/opt/wlm/toolkits/snmp/config/
Using WLM with BEA WebLogic Server	/opt/wlm/toolkits/weblogic/config/

These configurations are also available on http://www.hp.com/go/wlm, from the “case studies / example configurations” page.

How does WLM control applications?

WLM controls your applications after you:

1. Define workload groups for each workload
2. Place the applications or users that define the workloads into their own workload groups (the workloads' processes share the resources WLM makes available to the workload group)
3. Create one ore more SLOs for each workload group

With your workloads isolated in their own workload groups, WLM manages each group's CPU, real memory, and disk bandwidth resources according to the current configuration. WLM allocates resources to your workload groups by automating features of HP's Process Resource Manager (PRM), HP-UX Processor Sets (PSETs), HP-UX Virtual Partitions, and nPartitions that use Instant Capacity (formerly known as iCOD).

With WLM's PSET management, you can dedicate whole processors to your workloads. You can also let WLM adjust the number of processors assigned to a PSET-based workload group in response to SLOs. Similarly, WLM can manage virtual partitions. With one or more workloads running in each virtual partition, WLM can adjust the number of processors in the partitions based on the requirements of the workload groups in those partitions. With WLM's nPartition management, WLM uses Instant Capacity software to deactivate a CPU on one nPartition then activate a CPU on another nPartition where an SLO is in need of resources.

NOTE WLM adjusts only a workload group's CPU allocation in response to SLO performance. Thus, WLM's SLO management is most effective for workloads that are CPU-bound.

Each application must go into a workload group. By default, processes run by root go into the `PRM_SYS` group, and processes run by nonroot users go into the `OTHERS` group. However, you can change the workload group in which a particular user's processes run by adding user records to the WLM configuration file. Furthermore, you can specify the workload groups in which processes run by adding application records to your WLM configuration.

Be aware that application records take precedence over user records.

In addition, you can alter the workload group of an application using the `prmmove` and `prmrun` utilities, which are discussed below.

How do I put an application under WLM control?

WLM provides several methods to place processes in workload groups. It is important to understand these methods as they form the basis of the workload separation.

First, we define the workload groups for the workloads. The following snippet from a WLM configuration file creates three workload groups: `servers_grp`, `apache_grp`, and `OTHERS`. (The `OTHERS` group is a reserved workload group and must have ID 1. If you do not explicitly create this group, WLM will create it for you.)

```
prm {
    groups = OTHERS : 1,
         servers_grp : 2,
         apache_grp : 3;
}
```

Note that each workload group is given a name and a number. Later sections of the WLM configuration file assign resources to the groups. Processes within the groups then share the resources allocated to that group.

With the workload groups defined, the remainder of this section explores how processes can be placed in the workload groups.

Application records: Workload separation by binary name

One mechanism for separating workloads is the apps statement. This statement simply names a particular application binary and the group in which it should be placed. You can specify multiple binary-workload group combinations, separated by commas, in a single apps statement.

In the prm structure below, the apps statement causes the PRM application manager to place any newly running /opt/hpws/apache/bin/httpd executables in the group apache_grp.

```
prm {
    groups = OTHERS : 1,
         servers_grp : 2,
         apache_grp : 3;

    apps = apache_grp : /opt/hpws/apache/bin/httpd;
}
```

NOTE on polling: It is important to realize that the process is not placed in the workload group immediately after starting. Rather, the PRM application manager periodically polls for newly started processes that match records in the apps statement. Each matched process is placed in its designated workload group. For more information, see the section “Management of applications” on page 448.

User records: Workload separation by process owner UID

You can place processes in workload groups according to the UIDs of the process owners. You specify your user-workload group mapping in the `users` statement. Here is an example:

```
prm {
    groups = OTHERS : 1,
             testers : 2,
             coders : 3,
             surfers : 4;

    users = moe : coders surfers,
            curly : coders surfers,
            larry : testers surfers;
}
```

Besides the default `OTHERS` group, this example has three groups of users: `testers`, `coders`, and `surfers`. The user records cause processes started by users moe and curly to be run in group `coders` by default, and user larry's processes to be run in group `testers` by default. Each user is also given permission to run jobs in group `surfers` if they wish, using the `prmrun` or `prmmove` commands mentioned below. Users not belonging to either group are placed in `OTHERS` by default.

As previously noted, application records take precedence over user records.

`prmrun`: Starting a process in a workload group

You can explicitly start processes in particular workload groups using the `prmrun` command. Given the `groups` and `users` statements shown above, user larry running the command

```
# my_really_big_job &
```

would cause his job to be run in group `testers`. However, user larry also has permission to run processes in the group `surfers`. Thus, larry can use the `prmrun` command below

```
# /opt/prm/bin/prmrun -g surfers my_really_big_job &
```

to run his process in the group `surfers`.

prmmove: Moving an existing process to a workload group

Use the prmmove command to move existing processes to a different workload group. If larry from the above example has a job running with process ID (PID) 4065 in the group testers, he could move that process to group surfers by running the command:

```
# /opt/prm/bin/prmmove surfers -p 4065
```

Default: Inheriting workload group of parent process

If a process is not named in an apps statement or a users statement, or has not been started with prmrun or moved with prmmove, it simply starts and runs in the same group as its parent process. So for a setup like this

```
prm {
    groups = OTHERS : 1,
             mygrp : 2;
}
```

if user jdoe has an interactive shell running in group mygrp, any process spawned by that shell process would also run in group mygrp because its parent process was there. Simple inheritance is the mechanism that determines where most processes run, especially for short-lived processes.

How do I determine a goal for my workload?

NOTE Be aware of the resource interaction for each of your workloads. Limiting a workload's memory allocation can also limit its CPU use. For example, if a workload uses memory and CPU in the ratio of 1:2, limiting the workload to 5% of the memory implies that it cannot use more than 10% of the CPU—even if it has a 20% CPU allocation.

To characterize the behavior of a workload, use the following example WLM configuration:

/opt/wlm/examples/wlmconf/manual_entitlement.wlm

Using this configuration, you can directly set an entitlement (allocation) for a workload using the `wlmsend` command. By gradually increasing the workload's allocation with a series of `wlmsend` calls, you can determine how various amounts of CPU affect the workload and its performance with respect to some metric that you may want to use in an SLO for the workload.

In addition, you can compare this research with similar data for the other workloads that will run on the system. This comparison gives you insight into which workloads you can combine (based on their needed CPU) on a single system and still achieve the desired SLOs. Alternatively, if you cannot give a workload its optimal amount of CPU, you will know what kind of performance to expect with a smaller allocation.

Once you know how the workload behaves, you can more easily decide the type of goal, either metric or usage, you want for it. You may even decide to just allocate the workload a fixed amount of CPU or an amount of CPU that varies directly in relation to some metric. For information on the different methods for getting your workload CPU, see the section "SLO TYPES" in the wlm(5) man page.

Similar to the manual_entitlement.wlm configuration, the /opt/wlm/toolkits/weblogic/config/manual_cpucount.wlm configuration allows you to adjust a workload group's CPU allocation with a series of `wlmsend` calls. manual_cpucount.wlm, however, uses a PSET as the basis for a workload group—and changes the group's allocation by one whole CPU at a time.

Metrics for workload management

You provide metric data to WLM so that it can:

- Determine new resource allocations, allowing the workload groups to achieve their SLOs
- Set shares-per-metric allocations
- Enable or disable SLOs

Where do I get metrics?

Sources for metric data include:

- Existing metrics (from log files, scripts, or commands you already have in place)
- ARM-instrumented applications
- GlancePlus
- Oracle databases
- C source code modified to use the WLM API

For more information on these sources, see Chapter 7, “Supplying data to WLM,” on page 237.

How do I feed the metrics to WLM?

Once you have the data, you have a number of options for forwarding that data to WLM:

- Command-line/shell script
- Perl
- stdout of a command
- Instrumenting with the WLM API

Sending data from the command line or from a shell script

You can send data to update a metric's value using the `wlmsend` command on the command line or in a shell script.

The following example WLM configuration, available in its entirety at /opt/wlm/examples/wlmconf/manual_entitlement.wlm as well as on http://www.hp.com/go/wlm, allows you to set a workload group's CPU allocation from the command line using the `wlmsend` command.

```
# Create a workload group called grp1 and define the binaries that make it up.
# In this example, the apache webserver binary httpd is used as the workload.
prm {
    groups = OTHERS : 1, grp1 : 2;
    apps = grp1 : /opt/hpws/apache/bin/httpd;
}

# Have HP-UX WLM manage the CPU allocation such that the workload group grp1
# gets 1 share per metric. That is, setting the metric myapp.desired.allocation
# to 20 results in 20 shares for grp1, setting the metric
# myapp.desired.allocation to 50 results in 50 shares for grp1, and so forth.
slo grp1 {
    pri = 1;
    entity = PRM group grp1;
    cpushares = 1 total per metric myapp.desired.allocation;
}

# Relay a metric from the outside user.
tune myapp.desired.allocation {
    coll_argv = wlmrcvdc;
}
```

This method of getting data to WLM uses the `wlmsend` and `wlmrcvdc` commands. To use this method:

Step 1. Define a workload group and assign a workload to it

In your WLM configuration file, define your workload group in a `prm` structure using the `groups` keyword. Assign a workload to the group using the `apps` keyword. In the example configuration above, the default group for users, called `OTHERS`, is explicitly defined. (WLM creates this group if you do not explicitly define it.) The group `grp1` is also defined, and `httpd` is assigned to it.

```
prm {
    groups = OTHERS : 1, grp1 : 2;
    apps = grp1 : /opt/hpws/apache/bin/httpd;
}
```

Step 2. Set up `wlmrcvdc` to receive data for the metric

In your WLM configuration file, set up a `tune` structure with `wlmrcvdc` as the data collector (value for `coll_argv`). Name the `tune` structure *`metric`*. You will use *`metric`* with `wlmsend` as well.

```
tune metric {
...
   coll_argv = wlmrcvdc;
...
}
```

In our example, *`metric`* is myapp.desired.allocation, and the `tune` structure is:

```
tune myapp.desired.allocation {
    coll_argv = wlmrcvdc;
}
```

Step 3. Use the metric

You must now use the metric in an `slo` structure as part of a `goal` statement, `cpushares` statement, or `condition` statement in your WLM configuration file.

In the example above, the metric myapp.desired.allocation is used in a `cpushares` statement:

```
slo grp1 {
    pri = 1;
    entity = PRM group grp1;
    cpushares = 1 total per metric myapp.desired.allocation;
}
```

Step 4. Activate the configuration

```
# /opt/wlm/bin/wlmd -a config.wlm
```

substituting your configuration file's name for *`config.wlm`*.

Step 5. Send the data using `wlmsend`

When invoking `wlmsend` on the command line or in a script, indicate the *`metric`* being updated and its new *`value`*, which can be an integer or a floating-point value. Invoke `wlmsend` as follows:

```
wlmsend metric value
```

Alternatively, you can pipe the new value to `wlmsend`:

```
cmnd1 | cmnd2 | [... |] wlmsend metric
```

where *cmnd1* and *cmnd2* are commands for gathering or formatting the metric data. Be aware that piping may delay updates due to I/O buffering.

For the example above, `wlmsend` can update the metric on the command line. The following command sets the CPU allocation to 10 shares:

```
# /opt/wlm/bin/wlmsend myapp.desired.allocation 10
```

Sending data from a perl program

You can send data to update a metric’s value using the `wlmsend` command in a perl program.

The following example (/opt/wlm/examples/userguide/reporter.pl) shows how to feed metrics to WLM using a perl program. The script uses the `wlmsend` command:

```
#!/opt/perl/bin/perl -w
#
# Name:
#       reporter.pl
#
# Version information:
#
#       (C) Copyright 2002-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.5 $

# This script uses the ps command to see if the scripts loop2.pl and
# loop3.pl are running. If a script is running, the wlmsend command sends a
# value of 1 for the script; otherwise, it sends a value of 0.

# Here is an infinite loop

while (1) {
    $loop2_running = `env UNIX95=  ps -C loop2.pl | wc -l`;
    --$loop2_running; # Line count is one more due to header

    $loop3_running = `env UNIX95=  ps -C loop3.pl | wc -l`;
    --$loop3_running; # Line count is one more due to header

    # The following wlmsend commands set the values for the metrics
    # loop2_active and loop3_active to $loop2_running and
    # $loop3_running respectively. WLM picks up these values because of
    # the tune structures named for the metrics in the file
    # condition.wlm.

    system "/opt/wlm/bin/wlmsend loop2_active $loop2_running";
    system "/opt/wlm/bin/wlmsend loop3_active $loop3_running";

    sleep(60); # Wait for 60 seconds
}
```

The reporter.pl file is also available on http://www.hp.com/go/wlm.

To implement feeding of metrics from a perl program:

Step 1. Define a workload group and assign a workload to it

In your WLM configuration file, define your workload group in a `prm` structure using the `groups` keyword. Assign a workload to the group using the `apps` keyword.

The example configuration file that works with reporter.pl is /opt/wlm/examples/userguide/condition.wlm. This configuration defines two groups `g2` and `g3` and assigns two perl programs to the groups:

```
prm {
    groups = g2 : 2,
             g3 : 3;
    apps = g2 : /opt/perl/bin/perl loop2.pl,
           g3 : /opt/perl/bin/perl loop3.pl;
    gmincpu = g2 : 5, g3 : 5;
    gmaxcpu = g2 : 30, g3 : 60;
}
```

In this particular case, to ensure WLM places the perl programs in their assigned workload groups, you need to add “/opt/perl/bin/perl” (without the quotes) to your /opt/prm/shells file.

The configuration also uses the `gmincpu` and `gmaxcpu` keywords to place lower and upper bounds on the amount of CPU the workload groups can be allocated.

Step 2. Set up `wlmrcvdc` to receive data

In your WLM configuration file, set up a `tune` structure with `wlmrcvdc` as the data collector (value for `coll_argv`). Name the `tune` structure *`metric`*. You will use *`metric`* with `wlmsend` as well.

```
tune metric {
...
   coll_argv = wlmrcvdc;
...
}
```

In reporter.pl, the values for *`metric`* are `loop2_active` and `loop3_active`.

The corresponding `tune` structure in condition.wlm for `loop2_active` is:

```
tune loop2_active {
    coll_argv = wlmrcvdc;
}
```

Step 3. Use the metric

You must now use the metric in an `slo` structure as part of a `goal` statement, `cpushares` statement, or `condition` statement in your WLM configuration file. The example configuration condition.wlm uses the metric in a `condition` statement:

```
slo test2 {
    pri = 1;
    cpushares = 15 total;
    entity = PRM group g2;
    condition = metric loop2_active;
}
```

Step 4. Activate the configuration

```
# /opt/wlm/bin/wlmd -a config.wlm
```

substituting your configuration file's name for *`config.wlm`*.

Step 5. Send the data using `wlmsend` in your perl program

Use `wlmsend` in your perl program. When invoking `wlmsend`, indicate the *`metric`* being updated and its new *`value`*, which can be an integer or a floating-point value. Invoke `wlmsend` in a loop as follows:

```
system "/opt/wlm/bin/wlmsend $metric $value";
```

or an `open` statement with the `print` statement in a loop:

```
open(WLM, "|/opt/wlm/bin/wlmsend $metric");
```

```
print WLM "$value\n";
```

As we saw above, reporter.pl uses the first method to send data:

```
system "/opt/wlm/bin/wlmsend loop2_active $loop2_running";
```

If you would like WLM to take advantage of new data more quickly:

- Set `wlm_interval` to a smaller value in your WLM configuration file (The default interval is 60 seconds.)
- Decrease the `sleep` argument in reporter.pl

Using stdout of a command for values of a metric

If you have a command or script that displays values for a metric on stdout, you can use `wlmrcvdc` to send the values to WLM.

Assume you have a program called ordercounter. You could then set up a `tune` structure similar to the one below in your WLM configuration:

```
tune order_cnt {
    coll_argv = wlmrcvdc /opt/books/ordercounter;
}
```

The `wlmrcvdc` command forwards any output from ordercounter to WLM as values for the metric order_cnt.

To use the stdout of a command for WLM metrics:

Step 1. Define a workload group and assign a workload to it

In your WLM configuration file, define your workload group in a `prm` structure using the `groups` keyword. Assign a workload to the group using the `apps` keyword.

The `prm` structure below defines group `grp1` and assigns one workload to it:

```
prm {
    groups = grp1 : 2;
    apps = grp1 : /opt/books/orderprocessor;
}
```

Step 2. Set up `wlmrcvdc` to receive data

In your WLM configuration file, set up a `tune` structure with `wlmrcvdc` as the data collector (value for `coll_argv`). Name the `tune` structure *`metric`*. Given an application named *`command`* that prints the metric data on stdout, place *`command`* as an argument to `wlmrcvdc`.

In the configuration file:

```
tune metric {
...
   coll_argv = wlmrcvdc command;
...
}
```

When the configuration is activated, WLM automatically starts the application given by *command*.

As seen at the beginning of this section, a `tune` structure for the metric order_cnt might look like the following example:

```
tune order_cnt {
    coll_argv = wlmrcvdc /opt/books/ordercounter;
}
```

`ordercounter` should run continuously in order to keep updating the order_cnt metric.

Step 3. Use the metric

You must now use the metric in an `slo` structure as part of a `goal` statement, `cpushares` statement, or `condition` statement in your WLM configuration file.

The following `slo` structure uses the order_cnt metric to request two CPU shares for each unit of order_cnt. Thus, if order_cnt is equal to 1 for 100 orders, 2 for 200 orders, and so on, an order_cnt of 10 would result in a request of 20 CPU shares for the order-processing workload to handle the 1000 orders. The `slo` structure is:

```
slo orders {
   pri = 1;
   entity = PRM group grp1;
   cpushares = 2 total per metric order_cnt;
}
```

Step 4. Activate the configuration

```
# /opt/wlm/bin/wlmd -a config.wlm
```

substituting your configuration file's name for *config.wlm*.

`wlmrcvdc` now updates *metric* (order_cnt in this case) with the values from *command*'s stdout.

WLM API

If you are writing a data collector in the C programming language, you can use the WLM API to communicate metric data directly to WLM.

After creating your data collector with the WLM API (described in the section “Sending data from a collector written in C” on page 257), set up your WLM configuration file and activate it as follows:

Step 1. Define a workload group and assign a workload to it

In your WLM configuration file, define your workload group in a `prm` structure using the `groups` keyword. Assign a workload to the group using the `apps` keyword.

The following example, which comes from the example configuration file /opt/wlm/examples/wlmconf/metric_condition.wlm, explicitly defines the default group for users, called `OTHERS`. (WLM creates this group if you do not explicitly define it.) The group `jobs` is also defined, and the `submit` application is assigned to it.

```
prm {
    groups = OTHERS : 1,
             jobs : 2;
    apps =   jobs: /opt/batch/bin/submit;
}
```

Step 2. Set up the `tune` structure to receive data

In your WLM configuration file, set up a `tune` structure with the *`program`* that uses the WLM API as the data collector (value for `coll_argv`). Name the `tune` structure *`metric`*, which matches the name of the metric used in the WLM API call `wlm_mon_attach()`.

In the configuration file, set up the `tune` structure:

```
tune metric {
...
   coll_argv = program;
...
}
```

WLM automatically starts the application given by *`program`*.

The `tune` structure from the example metric_condition.wlm is:

```
tune avg_completion_time {
   coll_argv = /opt/batch/metrics/job_complete_time -minutes;
}
```

The *`metric`* is avg_completion_time. The *`program`*, which includes WLM API calls, is /opt/batch/metrics/job_complete_time.

NOTE The stdout and stderr for each application started by WLM are discarded. However, using the `coll_stderr` tunable, you can redirect stderr to a file of your choosing. For more information, see “Capturing your collectors’ stderr (optional)” on page 209.

Step 3. Use the metric

You must now use the metric in an `slo` structure as part of a `goal` statement, `cpushares` statement, or `condition` statement in your WLM configuration file.

The avg_completion_time metric is used below in a `goal` statement:

```
slo job_processing {
   pri = 1;
   mincpu = 40;
   maxcpu = 90;
   entity = PRM group jobs;
   goal = metric avg_completion_time < 10;
   condition = metric number_of_active_jobs > 0;
}
```

The metric in the `condition` statement is not mentioned here; however, you can read more about it in the metric_condition.wlm file.

Step 4. Activate the configuration

```
# /opt/wlm/bin/wlmd -a config.wlm
```

substituting your configuration file’s name for *`config.wlm`*.

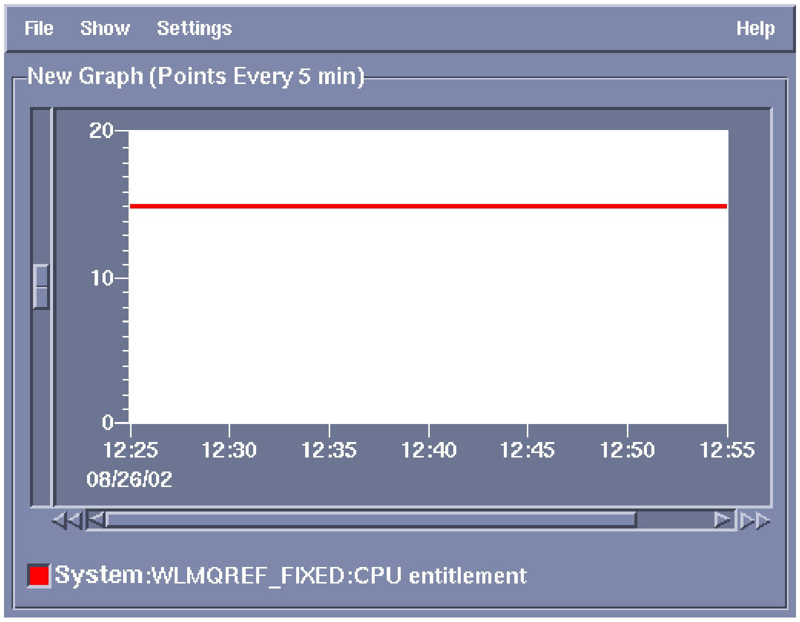
What are some common WLM tasks?

WLM is a powerful tool that allows you to manage your systems in numerous ways. The sections below explain some of the more common tasks that WLM can do for you.

Providing a fixed amount of CPU

WLM allows you to give a workload group a fixed amount of CPU.

In the following graphic (created using the HP PerfView product), the workload receives a constant 15 CPU shares.



WLM allows you to allocate a fixed amount of CPU using:

- portions of processors
- whole processors (processor sets)

Portions of processors

One method for providing a fixed amount of CPU is to set up an SLO to give a workload group a portion of each available processor. To set up this type of SLO:

Step 1. Define the workload group and assign a workload to it

In your WLM configuration file, define your workload group in a `prm` structure using the `groups` keyword. Assign a workload to the group using the `apps` keyword.

The following example defines a group named `sales`.

```
prm {
    groups = sales : 2;

    apps = sales : /opt/sales/bin/sales_monitor;
}
```

The `sales_monitor` application is placed in the `sales` workload group using the `apps` statement. For other methods on placing a workload in a certain group, see "How do I put an application under WLM control?" on page 67.

Step 2. Define a fixed-allocation SLO for the workload group

In the `slo` structure, use the `cpushares` keyword with `total` to request CPU for the workload group. The following SLO requests 15 CPU shares for the workload group `sales`. Based on SLO priorities and available CPU, WLM attempts to meet the request.

```
slo fixed_allocation_example {
    pri = 2;
    entity = PRM group sales;
    cpushares = 15 total;
}
```

Step 3. Activate the configuration

```
# /opt/wlm/bin/wlmd -a config.wlm
```

substituting your configuration file's name for *`config.wlm`*.

Whole processors (processor sets)

Another method for providing a workload group with a fixed amount of CPU is to define the group based on a processor set, or PSET. The processor sets feature is available for HP-UX 11i v1 (B.11.11) and can be downloaded from http://software.hp.com free of charge. Processor sets are included with HP-UX 11i v2 (B.11.23) and later.

As in the previous case, the workload requires a constant amount of CPU. Placing the workload in a workload group based on a PSET, the group has sole access to the processors in the PSET.

To place an application in a workload group based on a PSET:

Step 1. Define the workload group based on a processor set and assign a workload to it

In your WLM configuration file, define your workload group in a `prm` structure using the `groups` keyword. Assign a workload to the group using the `apps` keyword.

The following example shows the `sales` group as a PSET that is assigned two CPUs through a `gmincpu` statement. (When using `gmincpu` for a group based on a PSET, 100 represents a single processor. Thus, 200 represents two processors.) The application sales_monitor is assigned to the group:

```
prm {
    groups = sales : PSET;

    apps = sales : /opt/sales/bin/sales_monitor;

    gmincpu = sales : 200;
}
```

Step 2. Activate the configuration

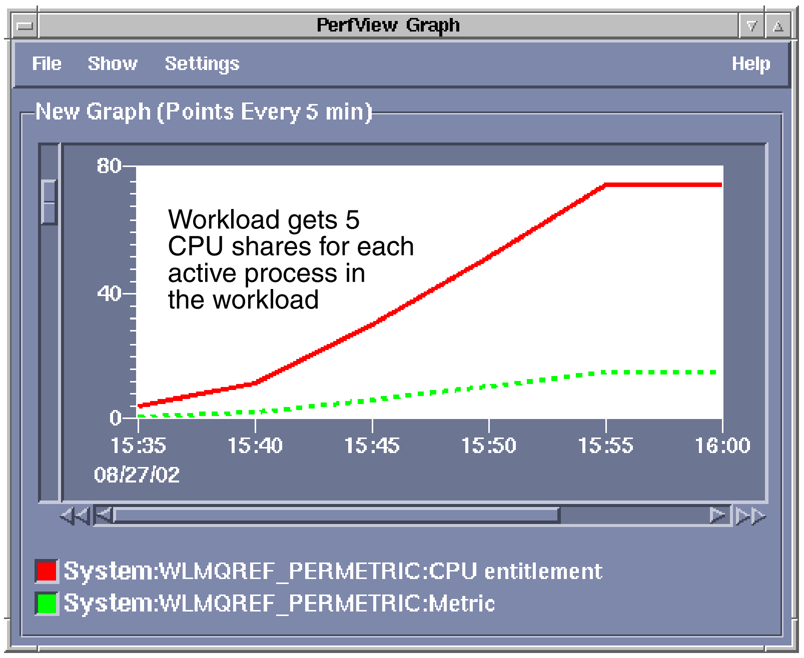
```
# /opt/wlm/bin/wlmd -a config.wlm
```

substituting your configuration file’s name for *`config.wlm`*.

Providing CPU in proportion to a metric

To adjust a workload's CPU allocation in step with a given metric, such as number of processes in the workload, users connected to a database, or any other metric, use the `cpushares` keyword with `per metric` in an `slo` structure.

In the figure below, the workload receives 5 CPU shares for each active process in the workload group. As the number of processes increases, the number of CPU shares increases.



To set up a workload group with an allocation that varies in proportion to a metric:

Step 1. Define the workload group and assign a workload to it

In your WLM configuration file, define your workload group in a `prm` structure using the `groups` keyword. Assign a workload to the group using the `apps` keyword.

In this example, the focus is the `sales` group, which will get a varying amount of CPU based on a metric:

```
prm {
    groups = sales : 2;

    apps = sales : /opt/sales/bin/sales_monitor;
}
```

Step 2. Define the SLO

The SLO in your WLM configuration file must specify a priority (`pri`) for the SLO, the workload group to which the SLO applies (`entity`), and the `cpushares` statement to request CPU in proportion to a metric.

The following `slo` structure shows the `cpushares` statement. This SLO requests 5 CPU shares for the `sales` group for each process in the application—as given by the metric application_procs.

```
slo per_metric_example {
    pri = 1;
    mincpu = 25;
    maxcpu = 50;
    entity = PRM group sales;
    cpushares = 5 total per metric application_procs;
}
```

However, if application_procs is less than five or greater than ten, the values for the optional keywords `mincpu` and `maxcpu` will force the request for additional CPU shares to be between 25 and 50.

Sources for a metric include data available through GlancePlus, SNMP, and any other metrics you may already have available.

Step 3. Set up WLM to take values for the metric

The simplest method for getting a metric to WLM is through the WLM data collector `wlmrcvdc`.

To receive a value for the metric application_procs, set up a `tune` structure. The following `tune` structure takes advantage of `glance_prm`, one of the commands that WLM provides to simplify pulling data from the optional HP product GlancePlus. In this case, `glance_prm` pulls the APP_ACTIVE_PROC metric for the `sales` workload group:

```
tune application_procs {
    coll_argv = wlmrcvdc glance_prm APP_ACTIVE_PROC sales;
}
```

As a result of this structure, each time the `glance_prm` command prints a value on standard out, `wlmrcvdc` sends the value to WLM for use in changing the CPU allocation for the `sales` group.

For information on other ways to use `wlmrcvdc`, see the wlmrcvdc(1M) man page.

Step 4. Activate the configuration

```
# /opt/wlm/bin/wlmd -a config.wlm
```

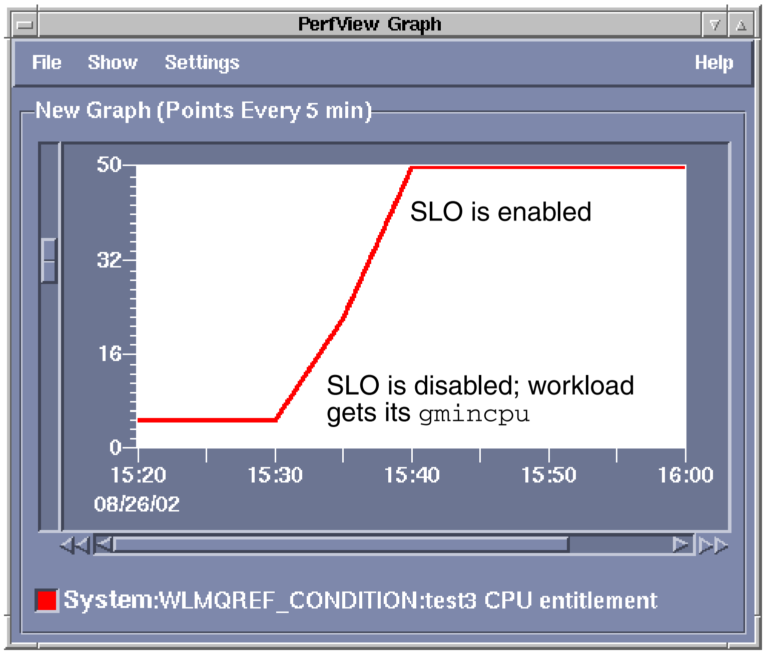
substituting your configuration file’s name for *`config.wlm`*.

Providing CPU for a given time period

For workloads that are only needed for a certain period of time, whether it is once a day, week, or any other period, WLM provides the `condition` keyword. This keyword is placed in an `slo` structure and enables the SLO when the `condition` statement is true.

By default, when the SLO is not enabled, its workload group gets its `gmincpu` value. If this value is not set, the workload group gets the default 1% of the CPU. (Setting the `transient_groups` keyword to 1 changes this behavior. For more information, see “Temporarily removing groups with inactive SLOs (optional)” on page 207.)

The figure below initially shows a workload group with a disabled SLO. With the disabled SLO, the group is given its `gmincpu` value, which in this case is 5 shares. The SLO is enabled at 15:38, when an application is started, and the workload group eventually receives 50 CPU shares.


PerfView Graph
File
Show
Settings
Help
New Graph (Points Every 5 min)
50
32
16
0
15:20
08/26/02
15:30
15:40
15:50
16:00
SLO is enabled
SLO is disabled; workload gets its gmincpu
🟥 System:WLMQREF_CONDITION:test3 CPU entitlement

To provide a workload group with CPU on a schedule:

Step 1. Define the workload group and assign a workload to it

In your WLM configuration file, define your workload group in a `prm` structure using the `groups` keyword. Assign a workload to the group using the `apps` keyword.

In this example, the `sales` group is the group of interest again.

```
prm {
    groups = sales : 2;

    apps = sales : /opt/sales/bin/sales_monitor;
}
```

Step 2. Define the SLO

The SLO in your WLM configuration file must specify a priority (`pri`) for the SLO, the workload group to which the SLO applies (`entity`), and either a `cpushares` statement or a `goal` statement so that WLM grants the SLO's workload group some CPU. The `condition` keyword determines when the SLO is enabled or disabled.

The following `slo` structure shows a fixed-allocation SLO for the `sales` group. When enabled, this SLO requests 25 CPU shares for the `sales` group. Based on the `condition` statement, the SLO is enabled between 8pm and 11pm every day.

```
slo condition_example {
    pri = 1;
    entity = PRM group sales;
    cpushares = 25 total;
    condition = 20:00 - 22:59;
}
```

Whenever the `condition` statement is false, the SLO is disabled and does not make any CPU requests for the `sales` group. When a group has no enabled SLOs, it gets its `gmincpu` value (if set); otherwise, it gets no more than a 1% CPU allocation.

You can use times, dates, or metrics to enable or disable an SLO.

For information on the syntax for the `condition` keyword, see the wlmconf(4) man page.

Step 3. Activate the configuration

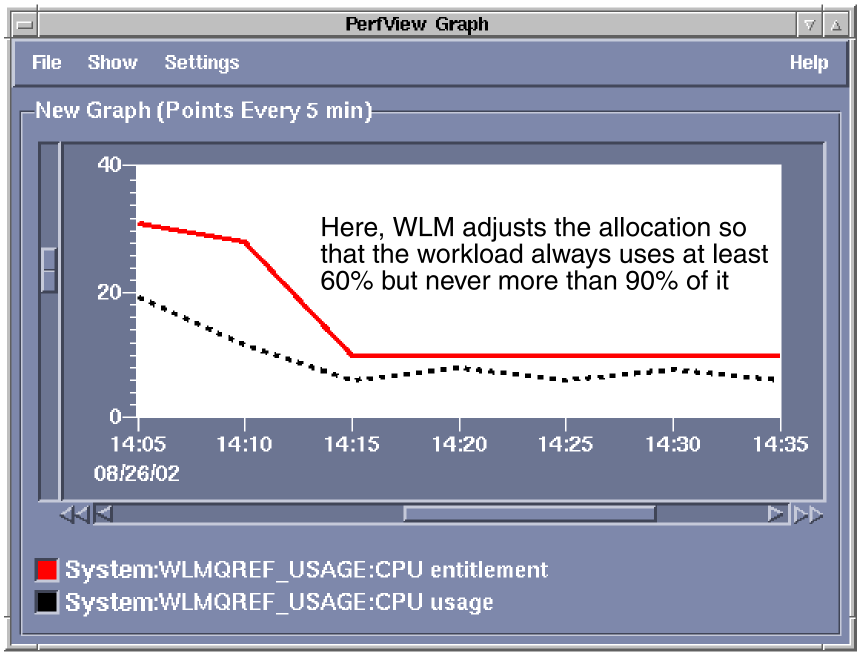
```
# /opt/wlm/bin/wlmd -a config.wlm
```

substituting your configuration file's name for *`config.wlm`*.

Providing CPU as it is needed

To ensure a workload gets the CPU it needs—without preventing other workloads access to unused CPU resources—WLM allows you to define usage goals. A major advantage with usage goals is that you do not need to track and feed a metric to WLM.

The following figure shows a workload that is initially using 20 CPU shares. The usage goal for the workload is to use between 60% and 90% of its allocation. With its beginning allocation at 32 shares, the workload is using 20/32 or 62.5% of its allocation. Later, the workload settles to using around 8 or 9 shares. WLM responds by reducing its allocation to 10 shares, still meeting the usage goal as long as usage stays between 6 and 9 shares.


PerfView Graph
File
Show
Settings
Help
New Graph (Points Every 5 min)
40
20
0
14:05
08/26/02
14:10
14:15
14:20
14:25
14:30
14:35
Here, WLM adjusts the allocation so that the workload always uses at least 60% but never more than 90% of it
🟥 System:WLMQREF_USAGE:CPU entitlement
⬛ System:WLMQREF_USAGE:CPU usage

With a usage goal, you indicate for a workload how much of its CPU allocation it should use. Then, a workload’s CPU allocation is reduced if it is not consuming enough of its current allocation, allowing other workloads to consume more CPU if needed. Similarly, if the workload is using a high percentage of its allocation, it is granted more CPU.

To define a usage goal for a workload group:

Step 1. Define the workload group and assign a workload to it

In your WLM configuration file, define your workload group in a `prm` structure using the `groups` keyword. Assign a workload to the group using the `apps` keyword.

With this example, the `sales` group is the group of interest.

```
prm {
    groups = sales : 2;

    apps = sales : /opt/sales/bin/sales_monitor;
}
```

Step 2. Define the SLO

The SLO in your WLM configuration file must specify a priority (`pri`) for the SLO, the workload group to which the SLO applies (`entity`), and a usage `goal` statement.

The following `slo` structure for the `sales` group shows a usage `goal` statement.

```
slo usage_example {
    pri = 1;
    mincpu = 20;
    maxcpu = 60;
    entity = PRM group sales;
    goal = usage _CPU 80 90;
}
```

With usage goals, WLM adjusts the amount of CPU it grants a workload group so that the group uses between 50% and 75% of its allocated CPU. In the example above though, with the values of 80 and 90 in the `goal` statement, WLM would try to increase or decrease the workload group’s CPU allocation until the group is using between 80% and 90% of the allocated share of CPU. However, in attempting to meet the usage goal, WLM’s new CPU allocations to the workload group will typically be within the `mincpu`/`maxcpu` range.

Step 3. Set your interval to 5

With a usage goal, you should typically set `wlm_interval` to 5.

```
tune {
    wlm_interval = 5;
}
```

Step 4. Activate the configuration

```
# /opt/wlm/bin/wlmd -a config.wlm
```

substituting your configuration file's name for *`config.wlm`*.

What else can WLM do?

Run in passive mode to verify operation

WLM provides a passive mode that allows you to see how WLM will approximately respond to a given configuration—without putting WLM in charge of your system's resources. Using this mode, you can check that your configuration behaves as expected—with minimal effect on the system. Besides being useful in understanding and experimenting with WLM, passive mode can be helpful in capacity-planning activities.

Auditing and billing

When you activate a WLM configuration or a WLM global arbiter configuration, you have the option of generating audit data. You can use the audit data files directly or view them using the `wlmaudit` command. For more information, see the wlmaudit(1M), wlmd(1M), and wlmpard(1M) man pages.

Automatically resize processor sets (PSETs)

With multi-processor systems, you can group processors together to form processor sets, also known as PSETs. By creating PSETs, you isolate CPU resources for users or applications.

WLM allows you to define workload groups based on PSETs. If you then specify SLOs for these workload groups, WLM automatically adjusts the number of processors in the PSETs based on progress toward the SLOs.

Automatically manage SLOs globally, across multiple virtual systems

HP-UX Virtual Partitions are software-based virtual systems, each running its own instance of the HP-UX operating system. WLM can move processors between these virtual systems to better achieve the SLOs you define using WLM within each virtual system.

For more information on this feature, see “Managing SLOs across partitions” on page 283 or the wlmpard(1M) and wlmparconf(4) man pages.

Automatically manage SLOs globally, across multiple nPartitions

nPartitions (nPars) are hardware-based partitions, each running its own instance of the HP-UX operating system. Using Instant Capacity (formerly known as iCOD) software, WLM can simulate the movement of processors between nPartitions—thereby managing your SLOs across nPartitions. (The processor movement is simulated by deactivating a processor on one nPartition and activating a processor on another nPar.)

For more information on this feature, see “Managing SLOs across partitions” on page 283 or the wlmpard(1M) and wlmparconf(4) man pages.

Optimize use of Temporary Instant Capacity and Pay Per Use resources

With Temporary Instant Capacity and Pay Per Use (PPU) systems, you pay only for the amount of CPU you use. Combining Temporary Instant Capacity (v6 or v7) or PPU v4 with WLM and its `utilitypri` keyword, you can optimize the resource utilization to the minimum needed to meet your SLOs, thereby reducing your costs.

For more information on this feature, see the wlmparconf(4) and wlmpard(1M) man pages.

Integrate with various third-party products

HP has developed a number of toolkits to assist you in quickly and effectively deploying WLM for use with your key applications. The freely available WLM Toolkits product consists of several toolkits, each with example configuration files. Toolkits are currently available for:

- Oracle® Databases
- HP-UX Apache-based Web Server software
- BEA WebLogic Server™
- SNMP agents
- SAS® software

In addition, WLM Toolkits provides a duration management toolkit.

WLM Toolkits are included with WLM. They are also freely available on http://software.hp.com.

For more information on the toolkits, see the *HP-UX Workload Manager Toolkits User's Guide* on http://docs.hp.com/hpux/netsys/ or the wlmtk(5) man page.

What status information does WLM provide?

WLM has three log files to keep you informed:

- /var/opt/wlm/msglog

  WLM prints errors and warnings about configuration file syntax on stderr. For messages about WLM's ongoing operations, WLM logs error and informational messages to /var/opt/wlm/msglog, as shown below:

```
05/07/02 14:12:44 [I] (p13931) Ready to forward data for metric
"m_apache_access_2min"
05/07/02 14:12:44 [I] (p13931) Using "/door00/wlm_cfg/apache_access.pl"
05/07/02 14:12:44 [I] (p13932) Ready to forward data for metric
"m_list.cgi_procs"
05/07/02 14:12:44 [I] (p13932) Using "/door00/wlm_cfg/proc_count.sh"
05/07/02 14:12:44 [I] (p13930) Ready to forward data for metric
"m_apache_access_10min"
05/07/02 14:12:44 [I] (p13930) Using "/door00/wlm_cfg/apache_access.pl"
05/07/02 14:12:44 [I] (p13929) Ready to forward data for metric "m_nightly_procs"
05/07/02 14:12:44 [I] (p13929) Using "/opt/wlm/lbin/coll/glance_prm"
05/07/02 14:12:44 [I] (p13921) wlmd 13921 starting
05/31/02 03:42:10 [I] (p13921) /var/opt/wlm/wlmdstats has been renamed to
/var/opt/wlm/wlmdstats.old.
```

- /var/opt/wlm/wlmdstats

  WLM can optionally produce a log file that includes data useful for verifying WLM's management or for fine-tuning your WLM configuration. This log file, /var/opt/wlm/wlmdstats, is created when you start the WLM daemon with the `-l` option:

  `# /opt/wlm/bin/wlmd -a` ***`config.wlm`*** **`-l slo`**

  For information on the `-l` option, see the wlmd(1M) man page.

  Here are some lines from a wlmdstats file:

```
1024328341 SLO=s_nice_xtra_min sloactive=1 goaltype=nogoal goal=nan goalsatis=1
met=nan metfresh=0 mementitl=0 cpuentitl=20 clipped=0 controlling=0
1024328341 SLO=s_team_playground_xtra_min sloactive=1 goaltype=nogoal goal=nan
goalsatis=1 met=nan metfresh=0 mementitl=0 cpuentitl=5 clipped=0 controlling=1
```

- /var/opt/wlm/wlmpardstats

  WLM’s global arbiter, used in managing SLOs across virtual partitions, can optionally produce a statistics log file. This log file, /var/opt/wlm/wlmpardstats, is created when you start the WLM global arbiter daemon with the `-l` option:

  ```
  # /opt/wlm/bin/wlmpard -a config.wlm -l vpar
  ```

  For information on the `-l` option, see the wlmpard(1M) man page.

How do I monitor WLM?

For monitoring WLM, there are several methods, as described below.

`ps [-P] [-R workload_group]`

The `ps` command has options that are specific to PRM, which WLM uses to define workload groups:

- `-P`

  Adds the column `PRMID`, showing the workload group for each process.

  ```
  # ps -P
  ```

  ```
  PRMID     PID TTY       TIME COMMAND
     g3    6793 ttyp1     1:52 loop.pl
     g3    6463 ttyp1     7:02 loop3.pl
     g2    6462 ttyp1     4:34 loop2.pl
  ```

- `-R` *workload_group*

  Lists only the processes in the group named by *workload_group*. Here is output showing processes in the workload group `g3`:

  ```
  # ps -R g3
  ```

  ```
  PID TTY       TIME COMMAND
  6793 ttyp1     1:29 loop.pl
  6463 ttyp1     6:41 loop3.pl
  ```

### wlminfo

The `wlminfo` command, available in /opt/wlm/bin/, displays information about SLOs, metrics, workload groups, virtual partitions or nPartitions, and the current host. Here is example `group` info:

# wlminfo group

```
Workload Group    PRMID  CPU Shares  CPU Util  Mem Shares  State
OTHERS                1       65.00      0.00        0.00  ON
g2                    2       15.00      0.00        0.00  ON
g3                    3       20.00      0.00        0.00  ON
```

wlmgui

The `wlmgui` command, available in /opt/wlm/bin/, graphically displays information about SLOs, metrics, workload groups, partitions, and the current host.

prmmonitor

The `prmmonitor` command, available in /opt/prm/bin/, displays current configuration and resource usage information:

```
PRM configured from file:  /var/opt/wlm/tmp/wmprmBAAa06128
File last modified:        Thu Aug 29 08:35:23 2002

HP-UX samples B.11.00 A 9000/889    08/29/02

Thu Aug 29 08:43:16 2002    Sample:  1 second
CPU scheduler state:  Enabled, CPU cap ON
                                                 CPU      CPU
PRM Group                       PRMID    Entitlement     Used
____________________________________________________________
OTHERS                              1         65.00%    0.00%
g2                                  2         15.00%    0.00%
g3                                  3         20.00%    0.00%

PRM application manager state:  Enabled  (polling interval: 30
seconds)
```

prmlist

The `prmlist` command, available in /opt/prm/bin, displays current CPU allocations plus user and application configuration information:

```
PRM configured from file:  /var/opt/wlm/tmp/wmprmBAAa06128
File last modified:        Thu Aug 29 08:35:23 2002

PRM Group                      PRMID    CPU Entitlement
-------------------------------------------------------
OTHERS                             1             65.00%
g2                                 2             15.00%
g3                                 3             20.00%

PRM User                       Initial Group                 Alternate Group(s)
--------------------------------------------------------------------------------
root                           (PRM_SYS)

PRM Application                Assigned Group                Alternate Name(s)
--------------------------------------------------------------------------------
/opt/perl/bin/perl             g2                            loop2.pl
/opt/perl/bin/perl             g3                            loop3.pl
```

GlancePlus

The optional HP product GlancePlus allows you to display resource allocations for the workload groups as well as list processes for individual workload groups.

wlm_watch.cgi

This CGI script, available in /opt/wlm/toolkits/apache/bin/, allows you to monitor WLM using a web browser interface to `prmmonitor`, `prmlist`, and other monitoring tools.

For information on setting up this script, see the wlm_watch(1M) man page.

Status and message logs

WLM provides the following logs:

- /var/opt/wlm/msglog
- /var/opt/wlm/wlmdstats
- /var/opt/wlm/wlmpardstats

For information on these logs, including sample output, see the section "What status information does WLM provide?" on page 96.

EMS

EMS (Event Monitoring Service) polls various system resources and sends messages when events occur. WLM's `wlmemsmon` command provides numerous resources for event monitoring. For information on these resources, see the wlmemsmon(1M) man page.

4 WLM concepts

This chapter discusses how workloads can be managed through service-level objectives. It also discusses WLM's two types of SLOs: shares-based and goal-based. With goal-based SLOs, WLM automatically allocates system resources to WLM workload groups based on the varying demands of the workloads the groups contain.

These topics are covered in the following sections:

- How WLM works
- Shares-based SLOs vs goal-based SLOs
- How WLM gets application data
- SLO violations
- How a workload is managed (controllers)
- How conflicting SLOs are resolved (arbiter)
- Allocating CPU: The rising tide model
- Example of WLM in use

How WLM works

The following tasks outline how WLM works:

1. Sets initial resource allocations.

   WLM sets resource allocations for your workload groups based on metrics for the workloads. When you first start WLM however, there are no metrics for the workloads. So, WLM sets these initial allocations, in terms of shares, as follows:

   CPU shares: Each workload group gets 1/*n* of the total CPU, where *n* is the number of workload groups

   Memory shares: Each workload group gets the default minimum memory allocation (although the group can use more if needed)

   Disk bandwidth shares: Are set as indicated in the configuration file (if they are specified at all)

   CPU shares are either relative or absolute. With relative shares, a workload group’s number of shares relative to the total number of shares for all the workload groups determines the group’s allocation. For example, if group `A` has 50 shares and the other groups in the configuration have 150 shares, group `A` has 50/200 or 25% of the CPU. With absolute CPU shares, 100 shares represent one CPU. So, you only need to know how many shares a group has to determine its allocation. For example, assume group `A` has 50 shares again. This is an allocation of half of a CPU.

2. Accepts or calculates performance data.

   For metric goals and shares-per-metric allocations, WLM accepts metrics from data collectors. For usage goals and fixed allocations, WLM calculates the metrics itself.

3. Compares performance data (reported performance) to stated goals (desired performance) for each active goal-based SLO.

4. Determines new CPU allocations so that reported performance converges on desired performance; also determines CPU allocations to satisfy requests for shares-per-metric allocations and fixed allocations.

5. Arbitrates between workloads when CPU resources are insufficient to meet the needs of all workloads.

   When CPU resources are not sufficient, certain workloads necessarily will not be able to reach their desired performance levels. In these cases, WLM allocates resources to the associated workload groups based on their SLOs' assigned priorities—allowing the higher-priority SLOs to better meet their goals at the expense of the lower-priority SLOs not meeting their goals.

6. Determines memory allocations based on whether any workloads have become active or gone inactive.

7. Sends request to the WLM global arbiter to increase or decrease the number of CPUs in the current partition as appropriate.

   You can use WLM within and across:

   - Virtual partitions
   - nPartitions that use Instant Capacity software (formerly known as iCOD software)

8. Assigns new CPU and memory shares.

9. Writes to log files.

   If you enabled statistics logging with the `wlmd -l` option, at the end of each WLM interval, WLM adds data for the interval to the file /var/opt/wlm/wlmdstats.

   Also, the WLM global arbiter writes to its own statistics log file if you enabled logging with the `wlmpard -l` option. This file, /var/opt/wlm/wlmpardstats, is written to at the end of each WLM global arbiter interval.

10. Sets EMS resources for monitoring of SLO compliance.

11. Repeats tasks 2 through 10 the next WLM interval.

12. WLM adds messages, at any time, to its message log /var/opt/wlm/msglog. This log, which WLM automatically creates, informs you of the WLM daemon's operations.

13. WLM and the global arbiter daemon generate audit data files, if enabled (using the `-t` option to `wlmd` and `wlmpard` when you activated your WLM configuration and global arbiter configuration).

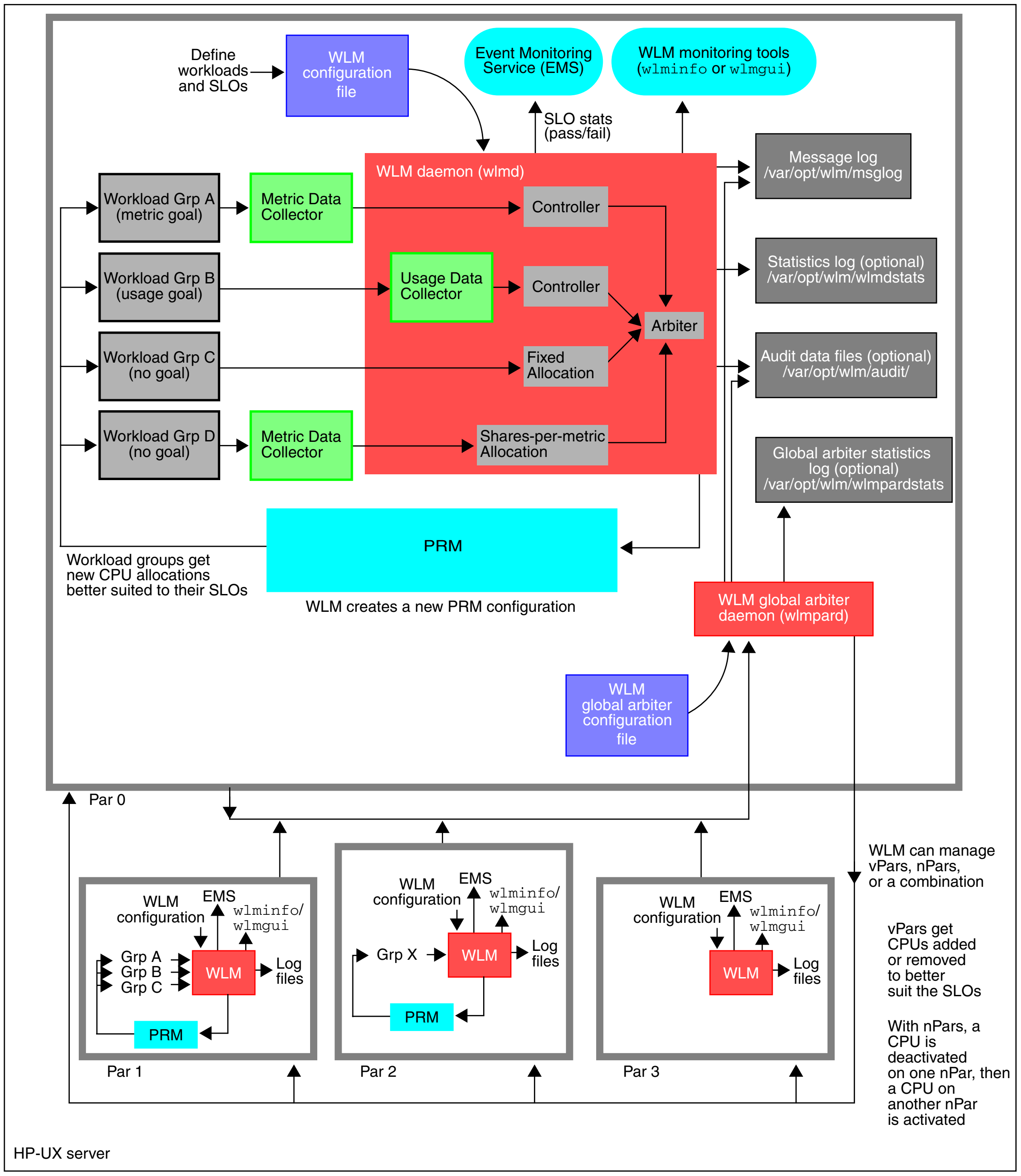
Figure 4-1 on page 105 illustrates how WLM works.

For more information on the components shown in the figure, see the following sections:

- Types of SLOs
  "Shares-based SLOs vs goal-based SLOs" on page 108
- Metric data collectors
  "Supplying data to WLM" on page 237
- WLM daemon
  "`wlmd` command syntax" on page 373
- WLM configuration file
  "Configuring WLM" on page 127
- WLM global arbiter
  "Managing SLOs across partitions" on page 283
- PRM
  "Understanding how PRM manages resources" on page 441
- EMS
  "Monitoring SLO compliance and WLM" on page 357

The following figure provides an overview of WLM functional flow. It shows WLM running in each partition on the system. If you are using WLM on a system without partitions, focus on Par 0 to understand how WLM controls resources on your system to ensure SLOs are met.

Figure 4-1 WLM overview


Define workloads and SLOs
WLM configuration file
Event Monitoring Service (EMS)
WLM monitoring tools (wlminfo or wlmgui)
SLO stats (pass/fail)
WLM daemon (wlmd)
Message log /var/opt/wlm/msglog
Workload Grp A (metric goal)
Metric Data Collector
Controller
Workload Grp B (usage goal)
Usage Data Collector
Controller
Statistics log (optional) /var/opt/wlm/wlmdstats
Arbiter
Workload Grp C (no goal)
Fixed Allocation
Audit data files (optional) /var/opt/wlm/audit/
Workload Grp D (no goal)
Metric Data Collector
Shares-per-metric Allocation
Global arbiter statistics log (optional) /var/opt/wlm/wlmpardstats
PRM
Workload groups get new CPU allocations better suited to their SLOs
WLM creates a new PRM configuration
WLM global arbiter daemon (wlmpard)
WLM global arbiter configuration file
Par 0
WLM configuration
EMS
wlminfo/ wlmgui
Grp A
Grp B
Grp C
WLM
Log files
PRM
Par 1
WLM configuration
EMS
wlminfo/ wlmgui
Grp X
WLM
Log files
PRM
Par 2
WLM configuration
EMS
wlminfo/ wlmgui
WLM
Log files
Par 3
WLM can manage vPars, nPars, or a combination
vPars get CPUs added or removed to better suit the SLOs
With nPars, a CPU is deactivated on one nPar, then a CPU on another nPar is activated
HP-UX server

Referring to Figure 4-1 on page 105, the WLM functional flow is as follows:

- The WLM configuration file specifies the goal-based or shares-based SLOs for each workload group. This file also provides the pathnames for the data collectors. HP-UX WLM reads the configuration file and starts the data collectors.
- With the WLM configuration file activated using the `-t` option to `wlmd`, WLM produces audit data in /var/opt/wlm/audit/.
- For each application with a metric goal, as the application runs, a data collector for that application reports the application's metrics. The measurement, for example, might be transaction response times for an online transaction processing (OLTP) application.
- For each metric goal, WLM creates a controller. The controllers are an internal component of WLM. Each controller receives the metric from the respective data collector. The metric is compared to the goal for that metric to determine if the application is overperforming or underperforming. The controller then requests more CPU shares for a workload group with underperforming applications or fewer CPU shares for a workload group with overperforming applications.
- For each application with a usage goal, WLM creates a controller. Each controller tracks its application's actual CPU usage (utilization of allocated CPU); no user-supplied metrics are required. The controller requests an increase or decrease to the workload group's CPU allocation to bring the utilization percentage within a configurable range.
- For applications without goals, WLM requests CPU based on the CPU allocation request values set in the SLO definitions. These requests could be for fixed allocations or for shares-per-metric allocations, with the metric coming from a data collector.
- The arbiter, an internal module of WLM, collects all the requests for CPU shares. These requests come from controllers or from the SLO definitions in the case of fixed allocations. The arbiter satisfies the requests based on priority. If there are not enough resources for every application to meet its goals, the arbiter satisfies the highest priority requests first. If multiple SLOs at the same priority cannot be satisfied, WLM raises the CPU allocation for each SLO's associated workload group to the same level, or to the SLO's CPU request—whichever is smaller.

- Optionally, WLM determines how much memory to distribute to meet the minimum memory requests and then, if any memory remains, divides it among the groups with active SLOs.
- WLM then creates a new PRM configuration applying the new CPU and optional memory shares. Also, WLM adds data to the statistics log /var/opt/wlm/wlmdstats if enabled through the `wlmd -l` option. The data collectors continue to feed application metrics to WLM, which at intervals calculates new resource allocations and performs any needed PRM reconfiguration.
- If configured for management of partitions, the WLM instance on each partition regularly requests from the WLM global arbiter a certain number of CPUs for its partition.

  The global arbiter adds data to the statistics log /var/opt/wlm/wlmpardstats if enabled through the `wlmpard -l` option.

  With the WLM global arbiter configuration file activated using the `-t` option to `wlmpard`, WLM produces audit data in /var/opt/wlm/audit/.

  With WLM performing cross-partition management, this WLM process flow being described is duplicated in each partition. The WLM global arbiter takes the CPU requests from the WLM instance on each partition. It then uses these requests to decide how to allocate CPUs to the partitions. Next, it adjusts a partition's number of CPUs in an effort to better meet the SLOs in the partition.

NOTE For partitions, you can bypass creating workload groups, treating the partition itself as the workload. Par 3 shows the scenario.

- The WLM monitoring tools `wlminfo` and `wlmgui` allow you to get a variety of types of WLM information.
- The status of the SLOs and information about the performance of WLM are sent to the Event Monitoring Service (EMS). Using an EMS client such as System Administration Manager (SAM), a system administrator can choose from a number of notification methods (such as email, SNMP traps, TCP, UDP, and OPC Messaging) for receiving events of specific interest.

- WLM keeps you up-to-date on the operations of its daemon by updating the message log /var/opt/wlm/msglog.

Shares-based SLOs *vs* goal-based SLOs

WLM supports two types of SLOs:

- Shares-based SLOs

  A shares-based SLO allows you to specify either a fixed allocation of shares or a shares-per-metric allocation for a workload group. A shares-per-metric allocation is of the form "*x* shares of the CPU for each metric *y*".

  A shares-based SLO has a priority. It may also have maximum and minimum CPU request bounds and an explicit shares request. However, it does not have an explicit goal.

- Goal-based SLOs

  A goal-based SLO has a specified performance or usage goal.

  WLM automatically changes CPU allocations for an SLO's associated workload group based on the:

  — Performance or usage (utilization) of the workload in the group

  — SLO's priority

  — Available resources

  A workload's performance is tracked by a data collector and reported to WLM, as explained in the section "How WLM gets application data" on page 109. For an SLO with a usage goal, WLM tracks the data itself.

You can simultaneously use an SLO with a fixed shares allocation and goal-based SLOs on the same workload group.

How WLM gets application data

You use a data collector for each workload group that has a performance goal. You can use one of WLM's provided data collectors or you can make your own. These collectors report their data to WLM on a regular basis. This data updates the values of metrics used in the WLM configuration. Once every interval (the default is 60 seconds), WLM checks the metric values and adjusts resource allocations to better achieve the SLOs, if needed.

For an illustration showing the role of data collectors in WLM operations, see Figure 4-1 on page 105.

For information on writing data collectors, see "Supplying data to WLM" on page 237.

For information on changing the WLM interval, see "Specifying the WLM interval (optional)" on page 203.

SLO violations

- For performance goals:

  An SLO violation occurs when a workload's performance varies from the goal in the wrong direction. Which direction is wrong depends on the goal definition. For example, if the goal is to keep response_time less than 5 seconds, a response_time value of 4.3 seconds varies from the goal in the right direction. However, a response_time value of 5.4 seconds varies in the wrong direction. Similarly, for a goal to have greater than 100 transactions/minute, a reported performance of 80 transactions/minute varies in the wrong direction.

  Regardless of the direction of underperformance or overperformance, WLM adjusts CPU allocations to more closely match the SLO's goal. In the case of an SLO violation, however, WLM also sets EMS resources to alert persons monitoring the system. For more information on EMS resources, see "What EMS resources are available?" on page 367.

  You can also track SLO violations using `wlminfo`.

- For usage goals:

  An SLO violation occurs if the CPU utilization is above the target utilization range. With usage goals, the goal is to keep CPU utilization (CPU used / CPU granted) within a certain range, 50% and 75% by default. When the workload goes above this range, giving it more CPU can bring it back into the range.

  Going below this range is not considered an SLO violation because the situation cannot be improved by providing the workload group with additional CPU resources.

How a workload is managed (controllers)

When a configuration is activated, WLM instantiates a controller for each SLO that has a performance goal or a usage goal. For SLOs with performance goals, controllers receive metric updates in the form of performance data from data collectors. With usage goals, WLM internally tracks the workload group's actual CPU usage versus its CPU allocation. The controllers then determine the CPU allocations to request to better achieve the desired performance or usage goals.

For an illustration showing controllers, see Figure 4-1 on page 105.

For information on specifying performance and usage goals, see "Specifying a goal (optional)" on page 183.

For information on how to tune controllers, see:

- "Tuning a workload group's SLO convergence: `cntl_kp` (optional)" page 212
- "Tuning a workload group's SLO convergence: `cntl_convergence_rate` (optional)" page 216
- "Tuning the goal buffer (optional)" on page 218

How conflicting SLOs are resolved (arbiter)

When CPU resources are insufficient to satisfy all SLO controller requests, WLM must decide how to distribute CPU resources among the workload groups. In these cases, WLM's arbiter considers each controller's prioritized requests for CPU and the associated workload groups' weights to decide the proper allocation of CPU.

