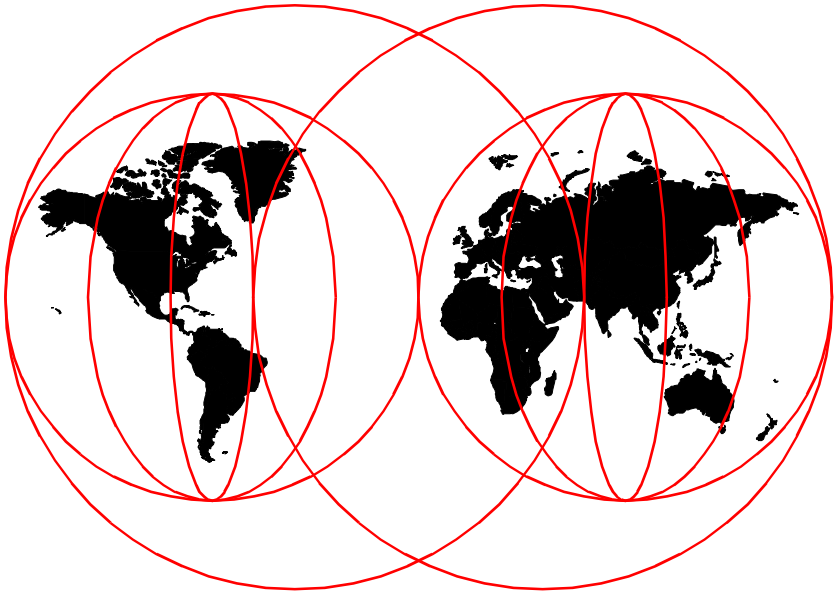
Workload Management:
SP and Other RS/6000 Servers

Janakiraman Balasayee, Bruno Blanchard, Subramanian Kannan, Akihiko Tanishita



International Technical Support Organization

www.redbooks.ibm.com

SG24-5522-00

SG24-5522-00

International Technical Support Organization

Workload Management: SP and Other RS/6000 Servers

March 2000

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix A, “Special notices” on page 241.

First Edition (March 2000)

This edition applies to PSSP Version 3 Release 1 Modification 1(5765-D51) for use with the AIX Operating System Version 4 Release 3 Modification 3 and to Version 2 Release 1 of LoadLeveler (5765-D61) and Version 2 Release 1 of Secureway Network Dispatcher in Websphere Performance pack product family (41L2594)

This document was created or updated on 20.03.2000.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

Workload management is a key issue in RS/6000 SP environments in which multiple tasks are to be executed on several SP nodes. Improper allocation of resources to these tasks can result in a waste of resources or the allocation of critical resources to less important tasks while higher-priority tasks wait. The goal of workload management is to optimize the allocation of resources to the tasks that are to be executed by an RS/6000 SP environment. The benefit to the customer is improved utilization of resources and effective management of the RS/6000 SP system.

This redbook discusses the tools that can be used to manage the workload in SP. However, most of the information provided in this book also applies to other RS/6000 Servers. This redbook contains step-by-step configuration procedures for the tools as well as useful scenarios and sample configurations. In Part 1, we discuss the installation and configuration of the workload management tools. In Part 2, we discuss example scenarios using the tools described in Part 1.

This redbook is intended to help consultants and IT managers, their technical teams, and RS/6000 sales teams in IBM that need to identify requirements and opportunities and plan for workload management on RS/6000 SPs. This redbook gives a broad understanding of workload management, and it will help you design and create solutions to optimize the use of your RS/6000 SP resources.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization.

Janakiraman Balasayee is an IT Specialist in the PSS department of IBM Global Services in India. He holds a Diploma in Electrical Engineering. He has 15 years of experience in the IT field. He provides country support for RS/6000, AIX, and SP. His areas of expertise include RS/6000 and SP hardware and AIX and SP system management.

Bruno Blanchard is an IT Architect working for IBM France at the IGS EMEA Technical Center in La Gaude. He holds an Engineer degree from Ecole Centrale de Paris and a Master of Science degree from Oregon State University. He has been with IBM since 1983 as a system engineer for VM on 43xx, 308x, 309x, and for AIX on PS/2, RT, RS/6000, and SP Systems. He is

a certified AIX and SP specialist, and his areas of expertise include Network Management on the IBM 2220, 2219, and 2225.