For an illustration showing the arbiter, see Figure 4-1 on page 105.

For information on priorities and weights, see "Specifying the priority (required)" on page 176 and "Weighting a group so it gets more CPU (optional)" on page 162.

Allocating CPU: The rising tide model

If all workload groups' demand for CPU can be met with current resources, WLM satisfies that demand. If however, demand exceeds supply, WLM uses the "rising tide" model to allocate CPU: At a given priority, WLM attempts to raise the allocation of the workload group with the lowest CPU allocation to the level of the next lowest allocation. If CPU resources remain, WLM then raises the allocations for those two groups to the level of the third lowest group. This process continues at the current priority until all CPU requests have been met or all resources have been distributed. If requests have been met and resources remain, WLM continues with the next priority.

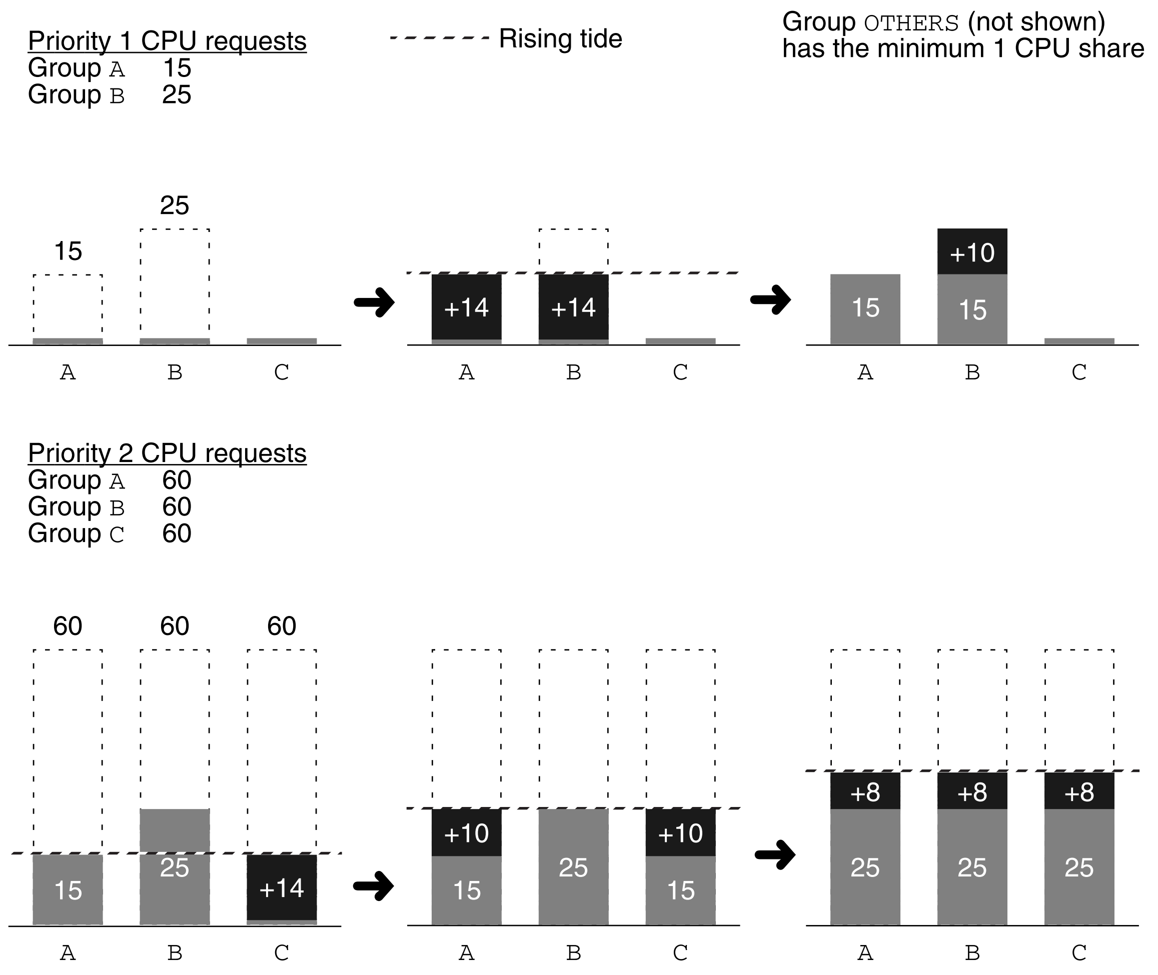
Consider Figure 4-2 on page 113 with groups `A`, `B`, and `C`. All groups start with 1 CPU share. At priority 1, only groups `A` and `B` are requesting CPU. Their requests of 15 and 25 shares are easily satisfied. WLM meets the lowest request first, granting 14 additional shares to both groups. WLM then grants an additional 10 shares to group `B` to meet its higher request. Group `C` and the default group `OTHERS` (not shown) both receive 1 share. After satisfying those CPU requests, 58 shares are left.

At priority 2, groups `A` `B`, and `C` all request 60 shares. These requests, of course, sum to 180 shares—exceeding the available 100 shares. With the 58 remaining shares, WLM can meet only one of the SLOs. With all three SLOs at the same priority, WLM attempts to satisfy them all as much as possible—rather than meet one SLO at the expense of the other two. The strategy of meeting each SLO as much as possible is similar to a rising tide lifting all boats: All workload groups are raised to the same CPU allocation.

Group `C` starts with 1 share. WLM first grants group `C` an additional 14 shares, raising its allocation to match the allocation for group `A`. WLM then adds 10 shares to the allocations for both `A` and `C`, matching that of group `B`. Now each group has 25 shares. The 24 remaining shares are distributed evenly among the three groups. The last share is still being used by the group `OTHERS` (not shown).

Figure 4-2 illustrates the rising tide model. Moving from left to right within a single priority shows how WLM grants additional CPU to the workload groups.

Figure 4-2 **CPU allocation: the rising tide model**



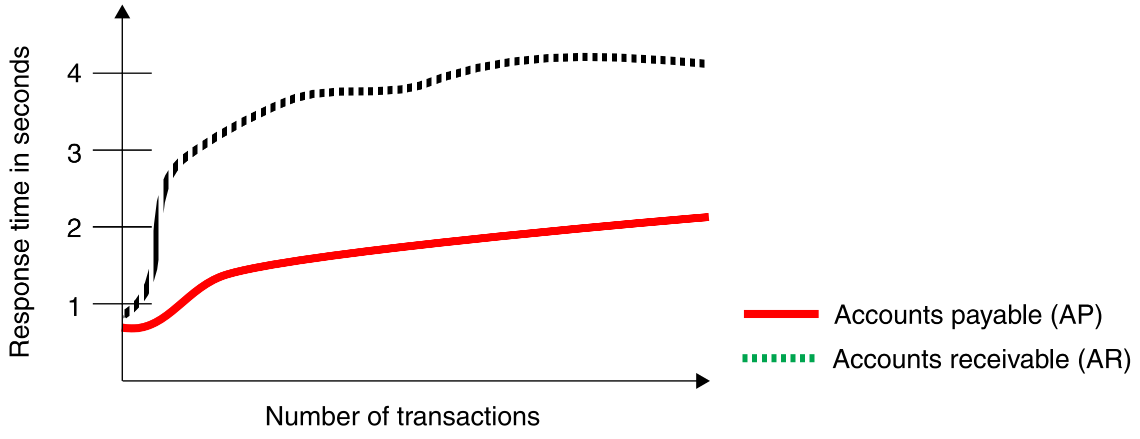
Example of WLM in use

Consider a server that runs two workloads:

- Accounts payable
- Accounts receivable

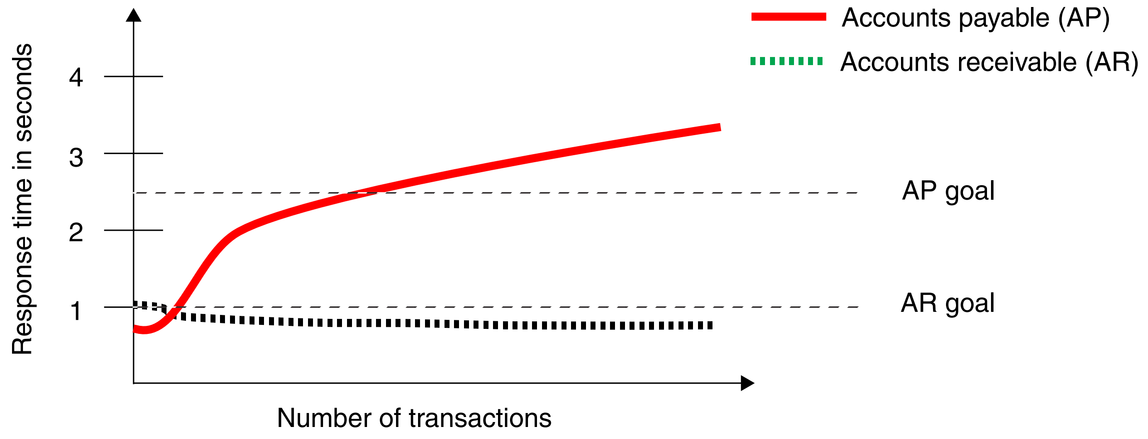
The accounts payable and accounts receivable workloads run constantly. Without WLM, the performance of these workloads varies greatly throughout the day, based mainly on the amount of work competing workloads have at any given time. Consequently, the server might be overloaded with accounts payable processing, greatly slowing the accounts receivable processing. Figure 4-3 illustrates this scenario.

Figure 4-3 **Server without WLM**



By using WLM, you can define SLOs for the workloads to specify their desired performance levels. For example, the accounts payable workload can be assigned a goal of paying out money in less than 2.5 seconds, while the accounts receivable workload is assigned a goal of depositing the incoming money in less than 1 second. WLM uses data from data collectors to determine how much a workload is underachieving or overachieving its desired performance level. WLM then automatically adjusts the CPU allocation, based on priorities, for workload groups containing workloads that are underachieving or overachieving. Figure 4-4 shows how the workloads are controlled under WLM.

Figure 4-4 Server with WLM



In this example, the accounts receivable workload has priority 1 and a response time goal of less than 1 second. The accounts payable workload has priority 2 and a response time goal of less than 2.5 seconds. When resources are not sufficient to meet the SLOs for both workloads, the accounts receivable workload is favored over the accounts payable workload. For another example, see “Specifying the priority (required)” on page 176.

5 How do I use WLM?

This chapter discusses the basic steps needed to use WLM. The remaining chapters explain these steps in detail.

The WLM configuration allows you to:

- Treat an entire vPar or nPar as a workload
- Create workload groups based on PSET or FSS groups to share a system or partition among several workloads

You then assign shares-based or goal-based SLOs to the workloads. Optionally, you indicate when each SLO is active. You also specify a means for collecting the performance data of each workload that has a performance goal.

Your performance collecting programs then use a utility called `wlmrcvdc` or an API to provide the data to WLM. The utility and the API are included with WLM and are explained in the section “What methods exist for sending data to WLM?” on page 250.

Next, WLM compares, at every interval, the reported and desired performance levels for each SLO. WLM then adjusts, if needed, CPU resources for the SLOs’ associated workloads based on the resources needed to satisfy the SLOs and their assigned priorities.

Also, WLM generates SLO status information that is made available through EMS.

Phasing in your WLM implementation

This section suggests one way in which to implement WLM. This is only a suggestion and may not be appropriate in all situations.

Step 1. Implement a static configuration with usage goals.

A static implementation has a set number of workload groups that are active at all times. One way to implement such a configuration is to set up usage goals for the workload groups.

Evaluate the system and workload performance under this configuration from one to seven days and fine-tune the configuration.

Step 2. Implement a configuration with metric goals for the workloads.

Again, evaluate the system and workload performance under the configuration from one to seven days and fine-tune the configuration.

Step 3. If applicable, implement a time-based configuration with goals.

Use the `condition` and `exception` keywords to indicate times when SLOs are active. These keywords are explained in the section “Specifying when the SLO is active (optional)” on page 192.

Set `mincpu` to 0 and `maxcpu` to 100. Monitor the system for one to two weeks to determine:

- How often are the goals met?
- Do the workload groups ever reach the `mincpu` or `maxcpu` values?
- Do the `mincpu` and `maxcpu` values need adjusting?

For more information on `mincpu` and `maxcpu`, see “Specifying the lower and upper bound requests on CPU (optional)” on page 180.

Steps for using WLM

The following steps outline how to configure WLM. Syntax information is available in “Configuring WLM” on page 127.

If you prefer not to work directly with a configuration file, you can use the WLM Configuration Wizard, invoked by the command /opt/wlm/bin/wlmcw, to create configurations. The wizard does not provide all the functionality available through a configuration file, but it does greatly simplify the process of creating a configuration. After creating a configuration file using the wizard, you can view the file to learn, and become more comfortable with, the syntax and possibly create more complex configurations. The WLM GUI, at /opt/wlm/bin/wlmgui, also allows you to configure WLM without directly editing a configuration file; however, you do need to be familiar with the configuration file syntax.

In addition, WLM provides a number of example configurations in /opt/wlm/examples/wlmconf/ that you can modify to fit your environment. Also, the WLM Toolkits provide numerous example configurations. For pointers to those configurations, see the wlmtk(5) man page.

To use WLM:

Step 1. Identify the workloads to run on a given system.

Each workload can consist of one or more applications and multiple users.

Step 2. Separate the workloads into three types:

- Workloads without goals (shares-based)
- Workloads with CPU usage goals (goal-based)
- Workloads with performance goals (goal-based)

For information on the types of workloads and their associated SLOs, see “Shares-based SLOs vs goal-based SLOs” on page 108.

Start a WLM configuration file using a text editor. Define workload groups for your workloads.

Step 3. For workloads without goals, add shares-based SLOs to your configuration.

Determine how much CPU each workload requires so that you can set appropriate CPU shares requests for the corresponding workload groups. One method for determining CPU needs is illustrated in the example configuration file “manual_entitlement.wlm” on page 320.

For information on the types of SLOs, see “Shares-based SLOs vs goal-based SLOs” on page 108.

Step 4. For workloads with CPU usage goals, add goal-based SLOs to your configuration. For information on usage goals, see “Specifying a goal (optional)” on page 183.

Step 5. For workloads with performance goals, add goal-based SLOs to your configuration:

a. Determine the performance metrics that are already available and implement other metrics that are needed in order to ensure that the SLOs are being met.

b. Send the metrics to WLM using one of WLM's built-in data collectors or develop your own collector.

Data collectors supply metric data to the WLM daemon. The daemon then uses this data to determine new resource allocations to enable the workload groups to achieve their SLOs.

You have a number of options when it comes to data collectors:

- The easiest data collector to set up is `wlmrcvdc` using the `sg_pkg_active` command, `wlmoradc` command, one of the `glance_*` commands, or one of the other commands given in the `wlmrcvdc` section of the appendix "WLM command reference" on page 373.
- You can also set up `wlmrcvdc` to forward the stdout of a data-collecting command to WLM.
- Combining `wlmsend` with `wlmrcvdc`, you can send data to WLM from the command line, a shell script, or a perl program.
- If you are writing a data collector in C, your program can interface directly with WLM through the libwlm(3) API.

NOTE Data collectors invoked by WLM run as root and can pose a security threat. Hewlett-Packard makes no claims of any kind with regard to the security of data collectors not provided by Hewlett-Packard. Furthermore, Hewlett-Packard shall not be liable for any security breaches resulting from the use of said data collectors.

For an overview of data collectors, see the section "How applications can make metrics available to WLM" on page 238.

For more information on built-in data collectors and how to create your own data collectors, see "Supplying data to WLM" on page 237.

Step 6. (Optional) Tune the controllers' behavior

Consider tuning controllers if your workload performance is not responding to load changes quickly enough or if workload performance is erratic.

Step 7. (Optional) For notification of SLO status changes, monitor WLM's EMS resources

For information on EMS, see "Monitoring SLO compliance and WLM" on page 357.

Step 8. (Optional) Activate the configuration in passive mode.

WLM operates in "passive mode" when you include the `-p` option in your command to activate a configuration. With passive mode, you can see approximately how a particular configuration is going to affect your system—without the configuration actually taking control of your system.

For information on passive mode, including its limitations, see "Passive mode versus actual WLM management" on page 226.

Activate the WLM file configfile in passive mode as follows:

```
# wlmd -p -a configfile
```

To see approximately how the configuration would affect your system, use the WLM utility `wlminfo`.

For `wlmd` syntax information, see "`wlmd` command syntax" on page 373.

Step 9. Activate the configuration.

NOTE WLM allows you to secure its communications. For information, see the section HOW TO SECURE COMMUNICATIONS in the wlmcert(1M) man page.

Activate your configuration—putting WLM in control of your system's resources—as follows:

```
# wlmd -a configfile
```

To generate audit data (`-t`) and log statistics (`-l all`), use the following command:

```
# wlmd -t -a configfile -l all
```

Step 10. Monitor SLO compliance

Using `wlmgui` or `wlminfo` with its `slo` command allows you to monitor your SLOs.

Also, the WLM EMS monitor makes various status data available to EMS clients. You can check this data to verify SLO compliance.

For more information on this topic, see “Monitoring SLO compliance and WLM” on page 357.

Step 11. Monitor data collectors.

Data collection is a critical link in the effective maintenance of your configured service-level objectives. When a data collector dies, each SLO that uses the data from the dead collector is affected. Consequently, you should monitor your data collectors so you can be aware when one dies.

When using `wlminfo slo`, there are two columns that can indicate the death of a data collector process: State and Concern. For more information on these columns, see the wlminfo(1M) man page.

Alternatively, you can configure EMS monitoring requests that notify you on the death of a data collector.

The SLO’s EMS resource:

/applications/wlm/slo_status/*SLONAME*

changes to:

```
WLM_SLO_COLLECTOR_DIED (5)
```

Use the EMS configuration interface (available in the SAM “Resource Management” application group) to set up monitoring requests to watch for this situation.

Step 12. (Optional) Configure global arbitration across partitions

You can set up WLM to automatically move CPUs between virtual partitions or nPartitions in response to the service-level objectives of the workloads in the partitions. The workloads would be workload groups inside the partitions or the partitions themselves if you did not define workloads groups in the partitions.

NOTE By default, WLM's global arbitration uses non-secured communications. If this is a concern, use the `wlmpard -s` option to secure communications—or use global arbitration only on trusted local area networks.

For more information on the global arbiter, including its passive mode, see the chapter "Managing SLOs across partitions" on page 283.

Step 13. (Optional) Set up WLM to automatically start its `wlmd` and `wlmpard` daemons at reboot.

Edit the /etc/rc.config.d/wlm file as explained in the sections "Setting WLM to start automatically at reboot" on page 230 and "Setting WLM global arbitration to start automatically at reboot" on page 231. You can also set variables in /etc/rc.config.d/wlm to start logging statistics and generating audit data automatically at reboot.

Reconfiguring WLM

To fine-tune an existing configuration, follow these steps:

Step 1. Edit the WLM configuration file

For information on the configuration file, see “Configuring WLM” on page 127.

Step 2. (Optional) Activate the configuration in passive mode:

```
# wlmd -p -a configfile
```

With passive mode, you can see approximately how a particular configuration is going to affect your system—without the configuration actually taking control of your system. To see how the configuration would affect your system, use the WLM utility `wlminfo`.

For `wlmd` syntax information, see “`wlmd` command syntax” on page 373.

Step 3. Activate the configuration:

```
# wlmd -a configfile
```

It is not necessary to shut WLM down before activating a new configuration.

Disabling WLM and its global arbiter

If you desire to temporarily return control of your system to the regular HP-UX resource scheduling, enter the following command to kill the WLM daemon:

```
# wlmd -k
```

After a de-activation, you can restart WLM using the last active configuration with the command:

```
# wlmd -A
```

To prevent WLM from starting automatically at reboot, set the `WLM_ENABLE` variable in the file /etc/rc.config.d/wlm to 0:

```
WLM_ENABLE=0
```

To stop the WLM global arbiter, enter the command:

```
# wlmpard -k
```

You can restart the arbiter using the last configuration with the command:

```
# wlmpard -A
```

To prevent the global arbiter from starting automatically at reboot, set the `WLMPARD_ENABLE` variable in the file /etc/rc.config.d/wlm to 0:

```
WLMPARD_ENABLE=0
```

6 Configuring WLM

This chapter introduces the WLM configuration file and how to activate the configuration so that WLM manages the system. It covers the creation and activation of the WLM configuration file.

The WLM configuration file is the main user interface for controlling WLM. This file is an ASCII file that you can edit with a text editor.

WLM does not have a default configuration file.

NOTE WLM and PRM have separate configuration files, each with its own syntax. While WLM and PRM configuration files can define an equivalent configuration, the files cannot be interchanged. For information on converting an existing PRM configuration file to a WLM configuration file, see “Migrating from PRM to WLM” on page 457.

A WLM configuration file consists of:

- The `version` keyword
- The `icod_thresh_pri` keyword
- The `icod_filter_intervals` keyword
- The `primary_host` keyword, discussed in Chapter 9, “Managing SLOs across partitions,” on page 283
- System-wide settings
- Zero or one `prm` structures
- `slo` structures
- `tune` structures

For information on these items, see the following sections:

- “Specifying WLM’s parser version” on page 134
- “Defining the PRM components” on page 139
- “Defining SLOs” on page 169
- “Tuning the metrics and the SLOs” on page 196

Configuration file syntactic conventions

The following syntactic conventions are used in the WLM configuration file:

- SLOs, tunables, and PRM information are represented as structures. You can list these structures in any order.
- A structure is an unordered list of keyword/value pairs.
- Unless otherwise specified, white space is ignored.
- The semicolon character (;) is used as a separator between keyword/value pairs.
- The equal character (=) is used to separate a keyword from its value.
- Curly brackets ({ and }) are used to delimit the specification of a structure.
- Some keywords are global and can only be defined outside the scope of a structure (outside curly brackets).
- Any keyword can appear at most once in a structure.
- The hash character (#) comments out the rest of the line, unless quoted as part of a name.
- Floating-point numbers cannot be expressed using scientific (exponent) notation.
- An unquoted number cannot be a name.

- The names you supply can consist of:
  - Uppercase letters (A-Z)
  - Lowercase letters (a-z)
  - Digits (0-9)
  - The underscore character (_)

    Do not use an underscore (_) to start the name of a workload group, an `slo` structure, or a metric.
  - The plus character (+)
  - The hyphen character (-)
  - The forward slash (/)

    Do not use slashes in the names of `slo` structures or metrics.
  - The period (.)
  - Quoted characters

    You can use any printable characters (except white space, <, >, &, `'`, ", \, and =) by enclosing them in double quotes (for example: `groups = "my@group":2;`). You cannot use the slash character (/) in the names of `slo` structures or metrics, even when quoted.

Using the WLM configuration wizard

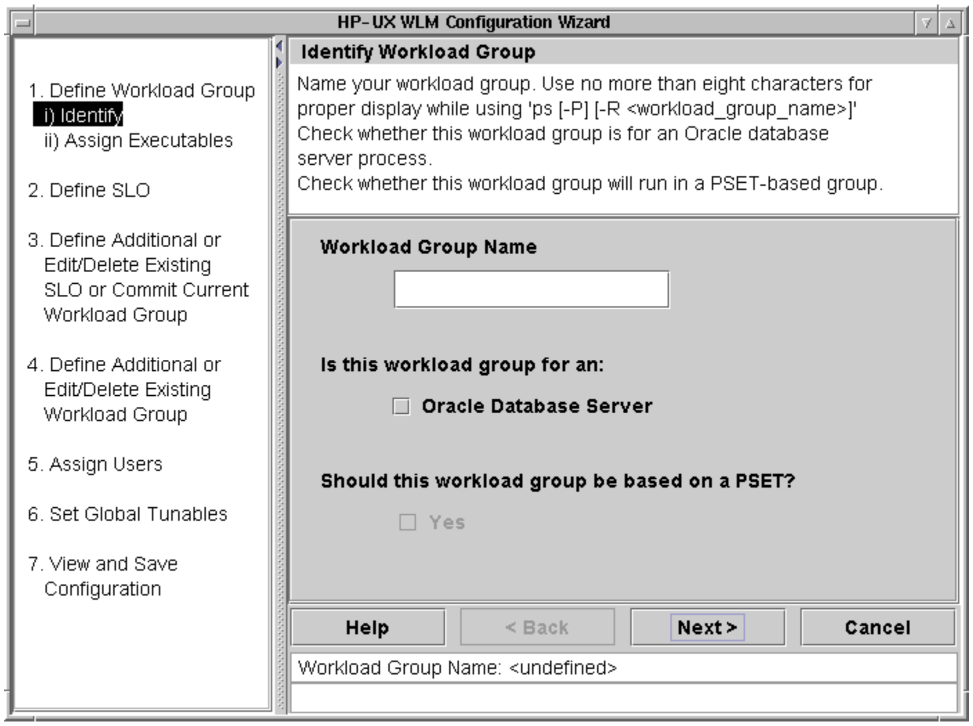
If you prefer not to work directly with a configuration file, you can use the WLM Configuration Wizard. Invoke the wizard using the following command:

```
# /opt/wlm/bin/wlmcw
```

The wizard does not provide all the functionality available through a configuration file, but it does greatly simplify the process of creating a configuration. After creating a configuration file using the wizard, you can view the file to learn, and become more comfortable with, the syntax and possibly create more complex configurations.

The following figure shows one of the Wizard's screens. The left pane shows the steps the Wizard guides you through.

Figure 6-1 HP-UX WLM Configuration Wizard



HP-UX WLM Configuration Wizard

1. Define Workload Group
   i) Identify
   ii) Assign Executables
2. Define SLO
3. Define Additional or Edit/Delete Existing SLO or Commit Current Workload Group
4. Define Additional or Edit/Delete Existing Workload Group
5. Assign Users
6. Set Global Tunables
7. View and Save Configuration

Identify Workload Group

Name your workload group. Use no more than eight characters for proper display while using 'ps [-P] [-R <workload_group_name>]'
Check whether this workload group is for an Oracle database server process.
Check whether this workload group will run in a PSET-based group.

Workload Group Name

Is this workload group for an:

☐ Oracle Database Server

Should this workload group be based on a PSET?

☐ Yes

Help | < Back | Next > | Cancel

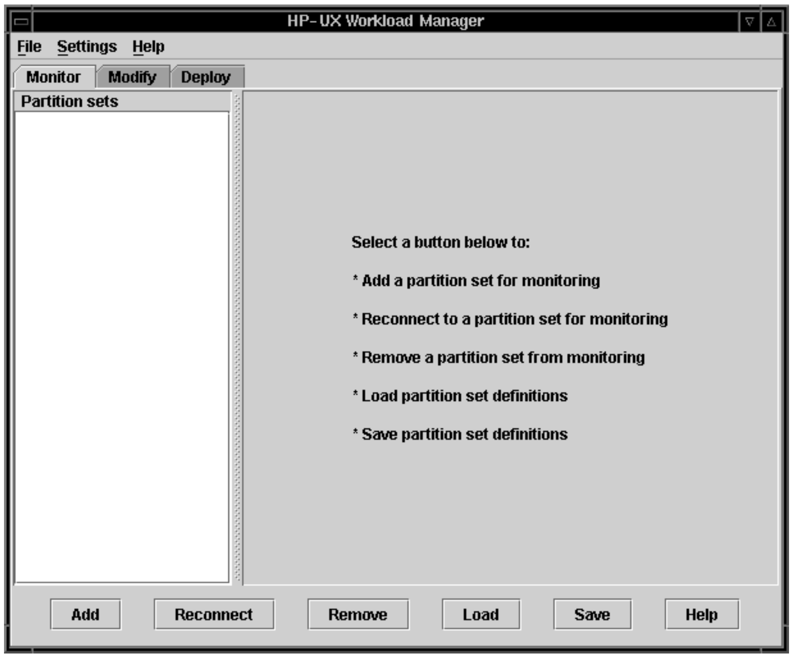
Workload Group Name: <undefined>

Using the WLM GUI

The WLM graphical user interface, or GUI, allows you to edit existing configurations and create new ones. You can then deploy the configurations locally or remotely. The GUI also provides current and historical data. However, using the GUI does require you to use configuration file syntax. To configure WLM using the GUI, use the Modify tab to create a file then go to the Deploy tab to activate the configuration.

The WLM GUI's initial screen is shown below. The GUI has three tabs, allowing you to monitor your SLOs, modify configurations, and deploy configurations. For tips on using these tabs, see the next section.

Figure 6-2 **HP-UX WLM GUI**



Select a button below to:

* Add a partition set for monitoring

* Reconnect to a partition set for monitoring

* Remove a partition set from monitoring

* Load partition set definitions

* Save partition set definitions

Tips on using the WLM GUI's tabs

Here are some tips to get you started using the various tabs.

- Monitoring (Monitor tab)

  Select a configuration to monitor by:

  — Selecting the **[Add]** button to specify the host running the configuration you want to monitor

  or

  — Selecting the **[Load]** button to load a definition file that you saved from a previous monitoring session

- Modifying (Modify tab)

  The Modify tab allows you to

  — Import a configuration that you have been monitoring so that you can modify it

  — Open a configuration file saved on the hard drive of an HP-UX system

  — Create a new configuration

  Although you save files in the Deploy tab, you name them in the Modify tab. Selecting a partition in the left pane changes the top, right pane to include a Filename field where you name the file. Be sure to select the **[Commit changes]** button and then the **[Validate]** button for a configuration before you go to the Deploy tab.

- Deploying (Deploy tab)

  You deploy a configuration you validated in the Modify tab by:

  1. Selecting the **[Import]** button to bring the configuration into the Deploy tab
  2. Selecting the **[Save & Activate]** button

Controlling system resources

Using the WLM GUI, configure WLM to control resource allocation as follows:

1. Select the Modify tab
2. Import an existing configuration (one that you had been monitoring), open a configuration file from the disk, or create a new one
3. Make any desired changes
4. Select the **[Commit changes]** button if you have made any modifications to the configuration
5. Select the **[Validate]** button
6. Select the Deploy tab
7. Select the **[Import]** button
8. Select the **[Save & Activate]** button

See the online help for information on using the WLM GUI to accomplish the following tasks:

- Reserve a portion of a system all the time
- Reserve a portion of a system at specified times
- Reserve a portion of a system based on an event
- Allocate CPU as used (set a utilization goal)
- Allocate CPU per some metric
- Automatically resize PSETs
- Automatically resize vPars
- Automatically resize nPars
- View the configuration being modified
- Monitor the applications (workload groups)
- Monitor WLM operations
- Start WLM
- Stop WLM
- Try WLM without it taking control of my system

Specifying WLM's parser version

Use the optional `version` keyword at the beginning of your configuration file to specify which version of the WLM configuration file parser to use with a particular configuration file.

This keyword is useful when future versions of the parser are not able to maintain backward-compatibility. However, WLM will be able to parse all configuration files properly once the correct parser version is known. By using the `version` keyword, you are telling WLM which parser to use for the configuration file.

The default and only valid value is 0:

```
version = 0;
```

Specify the `version` keyword at most once in any configuration, at the beginning of the configuration file, outside any `prm`, `slo`, and `tune` structures.

Notification of 'Instant Capacity needed' / Pay Per Use optimization

Use the optional `icod_thresh_pri` keyword to request notification of when Instant Capacity (formerly known as iCOD) or Pay Per Use (PPU) reserves could help you meet your SLOs.

NOTE This notification is supported with PPU v4 and with any version of Instant Capacity.

If you plan on using Instant Capacity in combination with WLM's vPar management, use vPars version A.03.01 or later.

WLM does not enable Instant Capacity or PPU reserves. It merely informs you that the reserves could help you meet your SLOs; you must manually activate the reserves if you feel it is appropriate. (Independent of this keyword though, you can use `wlmpard` to automate activation or these reserves.)

To enable notification, use the `icod_thresh_pri` keyword in your WLM configuration. This is a global keyword—it should be outside all `prm`, `slo`, and `tune` structures. The syntax for `icod_thresh_pri` is:

```
icod_thresh_pri = integer;
```

where

`icod_thresh_pri`

Is an optional keyword. Specifying this keyword allows you to use an EMS resource to notify you when Instant Capacity reserves could help meet SLOs.

`integer`

Is an integer value greater than or equal to 1. If Instant Capacity reserves exist, they are considered needed whenever SLOs with a priority from 1 to *`integer`* are failing. If an SLO with a priority in this range is failing, notification that Instant Capacity reserves are needed is sent using an EMS resource.

Notification is made through the following EMS resource:

/applications/wlm/icod_reserves_needed

When Instant Capacity reserves are not available or are not required, the EMS resource is set to:

```
ICOD_NEEDED_FALSE (0)
```

When Instant Capacity reserves exist and are needed, the EMS resource is set to:

```
ICOD_NEEDED_TRUE (1)
```

In this case, it may be possible to reduce SLO failures by activating some Instant Capacity reserves.

When using `icod_thresh_pri`, you can also use `icod_filter_intervals`, another global keyword that must be outside all `prm`, `slo`, and `tune` structures. The syntax is:

```
icod_filter_intervals = integer;
```

where

`icod_filter_intervals`

Is an optional keyword. It allows you to ignore short-term fluctuations in the conditions that affect the EMS resource.

`integer`

Is an integer value greater than or equal to 1. This value indicates the number of consecutive WLM intervals over which Instant Capacity reserves must be needed before the EMS resource is changed to `ICOD_NEEDED_TRUE`. Similarly, it is the number of intervals over which Instant Capacity reserves are not required before the EMS resource is changed to `ICOD_NEEDED_FALSE`.

For background information on Instant Capacity and PPU, see "Integrating with Temporary Instant Capacity/ Pay Per Use" on page 435.

System-wide settings

Use one or more of the following keywords outside all structures in your configuration to set values for the host (typically a virtual partition or nPartition).

To specify the minimum number of CPU shares a host receives, use the `hmincpu` keyword and the following syntax:

`hmincpu = `*`min`*`;`

where

`hmincpu`	Is an optional keyword. You cannot specify this keyword in a configuration that includes a `prm` structure.
`min`	Is the host's minimum number of CPU shares. The value must be an integer between 0 and the host's `hmaxcpu` value, inclusive. *`min`* is out of the total CPU, which is 100 multiplied by the number of CPUs (absolute CPU units are implied). The default value is 0.

To specify the maximum number of CPU shares a host receives, use the `hmaxcpu` keyword and the following syntax:

`hmaxcpu = `*`max`*`;`

where

`hmaxcpu`	Is an optional keyword. You cannot specify this keyword in a configuration that includes a `prm` structure.
`max`	Is the host's maximum number of CPU shares. The value must be an integer between the host's `hmincpu` value and the system's total CPU, inclusive. Total CPU is 100 multiplied by the number of CPUs (absolute CPU units are implied).

NOTE For information on the effect of the `hmincpu` and `hmaxcpu` keywords in passive mode, see “Passive mode versus actual WLM management” on page 226.

The larger a host CPU weight value you assign a host, the more CPU it receives when there is not enough CPU to satisfy all requests at a given priority level. A larger host CPU weight value also gets a host more CPU when all SLOs can be satisfied and there is excess CPU to be distributed.

To specify a host’s CPU weight, use the `hweight` keyword and the following syntax:

```
hweight = wt;
```

where

`hweight`	Is an optional keyword. You cannot specify this keyword in a configuration that includes a `prm` structure.
`wt`	Is the host’s CPU weight. The value must be an integer greater than or equal to 1. The default weight for a host is 1.

Defining the PRM components

Use a `prm` structure to define Process Resource Manager, or PRM, components—excluding the CPU allocations. The CPU allocations are controlled by WLM, as determined by the entries in the `slo` structures.

NOTE If you plan on managing only virtual partitions or nPartitions—with no FSS groups or PSETs inside them, there is no need to specify a `prm` structure. You can go immediately to the section, “Defining SLOs” on page 169.

WLM replaces any existing PRM configuration on the system with the PRM items specified in the `prm` structure. For example, if a system was using PRM before starting the WLM daemon, any PRM groups, application records, and user records defined in PRM must be defined in this structure as well, or they will not be in force when WLM takes control of the system.

Defining a `prm` structure consists of the following tasks:

- Specifying workload groups (optional)
- Specifying users’ workload group access (optional)
- Assigning applications to workload groups (optional)
- Assigning secure compartments to workload groups (optional)
- Specifying disk bandwidth shares (optional)
- Specifying a group’s minimum CPU (optional)
- Specifying a group’s maximum CPU (optional)
- Weighting a group so it gets more CPU (optional)
- Specifying a group’s minimum memory (optional)
- Specifying a group’s maximum memory (optional)
- Weighting a group so it gets more memory (optional)

A `prm` structure takes the following form:

```
prm {
   groups = { FSS_group_definition | PSET_group_definition } [, ...];
   [ users = user : init_group [alt_group1 alt_group2 ...] [, ...]; ]
   [ apps = group : application [alt_name1 alt_name2...] [, ...]; ]
   [ disks = group : volume ent [, ...]; ]
   [ gmincpu = group : min [, ...]; ]
   [ gmaxcpu = group : max [, ...]; ]
   [ weight = group : wt [, ...]; ]
   [ gminmem = group : min [, ...]; ]
   [ gmaxmem = group : max [, ...]; ]
   [ memweight = group : wt [, ...]; ]

}
```

where

FSS_group_definition has the syntax:

group : *group_ID*

PSET_group_definition has the syntax:

group : PSET

Here is an example `prm` structure:

```
prm {
   groups = finance : 2,
            sales : 3,
            marketing : PSET;

   users = jdoe : finance,
           pdoe : sales,
           admin : finance sales marketing;

   apps = finance : /bin/dbase "dbase*Finance",
          sales : /bin/dbase "dbase*Sales";

   gmincpu = finance : 20,
             sales : 10;

   gmaxcpu = sales : 20;


   gminmem = finance : 30,
             sales : 10,
             marketing : 20;

   disks = OTHERS : /dev/vg01 1,
           finance : /dev/vg01 35,
           sales : /dev/vg01 35;

}
```

Defining PRM components using the WLM GUI

In addition to defining the PRM components using a text editor, you can use the WLM Configuration Wizard (/opt/wlm/bin/wlmcw). Alternatively, you can use the WLM GUI (/opt/wlm/bin/wlmgui). For information on using the WLM GUI to accomplish various tasks, see its online help.

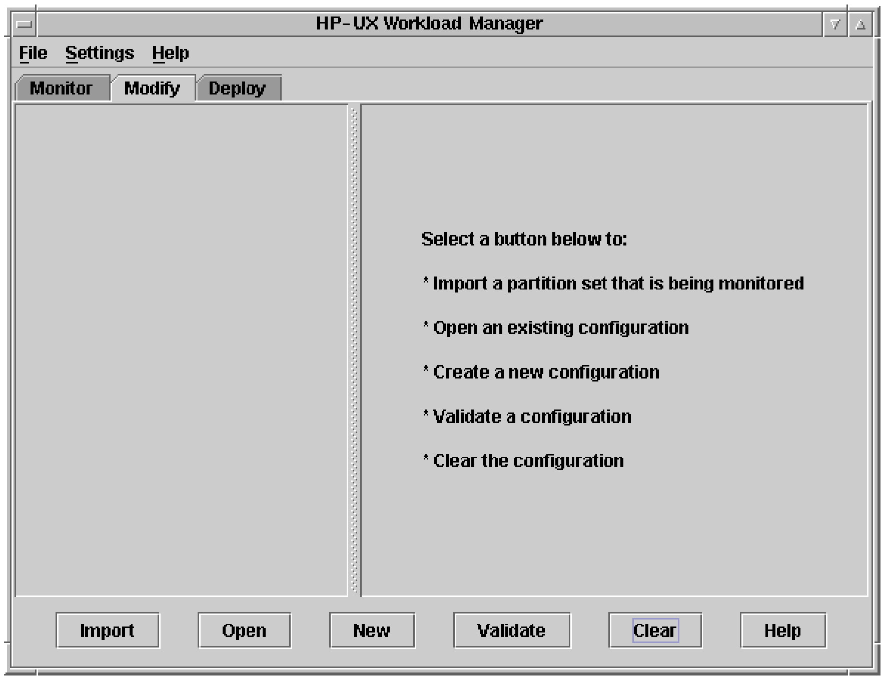
The following steps show how to define a workload group using the GUI.

Step 1. Set your `DISPLAY` environment variable.

Step 2. Start the GUI:

```
# /opt/wlm/bin/wlmgui
```

Step 3. Select the Modify tab.



Select a button below to:

* Import a partition set that is being monitored

* Open an existing configuration

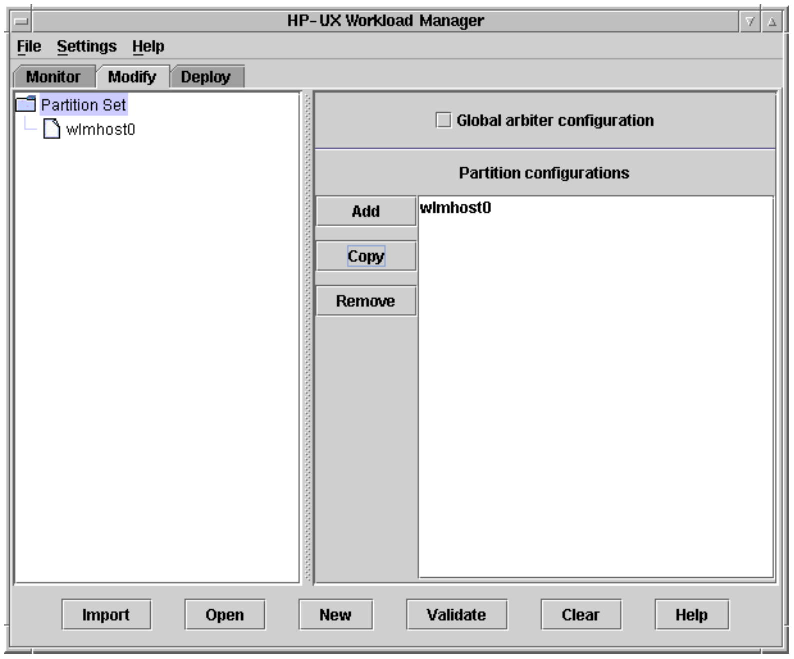
* Create a new configuration

* Validate a configuration

* Clear the configuration

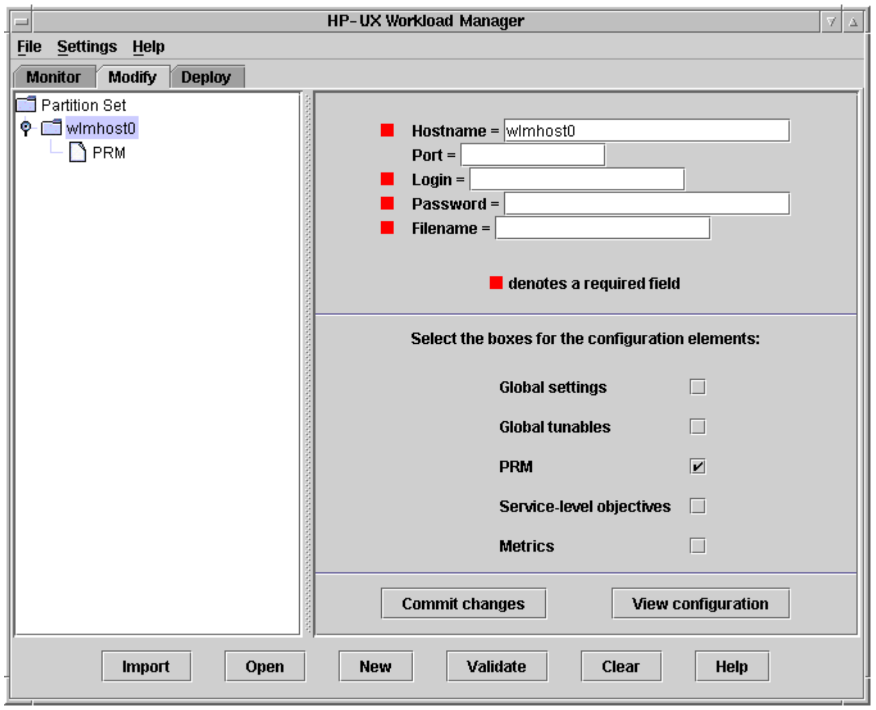
Step 4. Select the **[New]** button to start a new configuration. The left and right panes change as shown in the next step.

Step 5. Select the **[Add]** button. A new host name appears in the right pane and in the left pane.



Step 6. Select the name of the new host, wlmhost0 in this case, in the left pane. The right pane changes, allowing you to set the configuration elements you would like to see in the new configuration.

Check the “PRM” box in the right pane.



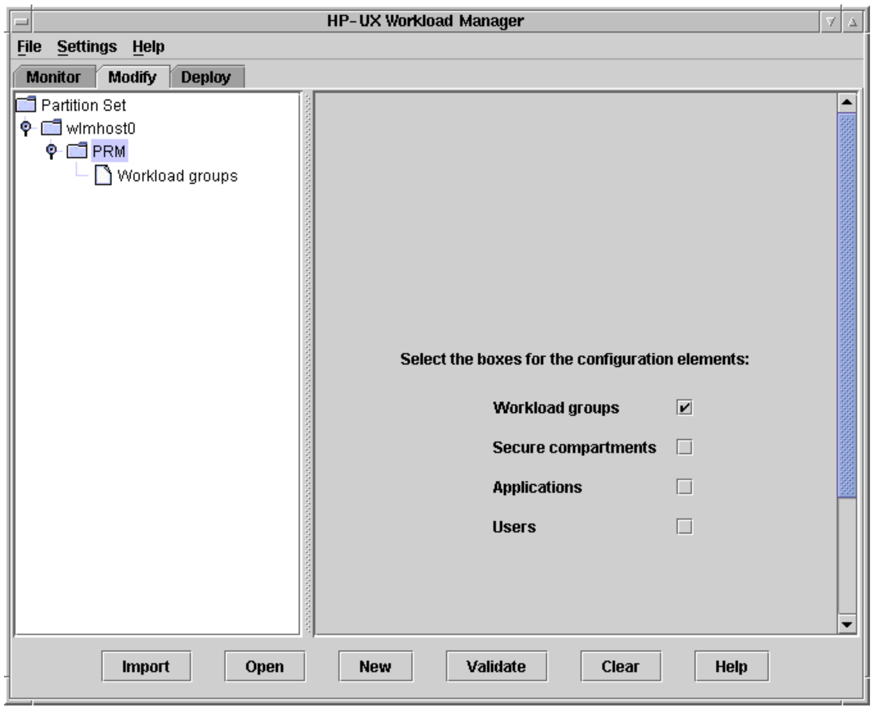
🟥 Hostname = wlmhost0
Port =
🟥 Login =
🟥 Password =
🟥 Filename =

🟥 denotes a required field

Select the boxes for the configuration elements:

Global settings	☐
Global tunables	☐
PRM	✓
Service-level objectives	☐
Metrics	☐

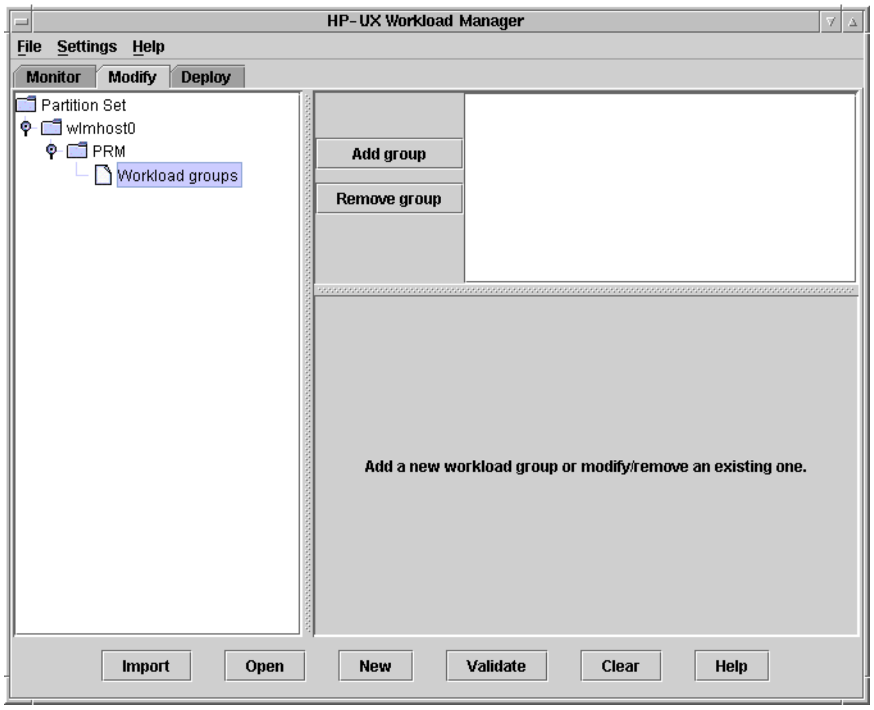
Step 7. Click “PRM” in the left pane. This changes the right pane to show a “Workload groups” item. Check the “Workload groups” box.



Select the boxes for the configuration elements:

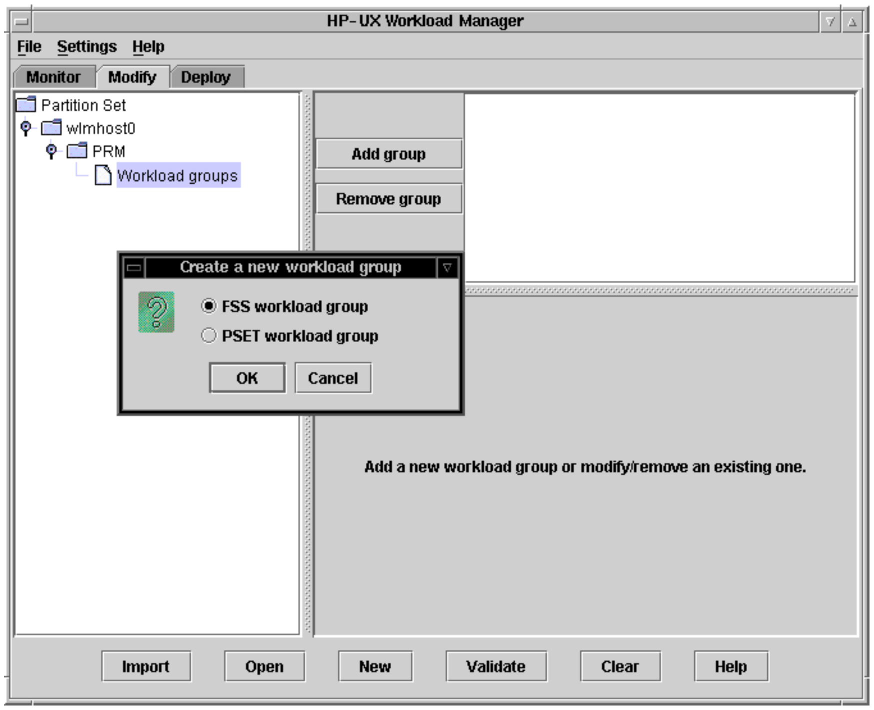
Workload groups [✓]
Secure compartments [ ]
Applications [ ]
Users [ ]

Step 8. Select the “Workload groups” item in the left pane. The right pane changes, allowing you to add a group.

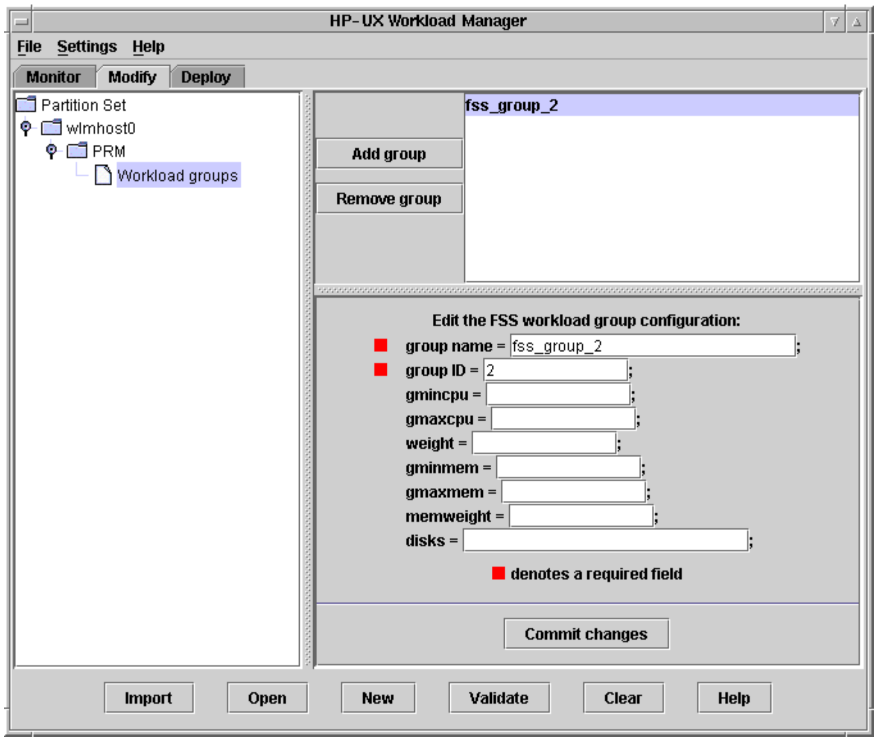


Add a new workload group or modify/remove an existing one.

Step 9. Select the **[Add group]** button. A new dialog appears. Accept the default (FSS workload group) by selecting the **[OK]** button.



Step 10. The right pane changes again, with the GUI starting to define a workload group for you. The GUI fills in the required fields for you, although you can change the values if you like. Fill in other fields if you like.



Edit the FSS workload group configuration:

🟥 group name = fss_group_2;
🟥 group ID = 2;
gmincpu = ;
gmaxcpu = ;
weight = ;
gminmem = ;
gmaxmem = ;
memweight = ;
disks = ;

🟥 denotes a required field

Step 11. Select the **[Commit changes]** button to save the current configuration in memory. To save changes to disk, go to the Deploy tab. (The file is saved to the file specified in the Filename field in the Modify tab when the system is selected in the left pane.)

The above steps highlight only one use of the WLM GUI. For additional information, see the GUI's online help.

Specifying workload groups (optional)

A workload group can be one of two types: FSS or PSET.

An FSS group is allocated CPU resources by the Fair Share Scheduler (FSS) in the HP-UX kernel. WLM can automatically adjust the CPU allocation for an FSS group based on the group's progress toward an SLO. You can define multiple SLOs for a single FSS workload group.

A PSET group is based on a processor set. A PSET group's CPU resources are determined by the number of CPUs assigned to the PSET group. With PSET groups, you can define one or more `slo` structures per group, but it is not required. If a PSET group has an SLO, WLM uses absolute CPU units. With these units, 100 represents one CPU. For more information on absolute CPU units, see the section "Using absolute CPU units" on page 205.

Processes in either type of group have equal access to the CPU resources allocated to their group. Within a group, the HP-UX standard scheduler is used.

Define these groups using the `groups` keyword. The `groups` keyword must appear exactly once in a configuration.

Specify workload group names and their IDs in the `prm` structure using the following syntax:

```
groups = { FSS_group_definition | PSET_group_definition } [, ...];
```

An *`FSS_group_definition`* has the following syntax:

```
group : group_ID
```

where

`group` Is the workload group name. Use names that are less than eight characters long for proper display by the `ps -P` command. Do not start the name with an underscore (_).

The `PRM_SYS` group is the default group for processes run by the root user. You can specify `PRM_SYS` as an FSS group (*`group_ID`* must be 0) and use it with the `apps` and `user` keywords in a `prm` structure. You cannot, however, use it with the `gminmem`, `gmaxmem`, or `memweight` keywords, in an `slo` structure, or as a PSET group.

The `OTHERS` group is the default group for users who are not assigned to groups. It is also the default group for applications that are not assigned to groups or are started by users who do not have assigned groups. You can specify `OTHERS` as an FSS group (*`group_ID`* must be 1), but not as a PSET group. It does not require an `slo` structure, but you can create one or more for it if you wish.

If you specify `gminmem`, `gmaxmem`, or `memweight` for any group in your configuration, be sure to specify a `gminmem` for any group, including `OTHERS`, for which 1% of the memory is not sufficient for its users and applications.

After WLM allocates CPU to groups with SLOs, `OTHERS` by default receives any remaining CPU. You can change this default with the `distribute_excess` tunable, explained in "Distributing excess CPU to your workloads (optional)" on page 206.

For any other FSS group you define, you must define an `slo` structure that references that group.

`group_ID` Is the workload group ID. These IDs must be uniquely assigned integer values from 0 to 63, inclusive. ID 0 is reserved for the `PRM_SYS` group. ID 1 is for the user default group `OTHERS`. If the `PRM_SYS` and `OTHERS` groups are not specified, they are created automatically.

A *`PSET_group_definition`* has the syntax:

```
group : PSET
```

where

`group`

Is the workload group name. Use names that are less than eight characters long for proper display by the `ps -P` command. Do not start the name with an underscore (_).

`group` cannot be either the `PRM_SYS` or `OTHERS` group.

NOTE If you specify `gminmem`, `gmaxmem`, or `memweight` for any group in your configuration, be sure to specify a `gminmem` for any group, including `OTHERS`, for which 1% of the memory is not sufficient for its users and applications.

With PSET groups, you can define one or more `slo` structures per group, but it is not required.

`PSET`

Indicates the group is to be based on a PSET.

NOTE You cannot define workload groups based on PSETs and use WLM's partition management (by specifying the `primary_host` keyword) in the same WLM configuration.

NOTE It is possible to create a WLM configuration that has more PSET-based workload groups than the underlying system has CPUs. However, if all the groups are active at the same time, some groups will necessarily not have a CPU assigned and their processes would be placed in `OTHERS`. Furthermore, if there are more PSET-based groups than CPUs and `transient_groups` is not set to 1, the allocation of CPUs to the PSET-based groups will never change. As a result, the extra groups do not add any value as they will never get any CPU.

Reserved workload groups

The `PRM_SYS` workload group (ID 0) is the default workload group for system processes. It is created automatically, but you can specify it in your configuration. You cannot however reference `PRM_SYS` outside a `prm` structure. Also, `PRM_SYS` cannot be created as a PSET group.

The `OTHERS` workload group (ID 1) is the default workload group for all nonsystem processes. It is also created automatically, but you can explicitly specify it in your configuration file if you would like to assign applications or users to it or have WLM manage its resources. If specified, this workload group does not have to be referenced in an `slo` structure.

`OTHERS` cannot be created as a PSET group.

Any workload group name starting with an underscore (_) is reserved for WLM use.

For information on `slo` structures, see “Defining SLOs” on page 169.

Specifying users’ workload group access (optional)

If the server where WLM runs is going to host users, you can specify the workload groups in which the users’ processes (including their shells) run. You can also specify any other workload groups the users should be able to access. Define this access with the `users` keyword.

The `users` keyword can appear, at most, once in a configuration.

WLM supports the use of netgroups (network-wide groups) to identify users. For more information on netgroups, see the netgroup(4) man page.

Specify the workload groups a user or netgroup member can access in the `prm` structure using the following syntax:

```
users = user : init_group [alt_group1 alt_group2 ...] [, ...];
```

where

`user` Is either an individual user’s login name or a string that begins with the plus character (+) followed by a netgroup name, such as +*`netgroupname`*. (Netgroups are defined in the file /etc/netgroup.)

If a netgroup is specified, then at configuration time, any member of that netgroup who is not listed individually assumes the *`init_group`* and *`alt_groupX`*s that the netgroup is assigned.

WLM ignores any line in /etc/netgroup that has an empty user field.

`init_group` Is the name of the initial workload group for the user or netgroup. This is the group `login` chooses when launching the user’s login shell, and the group `cron` chooses when scheduling jobs for that user.

For information on a user's group placement when the user belongs to multiple netgroups, see the *Process Resource Manager User's Guide*.

`alt_groupX`	(Optional) Is the name of one of the alternate workload groups for the user or netgroup. Alternate groups are groups other than the initial group in which the user or netgroup members are allowed to run processes.

NOTE Compartment records take precedence over application records, which take precedence over user records. (A process may start in a workload group other than its assigned group due to its inheriting the group of the process that started it. However, WLM will eventually move the process to the workload group specified in its record.)

Assigning applications to workload groups (optional)

Each workload group defines a workload. You can optionally assign one or more applications to a workload group using the `apps` keyword. However, because you can place applications in workload groups using `prmrun` and `prmmove`, you are not required to assign applications to workload groups with the `apps` keyword.

NOTE The system is polled every 30 seconds to ensure that processes are running in the appropriate workload groups. If a process forks child processes and immediately exits, the polling will likely miss the parent process. As a result, the parent process is never placed in its workload group. When the child processes are found during polling, WLM will not be able to determine the parent of the processes. Then, instead of being placed in the parent's assigned group, the child processes are placed in the group of the invoking user or in the `OTHERS` group. To avoid this condition, be sure parent processes exist for at least 30 seconds (the default polling interval). In various shells, you can use the `wait` command in a script to prevent the process from exiting immediately. (Placement of a process family can take up to two polling intervals: The parent process is moved to its assigned group in the first interval, while the child processes are moved to the group in the second interval.)

The `apps` keyword can appear, at most, once in a configuration.

Assign applications to workload groups in the `prm` structure using the following syntax:

```
apps = group : application [alt_name1 alt_name2...] [, ... ];
```

where

group

Is the name of the workload group in which the application should run.

application

Is the full path of the application, starting with a slash (/). The pathname must be to a binary or shell/interpreter.

You can use wildcards (`[`, `]`, `?`, `*`) to specify the filename, but not the directory name. If you use wildcards in a filename, enclose the full pathname in quotes and do not use alternate names for the application. For information on wildcards, see the PATTERN MATCHING NOTATION section in the regexp(5) man page.

If you specify an application filename using wildcard characters, all valid executables—without explicit application records—that match the pattern assume the *group* for the application.

For information on how to specify scripts in an `apps` statement, see the “Script example” on page 155.

alt_nameX

(Optional) Is an alternate name the application is assigned when executed. This is common for complex programs such as database programs that launch many processes and rename them. It is also common for shells and interpreters used in scripts; the names of the scripts are considered alternate names.

Using alternate names, you can place the various processes of a single application in different workload groups.

Wildcards (`[`, `]`, `?`, `*`) can be used when specifying alternate names. If using wildcards, enclose the alternate name in quotes.

If *`alt_nameX`* is not specified for an application, that application's group assignment is used for all processes with a file ID that matches the file ID of `application`. (The file ID is based on the file system device and the inode number.)

For an example showing how to specify scripts in an `apps` statement, see the "Script example" on page 155.

For an example showing how to specify an application that renames itself, see the "Renamed application example" on page 156.

NOTE Compartment records take precedence over application records, which take precedence over user records. (A process may start in a workload group other than its assigned group due to its inheriting the group of the process that started it. However, WLM will eventually move the process to the workload group specified in its record.)

Script example

To place a script in a workload group using an `apps` statement:

Step 1. Specify the full path of the shell or interpreter used in the script as the *`application`*.

Step 2. Specify the name of the script (just the name—no path) as the *`alt_name`*.

NOTE Because the full pathname is not required for the script, a rogue user can get access to workload groups—that would otherwise not be accessible—by using the name of the script for new scripts or wrappers.

Step 3. Ensure the shell or interpreter is listed in either /etc/shells or /opt/prm/shells.

For example, for a perl script named myscript.pl and using /opt/perl/bin/perl, the `apps` statement to place the script in group `scripts_grp` would be:

```
apps = scripts_grp : /opt/perl/bin/perl myscript.pl;
```

Renamed application example

To place a renamed application in a workload group using an `apps` statement:

Step 1. Specify the full path of the application as the *`application`*.

Step 2. Specify the new name the application takes once it is running as the *`alt_name`*.

For example, here is an `apps` statement for an Oracle database with two instances:

```
apps = Finance : /opt/oracle/bin/oracle "ora*Finance",
      Sales : /opt/oracle/bin/oracle "ora*Sales";
```

The Oracle database executable is the same in both cases; however, the alternate names specify different patterns to match. (Given the use of wildcards, the alternate names are quoted.) The patterns are based on the instance names. Consequently, processes starting with “ora” and ending with “Finance” are placed in the `Finance` workload group. Similarly, processes matching the pattern “ora*Sales” go into the workload group `Sales`.

Assigning secure compartments to workload groups (optional)

The HP-UX feature Security Containment, available starting with HP-UX 11i v2, allows you to create secure compartments, which provide file and process isolation. You can place one or more secure compartments in a single workload group. After creating your secure compartments, you can place them in workload groups using the `scomp` keyword.

The `scomp` keyword can appear, at most, once in a configuration.

Assign secure compartments to workload groups in the `prm` structure using the following syntax:

```
scomp = compartment : group [, ... ];
```

where

`compartment`

Is the name of a secure compartment you have already created using Security Containment or the /opt/prm/bin/srpgen utility.

`group`

Is the name of the workload group in which *`compartment`* should be placed.

NOTE Compartment records take precedence over application records, which take precedence over user records. (A process may start in a workload group other than its assigned group due to its inheriting the group of the process that started it. However, WLM will eventually move the process to the workload group specified in its record.)

Specifying disk bandwidth shares (optional)

NOTE To take advantage of disk bandwidth shares, your disks must be mounted and under the control of Logical Volume Manager (LVM). For information on LVM, see “Management of disk bandwidth” on page 446.

WLM does not dynamically allocate disk bandwidth shares to workload groups. To ensure that a workload receives a sufficient amount of bandwidth, assign the workload’s group a bandwidth share using the `disks` keyword.

The `disks` keyword can appear, at most, once in a configuration.

If you want to use disk bandwidth shares, then for each FSS group specified using the `groups` keyword, you must assign the group a disk bandwidth share using the `disks` keyword. In other words, there must be a one-to-one correspondence between the defined FSS groups and the groups that receive disk bandwidth shares.

NOTE If you use the `disks` keyword, you must explicitly create the `OTHERS` workload group and assign it a disk bandwidth share.

Specify disk bandwidth shares for logical volume groups in the `prm` structure using the following syntax:

```
disks = group : volume shares [, ...];
```

where

group	Is the workload group name. You cannot specify a PSET group in a `disks` statement.
volume	Names a logical volume group. *volume* must begin with /dev/v to be recognized.
shares	Is an integer value greater than or equal to 0. *shares* translates to a percentage of the disk bandwidth when dividing it by the number of disk bandwidth shares assigned to all the workload groups for the given volume group.

If one FSS workload group has disk shares for a volume group, then all FSS workload groups must have disk shares for that volume group.

Here is an example `disks` statement:

```
disks = OTHERS : /dev/vg01 1,
        finance : /dev/vg01 35,
        sales : /dev/vg01 35,
        marketing : /dev/vg01 29;
```

Specifying a group’s minimum CPU (optional)

You can assign workload groups a minimum number of CPU shares. This minimum is a hard lower limit (assuming the sum of the `gmincpu` values is less than or equal to the total CPU); the group receives less than this minimum only if it has no active SLOs and `transient_groups` is set to 1. This hard limit is different from the `mincpu` value in `slo` structures, which merely specifies the minimum CPU request the SLO controller can make.

To specify the minimum number of CPU shares a group receives, use the `gmincpu` keyword in the `prm` structure, according to the following syntax:

`gmincpu = `*`group`*` : `*`min`*` [, ...];`

where

group Is the workload group name.

min Is *group*’s minimum number of CPU shares. The value must be an integer between 0 and the group’s `gmaxcpu` value, inclusive.

min is out of the total CPU, which is 100 multiplied by the number of CPUs (if you set the tunable `absolute_cpu_units` to 1 or it is implied by other elements in your configuration)—or just 100 by default.

To have a minimum of 2 CPUs on an 8-CPU system with `absolute_cpu_units` enabled (`absolute_cpu_units = 1`), you would specify:

```
gmincpu = 200;
```

If you specify a *min* greater than the total CPU, WLM treats it as equal to the total CPU.

By default, the minimum CPU allocation for an FSS group is 1% of the total CPU. The default minimum allocation for a PSET group is one CPU. (If the `transient_groups` keyword is set to 1: FSS groups with no active SLOs are deleted and therefore use no resources; the minimum CPU allocation for PSET groups becomes 0.)

If you specify a *`min`* value that is less than 1% of the system's total CPU, the group receives the minimum allowed resource of 1%. For example, with `absolute_cpu_units` enabled in the WLM configuration on an 8-CPU system, the system's total CPU can be divided into 800 shares (100 for each CPU). A *`min`* value of 7 would be less than 1%, so WLM would automatically increase the group's CPU shares to 8 so that it represents at least 1%.

For more information on absolute CPU units, see the section "Using absolute CPU units" on page 205.

If the sum of all the `gmincpu` values is greater than the system's total CPU, the values are treated as CPU requests that are to be met before any other requests are considered. `weight` values do apply.

When configuring WLM on a partition, be sure to select a value for *`min`* that makes sense in terms of the limits placed on the partition when it was created—namely its minimum and maximum number of CPUs.

NOTE For information on the effect of this keyword in passive mode, see "Passive mode versus actual WLM management" on page 226.

Specifying a group's maximum CPU (optional)

You can assign workload groups a maximum number of CPU shares. This maximum is a hard upper limit; the group will never receive more than this maximum, as opposed to the `maxcpu` value in `slo` structures, which merely specifies the maximum CPU request the SLO controller can make. (The `OTHERS` group may receive more than its maximum if other groups have received their maximum CPU and there is still CPU left.)

To specify the maximum number of CPU shares a group receives, use the `gmaxcpu` keyword in the `prm` structure, according to the following syntax:

```
gmaxcpu = group : max [, ...];
```

where

group Is the workload group name.

max Is *group*'s maximum number of CPU shares. The value must be an integer greater than or equal to the group's `gmincpu` value.

max is out of the total CPU, which is 100 multiplied by the number of CPUs (if you set the tunable `absolute_cpu_units` to 1 or it is implied by other elements in your configuration)—or just 100 by default.

If you specify a *max* greater than the total CPU, WLM treats it as equal to the total CPU.

To have a maximum of 4 CPUs on an 8-CPU system with `absolute_cpu_units` enabled (`absolute_cpu_units = 1`), you would specify:

```
gmaxcpu = 400;
```

For more information on absolute CPU units, see the section "Using absolute CPU units" on page 205.

There is no default `gmaxcpu` value for a group. If you do not specify a `gmaxcpu` value, WLM takes the system's total CPU, in shares, as the `gmaxcpu` value.

When configuring WLM on a partition, be sure to select a value for *max* that makes sense in terms of the limits placed on the partition when it was created—namely its minimum and maximum number of CPUs.

NOTE For information on the effect of this keyword in passive mode, see "Passive mode versus actual WLM management" on page 226.

Weighting a group so it gets more CPU (optional)

The larger a CPU weight value you assign a group, the more CPU it receives when there is not enough CPU to satisfy all requests at a given priority level. A larger weight value also gets a group more CPU when all SLOs can be satisfied and there is excess CPU to be distributed. For this extra CPU to go to your workload groups, you must set the `distribute_excess` tunable in your configuration file. This tunable is described in the section "Distributing excess CPU to your workloads (optional)" on page 206. If you do not set this tunable, all the excess CPU goes to the default workload group `OTHERS`.

To specify a group's CPU weight, use the `weight` keyword in the `prm` structure, according to the following syntax:

```
weight = group : wt [, ...];
```

where

group	Is the workload group name.
wt	Is *group*'s CPU weight. The value must be an integer greater than or equal to 1. The default weight for a group is 1.

NOTE Weighting a group is beneficial only if the group is going to have an active SLO.

With weighted groups, WLM attempts to allocate CPU in the same weight-to-allocation ratio for each group. However, there are two exceptions, noted below.

A workload group's allocation supplied at higher priority levels:

- Is not reduced to achieve the weight-to-allocation ratio. Otherwise, weights would override SLO priorities, rendering them ineffective.
- Is not increased if it is greater than or equal to the SLO's requested CPU allocation—until all SLOs at the same priority are also satisfied.

Consider the following example. The Allocation column represents the allocations resulting from higher priority levels. The Requested allocation column represents the allocations the SLOs at the current priority are requesting.

In this example, WLM takes the amount of CPU to be distributed, in this case 60%, and begins allocating it according to the weights.

Table 6-1 **`weight` keyword example (weights take effect)**

Group	Allocation	Weight	Requested allocation	Final allocation
`A`	10	5	15	15
`B`	20	2	60	20
`C`	10	5	70	25
`OTHERS`	0	8	80	40

The sum of the weights is 20. With a weight of 5, both `A` and `C` should get 25% (5/20) of the total CPU. This translates to 15 additional CPU shares each. Similarly, `B` and `OTHERS`, with weights of 2 and 8, should get 10% and 40% of the total CPU, respectively.

However, we must take the exceptions mentioned above into account before distributing the CPU. Considering our exceptions, `B` already has twice the shares it would be entitled to based purely on weight. Consequently, `B` gets no more CPU.

Recall that WLM attempts to equally satisfy all SLOs at a given priority by allocating CPU in the same weight-to-allocation ratio.

With `B` satisfied, we focus on groups `A`, `C`, and `OTHERS`. The weight-to-allocation ratio for `A` and `C` is 5/10, or 1/2. The ratio for `OTHERS` is undefined because it currently has no allocation. Consequently, WLM first allocates shares to `OTHERS` to bring its weight-to-allocation ratio in line with the ratios of `A` and `C`. To match the ratio of 1/2, WLM grants `OTHERS` 16 shares so that its ratio is 8/16, or 1/2, as well.

This leaves 44 excess shares. WLM allocates 5 each to groups `A` and `C`, then 8 to `OTHERS`, maintaining the weight-to-allocation ratio. At this point, both `A` and `B` have 15 shares each and `OTHERS` has 24 shares. Group `A`’s SLO request is now satisfied. WLM will not allocate more shares to `A` until the other SLO requests are fulfilled.

WLM then allocates the remaining 26 shares to groups `C` and `OTHERS` based on their weights. Thus, `C` gets an additional 10 and `OTHERS` gets 16 for a total of 25 and 40 shares, respectively.

Interaction between `weight` and `distribute_excess`

Now consider how the `weight` keyword and the `distribute_excess` tunable interact. When set, the `distribute_excess` tunable causes excess CPU to go to the workload groups you define, as opposed to the default workload group `OTHERS`. In our example in Table 6-1 in the previous section, there is not enough CPU to satisfy all the requests. As such, there is no extra CPU to distribute—and specifying `distribute_excess` would not affect the outcome.

Consider the example in Table 6-2. The requested allocation for each group is 20, and there are no allocations from SLOs at higher priorities. Based on the weights, the allocations would be 50, 25, and 25 for `A`, `B`, and `C`, respectively. However, whenever the weight-based allocations are larger than the requests, the requests are used first. Thus, `A`, `B`, and `C` all receive a CPU allocation of 20. That leaves 40% of the CPU unallocated. With `distribute_excess` not set, all 40% goes to `OTHERS`.

Table 6-2 **`weight` example with `distribute_excess = 0` (off)**

Group	Weight	Requested allocation	Final allocation
`A`	2	20	20
`B`	1	20	20
`C`	1	20	20
`OTHERS`	1 (default)	1 (default)	40

Table 6-3 shows the same example, but with `distribute_excess` set. Again, `A`, `B`, and `C` all receive a CPU allocation of 20, leaving 40% of the CPU unallocated. This time though, `distribute_excess` is set. Consequently, the remaining 40% (less the 1% default to `OTHERS`) goes to `A`, `B`, and `C`—based on their weights. Dividing the weight of `A`, 2, by the sum of the weights, 4, A gets 50% of the total CPU, while B and C get 25%. WLM removes 1% from B to give to OTHERS so that it maintains the minimum CPU percentage required for all groups.

Table 6-3 **`weight` example with `distribute_excess = 1` (on)**

Group	Weight	Requested allocation	Final allocation
A	2	20	50
B	1	20	24
C	1	20	25
OTHERS	1 (default)	1 (default)	1

Specifying a group's minimum memory (optional)

You can assign workload groups a minimum percentage of the system's memory. This minimum is a hard lower limit; the workload group receives less than this minimum only if it has no active SLOs and `transient_groups` is set to 1.

NOTE When specifying a workload group's minimum memory, be very careful of resource interactions, which are explained in "How resource allocations interact" on page 447. In particular, be sure the workload groups get the memory they need. (Be careful to set values for `gminmem` and `gmemweight` so that the groups get enough memory even when all groups are active.)

To specify the minimum percentage a group receives, use the `gminmem` keyword in the `prm` structure, according to the following syntax:

```
gminmem = group : min [, ...];
```

where

group	Is the workload group name. If you specify `gminmem`, `gmaxmem`, or `memweight` for any group in your configuration, be sure to specify a `gminmem` for any group, including `OTHERS`, for which 1% of the memory is not sufficient for its users and applications. You cannot specify the `PRM_SYS` group in a `gminmem` statement.
min	Is *group*'s minimum percentage of memory. The value must be an integer between 1 and the group's `gmaxmem` value, inclusive. The default *min* value is 1.

Specifying a group's maximum memory (optional)

You can assign workload groups a maximum percentage of memory. This maximum is a hard upper limit, except for the `OTHERS` group. (The `OTHERS` group may receive more than its `gmaxmem` if all other groups have received their `gmaxmem` and there is still memory left.)

NOTE When specifying a workload group's maximum memory, be very careful of resource interactions, which are explained in "How resource allocations interact" on page 447. In particular, be sure the workload groups get the memory they need. (Be careful to set values for `gminmem` and `gmemweight` so that the groups get enough memory even when all groups are active.)

To specify the maximum percentage of memory a group receives, use the `gmaxmem` keyword in the `prm` structure, according to the following syntax:

```
gmaxmem = group : max [, ...];
```

where

group Is the workload group name.

You cannot specify the `PRM_SYS` group in a `gmaxmem` statement.

max Is *group*'s maximum percentage of memory. The value must be an integer between the group's `gminmem` value and 100, inclusive.

The default *max* value is 100.

Weighting a group so it gets more memory (optional)

The larger a memory weight value you assign a group, the more memory it receives when there is not enough memory to satisfy all `gmaxmem` requests.

To specify a group's memory weight, use the `memweight` keyword in the `prm` structure, according to the following syntax:

```
memweight = group : wt [, ...];
```

where

group Is the workload group name.

You cannot specify the `PRM_SYS` group in a `memweight` statement.

wt Is *group*'s memory weight. The value must be an integer greater than or equal to 1. If you do not specify a memory weight, WLM uses the group's CPU weight, which defaults to 1 if not specified.

NOTE Assigning a `memweight` to a PSET workload group is not useful because WLM grants the group memory when the configuration is activated, but never adjusts its memory allocation.

Consider the following example. We have four groups, with one being a PSET group. The `gminmem` serves as an allocation for the groups because each group gets the amount of memory represented by its `gminmem` value.

Table 6-4 **`memweight` example**

Group	Allocation (`gminmem`)	Weight	Final allocation
`A` (PSET)	10	Unspecified	10
`B`	30	3	42
`C`	20	2	28
`OTHERS`	20	1	20

First, `A` gets its `gminmem` because it is a PSET group. That leaves 90% of the memory to distribute. Next, `OTHERS` is granted its memory. It has a `memweight` of 1, and the sum of the `memweight` values is 6. As a result, `OTHERS` gets 1/6 * 90, or 15%. However, 15 is less than its `gminmem`, so it gets its `gminmem` of 20 instead. Now there is 70% left. However, WLM guarantees the `gminmem` for the groups `B` and `C`, 30 and 20 respectively. So, there is only 20% to distribute between the two groups `B` and `C`. This 20% is distributed so that the groups' final allocations are proportional to their weights.

Defining SLOs

Use `slo` structures to specify your service-level objectives and assign priorities to those objectives.

You define `slo` structures for a host partition or based on the workload groups you defined in the `prm` structure. For information on specifying the `prm` structure, see “Defining the PRM components” on page 139.

Defining an `slo` structure consists of the following tasks:

- Specifying the SLO name (required)
- Specifying the priority (required)
- Specifying the workload group to which the SLO applies (optional)
- Specifying the lower and upper bound requests on CPU (optional)
- Specifying a goal (optional)
- Specifying a shares-per-metric allocation request (optional)
- Specifying when the SLO is active (optional)

An `slo` structure can take one of two forms. Both forms require a priority. The first form may or may not have a goal.

```
slo slo_name {
   pri = priority;
   [ entity = PRM group group_name; ]
   [ mincpu = lower_bound_request; ]
   [ maxcpu = upper_bound_request; ]
   [ goal = goal_expression; ]
   [ condition = condition_expression; ]
   [ exception = exception_expression; ]
}
```

The second form of an `slo` structure uses the `cpushares` statement. This statement cannot be used with a `goal` statement.

```
slo slo_name {
   pri = priority;
   [ entity = PRM group group_name; ]
   cpushares =  value { more | total } [ per metric met [ plus offset ] ];
   [ mincpu = lower_bound_request; ]
   [ maxcpu = upper_bound_request; ]
   [ condition = condition_expression; ]
   [ exception = exception_expression; ]
}
```

Here are several example `slo` structures:

```
slo buying {
   pri = 1;
   mincpu = 50;
   maxcpu = 200;
   goal = metric stock_price_1 < 50;
   condition = 09:00 - 16:00;
   exception = Sat - Sun;
}

slo selling {
   pri = 1;
   mincpu = 50;
   maxcpu = 300;
   goal = metric stock_price_2 > 75;
   condition = 09:00 - 16:00;
   exception = Sat - Sun;
}
```

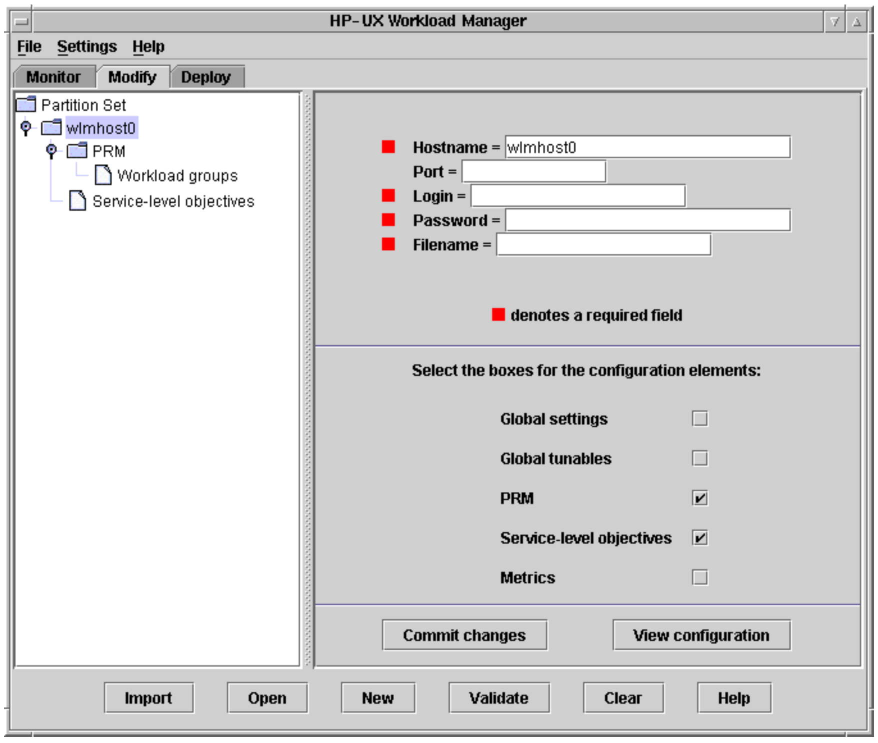
Defining SLOs using the WLM GUI

In addition to defining an SLO using a text editor, you can use the WLM Configuration Wizard (/opt/wlm/bin/wlmcw). Alternatively, you can use the WLM GUI (/opt/wlm/bin/wlmgui).

The following steps show how to define an SLO with a usage goal for an existing workload group using the GUI. It is assumed you went through the procedure on in the section “Defining PRM components using the WLM GUI” on page 141 to create a workload group. For more information on using the WLM GUI, see its online help.

Step 1. Select the name of the host, wlmhost0 in this case, in the left pane. The right pane allows you to set the type of configuration elements you would like to see in the new configuration.

Select the “Service-level Objectives (SLOs)” box in the right pane.



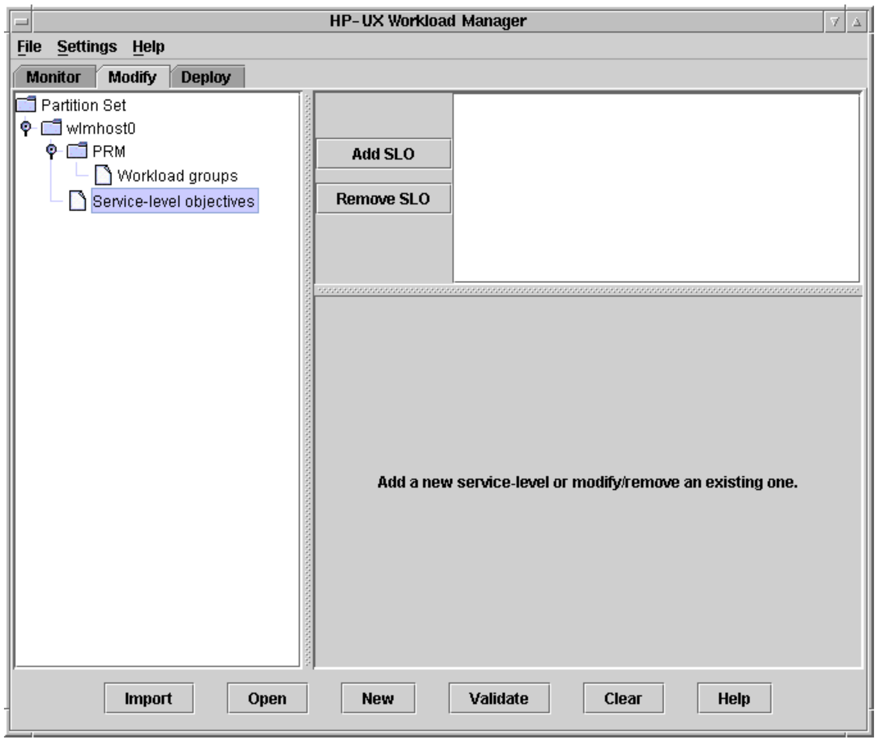
🟥 Hostname = wlmhost0
Port =
🟥 Login =
🟥 Password =
🟥 Filename =

🟥 denotes a required field

Select the boxes for the configuration elements:

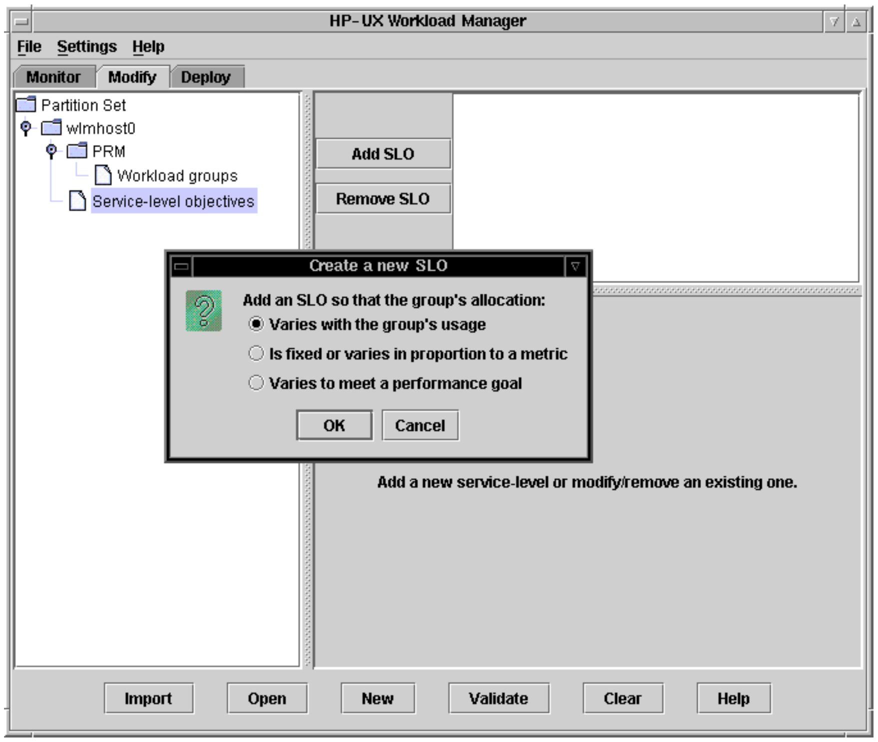
Global settings ☐
Global tunables ☐
PRM ✓
Service-level objectives ✓
Metrics ☐

Step 2. Select the “Service-level objectives” item in the left pane.



Add a new service-level or modify/remove an existing one.

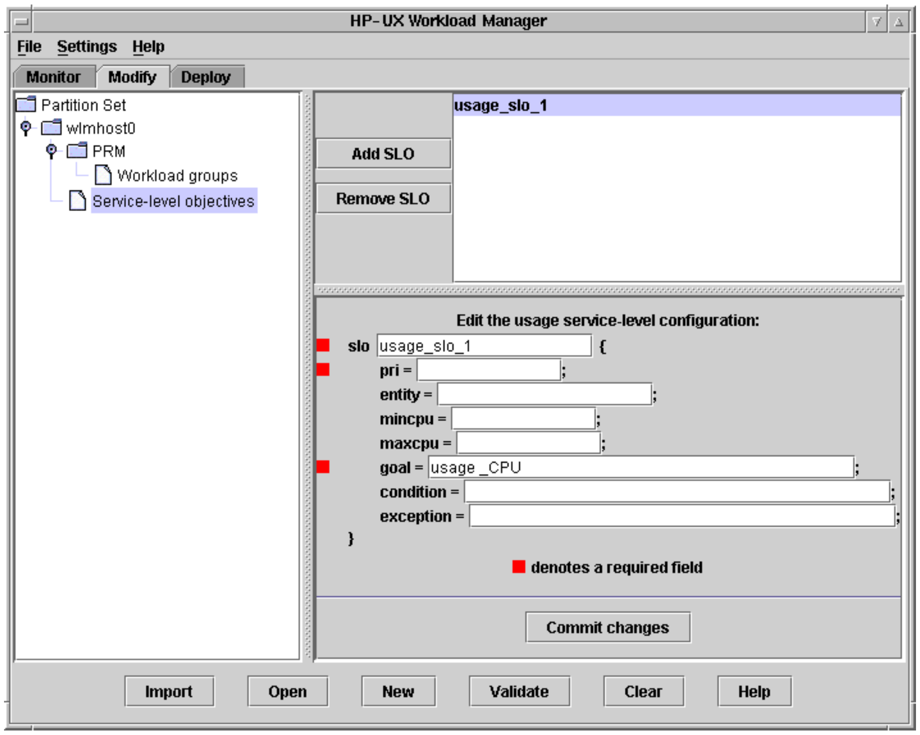
Step 3. Select the **[Add SLO]** button. A new dialog appears. The default option is “Varies with the group’s usage”. This option is the usage goal. Select the **[OK]** button.



Add an SLO so that the group's allocation:
[✓ Varies with the group's usage]
Is fixed or varies in proportion to a metric
Varies to meet a performance goal

Add a new service-level or modify/remove an existing one.

Step 4. The right pane changes, allowing you to define the SLO. The goal field defines the usage goal.

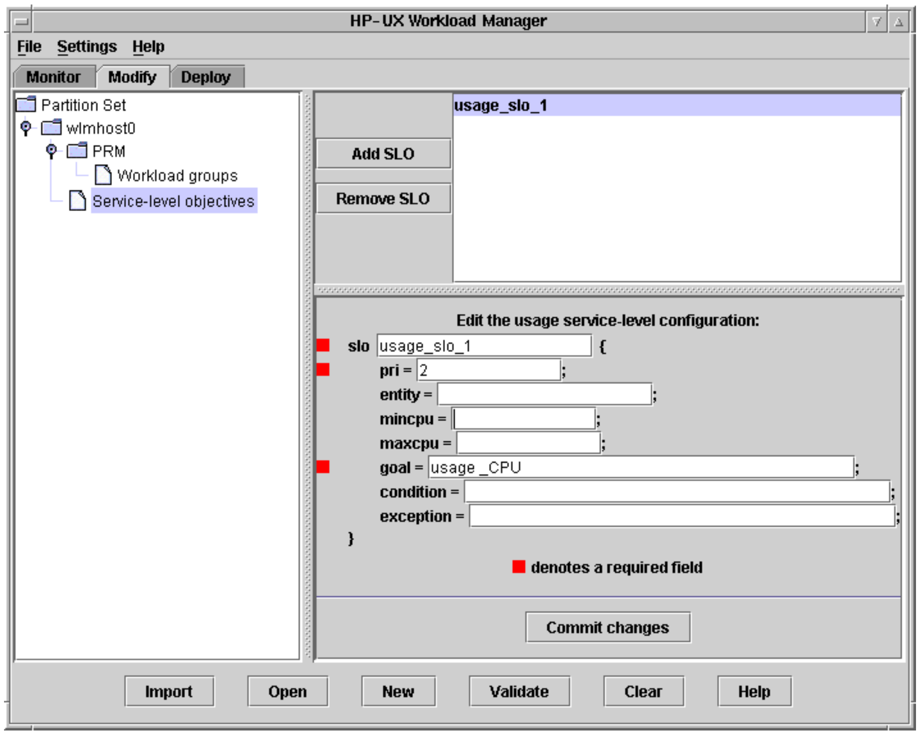


Edit the usage service-level configuration:

```
🟥 slo usage_slo_1 {
🟥     pri = ;
       entity = ;
       mincpu = ;
       maxcpu = ;
🟥     goal = usage _CPU;
       condition = ;
       exception = ;
   }
```

🟥 denotes a required field

Step 5. Fill in the fields as desired.



Edit the usage service-level configuration:

🟥 slo usage_slo_1 {
🟥 pri = 2;
entity = ;
mincpu = ;
maxcpu = ;
🟥 goal = usage _CPU;
condition = ;
exception = ;
}

🟥 denotes a required field

Step 6. Select the **[Commit changes]** button to save the current configuration in memory. To save changes to disk, go to the Deploy tab. (The file is saved to the file specified in the filename field in the Modify tab when the system is selected in the left pane.)

The above steps highlight only one use of the WLM GUI. For additional information, see the GUI's online help.

Specifying the SLO name (required)

The SLO name can consist of the following characters:

- Uppercase letters (A-Z)
- Lowercase letters (a-z)
- Digits (0-9)
- The underscore character (_), as long as it is not the first character
- The hyphen character (-)
- The period (.)
- Quoted characters

  Any characters not listed above (except the double quote) can be used as long as they are enclosed in double quotes. The slash character (/) is not allowed, even when quoted.

The SLO name cannot exceed 216 characters.

Specifying the priority (required)

WLM uses SLO priorities to determine CPU allocation when the combined CPU requests of all SLOs exceed 100%. In these cases, SLOs with higher priorities (priorities closer to 1) are granted CPU first. Use the `pri` keyword to assign priorities.

The `pri` keyword is required.

Specify an SLO's priority using the following syntax:

```
pri = priority;
```

where

priority	Is an integer greater than or equal to 1. The value of 1 is highest priority.

If there are any CPU resources remaining after WLM has met all the requests, the remainder is given to the `OTHERS` group by default. You can ensure that this remainder is given to more critical workload groups by using stretch goals—assigning a high and a low priority to each workload group and setting the low priority SLO to be more aggressive than the high priority SLO. Consequently, the low priority SLOs seek out the excess CPU. For more information on stretch goals, see "Goals vs stretch goals" on page 187.

Table 6-5 illustrates the idea.

Table 6-5 Capturing remaining CPU resources with stretch goals

Workload group	SLO priority
`Sales`	1
`Finance`	2
`Finance`	3
`Sales`	4

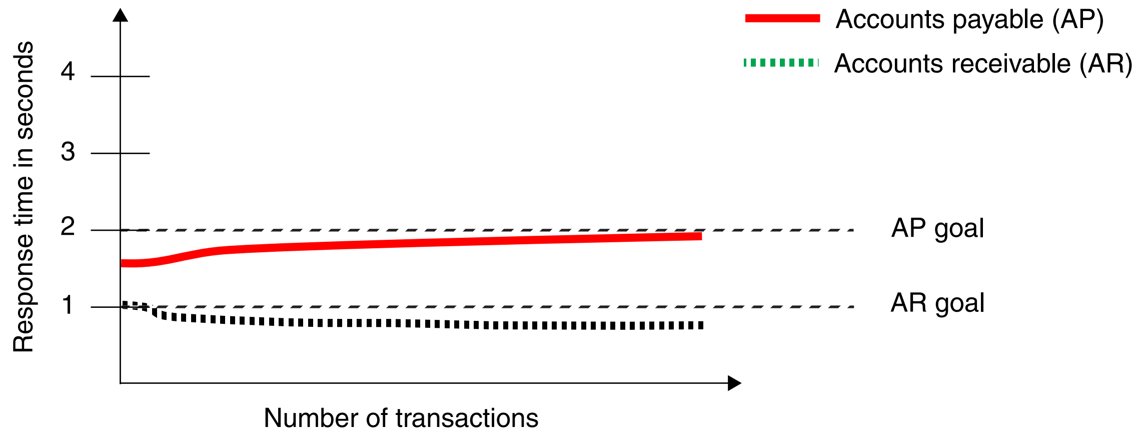
`Sales` and `Finance` are the only workload groups in the configuration. After the priority 1 and 2 SLOs are met, the remaining CPU goes to `Finance`. If there is any CPU left at that point, it goes to `Sales`.

You can also use any remaining CPU by setting the `distribute_excess` tunable, as explained in the section “Distributing excess CPU to your workloads (optional)” on page 206.

You can assign multiple `slo` structures the same priority. However, if all the SLOs have the same priority and there are not enough resources, some or all of the SLOs will not be met consistently. Thus, you must determine which SLOs must be met and assign priorities accordingly.

Consider the example below which has two workloads, where both have SLOs at the same priority. Fortunately, the server has the resources to meet both workloads’ goals, as shown in Figure 6-3.

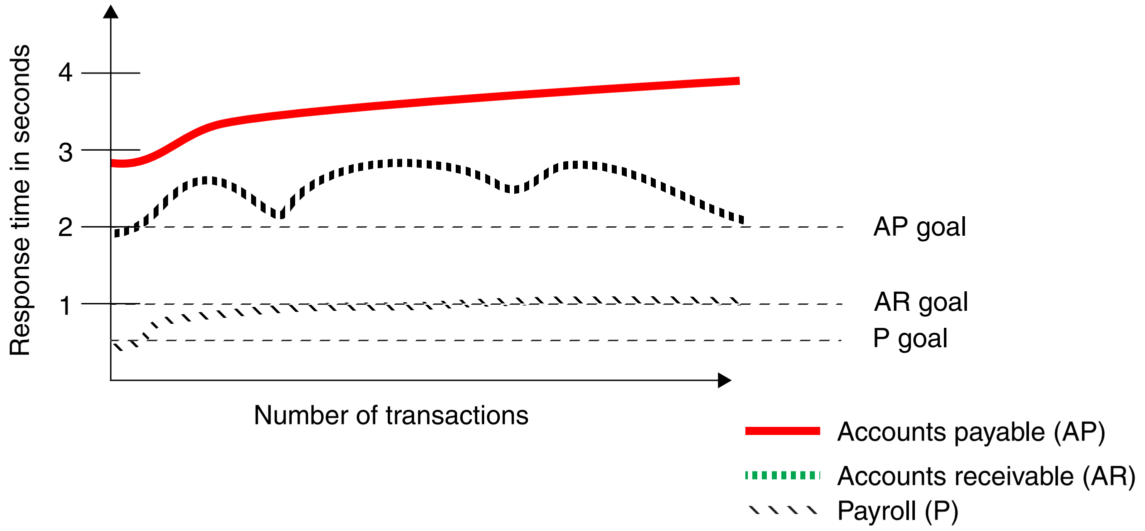
Figure 6-3 **Workloads with SLOs at same priority**



Now add a third workload named Payroll with an SLO at the same priority as those for the other workloads. This workload's SLO is to calculate each employee's overtime, commissions, bonuses, taxes, and total pay in less than 0.5 seconds. The system is now overloaded and cannot meet each SLO.

Because each SLO has the same priority, WLM cannot allocate enough resources to any one workload group to help it achieve its SLO. Consequently, no SLO is met, as shown in Figure 6-4.

Figure 6-4 **Additional workload with same priority SLO**



In this case, you should assign the SLOs different priorities so that WLM can determine how to allocate the limited system resources, allowing at least one SLO to be met. Other options are to relax the SLOs or purchase more hardware resources, if the applications are consistently in need of more resources.

Specifying the workload group to which the SLO applies (optional)

SLOs govern workloads that are based on workload groups. Therefore, each workload group must be assigned an SLO. If you want an SLO for a single application, place that application in a workload group by itself.

NOTE If you plan on managing only virtual partitions or nPartitions—with no FSS groups or PSETs inside, the `entity` keyword is not needed.

Specify the workload group to which an SLO applies using the `entity` keyword. The `entity` keyword is required. Use the following syntax:

```
entity = PRM group group_name;
```

where

group_name	Is a workload group name.

NOTE You cannot specify `PRM_SYS` in an `slo` structure.

Specifying the lower and upper bound requests on CPU (optional)

Specify lower and upper bound requests to prevent an SLO's controller from:

- Requesting so little CPU that the associated workload cannot perform reasonably well
- Requesting so much CPU that other workloads cannot perform reasonably well

Use the `mincpu` and `maxcpu` keywords to state the minimum and maximum allocations that the SLO's controller can request. Because these allocations are merely requests, the minimum allocation values across all SLOs do not have to sum to 100%. (For information on setting hard CPU limits, see "Specifying a group's minimum CPU (optional)" on page 159 and "Specifying a group's maximum CPU (optional)" on page 160.)

WLM grants requests based on the priority of the SLOs, their current performance, and available resources.

NOTE For information on the effect of these keywords in passive mode, see "Passive mode versus actual WLM management" on page 226.

Specify the lower and upper bounds on CPU for an SLO using the following syntax:

```
mincpu = lower_bound_request;
maxcpu = upper_bound_request;
```

where

lower_bound_request

Is an integer from 0 to *upper_bound_request*, inclusive. *lower_bound_request* is the minimum number of CPU shares the SLO's controller can request.

If an SLO does not also contain a `cpushares` statement or a `goal` statement, the `mincpu` value is used as the SLO's shares request. If an SLO does not contain a `mincpu` statement, 0 is used as the SLO's shares request—although a `gmincpu` value or `hmincpu` value may keep the allocation from actually being 0 shares.

lower_bound_request is out of the total CPU, which is 100 multiplied by the number of CPUs (if you set the tunable `absolute_cpu_units` to 1 or it is implied by other elements in your configuration)—or just 100 by default.

To have a minimum request of 2 CPUs on an 8-CPU system with `absolute_cpu_units` enabled (`absolute_cpu_units = 1`), you would specify:

```
mincpu = 200;
```

If you specify a *lower_bound_request* greater than the total CPU, WLM treats it as equal to the total CPU.

When configuring WLM on a partition, be sure to select a value for *lower_bound_request* that makes sense in terms of the limits placed on the partition when it was created—namely its minimum and maximum number of CPUs.

NOTE

The *lower_bound_request* value is not a hard limit: Higher priority SLOs may consume all CPU cycles before all SLOs are granted CPU cycles. However, the associated workload group's `gmincpu` value is a hard limit. It serves as the group's minimum CPU allocation when it represents more CPU resources than the `mincpu` values used in the group's SLOs. The `gmincpu` value is also used to determine the group's minimum CPU allocation in SLOs with `cpushares` statements but no `mincpu` statements. For information on the default `gmincpu` value, see the `gmincpu` section "Specifying a group's minimum CPU (optional)" on page 159. Similarly, `hmincpu` is a hard limit for allocation to a host.

`upper_bound_request`

Is an integer greater than or equal to *`lower_bound_request`*. *`upper_bound_request`* is the maximum number of CPU shares the SLO's controller can request.

`upper_bound_request` is out of the total CPU, which is 100 multiplied by the number of CPUs (if you set the tunable `absolute_cpu_units` to 1 or it is implied by other elements in your configuration)—or just 100 by default.

If you specify an *`upper_bound_request`* greater than the total CPU, WLM treats it as equal to the total CPU.

To have a maximum request of 3 CPUs on an 8-CPU system with `absolute_cpu_units` enabled (`absolute_cpu_units = 1`), you would specify:

```
maxcpu = 300;
```

When configuring WLM on a partition, be sure to select a value for *`upper_bound_request`* that makes sense in terms of the limits placed on the partition when it was created—namely its minimum and maximum number of CPUs.

NOTE

An *`upper_bound_request`* may be ignored if the associated workload group's CPU resources are already limited by the group's `gmaxcpu` value. In the case of

SLOs with `cpushares` statements but no `maxcpu` statements, the group’s `gmaxcpu` value is still used to limit CPU allocation. For information on the default value of `gmaxcpu`, see the `gmaxcpu` section “Specifying a group’s maximum memory (optional)” on page 166. Similarly, `hmaxcpu` can limit CPU allocation to a host.

If *`lower_bound_request`* and *`upper_bound_request`* are equal, WLM attempts to grant that exact amount of CPU; consequently, specifying a goal would not be particularly useful because WLM would not change the SLO’s CPU allocation to better achieve a goal.

For more information on absolute CPU units, see the section “Using absolute CPU units” on page 205.

Specifying a goal (optional)

An SLO’s goal specifies either:

- Whether the workload’s performance should be less than or greater than some specified floating-point value
- A range for a workload’s CPU utilization

Use the `goal` keyword to specify such goals.

The `goal` keyword is optional. If neither the `goal` nor `cpushares` keyword is specified, the SLO is allocated CPU according to its `mincpu` setting. For information on setting `mincpu`, see “Specifying the lower and upper bound requests on CPU (optional)” on page 180.

You cannot specify both a `goal` statement and a `cpushares` statement in the same SLO. Similarly, you cannot have a workload group with one SLO that has a `goal` statement and another SLO that has a `cpushares` statement that includes `more`.

Specify the goal of an SLO using the following syntax:

```
goal = goal_expression;
```

where

goal_expression

Indicates either a metric goal (performance goal) or a usage goal. Use a metric goal when you have performance data for the workload. Use a usage goal to ensure a workload group uses a certain percentage of its allocated CPU. Usage goals are especially beneficial when you have a workload that cannot provide performance data.

Specify a metric goal as follows:

`goal = metric` *`met relation value`*`;`

where

met	Is a text string (no longer than 255 characters) naming the metric to be used for the SLO. The *met* string must also be specified in a `tune` structure that indicates the path to your data collector executable that provides the metric. *met* cannot contain the slash character (/) or start with an underscore (_). For information on `tune` structures, see “Tuning the metrics and the SLOs” on page 196.	
relation	<	If *met* is supposed to be less than *value*.
	>	If *met* is supposed to be greater than *value*.
value	Is a numeric goal value, either integer (greater than or equal to 1) or floating-point number, not expressed in exponential notation.	

For example, here is a performance goal statement:

```
goal = metric response_time < 3.0;
```

Now consider a usage goal. With this type of goal, WLM adjusts a workload group’s allocation to more efficiently match the workload’s actual CPU usage. For example, if a workload is using 20 CPU shares and has a 50-share allocation, its utilization is 20/50 or 40%. WLM attempts to keep a workload’s utilization between a low and high bound. WLM tracks the usage metric internally, on a workload-group basis; you do not have to provide it.

NOTE Typically, you should set `wlm_interval` to 5 when your WLM configuration contains a usage goal.

Specify a usage goal in an `slo` structure as follows:

goal = usage _CPU [*low_util_bound* [*high_util_bound*]];

where

low_util_bound

Is an integer less than or equal to *high_util_bound*, but greater than or equal to 0. WLM attempts to keep the utilization percentage above this threshold value, which is 50 by default. It does so by reducing the requested allocation whenever utilization drops below this value.

NOTE Setting *low_util_bound* at or near 0 can result in an SLO that never reduces its requested share allocation. This results in the SLO not giving up its unused shares.

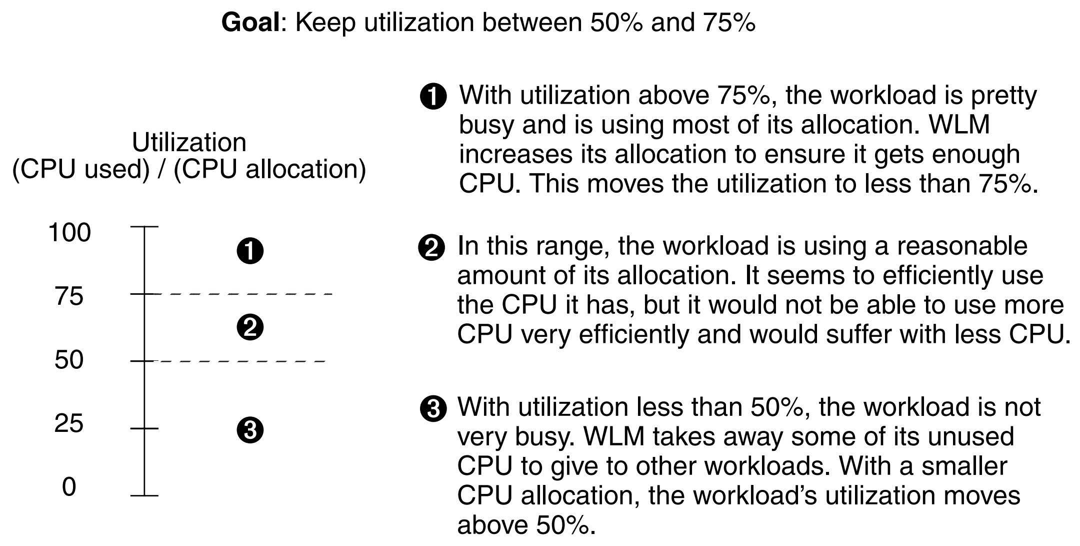
high_util_bound

Is an integer greater than or equal to *low_util_bound*, but less than or equal to 100. WLM attempts to keep the utilization percentage under this threshold value. It does so by increasing the requested allocation whenever utilization surpasses this value. When not specified, this value defaults to the specified *low_util_bound*; when both are not specified, this value defaults to 75.

NOTE Setting *high_util_bound* at or near 100 can result in an SLO that never requests a larger allocation.

Figure 6-5 shows conceptually how WLM works to keep CPU utilization within a range. The goal here is to keep utilization in the default range, 50% to 75%.

Figure 6-5 Usage goal conceptually



Goal: Keep utilization between 50% and 75%

❶ With utilization above 75%, the workload is pretty busy and is using most of its allocation. WLM increases its allocation to ensure it gets enough CPU. This moves the utilization to less than 75%.

❷ In this range, the workload is using a reasonable amount of its allocation. It seems to efficiently use the CPU it has, but it would not be able to use more CPU very efficiently and would suffer with less CPU.

❸ With utilization less than 50%, the workload is not very busy. WLM takes away some of its unused CPU to give to other workloads. With a smaller CPU allocation, the workload's utilization moves above 50%.

Here is an example of a usage `goal` statement:

```
goal = usage _CPU 60 80;
```

When WLM requests an allocation change, the request is proportional to how far the current utilization is below *`low_util_bound`* or above *`high_util_bound`*. Use `cntl_kp` or `cntl_convergence_rate` in a `tune` structure with `_CPU_`*`group_name`* as the metric value to adjust the proportion, causing the utilization to move into the desired range more quickly or slowly. For example, a `cntl_kp` value of 10 will cause the utilization to move in larger jumps than will a value of 0.1.

To get an idea of how such a `tune` structure would look, consider a workload group named `buy`. Its `tune` structure would look like the following:

```
tune _CPU_buy {
   cntl_kp = 0.1;
}
```

The `cntl_kp` value must be between 0 and 1,000,000 inclusive. The default value is 1.

For more information on `cntl_kp` or `cntl_convergence_rate`, see the section "Tuning a workload group's SLO convergence: `cntl_kp` (optional)" on page 212.

Goals *vs* stretch goals

You can specify one or more goals for a single workload group. The highest priority goal should represent the minimum acceptable service level. All other goals should indicate stretch goals—goals that may be hard to achieve, but are desired if possible. Assign stretch goals lower priorities than the primary goal.

Think of goals as "needs" and stretch goals as "wants". Needs (goals) are required to complete the task at the desired performance level. Wants (stretch goals) form a superset of the needs and may increase performance, but only at the sacrifice of the needs of other services. Consider this trade-off when assigning priorities to your goals and stretch goals.

The following example shows a goal at priority 1 with a desired response time of less than 2.0. The stretch goal, with a priority of 5, is for the response time to be less than 1.0. This stretch goal may be met, but only when the CPU requests of all SLOs at higher priorities are being met.

```
# This is an SLO for the finance group.
slo finance_query {
   pri = 1;
   mincpu = 20;
   maxcpu = 50;
   entity = PRM group finance;
   goal = metric fin_app.query.resp_time < 2.0;
   condition = Mon - Fri;
}

# This is a stretch goal for the finance group. If all other
# goals of higher priority have been met, apply more CPU to group
# finance, so its application runs faster.
slo finance_query_stretch {
   pri = 5;
   mincpu = 20;
   maxcpu = 80;
   entity = PRM group finance;
   goal = metric fin_app.query.resp_time < 1.0;
   condition = Mon - Fri;
}
```

Specifying a shares-per-metric allocation request (optional)

An SLO can directly express an allocation request using the `cpushares` keyword. This keyword allows you to make allocation requests of the form "*x* shares of the CPU for each metric *y*". For example, you could give an Oracle instance *n* CPU shares for each process in the instance.

The `cpushares` syntax is:

```
cpushares =  value { more | total } [ per metric met [ plus offset ] ];
```

where

`cpushares` Is an optional keyword for making CPU allocation requests. Using the variables in the syntax statement shown above, a request—which we will call *request_value*—equates to one of the following forms:

value
value * *met*
value * *met* + *offset*

A request is additive when using the `more` keyword and absolute when using the `total` keyword.

You cannot specify both a `goal` statement and a `cpushares` statement in the same SLO. Similarly, you cannot have a workload group with one SLO that has a `goal` statement and another SLO that has a `cpushares` statement that includes `more`.

When you specify the `cpushares` keyword, the `mincpu` and `maxcpu` keywords are optional.

value Specifies the number of CPU shares to request for the associated workload group. This value is either an integer or a floating-point number, not expressed in exponential notation.

`more` Makes additive allocation requests by adding the *request_value* to other additive requests at the same priority and to any requests from higher priority SLOs. If there are no higher priority SLOs, the request is added to the workload group's `gmincpu` value, which is 1% of the system's total CPU resources by default.

request_value is bounded by the SLO's *mincpu* and *maxcpu* values, if specified.

If the workload group can get a larger allocation from an SLO with an absolute allocation request at that priority, it does so. This absolute request can come from an SLO that uses `cpushares` with `total` or from an SLO that uses only the `mincpu` and `maxcpu` keywords.

`total`	Makes absolute allocation requests starting from 0. The request is exactly equal to *`request_value`* (discussed in the `cpushares` keyword description), within the bounds formed by the SLO's `mincpu` and `maxcpu` values, if specified. If the associated workload group can get a larger allocation of shares from higher priority SLOs or from other SLOs at the same priority making absolute allocation requests, it does so.
`per metric`	Is an optional clause. Using `per metric` forms an allocation request by multiplying *`value`* by the metric *`met`* in *`request_value`* (discussed in the `cpushares` keyword description).
`met`	Is the text string naming the metric to be used. The metric string must also be specified in a `tune` structure that indicates the path to the data collector executable providing the metric. *`met`* cannot contain the slash (/) character, start with an underscore (_), or be more than 255 characters.
`plus`	Is an optional clause, for use only with the `per metric` clause, that allows you to increase or decrease the number of CPU shares to be requested as part of the *`request_value`*, which is discussed in the `cpushares` keyword description.
`offset`	Is a floating-point value (either positive or negative) specifying the number of shares to use as an *`offset`* in the *`request_value`* (discussed in the `cpushares` keyword description). Consider using `absolute_cpu_units` (discussed in the *`value`* description above) to minimize the effects of a system's variable number of CPUs on your *`offset`* value.

Here are some *request_value* expressions and the corresponding cpushares statements:

```
request_value = 25

               cpushares = 25 total;

request_value = 10 * X + 30

               cpushares = 10 total per metric X plus 30;
```

Consider the following example of an additive allocation request:

```
slo additive_example {
    pri = 1;
    mincpu = 0;
    maxcpu = 50;
    entity = PRM group App1;
    cpushares = 5 more per metric application_procs;
}
```

Assume this is App1's only SLO at this priority. Also assume gmincpu is 20 and application_procs (the number of processes the application has running) is 3. The SLO requests 35 CPU shares: 5 shares for each of the 3 processes the application has running plus 20 shares from gmincpu.

The next example shows an absolute allocation request. It is the same as the additive example above—except it uses total in place of more.

```
slo absolute_example {
    pri = 1;
    mincpu = 0;
    maxcpu = 50;
    entity = PRM group App1;
    cpushares = 5 total per metric application_procs;
}
```

Again, assume application_procs is 3. This time, because it is an absolute request, the request starts from 0, not the previous allocation request. Consequently, the request is 5 CPU shares for each of the 3 application processes in workload group App1 for a total of 15. However, the request is ignored because the group already has 20 shares because of its gmincpu setting.

Specifying when the SLO is active (optional)

Use the keywords `condition` and `exception` to indicate when the SLO is active. The values of these keywords form Boolean expressions. The SLO is active when the `condition` expression is true, and the `exception` expression is false. By default, the expressions are set so that the SLO is always active.

The `condition` and `exception` keywords are optional.

If a workload group has no active SLOs, it is allocated:

- 1% of the total CPU (for FSS groups) or one CPU (for PSET-based groups), unless it has a `gmincpu` value requesting more. Then the workload group receives an allocation equal to that value.
- 1% of the memory, unless it has a `gminmem` value requesting more. Then the workload group receives an allocation equal to that value.

NOTE If the `transient_groups` tunable is set to 1, a workload group with no active SLOs is temporarily removed (for FSS groups) or assigned 0 CPUs (for PSET-based groups)—until it has at least one active SLO. In this case, the group receives no CPU and no memory—regardless of its `gmincpu` and `gminmem` values.

Specify when the SLO is active using the following syntax:

```
condition = condition_expression;
exception = exception_expression;
```

where

`condition_expression`

Is an expression that must be true for the SLO to be active. Form *`condition_expression`* as discussed below.

NOTE Do not create a `condition` statement that attempts to detect processes in a transient group using tools such as `glance` or `ps`. Whenever the group is deleted (FSS group) or assigned zero CPUs (PSET-based group), it is impossible for the system to place processes in the group. The condition will then never detect the processes it is looking for.

`exception_expression`

Is an expression that must be false for the SLO to be active. Form *`exception_expression`* as discussed below.

Use the following components to form either of the expressions:

- `metric` *`met`* `>` *`numeric_value`*
- `metric` *`met`* `<` *`numeric_value`*
- `metric` *`met`*
- Date
- Date range

Combine components to form Boolean expressions with the operators `&&` (AND), `||` (OR), `!` (NOT). Group components with parentheses `()`.

`met` must have an associated `tune` structure. *`numeric_value`* must be a floating-point value, either positive or negative. The `metric` *`met`* expression is false if *`met`* equals 0. For all other values, the expression is true.

In a Serviceguard cluster, where workloads may move between servers, use `metric` *`met`* to inform WLM of the active workloads. Ideally, a single WLM configuration file can be used on all the servers in the cluster. For more information, see "Integrating with Serviceguard" on page 429.

Specify dates or date ranges in *`condition_expression`* and *`exception_expression`* using the following components:

`[weekday] [mm/dd/ccyy] [hh:mm]`

where

`weekday`	Is `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, or `Sun`.
`mm/dd/ccyy`	Takes `mm` values 1-12, `dd` values 1-31, and `ccyy` values as four-digit years. Use an asterisk (*) for a component to indicate that all valid values are accepted.

`hh:mm`	Is hours and minutes on a 24-hour clock. Use an asterisk (*) for `hh` and/ or `mm` to indicate that all valid values are accepted.

Specify a date range using the components explained above according to one of the following formats:

- `mm/dd/ccyy - mm/dd/ccyy`
- `hh:mm - hh:mm`
- `weekday - weekday`
- `mm/dd/ccyy hh:mm - mm/dd/ccyy hh:mm`
- `weekday hh:mm - weekday hh:mm`
- `mm/dd/* - mm/dd/*`
- `*/dd/* - */dd/*`
- `*:mm - *:mm`
- `mm/dd/* hh:mm - mm/dd/* hh:mm`
- `*/dd/* hh:mm - */dd/* hh:mm`

For formats that do not specify the `hh:mm` component, the start-time defaults to 00:00, and the end-time defaults to 23:59.

Here are some examples showing various `condition` and `exception` statements. In the examples, consecutive statements are not used together unless explicitly stated.

```
exception = Sat - Sun; # SLO is active during the week
```

```
condition = 20:00 - 22:59; # Combining these two lines,
exception = Fri - Sun;     # the SLO is active from 8pm
# to 11pm every night except Friday, Saturday, and
# Sunday
```

```
condition = metric resp_time > 2; # SLO is active only
# when resp_time is greater than 2
```

```
condition = metric num_users > 5; # SLO is active only
# when the number of users logged in is greater than 5
```

```
exception = metric metricA; # SLO is active only when
# metricA is 0
```

```
condition = (metric metricA) && !(metric metricB);
# SLO is active when metricA is nonzero,
# and metricB is zero
```

```
condition = */01/2003;
# SLO is active on the 1st of every month in the
# year 2003
```

Tuning the metrics and the SLOs

You can tune metrics and SLOs using `tune` structures. Among other features, these structures allow you to specify the data collector and its command-line arguments, the frequency at which WLM checks for new performance data and adjusts CPU allocations, and the controllers' variables.

There are three types of `tune` structures:

- Global structure

  This `tune` structure applies to all metrics and SLOs. It does not reference the name of any metric or SLO. It can be specified, at most, once in the configuration file. Its syntax is:

```
tune {
   [ wlm_interval = number_of_seconds; ]
   [ absolute_cpu_units = 0_or_1; ]
   [ distribute_excess = 0_or_1; ]
   [ transient_groups = 0_or_1; ]
   [ wlmdstats_size_limit = number_of_megabytes; ]
   [ coll_argv = data_collector_and_arguments; ]
   [ coll_stderr = file; ]|
   [ cntl_smooth = smoothing_value; ]
   [ cntl_avg = 0_or_1; ]
   [ cntl_kp = proportional_term; ]
   [ cntl_convergence_rate = number_of_shares; ]
   [ cntl_margin = margin_value; ]
   [ cntl_base_previous_req = 0_or_1; ]
}
```

- Metric-specific structure

  A metric-specific `tune` structure applies to all SLOs using *`metric`*. This structure can be specified, at most, once per metric. Its syntax is:

```
tune metric {
   [ coll_argv = data_collector_and_arguments; ]
   [ coll_stderr = file; ]
   [ cntl_smooth = smoothing_value; ]
   [ cntl_avg = 0_or_1; ]
   [ cntl_kp = proportional_term; ]
   [ cntl_convergence_rate = number_of_shares; ]
   [ cntl_margin = margin_value; ]
   [ cntl_base_previous_req = 0_or_1; ]
}
```

- Metric/SLO-specific structure

  This `tune` structure applies to the specified metric/SLO pair. It can be specified, at most, once per metric/SLO pair. Its syntax is:

```
tune metric slo_name {
   [ cntl_kp = proportional_term; ]
   [ cntl_convergence_rate = number_of_shares; ]
   [ cntl_margin = margin_value; ]
   [ cntl_base_previous_req = 0_or_1; ]
}
```

In cases where two or more `tune` structures could apply in the same situation, the more specific structure type takes precedence. Thus, a metric/SLO-specific structure overrides a metric-specific structure, which in turn overrides a global structure.

Here is a sample `tune` structure:

```
tune sales_app.resp_time {
   coll_argv = /opt/sales_app/bin/sales_monitor -v;
}
```

Defining a `tune` structure consists of the following tasks:

- Specifying a data collector (optional)
- Specifying the WLM interval (optional)
- Using absolute CPU units
- Distributing excess CPU to your workloads (optional)
- Temporarily removing groups with inactive SLOs (optional)
- Capturing your collectors' stderr (optional)
- Smoothing metric values (optional)
- Controlling averaging in usage controllers (optional)
- Trimming the statistics log file automatically (optional)
- Tuning a workload group's SLO convergence: `cntl_kp` (optional)
- Tuning a workload group's SLO convergence: `cntl_convergence_rate` (optional)
- Tuning the goal buffer (optional)
- Releasing CPUs properly (optional)

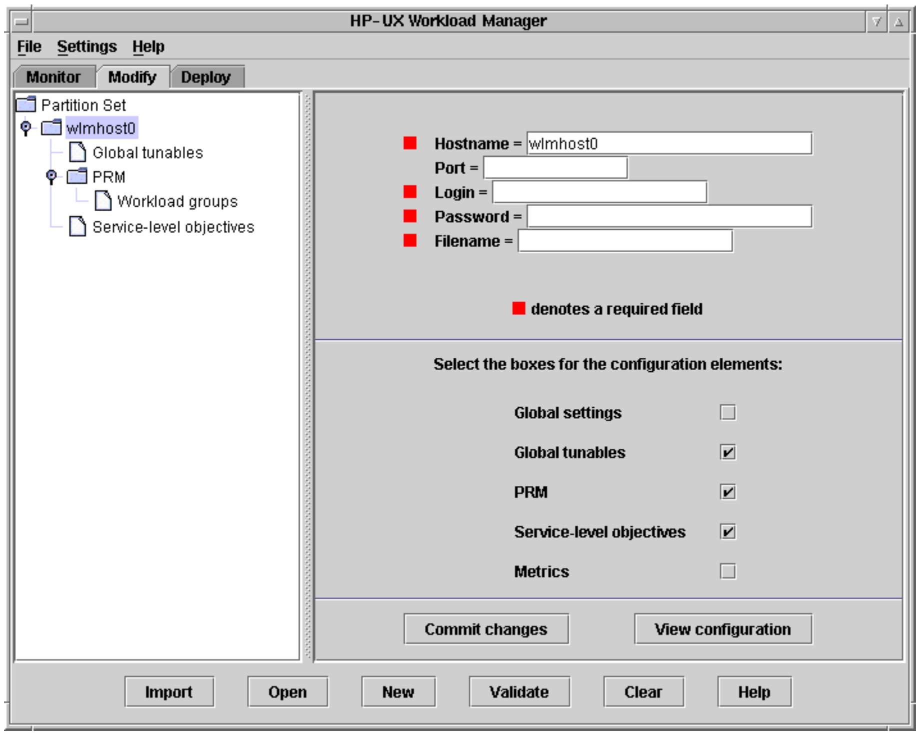
Tuning WLM using the WLM GUI

In addition to tuning with a text editor, you can use the WLM Configuration Wizard (/opt/wlm/bin/wlmcw). Alternatively, you can use the WLM GUI (/opt/wlm/bin/wlmgui).