Subramanian Kannan is a project leader at the International Technical Support Organization, Poughkeepsie Center. He has been with IBM since 1995 and has worked in different areas related to RS/6000 SP. He currently writes redbooks and teaches ITSO workshops with a focus on RS/6000 SP technology.

Akihiko Tanishita is an IT Specialist from IBM Japan. He holds a Master of Applied Electronics degree from the Science University of Tokyo. He entered IBM Japan in 1995. Since 1995, he has worked on several RS/6000 implementations at customer sites performing administration and problem determination as well as technical support for AIX, RS/6000, and RS/6000 SP.

Thanks to the following people for their invaluable contributions to this project:

Andre Albot
IBM Austin

Curt Christopher
IBM DesMoines

Marcello Barrios, Waiman Chan, Abbas Farazdel, Lalita Malik, Rod Stevens, Lisa Valletta
IBM Poughkeepsie

Chris Gage
IBM Raleigh

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in “IBM Redbooks review” on page 257 to the fax number shown on the form.
- Use the online evaluation form found at http://www.redbooks.ibm.com/
- Send your comments in an Internet note to redbook@us.ibm.com

Part 1. Workload management tools for RS/6000 SP

Chapter 1. Introduction

The term workload management has different meanings depending on the context in which it is used. In this redbook, we restrict ourselves to the following definition:

Workload management is the activity of managing the allocation of SP resources to the jobs in an SP environment in order to optimize their use.

In other words, the goal of workload management in this book is to improve and balance the utilization of SP resources and manage the workload in an SP environment effectively.

Therefore, this definition excludes the following activities:

- Performance monitoring and tuning

  Tuning consists of configuring the system parameters, and is not the purpose of this book. Tuning is addressed in the redbooks *RS/6000 Performance Tools in Focus,* SG24-4989, and *RS/6000 SP Performance Tuning,* SG24-5340. Performance monitoring may be needed to verify that a workload management tool has indeed improved overall system throughput.

- Parallel programming

  This activity consists of splitting a job into different tasks (processes and threads) that will execute each task on a different processor and communicate with each other in order to improve the performance of the application. This activity does not belong to workload management, which considers only defined tasks and tries to optimize their execution. Parallel programming is addressed in the redbook *RS/6000 Scalable POWERparallel System: Scientific and Technical Computing Overview,* SG24-4541.

1.1 The goal of this book

We intend to provide a reference book that you can use as a source of information for workload management tools with “how-to” examples that you can use on your SP systems to manage your jobs.

We expect this book to help you identify the tools that apply to your RS/6000 SP configuration depending on the type of applications the nodes will host.

This book can also be viewed as a tutorial. It provides practical installation and configuration instructions for tools that are specific to the SP environment and shows how these tools can be configured for sample scenarios. The sample scenarios also describe how two of these tools can work together to manage workload in an SP environment.

1.2 The scope of this book

This book addresses workload management on the RS/6000 SP system. It describes the use of three IBM products that handle workload management:

- LoadLeveler
- Secureway Network Dispatcher
- AIX Workload Manager, a feature supported in AIX in version 4.3.3

These three products are also available on stand-alone RS/6000 - not only on the RS/6000 SP. In this book, we explain how these products can be used in an SP environment. However, most of this information also applies to stand-alone RS/6000. This book can also be helpful to administrators and architects of stand-alone RS/6000-based environments.

Other workload management products may be available from third-party vendors but are outside the scope of this redbook.

1.3 The organization of this book

This redbook consists of two parts:

Part 1, “Workload management tools for RS/6000 SP” on page 1, describes the three IBM workload management tools available to the user of an SP environment: LoadLeveler, Secureway Network Dispatcher, and AIX Workload Manager.

Part 2, “Workload management sample scenarios” on page 133, presents scenarios in which workload management tools are needed. Each chapter describes a scenario and explains which of the tools can be used specifically for the scenario described. When two tools can be used together to manage the workload, we will describe how you can configure them.

Chapter 2. Overview of workload management tools

Remember the definition we gave in Chapter 1, "Introduction": Workload management is the activity of managing the allocation of SP resources to the jobs in an SP environment in order to optimize their use.

Customers use SP for Server Consolidation of multiple individual servers and Internet solutions, such as Web server, mail server, Enterprise computing, Business intellegent solutions, and Large and complex high-performance computing needs. In such environments, the SP is used to run interactive sessions, batch jobs, and large parallel jobs.