The following steps show how to set the WLM interval and use absolute CPU units using the GUI. It is assumed you went through the procedure on "Defining PRM components using the WLM GUI" on page 141 to define a partition set. For more information on using the WLM GUI, see its online help.

Step 1. Select the name of the host, wlmhost0 in this case, in the left pane. The right pane allows you to set the type of configuration elements you would like to see in the new configuration.

Select the "Global tunables" box in the right pane. A "Global tunables" item then appears in the left pane.



🟥 Hostname = wlmhost0
Port =
🟥 Login =
🟥 Password =
🟥 Filename =

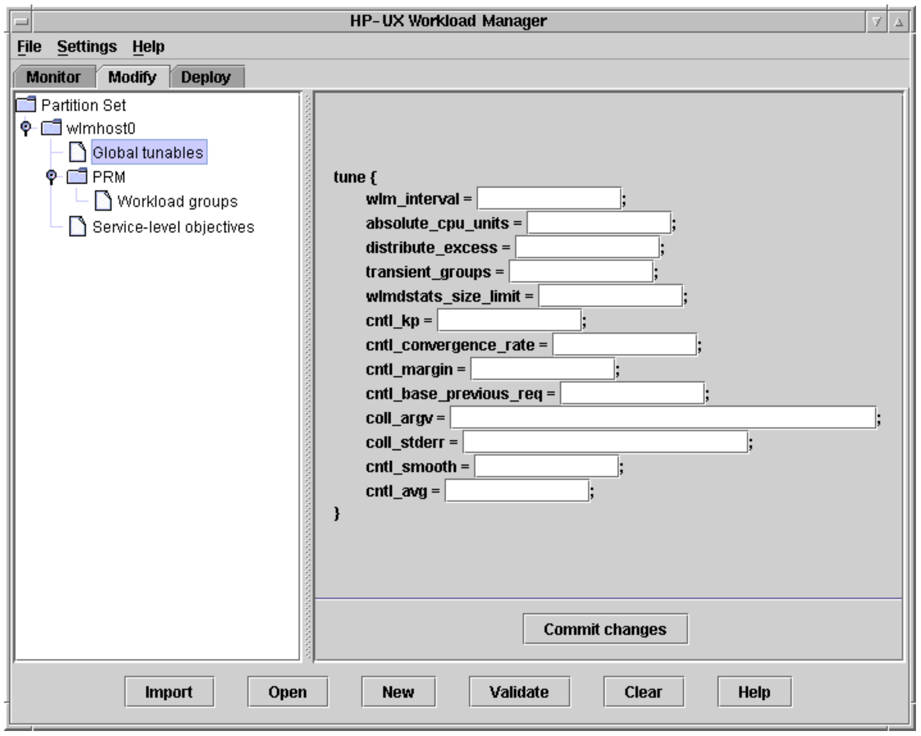
🟥 denotes a required field

Select the boxes for the configuration elements:

Global settings	☐
Global tunables	✓
PRM	✓
Service-level objectives	✓
Metrics	☐

Step 2. Select the “Global tunables” item in the left pane. The right pane changes, allowing you to set various tunables.

Set any tunables as desired.



```
tune {
    wlm_interval = ;
    absolute_cpu_units = ;
    distribute_excess = ;
    transient_groups = ;
    wlmdstats_size_limit = ;
    cntl_kp = ;
    cntl_convergence_rate = ;
    cntl_margin = ;
    cntl_base_previous_req = ;
    coll_argv = ;
    coll_stderr = ;
    cntl_smooth = ;
    cntl_avg = ;
}
```

Step 3. Select the **[Commit changes]** button to save the current configuration in memory. To save changes to disk, go to the Deploy tab. (The file is saved to the file specified in the filename field in the Modify tab when the system is selected in the left pane.)

The above steps highlight only one use of the WLM GUI. For additional information, see the GUI’s online help.

Specifying a data collector (optional)

Whenever you use a metric in your WLM configuration, you need to supply a value for that metric. The `coll_argv` statement launches a data collector to gather values for a metric. You can specify a data collector in a metric-specific `tune` structure to gather values for that one metric. If multiple metrics use the same command, such as `wlmrcvdc`, to collect values, you can specify the command in a global `tune` structure, instead of specifying it separately for each metric in its own metric-specific `tune` structure.

The `coll_argv` keyword is optional. However, if the SLO has a performance goal, you must specify a data collector. The `coll_argv` keyword is allowed in global and metric-specific `tune` structures.

NOTE Data collectors invoked by WLM run as root and can pose a security threat. Hewlett-Packard makes no claims of any kind with regard to the security of data collectors not provided by Hewlett-Packard. Furthermore, Hewlett-Packard shall not be liable for any security breaches resulting from the use of said data collectors.

Use the following syntax in a `tune` structure to specify a data collector and its command-line arguments:

```
coll_argv = data_collector_and_arguments;
```

where

`data_collector_and_arguments`

Is the full path to a data collector, plus any arguments. Separate arguments with white space. Use double quotes to form single arguments for an option and when using characters used in the syntax, such as semicolons, pound characters (#), and curly brackets.

This string cannot exceed 240 characters.

stdout and stderr for each data collector are sent to /dev/null; however, you can capture a collector's stderr output using the `coll_stderr` tunable, described in "Capturing your collectors' stderr (optional)" on page 209.

Here is an example `coll_argv` statement:

```
coll_argv = /opt/finance/bin/fin_mon -freq 3;
```

WLM provides a built-in data collector called `wlmrcvdc`. Specify this collector as follows:

```
coll_argv = wlmrcvdc [command];
```

where

command (optional)

Is a command provided for gathering data from GlancePlus, Oracle, other applications, or for checking on the status of a Serviceguard package, or a user-supplied data collector command.

For information on commands available with `wlmrcvdc`, see “`wlmrcvdc`” on page 393.

WLM simplifies the task of developing data collectors with its `wlmsend` and `wlmrcvdc` utilities. For information on these utilities and data collectors in general, see “Supplying data to WLM” on page 237.

Here is a partial WLM configuration showing the use of `wlmrcvdc` in a global `tune` structure.

```
# Global wlmrcvdc to collect any metric values sent in via wlmsend:
tune {
     coll_argv = wlmrcvdc;
}

# Collector to use instead of the global wlmrcvdc to get values
# for the metric order_transaction_time:
tune order_transaction_time {
     coll_argv = /home/orders/data_collector;
}

# Value of metric analysis_application_running given to WLM by wlmsend
# through global wlmrcvdc
slo analysis {
     ...
     condition = metric analysis_application_running;
     ...
}
```

```
# Value of metric job_count given to WLM by wlmsend through
# global wlmrcvdc
slo batch_processing {
     ...
     cpushares = 2 more per metric job_count;
     ...
}

# Uses metric value given by collector in metric-specific tune structure

slo order_processing {
     ...
     goal = metric order_transaction_time < 10.0;
     ...
}
```

If you are using WLM with Serviceguard, you might consider using `wlmrcvdc` in a global `tune` structure with `sg_pkg_active` to check the status of all the Serviceguard packages on the current system. For more information, see "Integrating with Serviceguard" on page 429 or the sg_pkg_active(1M) man page.

Specifying the WLM interval (optional)

Once per interval, WLM:

- Checks for new data from the data collectors
- Adjusts CPU allocations to better achieve SLOs, if necessary
- Adjusts memory allocations to compensate for any workload groups that became active or inactive in the last WLM interval

By specifying the interval, you can control how frequently these checks and adjustments occur.

The shorter the interval, the more often WLM checks for new performance data and alters the CPU allocation if workloads are not meeting their SLOs. If there is no new data, WLM does nothing.

With longer intervals, WLM is more likely to receive new performance data; however, a workload is also more likely to not achieve its SLO for longer periods.

The `wlm_interval` tunable is optional. It is allowed only in a global `tune` structure within the WLM configuration file.

NOTE The current WLM interval length is available to data collectors specified in `coll_argv` statements. The length can be retrieved through the `WLM_INTERVAL` environment variable.

Use the following syntax in a `tune` structure to specify the interval:

`wlm_interval =` *`number_of_seconds`*`;`

where

`number_of_seconds`

Specifies how often the WLM daemon checks for new data from the data collectors and adjusts CPU allocations. Valid values are from 1 to 86400 seconds (1 day).

The default is 60.

Typically, *`number_of_seconds`* should be 5 when your WLM configuration contains a usage goal.

When generating audit data (with the `wlmd -t` option), *`number_of_seconds`* should be no more than 60 to ensure the audit data sampling rate is sufficiently frequent to:

- Minimize the amount of audit data lost if the WLM daemon exits unexpectedly
- Minimize the effect of changes in the number of available CPUs (due to WLM's management of vPar, Instant Capacity, Temporary Instant Capacity, and Pay Per Use resources)

For example, the following global `tune` structure sets the interval to 15 seconds:

```
tune {
   wlm_interval = 15;
}
```

Using absolute CPU units

With the `absolute_cpu_units` tunable, you can control whether the CPU units used by the following keywords/tunables are absolute or relative:

- `mincpu`
- `maxcpu`
- `gmincpu`
- `gmaxcpu`
- `cpushares`
- `cntl_kp`
- `cntl_convergence_rate`

With absolute CPU units enabled (`absolute_cpu_units = 1`), you specify CPU units in terms of 100, where 100 represents an entire CPU. So, if you want a `mincpu` of half a CPU, you would specify:

```
mincpu = 50;
```

If you want a `maxcpu` of two CPUs, you would specify:

```
maxcpu = 200;
```

Absolute CPU units are used when the number of active CPUs changes due to WLM's management of Instant Capacity, Temporary Instant Capacity, Pay Per Use, or Virtual Partitions resources. Regardless of the number of active CPUs, the absolute CPU units represent the same amount of CPU.

With relative CPU units (`absolute_cpu_units` = 0, the default), the units you specify represent a percentage of the system's total CPU and are consequently relative to the number of active CPUs. For example, the following statement:

```
mincpu = 50;
```

is 50% of the system's CPU resources, which is 50% of one CPU on a system with only one active CPU, but is eight CPUs on a system with 16 active CPUs.

To use absolute CPU units, use the following statement in a global `tune` structure:

```
absolute_cpu_units = 1;
```

NOTE Several conditions involving `absolute_cpu_units` can cause your WLM configuration file to be invalid:

- Setting `absolute_cpu_units` equal to 0 and setting `primary_host`
- Setting `absolute_cpu_units` equal to 0 in a configuration that does not have a `prm` structure
- In a configuration with a `prm` structure:
  - — Setting `absolute_cpu_units` equal to 0 and specifying a PSET group
- In a configuration without a `prm` structure:
  - — Setting `absolute_cpu_units` equal to 0 and setting `hmincpu`
  - — Setting `absolute_cpu_units` equal to 0 and setting `hmaxcpu`
  - — Setting `absolute_cpu_units` equal to 0 and setting `hweight`

Distributing excess CPU to your workloads (optional)

By default, if any CPU remains after satisfying all SLOs, the remainder is given to the `OTHERS` group. To distribute this excess to the workload groups you have defined, set the `distribute_excess` tunable to 1 (true) in a global `tune` structure:

```
tune {
   distribute_excess = 1;
}
```

Only workload groups with active SLOs receive the excess CPU.

This distribution is based on balancing the weight-to-allocation ratios for the workload groups. These ratios are discussed in “Weighting a group so it gets more CPU (optional)” on page 162.

The distribution is subject to the group CPU maximum values specified by the `gmaxcpu` keyword. If the excess CPU is not fully distributed because of group CPU maximum values, the excess is given to the `OTHERS` group as usual—even if giving `OTHERS` this excess places it over its `gmaxcpu` value.

The default value for `distribute_excess` is 0, in which case the `OTHERS` group is given any excess CPU.

Temporarily removing groups with inactive SLOs (optional)

By default, all workload groups are always present—even those with no active SLOs. A group with no active SLOs still requires 1% of the total CPU (for FSS groups) or a whole CPU (for PSET-based groups), unless it has a `gmincpu` value requesting more. Then the workload group receives an allocation equal to that value. Similarly, if you are using WLM's memory management, the workload group with no active SLOs receives 1% of the memory, unless the workload group has a `gminmem` value requesting more.

If you would prefer these groups to go away temporarily (as long as they have no active SLOs) and consume no CPU or memory resources, set the `transient_groups` tunable to 1 in a global `tune` structure:

```
tune {
   transient_groups = 1;
}
```

With the `transient_groups` keyword set to 1: FSS groups with no active SLOs are deleted and therefore use no resources; the minimum CPU allocation for PSET groups becomes 0.

NOTE Using the `transient_groups` keyword does not throttle the resource usage of processes in transient groups that are inactive. In fact, those processes are moved—possibly to groups where they get more resources, as discussed in the "Placement of processes" sections below.

The `OTHERS` group is never removed, regardless of its number of active SLOs.

If a workload group has been temporarily removed, its `gmincpu` and `gminmem` values (if any) are ignored.

NOTE Do not create a `condition` statement that attempts to detect processes in a transient group using tools such as `glance` or `ps`. Whenever the group is deleted (FSS group) or assigned zero CPUs (PSET-based group), it is impossible for the system to place processes in the group. The condition will then never detect the processes it is looking for.

NOTE Setting `transient_groups` equal to 1 in a configuration that does not have a `prm` structure results in an invalid configuration.

Placement of processes for inactive FSS groups

With `transient_groups=1`, if an FSS workload group, say `mygrp`, has no active SLOs, but does have processes assigned to it by user records, application records, or compartment records, WLM moves its processes to a temporary group named `_IDLE_`. This group has only the minimum CPU and memory resources and can greatly restrict the progress of a process. When `mygrp` has active SLOs again, the processes placed in `mygrp` by records are moved back to `mygrp`.

If a process is not assigned to a group by a record, it is moved to `OTHERS` or `PRM_SYS` if its current group is removed. The process remains in `OTHERS` or `PRM_SYS` even when the group has active SLOs again. For this reason, be cautious when using only `prmrun` or `prmmove` to place a process in an FSS group that may be removed. Using user records, application records, or compartment records for processes that go into transient FSS groups ensures those processes return to the desired groups when the groups return.

Placement of processes for PSET-based groups

The placement of processes in a PSET-based group depends on whether the group has CPUs assigned to it. With `transient_groups=1`, the group can have 0 CPUs if you request it or if the group has no active SLOs.

Assume a PSET-based workload group, say `mygrp2`, has no CPUs. Any processes in the group when its last CPU is taken away, as well as processes you try to place in the group using `prmrun` or `prmmove` when the group has no CPUs, are placed in the `OTHERS` group.

NOTE To minimize the number of processes moved to `OTHERS`, assign at least one CPU to PSET-based groups that have running processes.

When `mygrp2` has CPUs again, the processes placed in `mygrp2` by application records, user records, or compartment records are moved back to `mygrp2`. Similarly, any processes you attempted to place in the group with `prmrun` or `prmmove` are moved to the group.

Capturing your collectors' stderr (optional)

Because they are started by a daemon process, data collectors do not have a stderr on which to communicate errors. The `coll_stderr` tunable allows you to capture these errors. Specify `coll_stderr` using the following syntax:

`coll_stderr =` *`file`*`;`

where

`coll_stderr`	Is an optional tunable with a default value of /dev/null. Specify this tunable in global or metric-specific `tune` structures.
`file`	Is either `syslog` (which corresponds to syslog on the system through the logger command, typically /var/adm/syslog/syslog.log) or the full path to a file.

For example, you can specify *`file`* in either of the following ways:

```
tune num_users {
   ...
   coll_stderr = syslog;
   ...
}

tune load_average {
   ...
   coll_stderr = /tmp/load_average.stderr;
   ...
}
```

NOTE There are no restrictions on the file specified. However, specifying system files or files created by applications on your system (such as WLM's /var/opt/wlm/msglog and /var/opt/wlm/wlmdstats) is not recommended as it can damage the files.

Smoothing metric values (optional)

Use the `cntl_smooth` keyword to get a running average of a metric from a data collector, smoothing out dips and spikes in the metric before it is used by WLM. Specify `cntl_smooth` using the following syntax:

`cntl_smooth = `*`smoothing_value`*`;`

where

`cntl_smooth`

Is an optional tunable that provides exponential smoothing. Specify this tunable in global or metric-specific `tune` structures.

`smoothing_value`

Is a floating-point value ranging from 0 to 0.999. The default value is 0, resulting in no smoothing. Values closer to 1 result in more smoothing.

NOTE Do not use `cntl_smooth` for metrics that are expected to equal zero in SLO `condition` expressions. For example, do not use smoothing globally if your configuration uses the `sg_pkg_active` collector to indicate a Serviceguard package is active (1) or inactive (0). You could use `cntl_smooth` in metric-specific `tune` structures in such a configuration, as long as those structures are not for Boolean metrics.

Smoothing occurs only in WLM intervals in which a new metric value is received.

The metric value reported in the log file /var/opt/wlm/wlmdstats is the smoothed value.

Controlling averaging in usage controllers (optional)

Use the `cntl_avg` keyword to control averaging in usage controllers. By default, averaging is enabled (`cntl_avg=1`). With averaging disabled (`cntl_avg=0`), only the most recent interval's usage data is used for a usage metric.

This tunable is ignored for metrics that are not CPU usage metrics.

Trimming the statistics log file automatically (optional)

You can automatically trim the statistics log file /var/opt/wlm/wlmdstats. This file is created when you use the `-l` option to `wlmd`. Enable automatic trimming of the file by using the `wlmdstats_size_limit` tunable. The syntax is:

`wlmdstats_size_limit = `*`number_of_megabytes`*`;`

where

`wlmdstats_size_limit`

Is an optional tunable. Specify this tunable in a global `tune` structure.

`number_of_megabytes`

Is an integer value from 0 to 2048. Once a file's size is greater than *`number_of_megabytes`*, the file is trimmed. The default value is 0, which disables automatic trimming. The wlmdstats file is trimmed by renaming it wlmdstats.old. Then a new wlmdstats file is started. The wlmdstats.old file includes a line at the end of the file indicating that the file has been automatically trimmed.

Tuning a workload group’s SLO convergence: `cntl_kp` (optional)

NOTE You can also tune convergence with the `cntl_convergence_rate` tunable discussed in the section “Tuning a workload group’s SLO convergence: `cntl_convergence_rate` (optional)” on page 216.

The `cntl_convergence_rate` tunable, which uses normalization to arrive at a new CPU allocation, is typically easier to use—requiring less analysis than `cntl_kp`.

WLM starts a performance controller for each goal-based SLO. It also starts a usage controller for each usage goal. Each controller calculates new CPU shares requests if the reported data indicates the workload group is overachieving or underachieving its goal. The controller then requests the new number of CPU shares.

NOTE Although convergence tuning is optional from a syntactic point of view, you should use `cntl_kp` or `cntl_convergence_rate` for each controller to fine-tune it to the characteristics of its workload.

The performance controllers receive two real-time inputs: The number of CPU shares the workload group had during the last WLM interval and the workload performance during that period. Usage controllers work on inputs of the workload group’s percent of CPU used and its CPU allocation for the last interval.

To determine the new CPU shares allocation, each controller effectively executes an algorithm that, when plugging in `cntl_kp` (as opposed to `cntl_convergence_rate`), is represented as follows:

```
New CPU allocation =
(Allocation last interval) + cntl_kp * P
```

where

`cntl_kp`	Is a tunable parameter for the controller to indicate a workload’s sensitivity to changes in CPU allocation. It is specified in the WLM configuration file.
`P`	Is the deviation from the goal.

WLM calculates `P` based on the goal type, as indicated below:

Performance goal with *`met`* < *`value`*:

`P = `*`met`*` - `*`value`*

where, as defined in the *`goal_expression`*, *`met`* is some metric for the workload and *`value`* is a goal value for the metric.

For example, if the goal (*`value`*) was that response time be less than 2 seconds, and the measured response time (*`met`*) was 3 seconds, the deviation from goal is calculated:

`P = 3 - 2 = 1`

Performance goal with *`met`* > *`value`*:

`P = `*`value`*` - `*`met`*

Usage goal:

`P = Actual utilization - Target utilization`

Target utilization is either *`low_util_bound`* or *`high_util_bound`*. If actual utilization is below *`low_util_bound`*, then

`P = Actual utilization - `*`low_util_bound`*

If actual utilization is above *`high_util_bound`*,

`P = Actual utilization - `*`high_util_bound`*

For example, if the goal is to keep CPU utilization between 50% and 75%, with the utilization at 85%, the deviation from the goal is:

`P = 85 - 75 = 10`

Use the optional `cntl_kp` tunable to tune convergence. It can be used in global, metric-specific, and metric/SLO-specific `tune` structures. Its syntax is:

```
cntl_kp = proportional_term;
```

where

`proportional_term`

Is a floating-point value between 0 and 1,000,000 (inclusive) used on the proportional term in the controller's expression. The default value is 1.

Change this value according to how sensitive the SLO's associated workload is to changes in CPU allocation and based on the magnitude of the goal being measured.

Recall the formula for determining new allocation requests:

```
New CPU allocation =
(Allocation last interval) + cntl_kp * P
```

If `P` is typically large, say 10 or 100 or higher, use a smaller `cntl_kp` to scale down the `cntl_kp * P` result so it does not overcorrect. Similarly, for very small values of `P` (less than 1), use `cntl_kp` to scale up the `cntl_kp * P` result.

If WLM changes allocations too rapidly, resulting in instability, decrease *`proportional_term`*. If WLM changes allocations too slowly, increase *`proportional_term`*.

With absolute CPU units enabled (`absolute_cpu_units = 1`), your *`proportional_term`* maintains more predictable behavior from your workload regardless of the number of available CPUs. Without absolute CPU units, the effect of a given *`proportional_term`* depends on the number of available CPUs, which could produce undesirable effects.

NOTE

A *proportional_term* equal to 0, when applied to the formula for determining new allocations

```
New CPU allocation =
(Allocation last interval) + cntl_kp * P
```

results in the formula

```
New CPU ent. = (Allocation last interval)
```

which leads to no change in an SLO's allocation request.

Also, using large values for *proportional_term* can produce unstable behavior, causing service levels and allocations to oscillate drastically.

In the following example, `cntl_kp` is set to 2 (twice the default) so that the SLO converges twice as fast on its goal:

```
tune sales_app.resp_time {
   coll_argv = /opt/sales_app/monitor;
   cntl_kp = 2.0;
}
```

For more information on how to tune, see the white paper "Tuning HP-UX Workload Manager" at /opt/wlm/share/doc/howto/tuning.html.

Tuning a workload group’s SLO convergence: `cntl_convergence_rate` (optional)

NOTE You can also tune convergence with the `cntl_kp` tunable discussed in the section “Tuning a workload group’s SLO convergence: `cntl_kp` (optional)” on page 212. (If `cntl_convergence_rate` is not zero, it is used instead of `cntl_kp`.)

Using the `cntl_convergence_rate` tunable changes the allocation algorithm to take the goal value into consideration, effectively normalizing the result by the goal’s magnitude. Given the normalization that `cntl_convergence_rate` provides, as well as the greater analysis required to fine-tune the `cntl_kp` tunable, you may want to use `cntl_convergence_rate` first. In many cases, setting `cntl_convergence_rate` = 1.0 will provide the desired convergence and meet your needs.

For each performance and usage goal, WLM must determine the appropriate number of CPU shares needed to achieve that goal. To do this, WLM creates a controller for each such goal.

NOTE Although convergence tuning is optional from a syntactic point of view, you should use `cntl_convergence_rate` or `cntl_kp` for each controller to fine-tune it to the characteristics of its workload.

To determine the new CPU shares allocation, each controller effectively executes an algorithm that, when plugging in `cntl_convergence_rate` (as opposed to `cntl_kp`), is represented as follows:

```
New CPU allocation = (Allocation last interval) +
                    (cntl_convergence_rate / 0.10) * (P / goal)
```

where

`cntl_convergence_rate`

Is a tunable parameter for the controller that indicates the number of CPU shares by which to adjust a workload group's allocation when it differs from its goal. It is specified in the WLM configuration file.

`P`

Is the deviation from the goal and is calculated as shown in the previous section, "Tuning a workload group's SLO convergence: `cntl_kp` (optional)".

There are two special cases to consider in the formula above:

- For `goal` = 0, "`(P / goal)`" becomes "`P`"
- For usage goals, `goal` becomes the average of the *`low_util_bound`* and *`high_util_bound`* values: "(low_util_bound + high_util_bound) / 2"

Use the optional `cntl_convergence_rate` tunable to tune convergence. It can be used in global, metric-specific, and metric/SLO-specific `tune` structures. Its value is inherited from a higher level `tune` structure if you do not specify it at a given level.

Its syntax is:

`cntl_convergence_rate =` *`number_of_shares`*`;`

where

`number_of_shares`

Is a floating-point value that represents the number of shares (either absolute or relative) by which a workload group's CPU allocation is adjusted when its service level differs from its goal by 10%. When the difference is less than 10%, the adjustment is proportionally smaller than *`number_of_shares`*.

Similarly, when the difference is greater than 10%, the adjustment is proportionally larger than *`number_of_shares`*.

The larger *`number_of_shares`* is, the larger WLM's adjustments to the workload group's CPU allocation are. This generally produces faster convergence on the SLO goal.

If WLM changes allocations too rapidly, resulting in instability, decrease *`number_of_shares`*. If WLM changes allocations too slowly, increase *`number_of_shares`*.

With absolute CPU units enabled (`absolute_cpu_units = 1`), your *`number_of_shares`* maintains more predictable behavior from your workload regardless of the number of available CPUs. Without absolute CPU units, the effect of a given *`number_of_shares`* depends on the number of available CPUs, which could produce undesirable effects.

Tuning the goal buffer (optional)

WLM attempts to meet a goal that is slightly different from the goal you specify in your configuration. By working toward this goal, WLM has a buffer to work within—and small fluctuations in the service level do not result in SLO violations.

You can specify the percentage size of this buffer with the `cntl_margin` tunable. This tunable is optional and can be used in global, metric-specific, and metric/SLO-specific `tune` structures.

The `cntl_margin` tunable syntax is:

```
cntl_margin = margin_value;
```

where

`margin_value`

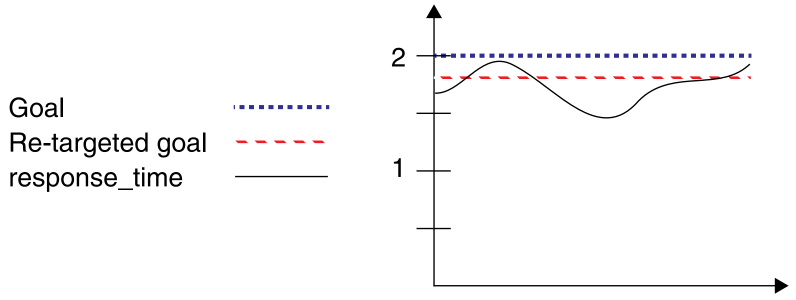
Is a floating-point value between 0 and 1 (inclusive) indicating a percentage distance away from the SLO's goal. The default value is 0.1, or 10%.

WLM uses this value to re-target the goal it is trying to converge on. For example, with a `margin_value` of 0.1 and a goal of *X*, the re-targeted goal is *X* - (0.1**X*). The controller adjusts the goal up or down (based on whether the goal is > or <) by this percentage to create a buffer zone. This adjustment prevents small fluctuations in the service level from causing SLO violations.

Consider Figure 6-6. Assume the goal in the configuration file is response_time < 2 seconds, and the default `cntl_margin` value of 0.1 is in effect. WLM internally re-targets the goal to 1.8 seconds (0.1*2 = 0.2; 2 - 0.2 = 1.8). As shown below, the response_time values are close to 1.8, with some values below and some above the re-targeted goal. Without `cntl_margin`, these values would converge from below and above on the goal in the configuration file. Each value above the goal would then cause an SLO violation. With the re-targeted goal, these fluctuations do not cause SLO violations.

NOTE The "`cntl_margin = margin_value`" tunable applies only to performance goals. Do not use it with usage goals.

Figure 6-6 Effect of the `cntl_margin` tunable parameter



For more information on how to use the `cntl_margin` tunable, see the white paper “Tuning HP-UX Workload Manager” at /opt/wlm/share/doc/howto/tuning.html.

Releasing CPUs properly (optional)

By default, the `cntl_base_previous_req` tunable is set to 1, which can be beneficial when you are using WLM’s global arbiter (`wlmpard`) to manage virtual partitions or nPartitions or when your WLM configuration has at least one PSET- based workload group with an SLO.

WLM creates a controller for each `slo` structure that has a `goal` statement. Controllers make requests to the WLM arbiter for shares allocations. Setting `cntl_base_previous_req=0` can result in some controllers requesting CPU more aggressively; however, the distribution of CPU resources may become unfair due to artificially high resource requests.

Use this optional tunable in global, metric-specific, and metric/SLO-specific `tune` structures

Example configuration

The following is an example configuration consisting of two workload groups and four SLOs. A discussion of the example is presented after the example.

```
# Define the workload groups
prm {
   groups = finance:2, sales:3;
   apps = finance: /opt/fin_app/count_money, sales: /opt/sales/do_ebiz;
}

# This is an SLO for the finance group.
slo finance_query {
   pri = 1;
   mincpu = 20;
   maxcpu = 50;
   entity = PRM group finance;
   goal = metric fin_app.query.resp_time < 2.0;
   condition = Mon - Fri;
}

# This is a stretch goal for the finance query group. If all other CPU
# requests of higher priority SLOs have been met, apply more CPU to
# group finance, so its application runs faster.
slo finance_query_stretch {
   pri = 5;
   mincpu = 20;
   maxcpu = 80;
   entity = PRM group finance;
   goal = metric fin_app.query.resp_time < 1.0;
   condition = Mon - Fri;
}

# On weekends, we do not expect any query transactions, but just in
# case, we will specify a nominal CPU allocation for this application
# for off-hours.
slo finance_query_weekend {
   pri = 1;
   cpushares = 5 total;
   entity = PRM group finance;
   condition = Sat - Sun;
}
```

```
# This is an SLO for the Sales group.
slo sales_query {
   pri = 1;
   mincpu = 40;
   maxcpu = 80;
   entity = PRM group sales;
   goal = metric sales_app.resp_time < 10.0;
}

tune {
   wlm_interval = 30;
}

tune fin_app.query.resp_time {
   coll_argv = /opt/fin_app/finance_collector -a 123 -v;
   cntl_kp = 0.8;
}

tune sales_app.resp_time {
   coll_argv = /opt/sales_app/monitor -threshold 2 -freq 30 ;
   cntl_kp = 2.0;
}
```

This configuration file specifies four SLOs. The `finance_query` SLO is active Monday through Friday. Its goal is that the metric fin_app.query.resp_time, which is provided by the executable /opt/fin_app/finance_collector, must always be less than 2.0. This goal is priority 1, the highest priority. The controller for this goal determines the CPU entitlement (allocation) needed to achieve the goal, but is bounded by the minimum and maximum CPU requests, which are 20% and 50%.

Additionally, a stretch goal is defined for this workload. The stretch goal is the SLO named `finance_query_stretch`, which is also only applicable Monday through Friday. The stretch goal is of lower priority (`pri = 5`). The stretch goal is that the metric fin_app.query.resp_time be less than 1.0. The controller for the stretch goal is bounded by the same minimum CPU allocation, but has a higher maximum.

The SLO named `finance_query_weekend` applies to this workload group as well. This SLO does not specify a performance goal, but requests a CPU allocation of 5% for weekends.

The `sales_query` SLO is active at all times. Its goal is that the metric sales_app.resp_time be less than 10.0. This metric is provided by the data collector /opt/sales_app/monitor. This SLO specifies that the controller for this SLO never requests a CPU allocation of less than 40%, nor more than 80%. This goal is priority 1.

The global tunable `wlm_interval` is set to 30, thus the WLM daemon checks for new performance data every 30 seconds.

The `finance_query` and `finance_query_stretch` SLOs both use the metric fin_app.query.resp_time. The `tune` structure for that metric specifies the settings of the tunables used by those SLOs. The `argv` for the process used to collect the metric fin_app.qurey.resp_time is given by:

```
coll_arg = /opt/fin_app/finance_collector -a 123 -v;
```

For the controllers that use this metric, the value of the proportional constant is set to 1.0. All other tunables are set according to the default values listed in the master tunables file.

The `sales_query` SLO uses the metric sales_app.resp_time. The `tune` structure for this metric specifies that when the daemon starts the data collector to collect this metric, it uses the `coll_argv` string `/opt/sales_app/monitor -threshold 2 -freq 30`. Also, this `tune` structure specifies the proportional constant used by the controllers for SLOs that use this metric. The value of the proportional constant (`cntl_kp`) for those controllers is 2.0. All other tunables are set according to the default values in the master tunables file.

For more examples files, see “Example configuration files” on page 305.

Trying a configuration without affecting the system

WLM provides a passive mode that allows you to see how WLM will approximately respond to a given configuration—without putting WLM in charge of your system's resources. Using this mode, you can analyze your configuration's behavior—with minimal effect on the system. Besides being useful in understanding and experimenting with WLM, passive mode can be helpful in capacity-planning activities.

A sampling of possible uses for passive mode are described below. (For information on features mentioned below, see their descriptions earlier in this chapter.) These uses help you determine:

- How does a `condition` statement work?

  Activate your configuration in passive mode then start the `wlminfo` utility. Use `wlmsend` to update the metric that is used in the `condition` statement. Alternatively, wait for the condition to change based on the date and time. Monitor the behavior of the SLO in question in the `wlminfo` output. Is it on or off?

NOTE Always wait at least 60 seconds (the default WLM interval) for WLM's changes to resource allocations to appear in the `wlminfo` output. (Alternatively, you can adjust the interval using the `wlm_interval` tunable in your WLM configuration file.)

- How does a `cpushares` statement work?

  Activate your configuration in passive mode then start the `wlminfo` utility. Use `wlmsend` to manipulate the metric used in the `cpushares` statement. What is the resulting allocation shown in the `wlminfo` output?

- How do goals work? Is my goal set up correctly?

  Activate your configuration and monitor the WLM behavior in the `wlminfo` output. What is the range of values for a given metric? Does WLM have the goal set to the level expected? Is WLM adjusting the workload group's CPU allocation?

- How might a particular `cntl_convergence_rate` value or the values of other tunables affect allocation changes?

  Create several configurations, each with a different value for the tunable in question. Activate one of the configurations and monitor the WLM behavior in the `wlminfo` output. Observe how WLM behaves differently under each of the configurations.

- How does a usage goal work?

  In passive mode, a usage goal's behavior might not match what would be seen in regular mode, but what is its basic behavior if the application load for a particular workload group is increased?

  Activate your configuration and monitor the `wlminfo` output to see how WLM adjusts the workload group's CPU allocation in response to the group's usage.

- Is my global configuration file set up as I wanted? If I used global arbitration on my production system, what might happen to the CPU layouts?

NOTE You can run `wlmpard` in passive mode with each partition's `wlmd` daemon running in regular mode. Thus, you can run `wlmpard` experiments on a production system without consequence.

In addition, passive mode allows you to validate workload group, application, and user configuration. For example, with passive mode, you can determine:

- Is a user's default workload group set up as I expected?
- Can a user access a particular workload group?
- When an application is run, which workload group does it run in?
- Can I run an application in a particular workload group?
- Are the alternate names for an application set up correctly?

Furthermore, using metrics collected with `glance_prm`, passive mode can be useful for capacity planning and trend analysis. For more information, see glance_prm(1M).

Passive mode versus actual WLM management

This section covers the following topics:

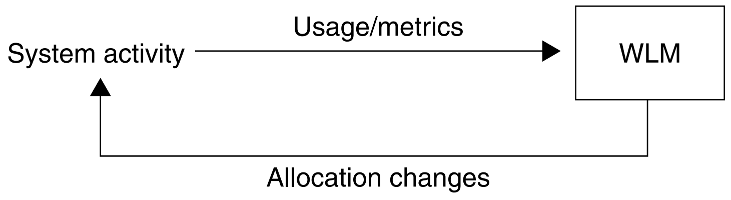
- The WLM feedback loop
- Effect of `mincpu` and `maxcpu` values
- Using `wlminfo` in passive mode
- The effect of passive mode on usage goals and metric goals

The WLM feedback loop

WLM's operations are based on a feedback loop: System activity typically affects WLM's arbitration of service-level objectives. This arbitration results in changes to CPU allocations for the workload groups, which can in turn affect system activity—completing the feedback loop.

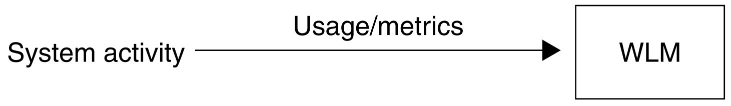
The diagram below shows WLM's normal operation, including the feedback loop.

Figure 6-7 **WLM's normal operation**



In passive mode, however, the feedback loop is broken, as shown below.

Figure 6-8 **WLM's passive operation**



Thus, in passive mode, WLM takes in data on the workloads. It even forms a CPU request for each workload based on the data received. However, it does not change the CPU allocations for the workloads on the system.

Effect of `mincpu` and `maxcpu` values

In passive mode, WLM does use the values of the following keywords to form shares requests:

- `mincpu`/`maxcpu`
- `gmincpu`/`gmaxcpu`
- `hmincpu`/`hmaxcpu`

However, because WLM does not adjust allocations in passive mode, it may appear that these values are not used.

Using `wlminfo` in passive mode

Use the `wlminfo` utility to monitor WLM in passive mode. Its output reflects WLM behavior and operation. It shows how much CPU WLM is requesting for a workload—given the workload's current performance. However, because WLM does not actually adjust CPU allocations in passive mode, WLM does not affect the workload's performance—as reported in usage values and metric values. Once you activate WLM in normal mode, it adjusts allocations and affects these values.

NOTE For the purposes of passive mode, WLM creates a PRM configuration with each of your workload groups allocated one CPU share, and the rest going to the reserved group `PRM_SYS`. (If your configuration has PSET-based workload groups, the PSETs are created but with 0 CPUs.) In this configuration, CPU capping is not enforced—unlike in normal WLM operation. Furthermore, this configuration will be the only one used for the duration of the passive mode. WLM does not create new PRM configurations to change resource allocations. Consequently, you should not rely on `prmlist` or `prmmonitor` to observe changes when using passive mode. These utilities will display the configuration WLM used to create the passive mode. However, you can use `prmmonitor` to gather CPU usage data.

The effect of passive mode on usage goals and metric goals

As noted above, in passive mode, WLM's feedback loop is not in place. The lack of a feedback loop is most dramatic with usage goals. With usage goals, WLM changes a workload group's CPU allocation so that the group's actual CPU usage is a certain percentage of the allocation. In passive mode, WLM does not actually change CPU allocations. Thus, an SLO with a usage goal might be failing; however, that same SLO might easily be met if the feedback loop were in place. Similarly, an SLO that is passing might fail if the feedback loop were in place. However, if you can suppress all the applications on the system except for the one with a usage goal, `wlminfo` should give you a good idea of how the usage goal would work under normal WLM operation.

Passive mode can have an effect on SLOs with metric goals as well. Because an application is not constrained by WLM in passive mode, the application might produce metric values that are not typical for a normal WLM session. For example, a database application might be using most of a system. As a result, it would complete a high number of transactions per second. The database performance could be at the expense of other applications on the system. However, your WLM configuration might scale back the database's access to resources to allow the other applications more resources. Thus, the `wlminfo` output would show WLM's efforts to reduce the database's CPU allocation. Because passive mode prevents a reduction in the allocation, the database's number of transactions per seconds (and system use) remains high. WLM, believing the previous allocation reduction did not produce the desired result, again lowers the database's allocation. Thus, with the removal of the feedback loop, WLM's actions in passive mode do not always indicate what it would do normally.

Because of these discrepancies, always be careful when using passive mode as an indicator of normal WLM operation. Use passive mode to see trends in WLM behavior—with the knowledge that the trends may be exaggerated because the feedback loop is not present.

Activating the configuration file

When activating a WLM configuration file, you can run WLM in passive mode—allowing you to verify a configuration before using it to control the system's resources. To use passive mode, specify `-p` with the `wlmd` command:

```
# wlmd -p -a configfile
```

Use the `wlminfo` utility to then monitor how WLM would have allocated CPU to your workload groups in *`configfile`*.

For information on passive mode, including its limitations, see "Passive mode versus actual WLM management" on page 226.

After fine-tuning your WLM configuration, activate it—letting WLM take control of your system's resources—as follows:

```
# wlmd -a configfile
```

Both command lines above check that *`configfile`* is valid. If the file is valid with respect to the WLM syntax, it is passed to PRM for validation. So, after passing WLM validation, you may encounter PRM validation errors. Once your file passes both validations, the WLM daemon is started. The daemon then starts all the data collectors specified in the configuration file. If a previous instance of `wlmd` is running, it is replaced by this new instance.

After starting the data collectors, every `wlm_interval` seconds, the WLM daemon checks for performance data to act on. The default interval value is 60 seconds. You can alter this value as explained in the section "Specifying the WLM interval (optional)" on page 203.

For information on activating the WLM global arbiter configuration, see "Managing SLOs across partitions" on page 283.

Managing duration

WLM offers duration management through the Duration Management Toolkit (DMTK), which is part of the WLM Toolkits (WLMTK) product.

WLM and DMTK help control CPU resources by granting a critical job only the amount of CPU needed to complete within a certain timeframe. Because the business-critical job is being duration managed, the extra compute resources on the server can be made available to other users or jobs without affecting the managed business-critical job. This translates to more efficient use of existing compute resources.

For more information, see the wlmdurdc(1M) man page.

Setting WLM to start automatically at reboot

NOTE You must either activate a valid WLM configuration or specify one with the variable `WLM_STARTUP_SLOFILE` in the file /etc/rc.config.d/wlm before you set WLM to start automatically at reboot. For information on activating a configuration, see “Activating the configuration file” on page 229.

You can set WLM to start automatically at reboot by setting the `WLM_ENABLE` variable in the file /etc/rc.config.d/wlm to 1:

```
WLM_ENABLE=1
```

When started at reboot, WLM automatically uses the most recent configuration file that was activated.

If you want WLM to activate a specific configuration when it automatically starts at reboot, edit the following line in the /etc/rc.config.d/wlm file:

```
WLM_STARTUP_SLOFILE="configfile"
```

where *configfile* is the configuration file to use at reboot.

For information on starting WLM's global arbiter automatically, see the section "Setting WLM global arbitration to start automatically at reboot" on page 231.

Setting WLM global arbitration to start automatically at reboot

NOTE You must either activate a valid WLM global arbiter configuration or specify one with the variable `WLMPARD_STARTUP_SLOFILE` in the file /etc/rc.config.d/wlm before you set WLM's global arbiter to start automatically at reboot. For information on activating a global arbiter configuration, see "Setting up cross-partition management" on page 285.

The WLM global arbiter is used to manage WLM SLOs across virtual partitions.

You can set the global arbiter to start automatically at reboot by setting the `WLMPARD_ENABLE` variable in the file /etc/rc.config.d/wlm to 1:

```
WLMPARD_ENABLE=1
```

When started at reboot, the WLM global arbiter automatically uses the most recent configuration file that was activated.

If you want the global arbiter to activate a specific configuration when it automatically starts at reboot, edit the following line in the /etc/rc.config.d/wlm file:

```
WLMPARD_STARTUP_SLOFILE="configfile"
```

where *configfile* is the configuration file to use at reboot.

For information on starting WLM automatically, see the section "Setting WLM to start automatically at reboot" on page 230.

Setting the WLM communications daemon to start automatically at reboot

The WLM communications daemon (`wlmcomd`) services requests from the WLM graphical user interface, `wlmgui`. The daemon must be running on a system for you to control that system's WLM instance or WLM global arbiter using `wlmgui`.

You can set `wlmcomd` to start automatically at reboot by setting the `WLMCOMD_ENABLE` variable in the file /etc/rc.config.d/wlm to 1:

```
WLMCOMD_ENABLE=1
```

With `wlmcomd` running on a system, WLM itself does not need to be running on that system because you can deploy WLM there with `wlmgui`. However, you can start WLM automatically if desired, as described in the section "Setting WLM to start automatically at reboot" on page 230.

Securing WLM communications automatically at reboot

The communications between WLM daemons are not secure by default. You can enable secure communications using the `-s` option with any of the daemons. You can start the daemons in secure mode at boot time with the following variables in the file /etc/rc.config.d/wlm:

`WLMD_SECURE_ENABLE`

`WLMPARD_SECURE_ENABLE`

`WLMCOMD_SECURE_ENABLE`

Enabling statistics logging at reboot

The following variables in /etc/rc.config.d/wlm allow you to log statistics starting at reboot:

- `WLM_STATS_LOGGING`

  Set WLM to start logging statistics at reboot by setting the `WLM_STATS_LOGGING` variable in the file /etc/rc.config.d/wlm. Valid values are:

  — `all`

  — `group`

  — `host`

  — `metric`

  — `slo`

  Values can be combined, separated by commas, with no white space:

  ```
  WLM_STATS_LOGGING=slo,metric
  ```

  Be sure to include double quotes when you request logging less often than once per WLM interval:

  ```
  WLM_STATS_LOGGING="slo=10,metric"
  ```

  Setting `WLM_STATS_LOGGING` has the same effect as using the `wlmd -l` option. For a description of the `-l` arguments, see the reference section “`wlmd`” on page 381.

  You can limit the size of /var/opt/wlm/wlmdstats as explained in the section “Trimming the statistics log file automatically (optional)” on page 211.

- `WLMPARD_STATS_LOGGING`

  Set the WLM global arbiter to start logging statistics at reboot by setting the `WLMPARD_STATS_LOGGING` variable in /etc/rc.config.d/wlm as follows:

  ```
  WLMPARD_STATS_LOGGING="vpar"
  ```

  You can limit the size of /var/opt/wlm/wlmpardstats as explained in "Trimming the global arbiter statistics log automatically (optional)" on page 295.

Disabling statistics logging

To disable statistics logging by WLM, restart WLM without the `-l`. For example:

- Restart WLM with the last valid configuration:

  ```
  # wlmd -A
  ```

- Restart WLM with a new configuration:

  ```
  # wlmd -a configfile
  ```

To disable statistics logging by the WLM global arbiter, restart `wlmpard` without the `-l`. For example:

- Restart the WLM global arbiter with the last valid configuration:

  ```
  # wlmpard -A
  ```

- Restart the WLM global arbiter with a new configuration:

  ```
  # wlmpard -a configfile
  ```

NOTE If logging is set in the /etc/rc.config.d/wlm file, logging will resume at the next reboot.

WLM and kernel parameters

WLM assumes your HP-UX kernel parameters are already set appropriately for the workloads you are running. Consequently, WLM does not change kernel parameters in order to meet the specified SLOs.

7 Supplying data to WLM

You supply data to WLM so it can manage SLOs that have performance goals, shares-per-metric allocations, or `condition`/`exception` statements with metric expressions. You gather this data with collectors provided by WLM or ones you develop yourself. Data collectors are specified in the WLM configuration file. WLM spawns the collectors when the configuration is activated.

Sources for data include:

- ARM (Application Response Measurement) metrics
- GlancePlus metrics
- Oracle database metrics
- Various third-party applications

  Metrics can be easily obtained for a number of third-party applications through the WLM Toolkits (WLMTK). In addition to the toolkit for Oracle databases, WLMTK provides toolkits for Apache, BEA WebLogic Server, and SAS.
- Existing application and system metrics you already track

Once this data is collected, it can be sent to WLM in one of two ways:

- WLM's `wlmrcvdc` and `wlmsend` utilities (which provide a command-line, shell script, and perl program interface)
- WLM's C language API

This chapter covers these topics from two views:

- Data-centric:

  I have this type of data. How do I get it to WLM?
- Transport-centric:

  I want to get data to WLM this way. What data is available for that transport method?

NOTE A white paper on data collectors (”Writing a Better WLM Data Collector”) is available at /opt/wlm/share/doc/howto/perfmon.html. This paper provides additional information on implementing your data collectors.

How applications can make metrics available to WLM

Consider the following overview of how to get metrics to WLM.

Time metrics from instrumentable applications

If the desired metric can be measured in units of time, and the application can be modified, we recommend using the ARM API provided by GlancePlus. WLM will then collect the ARM data from GlancePlus, synchronizing the values used by WLM with those used in VantagePoint Performance tools.

Adding ARM routines to an application is as simple as specifying your application with an `arm_init` call, marking the start of the time period to be measured with an `arm_start` call, and marking the end of the time period with an `arm_end` call.

For more ARM information, see “Sending ARM transaction data from a modified C program” on page 263.

Other data collection techniques

If your application cannot be modified to insert ARM calls, or if your metric does not have time units, then you should implement an external data collector. There are three types of external data collectors to consider:

- Independent collectors
- Stream collectors
- Native collectors

These collector types are explained below.

Independent collectors

Independent collectors use the `wlmsend` command to communicate a metric value to WLM. For more information on this command, see “What methods exist for sending data to WLM?” on page 250.

They are called “independent” because they are not started by the WLM daemon `wlmd`, and they are not required to run continuously.

This type of collector is ideal if you want to convey event information to WLM, such as application startup or shutdown.

One caveat of using this type of collector is that on start-up, WLM has no value for the metric until the collector provides one. For this reason, the collector should be structured to report a value periodically, even if it has not changed.

If your collector runs continuously, be careful if using pipes. The pipes may have internal buffering that must either be defeated or flushed to ensure the data is communicated in a timely manner.

To configure an independent collector for a metric called metricIC, place the following `tune` structure in your configuration file:

```
tune metricIC {
   coll_argv = wlmrcvdc;
}
```

Stream collectors

Stream collectors convey their metric values to WLM by writing them to stdout. WLM starts these data collectors when activating a configuration, and expects them to continue to run and provide metrics until notified of a WLM shutdown or restart.

Use this type of collector if the metric is available in a file or through a command-line interface. In this case, the collector can simply be a script containing a loop that reads the file or executes the command, extracts the metric value, writes it on stdout, and sleeps for one WLM interval. (The current WLM interval length is available through the `WLM_INTERVAL` environment variable to data collectors started by WLM through a `coll_argv` statement in the WLM configuration.)

Again, as with independent collectors, be careful if using pipes in the data collector. These pipes may have internal buffering that must either be defeated or flushed to ensure the data is communicated in a timely manner.

Because they are started by a daemon process (`wlmd`), stream collectors do not have a stderr on which to communicate errors. However, WLM provides the `coll_stderr` tunable that allows you to log each collector's stderr. In addition, a stream data collector can communicate using either syslog(3C) or logger(1) with the daemon facility.

To configure a stream collector for a metric called metricSC, place the following `tune` structure in your configuration file:

```
tune metricSC {
   coll_argv = wlmrcvdc collector_path collector_args ;
}
```

The `sg_pkg_active` data collector is an example of a stream collector, as are several of the Toolkit data collectors, including `time_url_fetch` (ApacheTK) and `wlmwlsdc` (WebLogicTK).

Native collectors

Native collectors use the WLM API to communicate directly with the WLM daemon. For information on WLM's API, see "Sending data from a collector written in C" on page 257.

Like stream collectors, these collectors are started by WLM when activating a configuration. WLM expects them to continue to run and provide metrics until notified of a WLM shutdown or restart. For tips on writing your own data collectors, see the white paper at /opt/wlm/share/doc/howto/perfmon.html.

This type of collector is appropriate if the desired metric values are obtained through calls to a C or C++ language API that is provided by the source of the metric. One example of such an API is the pstat(2) family of system calls used to obtain process statistics.

This type of collector establishes a direct connection with WLM using the WLM API function `wlm_mon_attach()`. Then, executed in a loop, the collector calls the API functions necessary to obtain the metric value, followed by a call to the WLM API function `wlm_mon_write()` to pass the value on.

Because they are started by a daemon process (`wlmd`), native collectors' output to stdout and stderr is discarded. However, WLM provides the `coll_stderr` tunable that allows you to log each collector's stderr. In addition, a native data collector can communicate using either syslog(3C) or logger(1) with the daemon facility.

To configure a native collector for a metric called metricNC, place the following `tune` structure in your configuration file:

```
tune metricNC {
   coll_argv = collector_path collector_args ;
}
```

`wlmrcvdc` is an example of a native collector.

What happens when there is no new data?

If an entire WLM interval passes and WLM does not receive any new data for a metric, all controllers using that metric simply request the same allocations for their associated workload groups as they did the previous interval. As always, the requests may not be met due to the CPU requests of other controllers.

NOTE The current WLM interval length is available through the `WLM_INTERVAL` environment variable to data collectors started by WLM (by specifying the data collector in a `coll_argv` statement in a WLM configuration file).

I have this type of data—how do I send it to WLM?

This section focuses on the type of data you have collected or plan to collect. Knowing the type of data, you can then determine the best way to get that data to WLM.

If you have any of the following types of data, send the data to WLM as explained in this section:

- ARM transaction data
- GlancePlus application data
- GlancePlus global data
- GlancePlus PRM-specific application data
- GlancePlus PRM-controlled volume group data
- Oracle database data
- Existing metrics

GlancePlus is an optional HP product. You must install it if you wish to use HP's ARM implementation or to collect any of the GlancePlus data listed above. For purchase information, contact your HP sales representative.

Table 7-1 gives an overview of the types of data you can collect and the methods for transporting that data to WLM.

Table 7-1 Type of data and its transport method

What type of data do you have?	Transport method
• The application is already instrumented with ARM API calls • You have the application source code and are going to instrument it with ARM API calls (TT_* metrics)	`glance_tt`
You already track GlancePlus "applications" you defined in your /var/opt/perf/parm file (APP_* metrics)	`glance_app`
• GlancePlus Global metrics (GBL_* metrics) • Table metrics (TBL_* metrics)	`glance_gbl`
GlancePlus PRM-specific application metrics (APP_PRM_* and APP_* metrics)	`glance_prm`
GlancePlus PRM By Volume Group metrics (PRM_BYVG_* metrics)	`glance_prm_byvg`
An SQL value or an execution time (walltime) resulting from executing SQL statements against an Oracle instance	`wlmoradc`
You have a program/script that reports metrics to stdout	`wlmrcvdc`
You have metrics you are collecting outside GlancePlus using a shell script or perl program	`wlmsend/wlmrcvdc`
You have metrics you are collecting outside GlancePlus using a C program	WLM API

The following sections explore each of the data collection methods in more detail.

ARM transaction data

For information on:

- How to get transaction data for an application that is already instrumented with ARM API calls to WLM
- How to instrument an application with ARM API calls

see “Sending ARM transaction data from a modified C program” on page 263.

For information on how to simulate transactions, see “Sending ARM transaction data from a script with simulated transactions” on page 269.

GlancePlus application data

You can define GlancePlus applications in the /var/opt/perf/parm file. Defining applications helps you see what is going on within a WLM workload group. You can also configure your WLM workload groups to be considered “applications.” For more information, see “Integrating with OpenView Performance Agent (OVPA) / OpenView Performance Manager (OVPM)” on page 412.

The following example shows an application definition for Oracle database software:

```
application = Oracle
file = ora*, sqldba, sqlplus, tnslsnr, lsnrctl
```

By creating such an entry in the parm file, we can access metrics for all Oracle and associated processes on the system.

Examples of GlancePlus application metrics are:

`APP_ACTIVE_PROC`

The number of processes in a GlancePlus application that are competing for the CPU.

`APP_ALIVE_PROC`

The number of processes that exist in a GlancePlus application.

`APP_CPU_TOTAL_UTIL`

The percentage of the total CPU time devoted to processes in this group during the interval. This indicates the relative CPU load placed on the system by processes in this group.

`APP_MEM_UTIL`

The approximate percentage of the system's physical memory used as resident memory by processes in this group during the interval.

For a list of metrics, see:

- The Application Metrics subsection of the Performance Metrics section in the GlancePlus online help, available through `gpm`
- One of the /opt/perf/paperdocs/gp/C/metrics.* documents
- The glance_app(5) man page (lists some of the more frequently used metrics)

To extract application data (APP_* metrics), use `wlmrcvdc` with the `glance_app` command in the configuration file. For example:

```
tune active_procs {
...
coll_argv = wlmrcvdc glance_app
            APP_ACTIVE_PROC     # Metric to monitor
            Oracle; # Application defined in parm file
...
}
```

For `wlmrcvdc` conceptual information, see "Sending data with `wlmsend` and `wlmrcvdc`: How it works" on page 270. For `wlmrcvdc` syntax information, see "`wlmrcvdc`" on page 393.

GlancePlus global data

Global data provides information on the system as a whole.

Examples of GlancePlus global metrics are:

`GBL_ACTIVE_PROC`	The number of active processes. A process is active if it exists and consumes some CPU.
`GBL_CPU_TOTAL_UTIL`	Percentage of time the CPU was active during the interval.
`GBL_NUM_USER`	The number of users logged into the system.

For a full list of the available data, see the GlancePlus online help, available through `gpm`.

To extract global data (GBL_*), use `wlmrcvdc` with the `glance_gbl` command in the configuration file. For example:

```
tune num_users {
...
coll_argv = wlmrcvdc glance_gbl
             GBL_NUM_USER;        # Metric to monitor
...
}
```

For `wlmrcvdc` conceptual information, see “Sending data with `wlmsend` and `wlmrcvdc`: How it works” on page 270. For `wlmrcvdc` syntax information, see “`wlmrcvdc`” on page 393.

GlancePlus PRM-specific application data

GlancePlus provides application data that is specific to workload (PRM) groups.

Examples of such GlancePlus metrics (APP_* metrics and APP_PRM_* metrics) are:

`APP_ACTIVE_PROC`	The number of processes in a workload (PRM) group that are competing for the CPU.
`APP_PRM_MEM_UTIL`	The percent of PRM memory used by processes (process-private space plus a process's portion of shared memory) within the PRM group during the interval. This metric is only available if the PRM memory manager is enabled.

For a full list of the available data, see the GlancePlus online help, available through `gpm`.

To extract PRM-specific application data, use `wlmrcvdc` with the `glance_prm` command in the configuration file. For example:

```
tune active_procs {
...
coll_argv = wlmrcvdc glance_prm
            APP_ACTIVE_PROC # Metric to monitor
            Grp3;  # Name of workload (PRM) group
...
}

tune mem_util {
...
coll_argv = wlmrcvdc glance_prm
            APP_PRM_MEM_UTIL # Metric to monitor
            Grp3;  # Name of workload (PRM) group
...
}
```

Use `glance_prm` to collect metrics of the form APP_* and APP_PRM_*. Although the APP_* metrics do not appear to be PRM-related, they are actually specific to workload (PRM) groups by virtue of `glance_prm`'s design.

For `wlmrcvdc` conceptual information, see “Sending data with `wlmsend` and `wlmrcvdc`: How it works” on page 270. For `wlmrcvdc` syntax information, see “`wlmrcvdc`” on page 393.

GlancePlus PRM-controlled volume group data

GlancePlus PRM by volume group data provides information on volume groups that are controlled by PRM.

Examples of PRM by volume group metrics are:

`PRM_BYVG_GROUP_UTIL`

A group’s current percentage of disk bandwidth relative to other PRM groups’ usage of the same volume group.

`PRM_BYVG_REQUEST`

The number of Kbytes (or Mbytes if specified) the PRM group requested to have read from or written to the logical volumes in the current volume group during the interval.

For a full list of the available data, see the GlancePlus online help, available through `gpm`.

To extract PRM by volume group data (PRM_BYVG_* metrics), use `wlmrcvdc` with the `glance_prm_byvg` command in the configuration file. For example:

```
tune vg_util {
...
coll_argv = wlmrcvdc glance_prm_byvg
             PRM_BYVG_GROUP_UTIL # Metric to monitor
             Grp17 /dev/vg03;
                   # Name of workload (PRM) group
                   # and logical volume group
...
}
```

For `wlmrcvdc` conceptual information, see “Sending data with `wlmsend` and `wlmrcvdc`: How it works” on page 270. For `wlmrcvdc` syntax information, see “`wlmrcvdc`” on page 393.

Oracle database data

Oracle database data provides SQL values or elapsed time (walltime) resulting from executing SQL statements against an Oracle instance.

Examples of Oracle database metrics are:

- Number of users connected to an instance
- Number of processes in an instance
- A particular user is connected
- A particular job is active

To extract Oracle database data, use `wlmrcvdc` with the `wlmoradc` command in the configuration file:

```
tune oracle.instance1.user_cnt {
    coll_argv =
        wlmrcvdc
          wlmoradc
            --configfile /opt/wlm/toolkits/oracle/config/user_cnt.oradc
            --home /oracle/app/oracle/product/8.1.5
            --instance instance1
    ;
}
```

For more ideas on the metrics you can collect, as well as information on `wlmoradc`, see “Integrating with Oracle® databases” on page 414 or the wlmoradc(1M) man page.

For `wlmrcvdc` conceptual information, see “Sending data with `wlmsend` and `wlmrcvdc`: How it works” on page 270. For `wlmrcvdc` syntax information, see “`wlmrcvdc`” on page 393.

Existing metrics

If you already maintain application-specific metrics or system metrics, you can send that data to WLM in one of two ways:

- Using the `wlmsend` utility
- Using the WLM API

Both the `wlmsend` utility and the API are explained in the section “What methods exist for sending data to WLM?” on page 250.

What methods exist for sending data to WLM?

The section focuses on how you want to get data to WLM, then determines what data is available for the given type of transport.

There are several methods for sending data to WLM:

- Sending data from the command line
- Sending data from a shell script
- Sending data from a perl program
- Sending data via stdout
- Sending data from a collector written in C
- Sending ARM transaction data from a modified C program
- Sending ARM transaction data from a script with simulated transactions

As your data collector gathers data, it must send the data to WLM. The easiest way to send data is by using the `wlmrcvdc` utility. Alternatively, you can send this data using the WLM API.

How you send the data is generally decided by how you collect the data. If you collect data using shell scripts or perl programs, the `wlmsend` and `wlmrcvdc` utilities provide the most convenient route for sending data. However, when using C programs to collect data, the WLM API is typically better.

Table 7-2 provides an overview of how to send your data to WLM. For more information on the transport methods, read the sections following the table.