Workload management of these different types of tasks cannot be met by a single product. IBM offers three products related to workload management, and this chapter provides an overview of these workload management tools.

2.1 LoadLeveler

LoadLeveler is a job management system that is based on the Condor job scheduler from the University of Wisconsin. IBM Research has made many enhancements to this base code, such as user-based priority scheduling, NFS/AFS support, and GUI. In this book, we discuss the current LoadLeveler version 2.1.

2.1.1 LoadLeveler goal

IBM LoadLeveler is a job scheduler for managing the user jobs in an SP environment. The LoadLeveler supports both serial and parallel jobs. The user's jobs are executed in SP nodes based on the job specification and resource requirements. LoadLeveler queues the user's jobs and executes them according to resource availability and priorities, thereby, managing the workload on SP nodes.

2.1.2 LoadLeveler architecture

LoadLeveler architecture consists of four components:

- Central Manager node
- Scheduling node(s)
- Executing node(s)
- Submit Only node(s)

The function of Central Manager is to manage the pool of nodes in the LoadLeveler Cluster. The Scheduling Nodes receive the jobs submitted by the users, store them in a queue, and request the central manager to allocate nodes for executing the jobs. The executing nodes run the jobs submitted by the users. The submit only nodes are for the users to submit the jobs. These nodes are generally the users' workstations, and LoadLeveler does not execute the jobs on these nodes.

A job is defined by a job command file. You can specify the name of the job, the job steps, and other LoadLeveler statements for required resources to execute the job.

The administrator can set up an alternate central manager, which automatically assumes central manager responsibilities in the event of a central manager failure.

2.1.3 LoadLeveler features

LoadLeveler provides the following features:

- A method of balancing workload among available SP nodes
- The ability to define a flexible computing environment as a LoadLeveler cluster with configurable administration and configuration files
- Commands and APIs for managing jobs
- The ability to request a suitable computing environment for jobs with the job command file.
- The ability to manage serial and parallel jobs.

2.1.4 When to use LoadLeveler

LoadLeveler is very useful in an environment in which one needs to manage serial, batch, and parallel jobs. Using LoadLeveler, the SP resources can be managed to balance the workload. You can take advantage of the LoadLeveler parameters to define the workload management criteria specific to their requirements. The LoadLeveler also increases the throughput by effectively managing the user jobs.

2.2 Secureway Network Dispatcher

In this book, we address version 2.1 of IBM Secureway Network Dispatcher. This is the new name of the product that was called eNetwork Dispatcher in version 2.0 and was called Interactive Network Dispatcher before that.

Currently, this version of the product is packaged with the Websphere performance pack software version 3.0.

2.2.1 Secureway Network Dispatcher goals

Secureway Network Dispatcher has two goals: One is aimed at the client, and one is aimed at the system administrator.

From a user point of view, the goal of Network Dispatcher is to present the client a service accessible through a TCP/IP network of servers with a unique universal IP address to access this service. This address presents the client with the view of a unique, possibly infinitely powerful, highly-available, virtual server.

For the system administrator, the goal of Network Dispatcher is to provide the means of spreading the workload of a service accessible through TCP/IP between several physical servers to maximize the resource use while providing a flexible and easy-to-manage environment.

Telnet servers, WEB (HTTP) servers, and FTP servers can take advantage of Network Dispatcher.

2.2.2 Network Dispatcher architecture

The Secureway Network Dispatcher consists of three components:

- The dispatcher
- Interactive Session Support (ISS)
- Content Based Routing (CBR)

Customers will usually not need all three products but will install one of them (Dispatcher or CBR) and, optionally, ISS.

2.2.2.1 Dispatcher

The Dispatcher is the component that provides load balancing support for TCP connections. It is installed on a server having the IP address to which all client request are sent. It then routes the user requests to one of the servers that will process it and return the answer directly to the client. There is no Network Dispatcher code to install either on the client or the application servers. The applications run unchanged. From a client point of view, the service is provided by the server on which the Dispatcher is executing. From a server administration point of view, the Dispatcher server is a front-end to several servers.

2.2.2.2 ISS

ISS is a component that can be used alone or in conjunction with Dispatcher. ISS has several roles: First, it provides a monitoring services that looks at loads on each of the servers managed by Network Dispatcher. Second, it distributes the results of this monitoring to the Dispatcher and to any other user defined “observer” that wishes to receive monitoring information. Third, it provides an intelligent name resolution service by either interfacing with an existing DNS server or by acting as a DNS server.