Table 7-2 Available transport methods

For sending data from	The transport method is

Command-line/shell script	On the command line or in a loop in a script:
	`wlmsend` *`metric value`*
	or
	`cmnd1` `|` *`cmnd2`* `| [... |] wlmsend` *`metric`*
	In combination with a `tune` structure in the configuration file:
	`tune` *`metric`* `{`
	`...`
	`   coll_argv = wlmrcvdc;`
	`...`
	`}`

Perl	In a loop in a perl program:
	`system "wlmsend $`*`metric`* `$`*`value`*`";`
	or an `open` statement with the `print` statement in a loop
	`open(WLM, "|wlmsend $`*`metric`*`");`
	`print WLM "$`*`value`*`\n";`
	In combination with a `tune` structure in the configuration file:
	`tune` *`metric`* `{`
	`...`
	`   coll_argv = wlmrcvdc;`
	`...`
	`}`

Table 7-2 Available transport methods (Continued)

For sending data from	The transport method is

stdout of *command*	A `tune` structure in the configuration file:
	`tune metric {`
	`...`
	`   coll_argv = wlmrcvdc command;`
	`...`
	`}`

program that uses WLM API (libwlm.sl)	A `tune` structure in the configuration file:
	`tune metric {`
	`...`
	`   coll_argv = program;`
	`...`
	`}`

Sending data from the command line

If you have data you want to send to WLM from the command line, use `wlmsend` on the command line and an associated `wlmrcvdc` in your configuration file. This method of sending data works for a single metric value that is specified on the `wlmsend` command line or for a stream of metric values that is piped to `wlmsend`, as shown below:

```
/opt/wlm/bin/wlmsend metric value
```

or

```
cmnd1 | cmnd2 | [... |] /opt/wlm/bin/wlmsend metric
```

Place a `wlmrcvdc` in the configuration file that is associated with the `wlmsend` by *metric*:

```
tune metric {
...
   coll_argv = wlmrcvdc;
...
}
```

In the following example, we have an `slo` structure to specify our goal. The `tune` structure invokes `wlmrcvdc` to monitor `job_time` values forwarded by `wlmsend`:

```
# Set up the SLO
slo data_cruncher {
   pri = 3;
   mincpu = 15;
   maxcpu = 25;
   entity = PRM group crunch;
   goal = metric job_time < 2.0;
}

# Set up wlmrcvdc
tune job_time {
   coll_argv = wlmrcvdc;
}
```

On the command line below, some program is updating *`logfile`*. With the `tail` command, `wlmsend` gets the latest updates to the file and sends them to a rendezvous point for `wlmrcvdc` to collect:

```
# tail -f logfile | /opt/wlm/bin/wlmsend job_time
```

This rendezvous point is created by `wlmrcvdc`.

For background information on how `wlmsend` and `wlmrcvdc` work together to get your data to WLM, as well as a warning about I/O buffering, see “Sending data with `wlmsend` and `wlmrcvdc`: How it works” on page 270.

Sending data from a shell script

If you have data you want to send to WLM from a shell script, use `wlmsend` in the shell script and an associated `wlmrcvdc` in your configuration file. This method of sending data works for a single metric value that is specified as an argument to `wlmsend` or for a stream of metric values that is piped to `wlmsend`, as shown below.

In this example, start in the configuration file by defining an `slo` structure to keep the metric `job_time` under 2.0, and then set up `wlmrcvdc` to receive the `job_time` metric values from `wlmsend`:

```
# Set up the SLO
slo data_cruncher {
   pri = 3;
   mincpu = 15;
   maxcpu = 25;
   entity = PRM group crunch;
   goal = metric job_time < 2.0;
}

# Set up wlmrcvdc
tune job_time {
   coll_argv = wlmrcvdc;
}
```

Next, set up `wlmsend` in a loop in a shell script. Here, the function `get_value` provides the metrics to feed `wlmsend`. Alternatively the script itself could be interacting with the workload to gather performance data. Add a `sleep` command to slow down how often metrics are retrieved. The loop might look like the following:

```
while (true)
do
   value = `get_value`
   /opt/wlm/bin/wlmsend job_time $value
   sleep 60
done
```

NOTE Run your data collection script in the group `PRM_SYS` to ensure it receives the proper resources.

For background information on how `wlmsend` and `wlmrcvdc` work together to get your data to WLM, as well as a warning about I/O buffering, see “Sending data with `wlmsend` and `wlmrcvdc`: How it works” on page 270.

Sending data from a perl program

If you have data you want to send to WLM from a perl program, use `wlmsend` in the perl program and an associated `wlmrcvdc` in your WLM configuration file. This method of sending data works for a single metric value that is specified as an argument to `wlmsend` or for a stream of metric values that is piped to `wlmsend`, as shown below.

In this example, start in the configuration file by defining an `slo` structure to keep the metric `job_time` under 2.0, and then set up `wlmrcvdc` to receive the `job_time` metric values from `wlmsend`:

```
# Set up the SLO
slo data_cruncher {
   pri = 3;
   mincpu = 15;
   maxcpu = 25;
   entity = PRM group crunch;
   goal = metric job_time < 2.0;
}

# Set up wlmrcvdc
tune job_time {
   coll_argv = wlmrcvdc;
}
```

Next, set up `wlmsend` in a loop in a perl program. Here, the function `get_value` provides the metrics to feed `wlmsend`. Alternatively the program itself could interact with the workload to gather performance data. Add a `sleep` command to slow down how often metrics are retrieved. The loop might look the following:

```
while (1) {
   $value = &get_value;
   system "/opt/wlm/bin/wlmsend job_time $value";
   sleep(60);
}
```

The loop could also look like the following loop, using a `print` statement instead of a system statement:

```
open(WLM, "| /opt/wlm/bin/wlmsend job_time");
# make the new file descriptor unbuffered
select ((select(WLM), $|=1)[0]);
while (1) {
   $value = &get_value;
   print WLM "$value\n";
   sleep(60);
}
```

NOTE Run your data collection script in the group `PRM_SYS` to ensure it receives the proper resources.

For background information on how `wlmsend` and `wlmrcvdc` work together to get your data to WLM, as well as a warning about I/O buffering, see "Sending data with `wlmsend` and `wlmrcvdc`: How it works" on page 270.

Sending data via stdout

You can use `wlmrcvdc` to forward the stdout of a command to WLM. If your data collector program prints its metrics to stdout, you can run the program by invoking it with `wlmrcvdc` in the WLM configuration file. `wlmrcvdc` automatically forwards stdout to WLM.

NOTE If the command exits with status zero, you can use `wlmsend` to continue feeding data to the rendezvous point for `wlmrcvdc`. (`wlmrcvdc` creates the rendezvous point when it is invoked by WLM.)

Consider the following example in which an instance of `wlmrcvdc` starts a data collector named `ordercounter`. As `ordercounter` writes values to its sdtout, `wlmrcvdc` reads those values and forwards them to the WLM daemon on behalf of the metric `order_cnt`:

```
tune order_cnt books {
   coll_argv = wlmrcvdc /opt/books/ordercounter;
   ...
}
```

WLM provides a number of commands to use with `wlmrcvdc` to forward GlancePlus metrics to WLM. For information on those commands, see:

- "ARM transaction data" on page 244
- "GlancePlus application data" on page 244
- "GlancePlus global data" on page 246
- "GlancePlus PRM-specific application data" on page 247
- "GlancePlus PRM-controlled volume group data" on page 248
- "Oracle database data" on page 249

For background information on how `wlmrcvdc` gets your data to WLM, see "Sending data with `wlmsend` and `wlmrcvdc`: How it works" on page 270.

Sending data from a collector written in C

To send metric information to WLM from a data collector written in C, you can:

- Develop a program that writes its data to stdout and use it with `wlmrcvdc`

  For information on this approach, see "Sending data via stdout" on page 256.

- Use the WLM API

  This section discusses the API.

NOTE You can also write a program that invokes the `wlmsend` command. However, if you are writing a program, it is recommended that you use `wlmrcvdc` to capture stdout or you use the API.

For information on how to set up your WLM configuration to receive this data, see Table 7-2 on page 251.

This API includes three functions. The function prototypes and return codes are defined in the include file /opt/wlm/include/wlm.h.

The three C functions are:

- `wlm_mon_attach()`

  Opens communication to the WLM daemon.
- `wlm_mon_write()`

  Writes data to the WLM daemon.
- `wlm_mon_detach()`

  Closes communication to the WLM daemon.

The compile line for a program using these functions is:

```
% cc -I/opt/wlm/include -L/opt/wlm/lib -lwlm [ flag ... ] file ...
```

NOTE The stdout and stderr for each data collector that WLM starts are discarded. However, WLM provides the `coll_stderr` tunable that allows you to log each collector's stderr. In addition, data collectors can communicate using either syslog(3C) or logger(1) with the daemon facility.

For information on signal handling, see "Handling signals in data collectors" on page 276.

Run your data collection program in the group `PRM_SYS` to ensure it receives the proper resources.

Each data collector should send data to WLM once per WLM interval, if possible. (For information on this interval, see "Specifying the WLM interval (optional)" on page 203.) If the data collector sends data to WLM less than once per interval (meaning an entire interval passes with no new data), the associated controller requests the same allocation as it did for the previous interval. When your data collector sends data more than once per interval, WLM only receives the last data sent before the interval ended. All the other data sent during the interval is lost. In this case, your data should reflect performance over a period of time rather than at a single point in time.

NOTE WLM runs data collectors with UID set to root. If a data collector that uses the WLM API changes its UID, all subsequent calls to the WLM API will fail.

NOTE If a data collector exits unexpectedly, restart it by re-activating the configuration:

```
# wlmd -a configfile
```

Re-activating the configuration resets all resource allocations, causing WLM to reconverge on the allocations needed to meet the configuration's SLOs.

The API functions are described in the following sections.

Opening communications with `wlm_mon_attach()`

The data collector uses the `wlm_mon_attach()` function to open a connection to the WLM daemon in order to send performance data to the daemon.

To use this function, you must reference the include file /opt/wlm/include/wlm.h.

NOTE This function is not thread-safe: Multiple threads cannot call the function at the same time.

The syntax is:

```
int wlm_mon_attach(char *metric);
```

where

metric

Is the name of the metric being passed on this connection. This string must exactly match the *metric* string used in the configuration file, in the specification of the goal.

For convenience, when the WLM daemon starts the data collector executable, it places the name of the metric it is expecting from the collector in the `WLM_METRIC_NAME` environment variable. Using this, a collector that is capable of providing multiple metrics can determine which metric the daemon is expecting.

The exit status values for `wlm_mon_attach()` are:

`>= 0`	Success. Identifies the communication channel for passing data for the metric specified in the *metric* argument. Use this handle in subsequent calls to the API.
`-1`	Failure

Example:

```
handle_id = wlm_mon_attach(getenv("WLM_METRIC_NAME"));
if (handle_id == -1) {
    fprintf(output, "wlm_mon_attach failed: %s\n", strerror(errno));
    exit(1);
}
```

Sending data with `wlm_mon_write()`

The data collector uses the `wlm_mon_write()` function to write messages to the communication channel specified by the supplied *`handle_id`* parameter.

To use this function, you must reference the include file /opt/wlm/include/wlm.h.

NOTE This function is not thread-safe: Multiple threads cannot call the function at the same time.

The syntax is:

```
ssize_t wlm_mon_write(int handle_id, void *buf, size_t numbytes);
```

where

`handle_id`

Is the handle identifier returned by the `wlm_mon_attach()` call.

`buf`

Is a pointer to the double-precision floating-point value to be transferred to the data queue.

`numbytes`

Is the number of bytes to write to the data queue. The data must be of type `double`. Thus, the value of this parameter must be `sizeof(double)`.

The exit status values for `wlm_mon_write()` are:

`>= 0`	Success (number of bytes written)
`-1`	Failure—if `errno` is not `EAGAIN`, your data collector should exit

Example:

```
if (metric_cnt > 0) {
    metric_avg = metric_sum / metric_cnt;
    if (wlm_mon_write(handle_id, &metric_avg,
        sizeof(double)) == -1) {
        fprintf(output, “wlm_mon_write failed: %s\n”, strerror(errno));
        exit(1);
    }
}
```

Closing communications with `wlm_mon_detach()`

The data collector uses the `wlm_mon_detach()` function to close (detach from) a communication channel with the WLM daemon.

To use this function, you must reference the include file /opt/wlm/include/wlm.h.

NOTE This function is not thread-safe: Multiple threads cannot call the function at the same time.

The syntax is:

`int wlm_mon_detach(int` *`handle_id`*`);`

where

`handle_id`	Is the communication channel identifier returned by the `wlm_mon_attach()` call.

The exit status values for `wlm_mon_detach()` are:

`0`	Success
`-1`	Failure

Sending ARM transaction data from a modified C program

ARM, or Application Response Measurement, was developed to provide an open, vendor-neutral approach for measuring end-to-end application response time. ARM consists of six library calls that are inserted in application source code. These calls isolate transactions so that you can collect data on and manage these transactions proactively.

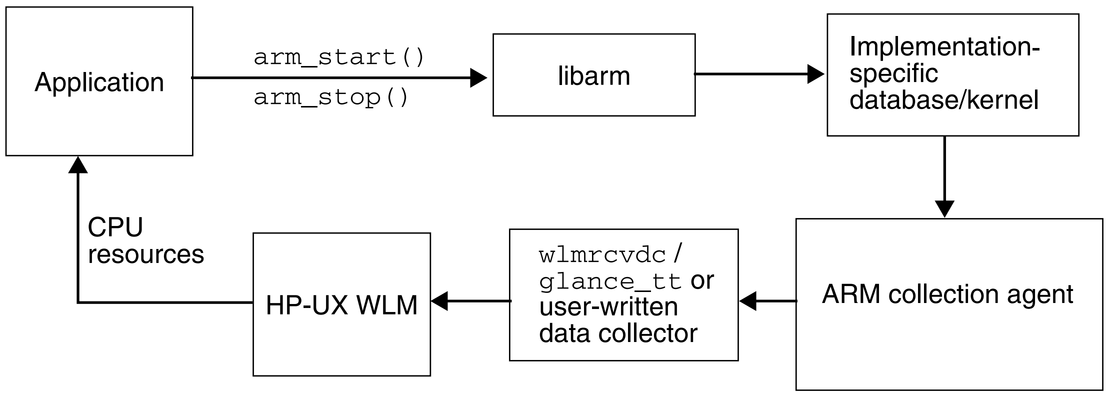
If you have the C source code to an application and are inclined to modify the source, you can instrument the application with ARM API calls to measure the events or transactions that are most important to your business.

NOTE HP's ARM implementation is available through GlancePlus. Install GlancePlus on each system where you want to track ARM data.

If you are using HP's ARM implementation, you can send transaction data to WLM using `wlmrcvdc` with the `glance_tt` command.

Figure 7-1 presents an overview of how an application that is instrumented with ARM API calls works with WLM. First, the application invokes the ARM API calls, made available through libarm. libarm then feeds the data from the ARM calls to an implementation-specific storage facility. An ARM collection agent, in this case GlancePlus, then picks up this data. Next, `wlmrcvdc` / `glance_tt` or a user-written data collector sends the relevant data to WLM to use in determining CPU allocations. The new allocations then affect the application's performance, which is again tracked by the ARM API calls.

Figure 7-1 **Interaction of WLM and ARM-instrumented applications**


Application
arm_start()
arm_stop()
libarm
Implementation-specific database/kernel
ARM collection agent
wlmrcvdc / glance_tt or user-written data collector
HP-UX WLM
CPU resources

Examples of transaction metrics are:

`TT_WALL_TIME`

The total time, in seconds, of all transactions completed during the last interval for this transaction.

`TT_WALL_TIME_PER_TRAN`

The average transaction time, in seconds, during the last interval for this transaction.

For a list of metrics, see:

- The Transaction Metrics subsection of the Performance Metrics section in the GlancePlus online help, available through `gpm`
- One of the /opt/perf/paperdocs/gp/C/metrics.* documents
- The glance_tt(5) man page (lists some of the more frequently used metrics)

The ARM API consists of six routines, which are described in the following table. For complete usage information on these routines, see the arm(3) man page, which is installed when you install GlancePlus. (To see this man page, be sure "/opt/perf/man" is part of your `MANPATH` environment variable.)

Table 7-3 ARM API routines

ARM API routine	Description
`arm_init`	Initializes the ARM environment for the application
`arm_getid`	Names each transaction that is to be monitored
`arm_start`	Indicates the start of a transaction
`arm_update`	Updates statistics for a long-running transaction
`arm_stop`	Indicates the end of a transaction
`arm_end`	Prepares the ARM environment for the application's exit

You add the API calls to your source, isolating the desired transactions. For example, consider the following pseudo-code that tracks individual times for various tasks and also gets a time for all three processes together:

```
arm_init(“application_name”, “user_name”)
   arm_getid(“total_time”)
   arm_getid(“pull_data”)
   arm_getid(“process_data”)
   arm_getid(“report_data”)

   loop until end of program
      arm_start(total_time)

         arm_start(pull_data)
         (get the data)
         arm_stop(pull_data, status)

         arm_start(process_data)
         (process the data)
         arm_stop(process_data, status)

         arm_start(report_data)
         (generate report)
         arm_stop(report_data, status)

      arm_stop(total_time, status)
   end loop
arm_end
```

To use ARM transaction data with WLM:

Step 1. Modify your program to use the ARM API calls described above. To use these calls, you must reference the include file /opt/perf/include/arm.h.

Be sure to:

- Register *ARM_app* with ARM through the `arm_init` API call
- Mark the start and stop of each transaction in which you are interested

Step 2. Compile your program, specifying the locations for the ARM header file and library—being sure to link against the ARM library.

For HP-UX 11.0 and HP-UX 11i v1 (B.11.11), the compile/link line is:

```
% /opt/ansic/bin/cc -Ae prog.c -I /opt/perf/include -L /opt/perf/lib -larm
```

The `-Ae` option ensures you compile using the extended ANSI mode.

For HP-UX 11i v2 (B.11.23) on Itanium-based platforms, the compile/link line for 32-bit IA ARMed programs is:

```
% /opt/ansic/bin/cc -Ae prog.c -I /opt/perf/include \
-L /opt/perf/lib/hpux32 -larm
```

For HP-UX 11i v2 on Itanium-based platforms, the compile/link line for 64-bit IA ARMed programs is:

```
% /opt/ansic/bin/cc -Ae +DD64 prog.c -I /opt/perf/include \
-L /opt/perf/lib/hpux64 -larm
```

Step 3. Ensure that `ttd` is configured and running. For more information, see ttd(1).

Step 4. Start the program you instrumented with ARM calls.

The start and stop times for your transactions will now be made available through HP's ARM implementation. For more information, see ttd(1) and midaemon(1).

Step 5. Edit your WLM configuration to pick up the transaction data.

To extract ARM data (from HP's ARM implementation) and forward it to WLM, use `wlmrcvdc` with the `glance_tt` command in the configuration file:

```
tune metric {
...
coll_argv = wlmrcvdc glance_tt
            TT_*_metric  # Metric you choose to monitor
            ARM_app  # Application registered with ARM
            transaction [user]; # Transaction in the
            # application to get the metric for, along
            # with an application user
...
}
```

The metric you choose, generically referred to as *`TT_*_metric`* above, is typically from the Transaction metric class (TT_*), but can also be an expression that combines a transaction metric with metrics from the Global (GBL_*) and Table (TBL_*) metric classes.

You can fine-tune transaction tracking in the /var/opt/perf/ttd.conf file if desired. This file has an entry (`tran=*`) that enables all transactions to be tracked. However, you may want to fine-tune what is tracked. For more information, see the ttd(1) man page and the ttd.conf file or the ttd.conf(4) man page.

For more information on the `wlmrcvdc` utility, see "Sending data via stdout" on page 256.

Step 6. Activate your WLM configuration:

```
# wlmd -a configfile
```

For information on the ARM API and the ARM software developer's kit, see:

http://www.cmg.org/regions/cmgarmw

Sending ARM transaction data from a script with simulated transactions

Using transactions that simulate the key transactions of your workload can simplify data collecting. These transactions may not provide the fine granularity that you can achieve by placing ARM API calls in the source code. However, you do avoid modifying the application with instrumentation.

Set up simulated transactions as follows:

Step 1. Create a simulation. For example, use remote terminal emulation (RTE) software to capture sessions in which key transactions of the workload (that WLM is going to manage) are executed.

Step 2. Save each captured session to a file, then add ARM API calls to the file. (In some cases, the RTE software produces files that are editable as C code.)

Step 3. Edit the WLM configuration file so that:

- Your data collector tracks the performance of the script
- The script is assigned to the same workload group as the application that is to be managed

Step 4. Run the script and the application to be managed.

Step 5. Send the ARM data to WLM using the `wlmrcvdc` utility described in the section “Sending ARM transaction data from a modified C program” on page 263.

For additional information on using RTE software, see the white paper “Service Management Using the Application Response Measurement API Without Application Source Code Modification” on the following web site:

http://www.cmg.org/regions/cmgarmw

Sending data with wlmsend and wlmrcvdc: How it works

The simplest way to get data to WLM is to use `wlmsend` and `wlmrcvdc`, or in some cases, just `wlmrcvdc`. These utilities allow you to send data to WLM from the command line, shell scripts, perl programs, and stdout.

Use `wlmsend` on the command line, in a shell script, or in a perl program. Use `wlmrcvdc` in the WLM configuration file.

NOTE Avoid the process overhead of using `wlmsend` in a compiled data collector; let `wlmd` invoke your data collector through the configuration file if possible. Use the API described in "Sending data from a collector written in C" on page 257 to write such a data collector.

The `wlmrcvdc` utility creates a rendezvous point and forwards data from that rendezvous point to WLM. Use `wlmsend` to send data to a rendezvous point to be collected by a `wlmrcvdc` invocation.

If you use `wlmsend`, you must also use `wlmrcvdc`. You can, however, use `wlmrcvdc` without `wlmsend` if you specify `wlmrcvdc` with a data-collecting command.

The syntaxes for the commands follow:

```
wlmrcvdc [-h] [-V] [-u user] [-g group] [command [args...]]
```

```
wlmsend [-h] [-V] [-w wait_time] metric [value]
```

For an explanation of the syntaxes, see "WLM command reference" on page 373.

NOTE Be careful of I/O buffering when feeding data to `wlmsend`. For example, the following line works as expected:

```
tail -f logfile | /opt/wlm/bin/wlmsend job_time
```

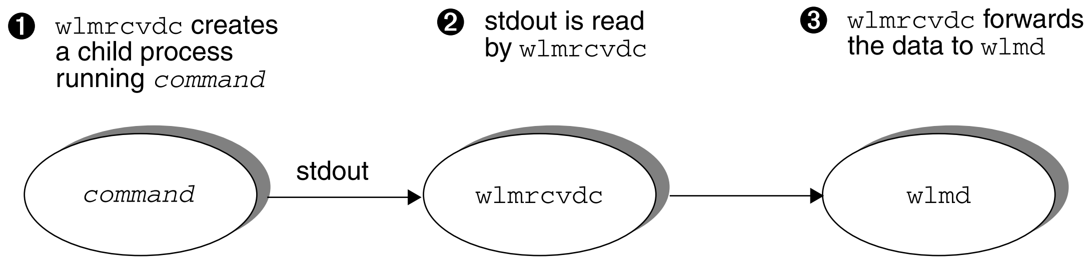
However, adding `awk` or other utilities that buffer I/O can result in data being delayed, or not even sent, to `wlmsend`—depending on when the buffer is flushed:

```
tail -f logfile | awk '{print $1}' | /opt/wlm/bin/wlmsend job_time
```

Always check the file /var/opt/wlm/wlmdstats, created by `wlmd -l metric` as explained in wlmd(1M), to ensure that your data gets to WLM in a timely fashion.

Figure 7-2 illustrates the command-pipe mode of operation for `wlmrcvdc`. In this mode, `wlmrcvdc` starts a command, reads its stdout, and forwards that data to the WLM daemon, `wlmd`.

Figure 7-2 **`wlmrcvdc`: command-pipe mode (`wlmrcvdc` *`command`* `[`*`args`*`]`)**


❶ wlmrcvdc creates a child process running command
❷ stdout is read by wlmrcvdc
❸ wlmrcvdc forwards the data to wlmd
command
stdout
wlmrcvdc
wlmd

Consider the following example, in which an instance of `wlmrcvdc` starts a data collector named `ordercounter`. As `ordercounter` writes values to its sdtout, `wlmrcvdc` reads those values and forwards them to the WLM daemon on behalf of the metric `order_cnt`:

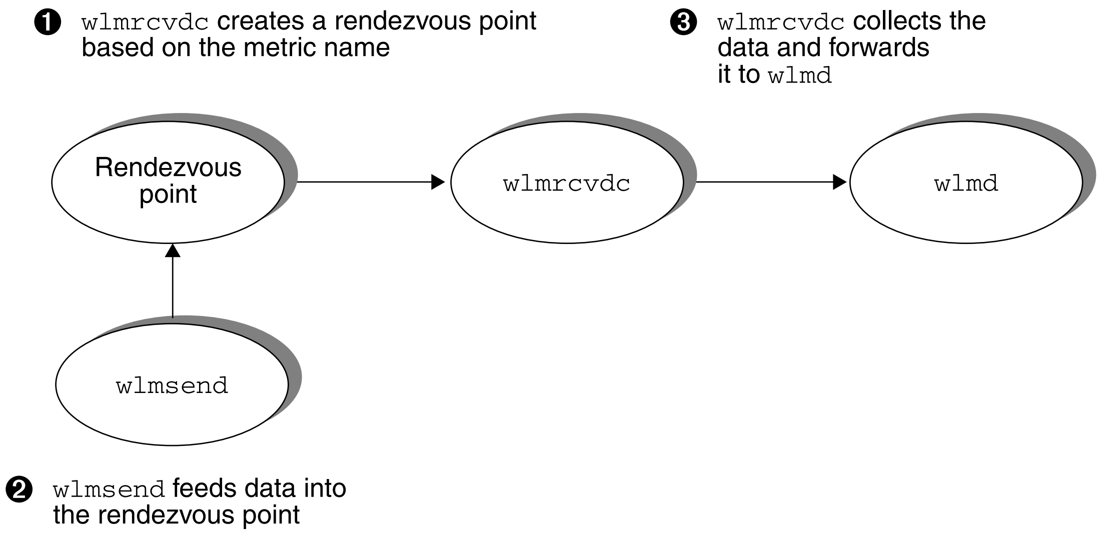
```
tune order_cnt books {
   coll_argv = wlmrcvdc /opt/books/ordercounter;
   ...
}
```

While using the program `ordercounter` to send data, you can also use `wlmsend` to forward data for the metric `order_cnt`.

Figure 7-3 illustrates the FIFO-file mode of operation for `wlmrcvdc`. In this mode, `wlmrcvdc` monitors a FIFO rendezvous point it created. `wlmsend` makes data available at the rendezvous point. `wlmrcvdc` picks up the data and forwards it to the WLM daemon.

Each `wlmrcvdc` process has its own FIFO rendezvous point based on the metric name used in the `tune` structure that invokes that particular `wlmrcvdc` process. `wlmsend` determines the correct rendezvous point from its metric argument.

Figure 7-3 **`wlmrcvdc`: FIFO-file mode (`wlmrcvdc`)**


❶ wlmrcvdc creates a rendezvous point based on the metric name
❸ wlmrcvdc collects the data and forwards it to wlmd
Rendezvous point
wlmrcvdc
wlmd
wlmsend
❷ wlmsend feeds data into the rendezvous point

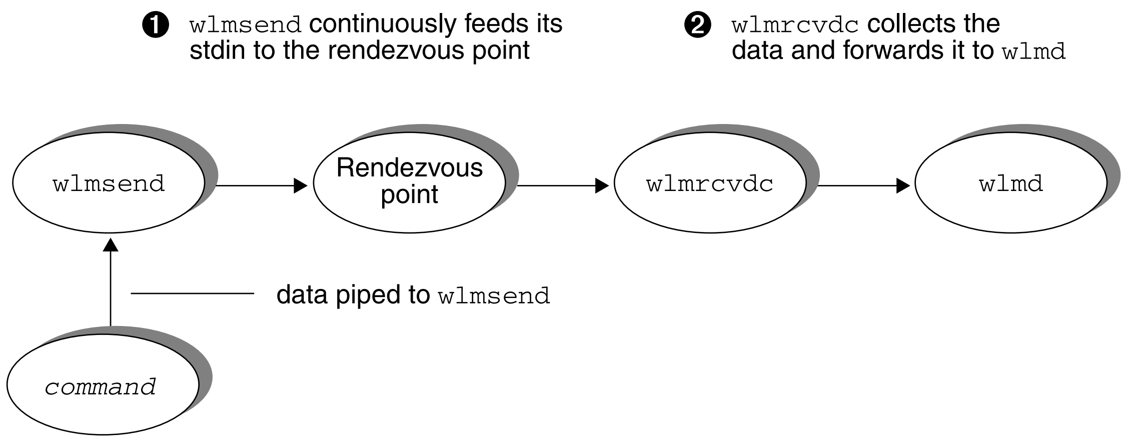
In this next example, when the WLM configuration file is activated, `wlmrcvdc` creates a rendezvous point, with a name based on the metric name `resp_time`. The rendezvous point is owned by the user `admin` in the group `sys`:

```
tune resp_time {
   coll_argv = wlmrcvdc -u admin -g sys;
}
```

Now all that remains is for you to use `wlmsend` in a data collector script to forward values to the established FIFO rendezvous point.

Figure 7-4 illustrates the command-output mode of operation for `wlmsend`. In this mode, `wlmsend` forwards its stdin to the rendezvous point that `wlmrcvdc` monitors. `wlmrcvdc` then sends the data on to WLM.

Figure 7-4 **wlmsend: command-output mode (*command* | wlmsend *metric*)**


❶ wlmsend continuously feeds its stdin to the rendezvous point
❷ wlmrcvdc collects the data and forwards it to wlmd
wlmsend
Rendezvous point
wlmrcvdc
wlmd
data piped to wlmsend
command

This example of `wlmsend`'s command-output mode shows how to set up the WLM configuration file and `wlmsend` to work together to forward the data to WLM.

First, in the configuration file, we define an `slo` structure to keep the metric `job_time` under 2.0. Also, we set up `wlmrcvdc` to receive the `job_time` metric values from `wlmsend`:

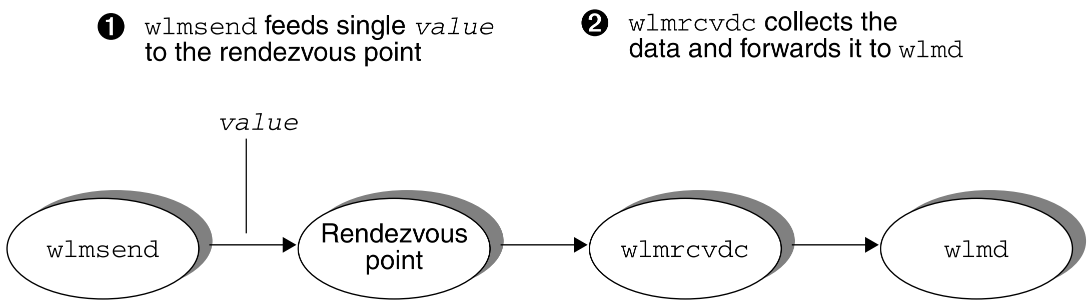
```
# Set up the SLO
slo data_cruncher {
   pri = 3;
   mincpu = 15;
   maxcpu = 25;
   entity = PRM group crunch;
   goal = metric job_time < 2.0;
}

# Set up wlmrcvdc
tune job_time {
   coll_argv = wlmrcvdc;
}
```

Second, we set up `wlmsend` in a data collector shell script, a perl program, or on the command line to feed data to the rendezvous point. If the data is coming from a file, say *logfile*, we can forward the data to WLM as follows:

```
# tail -f logfile | /opt/wlm/bin/wlmsend job_time
```

Figure 7-5 illustrates the single-value mode of operation for `wlmsend`. In this mode, `wlmsend` is repeatedly invoked with a single metric value. However, it is invoked in a loop in a shell script or a perl program that updates that metric value.

Figure 7-5 **`wlmsend`: single-value mode (`wlmsend` *`metric value`*)**


❶ wlmsend feeds single value to the rendezvous point
❷ wlmrcvdc collects the data and forwards it to wlmd
value
wlmsend
Rendezvous point
wlmrcvdc
wlmd

In this example, the setup is the same as in the previous example. Thus, in the configuration file, we define an `slo` structure to keep the metric `job_time` under 2.0, and we set up `wlmrcvdc` to receive the `job_time` metric values from `wlmsend`:

```
# Set up the SLO
slo data_cruncher {
   pri = 3;
   mincpu = 15;
   maxcpu = 25;
   entity = PRM group crunch;
   goal = metric job_time < 2.0;
}

# Set up wlmrcvdc
tune job_time {
   coll_argv = wlmrcvdc;
}
```

Next, we set up `wlmsend` in a loop in either a shell script or a perl program. The function `get_value` provides the metrics to feed `wlmsend`. We add a `sleep` command to slow down how often metrics are retrieved. For a shell script, that loop might look like the following:

```
while (true)
do
   value = ‘get_value‘
   /opt/wlm/bin/wlmsend job_time $value
   sleep 60
done
```

In a perl program, the loop might look like one of the following:

```
while (1) {
   $value = &get_value;
   system “/opt/wlm/bin/wlmsend job_time $value”;
   sleep(60);
}
```

Using a `print` statement instead of a `system` statement:

```
open(WLM, “| /opt/wlm/bin/wlmsend $metric”);
# make the new file descriptor unbuffered
select ((select(WLM), $|=1)[0]);
while (1) {
   $value = &get_value;
   print WLM “$value\n”;
   sleep(60);
}
```

Handling signals in data collectors

When a data collector is spawned, it inherits the signal handling actions that the WLM daemon inherits. (Ordinarily, these would be the default actions described in the signal(5) man page.) Additionally, when the data collector process begins execution, its inherited signal mask should be 0, so that no signals are currently being blocked.

Currently, `SIGTERM` is the only signal that is delivered by WLM to the spawned data collectors.

WLM uses the signal `SIGTERM` to terminate a spawned data collector when it is no longer required. A well-behaved data collector will install a handler function for `SIGTERM`, so that `wlm_mon_detach()` can be called (if attached) as part of its cleanup. This is the recommended approach, though it is not required to explicitly detach. In any case, `SIGTERM` should not be ignored (that is, set to `SIG_IGN`).

8 Auditing and billing

WLM produces audit information when you activate a configuration using the `-t` option with either the WLM daemon `wlmd` or the WLM global arbiter daemon `wlmpard`:

```
# wlmd -t -a configfile
```

or

```
# wlmpard -t -a configfile
```

Once you've activated a configuration using `-t`, use the `wlmaudit` command to display the audit data:

```
# wlmaudit
```

The `wlmaudit` command allows you to specify a date range for the data to display. By default, the output is plain text; however, you can display output in formatted HTML as well.

For command syntax, see the "WLM command reference" on page 373 or the wlmaudit(1M) man page.

NOTE In order to ensure the smooth running of the operating system and key HP-UX daemons, WLM runs these processes in a special workload group called `PRM_SYS`. This group is not restrained by WLM: It consumes system resources as needed. As a result, a workload group's CPU usage may be less than its allocation (or entitlement as it is seen in the `wlmaudit` report) because `PRM_SYS` requires some of the group's resources. However, the low usage could also be the result of the group's low CPU demands, or a combination of the two factors. Also, there may be times when CPU usage is slightly above the allocation due to dynamics in the CPU scheduler that WLM uses.

Example `wlmaudit` report

Here is a sample `wlmaudit` report. (Periods after January are removed for brevity.)

```
Date: 08/26/2003
Host: host1
Subject: WLM Audit Report

Summary for wlmd on host1 from 01/01/2003 to 08/26/2003:

                                Duration  CPU Entitlement      CPU Usage
    Entity                          (hr)        (CPU-hr)        (CPU-hr)
    --------------------------------------------------------------------
    OTHERS                      3758.526       16638.380        1039.816
    PRM_SYS                     3758.526        1389.540         917.580
    _IDLE_                      2928.954         159.613           0.000
    g_apache                    3758.526        2040.202          31.853
    g_nice                      3758.526        2093.390         504.050
    g_nightly                   3758.526         406.796         156.060
    g_team                      3758.526         984.799           0.210


Detail audit report for wlmd on host1:

    Period: Jan 2003
    ----------------
        Entity: OTHERS
        Daily records:

            Date             Duration    Entitlement          Usage
            -------------------------------------------------------
            01/24/2003          1.663          5.338          0.111
            01/25/2003         24.000        109.727          1.982
            01/26/2003         24.000        109.565          1.122
            01/27/2003         24.000        105.233         13.105
            01/28/2003         24.000        103.373         10.297
            01/29/2003         24.000        102.295          9.128
            01/30/2003         24.000        104.443         15.542
            01/31/2003         24.000        100.606          8.371
```

```
Entity: PRM_SYS
Daily records:

    Date                Duration    Entitlement           Usage
    -------------------------------------------------------
    01/24/2003             1.663          3.052           0.508
    01/25/2003            24.000         29.685           4.938
    01/26/2003            24.000         29.448           4.877
    01/27/2003            24.000         33.425           5.507
    01/28/2003            24.000         35.716           5.872
    01/29/2003            24.000         36.154           5.960
    01/30/2003            24.000         39.523           6.469
    01/31/2003            24.000         40.943           6.763

Entity: g_apache
Daily records:

    Date                Duration    Entitlement           Usage
    -------------------------------------------------------
    01/24/2003             1.663          0.545           0.007
    01/25/2003            24.000          7.338           0.030
    01/26/2003            24.000          7.200           0.028
    01/27/2003            24.000         11.487           0.228
    01/28/2003            24.000         13.285           0.307
    01/29/2003            24.000         13.835           0.371
    01/30/2003            24.000         12.024           0.253
    01/31/2003            24.000         13.412           0.389

Entity: g_nice
Daily records:

    Date                Duration    Entitlement           Usage
    -------------------------------------------------------
    01/24/2003             1.663          2.283           1.032
    01/25/2003            24.000         11.824           2.428
    01/26/2003            24.000         12.490           2.563
    01/27/2003            24.000         11.609           2.550
    01/28/2003            24.000         11.973           2.788
    01/29/2003            24.000         12.232           2.884
    01/30/2003            24.000         11.779           2.852
    01/31/2003            24.000         13.544           3.439
```

```
Entity: g_nightly
Daily records:

    Date                Duration    Entitlement            Usage
    -------------------------------------------------------
    01/24/2003             1.663          0.826            0.000
    01/25/2003            24.000          3.671            1.035
    01/26/2003            24.000          3.401            0.984
    01/27/2003            24.000          4.400            0.936
    01/28/2003            24.000          3.696            1.170
    01/29/2003            24.000          3.785            1.125
    01/30/2003            24.000          3.711            1.085
    01/31/2003            24.000          3.716            1.096

Entity: g_team
Daily records:

    Date                Duration    Entitlement            Usage
    -------------------------------------------------------
    01/24/2003             1.663          0.478            0.000
    01/25/2003            24.000          6.503            0.000
    01/26/2003            24.000          6.467            0.000
    01/27/2003            24.000          5.764            0.000
    01/28/2003            24.000          5.802            0.000
    01/29/2003            24.000          5.894            0.000
    01/30/2003            24.000          5.576            0.000
    01/31/2003            24.000          5.960            0.000
```

Audit data files

`wlmaudit` takes the comma-separated data from audit files and displays it in an easily readable report. However, if you would like to use these files directly, they are located in the directory /var/opt/wlm/audit/. The files are named based on the daemon of origin and the date: wlmd.*monyyyy* and wlmpard.*monyyyy*, with *monyyyy* representing the month and year, as in nov2001.

Here is an example entry from one of these files:

```
host1, 10/10/2002, group1, PRM, 10.000000, 0.000417, 0.000153
```

Each line has seven fields:

Field 1	The name of the host from which the data was gathered
Field 2	The date (mm/dd/yyyy) the data was gathered
Field 3	The name of the entity on which the data is focused
Field 4	The type of entity. Possible types are:

NPAR	an nPartition
PRM	a PRM group (also known as a workload group)
VPAR	a virtual partition

Field 5	Approximate number of hours of data available for the entity
Field 6	The CPU entitlement, or allocation, for the entity for the date shown (in CPU hours)
Field 7	The CPU usage for the entity for the date shown (in CPU hours)

Enabling auditing at reboot

You can automatically set the WLM daemon, `wlmd`, and the WLM global arbiter daemon, `wlmpard`, to generate audit data by setting the following variables to 1 in the file /etc/rc.config.d/wlm:

- `WLMD_AUDIT_ENABLE`
- `WLMPARD_AUDIT_ENABLE`

Both variables are set to 0 (no audit data) by default.

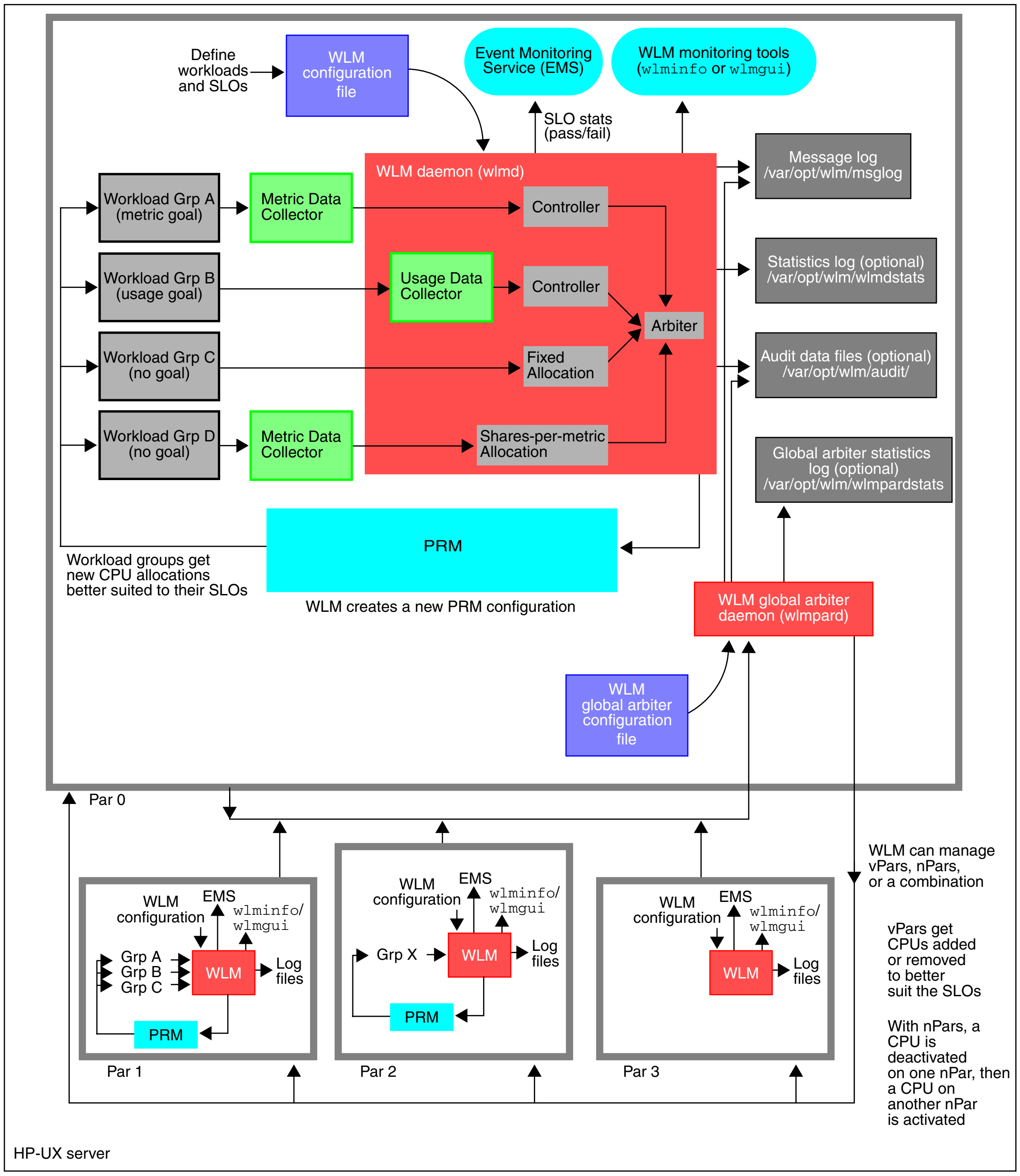
9 Managing SLOs across partitions

WLM can manage SLOs across virtual partitions and nPartitions. You must use Instant Capacity CPUs (formerly known as iCOD CPUs) on the nPartitions for WLM management. WLM provides a global arbiter, `wlmpard`, that can take input from the WLM instances on the individual partitions. The global arbiter then moves CPUs between partitions, if needed, to better achieve the SLOs specified in the WLM configuration files that are active in the partitions. These partitions can be nested—and even contain FSS workload groups. (`wlmpard` can be running in one of the managed partitions or on a supported platform with network connectivity to the managed partitions.)

As noted above, WLM's virtual partition (vPar) management migrates CPUs between virtual partitions. However, with nPartitions being physical entities, WLM's nPartition management only gives the appearance of migrating CPUs. This pseudo-migration is achieved—using the Instant Capacity software— by deactivating a CPU on one nPartition then activating a CPU on another nPartition. WLM maintains the same number of active CPUs, avoiding charges for additional CPUs.

Figure 9-1 shows WLM managing the partitions on a server. For a detailed explanation of the figure, see "How WLM works" on page 102.

Figure 9-1 WLM managing partitions


Define workloads and SLOs
WLM configuration file
Event Monitoring Service (EMS)
WLM monitoring tools (wlminfo or wlmgui)
SLO stats (pass/fail)
WLM daemon (wlmd)
Workload Grp A (metric goal)
Metric Data Collector
Controller
Workload Grp B (usage goal)
Usage Data Collector
Controller
Arbiter
Workload Grp C (no goal)
Fixed Allocation
Workload Grp D (no goal)
Metric Data Collector
Shares-per-metric Allocation
Message log /var/opt/wlm/msglog
Statistics log (optional) /var/opt/wlm/wlmdstats
Audit data files (optional) /var/opt/wlm/audit/
Global arbiter statistics log (optional) /var/opt/wlm/wlmpardstats
PRM
Workload groups get new CPU allocations better suited to their SLOs
WLM creates a new PRM configuration
WLM global arbiter daemon (wlmpard)
WLM global arbiter configuration file
Par 0
WLM configuration
EMS
wlminfo/ wlmgui
Grp A
Grp B
Grp C
WLM
Log files
PRM
Par 1
WLM configuration
EMS
wlminfo/ wlmgui
Grp X
WLM
Log files
PRM
Par 2
WLM configuration
EMS
wlminfo/ wlmgui
WLM
Log files
Par 3
WLM can manage vPars, nPars, or a combination
vPars get CPUs added or removed to better suit the SLOs
With nPars, a CPU is deactivated on one nPar, then a CPU on another nPar is activated
HP-UX server

Setting up cross-partition management

The following steps give an overview of how to implement cross-partition management.

NOTE

Unless you set up secure communications using wlmcert(1M) and use the `wlmpard -s` option, WLM's global arbitration uses non-secured communications. A rogue user could manipulate the communications, resulting in one or more partitions being granted an incorrect number of CPUs. If you are not using the `wlmpard -s` option, use global arbitration only on trusted local area networks.

For information on using global arbitration with firewalls, see the wlmparconf(4) man page.

NOTE

Do not configure WLM to manage PSET-based workload groups and either virtual partitions or nPartitions at the same time.

Do not adjust (using a `vparmodify`, `icap_modify`, or `icod_modify` command) any partition WLM is managing while `wlmpard` is running.

Do not use partition management and the Pay Per Use Toolkit (by placing `utilitydc` in your WLM configuration file) at the same time.

Do not use cell-specific CPU bindings or user-assigned CPU bindings on virtual partitions you are going to manage with WLM.

Partition names cannot contain spaces. Also, if `icod_stat` or `icap_stat` truncate the name of an nPartition, use `parmodify` to shorten the name so that it is not truncated.

Use WLM's vPar management in combination with Instant Capacity only when using vPars version A.03.01 or later. Use WLM's nPartition management with the Instant Capacity versions specified in the WLM Release Notes (/opt/wlm/share/doc/Rel_Notes).

WLM allocates CPUs to a partition based on the CPU limits of the partition (physical limits for nPars; logical limits for virtual partitions). For example, WLM adjusts the number of CPUs assigned to a vPar within the limits of the given vPar's minimum and maximum number of CPUs, which you set using `vparmodify`.

Step 1. (Optional) Set up secure WLM communications

Follow the procedure HOW TO SECURE COMMUNICATIONS in the wlmcert(1M) man page—skipping the step about starting/restarting the WLM daemons. You will do that later in this procedure.

Step 2. Create a WLM configuration file for each partition

Each partition on the system must have the WLM daemon `wlmd` running. Create a WLM configuration file for each partition, ensuring each configuration uses the `primary_host` keyword to reference the partition where the global arbiter is running. For information on the `primary_host` syntax, see "Setting up your WLM configuration file" on page 290.

NOTE If the `cntl_base_previous_req` tunable is set to 0 in a WLM configuration that contains goal-based SLOs, those SLOs may not release CPUs properly when the CPUs are no longer needed. `cntl_base_previous_req` is set to 1 by default. For more information, see "Releasing CPUs properly (optional)" on page 220 or the wlmconf(4) man page.

On systems with WLM installed, you can use the example vPar configuration:

/opt/wlm/examples/wlmconf/vpar_usage_goal.wlm

NOTE Do not edit this configuration file in place. Copy it to a location outside /opt/wlm/. Then edit and activate the configuration from the new location.

This configuration specifies a usage goal for its workload group. The file is included in this book in the section "vpar_usage_goal.wlm" on page 352. (Be sure to use the vpar_usage_goal.wlmpar file for the WLM global arbiter.)

There is also an example nPartition configuration:

/opt/wlm/examples/wlmconf/npar_icod_manual_allocation.wlm

NOTE Do not edit this configuration file in place. Copy it to a location outside /opt/wlm/. Then edit and activate the configuration from the new location.

With this configuration, you manually request the number of CPUs for an nPartition using the `wlmsend` command to feed the request to WLM. The file is in this book in the section "npar_icod_manual_allocation.wlm" on page 326. (Be sure to use the npar_icod_manual_allocation.wlmpar file for the WLM global arbiter.)

Step 3. (Optional) Activate each partition's WLM configuration in passive mode

WLM operates in "passive mode" when you include the `-p` option in your command to activate a configuration. With passive mode, you can see approximately how a particular configuration is going to affect your system—without the configuration actually taking control of your system.

Activate each partition's WLM configuration file *`configfile`* in passive mode as follows:

```
# wlmd -p -a configfile
```

To see approximately how the configuration would affect your system, use the WLM utility `wlminfo`. Fine-tune each partition's configuration until the `wlminfo` output is as desired.

For information on passive mode, including its limitations, see "Passive mode versus actual WLM management" on page 226.

Step 4. Activate each partition’s WLM configuration

After verifying and fine-tuning each partition’s WLM configuration file *configfile*, activate it as follows:

```
# wlmd -a configfile
```

To use secure communications, activate the file using the `-s` option:

```
# wlmd -s -a configfile
```

Step 5. Create a configuration file for the global arbiter

On the system running the global arbiter, create a configuration file for the global arbiter. (If this system is being managed by WLM, it will have both a WLM configuration file and a WLM global arbiter configuration file. You can set up and run the global arbiter configuration on a system that is not managed by WLM if needed for the creation of fault-tolerant environments or Serviceguard environments.)

This global arbiter configuration file is required. In the file specify the:

- Global arbiter interval
- Port used for communications between the partitions
- Size at which to truncate wlmpardstats log file
- Lowest priority at which Temporary Instant Capacity (v6 or v7) or Pay Per Use (v4) resources are used

NOTE Specifying this priority ensures WLM maintains compliance with your license for Temporary Instant Capacity: When the license is exhausted, WLM will no longer attempt to use the temporary resources.

For information on the syntax of this file, see the section “Setting up your WLM global arbiter configuration file” on page 291 or the wlmparconf(4) man page.

On systems with WLM installed, you can use the example configuration /opt/wlm/examples/wlmconf/vpar_usage_goal.wlmpar or /opt/wlm/examples/wlmconf/npar_icod_manual_allocation.wlmpar. These files are included in this book in the chapter “Example configuration files” on page 305.

Step 6. (Optional) Activate the global arbiter configuration file in passive mode

Similar to WLM’s passive mode, the WLM global arbiter has a passive mode (also enabled with the `-p` option) that allows you to verify `wlmpard` settings before letting it control your system.

Activate the global arbiter configuration file *`configfile`* in passive mode as follows:

```
# wlmpard -p -a configfile
```

Again, to see approximately how the configuration would affect your system, use the WLM utility `wlminfo`.

Step 7. Activate the global arbiter

Activate the global arbiter configuration file as follows:

```
# wlmpard -a configfile
```

To use secure communications, activate the arbiter using the `-s` option:

```
# wlmpard -s
```

NOTE In a complex (a system divided into either nPartitions or virtual partitions):

- If Instant Capacity is on the complex, you must use exactly one `wlmpard` process to manage all the partitions in the complex that are under WLM control.
- If Instant Capacity is not on the complex, you must use a separate `wlmpard` process for each nPartition—with each `wlmpard` managing only the virtual partitions within its nPartition.

For information on the `wlmpard` syntax, see the section “WLM command reference” on page 373 or the wlmpard(1M) man page.

Setting up your WLM configuration file

In each partition, you must activate a WLM configuration that indicates the host name for the system running the WLM global arbiter. Indicate the host of the global arbiter as follows, specifying the `primary_host` keyword outside all `prm`, `slo`, and `tune` structures:

```
primary_host = hostname [ : port_number ] ;
```

where

`primary_host`	Is an optional keyword that identifies the host name of the system running the global arbiter.
`hostname`	Is the name or IP address of the host running the arbiter.
`port_number`	(Optional) Is a port number greater than 0 indicating the port that the global arbiter is to monitor. If you specify a port number in a WLM configuration file on a partition, you must specify the same port number in the WLM global arbiter configuration file and in the WLM configuration file in each partition. For more information on how a port is chosen, see the section “Setting up your WLM global arbiter configuration file” on page 291 or the wlmparconf(4) man page.

NOTE Recall that WLM uses absolute CPU units when a configuration includes the `primary_host` keyword. As a result, 100 represents a single CPU, and any CPU shares you specify are out of 100 multiplied by the number of CPUs. For example, 200 represents 2 CPUs. For more information on absolute CPU units, see “Using absolute CPU units” on page 205.

For information on creating the rest of the WLM configuration file, see the chapter “Configuring WLM” on page 127 or the wlmconf(4) man page.

Setting up your WLM global arbiter configuration file

On the system where the global arbiter is going to run, create a global arbiter configuration file.

The global arbiter configuration file allows you to control settings for HP-UX WLM's global arbiter, which governs cross-partition management and management of Temporary Instant Capacity and Pay Per Use resources.

WLM can manage CPU resources for virtual partitions and nPartitions. (The partitions can even be nested and contain FSS workload groups.) WLM's virtual partition management migrates CPUs between virtual partitions. With nPartitions being physical entities, WLM's nPartition management—through the use of Instant Capacity software—only gives the appearance of migrating CPUs. This pseudo-migration is achieved by deactivating a CPU on one nPartition and then activating a CPU on another nPartition. WLM maintains the same number of active CPUs, avoiding charges for additional CPUs.

NOTE The configuration file is required.

The file's syntax is free-form, allowing white space to be used freely in formatting the file. Comment lines are preceded with the hash mark (#). Use `wlmpard` to activate a global arbiter configuration file.

The WLM global arbiter configuration file consists of a `par` structure that allows you to specify:

- How frequently the WLM global arbiter checks for input from the individual WLM instances running in the partitions
- The port that the WLM global arbiter should monitor for input from the individual WLM instances
- The size, in megabytes, at which the wlmpardstats file is automatically truncated
- The lowest priority at which WLM should use Temporary Instant Capacity or Pay Per Use resources to manage the total CPU capacity available

NOTE If you specify a port number in the `primary_host` statement of any of the WLM configuration files or in the WLM global arbiter configuration file, you must specify the same port number in all the files.

The `par` structure is shown below, followed by an example structure and then an explanation of the syntax.

NOTE If your configuration uses a previously supported structure type, such as `vpar` or `npar_icod`, WLM silently interprets these as a `par` structure.

par structure

The structure takes the following format:

```
par {
    [ interval = number_of_seconds; ]
    [ port = port_number; ]
    [ wlmpardstats_size_limit = number_of_megabytes; ]
    [ utilitypri = integer; ]

}
```

Here is an example `par` structure:

```
par {
    interval = 180;
    port = 3701;
    wlmpardstats_size_limit = 3;
}
```

The keywords are discussed in the sections that follow.

Specifying the global arbiter interval (optional)

Specify how often the WLM global arbiter is to check for input from the WLM instances—and potentially alter the number of CPUs for the partitions—using the keyword `interval`. The syntax is:

`interval =` *`number_of_seconds`*`;`

where

`interval`

Is a optional keyword.

`number_of_seconds`

Is an integer value greater than or equal to 1 indicating how often, in seconds, WLM should consider moving CPUs between partitions.

NOTE The interval for your global arbiter should be larger than the largest WLM interval being used on the system.

When generating audit data (with the `wlmpard -t` option), *`number_of_seconds`* should be no more than 120 (twice the suggested interval for the WLM instances when auditing). This value ensures the audit data sampling rate is sufficiently frequent to:

- Minimize the amount of audit data lost if the WLM global arbiter daemon exits unexpectedly
- Minimize the effect of changes in the number of available CPUs (due to WLM's management of partition resources)

Specifying the port (optional)

The `port` keyword is an optional keyword specifying which port the WLM global arbiter should monitor for input from the various WLM instances. Indicate a port using the following syntax:

`port =` *`port_number`*`;`

where

`port`	Is an optional keyword.
`port_number`	Is a port number greater than 0 that the WLM global arbiter should monitor to collect input from the WLM instances running on the partitions. If you specify a port number in the WLM global arbiter configuration file, you must specify the same port number in each partition's WLM configuration file.

If you do not specify a port, `wlmpard` searches the file /etc/services for the first line with the following format:

`hp-wlmpar` *`port_number`*`/tcp`

If such an entry is found, *`port_number`* is used as the port. If such an entry is not found, the default port of 9691 is used, assuming it is not in /etc/services with protocol tcp. If it is, an error is issued.

Trimming the global arbiter statistics log automatically (optional)

You can automatically trim the global arbiter statistics log file /var/opt/wlm/wlmpardstats. This file is created when you use the `-l` option to `wlmpard`. Enable automatic trimming of the file by using the `wlmpardstats_size_limit` keyword. The syntax is:

```
wlmpardstats_size_limit = number_of_megabytes;
```

where

`wlmpardstats_size_limit`

Is an optional keyword.

`number_of_megabytes`

Is an integer value from 0 to 2048. Once a file's size is greater than *`number_of_megabytes`*, the file is trimmed. The default value is 0, which disables automatic trimming. The wlmpardstats file is trimmed by renaming it wlmpardstats.old. Then a new wlmpardstats file is started. The wlmpardstats.old file includes a line at the end of the file indicating that the file has been automatically trimmed.

Specifying the priority at which to use Temporary Instant Capacity or Pay Per Use resources (optional)

NOTE While `wlmpard` has always managed CPU migration across partitions for WLM, it also now provides management of Temporary Instant Capacity and Pay Per Use resources for WLM. This management is available on standalone systems, as well as across a collection of partitions. With the `utilitypri` keyword mentioned below, `wlmpard` optimizes the Temporary Instant Capacity or Pay Per Use resources. It determines the amount of resources needed to meet your workloads' SLOs, then keeps the total number of active CPUs on the system to the minimum needed to meet that resource demand. By minimizing the number of active CPUs, WLM reduces your costs.

If you have Temporary Instant Capacity (v6 or v7) or Pay Per Use (v4) resources available on a system, WLM can help you optimize the use of those resources—using them only as needed to meet the service-level objectives for your workloads.

To use `wlmpard` to optimize the use of these resources, follow the steps given above in the section "Setting up cross-partition management" on page 285—with one exception: Specify the `utilitypri` keyword in the global arbiter configuration file. This keyword works with Temporary Instant Capacity v6 and v7 and with PPU v4.

NOTE The `utilitypri` keyword allows WLM—when Temporary Instant Capacity is available—to adjust the CPU total to meet demand. Also, this keyword ensures WLM maintains compliance with your license for Temporary Instant Capacity: When the license is exhausted, WLM will no longer attempt to use the temporary resources.

This keyword has the syntax:

```
utilitypri = integer;
```

where

`utilitypri`

Is an optional keyword.

`integer`

Is an integer value greater than or equal to 1. If Temporary Instant Capacity or PPU resources exist, they are used whenever SLOs with a priority from 1 to *`integer`* (inclusive) are demanding more CPU.

10 Management of nested nPars / vPars / FSS groups

You can manage any combination of FSS workload groups inside virtual partitions inside nPartitions if desired.

NOTE You cannot use PSET workload groups with partition management.

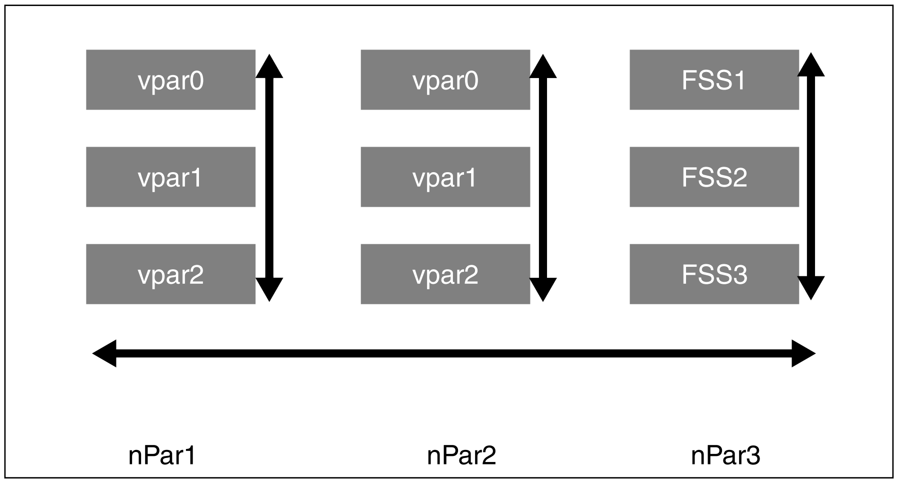
Setting up the various configurations

The discussions below only indicate where configuration files are needed. For information on the WLM configuration file, see “Configuring WLM” on page 127. For detailed information on setting up your `wlmpard` configuration, see “Managing SLOs across partitions” on page 283.

Managing FSS groups inside vPars inside nPars (Instant Capacity available in the complex)

Consider a complex, such as the one shown in Figure 10-1, consisting of three nPartitions, with two of those nPartitions each having three virtual partitions, and the third partition divided into FSS workload groups.

Figure 10-1 Nested partitions to be managed (with Instant Capacity)



Assume you want to share CPU resources, as indicated by the arrows in the figure, among the:

- Virtual Partitions within nPar1
- Virtual partitions within nPar2
- FSS workload groups within nPar3
- Three nPartitions

Instant Capacity (formerly known as iCOD) is available on the complex so WLM can migrate licenses between the nPartitions to simulate CPU movement. With Instant Capacity present, you must have only one `wlmpard` managing the partitions. In particular:

Step 1. Set up a `wlmpard` configuration on some system, say *`mysystem`*

`mysystem` can be in the complex or on another system on your network.

In this file, say *`myfile`*, specify the `utilitypri` keyword to indicate the range of SLO priorities that can cause WLM to use Temporary Instant Capacity resources.

If you are not using Temporary Instant Capacity resources, you can omit the `utilitypri` keyword.

NOTE Specifying this keyword ensures WLM maintains compliance with your license for Temporary Instant Capacity: When the license is exhausted, WLM will no longer attempt to use the temporary resources.