To use ISS services, the ISS code has to be installed on all servers to be monitored and gathered in a “cell”. One of these servers is selected to perform the role of cell manager, that is, monitoring all other servers and providing inputs to the Dispatcher, observers, and DNS servers. Other members in the cell monitor the cell manager and cooperate for one of them to take over the cell manager role in case it fails.

2.2.2.3 Content-based routing (CBR)

When only HTTP traffic is to be balanced, content-based routing (CBR) is an alternative to Dispatcher (no other protocol is managed by CBR). CBR works in conjunction with the IBM Web Traffic Express (WTE) caching proxy server to provide a load balancing service based on the content of HTTP packets.

As for the Dispatcher, CBR is only installed on a front-end between the HTTP client and the HTTP servers. This front-end must be a WTE proxy server. The client and server application are unchanged.

2.2.3 Network Dispatcher main features

Network Dispatcher has many features. We mention a few of them in this overview and present detailed information on these features in the following chapters. The following are the main features of Network Dispatcher.

2.2.3.1 Customization

Network Dispatcher is an evolutionary product that offers the user the option of customization and expansion. Users can start using Network Dispatcher with the default rules provided with the Dispatcher, ISS, or CBR. Then, if customers want to add their own set of monitoring commands, parameters, and so on, they can write their own advisors for the Dispatcher or CBR or add observers for ISS.

2.2.3.2 Expandability

Since the goal of Network Dispatcher is to hide the real implementation of servers to the service client, it of course provides the possibility to add (or remove) servers with no service down time.

2.2.3.3 High availability

The Dispatcher can be configured with an active server and a stand-by backup server. Both servers communicate through a heartbeat protocol to check that their partner is alive. If the stand-by server discovers that the active server is not responding, it can automatically take over the dispatching function.

2.2.3.4 Multiple locations support

Both the Dispatcher and ISS are designed to provide workload management between servers that are either all on the same site or on sites thousands of kilometers away.

2.2.3.5 Single point of control

Version 2.1 of Network Dispatcher provides an authenticated remote control feature that provides a secure way of managing the Network Dispatcher-controlled machines from a single point of control. This point of control need not be on the Dispatcher or server machine. It can, for example, be installed on the Network Dispatcher administrator.

2.2.4 When to use Secureway Network Dispatcher

Network Dispatcher is the product to be installed by all customers who provide a service over an intranet or the Internet in an environment that demands service adaptability to frequently-changing request types.

If you want to use an RS/6000 SP to host HTTP servers, Network Dispatcher will allow you to have all your SP nodes seen as only one IP address, whatever the number of nodes may be. Network Dispatcher causes the allocation of some functions to nodes (cgi servers on some nodes, html file server on others) to be transparent to your client.

If you have an HTTP server that faces peak access load at some time of the year, Network Dispatcher will allow you to add physical servers during this period to answer the request and then reassign them to their other tasks the remaining part of the year. In an RS/6000 SP environment, nodes can be dynamically added or subtracted from the pool of nodes that constitutes the HTTP server.

If you are looking for a virtual IP address that will always be available and remains the same so that your customers can always contact your server even when you change your machines, upgrade your SP nodes, or face a power loss in one site, Network Dispatcher can help you.

If you wish to have all your customers contact you by the same name and IP address wherever they are in the world, and if you want to install many servers around the world close to these customers so that they will not flee your site because of excessive response times, ISS will help you.

2.3 AIX Workload Manager

This section describes AIX Workload Manager, which is packaged with AIX Version 4.3.3. WLM monitors and regulates the allocation of system resources for user applications running on an RS/6000 server.

2.3.1 The goal of AIX Workload Manager

The goal of AIX Workload Manager is to provide ways of controlling resources within an RS/6000 server or in an SP node in order to balance the workload at the node level by assigning relative priorities to various sets of tasks or to prevent one application from monopolizing the system resources. AIX Workload Manager provides the system administration tools to control the CPU time and physical memory allocation within an RS/6000 server or node in SP.

RS/6000 SP nodes are among the most powerful RS/6000 machines, and they are often used in server consolidation environments. Therefore, administrators of such RS/6000 SP systems will take advantage of using the new AIX Workload Manager feature of AIX to optimize resource allocation within a node.