Step 2. Start `wlmpard` on *`mysystem`*

Use the file you created in the previous step with `wlmpard`:

```
# wlmpard -a myfile
```

Step 3. Set up and start a configuration for `wlmd` in each virtual partition

Each configuration must include a `primary_host` statement of the form:

```
primary_host = mysystem;
```

This configuration can be just for the virtual partition itself, or you can add a `prm` structure and create FSS workload groups inside the virtual partition.

Step 4. Set up and start a configuration for `wlmd` in nPar3 to manage the FSS workload groups

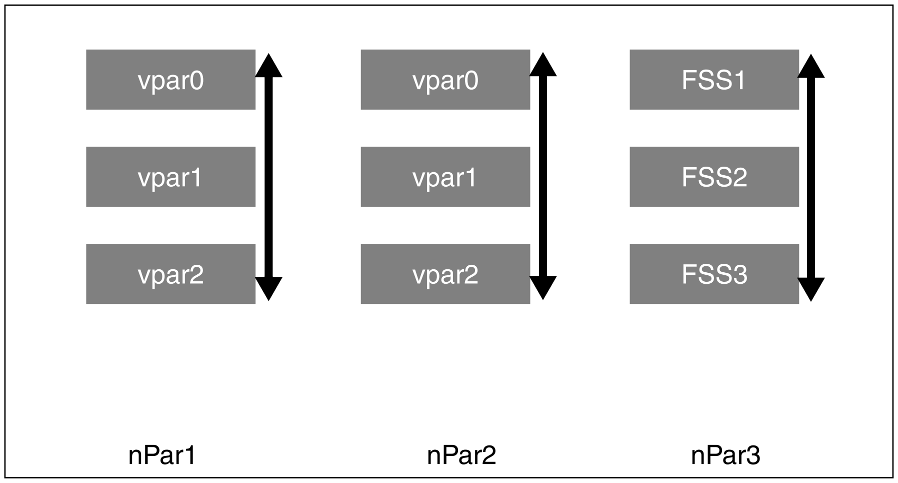
This configuration must include a `primary_host` statement of the form:

```
primary_host = mysystem;
```

Managing FSS groups inside vPars inside nPars (Instant Capacity not available)

Consider the same complex as before, but without Instant Capacity. CPU movement between the nPartitions cannot be simulated. Figure 10-2 shows this complex.

Figure 10-2 **Nested partitions to be managed (without Instant Capacity)**



Again, assume you want to share CPU resources, as indicated by the arrows in the figure, among the:

- Virtual Partitions within nPar1
- Virtual partitions within nPar2
- FSS workload groups within nPar3

With Instant Capacity not available, you must have one `wlmpard` for each nPartition containing virtual partitions. In particular:

Step 1. Set up nPar1

a. Set up and start a configuration for `wlmpard` on some system, say *`mysystem1`*

`mysystem1` can be in the complex or another system on your network.

b. Set up and start a configuration for `wlmd` in each virtual partition in the nPartition

Each configuration must include a `primary_host` statement of the form:

```
primary_host = mysystem1;
```

Step 2. Set up nPar2

a. Set up and start a configuration for `wlmpard` on some system, say *`mysystem2`*, and start `wlmpard`

`mysystem2` can be in the complex or another system on your network.

b. Set up and start a configuration for `wlmd` in each virtual partition in the nPartition

Each configuration must include a `primary_host` statement of the form:

```
primary_host = mysystem2;
```

Step 3. Set up and start a configuration for `wlmd` in nPar3 to manage the FSS workload groups

Managing FSS groups inside vPars

If your system is not broken into nPartitions, you do not need to set up your configurations based on whether the complex has Instant Capacity. You just need to:

Step 1. Run one `wlmpard` to manage the system's virtual partitions

Each virtual partition will have a configuration for `wlmd` with a `primary_host` statement naming the system where `wlmpard` is running.

Step 2. Create your FSS workload groups in each virtual partition's WLM configuration as desired

11 Example configuration files

This chapter presents the example configuration files available from the directory /opt/wlm/examples/wlmconf/, as well as an example from /opt/wlm/toolkits/weblogic/config/. These examples show how to use the syntax discussed in “Configuring WLM” on page 127.

NOTE Copy these examples to another directory before modifying them. Items kept under /opt/wlm/ may be replaced or altered by future WLM product updates.

The following list provides an overview of the available example files. The files are listed alphabetically, not by complexity.

- “distribute_excess.wlm” on page 308

  Example configuration file demonstrating the use of the `weight` and `distribute_excess` tunables. This functionality is used to manage the distribution of resources among workload groups after honoring performance goals specified in `slo` structures.

- “enabling_event.wlm” on page 311

  A configuration file demonstrating the use of WLM to enable or disable an SLO when a certain event occurs.

- “entitlement_per_process.wlm” on page 313

  A configuration file that demonstrates the use of a shares-per-metric entitlement request. A workload group’s entitlement is based directly on the number of currently active processes running in the group.

- “fixed_entitlement.wlm” on page 316

  This example configuration illustrates the use of WLM in allocating a fixed entitlement to a particular group of users.

- "manual_cpucount.wlm" on page 318

  A configuration file to help a WLM user characterize the behavior of a workload. The goal is to determine how a workload, placed in a PSET-based workload group, responds to a series of changes in the number of CPUs in the PSET. This example is in the directory /opt/wlm/toolkits/weblogic/config/.

- "manual_entitlement.wlm" on page 320

  A configuration file to help a WLM user characterize the behavior of a workload. The goal is to determine how a workload responds to a series of entitlements.

- "metric_condition.wlm" on page 323

  Configuration file to illustrate that an SLO can be enabled based upon the value of a metric (in this case, the metric is provided by a glance data collector provided with the WLM product). Metrics can be used in both the `goal` statement and the `condition` statement of a single SLO.

- "npar_icod_manual_allocation.wlm" on page 326, "npar_icod_manual_allocation.wlmpar" on page 329

  These configuration files demonstrate WLM's ability to resize nPartitions, or nPars. The resizing is accomplished by deactivating CPUs on some nPartitions then activating CPUs on other nPartitions on the system. The number of active CPUs remains the same so no additional charge is incurred. Configure WLM in each nPartition on the system using the .wlm file. Configure the WLM global arbiter in one nPartition using the .wlmpar file.

- "performance_goal.template" on page 331

  This file has a different filename extension (.template vs. .wlm). That is simply because this file distinguishes between configuration file special keywords and user-modifiable values by placing the items that a user would need to customize within square brackets ([,]). Because of the presence of the square brackets, the sample file will not pass the syntax-checking mode of `wlmd` (`wlmd -c` *`template`*).

- "stretch_goal.wlm" on page 334

  Example configuration file to demonstrate how to use multiple SLOs for the same workload (but at different priority levels) to specify a stretch goal for a workload. A stretch goal is one that we would like to have met if all other higher-priority SLOs are being satisfied and there are additional CPU cycles available.

- "time_activated.wlm" on page 337

  This configuration file demonstrates the use of WLM in allocating a fixed entitlement to a particular group of users only during a certain time period.

- "transient_groups.wlm" on page 339

  This configuration file demonstrates transient groups. The groups in this configuration consume no resources when they have no active SLOs.

- "twice_weekly_boost.wlm" on page 342

  A configuration file that demonstrates a conditional entitlement with a moderately complex condition.

- "usage_goal.wlm" on page 347

  This configuration demonstrates the usage goal for service-level objectives. This type of goal is very different from the typical performance goal in that it does not require explicit metric data.

- "user_application_records.wlm" on page 349

  A configuration file that demonstrates the use of, and precedence between, user and application records in placing processes in workload groups.

- "vpar_usage_goal.wlm" on page 352,
  "vpar_usage_goal.wlmpar" on page 355

  These configuration files demonstrate WLM's ability to resize HP-UX Virtual Partitions, shifting CPUs between the partitions on a system. Configure WLM in each vPar on the system using the .wlm file. Configure the WLM global arbiter in one vPar using the .wlmpar file.

NOTE The section "How do I get started with ODBTK?" on page 417 describes example configuration files you can use with Oracle instances. For pointers to the example configurations for the various WLM toolkits, see the EXAMPLES section of the wlmtk(5) man page.

distribute_excess.wlm

This example features the `distribute_excess` and `weight` keywords.

```
#
# Name:
#       distribute_excess.wlm
#
# Version information:
#
#       (C) Copyright 2001-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.8 $
#
# Caveats:
#       DO NOT MODIFY this script in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       Demonstrate the use of the weight and distribute_excess keywords
#       to distribute resources among groups.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.01.01
#       or later.
#
```

```
# prm structure
#
#       In this example we have two groups using the machine, Orders
#       and Sales.  In the event there are additional resources available,
#       we'd like for Sales to get three shares for every share that Orders
#       receives.
#
#       See wlmconf(4) for complete HP-UX WLM configuration information.
#
prm {
        groups= OTHERS : 1,
                Orders : 2,
                Sales : 3;
        weight= Orders : 1,
                Sales : 3;
}

#
# slo structure
#
#       In this example we have two SLOs, one for actual order processing
#       and another for sales' product availability requests.  These are
#       deemed to be equal in priority and have goals based on the
#       response times of their applications.
#
#       In the event that both goals are met, we want to distribute the
#       excess CPU resources according to the weight specification above.
#       See the tune structure explanation for details on what the impact
#       might be in this situation.
#
slo order_processing {
        pri = 1;
        mincpu = 0;
        maxcpu = 100;
        entity = PRM group Orders;
        goal = metric order_transaction_time < 10.0;
}

slo product_availability {
        pri = 1;
        mincpu = 0;
        maxcpu = 100;
        entity = PRM group Sales;
        goal = metric sales_request_time < 5.0;
}
```

```
# tune structure
#
#       We must define a data collector for every metric used in
#       SLO definitions.  In this example, we add a global tune structure
#       where we enable distribution of excess shares to groups defined in
#       the prm structure above.  This is accomplished by simply setting
#       the keyword distribute_excess to 1 (TRUE).  Without this setting,
#       HP-UX WLM would allocate all excess shares to the OTHERS group.
#
#       HP-UX WLM will, by default, provide an allocation sufficient to meet
#       the specified goals.  Let us assume that both goals are being met and
#       that the Orders group is consuming 20% of the CPU resources and
#       the Sales group is also consuming 20%.  Then 60% of the machine is
#       available for distribution to the two groups.  If distribute_excess
#       is enabled, WLM will ultimately allocate about 25% of the CPU
#       resources to the Orders group and about 75% of the CPU resources to
#       the Sales group, reflecting the 3:1 weight ratio specified for the
#       groups in the prm structure. (The OTHERS group must get a minimum
#       of 1% of the CPU. This 1% will be taken from either the Sales group
#       or Orders group. As a result, WLM will not be able to grant CPU
#       exactly in the 3:1 weight ratio.)
#
#       Enabling distribute_excess is likely to reduce the response
#       times for both groups well below the goals specified (for example,
#       the sales_request_time may drop down to 3 seconds).  Without the
#       distribute_excess keyword set, HP-UX WLM would keep the response
#       times close to the goals (so, sales_request_time would be kept
#       around 4.5 seconds).
#
#       With distribute_excess enabled, HP-UX WLM allocates excess shares so
#       that the resulting allocations are as close as possible to the
#       weight distribution without decreasing allocations resulting from
#       SLOs.  Thus, HP-UX WLM will distribute CPU resources sufficient to
#       meet the SLOs and only after that action will HP-UX WLM begin
#       distributing excess shares to achieve the weight distribution.
#
#       Consider yet another scenario where SLO goals are being met.
#       However, the Orders group is using 50% of the resources to meet its
#       response time goal, and the Sales group is using only 20%.  HP-UX WLM
#       tries to match the allocations according to the weight
#       specifications, but it will not reduce an allocation if doing so
#       will cause the SLOs to fail (the goals to not be achieved).
#       HP-UX WLM will give the Sales group the remaining 30% of the CPU
#       resources so that the resulting allocation is 50% for each group.
#       This does not meet the 3:1 ratio (that is, 75%/25%), but doing so
#       would cause the order_processing SLO to fail. (As OTHERS must get
#       at least 1% of the CPU, the allocation for either the Sales group
#       or Orders group would be reduced by 1%.)
#
tune {
        distribute_excess = 1;
}

tune sales_request_time {
        coll_argv = /home/sales/data_collector;
}

tune order_transaction_time {
        coll_argv = /home/orders/data_collector;
}
```

For more information on the `distribute_excess` and `weight` keywords, see “Distributing excess CPU to your workloads (optional)” on page 206 and “Weighting a group so it gets more CPU (optional)” on page 162.

enabling_event.wlm

This example enables an SLO based on an event.

```
#
# Name:
#       enabling_event.wlm
#
# Version information:
#
#       (C) Copyright 2001-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.8 $
#
# Caveats:
#       DO NOT MODIFY this script in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
# Purpose:
#       Demonstrate the use of HP-UX WLM to enable a service-level objective
#       (SLO) when a certain event occurs.
#
# Dependencies:
#       This example was designed to run with version HP-UX WLM A.01.02
#       or later.
#

#
# prm structure
#
#       See wlmconf(4) for complete HP-UX WLM configuration information.
#
#       Define workload groups in the prm structure.  Individual users are
#       assigned to groups in this structure as well.
#
#       In this example configuration, the user backup_meister can execute
#       applications in only the Backups group.
#

prm {
        groups = Backups : 3;
        users = backup_meister: Backups;
}

#
# slo structure
#
#       The following slo structure is enabled and disabled based on the
#       value of the metric backup_running.  If backup_running's value is
#       zero then the SLO is disabled, and any application running in the
#       group Backups will get only 1% of the CPU resources.  When the
#       value is nonzero (for example, 1) the SLO is enabled and the groups
#       within the Backups group get a fixed allocation of 60% of the CPU
#       resources on the system.
#
slo backup_processing {
        pri = 1;
        cpushares = 60 total;
        entity = PRM group Backups;
        condition = metric backup_running;
}
```

```
# tune structure
#
#       Use of a metric in a SLO structure, whether it is a goal or
#       condition, requires that the metric have a tune structure associated
#       with it.  The structure must specify the data collector for the
#       metric.  In this example, the collector specified with the coll_argv
#       keyword is wlmrcvdc.  This allows the metric backup_running to be set
#       with wlmsend from a command line.  For example,
#
#                % wlmsend backup_running 1
#
#       would enable the SLO above.
#
tune backup_running {
        coll_argv = wlmrcvdc;
}
```

For information on the `condition` keyword, see “Specifying when the SLO is active (optional)” on page 192.

entitlement_per_process.wlm

The following example shows how to request 5 CPU shares for each active process in a workload group.

```
#
# Name:
#       entitlement_per_process.wlm
#
# Version information:
#
#       (C) Copyright 2001-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.11 $
#
# Caveats:
#       DO NOT MODIFY this file in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
```

```
# Purpose:
#       Demonstrate the use of a shares-per-metric goal.
#
#       A machine hosts a group’s web servers.  However, the group would
#       like to use the rest of the CPU cycles without impacting the web
#       servers, which are often not busy.  From past experience, they
#       have determined an httpd base entitlement (allocation) of 10
#       percent of the CPU gives good response for random queries, and that
#       sustained web processing can be done by providing an additional 5
#       shares per active (not sleeping) process.  The httpd processes and
#       their children (CGI scripts, servlets, etc) that were recently
#       active are included in the count.  See the glance(1) documentation
#       for more information on available metrics.
#
# Components:
#       The glance toolkit is used.  See glance_app(1M) for more
#       information on the glance data collectors.
#
# Allocation change frequency:
#       Because wlm_interval is not explicitly set, the 60 second default
#       is used. This means wlmd will collect measurements and make
#       allocation changes every 60 seconds.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.01.02 or
#       later. It uses the cpushares keyword introduced in A.01.02, so
#       is incompatible with earlier versions of HP-UX WLM.
#




#
# prm structure
#       Create workload groups.  Designate which workload binaries will
#       be placed in each.  We will be actively managing one workload,
#       apache, and leaving the rest of the CPU resources to workload
#       group OTHERS.
#
#       See wlmconf(4) for complete HP-UX WLM configuration information.
#
prm {
        groups = OTHERS : 1,
                 servers : 2 ;

        apps = servers : /opt/hpws/apache/bin/httpd ;
}
```

```
# Grant 5 shares to servers for every active httpd process.
# Never allow the allocation to fall below 10 shares, nor to
# rise above 90% of the CPU resources.
#
slo servers_proportional {
        pri = 1;
        mincpu = 10;
        maxcpu = 90;
        entity = PRM group servers;
        cpushares = 5 more per metric apache.active_procs ;
}

# Any CPU that remains after satisfying the above SLO is given to the
# OTHERS group by default. You can change this default using the
# distribute_excess keyword. For more information on this keyword, see
# the wlmconf(4) man page.

#
# Collect the glance metric APP_ACTIVE_PROC as a measure
# of how many httpd processes are actually busy.
#
# NOTE: If the number of active processes is too volatile, consider using the
# smooth(1M) tool to calculate a running average, smoothing out the extreme
# values. smooth is shown below, commented out.
#
tune apache.active_procs {
    coll_argv =
        wlmrcvdc
#           smooth
                glance_prm APP_ACTIVE_PROC servers ;
}
```

For information on the `cpushares` keyword, see “Specifying a shares-per-metric allocation request (optional)” on page 188.

fixed_entitlement.wlm

The example below shows a fixed entitlement for a workload group.

```
#
# Name:
#       fixed_entitlement.wlm
#
# Version information:
#
#       (C) Copyright 2001-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.6 $
#
# Caveats:
#       DO NOT MODIFY this script in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       Demonstrate the use of HP-UX WLM in allocating a fixed entitlement
#       (allocation) to a particular group of users.
#
# Dependencies:
#       This example was designed to run with version HP-UX WLM A.01.02
#       or later.
#


#
# prm structure
#
#       See wlmconf(4) for complete HP-UX WLM configuration information.
#
#       All workload groups must be defined in the prm structure.  Individual
#       users are assigned to groups in this structure as well.
#
#       In this example, the user bob will have all of his applications
#       run in the Marketing workload group.  The user sally can execute
#       applications in either Marketing or OTHERS.  Marketing is listed
#       first for sally, so it is the group that all of her applications
#       will execute in by default.  Because OTHERS is also listed, it is an
#       alternate group for sally.  She can move existing processes to, or
#       start new processes in, alternate groups using the prmmove or prmrun
#       commands.  For more information, see prmmove(1) and prmrun(1).
#
#       Note that the group OTHERS is a group created automatically.
#       Applications run by users not referenced in the prm structure will
#       execute in the OTHERS group.  So, only bob and sally can execute
#       applications in the Marketing group.
#

prm {
        groups = Marketing : 2;
        users = bob : Marketing,
                sally: Marketing OTHERS;
}

#
# slo structure
#
#       The keywords pri (priority) and entity (group to which
#       the SLO applies) are required for any SLO specification.
#
#       The cpushares statement requests a fixed allocation of 30% of the
#       CPU resources on the system for the Marketing group.
#
slo Marketing_Analysis {
        pri = 1;
        cpushares = 30 total;
        entity = PRM group Marketing;
}
```

For information on the `cpushares` keyword, see “Specifying a shares-per-metric allocation request (optional)” on page 188.

manual_cpucount.wlm

This example is similar to the previous one: It allows you to easily step through a number of different CPU allocations for a workload group. The difference is that this configuration uses a PSET-based workload group. When you change CPU allocations, you are changing the number of CPUs assigned to the group's PSET. The file is located at /opt/wlm/toolkits/weblogic/config/manual_cpucount.wlm.

```
#
# Name:
#       manual_cpucount.wlm
#
# Version information:
#
#       (c) Copyright 2002, Hewlett-Packard Company, all rights reserved.
#
#       $Revision: 1.8 $
#
# Caveats:
#       DO NOT MODIFY this script in its /opt/wlm/toolkits location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       This example is similar to the
#              /opt/wlm/examples/wlmconf/manual_entitlement.wlm
#       example. It lets a user or system administrator easily set the
#       CPU count for a PRM PSET group using the wlmsend command.
#
#       This is useful for benchmarking or capacity-planning situations--
#       a benchmark can be run against an instance, then the PSET
#       CPU count changed. After the count is changed, the benchmark
#       can be measured again, all without restarting WLM or the instance
#       or manually reconfiguring the PSET.
#
# Components:
#       This example does use a data collector, but it has no other
#       required configuration file.
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.02.01 or
#       later. It uses the CPU movement among PSET groups feature
#       introduced in WLM A.02.01. Consequently, it is incompatible with
#       earlier versions of HP-UX WLM.
#

#
# prm structure
#       There is a single WebLogic instance, instA, being controlled
#       in a WLM workload group, wls1_grp.
#
#       The users statement allows the user named 'wladmin' to run jobs in
#       OTHERS or wls1_grp, with group wls1_grp being his default group.
#

prm {
    groups =
        OTHERS : 1,
        wls1_grp : PSET
    ;

    # Now use a user record to place all processes owned by 'wladmin'
    # in the target group. This assumes that 'wladmin' runs the instance
    # but no other resource-intensive jobs. If the user 'wladmin' runs
    # other UNIX processes, they will be placed in the wls1_grp too, and
    # compete for its resources.
    users =
        wladmin: wls1_grp OTHERS
    ;
}


#
# Create an SLO that allocates CPU to group wls1_grp based on the metric
# wls1_grp.desired.cpucount. If wls1_grp.desired.cpucount is 1.0, the
# allocation is 100, which is one CPU. 2 is 200, or two CPUs, and so on.
#
slo wls_base {
        pri = 1;
        entity = PRM group wls1_grp;
        cpushares = 100 total per metric wls1_grp.desired.cpucount;
}
```

```
# Tell wlmrcvdc to watch for metrics coming in via command lines:
#        % /opt/wlm/bin/wlmsend wls1_grp.desired.cpucount 1
# or
#        % /opt/wlm/bin/wlmsend wls1_grp.desired.cpucount 2
#
tune wls1_grp.desired.cpucount {
        coll_argv = wlmrcvdc ;
}

#
# Check for new metrics every 5 seconds.
# Also, turn on absolute CPU units, so resources on a 4-CPU box are
# represented as 400 shares instead of 100 shares.
#
tune {
    wlm_interval = 5;
    absolute_cpu_units = 1;
}
```

For information on the `cpushares` keyword, see “Specifying a shares-per-metric allocation request (optional)” on page 188. For information on using `wlmrcvdc` and `wlmsend` in this manner, see “Sending data from the command line” on page 252.

manual_entitlement.wlm

This next example is helpful in determining what type of performance to expect from a workload at various amounts of CPU. With that information, you can create better service-level objectives for the workload. As such, this example is a good model to follow when setting up your WLM configuration.

```
#
# Name:
#       manual_entitlement.wlm
#
# Version information:
#
#       (C) Copyright 2001-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.13 $
# Caveats:
#       DO NOT MODIFY this file in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       This example demonstrates the use of a parametric entitlement
#       (allocation) to characterize the behavior of a workload.  The user
#       can directly set an allocation for a workload with wlmsend.  By
#       gradually increasing the workload's allocation with a series of
#       wlmsend commands, the operator can determine how the workload
#       behaves with various amounts of CPU.  This research can then be
#       compared with similar data for the other workloads that will run on
#       the system.  This comparison gives you insight into which workloads
#       you can combine (based on their needed CPU) on a single system
#       and still achieve the desired SLOs.  Alternatively, if you
#       cannot give a workload its optimal amount of CPU, you will know
#       what kind of performance to expect with a smaller allocation.
#
#       The user's workload is placed in grp1 and is associated with
#       a metric called myapp.desired.allocation. The metric value
#       represents an absolute allocation request as defined by the
#       controlling SLO's cpushares statement (see the definition for
#       grp1 below).
#
#       The workload is tested by sending a series of allocation requests
#       like the following:
#
#       % wlmsend myapp.desired.allocation 10
#       % <manually measure workload performance under new allocation>
#       % wlmsend myapp.desired.allocation 20
#       % <manually measure workload performance under new allocation>
#       % wlmsend myapp.desired.allocation 30
#       % <manually measure workload performance under new allocation>
#       % wlmsend myapp.desired.allocation 40
#       % <manually measure workload performance under new allocation>
#       % wlmsend myapp.desired.allocation 50
#       % <manually measure workload performance under new allocation>
#       % wlmsend myapp.desired.allocation 60
#       % <manually measure workload performance under new allocation>
#       etc.
#
#       After each allocation request is sent, the user must manually
#       measure the workload performance to determine its response to
#       the specific allocation.
# Components:
#       Uses the wlmsend and wlmrcvdc tools to relay a metric from
#       an outside user.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.01.02 or
#       later. It uses the cpushares keyword introduced in A.01.02, so
#       is incompatible with earlier versions of HP-UX WLM.
#
#



#
# prm structure
#       Create a workload group called grp1 and define the binaries that
#       make it up. In this example, the apache webserver binary
#       /opt/hpws/apache/bin/httpd and the apache benchmark utility
#       /opt/hpws/apache/bin/ab are used as the workloads. This
#       also illustrates how to place two different binaries into
#       the same workload group. See the example configurations in
#       /opt/wlm/toolkits/apache/config/ for more WLM configurations
#       to place and control apache.
#
#       See wlmconf(4) for complete HP-UX WLM configuration information.
#
prm {
        groups = OTHERS : 1, grp1 : 2;
        apps =  grp1 : /opt/hpws/apache/bin/httpd,
                grp1 : /opt/hpws/apache/bin/ab;
}

#
# Have HP-UX WLM manage the CPU allocation such that the workload group
# grp1 gets 1 share per metric.  That is, setting the metric
# myapp.desired.allocation to 20 results in 20 shares for grp1,
# setting the metric myapp.desired.allocation to 50 results in 50
# shares for grp1, and so forth.
#
slo grp1 {
        pri = 1;
        entity = PRM group grp1;
        cpushares = 1 total per metric myapp.desired.allocation;
}
```

```
# Any CPU that remains after satisfying the above SLO is given to the
# OTHERS group by default. You can change this default using the
# distribute_excess keyword. For more information on this keyword, see
# the wlmconf(4) man page.

#
# Relay a metric from the outside user.
#
tune myapp.desired.allocation {
    coll_argv = wlmrcvdc;
}
```

For information on the `cpushares` keyword, see “Specifying a shares-per-metric allocation request (optional)” on page 188. For information on using `wlmrcvdc` and `wlmsend` in this manner, see “Sending data from the command line” on page 252.

metric_condition.wlm

This example shows a metric enabling an SLO only when the associated workload has work to do.

```
#
# Name:
#       metric_condition.wlm
#
# Version information:
#
#       (C) Copyright 2001-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.7 $
#
# Caveats:
#       DO NOT MODIFY this file in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
# Purpose:
#       Demonstrate the use of HP-UX WLM to enable a service-level objective
#       (SLO) when a measurable metric value is reached.  Also make use of a
#       glance data collector to provide a metric value.
#
# Components:
#       The glance toolkit (included with HP-UX WLM) is used. See
#       glance_prm(1M) for more information on the glance data collectors.
#
# Dependencies:
#       This example was designed to run with version HP-UX WLM A.01.01
#       or later.
#
# See wlmconf(4) for complete HP-UX WLM configuration information.
#

#
# prm structure
#       Define all workload groups in the prm structure.
#
prm {
    groups = OTHERS : 1,
             jobs : 2;
    apps =   jobs: /opt/batch/bin/submit;
}

# slo structures
#
#       This SLO has a metric goal.  It also has a metric used in the
#       condition statement.  Both these metrics will require that more
#       information be provided in metric-specific tune structures.
#
#       This SLO becomes active based on the number of active jobs, once
#       the server actually has jobs submitted to it. It attempts to keep
#       the average job-completion time under 10 minutes (these are batch
#       jobs, so we're not dealing in seconds of response time) when there
#       are any active jobs.  If there are no jobs, this SLO is inactive.
#
slo job_processing {
   pri = 1;                         # SLO of highest priority
   mincpu = 40;                     # minimum CPU allocation (percentage)
   maxcpu = 90;                     # maximum CPU allocation (percentage)
   entity = PRM group jobs;
   goal = metric avg_completion_time < 10;
   condition = metric number_of_active_jobs > 0;
}
```

```
# tune structures
#
#       These structures provide the data collector information
#       for the metrics used in the slo structure above.
#
#       One data collector is a hypothetical application, written to
#       calculate and provide average job-completion time in minutes.
#
#       The other metric is calculated using the glance toolkit and
#       a glance metric called APP_ALIVE_PROC.
#

tune avg_completion_time {
   coll_argv = /opt/batch/metrics/job_complete_time -minutes;
}

tune number_of_active_jobs {
   coll_argv = wlmrcvdc glance_prm APP_ALIVE_PROC jobs;
}
```

For information on the `condition` keyword, see “Specifying when the SLO is active (optional)” on page 192.

npar_icod_manual_allocation.wlm

This file, in combination with the global arbiter configuration file in the next section, migrates CPUs between HP nPartitions (nPars) based on the number of CPUs you request on the command line using `wlmsend`.

Activate the WLM configuration file in each nPar. However, you only need to activate the WLM global arbiter's configuration in one system on your network.

```
#
# Name:
#       npar_icod_manual_allocation.wlm
#
# Version information:
#
#       (C) Copyright 2003-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.8 $
#
# Caveats:
#       DO NOT MODIFY this file in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       This example, used with npar_icod_manual_allocation.wlmpar,
#       demonstrates WLM's ability to resize nPartitions using iCOD
#       (Instant Capacity on Demand) software. The applications within
#       an nPartition are considered a single workload. WLM shifts CPUs
#       between the nPartitions based on the requests and priorities of
#       these workloads. (The shifting is actually just the deactivating of
#       a CPU on one nPartition and activating of a CPU on another
#       nPartition. The total number of active CPUs remains
#       constant--avoiding a charge for additional CPUs.)
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.03.00 or
#       later. (A.03.00 was the first version to support strictly
#       host-based configurations.)
#
```

```
# To implement WLM's dynamic nPartition resizing:
#       1. Set the primary_host keyword in this file
#
#       2. Copy this WLM configuration to each nPartition in the system
#
#       3. Adjust the priority for the SLO named slo_myslo to reflect the
#          priority of the applications for the current nPartition relative
#          to the priority of the applications in all the other nPartitions
#
#       4. Customize the configuration file
#          npar_icod_manual_allocation.wlmpar used by the WLM global
#          arbiter (see that file for details)
#
#       5. Activate the npar_icod_manual_allocation.wlmpar file on the
#          primary host:
#
#          wlmpard -a npar_icod_manual_allocation.wlmpar
#
#       6. Activate the WLM configuration file on each nPartition:
#
#          wlmd -a npar_icod_manual_allocation.wlm
#
#       NOTE: You will activate both the npar_icod_manual_allocation.wlmpar
#       file and the npar_icod_manual_allocation.wlm file on the primary
#       host.
#
#       7. Change the number of CPUs a partition requests using wlmsend
#
#          Using the wlmsend command on a partition, you can explicitly
#          request a certain number of CPUs for that partition. The
#          configuration below defines a metric named num_cpus. You can
#          change the value of this metric with wlmsend. For example, the
#          following command requests 3 CPUs:
#
#          % /opt/wlm/bin/wlmsend num_cpus 3
#
#          After using wlmsend, wait one wlm_interval (set at 5 seconds
#          below) then use the wlminfo command from the next step to see
#          how the group's allocation is affected. Continue changing the
#          num_cpus value to see how the workload behaves with various
#          numbers of CPUs. With small nPartitions, you should be able to
#          step through all the available CPUs and evaluate workload
#          response quickly.
#
```

```
#       8. Monitor each SLO’s request for CPU shares with the command:
#
#          wlminfo slo -l -v
#
#          The output will show the shares request (“Req” column)
#          change for the nPartition as you change the num_cpus value
#          using wlmsend.

# Now for the actual configuration components.
#
# The primary_host keyword specifies the host name for the nPartition where
# WLM’s global arbiter will run. (This keyword has the same value on each
# nPartition.)
#
# See wlmconf(4) for complete HP-UX WLM configuration information.
#
primary_host = myserver;                # Change this value

#
# Set the interval on which WLM takes CPU requests and makes changes in CPU
# allocations to 5 seconds. (The default interval is 60 seconds. Using a
# smaller interval allows WLM to respond more quickly to changes in
# workload performance. If you change this interval, be sure the global
# arbiter interval, set in vpar_usage_goal.wlmpar, is greater than the new
# wlm_interval value you set here.)
#
tune {
     wlm_interval = 5;
}

#
# Change the priority (pri) value after copying this file to each nPartition.
# This value (1 being the highest priority) indicates the importance of the
# current nPartition’s applications relative to the importance of the
# applications in all the other nPartitions.
#
# When managing nPartitions, WLM equates 1 CPU to 100 shares. The cpushares
# statement causes the SLO to request 100 shares, or 1 CPU, multiplied by
# the metric num_cpus. So, if num_cpus = 7, the SLO requests 7 CPUs for
# the nPartition.
#
slo slo_myslo {
        pri = 1;                        # Change this value
        cpushares = 100 total per metric num_cpus;
}
```

```
# The following structure sets up the metric num_cpus that is used in the
# cpushares statement above. This set up allows you to change the metric's
# value using wlmsend on the command line. See Steps 7 and 8 above for
# usage and monitoring examples.
#
tune num_cpus {
        coll_argv = wlmrcvdc;
}
```

For more information on WLM's partition management, see "Managing SLOs across partitions" on page 283.

npar_icod_manual_allocation.wlmpar

This file is a configuration file for the WLM global arbiter. In combination with the WLM configuration file in the previous section, this file migrates CPUs between HP nPartitions based on the number of CPUs you request for an nPartition on the command line using `wlmsend`.

Activate the WLM global arbiter's configuration in only one nPartition on the system. However, activate the WLM configuration file in each nPartition.

```
#
# Name:
#       npar_icod_manual_allocation.wlmpar
#
# Version information:
#
#       (C) Copyright 2003-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.5 $
#
# Caveats:
#       DO NOT MODIFY this file in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
```

npar_icod_manual_allocation.wlmpar

```
# Purpose:
#       This example, used with npar_icod_manual_allocation.wlm,
#       demonstrates WLM's ability to resize nPartitions
#       using iCOD software.
#
#       This configuration is for the WLM global arbiter, wlmpard. wlmpard
#       takes CPU requests from the WLM instances in each nPartition. It
#       then shifts CPUs between the nPartitions based on the requests and
#       priorities of the workloads in the nPartitions.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.03.00 or
#       later.
#
# To implement WLM's dynamic nPartition resizing:
#       See instructions in npar_icod_manual_allocation.wlm for the steps
#       needed to configure WLM for dynamic nPartition resizing.
#

# The WLM global arbiter's interval is set to 10 seconds. Every interval,
# the arbiter takes CPU requests from the WLM instances running on the
# partitions and makes changes in the partitions' CPU allocations. (If you
# change the global arbiter interval, be sure it is greater than the
# wlm_interval value you use in any of the partition's WLM configuration
# files.)
#
# See wlmparconf(4) for complete HP-UX WLM global arbiter configuration
# information.

par {
    interval = 10;
}
```

For more information on WLM's partition management, see "Managing SLOs across partitions" on page 283.

performance_goal.template

The following file is a template showing how to use various components of the configuration file.

```
#
# Name:
#       performance_goal.template
#
# Version information:
#
#       (C) Copyright 2001-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.4 $
#
# Caveats:
#       DO NOT MODIFY this file in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       This file contains a SAMPLE HP-UX WLM configuration file, meant
#       to expand on the descriptions given in the wlmconf(4) man page.
#
#       Specifically in this file, we illustrate the use of performance-
#       based goals to determine whether an SLO is being met.
#
#       Keep in mind that this example is for illustration purposes ONLY.
#       While it does contain all the required components, the specific
#       values (such as group, user, SLO names, and executable locations)
#       are hypothetical.  The service-level objectives (SLOs) and all
#       other identifiers must be tailored to your environment.
#
#       This sample file can be viewed as a simple template, as the non-
#       keyword items that the user would need to customize are contained
#       within square brackets ([]'s).  Because of these brackets, this
#       file will not “pass” the syntax-checking mode of wlmd.
#
```

```
##################
# PRM Components #
##################

#
# prm structure
#
#       First, we define the workload groups.
#
#       For this example, we will assume two basic workload groups:
#               finance and sales
#       Each group only has one application that we will monitor.
#
prm {
   groups = [finance:2], [sales:3];
   apps = [finance]: [/opt/fin_app/count_money], # app for finance group
          [sales]: [/opt/sales/do_ebiz];          # app for sales group
}

############################
# Service-Level Objectives #
############################

#
# This is an SLO for the finance group and is only “active” on weekdays.
# It has a response-time goal, desiring that finance queries should
# complete in less than 2 seconds.  A data collector to provide these
# response-time metrics to HP-UX WLM is specified later in a tune
# structure (a tune structure is required for all metrics used in an
# slo structure).
#
# In the goal line of this SLO, even though “metric” is a keyword, it could
# also have been contained within the brackets (as a user-configurable item).
# This is because there are other types of goal statements (such as usage)
# besides a metric (performance-based) goal.  However, as this template is
# focused on showing the use of performance goals, “metric” is used
# explicitly, leaving just the “what-am-I-measuring?” piece as configurable.
#
slo [finance_query] {
   pri = [1];                          # SLO of highest priority
   mincpu = [20];                      # minimum CPU allocation (%)
   maxcpu = [50];                      # maximum CPU allocation (%)
   entity = PRM group [finance];
   goal = metric [fin_app.query.resp_time < 2.0];
   condition = [Mon - Fri];            # only active on weekdays
}
```

```
# On weekends, we do not expect any query transactions, but just in
# case, we will specify a nominal, fixed CPU allocation for this
# application for off-hours.
#
slo [finance_query_weekend] {
   pri = [1];
   mincpu = [5];
   maxcpu = [5];
   entity = PRM group [finance];
   condition = [Sat - Sun];
}

#
# This is an SLO for the Sales group, active at all times (there are no
# conditions or exceptions).
#
slo [sales_query] {
   pri = [1];
   mincpu = [40];
   maxcpu = [80];
   entity = PRM group [sales];
   goal = metric [sales_app.resp_time < 10.0];
}

#########################
# Global tune structure #
#########################

#
# Have HP-UX WLM check for new performance data (and make any necessary
# adjustments to allocations) every 30 seconds.
#
tune {
   wlm_interval = [30];                # wlmd polling interval
}

###################################
# Metric-specific tune structures #
###################################

#
# This structure specifies the data collector (coll_argv) for the
# fin_ap.query.resp_time metric, used in the goal statements of the
# finance SLOs (above). This theoretical application
# (/opt/fin_app/finance_collector) is developed or otherwise provided
# by the user.
# For more information on how to develop a data collector (also known as
# performance monitor), please see /opt/wlm/share/doc/howto/perfmon.html.
#
# This structure also specifies a constant (cntl_kp), which controls the
# rate of service-level convergence toward its goal.  For more information
# on tuning this value, see /opt/wlm/share/doc/howto/tuning.html.
#
tune [fin_app.query.resp_time] {
   coll_argv = [/opt/fin_app/finance_collector -a 123 -v];
   cntl_kp = [1.0];
}

#
# Structure specifying similar information for the sales_app.resp_time
# metric.
#
tune [sales_app.resp_time] {
   coll_argv = [/opt/sales_app/monitor -threshold 2 -freq 30];
   cntl_kp = [2.0];
}
```

For related information, see “Configuring WLM” on page 127.

stretch_goal.wlm

Stretch goals are secondary goals and are of lower priority than a workload group’s main SLO. Here, we have a stretch goal to provide a workload group with more CPU if it is available.

```
#
# Name:
#       stretch_goal.wlm
#
# Version information:
#
#       (C) Copyright 2001-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.6 $
# Caveats:
#       DO NOT MODIFY this file in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       Demonstrate the use of multiple SLOs at different priority levels,
#       but for the same workload, used to facilitate a “stretch” goal (one
#       that we’d like to have met if all other higher-priority SLOs are
#       being met).
#
#       This file is very similar to the performance_goal.template file.
#       It is provided without the user-customizable fields in brackets.
#       But, more importantly, we have added an SLO to illustrate how one
#       can accomplish stretch (“would-be-nice-if-there-are-extra-cycles”)
#       goals.

#
# PRM structure
#
#       In this example, we have two groups again:
#              finance and sales
#       Each workload has one application.
#
prm {
   groups = finance:2,
            sales:3;
   apps = finance: /opt/fin_app/count_money,  # application for finance group
          sales: /opt/sales/do_ebiz;          # application for sales group
}

#
# slo structures
#

#
# This is an SLO for the finance group, only “active” on weekdays.
slo finance_query {
   pri = 1;                       # SLO of highest priority
   mincpu = 20;                   # minimum CPU allocation (percentage)
   maxcpu = 50;                   # maximum CPU allocation (percentage)
   entity = PRM group finance;
   goal = metric fin_app.query.resp_time < 2.0;
   condition = Mon - Fri;         # only active on weekdays
}
```

```
# This is a “stretch” goal for the finance query group. If all other
# goals of higher priority (lower “pri” integral values) have been met,
# apply more CPU to group finance, so its application runs faster
# during prime time (Monday through Friday between 9am and 5pm).
#
slo finance_query_stretch {
   pri = 5;
   mincpu = 20;
   maxcpu = 60;                 # let it take more CPU if available
   entity = PRM group finance;
   goal = metric fin_app.query.resp_time < 1.0;
   condition = (Mon - Fri) && (09:00 - 17:00);
}

#
# This is an SLO for the Sales group, active at all times (there are no
# conditions or exceptions).
#
slo sales_query {
   pri = 1;
   mincpu = 40;
   maxcpu = 80;
   entity = PRM group sales;
   goal = metric sales_app.resp_time < 10.0;
}

#
# tune structures
#
#       This first structure specifies the data collector (coll_argv) for
#       the fin_ap.query.resp_time metric, used in the goal statements
#       of the finance SLOs (above). This hypothetical application
#       (/opt/fin_app/finance_collector) is developed or otherwise provided
#       by the user.
#
#       NOTE: Because the data collectors for these metrics are
#       hypothetical applications, this file will not pass the syntax-
#       checking mode of wlmd (wlmd -c stretch_goal.wlm).  A real data-
#       collecting program must exist, as HP-UX WLM will launch it and rely
#       upon it to provide performance metrics (or the HP-provided interface,
#       wlmrcvdc, can be used when appropriate).
#
tune fin_app.query.resp_time {
   coll_argv = /opt/fin_app/finance_collector -a 123 -v;
}
```

```
#
#       tune structure specifying similar information for the
#       sales_app.resp_time metric.
#
tune sales_app.resp_time {
   coll_argv = /opt/sales_app/monitor -threshold 2 -freq 30;
}
```

For information on the `condition` keyword, see “Specifying when the SLO is active (optional)” on page 192. Also see “Goals vs stretch goals” on page 187.

time_activated.wlm

The following example enables an SLO based on time, or in this case, date.

```
#
# Name:
#       time_activated.wlm
#
# Version information:
#
#       (C) Copyright 2001-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.7 $
#
# Caveats:
#       DO NOT MODIFY this script in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       Demonstrate the use of HP-UX WLM in allocating a fixed allocation
#       to a particular group of users only during a certain time period.
#
# Dependencies:
#       This example was designed to run with version HP-UX WLM A.01.02
#       or later.
#
```

```
# prm structure
#
#       See wlmconf(4) for complete HP-UX WLM configuration information.
#
#       Define all workload groups in the prm structure.  Individual
#       users are assigned to particular groups in this structure as well.
#
#       In this example configuration, the user don can execute
#       applications in either the Payroll or OTHERS workload groups.  So,
#       he can move existing processes to, or start new processes in,
#       OTHERS using the prmmove or prmrun commands.  For more information,
#       see prmmove(1) and prmrun(1).  The users paysupv and payadm can
#       execute applications in the Payroll group but nowhere else.
#
#       Note that the group OTHERS is created automatically.  Applications
#       run by users not referenced in the prm structure will execute in
#       the OTHERS group.  Given the prm structure below, only don,
#       paysupv and payadm can execute applications in the Payroll
#       group.

prm {
        groups= Payroll : 3;
        users=  don: Payroll OTHERS,
                paysupv: Payroll,
                payadm: Payroll;
}

#
# slo structure
#
#       The keywords pri (priority), and entity (group to which
#       the SLO applies) are required for any SLO specification.
#
#       The cpushares statement requests a fixed allocation of 80% of the
#       CPU resources on the system for the Payroll group.
#
#       With the condition set as shown, this SLO is only in effect
#       on the 6th and 21st of each month.  On these dates, applications
#       executing in the Payroll group will have 80% of the system's CPU
#       resources dedicated to them.  Note that on all other dates, the
#       SLO is disabled, and only the minimum default of 1% of the CPU
#       will be available to users in the Payroll group!
#
```

```
slo Payroll_Processing {
        pri = 1;
        cpushares = 80 total;
        entity = PRM group Payroll;
        condition = */06/* || */21/*;
}
```

For information on the `condition` keyword, see “Specifying when the SLO is active (optional)” on page 192.

transient_groups.wlm

This example shows how you can reduce the resources used by groups with no active SLOs.

```
#
# Name:
#       transient_groups.wlm
#
# Version information:
#
#       (C) Copyright 2003-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.5 $
#
# Caveats:
#       DO NOT MODIFY this file in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       This example demonstrates WLM’s management of groups with no active
#       SLOs. By default, WLM workload groups continue to consume system
#       resources even when the groups have no active SLOs.
#
```

```
#       By setting the transient_groups keyword to 1:
#
#       * Whenever an FSS group has no active SLOs, the group is removed
#         from the configuration and therefore consumes no resources
#
#       * Whenever a PSET-based group has no active SLOs, the group gets 0
#         CPUs
#
#       See the discussion of the transient_groups keyword in the
#       wlmconf(4) man page for information on where processes belonging to
#       such groups are placed.
#


#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.02.02 or
#       later. (A.02.02 was the first version to support PSET-based groups
#       as transient groups.)
#

# This example has an Apache web server that is used as a front-end to
# billing and paying applications, both of which are in HP Serviceguard
# packages. The Apache workload is run in an FSS group, as is the Billing
# workload. The other workload is run in a PSET group. The default user
# group OTHERS is explicitly defined.

prm {
    groups = OTHERS : 1,
             Apache : 2,
             Billing : 3,
             Paying : PSET;

# The workloads are placed in their workload groups:
    apps = Apache : /opt/hpws/apache/bin/httpd,
           Billing : /opt/orders/bin/billing,
           Paying : /opt/orders/bin/paying;
}
```

```
# Set up wlmrcvdc to pick up metrics indicating whether the Serviceguard
# packages are active; set transient_groups keyword.
#
# Have WLM modify allocations (if necessary) every 5 seconds
# because the configuration includes usage goals.
#
tune {
   coll_argv = wlmrcvdc sg_pkg_active;
   transient_groups = 1;
   wlm_interval = 5;
}

# The SLO Paying_slo applies to a PSET-based group, so absolute CPU units
# are in effect. Thus, each CPU represents 100 shares. The mincpu and
# maxcpu values can be up to (n * 100) shares, where n is the total
# number of CPUs on the system.

# The SLO for the Apache workload is going to request 1 to 2 CPUs. It has a
# usage goal, which will ensure the group uses a certain percentage of its
# CPU allocation. This SLO is priority 1, as Apache must get the CPU it
# needs.
slo Apache_slo {
    pri = 1;
    mincpu = 100;
    maxcpu = 200;
    entity = PRM group Apache;
    goal = usage _CPU;
}

# Whenever the Billing Serviceguard package is active, this SLO’s associated
# workload group is allocated 1 to 4 CPUs, based on a usage goal. When the
# package is not active, the Billing workload group, which is an FSS group,
# is removed from the configuration and consumes no resources. This SLO is
# priority 2; it is not as critical as the Apache workload.
slo Billing_slo {
    pri = 2;
    mincpu = 100;
    maxcpu = 400;
    entity = PRM group Billing;
    goal = usage _CPU;
    condition = metric Billing_active;
}
```

```
# Whenever the Paying Serviceguard package is active, this SLO’s associated
# workload group is allocated 1 to 4 CPUs, based on a usage goal. When the
# package is not active, the Paying workload group, which is based on a
# PSET group, gets no CPUs. This SLO is priority 3; it gets CPU only after
# the Apache_slo and Billing_slo are satisfied.
slo Paying_slo {
    pri = 3;
    mincpu = 100;
    maxcpu = 400;
    entity = PRM group Paying;
    goal = usage _CPU;
    condition = metric Paying_active;
}
```

For information on transient groups, see the “Temporarily removing groups with inactive SLOs (optional)” on page 207.

twice_weekly_boost.wlm

The next example presents an interesting mix of conditions for when SLOs are active.

```
#
# Name:
#       twice_weekly_boost.wlm
#
# Version information:
#
#       (C) Copyright 2001-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.11 $
#
# Caveats:
#       DO NOT MODIFY this file in its /opt/wlm/toolkits location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       Demonstrate a conditional allocation with a moderately complex
#       condition.
#       A baseball park’s server runs a number of different workloads
#       for two groups: the front office and the scouting staff.  The
#       basic allocations are 30 shares for front_office, 30 shares for
#       scouting, and the remainder (40 in this case) for OTHERS.
#
#       The front office staff has one important, long running job,
#       “analyze_ticket_sales”, which is usually run every Tuesday and
#       Thursday.  During these days, the front_office group receives a
#       ‘boost’, and its allocation is 60 shares.
#
#       For these regularly scheduled runs, the system administrator relies
#       on HP-UX WLM to automatically re-allocate CPU resources.
#
#       In addition to those regular runs, occasionally a special events
#       ticket sale is held. For these sales, the job is run at an
#       unscheduled time, with an allocation of 70 shares.
#
#       To manually enable the front office boost for special events,
#       the system administrator executes this command:
#
#       % wlmsend front_office.boost_enable 1
#
#       To manually disable after the special event, the system
#       administrator executes this command:
#
#       % wlmsend front_office.boost_enable 0
#
#       The scouting staff has one particularly important, long running
#       job, /opt/baseball/scouting/bin/batting_averages, which is
#       usually run every Monday and Wednesday, when it receives 60
#       shares.
#
#       In addition to those regular runs, occasionally farm league
#       makeup games are held, and the job must be run at an unscheduled
#       time.  During these times, the scouting group receives a manual
#       ‘boost’ to 70 shares.
#
#       To manually enable the scouting boost for special events,
#       the system administrator executes this command:
#
#       % wlmsend scouting.boost_enable 1
#
#       To manually disable after the special event, the system
#       administrator executes this command:
#
#       % wlmsend scouting.boost_enable 0
#       Manually requested boosts receive a higher priority than
#       the automatic date-based boosts. This is achieved with the ‘pri’
#       keyword in the slo definitions.
#
#       In the unusual case that the front office *and* the scouting
#       team manually boost their allocations, the front office takes
#       priority, and the scouting boost is disallowed.
#
# Allocation change frequency:
#       Because wlm_interval is not explicitly set, the 60 second default
#       is used. This means wlmd will collect measurements and make
#       allocation changes every 60 seconds.
#
# Components:
#       Uses the wlmsend and wlmrcvdc tools to relay a metric from
#       an outside user.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.01.02 or
#       later.
#



#
# prm structure
#       Create workload groups and designate which workload binaries
#       will be placed in each. Because the baseball application
#       is nicely split into scouting/bin and finance/bin, wildcards
#       can be used to place the processes in the correct workload
#       groups.
#
#       See wlmconf(4) for complete HP-UX WLM configuration information.
#
prm {
        groups = OTHERS : 1,
                 front_office : 2,
                 scouting : 3 ;

        apps = scouting : “/opt/baseball/scouting/bin/*”,
               front_office : “/opt/baseball/finance/bin/*” ;
}




##########
```

```
#
# Give the front office its basic allocation of 30 shares.
#
slo front_office_basic {
        pri = 3;
        entity = PRM group front_office;
        cpushares = 30 total;
}

#
# When the day is correct, boost the front office to 60 shares, unless
# scouting has requested a manual boost.
#
slo front_office_date_boost {
        pri = 2;
        entity = PRM group front_office;
        cpushares = 60 total;
        condition = (Tue || Thu);
        exception = (metric scouting.boost_enable > 0) ;
}

#
# If the system administrator requests it, boost the front
# office to 70 shares.
#
slo front_office_manual_boost {
        pri = 1;
        entity = PRM group front_office;
        cpushares = 70 total;
        condition = (metric front_office.boost_enable > 0) ;
}

##########
#
# Give the scouting staff its basic allocation of 30 shares.
#
slo scouting_basic {
        pri = 3;
        entity = PRM group scouting;
        cpushares = 30 total;
}




# When the day is correct, boost scouting to 60 shares, unless
# scouting has requested a manual boost.
#
slo scouting_date_boost {
        pri = 2;
        entity = PRM group scouting;
        cpushares = 60 total;
        condition = (Mon || Wed);
        exception = (metric scouting.boost_enable > 0) ;
}

#
# If the system administrator requests it, boost scouting to 70 shares.
#
slo scouting_manual_boost {
        pri = 1;
        entity = PRM group scouting;
        cpushares = 70 total;
        condition = (metric scouting.boost_enable > 0) ;

        # front office manual boost overrides scouting manual boost
        exception = (metric front_office.boost_enable > 0) ;
}

# Any CPU that remains after satisfying the above SLOs is given to the
# OTHERS group by default. You can change this default using the
# distribute_excess keyword. For more information on this keyword, see
# the wlmconf(4) man page.

#
# Relay the boost enable metrics from the outside user.
#
tune front_office.boost_enable {
    coll_argv = wlmrcvdc;
}

tune scouting.boost_enable {
    coll_argv = wlmrcvdc;
}
```

For information on the `condition` keyword, see “Specifying when the SLO is active (optional)” on page 192.

usage_goal.wlm

This example shows a CPU usage goal, where WLM attempts to keep a workload’s CPU utilization, defined as (CPU used) / (CPU allocated), between a certain range.

```
#
# Name:
#       usage_goal.wlm
#
# Version information:
#
#       (C) Copyright 2001-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.9 $
#
# Caveats:
#       DO NOT MODIFY this script in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       Demonstrate the usage goal for service-level objective (SLO)
#       specifications.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.01.01
#       or later.
#

#
# prm structure
#
#       See wlmconf(4) for complete HP-UX WLM configuration information.
#
prm {
        groups= OTHERS : 1,
                Orders : 2,
                Batch: 3;

        gmincpu = OTHERS : 10;
}
```

```
# slo structures
#

#
#       This SLO is defined with a CPU usage, or utilization, goal.  This
#       is a special goal in that WLM tracks the utilization for you.
#       By default, WLM attempts to keep the group’s utilization of its
#       allocated CPU between 50% and 75%.  If utilization falls below 50%
#       (due perhaps to fewer applications running), WLM reduces the Orders
#       group’s allocation, making more CPU available to the OTHERS and
#       Batch groups.  Similarly, when utilization is above 75%, WLM
#       allocates more CPU to the group.
#
slo order_processing {
        pri = 1;
        mincpu = 20;
        maxcpu = 80;
        entity = PRM group Orders;
        goal = usage _CPU;
}

#
#       This SLO is also defined with a CPU usage goal.  It is at a
#       lower priority than the SLO for the Orders group.  We also
#       explicitly specify the utilization bounds.  We use a small value
#       (25) for the upper bound, because we want this group to request
#       more CPU whenever there is any activity in the group.  The
#       order_processing SLO is higher priority, so it can steal CPU from
#       this SLO whenever it needs it.  The gmincpu value for OTHERS
#       ensures that we never completely starve that group.
#
#       If the boundary values are not specified (as in the previous SLO),
#       then they default to 50 and 75.
#
slo batch_processing {
        pri = 99;
        mincpu = 5;
        maxcpu = 100;
        entity = PRM group Batch;
        goal = usage _CPU 15 25;
}
```

```
# tune structure
#
#       Have WLM modify allocations (if necessary) every 5 seconds
#       because the configuration includes usage goals.
#

tune {
    wlm_interval = 5;
}
```

For more information on usage goals, “Specifying a goal (optional)” on page 183.

user_application_records.wlm

The next example shows how to place applications in workload groups. It also shows that application records take precedence when both user records and application records are in effect for an application.

```
#
# Name:
#       user_application_records.wlm
#
# Version information:
#
#       (C) Copyright 2001-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.11 $
#
# Caveats:
#       DO NOT MODIFY this file in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       Demonstrate the use of, and precedence between, user and application
#       records in placing processes in workload groups.
#
#       A development server is funded and used by two groups, test and
#       development.  Along with normal test and development work, the
#       server also supports the two groups’ joint website (via apache),
#       and is the box employees use to fetch web pages (with netscape).
#       However, in the past, both web functions, apache and netscape,
#       have negatively affected the performance of the box.  To correct
#       this problem, users' netscape sessions are limited to 10% of the
#       CPU, and the httpd processes are limited to 10%. The remaining
#       80 percent is split 35 for test, 35 for development, and 10
#       percent for users not belonging to either group.
#
# Components:
#       No external data collectors are required.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.01.02 or
#       later. It uses the cpushares keyword introduced in A.01.02, so
#       is incompatible with earlier versions of HP-UX WLM.
#



#
# prm structure
#       Create workload groups. Designate which workload binaries
#       and users will be placed in each. We will be managing
#       two workloads, apache and netscape, and two groups of users,
#       testers and coders. Users not belonging to either group
#       are placed in OTHERS.
#
#       The users section places individuals into a workload group based
#       on their login.  Users not listed will run in OTHERS.  Because
#       application records take precedence over user records, if user larry
#       runs netscape, that process will run in workload group 'surfers'
#       rather than 'testers'.  In a similar manner, if user larry runs
#       httpd, it will run in 'servers' because the app record is used
#       even though the user, larry, does not have permission to
#       explicitly start or move other jobs to 'servers'.  Consult
#       wlmconf(4) and prmconf(4) for more information.
#
#
#       Note that netgroups can be used in the users record.  For
#       instance, if all development staff belonged to a netgroup
#       'devel_netgrp', this line could be used to place them all in
#       workload group 'coders':
#
#       users = +devel_netgrp : coders surfers,
#              +test_netgrp : testers surfers;
#
#       See wlmconf(4) for complete HP-UX WLM configuration information.
```

```
#
prm {
        groups = OTHERS : 1,
                 testers : 2,
                 coders : 3,
                 servers : 4,
                 surfers : 5;

        apps = servers : /opt/hpws/apache/bin/httpd,
               surfers : /opt/netscape/netscape;

        users = moe : coders surfers,
                curly : coders surfers,
                larry : testers surfers;
}

#
# Grant 35 shares to coders.
#
slo coders_fixed {
        pri = 1;
        entity = PRM group coders;
        cpushares = 35 total;
}

#
# Grant 35 shares to testers.
#
slo testers_fixed {
        pri = 1;
        entity = PRM group testers;
        cpushares = 35 total;
}

#
# Grant 10 shares to servers.
#
slo servers_fixed {
        pri = 1;
        entity = PRM group servers;
        cpushares = 10 total;
}



# Grant 10 shares to surfers.
```

```
#
slo surfers_fixed {
        pri = 1;
        entity = PRM group surfers;
        cpushares = 10 total;
}

# Any CPU that remains after satisfying the above SLOs is given to the
# OTHERS group by default. You can change this default using the
# distribute_excess keyword. For more information on this keyword, see
# the wlmconf(4) man page.
```

For information on user and application records, see “Specifying users’ workload group access (optional)” on page 152 and “Assigning applications to workload groups (optional)” on page 153.

For more information on how the application manager works, see “How application processes are assigned to workload groups at start-up” on page 449 and “How the application manager affects workload group assignments” on page 452.

vpar_usage_goal.wlm

This file, in combination with the global arbiter configuration file in the next section, migrates CPUs between HP-UX Virtual Partitions based on usage goals.

Activate the WLM configuration file in each vPar. However, you only need to activate the WLM global arbiter’s configuration in one system on your network.

```
#
# Name:
#       vpar_usage_goal.wlm
#
# Version information:
#
#       (C) Copyright 2003-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.11 $
# Caveats:
#       DO NOT MODIFY this file in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
#
# Purpose:
#       This example, used with vpar_usage_goal.wlmpar, demonstrates WLM's
#       ability to resize HP-UX Virtual Partitions. The applications within
#       a virtual partition are considered a single workload. WLM shifts
#       CPUs between the virtual partitions based on the requests and
#       priorities of these workloads.
#
#       A usage goal is placed on the workload. This goal results in
#       WLM's automatically allocating more CPU to the virtual partition as
#       it is needed. Similarly, when the group is not busy, WLM reduces
#       its CPU allocation.
#
#       The workload can request up to 12800 shares. This number was chosen
#       because the current maximum number of CPUs in a server is 128.
#       When managing virtual partitions, WLM equates 1 CPU to 100 shares.
#       Multiplying 128 CPUs by 100 shares/CPU gives 12800 shares. Using
#       this value ensures that the workload's group will not be limited in
#       how much CPU it can consume. However, WLM does allocate CPUs within
#       the limits you set for a virtual partition when you specify its
#       minimum and maximum number of CPUs using vparmodify.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.03.00 or
#       later. (A.03.00 was the first version to support strictly
#       host-based configurations.)

#
# To implement WLM's dynamic virtual partition resizing:
#       1. Set the primary_host keyword in this file
#
#       2. Copy this WLM configuration to each virtual partition in the
#          system
#
#       3. Adjust the priority for the SLO named slo_myslo to reflect the
#          priority of the applications for the current virtual partition
#          relative to the priority of the applications in all the other
#          virtual partitions
#
#       4. Customize the configuration file vpar_usage_goal.wlmpar used by
#          the WLM global arbiter (see that file for details)
#       5. Activate the vpar_usage_goal.wlmpar file on the primary host:
#
#          wlmpard -a vpar_usage_goal.wlmpar
#
#       6. Activate the WLM configuration file on each virtual partition:
#
#          wlmd -a vpar_usage_goal.wlm
#
#       NOTE: You will activate both the vpar_usage_goal.wlmpar file and
#       the vpar_usage_goal.wlm file on the primary host.
#
#       7. Monitor each SLO's request for CPU shares with the command:
#
#          wlminfo slo -l -v
#
#          The output will show the shares request ("Req" column)
#          increasing for the partitions as the loads on those partitions
#          increase.

#
#  The primary_host keyword specifies the host name for the virtual
#  partition where WLM's global arbiter will run. (This keyword has the
#  same value on each virtual partition.)
#
# See wlmconf(4) for complete HP-UX WLM configuration information.
#
primary_host = myserver;                # Change this value

#
# Set the interval on which WLM takes CPU requests and makes changes in CPU
# allocations to 5 seconds. (The default interval is 60 seconds. Using a
# smaller interval allows WLM to respond more quickly to changes in
# workload performance. If you change this interval, be sure the global
# arbiter interval, set in vpar_usage_goal.wlmpar, is greater than the new
# wlm_interval value you set here.)
#
tune {
     wlm_interval = 5;
}

#
# Change the priority (pri) value after copying this file to each virtual
# partition. This value (1 being the highest priority) indicates the
# importance of the current virtual partition's applications relative to
# the importance of the applications in all the other virtual partitions.
#
```

```
slo slo_myslo {
        pri = 1;                        # Change this value
        mincpu = 1;
        maxcpu = 12800;
        goal = usage _CPU;
}
```

For more information on WLM's partition management, see "Managing SLOs across partitions" on page 283.

vpar_usage_goal.wlmpar

This file is a configuration file for the WLM global arbiter. In combination with the WLM configuration file in the previous section, this file migrates CPUs between HP-UX Virtual Partitions based on usage goals.