2.3.2 AIX Workload Manager architecture

AIX Workload Manager has been a feature of AIX since Version 4.3.3. It is included in AIX and is not an additional product.

AIX Workload Manager is only aimed at managing resources within one AIX system, that is, within a uniprocessor or SMP machine. It does not allow workload management between different AIX systems. In the case of an RS/6000 SP environment, WLM is used and configured on each node independently of the other nodes. In the case where workload management is to be used to distribute load between the SP nodes, WLM can be used on each node in conjunction with LoadLeveler or Secureway Network Dispatcher. These tools will provide the load balancing between the nodes while WLM manages resource allocation within each node.

WLM introduces the concept of class to AIX.

A class is a collection of processes. WLM monitors the CPU and physical memory utilization for all the classes of jobs and regulates their resource consumption using minimum, maximum and target values set for each class by the system administrator.

WLM automatically assigns every process to a class using a set of assignment rules given by the system administrator. This class assignment is done based on the value of three attributes of the process: User ID, group ID, and the pathname of the application file it executes. When a process is started, it is assigned a class by WLM by comparing the values of those three attributes to the values given in the assignment rules file.

Classes can be given a relative importance using an attribute of the class called the tier number (zero to nine). A class with a lower tier number will be considered more important and, thus, will have resources applied preferentially to a less critical class with a higher tier number.

2.3.3 AIX Workload Manager features

AIX Workload Manager is an optional feature of AIX. It is the responsibility of system administrators to decide how and when to use it. If WLM is not turned on, the AIX scheduler and virtual memory manager allocate resources to the processes using the same rules and priority mechanisms as previous releases of AIX.

WLM can be dynamically turned on or off while the AIX system is running and applications are in use. Class characteristics can be modified online.

2.3.4 When to use AIX Workload Manager

WLM is mainly intended to be used in powerful or large SMP systems. If your SP contains many SMP nodes (S70 or S80), WLM can help you maximize the throughput of these nodes.

WLM is to be used when a group of users or applications are to be guaranteed a fair allocation of system resources or when they have to be protected from other users or processes that are running on the same system and could end using up all available resources.

When one group of processes has absolute priority over other groups of process, WLM can be used to ensure that all resources will be allocated to this first group until it has completed the tasks.

Chapter 3. LoadLeveler

This chapter describes the installation and configuration of LoadLeveler Version 2.1 in an SP environment. The objective is to help you understand the key concepts, plan your LoadLeveler cluster, and perform the installation and configuration. First, we will try to do a basic configuration, and, later, we will discuss the management of key LoadLeveler features.

3.1 LoadLeveler and RS/6000 SP overview

LoadLeveler is a distributed network-wide job management software for scheduling jobs. RS/6000 SP can be viewed as a network of servers. SP users execute serial and batch jobs in these nodes. In such a scenario, it is extremely important to balance the workload across the SP nodes. If some nodes are heavily loaded while other nodes are idle, valuable system resources are not being fully utilized. LoadLeveler is a product that can manage your resources effectively by distributing the load across all the nodes in an SP environment.

LoadLeveler provides tools to help users build, submit, and manage batch jobs quickly and effectively in a dynamic environment. Customers can extend the LoadLeveler functions by developing applications using Application Programming Interface (API) support. One such example is the scheduler developed by the Cornell Theory Center (EASY-LL).

LoadLeveler can be used for workload balancing of both serial and parallel jobs. For parallel jobs, a parallel operating environment interfaces with LoadLeveler to obtain the multiple SP nodes required for the job's parallel tasks. With the CSS function of PSSP V3.1, LoadLeveler can support up to four user space tasks per SP switch adapter. LoadLeveler is used in many RS/6000 SP sites worldwide.

3.2 Architecture

LoadLeveler V2.1 forms a group of RS/6000 and/or nodes of RS/6000 SP as a LoadLeveler cluster managed by LoadLeveler daemons running on these nodes. From the outside of this cluster, users view it as a single computational resource.

Inside the cluster, each node is customized for characteristics, such as the maximum number of jobs the node can accept or which types of jobs it can accept. Also, each node has one or multiple roles to process jobs and

manage node status and job status by communicating with each other. As a result, LoadLeveler is aware of the resources in the cluster and schedules the jobs accordingly.

When a job is submitted to LoadLeveler, LoadLeveler examines the job and matches the requirement with the available resources. If there is no available resource to execute the job, the job is placed in the LoadLeveler queue. When the required resources become available, LoadLeveler dispatches the job to nodes.

In a LoadLeveler cluster, there exist four machine roles: Central Manager machine, Scheduling machine, Executing machine, and Submitting machine. Any node in a cluster can perform one or more roles. These roles are described in the following sections.

3.2.1 Central Manager machine

The Central Manager manages all the resources of the LoadLeveler cluster by continuously interacting with the LoadLeveler processes (known as deamons). The Central Manager gathers the status information on availability, job status, etc. to manage the jobs and resources.

The role of the Central Manager machine is to examine the submitted job's requirement and find one or more nodes in the cluster to run the job. The negotiator daemon, LoadL_negotiator, runs on this machine to performs this role. The LoadL_master and LoadL_negotiator daemons are the minimum requirement to run on the Central Manager machine. In addition, you can also enable the running of the LoadL_schedd, LoadL_startd, and LoadL_kbdd daemons in the Central Manager machine.

You can have only one negotiator daemon running in a LoadLeveler cluster. The negotiator daemon receives the job and node status messages from the Schedd daemon running in the scheduling machine and from the Startd daemon running in the executing machine. It maintains the status of the jobs and nodes in the cluster.

For High Availability, you can define one or more nodes in the cluster as the alternate central manager. The alternate central manager communicates with the primary central manager at periodic intervals to stay informed of the status of the primary central manager. This is also called the heartbeat. The time duration between heartbeats is defined by the keyword CENTRAL_MANAGER_HEARTBEAT_INTERVAL in the configuration file.

There is one more variable that has to be defined in the configuration file when you enable the alternate central manager. This keyword is

CENTRAL_MANAGER_TIMEOUT, which defines the number of heartbeat intervals the alternate central manager should wait before declaring the primary central manager inoperative or dead.

Once the alternate central manager declares the primary central manager inoperative, the LoadL_master daemon running on the alternate central manager starts the LoadL_negotiator daemon. When an alternate becomes the central manager, the jobs are not lost, but it may take some time to show the correct status when you give the LoadLeveler commands.

When the negotiator daemon receives the new job request from the schedd daemon, the negotiator daemon examines the job requirement and schedules the job based on the criteria and policy options. Once the daemon finds a job that can be executed, the negotiator daemon requests the schedd to begin taking steps to run the job.

The negotiator daemon handles the following requests from users to:

- Set priorities
- Query about jobs
- Remove jobs
- Hold or release jobs
- Favor or unfavor users or jobs

3.2.2 Executing machine

The jobs submitted by users are executed in this machine. This machine performs two main functions: It informs the job status and the machine resource information to the negotiator daemon on the central manager. The LoadL_master and LoadL_startd daemons are the minimum that should run on this machine.

The LoadL_startd daemon reports the following information to the negotiator daemon:

- The state of the startd daemon.
- The time when the current state was entered.
- The physical memory on the machine in megabytes.
- The free disk space in kilobytes on the file system where the executables for the LoadLeveler jobs assigned to this machine are stored.
- The number of seconds since the keyboard or mouse was last used. It also includes any telnet or interactive activity from any remote machine.

- The number of CPUs installed.
- The CPU load on the system is measured using the Berkeley's one-minute load average. The load average is the average of the number of processes ready to run or waiting for disk I/O to complete.
- Name of the current machine
- Adapter on the machine
- Available classes on the machine

To execute jobs, the startd daemon receives the job to be run from the schedd daemon in the scheduling machine.

When the startd starts a job on the executing machine, the startd daemon spawns the LoadL_starter starter process to start. The starter process is responsible for running the job and reporting the status back to the startd daemon. This starter process runs the job by forking a child process that runs with the user ID and group ID of the submitting user. The starter child creates a new process group to execute the user's program or a shell.

3.2.3 Scheduling machine

When a job is submitted to LoadLeveler, it is placed in a queue managed by a scheduling machine, and the information of the job is kept within this machine. There can be more than one scheduling machine in a cluster. The LoadL_master and LoadL_schedd daemons are the minimum that should run on this machine. The scheduling machine can also be an executing machine in the cluster. In that case, the LoadL_startd and LoadL_kbdd daemons will also run in this machine.