Activate the WLM global arbiter's configuration in only one system on your network. However, activate the WLM configuration file in each vPar to be managed by WLM.

```
#
# Name:
#       vpar_usage_goal.wlmpar
#
# Version information:
#
#       (C) Copyright 2003-2005 Hewlett-Packard Development Company, L.P.
#
#       $Revision: 1.5 $
#
# Caveats:
#       DO NOT MODIFY this file in its /opt/wlm/examples/wlmconf location!
#       Make modifications to a copy and place that copy outside the
#       /opt/wlm/ directory, as items below /opt/wlm will be replaced
#       or modified by future HP-UX WLM product updates.
# Purpose:
#       This example, used with vpar_usage_goal.wlm, demonstrates WLM’s
#       ability to resize HP-UX Virtual Partitions.
#
#       This configuration is for the WLM global arbiter, wlmpard. wlmpard
#       takes CPU requests from the WLM instances in each virtual
#       partition. It then shifts CPUs between the virtual partitions based
#       on the requests and priorities of the workloads in the virtual
#       partitions.
#
# Dependencies:
#       This example was designed to run with HP-UX WLM version A.03.00 or
#       later.
#
# To implement WLM’s dynamic virtual partition resizing:
#       See the instructions in vpar_usage_goal.wlm for the steps needed to
#       configure WLM for dynamic virtual partition resizing.
#

# The WLM global arbiter’s interval is set to 10 seconds. Every interval,
# the arbiter takes CPU requests from the WLM instances running on the
# partitions and makes changes in the partitions’ CPU allocations. (If you
# change the global arbiter interval, be sure it is greater than the
# wlm_interval value you use in any of the partitions’ WLM configuration
# files.)
#
# See wlmparconf(4) for complete HP-UX WLM global arbiter configuration
# information.

par {
    interval = 10;
}
```

For more information on WLM’s partition management, see “Managing SLOs across partitions” on page 283.

12 Monitoring SLO compliance and WLM

WLM allows you to monitor SLO compliance and a wealth of other information through `wlminfo`, `wlmgui`, and EMS. For more information, see:

- "Monitoring WLM with the `wlminfo` command" on page 357
- "Monitoring WLM with the `wlmgui` command" on page 360
- "Monitoring WLM with EMS" on page 367

Monitoring WLM with the `wlminfo` command

The `wlminfo` command provides various WLM data, with reports focusing on SLOs, metrics, or workload groups. The command has both a command-line interface and a graphical interface.

For information on `wlminfo` options and output, see the wlminfo(1M) man page.

NOTE In order to ensure the smooth running of the operating system and key HP-UX daemons, WLM runs these processes in a special workload group called `PRM_SYS`. This group is not restrained by WLM: It consumes system resources as needed. As a result, a workload group's CPU usage (shown in the 'CPU Util' column) may be less than its CPU shares because `PRM_SYS` requires some of the group's resources. However, the low usage could also be the result of the group's low CPU demands, or a combination of the two factors. Also, there may be times when CPU usage is slightly above the shares value due to dynamics in the CPU scheduler that WLM uses.

A few examples of `wlminfo` are shown below.

In the first example, we focus on SLOs. Entering `wlminfo slo -v`, we get output that includes the SLOs' goals, as well as the metrics that show how the workloads are performing relative to the goal. Also, we see from the 'Concern' column that two SLOs are Disabled, most likely due to a `condition` statement. This column helps highlight information. The 'Req' column for these SLOs shows a dash (-), indicating there are no requests being made for the SLOs. The 'State' column indicates whether an SLO is passing or is off.

```
% /opt/wlm/bin/wlminfo slo -v
```

```
Tue Nov 11 16:06:09 2003

SLO Name         Group    Pri Metric    Goal     Req Shares  State Concern
s_nightly_kickof g_nightl   1      -       -       -      0  OFF   Disabled
s_nightly_run    g_nightl   1   30.5      75       -      0  OFF   Disabled
s_apache_10min   g_apache   2      7       -      72     72  PASS
s_apache_2min    g_apache   2      0       -      72     72  PASS
s_nice           g_nice     2   30.5      50      30     30  PASS
s_others         OTHERS     2   1.35      50     433    462  PASS
s_team_playgroun g_team     2      0      75       1     30  PASS
s_apache_more    g_apache   3      0       -      72     72  PASS
s_nice_xtra_min  g_nice     3      -       -      30     30  PASS
s_team_playgroun g_team     3      -       -      30     30  PASS
```

Now we focus on workload groups. Entering `wlminfo group`, we see by the 'CPU Util' column how much CPU the processes in the workload groups are using.

```
% /opt/wlm/bin/wlminfo group
```

```
Tue Nov 11 16:06:27 2003

Workload Group    PRMID  CPU Shares  CPU Util  Mem Shares  State
OTHERS                1      432.00    171.34        0.00  ON
g_nice                2       84.00     49.22        0.00  ON
g_nightly             3        0.00      0.00        0.00  OFF
g_team                4        6.00      0.00        0.00  ON
g_apache              5       72.00      0.00        0.00  ON
_IDLE_                7        6.00      0.00        0.00  ON
```

Last, we look at the metrics used in the current WLM configuration. The 'PID' column shows the PID of the data collector providing the metric to WLM. In this example, the WLM daemon, `wlmd`, with PID 2103 is providing many of the metrics itself, for usage goals. (Because `wlmd` starts, and then starts the data collectors, `wlmd` typically has the lowest PID.) The 'State' column indicates whether the metric value was updated in the interval (`NEW`), no value has been received for the metric since the WLM daemon started (`INIT`), or the metric's value was not updated in the interval (`OLD`). The last column shows the value received for the metric.

```
% /opt/wlm/bin/wlminfo metric
```

```
Tue Nov 11 16:06:45 2003

Metric Name                PID State Value
_CPU_g_nightly            2103 NEW   30.549558
m_nightly_on              2107 OLD   0.000000
m_nightly_procs           2108 OLD   6.600000
_CPU_g_team               2103 NEW   0.000000
_CPU_OTHERS               2103 NEW   65.095184
_CPU_g_nice               2103 NEW   17.218712
m_apache_access_10min     2109 NEW   7.000000
m_apache_access_2min      2110 NEW   0.000000
m_list.cgi_procs          2111 NEW   0.000000
```

Monitoring WLM with the `wlmgui` command

To display monitoring data graphically, use the `wlmgui` command.

The interface has three tabs:

- Monitor
- Modify
- Deploy

You can perform these operations on the local system or on remote systems.

To begin monitoring:

Step 1. Set your DISPLAY environment variable:

Step 2. Start the WLM GUI:

```
# /opt/wlm/bin/wlmgui
```

Step 3. Select the Monitor tab

Step 4. Select the **[Add]** button

You will be prompted for a system running WLM to monitor

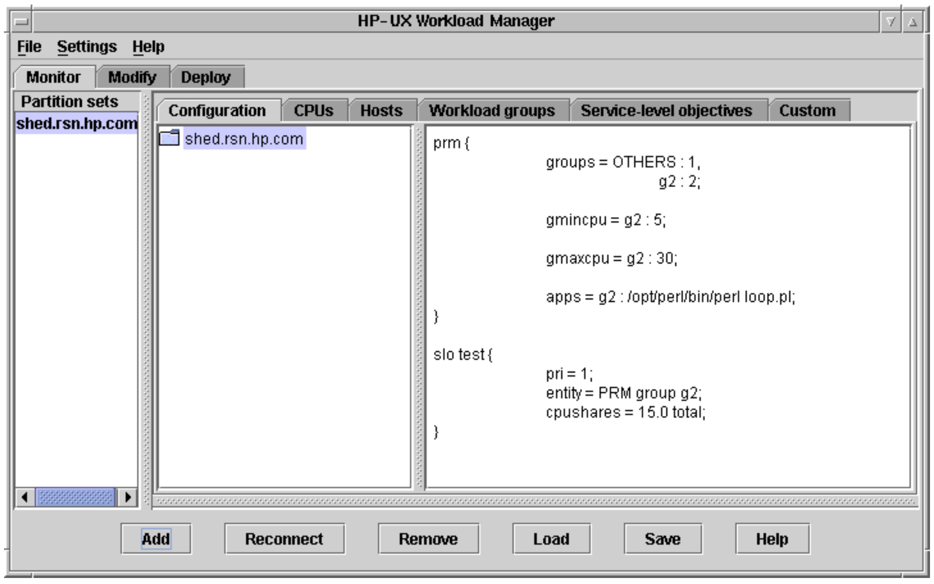
For additional information on navigating the WLM GUI, see its online help.

Monitoring the configuration

You can view WLM configurations as well as WLM global arbiter configurations using the GUI. To see a configuration, in the Monitor tab, select the Configurations tab.

Figure 12-1 shows parts of a configuration. Scroll bars are available to view the entire configuration.

Figure 12-1 **Monitoring a configuration**


HP-UX Workload Manager
File Settings Help
Monitor Modify Deploy
Partition sets
shed.rsn.hp.com
Configuration CPUs Hosts Workload groups Service-level objectives Custom
shed.rsn.hp.com
Add Reconnect Remove Load Save Help

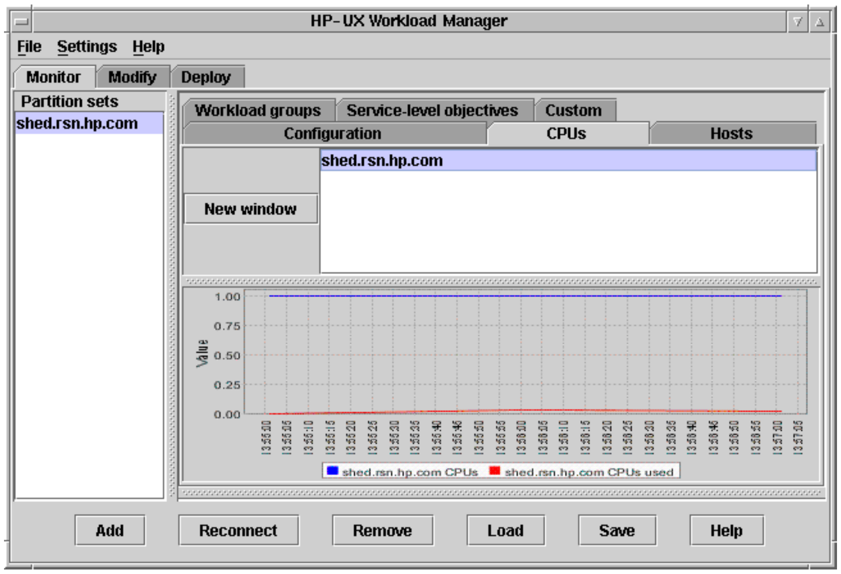
```
prm {
                groups = OTHERS : 1,
                                g2 : 2;

                gmincpu = g2 : 5;

                gmaxcpu = g2 : 30;

                apps = g2 : /opt/perl/bin/perl loop.pl;
}

slo test {
                pri = 1;
                entity = PRM group g2;
                cpushares = 15.0 total;
}
```

Monitoring the number of CPUs

The GUI allows you to monitor how the number of CPUs has changed over time. This feature is useful when managing partitions. To see this graph, in the Monitor tab, select the CPUs tab.

Figure 12-2 shows the system has had a constant number of CPUs active.

Figure 12-2 Monitoring the number of CPUs

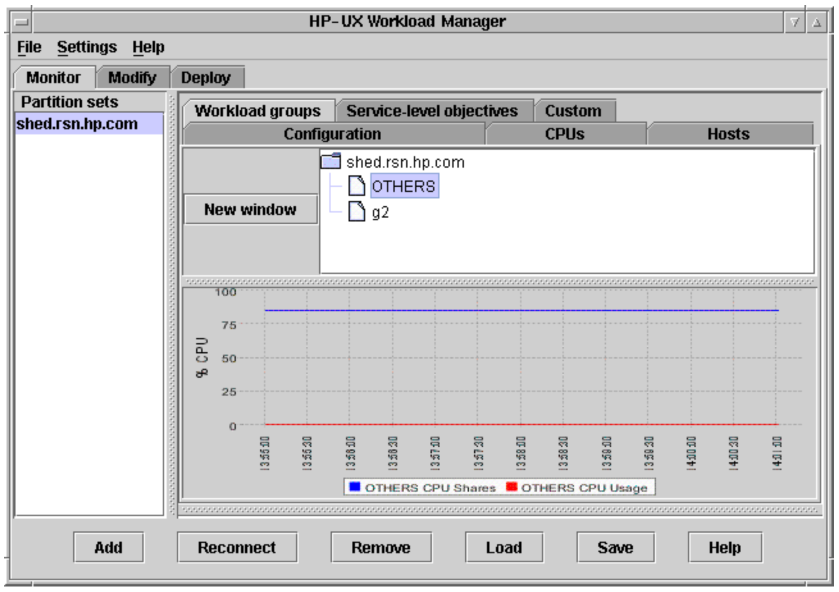


Monitoring the workload groups

By default, the “Workload groups” tab allows you to monitor the CPU shares and CPU usage for one or more groups.

Figure 12-3 shows how much CPU is allocated to the OTHERS group as well as how much it is using.

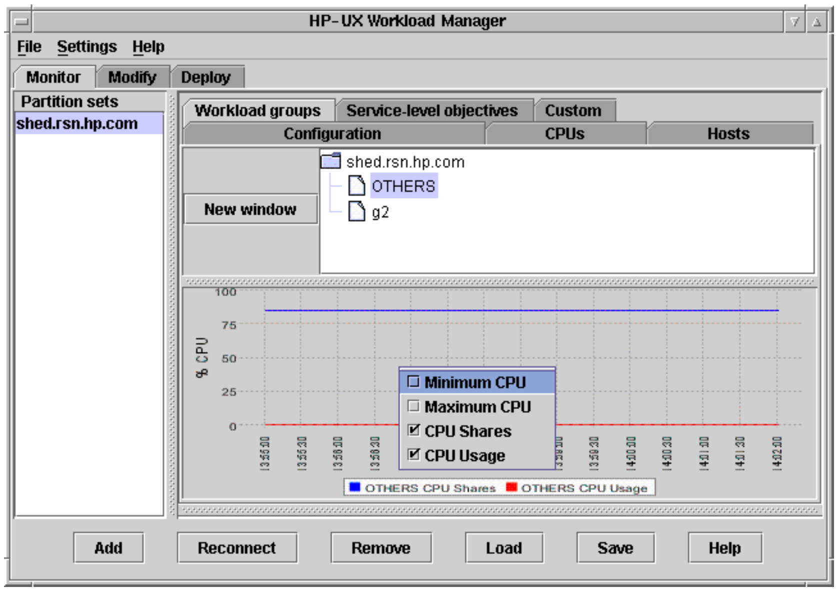
Figure 12-3 Monitoring a workload group



In addition to the default 'CPU Shares' and 'CPU Usage' values, you can graph the selected groups' 'Minimum CPU' and 'Maximum CPU' values. These correspond to the `gmincpu` and `gmaxcpu` keywords in the configuration.

To adjust what values are being graphed, right-click in the graph area to display the menu of options. Figure 12-4 shows this menu.

Figure 12-4 Items to graph when monitoring a workload group

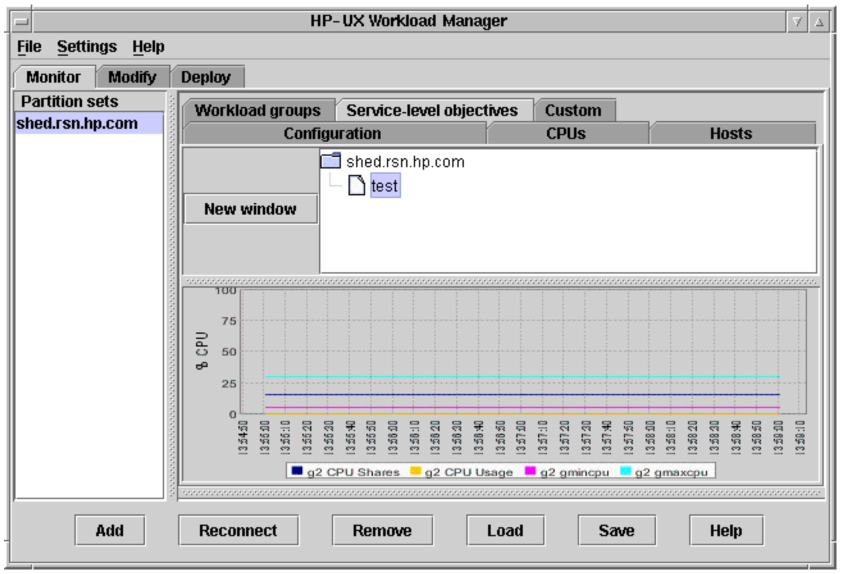


Monitoring SLOs

In the “Service-level objectives” tab, we can see graphs of metrics used in SLOs—along with the CPU allocations WLM granted in response to the changing metrics. This view provides a host of other data.

Figure 12-5 shows details for the SLO for the `test` group.

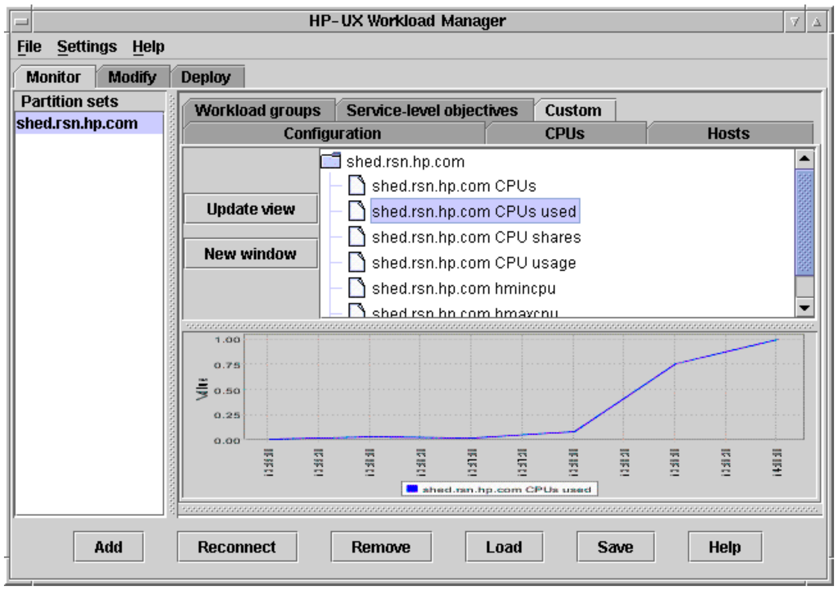
Figure 12-5 **Graphing SLOs**



Monitoring items you define

The Custom tab allows you to pull together any graphable items you desire. Figure 12-6 shows one item graphed.

Figure 12-6 **Custom graphing**

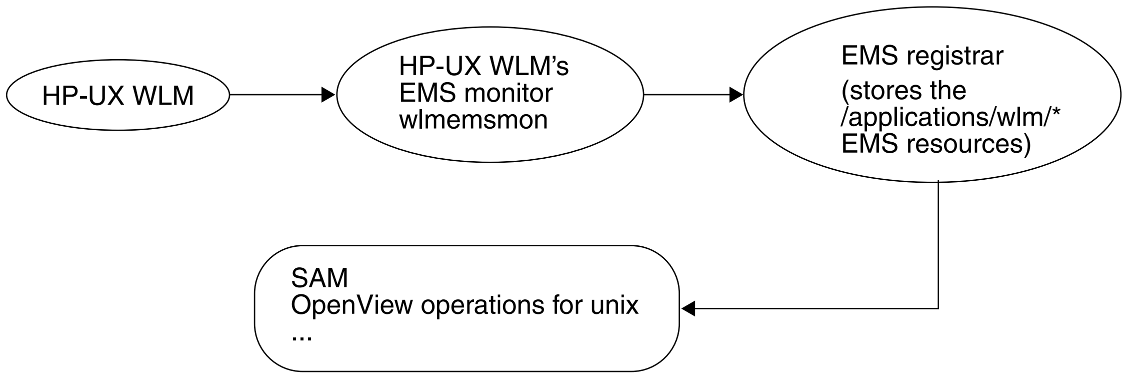


Monitoring WLM with EMS

WLM provides an EMS monitor to track the workloads' SLO compliance and WLM itself. You can configure EMS to send notification of items such as SLO performance. The monitor places this data in the standard EMS registrar for access from SAM, HP OpenView operations for unix, and other EMS clients.

Figure 12-7 illustrates the role of EMS in using WLM.

Figure 12-7 Interaction between WLM and EMS


HP-UX WLM
HP-UX WLM's EMS monitor wlmemsmon
EMS registrar (stores the /applications/wlm/* EMS resources)
SAM
OpenView operations for unix
...

What EMS resources are available?

The WLM EMS monitor, /opt/wlm/lbin/wlmemsmon, provides information on how well WLM and the managed applications are performing. `wlmemsmon` monitors the WLM daemon `wlmd` and provides EMS resources for an EMS client to monitor. The EMS client can also receive event notification periodically, when a value changes, or when a value crosses some threshold. For more information on EMS capabilities or operational paradigm, see the *Using the Event Monitoring Service* manual.

Table 12-1 provides a quick reference for the EMS resources that WLM provides. For more detailed information, see the sections following the table.

Table 12-1 Overview of WLM's EMS resources

To determine	Check the following EMS resource
Whether the WLM daemon is up or down	/applications/wlm/daemon_status
When the current configuration was activated	/applications/wlm/config_modify
The priority for the SLO *slo_name*	/applications/wlm/slo_config/*slo_name*/priority
The metric that SLO *slo_name* uses	/applications/wlm/slo_config/*slo_name*/metric
The workload group to which SLO *slo_name* applies	/applications/wlm/slo_config/*slo_name*/groupname
Whether SLO *slo_name* is: • Without an active data collector • Without performance data since the data collector started • Inactive • Passing • Failing due to higher priority SLOs • Failing due to its `maxcpu` setting • Failing for any other reason	/applications/wlm/slo_status/*slo_name*

WLM status and time of last configuration

EMS resources regarding WLM status and the time of the last configuration include:

- /applications/wlm/daemon_status

  Indicates the state of the WLM daemon. This resource is polled and is of type `ENUM`. Possible values are:

`WLM_DOWN`	WLM daemon not running
`WLM_UP`	Daemon is running: PRM configuration is actively being managed by the WLM daemon

- /applications/wlm/config_modify

  Indicates the time when the last WLM configuration was activated. This resource is polled and is of type `STRING`.

SLO configuration data

EMS resources regarding WLM's SLO configuration include:

- /applications/wlm/slo_config/

  This resource class contains resources for each SLO configured in WLM. Each of the resources provides information specific to that SLO.

- /applications/wlm/slo_config/*slo_name*/

  This resource class contains resources that provide information about the SLO *slo_name*.

- /applications/wlm/slo_config/*slo_name*/priority

  This resource specifies the priority for *slo_name*. It is polled and of type `INT`.

- /applications/wlm/slo_config/*slo_name*/metric

  This resource specifies the *metric* to which the SLO applies. It is polled and of type `STRING`.

- /applications/wlm/slo_config/*slo_name*/groupname

  This resource specifies the workload group *groupname* to which the SLO applies. It is polled and of type `STRING`.

SLO status updates

EMS resources regarding the status of WLM’s SLOs include:

- /applications/wlm/slo_status/

  This class contains a resource instance for each SLO. That instance provides the status of that SLO.

- /applications/wlm/slo_status/*`slo_name`*

  This provides the status for *`slo_name`*. It is both polled and asynchronous, and of type `ENUM`. Possible values are:

`WLM_SLO_COLLECTOR_DIED`	The data collector for this SLO exited; consequently, the status (passing or failing) is unknown
`WLM_SLO_STARTUP_NODATA`	The data collector for this SLO has started but not yet sent data to WLM; consequently, the status (passing or failing) is unknown
`WLM_SLO_INACTIVE`	This SLO is currently inactive
`WLM_SLO_PASS`	This SLO is currently passing (meeting its goal)
`WLM_SLO_CLIP_FAIL`	This SLO is currently failing, but is not receiving as large an allocation as requested by the controller, due to higher-priority SLOs
`WLM_SLO_AT_MAXCPU_FAIL`	This SLO is currently failing, but is receiving the maximum CPU allocation allowed for this SLO
`WLM_SLO_FAIL`	This SLO is failing

Configuring EMS notification

Use SAM to configure how and when you should be notified of the values of WLM resources:

Step 1. Log in as root and start SAM:

```
# /usr/sbin/sam
```

Step 2. Double-click the Resource Management icon.

Step 3. Double-click the Event Monitoring Service icon.

Step 4. Select Actions followed by Add Monitoring Request from the menu bar.

Step 5. Double-click the following resource class:

applications

Step 6. Double-click the following resource class:

wlm

Step 7. Navigate to the desired resource.

Step 8. Double-click the resource.

Step 9. Specify the Monitoring Request Parameters to indicate how you want to receive notification of various WLM events.

Step 10. Select the OK button.

A WLM command reference

This appendix describes the following WLM commands:

- `wlmaudit`

  WLM audit report generator
- `wlmcert`

  WLM certificate manager
- `wlmcomd`

  WLM communications daemon
- `wlmcw`

  WLM configuration wizard
- `wlmd`

  WLM daemon
- `wlmgui`

  WLM graphical user interface
- `wlminfo`

  WLM information monitor
- `wlmpard`

  WLM global arbiter (cross-partition management)
- `wlmrcvdc`

  WLM built-in data collector
- `wlmsend`

  Command that makes your command-line or script data available to the `wlmrcvdc` utility for forwarding to the WLM daemon

wlmaudit

The `wlmaudit` command displays audit data generated by WLM and its global arbiter.

You must use the `-t` option with the WLM daemon `wlmd` or the global arbiter daemon `wlmpard` before using `wlmaudit`.

`wlmaudit` uses /opt/perl/bin/perl to display data from audit files stored in the /var/opt/wlm/audit/ directory. For information on the structure of these files, see the wlmaudit(1M) man page.

The command syntax is:

```
wlmaudit -h

wlmaudit -V

wlmaudit [-d {wlmd | wlmpard}] [-s start_date] [-e end_date] [-o html]
```

where

`-h`

Displays usage information and exits. This option overrides all other options.

`-V`

Displays version information and exits. This option overrides all other options other than `-h`.

`-d wlmd | wlmpard`

Specifies the daemon for which to display audit data. If you do not specify this option, `wlmaudit` displays the audit data from both daemons, if available.

`-s` *`start_date`*

Instructs `wlmaudit` to display audit data beginning from *`start_date`*. The default is 01/01/1900. Use the format mm/dd/yyyy when specifying *`start_date`*.

`-e` *`end_date`*

Instructs `wlmaudit` to display audit data up to *`end_date`*. The default is the date on the current system. Use the format mm/dd/yyyy when specifying *`end_date`*.

`-o html`

Displays audit data in a formatted HTML report. The default is text.

wlmcert

`wlmcert` allows you to manage your WLM security certificates.

For information on setting up secure communications, see the section HOW TO SECURE COMMUNICATIONS in the wlmcert(1M) man page

The command syntax is:

```
wlmcert -h [cmd]

wlmcert -V

wlmcert reset

wlmcert install -c certificate

wlmcert delete -c certificate

wlmcert list

wlmcert extract [-d directory]
```

where

`-h [`*`cmd`*`]`

Displays usage information and exits. This option overrides all other options.

To get usage information for the command `reset`, `install`, `delete`, `list`, or `extract`, specify the command after `-h`. For example:

```
# wlmcert -h reset
```

`-V`

Displays version information and exits. This option overrides all options other than `-h`.

`reset`

Creates the certificates for the system on which the command is executed.

Only root can execute this operation.

This operation is performed automatically when you install WLM. After running this operation:

- The system trusts itself
- You can use the `wlmcert extract` command to make a copy of the system's certificate, which you can then add to other systems' WLM certificate repositories (truststores) to enable secure communications between the current system and those systems

`install -c` *`certificate`*

Adds the named *`certificate`* to the WLM truststore on the current system.

Only root can execute this operation.

The current system can communicate securely with any system for which it has a certificate in its truststore. When using WLM's management of virtual partitions or nPartitions, each partition must have in its truststore the certificate for every other partition with which it is being managed.

`delete -c` *`certificate`*

Removes the named *`certificate`* from the WLM truststore on the current system.

Only root can execute this operation.

`list`

Lists the certificates in the WLM truststore on the current system.

The current system can communicate securely with any system for which it has a certificate in its truststore. When using WLM's management of virtual partitions or nPartitions, each partition must have in its truststore the certificate for every other partition with which it is being managed.

`extract [-d` *`directory`*`]`

Extracts the WLM certificate for the current system, placing it in the named *`directory`*. If a directory is not specified, the certificate is placed in the current directory.

The certificate is named *`host`*.pem, where *`host`* is the name of the current system.

wlmcomd

The `wlmcomd` communications daemon services requests from the HP-UX Workload Manager (WLM) graphical user interface, `wlmgui`, allowing local and remote access to the system.

You must start `wlmcomd` to use `wlmgui`.

Start `wlmcomd` on any system running a WLM daemon (`wlmd`) or a WLM global arbiter daemon (`wlmpard`) that you would like to interact with using `wlmgui`. (`wlmpard` is needed only if you are using WLM's vPar management or its Instant Capacity-based nPartition management. You only need to start one `wlmcomd` even if the system is running both `wlmd` and `wlmpard`.)

You can start `wlmcomd` on the command line or at boot time by editing the file /etc/rc.config.d/wlm, as described in the section "Setting the WLM communications daemon to start automatically at reboot" on page 232.

You can also start `wlmcomd` at boot time by editing the file /etc/rc.config.d/wlm.

The syntax for wlmcomd is:

```
wlmcomd -h
```

```
wlmcomd -V
```

wlmcomd [-p *port*]

```
wlmcomd [-s]
```

```
wlmcomd -k
```

where

`-h`	Displays usage information and exits. This option overrides all other options.
`-V`	Displays version information and exits. This option overrides all options other than `-h`.
`-p` *port*	Changes the port that `wlmcomd` monitors for requests to *port*. When using this option, be sure to configure `wlmgui` to send requests to *port* as well. *port* is a port number greater than 0. If you do not specify a port, `wlmcomd` searches the file /etc/services for the first line with the following format: `hp-wlmcom` *port_number*`/tcp` If such an entry is found, *port_number* is used as the port. If such an entry is not found, the default port of 9692 is used, assuming it is not in /etc/services with protocol tcp. If it is, an error is issued.
`-s`	Causes WLM to run in secure mode—assuming you have distributed security certificates to the systems in question. For more information on using security certificates, see the wlmcert(1M) man page. You can start `wlmcomd` in secure mode at boot time by editing the file /etc/rc.config.d/wlm.
`-k`	Kills any running instance of `wlmcomd`, preventing the servicing of any more requests. However, requests that are already being serviced (via children of `wlmcomd`) are not interrupted. Thus, active monitoring sessions can continue, without service interruption.

NOTE If you are not using secure communications (enabled through the `wlmpard -s` option), use `wlmcomd` only on trusted LANs where you trust all the users: All data exchanged between `wlmcomd` and `wlmgui`, including the user’s password, is transmitted without encryption over the network.

Restrict communications between `wlmcomd` and `wlmgui` to only authorized users to improve security.

Each connection to `wlmcomd` represents a separate process on the system. As such, each connection consumes resources, such as open file descriptors, a process ID, memory, and so forth. A large number of connections could result in denial of service. You can restrict connections by deploying `wlmcomd` on systems behind a firewall that blocks access to the port being used.

wlmcw

The `wlmcw` command starts the WLM configuration wizard. The wizard greatly simplifies the creation of your initial WLM configuration. The wizard is only for creating new configurations. It cannot edit existing configurations. Also, it provides only a subset of the WLM functionality in order to simplify the initial configuration process. After you create a configuration, you can view it to gain a better understanding of how to create more complex configurations manually.

Because the wizard is an X-windows application, be sure to set your `DISPLAY` environment variable before attempting to start the wizard.

`wlmcw` requires Java™ Runtime Environment version 1.4.2.

The syntax for `wlmcw` is:

```
wlmcw [ -s { small | medium | large } ]
```

where

`-s` *`size`*	Changes the font point size used by `wlmcw` to *`size`*. Acceptable values for *`size`* are:	
	`small`	Default system font point size.
	`medium`	Default system font point size plus 4.
	`large`	Default system font point size plus 6.

If `-s` is not specified, the wizard uses the default system font point size plus 2.

After creating your configuration file, called *`configfile`* for example, you can activate it in passive mode as follows:

```
# wlmd -p -a configfile
```

With passive mode, you can see approximately how a particular configuration is going to affect your system—without the configuration actually taking control of your system's resources. To see how the configuration would affect your workloads, use the WLM utility `wlminfo`. Fine-tune the configuration until the `wlminfo` output is as desired, then activate the configuration as follows:

```
# wlmd -a configfile
```

wlmd

The `wlmd` (daemon) command controls the WLM daemon, allowing you to activate configurations as well as stop the daemon. You must log in as root to run `wlmd`, unless you are just using the `-c` option. Valid option combinations are shown below:

```
wlmd -h

wlmd -V

wlmd -C

wlmd [-p] [-s] [-t] [-W] [-i] -A [-l logoption[=n][,...]]

wlmd [-p] [-s] [-t] [-W] [-i] -a configfile [-l logoption[=n][,...]]

wlmd [-W] -c configfile

wlmd -k
```

where:

`-h`	Displays usage information and exits. This option overrides all other options.
`-V`	Displays version information and exits. This option overrides all options other than `-h`.
`-C`	Displays the most recent configuration, appending two commented lines that indicate the origin of the configuration.

`-i` Initializes workload group assignments, ensuring a new configuration’s user records and application records are used when the same workload groups exist in the active and new WLM configurations.

Use this option when the following conditions are met:

- You have workload groups that are in both the active WLM configuration and the new configuration that you want to activate
- You are going to activate the new configuration without first stopping the WLM daemon

Without `-i`, if a currently running process is in a workload group that also exists in the new configuration, the process stays in that group regardless of application records or user records in the new configuration.

The `-i` option is only valid with the `-a` or `-A` options.

`-p` Causes WLM to run in passive mode. In this mode, you can see how a particular configuration is going to approximately affect your system—without the configuration actually taking control of your system.

Using this mode allows you to analyze and fine-tune a configuration.

To see how the configuration would affect your workloads, use the WLM utility `wlminfo`.

The `-p` option must be used with the `-a` or `-A` options.

For information on passive mode, including its limitations, see “Passive mode versus actual WLM management” on page 226.

`-s` Causes WLM to run in secure mode—assuming you have distributed security certificates to the systems in question. For more information on using security certificates, see the wlmcert(1M) man page.

You can start `wlmd` in secure mode at boot time by editing the file /etc/rc.config.d/wlm.

`-t`	Generates comma-separated audit data files. These files are placed in the directory /var/opt/wlm/audit/ and are named wlmd.*monyyyy*, with *monyyyy* representing the month and year the data was gathered. You can access these files directly or through the `wlmaudit` command. (`wlmaudit` has audit data to display only when you use the `-t` option.) For information on `wlmaudit` or on the format of the data files, see the wlmaudit(1M) man page. Be sure to set `wlm_interval` in your WLM configuration file as indicated in the wlmconf(4) man page when you use the `-t` option.
`-A`	Activates a copy of the most recent configuration.
`-W`	Prints warning messages found when parsing the configuration file as error messages. The `-W` option is only valid with the `-A`, `-a`, and `-c` options.
`-a` *`configfile`*	Activates the configuration specified in the file *`configfile`*. If *`configfile`* is not valid, an error message is displayed, and `wlmd` exits. The `-a` option cannot be used with the `-c` option.
`-c` *`configfile`*	Checks the configuration specified in the file *`configfile`* for syntax errors. The current configuration is not affected. This option cannot be used with the `-a` option.
`-l` *`logoption`*	Logs statistics in the file /var/opt/wlm/wlmdstats to assist you in performance tuning. You must use `-A or -a` *`configfile`* with `-l` *`logoption`*. Valid *`logoption`* values are:

`all`	Logs group, host, metric, and SLO statistics every WLM interval.
`all=`*`n`*	Logs group, host, metric, and SLO statistics every *n* WLM intervals.
`group`	Logs group statistics every WLM interval.
`group=`*`n`*	Logs group statistics every *n* WLM intervals.

`host`	Logs host statistics every WLM interval.
`host=`*`n`*	Logs host statistics every *`n`* WLM intervals.
`metric`	Logs metric statistics every WLM interval.
`metric=`*`n`*	Logs metric statistics every *`n`* WLM intervals.
`slo`	Logs SLO statistics every WLM interval.
`slo=`*`n`*	Logs SLO statistics every *`n`* WLM intervals.

When the same *`logoption`* is specified multiple times, the last one specified takes precedence.

Combine multiple values separating them with a comma. The following combination requests all the statistics—with the exception of the host statistics, which have been turned off:

```
-l all,host=0
```

The interval is 60 seconds by default, but can be changed as explained in “Specifying the WLM interval (optional)” on page 203.

For information on setting logging as a default, see “Enabling statistics logging at reboot” on page 234.

Use the `wlminfo` command to review statistics from /var/opt/wlm/wlmdstats. For example, to view SLO data, enter:

```
% wlminfo slo -o
```

In place of `slo`, you can also use `group`, `host`, or `metric`. For more information on `wlminfo`, see the wlminfo(1M) man page.

You can enable automatic trimming of the wlmdstats file by using the `wlmdstats_size_limit` tunable in your WLM configuration. For more information, see the wlmconf(4) man page.

`-k`	Stops (kills) `wlmd`.

NOTE Do not use `prmconfig -r` while `wlmd` is active. Use `wlmd -k` to stop WLM.

wlmgui

The `wlmgui` command invokes the WLM graphical user interface. It allows you to create, modify, and deploy WLM configurations both locally and remotely. In addition, it provides monitoring capabilities. Valid option combinations are:

```
wlmgui
```

```
wlmgui -h
```

```
wlmgui -V
```

where:

`-h`	Displays usage information and exits. This option overrides all other options.
`-V`	Displays version information and exits. This option overrides all options other than `-h`.

`wlmgui` is part of the B8843CA WLM product bundle. If you would like to install `wlmgui` on HP-UX systems where WLM is not installed, you can install just the WLMUtilities bundle from the quarterly AR CD-ROM or from http://software.hp.com. For Microsoft Windows, you can download `wlmgui` from http://software.hp.com.

You can run `wlmgui` on a system where WLM is running or on any remote system with the appropriate JRE (Java Runtime Environment) version installed. (For JRE version requirements, see the DEPENDENCIES section in the wlmgui(1M) man page.)

You must start `wlmcomd` on each system that has a WLM (`wlmd`) or a WLM global arbiter (`wlmpard`) that you want to manage using `wlmgui`. (`wlmpard` is needed only if you are using WLM's vPar management or its Instant Capacity-based nPartition management.)

As a security measure, `wlmcomd` must be explicitly started:

```
/opt/wlm/bin/wlmcomd
```

You can also start `wlmcomd` at boot time by editing the file /etc/rc.config.d/wlm.

Be sure to set your `DISPLAY` environment variable before attempting to start `wlmgui`.

NOTE Do not use `wlmgui` over the internet. Use `wlmgui` only on trusted LANs where you trust all the users: All data exchanged between `wlmcomd` and `wlmgui`, including the user's password, is transmitted without encryption over the network.

Restrict communications between `wlmgui` and `wlmcomd` to only authorized users to improve security.

Also see the WARNINGS section of the wlmgui(1M) man page.

wlminfo

The `wlminfo` command provides various WLM data. You indicate the type of data to display by specifying a command with `wlminfo`. Commands include `slo`, `metric`, `group`, `host`, and `par`. Each command has its own options. Valid option combinations are shown below:

```
wlminfo -h [cmd]

wlminfo -V

wlminfo -i

wlminfo slo [-s slo] [-g grp] [-m met] [-l] [-o] [-v] [-b { 0 | 1 }] [-q]

wlminfo metric [-m met] [-l] [-o] [-b { 0 | 1 }] [-q]

wlminfo group [-g grp] [-l] [-o] [-b { 0 | 1 }] [-q] [-S]

wlminfo host [-l] [-o] [-b { 0 | 1 }] [-q]

wlminfo par [-h host] [-l] [-o] [-b { 0 | 1 }] [-q]

wlminfo proc [-g grp] [-l] [-t secs] [-n cnt] [-p] [-v] [-b { 0 | 1 }]
```

where

`-h [`*`cmd`*`]`

Displays usage information and exits. If you specify *cmd*, the usage information is limited to *cmd* data. This option overrides all other options and commands.

`-V`

Displays version information and exits. This option overrides all commands and any options other than `-h`.

`-i`

Launches `wlminfo` in interactive mode, displaying a graphical user interface (GUI). This option overrides all commands.

Be sure WLM is enabled and your `DISPLAY` environment variable is set before you use this option.

Launching `wlminfo` in this mode requires Java™ Runtime Environment version 1.4.2 or later in /opt/java1.4/jre/bin/java.

`slo [`*`options`*`]`

Displays data about SLOs, including their priorities and associated workload groups. For information on options to this command, see the wlminfo(1M) man page.

`metric [`*`options`*`]`

Displays data about metrics, including their values. For information on options to this command, see the wlminfo(1M) man page.

`group [`*`options`*`]`

Displays data about workload groups, including their CPU allocations. For information on options to this command, see the wlminfo(1M) man page.

`host [`*`options`*`]`

Displays data about the host. For information on options to this command, see the wlminfo(1M) man page.

`par [`*`options`*`]`

Displays data about virtual partitions or nPartitions—if `wlmpard` is running. For information on options to this command, see the wlminfo(1M) man page.

`proc [`*`options`*`]`

Displays data about the most active processes. For information on options to this command, see the wlminfo(1M) man page.

For a description of the `wlminfo` output, see the wlminfo(1M) man page.

wlmpard

The `wlmpard` command controls the HP-UX WLM global arbiter, which governs cross-partition management as well as management of Temporary Instant Capacity and Pay Per Use resources.

Every global arbiter interval (120 seconds by default), the WLM global arbiter checks for CPU requests from the partitions using that global arbiter.

When managing partitions, the arbiter then moves CPUs between partitions, if needed, to better achieve the SLOs specified in the WLM configuration files that are active in the partitions. (Given the physical nature of nPartitions, WLM only simulates CPU movement—as described in Chapter 9, “Managing SLOs across partitions,” on page 283.) The WLM daemon `wlmd` must be running in each partition. Also, the WLM configuration in each partition must use the keyword `primary_host` to reference the name of the host of the partition where the global arbiter is running.

For information on the syntax of the global arbiter configuration file, see wlmparconf(4).

Only root can run `wlmpard`. Valid option combinations are shown below:

```
wlmpard -h

wlmpard -V

wlmpard -C

wlmpard [-p] [-s] [-t] [-n] [-l par[=n]] -A

wlmpard [-p] [-s] [-t] [-n] [-l par[=n]] -a configfile

wlmpard [-n] -c configfile

wlmpard -k
```

where:

`-h`

Displays usage information and exits. This option overrides all other options.

`-V`

Displays version information and exits. This option overrides all options other than `-h`.

`-C`

Displays the most recent global arbiter configuration, appending two commented lines that indicate the origin of the configuration.

`-n`

Prevents the global arbiter from running in daemon mode (that is, forces it to run in the foreground).

`-p`

Causes the global arbiter to run in passive mode. In this mode, you can see how a particular global arbiter configuration is going to approximately affect your system—without the configuration actually taking control of your system. Using this mode allows you to verify and fine-tune a configuration.

To see how the configuration would affect your workloads and virtual partitions, use the WLM utility `wlminfo`.

The `-p` option must be used with the `-a` or `-A` options.

For information on passive mode, including its limitations, see “Passive mode versus actual WLM management” on page 226.

`-s`

Causes the global arbiter to run in secure mode—assuming you have distributed security certificates to the systems in question. For more information on using security certificates, see the wlmcert(1M) man page.

You can start `wlmpard` in secure mode at boot time by editing the file /etc/rc.config.d/wlm.

`-t`

Generates comma-separated audit data files. These files are placed in the directory /var/opt/wlm/audit/ and are named wlmpard.*monyyyy*, with *monyyyy* representing the month and year the data was gathered. You can access these files directly or through the `wlmaudit` command. For information on `wlmaudit` or on the format of the data files, see the wlmaudit(1M) man page.

Be sure to set `interval` in your WLM global arbiter configuration file as indicated in the wlmparconf(4) man page when you use the `-t` option.

`-A`

Activates a copy of the most recent global arbiter configuration.

NOTE WLM activation may take longer than usual when managing nPars.

`-a [`*`configfile`*`]`

Activates the configuration specified in the file *`configfile`*. If *`configfile`* is not valid, an error message is displayed, and `wlmpard` exits.

NOTE WLM activation may take longer than usual when managing nPars.

`-c` *`configfile`*

Checks the configuration specified in the file *`configfile`* for syntax errors. The current configuration is not affected.

`-l par`

Logs statistics in the file /var/opt/wlm/wlmpardstats. You must use `-A or -a` *`configfile`* with `-l par`.

When using `-l`, specifying:

`par`	Logs statistics every global arbiter interval.
`par=`*`n`*	Logs statistics every *n* global arbiter intervals.

Change the interval as explained in "Specifying the global arbiter interval (optional)" on page 293.

For information on setting logging as a default, see "Enabling statistics logging at reboot" on page 234.

You can use the `wlminfo` command to review statistics from /var/opt/wlm/wlmpardstats:

```
% wlminfo par -o
```

For more information on `wlminfo`, see the wlminfo(1M) man page.

You can enable automatic trimming of the wlmpardstats file by using the keyword `wlmpardstats_size_limit` in your WLM global arbiter configuration. For more information, see the wlmparconf(4) man page.

`-k`

Kills any running instance of `wlmpard`. Use this option to shutdown the HP-UX Workload Manager global arbiter.

NOTE WLM shutdown may take longer than usual when managing nPars.

wlmrcvdc

The `wlmrcvdc` utility collects data and forwards it to the WLM daemon. It can collect this data from either of the following rendezvous points:

- Named pipe (FIFO)

  You send data to named pipes using `wlmsend`, discussed in the section “`wlmsend`” on page 398.

  `wlmrcvdc` creates the named pipe, using access permissions of 0600.

- A command’s standard output

For examples showing how to get data to WLM, see “What methods exist for sending data to WLM?” on page 250.

`wlmrcvdc` always creates a named pipe to be fed metric data by executions of `wlmsend`. If a command is also specified, `wlmrcvdc` starts the command in the background and reads metric values from its standard output. If command exits with 0, `wlmrcvdc` will continue running using the named pipe rendezvous point; otherwise, `wlmrcvdc` will exit with error.

NOTE If you use both a named pipe and a command to send the values of a single metric to WLM, be aware that the WLM daemon checks for new metric values once every WLM interval, and that the daemon uses only the last value sent during this interval. This value may have come from the named pipe or the command, depending on which one was updated more recently.

Use `wlmrcvdc` only in WLM configuration files. The syntax is as follows:

`wlmrcvdc [-h] [-V] [-u` *`user`*`] [-g` *`group`*`] [`*`command`*` [`*`args`*`...]]`

NOTE `wlmrcvdc` accepts any unit-less integer or floating-point number. If a value is invalid, it is discarded and a warning is entered in the log file /var/opt/wlm/msglog.

The `wlmrcvdc` name comes from receive (rcv) data collector (dc).

This utility simplifies the process of providing WLM the data it needs to gauge SLO performance, set shares-per-metric allocations, and enable/disable SLOs. By using `wlmrcvdc`, you can avoid using the WLM API, which is discussed in "Sending data from a collector written in C" on page 257.

On the command line, the following options are available:

`-h`	Displays usage information and exits. This option overrides all other options.
`-V`	Displays version information and exits. This option overrides all other options except `-h`.

Otherwise, use `wlmrcvdc` only in a WLM configuration file, as a value for a `coll_argv` keyword:

```
tune metric {
...
   coll_argv = wlmrcvdc options_and_arguments;
...
}
```

For information on `coll_argv`, see "Specifying a data collector (optional)" on page 201.

The *`options_and_arguments`* are based on whether the data source is a named pipe or a command's standard output:

- Named pipe

  `[-u user] [-g group]`

- A command's standard output

  `[-u user] [-g group] command [args...]`

where:

`-u` *`user`*

Sets the user (owner) of the FIFO file to *`user`* (symbolic or numeric) and gives *`user`* write permission on the rendezvous point. If *`command`* is specified, this option also sets the owner of the command process to *`user`*. By default, *`user`* is root.

`-g` *`group`*

Sets the UNIX group of the FIFO file to *`group`* (symbolic or numeric) and changes permissions on the rendezvous point to 0620. If command is specified, this option also sets the UNIX group of the command process to *`group`*. By default, *`group`* is bin.

`command` `[`*`args`*`...]`

Instructs `wlmrcvdc` to start *`command`* with the specified *`args`* arguments in the background and use its standard output as the rendezvous point.

Use the full path to *`command`* unless you are using one of the commands described below.

Although you are able to use other commands, WLM provides a number of commands to be used in this context:

`glance_app`

Retrieves data for applications defined in the GlancePlus file /var/opt/perf/parm (For more information on this command, see the glance_app(1M) man page.)

`glance_gbl`

Retrieves GlancePlus global data (system data) (For more information on this command, see the glance_gbl(1M) man page.)

`glance_prm`

Retrieves general PRM data and PRM data for specific workload groups (also known as PRM groups) (For more information on this command, see the glance_prm(1M) man page.)

`glance_prm_byvg`

Retrieves PRM data regarding logical volumes (For more information on this command, see the glance_prm_byvg(1M) man page.)

`glance_tt`/`glance_tt+`

Retrieve ARM transaction data for applications registered through the ARM API function `arm_init()` (For more information on these commands, see the glance_tt(1M) man page.)

`sg_pkg_active`

Checks on the status of a Serviceguard package (For information on this command, see the sg_pkg_active(1M) man page.)

`time_url_fetch`

Measures the response time for fetching a URL using the Apache `ab` tool. You can use this command with the WLM Apache Toolkit (ApacheTK) to manage your Apache-based workloads (For information on this command, see the time_url_fetch(1M) man page.)

`wlmdurdc`

Helps manage the duration of processes in a workload group (For more information on this command, see the wlmdurdc(1M) man page.)

`wlmoradc`

Produces an SQL value or an execution time (walltime) that results from executing SQL statements against an Oracle instance (For information on this command, see the wlmoradc(1M) man page.)

`wlmwlsdc`

Gathers WebLogic Server Management Bean (MBean) information to track how busy WebLogic instances are (For information on this command, see the wlmwlsdc(1M) man page.)

NOTE stderr for *`command`* is discarded; instead, use syslog(3C) or logger(1) with the daemon facility. Also, be aware that `wlmrcvdc` immediately sends data in the rendezvous point to `wlmd`. However, `wlmd` only uses the last piece of data sent before the WLM interval ends.

For more information on `wlmrcvdc`, see “Sending data with `wlmsend` and `wlmrcvdc`: How it works” on page 270.

wlmsend

The `wlmsend` utility sends data to a rendezvous point for the `wlmrcvdc` utility to collect. Use `wlmsend` on the command line, in a shell script, or in a perl program.

For examples showing how to use `wlmsend` to get data to WLM, see "What methods exist for sending data to WLM?" on page 250.

The syntax is:

```
wlmsend [-h] [-V] [-w wait_time] metric [value]
```

`-h`	Displays usage information then exits. This option overrides all other options.
`-V`	Displays version information then exits. This option overrides all other options except `-h`.
`-w` *`wait_time`*	Waits *`wait_time`* seconds for the rendezvous point to be created before exiting. The default is 5 seconds.
`metric`	Sets the name of the rendezvous point to *`metric`*. This argument is required and must match a *`metric`* string specified in a `goal`, `cpushares`, `condition`, or `exception` statement in the WLM configuration file.
`value`	Designates the metric *`value`* to send to the rendezvous point. This numeric argument must be the last argument on the command line. If *`value`* is not provided, metric values separated by white space will be taken from standard input and sent to the rendezvous point. `wlmrcvdc` accepts any unit-less integer or floating-point number. If a value is invalid, it is still sent to the rendezvous point. However, `wlmsend` prints an error message and exits with status 1.

You can use `wlmsend` to feed piped data to WLM:

```
glance_adviser_command | data_formatter | wlmsend metric
```

NOTE

Be careful of I/O buffering when feeding data to `wlmsend`. For example, the following line works as expected:

```
tail -f logfile | wlmsend job_time
```

However, adding `awk` or other utilities that buffer I/O can result in data being delayed, or not even sent, to `wlmsend`—depending on when the buffer is flushed:

```
tail -f logfile | awk '{print $1}' | wlmsend job_time
```

Always check the file /var/opt/wlm/wlmdstats, created by `wlmd -l metric` as explained in wlmd(1M), to ensure that your data gets to WLM in a timely fashion.

For more information on `wlmsend`, see “Sending data with `wlmsend` and `wlmrcvdc`: How it works” on page 270.

B WLM configuration file syntax overview

This appendix provides a quick reference for the WLM configuration file syntax, indicating the required and optional components. Optional components are enclosed in square brackets (`[]`). Pointers to detailed syntax information are also included.

The syntax information is shown below, followed by an example WLM configuration.

```
[ version = 0; ]
[ primary_host = hostname [ : port_number ]; ]

[
prm {
   groups = { FSS_group_definition | PSET_group_definition } [, ...];
   [ users = user : init_group [alt_group1 alt_group2 ...] [, ...]; ]
   [ apps = group : application [alt_name1 alt_name2...] [, ... ]; ]
   [ scomp = compartment : group [, ...]; ]
   [ disks = group : volume ent [, ...]; ]
   [ gmincpu = group : min [, ...]; ]
   [ gmaxcpu = group : max [, ...]; ]
   [ weight = group : wt [, ...]; ]
   [ gminmem = group : min [, ...]; ]
   [ gmaxmem = group : max [, ...]; ]
   [ memweight = group : wt [, ...]; ]
}
]

slo slo1_name {
   pri = priority;
   [ entity = PRM group group_name; ]
   [ mincpu = lower_bound_request; ]
   [ maxcpu = upper_bound_request; ]
   [ goal = goal_expression; ]
   [ condition = condition_expression; ]
   [ exception = exception_expression; ]
}

slo slo2_name {
   pri = priority;
   [ entity = PRM group group_name; ]
   cpushares =  value { more | total } [ per metric met [ plus offset ] ];
   [ mincpu = lower_bound_request; ]
   [ maxcpu = upper_bound_request; ]
   [ condition = condition_expression; ]
   [ exception = exception_expression; ]
}
```

```
tune [ metric [ slo_name ] ] {
    [ coll_argv = data_collector; ]
    [ wlm_interval = number_of_seconds; ]
    [ absolute_cpu_units = 0_or_1; ]
    [ distribute_excess = 0_or_1;]
    [ transient_groups = 0_or_1;]
    [ coll_stderr = file; ]
    [ cntl_smooth = smoothing_value; ]
    [ cntl_avg = 0_or_1; ]
    [ wlmdstats_size_limit = number_of_megabytes; ]
    [ cntl_kp = proportional_term; ]
    [ cntl_convergence_rate = number_of_shares; ]
    [ cntl_margin = margin_value; ]
    [ cntl_base_previous_req = 0_or_1; ]

}
```

where

FSS_group_definition has the syntax:

```
group : group_ID
```

PSET_group_definition has the syntax:

```
group : PSET
```

For more information on the three structure types, see the following sections:

- "Defining the PRM components" on page 139
- "Defining SLOs" on page 169

  This section explains the two different forms of the `slo` structure shown above.
- "Tuning the metrics and the SLOs" on page 196

Use the following example to better understand the syntax. For an explanation of the file’s components, see “Configuring WLM” on page 127.

```
# Define the workload groups (which are based on PSETs and FSS groups)
prm {
   groups = finance : 2,
            sales : 3,
            marketing : PSET;

   users = jdoe : finance,
           pdoe : sales,
           admin : finance sales marketing;

   apps = finance : /opt/fin_app/count_money,
          sales : /opt/sales/do_ebiz;

   gmincpu = finance : 20,
             sales : 10;

   gmaxcpu = sales : 20;

   gminmem = finance : 30,
             sales : 10,
             marketing : 20;
}

# This is an SLO for the finance group.
slo finance_query {
   pri = 1;
   mincpu = 20;
   maxcpu = 50;
   entity = PRM group finance;
   goal = metric fin_app.query.resp_time < 2.0;
   condition = Mon - Fri;
}
```

```
# This is a stretch goal for the finance query group. If all other CPU
# requests of higher priority SLOs have been met, apply more CPU to
# group finance, so its application runs faster.
slo finance_query_stretch {
   pri = 5;
   mincpu = 20;
   maxcpu = 80;
   entity = PRM group finance;
   goal = metric fin_app.query.resp_time < 1.0;
   condition = Mon - Fri;
}
# On weekends, we do not expect any query transactions, but just in
# case, we will specify a nominal CPU allocation for this application
# for off-hours.
slo finance_query_weekend {
   pri = 1;
   cpushares = 5 total;
   entity = PRM group finance;
   condition = Sat - Sun;
}

# This is an SLO for the Sales group.
slo sales_query {
   pri = 1;
   mincpu = 40;
   maxcpu = 80;
   entity = PRM group sales;
   goal = metric sales_app.resp_time < 10.0;
}

tune {
   wlm_interval = 30;
}

tune fin_app.query.resp_time {
   coll_argv = /opt/fin_app/finance_collector -a 123 -v;
   cntl_kp = 0.8;
}

tune sales_app.resp_time {
   coll_argv = /opt/sales_app/monitor -threshold 2 -freq 30;
   cntl_kp = 2.0;
}
```

C HP-UX command/system call support

Several HP-UX commands and system calls support WLM in assigning users and applications to the proper workload groups. Other commands have options that allow you to use WLM more efficiently. These are standard HP-UX commands and system calls; they are not shipped as part of the WLM or PRM products.

Table C-1 lists HP-UX commands and system calls that support workload groups.

Table C-1 HP-UX commands/system calls that support workload groups

Command/ system call	Supports WLM as follows
`at`	Places the scheduled job in the user's initial workload group. If the user does not have an initial group, the job is placed in the user default group, `OTHERS` (workload group ID 1).
`cron`	Places the scheduled job in the user's initial workload group. If the user does not have an initial group, the job is placed in the user default group, `OTHERS` (workload group ID 1).
`login`	Places the login process in the user's initial workload group. If the user does not have an initial group, the login process is placed in the user default group, `OTHERS` (workload group ID 1).
`exec()`	Process remains in its current workload group.
`fork()`	Starts child process in the parent's workload group.
`pstat()`	Returns a process's workload group ID.

Table C-2 describes HP-UX commands that have options for WLM.

Table C-2 **WLM options in HP-UX commands**

Command	Option	Description
`acctcom`	`-P`	Displays the workload group ID (PRMID) of each process.
`acctcom`	`-R` *`group`*	Displays only processes belonging to the workload group given by *`group`*, which is specified by workload group name or workload group ID.
`id`	`-P`	Displays the workload group ID (PRMID) and name of the invoking user’s initial group.
`ps`	`-P`	Adds a column named `PRMID` to the `ps` output that gives the workload group name associated with each process.
`ps`	`-R` *`group_list`*	Displays only the processes that belong to workload groups specified in *`group_list`*. *`group_list`* must consist of workload group IDs (PRMIDs) or workload group names. Groups must be separated by commas; no spaces are allowed.

For more information on these commands, see their man pages.

D Integration with other products

WLM integrates with various other products to provide greater functionality. Currently, these other products are:

- Apache web server
- nPartitions
- Security Containment
- OpenView Performance Agent for UNIX / OpenView Performance Manager for UNIX
- Oracle databases
- Processor sets
- SAS® Software
- HP Systems Insight Manager and Servicecontrol Manager
- Serviceguard
- SNMP
- Temporary Instant Capacity / Pay Per Use
- Virtual partitions
- BEA WebLogic Server

The integration with these products is described below.

Integrating with Apache

WLM can help you manage and prioritize Apache-based workloads through the use of the WLM Apache Toolkit (ApacheTK), which is part of the freely available product WLM Toolkits (WLMTK). WLM can be used with Apache processes, Tomcat, CGI scripts, and related tools using the HP-UX Apache-based Web Server.

Why use ApacheTK?

Assume your enterprise (or corporate) applications use Apache as a front end to dispatch work to workloads running Java software, CGI scripts, and other such processes. You can then use WLM and ApacheTK to manage these processes in a manner that reflects the business priorities they support.

Tools in ApacheTK

ApacheTK includes a white paper and example configuration files that demonstrate resource separation of Apache-based workloads. It also includes the simple data collector `time_url_fetch`, which uses the Apache benchmark tool `ab` to time URL response times. Also, ApacheTK provides the script `wlm_watch.cgi`, which provides a web-based view of the standard `prmmonitor`, `prmlist`, and other WLM and PRM command-line tools. These are view-only reports—no modifications are allowed from the CGI script.

How do I get started with ApacheTK

The best way to use ApacheTK is to read the white paper *Using HP-UX Workload Manager with Apache* available from /opt/wlm/toolkits/apache/doc/apache_wlm_howto.html and on the web at http://www.hp.com/go/wlm. The paper guides you through the steps and tools needed to have WLM:

- Separate Apache from Oracle database instances
- Separate Apache from batch work
- Isolate a resource-intensive CGI workload
- Isolate a resource-intensive servlet workload
- Separate all Apache Tomcat workloads from other Apache workloads
- Separate two departments' applications using two Apache instances
- Separate module-based workloads with two Apache instances
- Manage Apache CPU allocation by performance goal

For more ApacheTK information

If you would like to learn more about ApacheTK and Apache, see:

- /opt/wlm/toolkits/apache/doc/apache_wlm_howto.html
- wlmtk(5) man page
- time_url_fetch(1M) man page
- wlm_watch(1M) man page
- *HP-UX Workload Manager Toolkits User's Guide* (opt/wlm/toolkits/doc/WLMTKug.pdf)
- *HP-UX Workload Manager Toolkits A.01.08 Release Notes* (/opt/wlm/toolkits/doc/Rel_Notes)
- http://www.hp.com/go/webservers, then select the Apache link

Integrating with nPartitions (nPars)

You can run WLM within and across nPartitions. For systems with partitions using Instant Capacity CPUs, WLM provides a global arbiter, `wlmpard`, that can take input from the WLM instances on the individual partitions. The global arbiter then “moves” CPUs between partitions, if needed, to better achieve the SLOs specified in the WLM configuration files that are active in the partitions. (This movement is achieved by deactivating a CPU in one nPar, then activating a CPU in another nPar. The total number of active CPUs has remained constant—avoiding a charge for an additional CPU. For more information, see the wlmpard(1M) and wlmparconf(4) man pages. (Instant Capacity was formerly known as iCOD.)

For more information on configuring this integration, including the nesting of virtual partitions within nPartitions, see Chapter 9, “Managing SLOs across partitions,” on page 283.

Integrating with Security Containment (to form Secure Resource Partitions)

The HP-UX feature Security Containment provides file and process isolation and is available starting with HP-UX 11i v2. Combining that isolation with WLM's workload groups, you can form Secure Resource Partitions, which give your workload groups both isolation and automatic resource allocation.

To integrate the two products:

Step 1. Create a configuration for Security Containment

The `srpgen` utility, located in /opt/prm/bin/, automates the creation of a configuration for Security Containment and PRM. For information, see the srpgen(1) man page.

For information on Security Containment, see the compartments(5) man page.

Step 2. Activate the Security Containment configuration

Use the `setrules` command to activate your configuration. For information, see the setrules(1M) man page.

Step 3. Create your WLM configuration

Use the /opt/wlm/bin/wlmprmconf utility to convert the PRM configuration created in Step 1 to a WLM configuration.

Step 4. Activate your WLM configuration

Use the `wlmd` command to activate your WLM configuration.

Integrating with OpenView Performance Agent (OVPA) / OpenView Performance Manager (OVPM)

You can treat your workload groups as applications and then track their application metrics in OpenView Performance Agent for UNIX as well as in OpenView Performance Manager for UNIX.

NOTE

NOTE: If you complete the procedure below, OVPA/OVPM will track application metrics only for your workload groups; applications defined in the parm file will no longer be tracked. GlancePlus, however, will still track metrics for both workload groups and applications defined in your parm file.

To track application metrics for your workload groups:

1. Edit /var/opt/perf/parm

   Edit your /var/opt/perf/parm file so that the “log” line includes “application=prm” (without the quotes). For example:

   ```
   log global application=prm process dev=disk,lvm
   transaction
   ```

2. Restart the agent

   With WLM running, execute the following command:

   ```
   % mwa restart scope
   ```

NOTE The WLM workload groups must be enabled at the time the scopeux collector is restarted by the `mwa restart scope` command. If WLM is not running, or `transient_groups` is set to 1 in your WLM configuration, data for some—or all—workload groups may be absent from OpenView graphs and reports. Also, it may affect alarms defined in /var/opt/perf/alarmdefs.

For more information, see http://openview.hp.com.

Now all the application metrics will be in terms of workload (PRM) groups. That is, your workload groups will be “applications” for the purposes of tracking metrics.

Integrating with Oracle® databases

WLM allows you to place Oracle database instances and other applications in their own WLM workload groups. With the instances and applications separated in this manner, WLM can then manage the performance of each instance and application through prioritized SLOs. These SLOs typically attempt to ensure:

- A consistent level of performance
- The needs of critical instances are met—even during peak demand
- Resources are allocated based on time of day, system events, or application and database metrics

HP has developed a toolkit, HP-UX WLM Oracle Database Toolkit (ODBTK), to simplify getting metrics on Oracle database instances into WLM. This toolkit, which is included with WLM as part of the WLM Toolkits product, works with:

Oracle 8.0.x, Oracle 8.1.5, Oracle 8.1.6, Oracle 8.1.7, Oracle 9.0.1

For information on toolkit requirements, see the *HP-UX Workload Manager Toolkit User's Guide*, available at /opt/wlm/toolkits/doc/WLMTKug.pdf

Why use Oracle database metrics with WLM?

The key benefit of using Oracle database metrics with WLM is that you can use the database metrics to manage the performance of your instances. You specify SLOs for the instances based on the metrics.

For example, with these metrics you can:

- Keep response times for your transactions below a given level by setting response-time SLOs
- Increase an instance’s available CPU when a particular user connects to the instance
- Increase an instance’s available CPU when more than *n* users are connected
- Increase an instance’s available CPU when a particular job is active
- Give an instance *n* CPU shares for each process in the instance
- Give an instance *n* CPU shares for each user connection to the instance

For examples showing how to set up these types of SLOs, see the files with names ending in “.wlm” in the directory /opt/wlm/toolkits/oracle/config/.

Tools in the HP-UX WLM Oracle Database Toolkit (ODBTK)

This toolkit includes two tools:

`wlmoradc`

`wlmoradc` is a data collector for Workload Manager and is designed to provide an easy building block for Oracle instance management with `wlmd`.

It takes one or more SQL statements and uses the Oracle tool SQL*Plus to connect to an Oracle instance and execute the statements, returning either the raw value returned from the SQL or the elapsed execution time. The results are sent to stdout for human viewing, logging, or most often, for use by `wlmd` by means of the `wlmrcvdc` or `wlmsend` utilities.

`smooth`

`smooth` takes a stream of newline-delimited numbers and outputs a stream of numbers that are a running average of the last *n* values, where *n* is a value that can be set on the command line. The principal use for the `smooth` utility is to remove short spikes or dips in data collector output used with WLM, but can be applied to any stream of floating-point numbers.

Although not part of ODBTK, the functionality provided by the `cpushares` keyword in the WLM configuration file fits quite nicely with the toolkit.

NOTE The `cpushares` keyword is available starting with WLM Version A.01.02.

A CPU share is a portion of the CPU. The `cpushares` keyword allows shares-per-metric allocations. This enables you to create goal-based SLOs in WLM of the form "*x* percent of the CPU for each metric *y*," such as three CPU shares for each process in a workload group. (This type of allocation can also be thought of as a parametric allocation. It is calculated directly from the metric and is thus a function of the metric. This type of function is a parametric equation.)

What metrics are available?

The following types of database metrics are available:

- Time elapsed while SQL code executes
- Value returned by executed SQL code

  This value can be information from Oracle V$ tables. These tables provide dynamic performance data for Oracle instances and allow the Oracle database administrator to see current performance information.

How do I get started with ODBTK?

The toolkit comes with numerous example files that you can copy and modify to fit your needs. These files include WLM configuration files and `wlmoradc` configuration files, which include SQL.

The files are available in /opt/wlm/toolkits/oracle/config/. Table D-1 describes the WLM example configuration files.

Table D-1 **ODBTK's example WLM configuration files described**

WLM configuration file	Purpose
alpha_shares_per_user.wlm	Demonstrate how to customize the supplied file shares_per_user.wlm for a slightly different set of instances
batchuser_boost.wlm	Demonstrate a conditional allocation, with the allocation enforced when a certain user connects
manual_payroll_boost.wlm	Demonstrate a conditional allocation, with the allocation enforced when a certain application becomes active; also, demonstrate how to feed "external" metrics to WLM using `wlmsend`
shares_per_process.wlm	Demonstrate an allocation where each instance gets a certain number of CPU shares per process
shares_per_user.wlm	Demonstrate an allocation where each instance gets a certain number of CPU shares per user connection
timed_select_scott.wlm	Demonstrate a response-time goal using a SCOTT/TIGER table

Table D-1 ODBTK’s example WLM configuration files described

WLM configuration file	Purpose
timed_sys_table.wlm	Demonstrate a response-time goal using V$ Oracle system tables
user_cnt_boost.wlm	Demonstrate a conditional allocation, with a new allocation enforced when more than a set number of users connect

Table D-2 describes the `wlmoradc` configuration files. These files can be used to set database environment variables for `wlmoradc` to use. You can also place the SQL statements for `wlmoradc` to execute in these files.

Table D-2 `wlmoradc` example configuration files described

`wlmoradc` configuration file	Purpose
process_cnt.oradc	Report number of user processes using an instance
select_scott_resptime.oradc	Demonstrate a timed proxy transaction with the SCOTT/TIGER tables
sys_table_resptime.oradc	Demonstrate a timed proxy transaction against V$ Oracle system tables
user_cnt.oradc	Report number of users connected to an instance

ODBTK examples

As noted in the previous section, ODBTK comes with many useful examples in the directory /opt/wlm/toolkits/oracle/config/. We will look at parts of those files here.

The first example draws from /opt/wlm/toolkits/oracle/config/shares_per_user.wlm. The `slo` structure gives workload group `Ora_grp_1` three CPU shares for each user connected to the associated Oracle instance.

```
slo Ora_1_slo {
        pri = 1;
        mincpu = 5;
        maxcpu = 90;
        entity = PRM group Ora_grp_1;
        cpushares = 3 total per metric oracle.instance1.user_cnt;
}
```

Looking at the `tune` structure in the same file, we see that the metric oracle.instance1.user_cnt is provided by `wlmoradc`, which uses the SQL in the configuration file user_cnt.oradc to get the user count.

```
tune oracle.instance1.user_cnt {
    coll_argv =
        wlmrcvdc
          wlmoradc
            --configfile /opt/wlm/toolkits/oracle/config/user_cnt.oradc
            --home /oracle/app/oracle/product/8.1.5
            --instance instance1
    ;
}
```

The next example is from /opt/wlm/toolkits/oracle/config/user_cnt_boost.wlm. The second SLO, Ora_1_slo, provides a minimum 20 share allocation. The first SLO, Ora_1_slo_boost, becomes active and boosts the allocation to 40 shares if the metric oracle.instance1.user_cnt is 11 or more.

```
slo Ora_1_slo_boost {
        pri = 1;
        cpushares = 40 total;
        entity = PRM group Ora_grp_1;
        condition = metric oracle.instance1.user_cnt > 10;
}

slo Ora_1_slo {
        pri = 3;
        cpushares = 20 total;
        entity = PRM group Ora_grp_1;
}
```

Again, the `tune` structure shows `wlmoradc` providing the metric to WLM. This `tune` structure also uses the `smooth` tool, which calculates a running average of the last five user counts, smoothing out extreme values when the user count is volatile.

```
tune oracle.instance1.user_cnt {
   coll_argv =
       wlmrcvdc
           smooth -c 5
           wlmoradc
             --configfile /opt/wlm/toolkits/oracle/config/user_cnt.oradc
             --home /oracle/app/oracle/product/8.1.5
             --instance instance1
   ;
}
```

For more ODBTK information

If you would like to learn more about ODBTK, see:

- wlmtk(5) man page
- wlmoradc(1M) man page
- smooth(1M) man page
- *HP-UX Workload Manager Toolkits User's Guide* (opt/wlm/toolkits/doc/WLMTKug.pdf)
- *HP-UX Workload Manager Toolkits A.01.08 Release Notes* (/opt/wlm/toolkits/doc/Rel_Notes)

Integrating with processor sets (PSETs)

Processor sets allow you to group processors together, dedicating those CPUs to certain applications. WLM can automatically adjust the number of CPUs in a PSET-based workload group in response to SLO performance. Combining PSETs and WLM, you can dedicate CPU resources to a group without fear of the group's needing additional CPUs when activity peaks or concern that the group, when less busy, has resources that other groups could be using.

For information on configuring this integration, see the section "Specifying workload groups (optional)" on page 149.

Integrating with SAS® software

WLM provides integration with SAS software through the WLM Toolkit for Base SAS Software (SASTK). This toolkit, which is part of the freely available WLM Toolkits, or WLMTK, product relies upon the WLM Duration Management Toolkit, DMTK.

WLM and DMTK help control CPU resources by granting a critical job only the amount of CPU needed to complete within a certain timeframe. Because the business-critical job is being duration managed, the extra compute resource on the server can be made available to other users or jobs without affecting the managed business-critical job. This translates to more efficient use of existing compute resources. This feature, known as duration management, is useful in Base SAS environments and many other environments.

Furthermore, even without DMTK, WLM can grant computing resources upon demand by providing an “express lane” to ensure that your most important jobs are given top priority. Express lanes can be particularly beneficial in a Base SAS environment where any user has the ability to launch a job that can, regardless of its business priority, significantly affect the performance of other jobs.

Why use SASTK?

Combining SASTK with DMTK, you can control your SAS jobs using:

- Job duration providing jobs just the right amount of CPU—not too much or too little—to finish within user-specified time ranges
- Examples that show how express lanes can be used to quickly complete urgent jobs

DMTK does not reduce the amount of CPU time an application must have to complete; it merely attempts to regulate the application’s access to CPU resources. For example, if an application takes one hour to complete when using 100% of the CPU, DMTK cannot make its duration less than one hour.

Tools in SASTK

SASTK provides a macro:

`hp_wlmtk_goals_report`

This is a SAS macro that is useful in instrumenting SAS jobs to:

- Get profile data indicating elapsed time for a job
- Inform `wlmdurdc` of a job's percent completed

How do I get started with SASTK?

To get ideas on how to implement SASTK in your environment, look at the example configuration files that come with SASTK. These files are in /opt/wlm/toolkits/sas/config/. Also, see how the example scripts in /opt/wlm/toolkits/sas/examples/ may help you.

For more SASTK information

If you would like to learn more about SASTK, see:

- wlmtk(5) man page
- wlmdurdc(1M) man page
- hp_wlmtk_goals_report(1M) man page
- *HP-UX Workload Manager Toolkits User's Guide* (opt/wlm/toolkits/doc/WLMTKug.pdf)
- *HP-UX Workload Manager Toolkits A.01.08 Release Notes* (/opt/wlm/toolkits/doc/Rel_Notes)

Integrating with HP Systems Insight Manager (SIM) and Servicecontrol Manager (SCM)

This section discusses how you can use WLM with the HP products Systems Insight Manager and Servicecontrol Manager. These products both provide a single point of administration for multiple HP-UX systems. Systems Insight Manager is the newer product. The WLM integration with these products allows system administrators at a SIM / SCM Central Management Server (CMS) to perform the following activities on nodes in the SIM / SCM cluster that have WLM installed:

- Workload Manager Console
- Activate WLM Configuration
- Enable WLM
- Disable WLM
- Start WLM
- Stop WLM
- Reconfigure WLM
- Distribute WLM configuration files to the selected nodes
- Retrieve currently active WLM configuration files from the nodes
- Check the syntax of WLM configuration files, on either the CMS or the selected nodes
- View, rotate, and truncate WLM log files

What WLM tasks are available through SIM / SCM?

The sections below are named for the HP-UX WLM tools available through SIM and SCM.

Workload Manager Console

Launch the WLM GUI after selecting the nodes to manage and setting your `DISPLAY` variable.

Activate WLM Configuration

Gracefully shuts down WLM on the selected nodes, then restarts it, activating the WLM configuration file at /var/opt/wlm/SCM-managed.wlm.

Run this tool after placing a configuration on the node using the tool Install WLM Configuration.

Enable WLM

Sets the variable `WLM_ENABLE` to 1 in the file /etc/rc.config.d/wlm on the selected nodes. This allows WLM to start automatically whenever the system is booted or "`/sbin/init.d/wlm start`" is run.

Disable WLM

Sets the variable `WLM_ENABLE` to 0 in the file /etc/rc.config.d/wlm on the selected nodes. This prevents WLM from starting automatically whenever the system is booted or "`/sbin/init.d/wlm start`" is run.

Install WLM Configuration

Prompts you for a configuration file, then copies the file to the selected nodes, placing it in /var/opt/wlm/SCM-managed.wlm. Run the tool Activate WLM Configuration to point WLM to the new configuration.

Restart WLM

Stops and restarts WLM by invoking the command

```
/sbin/init.d/wlm stop
```

followed by the

```
/sbin/init.d/wlm start
```

command. When restarted, WLM uses:

- The configuration *`configfile`* in the file /etc/rc.config.d/wlm in the line

  `WLM_STARTUP_SLOFILE="`*`configfile`*`"`

- The last activated configuration file—if `WLM_STARTUP_SLOFILE` is not specified

Retrieve WLM Configuration

Prompts you for a destination directory on the CMS, then places the currently activated configuration files from the selected nodes in the specified directory. These files are named `$HOST`.wlm, with `$HOST` replaced by the names of the nodes the files were retrieved from.

Rotate WLM Log Files

Rotates the /var/opt/wlm/msgslog files on the selected nodes by copying the current log file to a date-stamped log file, then truncating the current log file:

```
cp /var/opt/wlm/msglog /var/opt/wlm/msglog.`date +%Y%m%d-%H%M`
cp /dev/null /var/opt/wlm/msglog
```

The previous log files are then available as msglog.YYYYMMDD-HHMM.

Rotate Statistics Log Files

Rotates the /var/opt/wlm/wlmdstats files on the selected nodes by copying the current statistics log file to a date-stamped statistics log file, then truncating the current statistics log file:

```
cp /var/opt/wlm/msglog /var/opt/wlm/wlmdstats.`date +%Y%m%d-%H%M`
cp /dev/null /var/opt/wlm/wlmdstats
```

The previous statistics log files are then available as wlmdstats.YYYYMMDD-HHMM.

Start WLM

Starts WLM using the command `/sbin/init.d/wlm start`.

Stop WLM

Stops WLM using the command `/sbin/init.d/wlm stop`.

Syntax Check Configuration

Prompts you for a configuration file, then copies the file to the selected nodes and checks the files' syntax. Output from the syntax checks is sent back to the CMS.

Checking the syntax after distributing the files allows you to verify that the applications, data collectors, and any users in the configuration file actually exist on the selected nodes. However, if you have other types of syntax issues, you will have to fix the issues in each of the distributed files—or fix them once in the CMS file and redistribute.

Syntax Check on CMS

Prompts you for a configuration file, then checks the file's syntax on the CMS.

Running this command before distributing a configuration file to the selected nodes allows you to fix syntax errors in one file rather than in all the distributed copies. Be aware, however, that this check will flag missing data collectors, applications, and users that may be on the selected nodes, but are not on the CMS.

Truncate Log Files

Truncates the /var/opt/wlm/msgslog files on the selected nodes by running the command:

```
cp /dev/null /var/opt/wlm/msglog
```

Truncate Statistics Log Files

Truncates the /var/opt/wlm/wlmdstats statistics log files on the selected nodes by running the command:

```
cp /dev/null /var/opt/wlm/wlmdstats
```

View WLM Log Files

Displays the /var/opt/wlm/msglog file from each selected node by running the command:

```
xterm -e $PAGER /var/opt/wlm/msglog
```

View Statistics Log Files

Displays the /var/opt/wlm/wlmdstats statistics log file from each selected node by running the command:

```
xterm -e $PAGER /var/opt/wlm/wlmdstats
```

Accessing the WLM tools

NOTE Using WLM with SIM / SCM requires that you install the fileset CMSConfig.WLMB-CMS-Tools on the SIM / SCM CMS. This fileset is available from the depot /var/opt/mx/depot11 on the host where WLM has been installed.

Also be aware: If you are installing this fileset on a CMS that has WLM installed, the installation will fail if the fileset is not compatible with (does not have the same revision string as) the installed WLM.

How you access the tools depends on whether your are using SIM or SCM. (With SCM, the version also determines how you access the tools.)

With SIM:

Step 1. Start SIM in a web browser.

Step 2. Select the menu item corresponding to your desired WLM tool:

Optimize -> Workload Manager -> *Desired Tool*

Using a SCM version prior to 3.0, access the WLM tools as follows:

Step 1. Start SCM.

Step 2. Double-click the Tools icon.

Step 3. Double-click the Workload Management icon.

Step 4. Double-click the desired tool.

For SCM version 3.0 and later, SCM has a web interface. To access the WLM tools:

Step 1. Start SCM in a web browser.

Step 2. Select a node on which to run the WLM tools.

Step 3. In the left portion of the browser window, select the Tools tab if it is not already selected.

Step 4. Select the “Workload Management” item in the Tools list.

Step 5. Select the WLM tool you would like to run.

For more SCM information

For SCM documentation, visit:

http://docs.hp.com/hpux/netsys/

Integrating with Serviceguard

This section discusses how you can better use WLM with Serviceguard. The optional HP product Serviceguard provides users and applications with a high availability environment. Serviceguard makes this environment possible by moving applications from one server to another when the original server or application session is unable to complete the desired jobs.

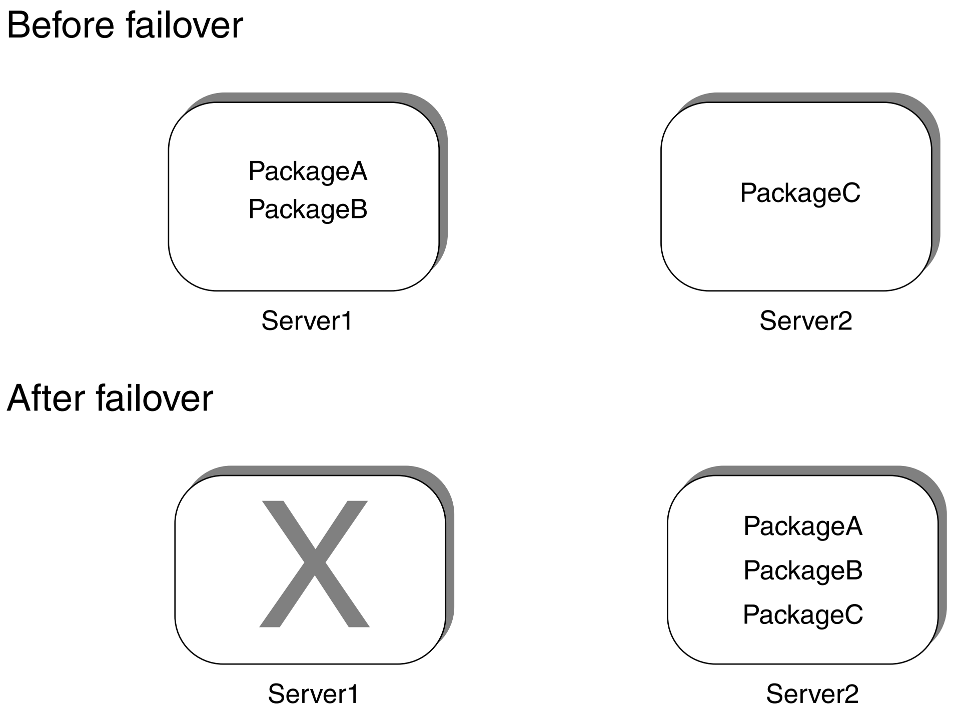
By using the same WLM configuration on each server in a Serviceguard cluster, you can better ensure that your workloads achieve their SLOs—regardless of where they run in the cluster.

The WLM configuration file allows you to inform `wlmd` whether a particular SLO is active on the current server. System resources assigned to the inactive SLOs' associated workload groups become available to workload groups with active SLOs.

An active SLO is an SLO that WLM is trying to enforce. All SLOs are active by default. An SLO's time as active can be limited through `condition` and `exception` statements in the `slo` structure in the configuration file. These statements allow you to make SLOs active based on time or on a metric's value. You can use a metric's value to indicate whether a Serviceguard package is active.

Consider the example in Figure D-1. There are two servers in a Serviceguard cluster. One server runs two packages, while the second server runs one package. In this case, all packages are able to comfortably meet their SLOs. However, Server1 crashes in the middle of the night. Its packages then fail over to Server2. Now that Server2 has three packages, there is concern about the packages meeting their SLOs. When packages failover, WLM realizes—because of `condition` statements in the configuration file—that new packages have appeared. Based on the packages' priorities, WLM allocates resources to the packages so they can potentially still achieve their SLOs.

Figure D-1 WLM integration with Serviceguard



Steps for integration

To integrate WLM with Serviceguard:

Step 1. Install WLM on each node in your Serviceguard cluster.

Step 2. Edit a single WLM configuration file to handle all the Serviceguard packages in the cluster. The example below assumes there are two packages: pkgA and pkgB. This configuration file must:

a. Place all the packages' applications in workload groups in the `prm` structure:

```
prm {
   ...
   groups = pkgA:2, pkgB:3;
   apps = pkgA:/opt/dbase/bin/sales_dbase,
          pkgB:/opt/dbase/bin/finance_dbase;
    ...
}
```

b. Set up `tune` structures to report a status metric for each package:

```
tune pkgA_active {
   coll_argv = wlmrcvdc sg_pkg_active;
}

tune pkgB_active {
   coll_argv = wlmrcvdc sg_pkg_active;
}
```

These `tune` structures set the metric values `pkgA_active` and `pkgB_active` to 1 if their packages are indeed active on the current node. The key component is the `sg_pkg_active` data collector, which is included with WLM. For more information on this data collector, see sg_pkg_active(1M).

Rather than creating `tune` structures for each package, you can set up a global `tune` structure that takes in package status data from any running `sg_pkg_active`:

```
tune {
   coll_argv = wlmrcvdc sg_pkg_active;
}
```

c. Set up `slo` structures for each status metric, with the SLO being active when the package is active:

```
slo pkgA_slo {
   ...
   condition = metric pkgA_active;
   ...
}

slo pkgB_slo {
   ...
   condition = metric pkgB_active;
   ...
}
```

Recall that the `condition` statement governs when an SLO is active: If an SLO’s `condition` expression is true, the SLO is active. In these cases, the expressions “`metric pkgA_active`” and “`metric pkgB_active`” are true when the `pkgA_active` and `pkgB_active` metrics are 1.

NOTE Workload groups that have only inactive SLOs continue to consume system resources. To prevent this needless consumption, use the `transient_groups` tunable, which is described in the next substep.

d. (Optional) Use `transient_groups` in a global `tune` structure to temporarily remove workload groups with no active SLOs

By default, if a package’s workload group has no active SLOs, WLM reduces its resource shares. Such a workload group receives 1% of the total CPU (for FSS groups) or one CPU (for PSET-based groups), unless it has a `gmincpu` value requesting more. Similarly, if you are using WLM’s memory management, the workload group with no active SLOs receives 1% of the memory, unless the group has a `gminmem` value requesting more.

If you would prefer these groups to go away temporarily (as long as they have no active SLOs) and consume no CPU or memory resources, set the `transient_groups` tunable to 1 in a global `tune` structure:

```
tune {
   transient_groups = 1;
}
```

With the `transient_groups` keyword set to 1: FSS groups with no active SLOs are deleted and therefore use no resources; the minimum CPU allocation for PSET groups becomes 0.

If a workload group has been temporarily removed, its `gmincpu` and `gminmem` values (if any) are ignored.

NOTE The `OTHERS` workload group is never removed—regardless of how many active SLOs it has.

Step 3. Validate the syntax of the configuration file:

```
# /opt/wlm/bin/wlmd -c configfile
```

Correct any errors that are found.

Step 4. Distribute the WLM configuration file to all the nodes in the cluster.

Step 5. Activate WLM with your configuration file on all the nodes:

```
# /opt/wlm/bin/wlmd -a configfile
```

For information on Serviceguard, see the manual *Managing MC/ServiceGuard*, available at http://docs.hp.com.

For information on the `condition` keyword, see "Specifying when the SLO is active (optional)" on page 192.

For more information on the `sg_pkg_active` data collector, see sg_pkg_active(1M).

Integrating with the HP-UX SNMP agent

WLM provides integration with the HP-UX SNMP agent through the WLM SNMP Toolkit (SNMPTK). This toolkit is part of the freely available WLM Toolkits, or WLMTK.

SNMPTK provides a WLM data collector called `snmpdc`, which fetches values from an SNMP agent so you can use them as metrics in your WLM configuration.

Why use SNMPTK?

SNMPTK allows easy access to SNMP agent metrics that you can use in your WLM configuration to:

- Drive SLO goals
- Set up shares-per-metric allocations
- Enable and disable SLOs

Sources of metrics include:

- PRM data, available starting at the OID `.1.3.6.1.4.1.11.5.4.2.1` (hp.hpSysMgt.hpUXSysMgt.hpPRM.prmReadOnly)
- Pay Per Use data, available starting at the OID `.1.3.6.1.4.1.11.11.1` (hp.hpUtility.cpuppu)

Tools in SNMPTK

SNMPTK provides the `snmpdc` data collector for pulling data from the SNMP agent.

How do I get started with SNMPTK?

To get ideas on how to implement SNMPTK in your environment, look at the example configuration files that come with SNMPTK. These files are in /opt/wlm/toolkits/snmp/config/.

For more SNMPTK information

If you would like to learn more about SNMPTK, see:

- wlmtk(5) man page
- snmpdc(1M) man page
- *HP-UX Workload Manager Toolkits A.01.08 Release Notes* (/opt/wlm/toolkits/doc/Rel_Notes)

Integrating with Temporary Instant Capacity/ Pay Per Use

This section discusses the use of WLM with Temporary Instant Capacity (v6 or v7) or Pay Per Use (v4). (Instant Capacity was formerly known as iCOD.) In particular, with WLM managing the use of these resources, you can ensure your workloads use only the amount of CPU needed for them to meet their SLOs.

Temporary Instant Capacity activates capacity in a temporary "calling-card fashion" in 30-CPU-day increments. A CPU day equals 24 hours for one CPU. With this option, you can activate and deactivate processors. One Temporary Instant Capacity license is applied to a system so you can turn on and off any number of Instant Capacity CPUs on your system. With Temporary Instant Capacity on the system, any number of Instant Capacity CPUs can be activated as long as the license has a positive balance.

Also, HP offers a Pay Per Use (PPU) feature that provides the capacity to support peak anticipated demand, but with payment for the HP server based on monitored usage of that capacity. On systems with a PPU v4, this capacity can be increased or decreased by whole CPUs—as needed, with billing determined by the number of active CPUs. Starting with PPU v5, all CPUs on a PPU system are active and billing is based on your percentage usage of those CPUs.

If you have WLM on either a Temporary Instant Capacity system (using v6 or v7) or a PPU system (using v4), you can configure WLM to minimize the costs of using these resources by optimizing the amount of time the resources are used to meet the needs of your workloads.

To take advantage of this optimization, use the `utilitypri` keyword in your global arbiter configuration as explained in the wlmparconf(4) and wlmpard(1M) man pages.

NOTE Specifying this keyword ensures WLM maintains compliance with your license for Temporary Instant Capacity: When the license is exhausted, WLM will no longer attempt to use the temporary resources.

For more Temporary Instant Capacity / Pay Per Use information

Fore more information on Temporary Instant Capacity, visit:

- http://search.hp.com/, and search for “Temporary Instant Capacity”

Fore more information on Pay Per Use, visit:

- http://www.hp.com/go/utility
- http://docs.hp.com, and search for “Pay Per Use”

Integrating with virtual partitions

You can run WLM within and across virtual partitions. For systems with partitions, WLM provides a global arbiter, `wlmpard`, that can take input from the WLM instances on the individual partitions. The global arbiter then moves CPUs between partitions, if needed, to better achieve the SLOs specified in the WLM configuration files that are active in the partitions.

For more information on configuring this integration, including the nesting of virtual partitions within nPartitions, see Chapter 9, “Managing SLOs across partitions,” on page 283.

Integrating with BEA WebLogic Server

WLM can help you manage and prioritize WebLogic-based workloads through the use of the WLM BEA WebLogic Toolkit (WebLogicTK), which is part of the freely available product WLM Toolkits (WLMTK).

Why use WebLogicTK?

Using WLM with WebLogic you can move CPUs to or from WebLogic Server instances as needed to maintain acceptable performance. By managing the instances' CPU resources, the instances will tend to use less net CPU resources over time. You can then use the additional CPU resources for other computing tasks.

As indicated above, WLM and WebLogicTK control CPU allocation to individual WebLogic instances. However, the latest version of the paper *Using HP-UX Workload Manager with BEA WebLogic* expands the methods for controlling instances to control WebLogic Server clusters.

Tools in WebLogicTK

WebLogicTK includes a white paper and example configuration files that demonstrate resource separation of WebLogic-based workloads. It also includes the data collector `wlmwlsdc`, which tracks metrics for WebLogic instances. Also, WebLogicTK provides the `expsmooth` utility for smoothing values from WLM data collectors by computing a running average in which the importance of old values diminishes (decays) over time.

How do I get started with WebLogicTK

The best way to use WebLogicTK is to read the white paper *Using HP-UX Workload Manager with BEA WebLogic Server* available from /opt/wlm/toolkits/weblogic/doc/weblogic_wlm_howto.html and http://www.hp.com/go/wlm. The paper guides you through example WLM configurations that show how to manage various types of WebLogic instances.

For more WebLogicTK information

If you would like to learn more about WebLogicTK, see:

- /opt/wlm/toolkits/weblogic/doc/weblogic_wlm_howto.html.
- wlmtk(5) man page
- wlmwlsdc(1M) man page
- expsmooth(1M) man page
- *HP-UX Workload Manager Toolkits User's Guide* (opt/wlm/toolkits/doc/WLMTKug.pdf)
- *HP-UX Workload Manager Toolkits A.01.08 Release Notes* (/opt/wlm/toolkits/doc/Rel_Notes)

E Useful PRM utilities

This appendix highlights PRM utilities that WLM users may find helpful.

NOTE

These PRM utilities are helpful only when the current WLM configuration includes a `prm` structure.

For additional information, refer to the man pages or to the *Process Resource Manager User's Guide* (available in /opt/prm/doc/).

Useful PRM utilities are:

- `prmanalyze` (with any options)

  Allows you to analyze resource usage and contention to help plan configurations.

- `prmavail` (with any options)

  Displays estimated resource availability to help plan configurations.

- `prmconfig` (with no options or with `-u`)

  Displays configuration information (with no options).

  Releases, or unlocks, the configuration lock (`-u`). If you need to release a configuration lock, make sure the WLM daemon is completely stopped by entering the following command:

  ```
  # wlmd -k
  ```

NOTE

The `prmconfig` command has other options, but you should not use them with WLM.

- `prmlist` (with any options)

  Displays the current workload group, memory, user, application, and disk configuration information.

- `prmmonitor` (with any options)

  Monitors current PRM configuration and resource usage by workload group.

- `prmmove` (with any options)

  Moves processes or groups of processes to another workload group.

- `prmrun` (with any options)

  Runs an application in its assigned group or in a specified group.

Do not use the `prmloadconf` and `prmrecover` tools with WLM.

F Understanding how PRM manages resources

One of the ways in which WLM performs resource management is through HP Process Resource Manager (PRM).

This appendix focuses on management using PRM. The appendix is for background information and is included here only for completeness, as WLM uses some of these concepts and tools.

Table F-1 provides an overview of the resource management.

Table F-1 Resources managed

Resource managed	Management algorithm
CPU	You set minimum and maximum CPU request values and/or a shares-per-metric request, then WLM automatically adjusts CPU based on workload performance, priority, and available CPU. With WLM, you can also set hard limits on CPU usage with the `gmincpu` and `gmaxcpu` keywords.
Disk bandwidth	PRM re-orders the Logical Volume Manager (LVM) volume group's I/O requests in the kernel based on the PRM group shares.
Real memory	If the system is paging or a workload group is exceeding its cap, the `prm2d` memory manager forces the group's processes to re-use their own workload group's memory. With WLM, you can also set hard limits on memory usage with the `gminmem` and `gmaxmem` keywords.

Management of CPU

When you configure WLM, you specify minimum and maximum requests for CPU shares and/or a shares-per-metric request for each SLO. Based on the SLO's priority, the workload's performance, and the system's available CPU, WLM manages each workload group's number of CPU shares, increasing or decreasing that number automatically.

WLM drives each workload group's CPU to a certain number of shares, resulting in a new PRM configuration. PRM then enables the new configuration, giving groups with more shares more opportunities to use CPU time. (WLM's partition management is performed without the PRM configuration.)

For an overview of the method WLM uses to allocate CPU, see "Allocating CPU: The rising tide model" on page 112.

Management of CPU for real-time processes

Although PRM is designed to treat processes fairly based upon their assigned shares, PRM does not restrict real-time processes. Real-time processes using either the POSIX.4 real-time scheduler (`rtsched`) or the HP-UX real-time scheduler (`rtprio`) keep their assigned priorities because timely scheduling is crucial to their operation. Hence, they are permitted to exceed their group's number of CPU shares and its cap. However, the CPU they use is still counted as part of the CPU allocated to their groups. Thus, they can prevent other processes in their groups from running.

Management of real memory

NOTE WLM manages memory based on your use of the keywords `gminmem`, `gmaxmem`, and `memweight` in your WLM configuration.

A portion of real memory is always reserved for the kernel (/stand/vmunix) and its data structures, which are dynamically allocated. The amount of real memory not reserved for the kernel and its data structures is termed available memory. Available memory is consumed by user processes and also nonkernel system processes such as network daemons. Because the size of the kernel varies depending on the number of interface cards, users, and values of the tunable parameters, available memory varies from system to system.

PRM memory management allows you to prioritize how available memory is allocated to user and application processes. This control enables you to ensure that critical users and applications have enough real memory to make full use of their CPU time.

Processes in the `PRM_SYS` group (ID 0) and the kernel get as much memory as they need. They are not subject to PRM memory constraints.

The `prm2d` memory manager is the default as of HP-UX 11i v1 (B.11.11). `prm2d` is the recommended manager.

The `prm2d` memory manager partitions memory with each workload group getting a partition. A partition includes *x* Mbytes of memory, where *x* Mbytes is equivalent to the group's entitled percent of the available memory. Each partition pages separately.

When system memory use is not at 100%, a workload group that does not have its memory use capped can freely borrow excess memory pages from other workload groups. If a process requires memory and its memory use is capped, processes in the same workload group as the original process are forced to page to free up memory.

When system memory use is at a peak, any borrowed memory pages are returned to the owning workload groups. In addition, if a group is exceeding its memory allocation, `prm2d` forces members of that group to re-use their own memory pages.

`prm2d` does not allow groups to exceed their memory caps.

Capping memory use

You can optionally specify a memory cap (upper bound) for a workload group using the `gmaxmem` keyword in your WLM configuration, as explained in “Specifying a group’s maximum memory (optional)” on page 166. Typically, you might choose to assign a memory cap to a workload group of relatively low priority, so that it does not place excessive memory demands on the system. With `prm0d`, memory caps are enforced only when the system is paging. `prm2d` never allows the memory use of processes in a workload group to exceed the memory cap of that workload group.

NOTE Processes in the `PRM_SYS` group (ID 0) and the kernel are not subject to memory capping.

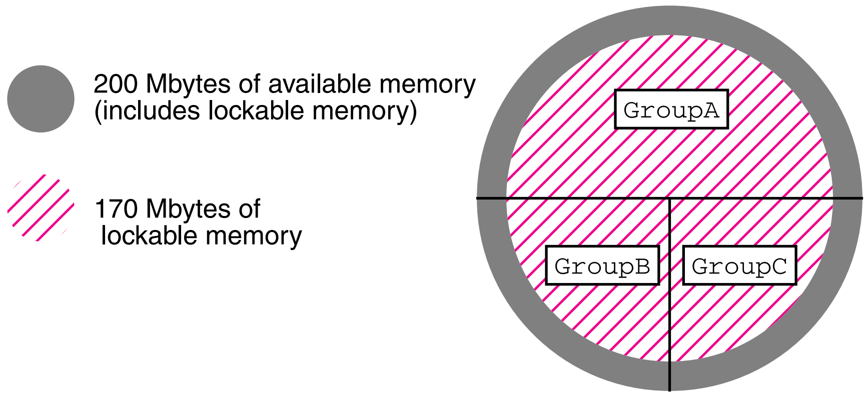
Management of locked memory

Real memory can be locked (that is, its pages kept in memory for the lifetime of a process) by the kernel, by the `plock()` system call, or by the `mlock()` system call.

Locked memory cannot be paged or swapped out. Typically, locked real memory holds frequently accessed programs or data structures, such as critical sections of application code. Keeping them memory-resident improves system performance. Locked memory is extensively used in real-time environments, like hospitals, where some processes require immediate response and must be constantly available.

With the `prm2d` memory manager, locked memory is distributed based on the assigned memory allocations. For example, assume a system has 200 Mbytes of available memory, 170 Mbytes of which is lockable. Lockable memory divided by available memory is 85%. If `GroupA` has a 50% memory allocation, it gets 100 Mbytes of real memory. Of that amount, 85% (or 85 Mbytes) is lockable. Notice that 85 Mbytes/170 Mbytes is 50%, which is the group’s memory allocation. Figure F-1 illustrates this idea.

Figure F-1 **Locked memory distribution by `prm2d` memory manager**



PRM does not suppress a process that uses locked memory once the process has the memory because suppressing the process will not cause it to give back memory pages. However, the memory resources that such a process consumes are still counted against its PRM group. If processes using locked memory consume as much or more memory than their group is entitled to, other processes in that group may be suppressed until the demands of the processes with locked memory are lower than the group's share.

Also, if a workload group is using all of its lockable memory, you cannot reduce its memory allocation. This is because a reduction in its allocation would result in a reduction in its amount of lockable memory. However, because the group is already using all of its lockable memory, it cannot give that memory up.

Management of disk bandwidth

PRM manages disk bandwidth at the logical volume group level. As such, your disks must be mounted and under the control of Logical Volume Manager (LVM) to take advantage of PRM disk bandwidth management.

LVM divides the disk in much the same way as the hard partitions implemented under earlier versions of HP-UX for the Series 800 systems. However, logical volumes are much easier to reconfigure than partitions, and they can span two or more disks. These two attributes make LVM a much more powerful and flexible tool than hard partitions.

LVM uses the concept of a volume group, which is a collection of one or more disks. A volume group can be divided into several partitions, which are called logical volumes.

NOTE When setting up LVM, do not create swap partitions in any volume group that is under PRM control.

For information on using LVM, see *Managing Systems and Workgroups: A Guide for HP-UX System Administrators*. This book is available on the web at http://docs.hp.com.

PRM controls disk bandwidth by re-ordering a volume group's I/O requests. This has the effect of delaying the I/O requests of low-priority processes and accelerating those of higher-priority processes.

Disk bandwidth management works only when there is contention for disk bandwidth, and it works only for actual I/O to the disk. (Commonly, I/O on HP-UX is staged through the buffer cache to minimize or eliminate as much disk I/O as possible.)

Disk bandwidth management works on disk devices, stripes, and disk arrays. It does not work on tape or network devices.

When you change share allocations on a busy disk device, it typically takes 30 seconds for the actual bandwidth to conform to the new allocations.

Multiple users accessing raw devices (raw logical volumes) will tend to spend most of their time seeking. The overall throughput on this group will tend to be very low. This degradation is not due to PRM's disk bandwidth management.

When performing file system accesses, you need approximately six disk bandwidth consumers in each workload group before I/O scheduling becomes noticeable. With two users, you just take turns. With four, you still spend a lot of your time in system call overhead relative to the peak device bandwidth. At six, PRM disk bandwidth management begins to take effect. The more demand you put on the system, the closer the disk bandwidth manager approaches the specified values for the shares.

How resource allocations interact

You can assign different numbers of shares for CPU, memory, and disk bandwidth to a workload group depending on the group's requirements for each type of resource. To optimize resource use, it is important to understand the typical demands for resources within a workload group.

For example, suppose the DesignTool application is assigned to workload group `DTgroup`, and it is the only application running in that group. Suppose also that the DesignTool application uses CPU and memory in an approximate ratio of 2 to 3. For optimal results, you should assign the resource shares for `DTgroup` in the same ratio. For example, assign 10 CPU shares and 15 memory shares or 20 CPU shares and 30 memory shares.

If the percentages assigned do not reflect actual usage, then a workload group may not be able to fully utilize a resource to which it is entitled. For instance, assume you assign 50 CPU shares and 30 memory shares to `DTgroup`. At times of peak system load, `DTgroup` is able to use only approximately 20 CPU shares (although it is assigned 50 shares) because it is limited to 30 memory shares. (Recall that DesignTool uses CPU and memory at a ratio of 2 to 3.) Conversely, if `DTgroup` is assigned 10 CPU shares and 30 memory shares, then at times of peak system load, `DTgroup` is only able to utilize 15 memory shares (not its 30 shares), because it is restricted to 10 CPU shares.

To use system resources in the most efficient way, monitor typical resource use in workload groups and adjust shares accordingly. You can monitor resource use with the `wlminfo` command, the `prmanalyze` command, the `prmmonitor` command, or the optional HP product GlancePlus.

For `wlminfo` information, see the wlminfo(1M) man page. For `prmanalyze` information, see the prmanalyze(1) man page. For more information on `prmmonitor`, see the prmmonitor(1) man page.

Management of applications

This section describes how PRM assigns processes to run in workload groups. The following topics are discussed:

- How application processes are assigned to workload groups at start-up
- How child processes are handled
- Pattern matching for filenames
- Pattern matching for renamed application processes
- How the application manager affects workload group assignments

When an application is started, it runs in the initial workload group of the user that invoked it. If the application is assigned to a workload group by a record in the WLM configuration file, the application manager soon moves the application to its assigned group. A user who does not have access to an application's assigned workload group can still launch the application as long as the user has execute permission to the application. An application can be assigned to only one workload group at a time. Child processes inherit their parent's workload group. Therefore, all the application's child processes run in the same workload group as the parent application by default.

You can explicitly place an application in a workload group with two commands. Use the `prmrun` command to start an application in a specified group. Use the `prmmove` command to move an existing application to another group.

These rules may not apply to processes that bypass `login`. See the *Process Resource Manager User's Guide* for more information.

How application processes are assigned to workload groups at start-up

Table F-2 describes what workload groups an application process is started in based on how the application is started.

Table F-2 Group assignments at process start-up

Process initiated	Process runs in workload group as follows
By user By `at` By `cron` Upon login	Process runs in the user's initial group. If the user does not have an initial group, the process runs in the user default group, `OTHERS`. (If the process has a compartment record or an application record, it still starts in the invoking user's initial group. However, the application manager will soon move the process to its assigned group—with compartment records taking precedence over application records. Record types are discussed in "Configuring WLM" on page 127.)
By `prmrun` {`-g` *`targetgrp`* | `-i`}	Process runs in the workload group specified by *`targetgrp`* or in the user's initial group. The application manager cannot move a process started in this manner to another group. For more information on the PRM utility `prmrun`, see "Useful PRM utilities" on page 439 and the prmrun(1) man page.
By `prmrun` *`application`* (`-g` *`targetgrp`* is not specified)	Process runs in the application's assigned workload group. If the application does not have a group, an error is returned.
By `prmmove` {*`targetgrp`* | `-i`}	Process runs in the workload group specified by *`targetgrp`* or in the user's initial group. The application manager cannot move a process started in this manner to another group. For more information on the PRM utility `prmmove`, see "Useful PRM utilities" on page 439 and the prmmove(1) man page.
By another process	Process runs in the parent process's group.

How child processes are handled

When they first start, child processes inherit the workload groups of their parent processes. At configurable polling intervals, the application manager checks the PRM configuration file (maintained by WLM) against all processes currently running. If any processes should be assigned to different workload groups, the application manager moves those applications to the correct workload groups. (You can configure the polling interval with "`prmconfig -I` *`interval`* `APPL`". For information, see prmconfig(1).)

If you move a parent process to another workload group (with the `prmmove` command), all of its child processes remain in the original workload group. If the parent and child processes should be kept together, move them as a process group or by user login name.

Pattern matching for filenames

Application filenames in application records can contain pattern matching notation as described in the regexp(5) man page. This feature allows you to assign all appropriate applications that reside in a single directory to a workload group—without creating an application record for each individual application.

The wildcard characters (`[`, `]`, `*`, and `?`) can be used to specify application filenames. However, these characters cannot be used in directory names. For example, the following record is valid:

```
apps = PRM_SYS : "/opt/wlm/bin/wlm[bcd]";
```

However, the next record uses a wildcard in the directory name and is not valid:

```
apps = PRM_SYS : "/opt/wl?/bin/wlmd"; #INVALID
```

NOTE Be sure to quote strings that include wildcard characters.

To assign all the applications in a directory to a workload group, create an application record similar to the following, with the filename specified only by an asterisk (`*`):

```
apps = GroupS : "/opt/special_apps/bin/*";
```

Filenames are expanded to their complete names when a PRM configuration is loaded by WLM. Explicit application records take precedence over application records that use wildcards. If an application without an explicit record is matched by several records that use pattern matching, the record closest to the beginning of the configuration file is used.

NOTE If you use wildcards in an application record to specify the application filename, you cannot use alternate names for that application record.

Pattern matching for renamed application processes

Alternate names specified in application records can also contain pattern matching notation as described in the regexp(5) man page.

NOTE Use pattern matching only when it is not practical to list all possible alternate names.

Many complex applications, such as database applications, may assign unique names to new processes or rename themselves while running. For example, some database applications rename processes based on the database instance, as shown in this list of processes associated with a payroll database instance:

```
db02_payroll
db03_payroll
db04_payroll
dbsmon_payroll
dbwr_payroll
dbreco_payroll
```

To make sure all payroll processes are put in the same workload group, use pattern matching in the alternate names field of the application record, as shown below:

```
apps = business_apps : “/usr/bin/database db*payroll”;
```

For alternate names and pattern matching to work, the processes must share the same file ID. (The file ID is based on the file system device and the file's inode number.) PRM performs this check to make sure that only processes associated with the application named in the application record are put in a configured workload group.

If there are multiple application records with alternate names that match an application name due to redundant pattern matching resolutions, the "first" record to match the application name takes precedence. For example, the application `abb` matches both of the following application records:

```
apps = GroupA : /opt/foo/bin/bar "a*",
       GroupB : /opt/foo/bin/bar "*b";
```

Because the `*b` record is first (based on ASCII dictionary order), the application `abb` would be assigned to the workload group `GroupB`.

Knowing the names of all the processes spawned and renamed by the applications can help in creating pattern matching that is only as general as it needs to be. Eliminate redundant name resolutions whenever possible, and make sure pattern matching does not cause unwarranted moves.

For information on how alternate name pattern matching affects precedence, see the next section.

How the application manager affects workload group assignments

The PRM application manager checks that applications are running in the correct workload groups every *`interval`* seconds. The default *`interval`* is 30 seconds; however, you can change it with "`prmconfig -I` *`interval`* `APPL`" as explained in the prmconfig(1) man page.

The application manager uses the following checks in the order given to determine whether to move an application to a workload group specified in an application record or user record:

1. Did a user manually move (using `prmrun` or `prmmove`) the application to the current workload group?

   If yes, leave the application as is.

   If no, continue with the checklist.

2. Is the application running in a secure compartment that is mapped to a PRM group using a compartment record? (Use the HP-UX feature Security Containment to create secure compartments.)

   - If yes, move the application to the group assigned in the record.
   - If no, continue with the checklist.

3. Does the file ID of the application match the file ID for the full pathname of any application listed in an application record in the current configuration?

   If no, skip to Step 3.

   If yes, continue with the checklist.

   a. Is the application name an exact match of an alternate name given in an application record in the current configuration?

      If yes, move the application to the group assigned in the record.

      If no, continue with the checklist.

   b. Does the application name match any of the alternate names specified by pattern (regular expression) that are in application records in the current configuration?

      If the application name matches only one alternate name, move the application to the group specified in that record.

      If the application name matches multiple alternate names specified by pattern, move the application to the workload group specified in the “first” matching record. Because the application matches each of these records, the “first” matching record is determined by sorting the alternate names specified by pattern in lexicographical (ASCII dictionary) order.

      If no, continue with the checklist.

   c. Are there any application records with this file ID that do not specify an alternate name?

      If yes, move the current application to the workload group specified in the application record.

      If no, continue with the checklist.

4. Is the application in the `PRM_SYS` group and does it belong to a nonroot user?

   If yes, move the application to the initial workload group specified in the user record for the owner or to the `OTHERS` group if the user does not have a record.

   If no, continue with the checklist.

5. Is the application run by a nonroot user and does it have a user ID different from its parent?

   If yes, determine the user's initial group and the group in which the application is currently running. If these groups are not the same, move the application to the initial group.

   If no, leave the application as is.

To illustrate these rules, consider the following application records:

```
apps = GroupA : /bin/call_home,
       GroupB : "/bin/cal*",
       GroupC : /bin/cal,
       GroupD : "/bin/c*",
       GroupE : /bin/call_home phone_home "tele*_home",
       GroupF : /opt/foo/bin/bar,
       GroupG : /opt/foo/bin/bar_none,
       GroupZ : /bin/call_home "*home";
```

Assume a user starts an application, `my_favorite_app`, without using `prmrun`:

```
% my_favorite_app
```

Because the application does not have an application record, it does not meet any of the criteria above and starts in the invoking user's workload group.

Now assume the user starts the `bar_none` application, which has a record, but is started in a group specified using `prmrun`.

```
% prmrun -g GroupA bar_none
```

In this case, the application manager determines that the application has been moved manually and leaves it as is, in `GroupA`.

Next, assume the user launches the `bar` application, which also has an application record.

```
% bar
```

The application starts in the invoking user's initial group. However, the application manager will soon place the application in the group specified in the application record, `GroupF`.

The user then starts another program:

```
% phone_home
```

This application name is an exact match of an alternate name. If `phone_home` has the same file ID as `/bin/call_home`, the `phone_home` process is placed in `GroupE`.

Another user on the system starts a program:

```
% telegraph_home
```

This application name matches two alternate names, both specified using regular expressions: `tele*_home` and `*home`. Sorting based on the ASCII dictionary, the application matches `*home` first. Assuming `telegraph_home` has the same file ID as `/bin/call_home`, it is placed in `GroupZ`.

Starting one more program:

```
% call_home
```

The `call_home` command is matched by the first and eighth records. (The second and fourth records do not match because PRM expands the regular expressions when the configuration is loaded and finds `call_home` already has a record.) The eighth record takes precedence because it has an alternate name, and the `call_home` process is placed in `GroupZ`.

NOTE Be careful when constructing regular expressions: As shown with the eighth record above, an expression that is not specific enough can override explicit application records.

Lastly, a user starts the following program:

```
% calendar
```

The second and fourth records both seem to match the `calendar` command. The expressions are expanded in the order they appear in the configuration file. So, the second record is expanded first and is used for the `calendar` process, placing it in `GroupB`.

G Migrating from PRM to WLM

This appendix provides information on converting PRM configuration files to WLM configuration files.

If your PRM configuration places users in PRM groups, these users can be grouped similarly in WLM workload groups.

If you are migrating from PRM to WLM, you can quickly convert your PRM configuration files to WLM configuration files with the `wlmprmconf` utility.

This utility, available at /opt/wlm/bin/wlmprmconf, has the following syntax:

```
wlmprmconf [-f] PRMconfigfile WLMconfigfile
```

`wlmprmconf` takes a PRM configuration (see prmconf(4)) as the input file *`PRMconfigfile`* and translates it to the equivalent WLM configuration (see wlmconf(4)), storing the result in the file *`WLMconfigfile`*. You can then edit the WLM configuration file to add SLO goals, change SLO priorities, add time-based conditions and exceptions, modify WLM tuning parameters, and so forth.

`wlmprmconf` validates the input PRM configuration using `prmconfig` before the translation begins. If `prmconfig` is not installed or if the input configuration is invalid, `wlmprmconf` exits without creating the WLM configuration file. However, using the `-f` option, you can force `wlmprmconf` to create the WLM configuration file in both of these cases.

After the translation, `wlmd` checks the validity of the generated WLM configuration. If the output configuration is invalid, `wlmprmconf` prints the `wlmd` error messages and exits without deleting the generated configuration file. You can then modify the generated WLM configuration file without modifying the original PRM configuration file.

`wlmd` has stricter rules than `prmconfig` regarding the validity of a configuration. In addition, `wlmd` treats most PRM warnings as errors. Therefore, it is possible for a valid PRM configuration file to be translated into a syntactically correct WLM configuration file that fails the `wlmd` check. For example, the user names listed in the PRM user records may not be defined on the system, or the application pathnames may point to executables that do not exist. In such cases, modify the generated WLM configuration file to correct the errors reported by `wlmd`.

`wlmprmconf` expects that, if specified in the PRM configuration file, groups `PRM_SYS` and `OTHERS` have IDs 0 and 1, respectively. Conversely, if IDs 0 and 1 are specified, the corresponding group names must be `PRM_SYS` and `OTHERS`, respectively. When the PRM configuration file does not follow these criteria, `wlmprmconf` generates a WLM configuration file with comments indicating the inconsistency. The `wlmd` check that WLM performs on the file it generates also highlights the inconsistency.

Glossary

Active workload group A workload group with at least one active SLO.

Active SLO An SLO is active under either of the following situations:

- It has no `condition` or `exception` statements specified
- Its `condition` statement is true and/or its `exception` statement is false, allowing WLM to try to enforce the SLO

You might set an SLO to be active, for example, based on time of day or day of week or whether a Serviceguard package is running on the local host.

Alternate name Other names assigned to processes spawned by an application. This is most common for complex programs such as database programs that launch many processes and rename them.

Alternate group A workload group other than the user's initial group that a user can access using `prmrun` or `prmmove`. These groups are listed in the user records (or their netgroups' user records) in the configuration file following the initial group, as shown below:

```
users = user : init_group [alt_group1
alt_group2 ...] [, ...];
```

Application manager A PRM daemon that polls the configuration and the running processes to ensure all processes are in the proper workload groups.

API Application Program Interface. WLM provides an API so that your data collectors can send it metrics for your workloads.

ARM Application Response Measurement. A standard for transaction-based (bounded events) response-time instrumentation.

Controller A control algorithm that requests new CPU allocations for workload groups.

CPU manager WLM uses the fair share scheduler (FSS) to manage CPU for FSS workload groups. (PSET workload groups are assigned whole CPUs and do not require the CPU manager.)

Deactivated processor A processor that either has not yet been activated or that has been turned off by the Instant Capacity software (formerly known as iCOD software) and returned to the pool of inactive processors. These processors are available for activation.

Disk manager A kernel routine that monitors bandwidth at the logical volume group level, rearranging I/O requests as needed to ensure disk bandwidth shares.

EMS Event Monitoring Service. An HP product for monitoring system resources and sending/receiving notifications when events occur. WLM uses the EMS notification framework to send alarms.

Entitlement The minimum percentage (lower limit) of a resource that a workload group receives. For CPU, this percentage is either the number of CPU shares the workload group is assigned relative to the number of CPUs multiplied by 100 (absolute CPU units are enabled)—or the number of CPU shares a workload group is assigned relative to the total number of CPU shares assigned (absolute CPU units are not being used). For memory, you can specify this percentage directly in the WLM configuration file. For disk bandwidth, the percentage is based on a workload group's number of shares relative to the total number of shares assigned.

File ID ID used by the application manager to place processes in the appropriate workload groups. The file ID is based on the file system device and inode number.

FSS workload group A WLM workload group that has its CPU use managed by the fair share scheduler (FSS). Compare to the definition for "PSET workload group."

Goal A measure of workload performance or resource utilization that must be achieved to meet an SLO. Compare to the definition for "Stretch goal."

Global arbiter A WLM daemon used in managing WLM instances on various virtual partitions.

iCAP processor A processor that is physically installed in an Instant Capacity (iCAP) system, but is not licensed, nor activated. After licensing, Instant Capacity processors can be turned on by the Instant Capacity software or during installation. Licensed processors are activated with the `icod_modify` command while HP-UX is running. (Instant Capacity was formerly known as iCOD.)

Initial group The first workload group listed in a user record in a configuration file. Typically, the applications a user launches run in the user's initial group—assuming those applications are not specified in application records.

This is the group `prmconfig`, `prmmove -i`, `login`, `at`, and `cron` use to determine where to place user processes. If a user does not have a user record or is not in a netgroup that has a user record, the user default group `OTHERS` becomes the user's initial group. The initial group is shown below as *`init_group`*:

```
users = user : init_group [alt_group1
alt_group2 ...] [, ...];
```

Inode number HP-UX uses data structures known as inodes to store data about files in a file system. The file system device combined with an inode number uniquely identifies a given file.

Logical volume group A single logical disk device under Logical Volume Manager (LVM) formed from one or more physical disk drives. WLM manages disk bandwidth on a logical volume group basis.

Logical Volume Manager (LVM) A disk-management tool used to partition physical disk drives.

Memory manager A daemon that monitors use of real memory on the system to ensure that workload groups are granted their memory shares. This daemon also enforces memory capping.

nPartitions Also known as nPars. HP's nPartitions offer both hardware isolation and operating system isolation.

OTHERS group The reserved workload group `OTHERS` with ID 1. WLM uses this group as the initial group for any user who does not have a user record in the WLM configuration file.

Partitions See the definition for virtual partitions.

PID Process ID.

PRM Process Resource Manager. An HP product used to dynamically divide resource utilization among different applications and users. WLM builds on the features of PRM and provides automatic resource allocation.

PRM_SYS group The reserved workload group `PRM_SYS` with ID 0. WLM places all system processes in this group by default. System processes are processes started by a user with UID 0.

Process group Every process (except system processes, such as `init` and `swapper`) belongs to a process group. A newly created process joins the process group of its creator. When you create a job, the shell assigns all the processes in the job to the same process group. Signals can propagate to all processes in a process group; this is a principal advantage of job control.

Process group ID Each process group is uniquely identified by an integer called a process group ID. Each process group also has a process group leader. The process group's ID is the same as the process ID of the process group leader. Every process in a process group has the same group ID.

Process ID An integer, assigned to a process at creation, that uniquely identifies the process to HP-UX.

PSET workload group A WLM workload group that is assigned CPU resources based on the processor set (PSET) upon which it is defined. Compare to the definition for "FSS workload group."

Real memory Real memory is shared by all processes. The data and instructions of any process (a program in execution) must be available to the CPU by residing in real memory at the time of execution.

Rendezvous point A designated location for holding performance data. The `wlmsend` utility forwards data for a given metric to a rendezvous point. The `wlmrcvdc` utility monitors the rendezvous point, receiving the data and sending it the WLM daemon.

Secure Resource Partition If you use the HP-UX feature Security Containment, you can create secure compartments to isolate files and processes. WLM allows you to place your secure compartments in workload groups, providing automatic resource allocation control to the compartments as well. This combination of secure compartments and workload groups is known as Secure Resource Partitions.

Shares Shares are the amounts of CPU, real memory, or disk bandwidth assigned to workload groups. A CPU share is 1/100 of a single CPU or 1/100 of each CPU on the system, depending on WLM's mode of operation.

SLA Service-level agreement. An agreement or contract between an IT group or a data center and its customer, based upon measurable service-level objectives (SLOs).

SLO Service-level objective. Generally a specific, measurable item/objective within a broader, more comprehensive service-level agreement (SLA) contract.

SLO violation An SLO violation occurs when the performance of a workload does not meet the stated goal for that workload. When such a violation happens, WLM adjusts CPU allocations, based on SLO priorities and available CPU, to better meet the SLO's performance goal.

Stretch goal Any goal that is not the highest priority goal for the associated workload. Stretch goals are met only after the higher priority goals have been met. A stretch goal indicates desired performance, whereas a goal indicates required performance. Compare to the definition for "Goal."

Strictly host-based A WLM configuration is strictly host-based when it does not include a `prm` structure.

System administrator A person responsible for day-to-day system configuration and maintenance. This person has root user capabilities.

Total CPU The maximum amount of CPU resources on the system. With absolute CPU units, total CPU is 100 multiplied by the number of CPUs. When using relative CPU units, total CPU is 100. Absolute CPU units are advantageous on systems where the number of active CPUs change due to Instant Capacity, Temporary Instant Capacity, PPU (versions prior to B.05.00), or virtual partitions. On such systems, the CPU shares you specify remain the same amount of CPU regardless of the number of active CPUs.

Transient group A workload group that is temporarily removed from the active WLM configuration because it has no active SLOs and the `transient_groups` keyword is set to 1.

User A user is any person using the system. Each user has a unique name and ID, corresponding to their login and real user ID defined in password files (such as /etc/passwd).

User default group The reserved workload group `OTHERS` with ID 1. WLM uses this group as the initial group for any user who does not have a user record in the WLM configuration file.

Virtual partitions Also known as vPars. HP-UX Virtual Partitions offer a unique granularity of partitioning. For each single CPU, you can create one virtual partition running its own "instance" of the HP-UX operating system. Complete software isolation is provided between virtual partitions. In addition, HP-UX Virtual Partitions provides the ability to dynamically move CPU resources between virtual partitions.

WLM Workload Manager. HP-UX WLM provides automatic resource allocation and application performance management through the use of prioritized service-level objectives (SLOs).

WLM interval The frequency at which WLM checks for new performance data for the managed workloads and then adjusts CPU allocations. The length of the interval is configurable, but defaults to 60 seconds.

wlmd The WLM daemon, which is used to activate WLM configurations.

wlmpard The WLM global arbiter daemon, which is used to activate WLM configurations for cross-partition management.

wlmemsmon The WLM EMS monitor, which collects WLM and SLO status information and then makes that information available to EMS clients.

Workload A collection of processes whose performance is managed as a single unit. Examples include the processes that belong to an application or all the processes owned by a user.

Workload group Any group defined in the `prm` structure with the `groups` keyword.

You assign critical applications to workload groups. WLM then manages the performance of FSS workload groups by adjusting their CPU, while assigning PSET workload groups whole CPUs for their dedicated use.

See also "Active workload group."

Index

Index

Index

Index

Index