The schedd daemon receives all the jobs submitted and manages the list. The schedd daemon forwards the job information to the negotiator daemon running on the central manager machine as soon as it is received and waits for the negotiator to select the machine to run the job. When the schedd is authorized to run the job from the negotiator daemon, it contacts the startd daemon on the executing machines selected by the negotiator daemon.

3.2.4 Submit-only machine

This type of machine can only be used for submitting, querying, and cancelling jobs. There will not be any LoadLeveler daemons running on the submit-only machines. This machine cannot be a resource for processing a job in a LoadLeveler cluster. This feature allows machines that are outside the LoadLeveler cluster to submit jobs. When a job is submitted from this machine, the schedd daemon receives the job and processes the request.

A Graphical User Interface (GUI) is also available in Submit-only machines for submitting jobs.

3.2.5 How LoadLeveler processes a job

In a LoadLeveler cluster, users describe their processing requirement in a job command file. When the job command file is submitted to LoadLeveler, LoadLeveler processes the job as shown in Figure 1.

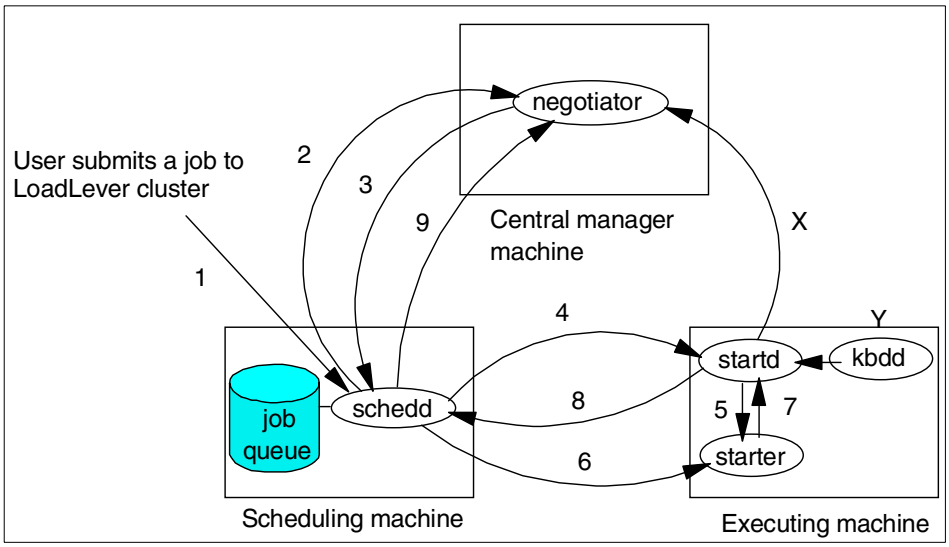


Figure 1. Flow of how LoadLeveler processes a job

The steps involved from the time the user submits the job to the time the user is notified that the job is completed are as follows:

1. When a user submits a job to the LoadLeveler cluster, the schedd daemon on the scheduling machine receives the job. A job is received as a job object and placed in a job queue.
2. The schedd daemon contacts the negotiator daemon on the Central Manager machine to report that a new job has been placed in the queue and sends job description information to the negotiator.
3. The negotiator daemon examines the job description information and decides which node or nodes should execute the job. Then, the negotiator daemon contacts the schedd daemon to begin taking steps to run the job. This process is called a *permit to run*.

4. The schedd daemon contacts the startd daemon on the executing machine and requests to start the job.
5. The startd daemon spawns the starter process. The starter runs the job by forking a child process. The starter process manages all the processes associated with a job step.
6. The schedd daemon sends the starter process the job information and the executable.
7. When the job is completed, the starter process notifies the completion status to the startd.
8. The startd daemon notifies the schedd daemon that the job has been completed.
9. The schedd daemon examines the information it has received and reports to the negotiator that the job has been completed.

3.3 Installation and configuration

This section discusses the installation and basic configuration of LoadLeveler V2.1 in an SP environment.

Installing and configuring LoadLeveler can be complex. The information in this section will help you install LoadLeveler on multiple nodes in an SP environment. This section is not a replacement for the LoadLeveler Publications. For detailed information and administration of LoadLeveler, refer to the following LoadLeveler publications:

- *LoadLeveler for AIX Version 2 Release 1 Installation Memo*, GI10-0642
- *LoadLeveler for AIX: Using and Administering*, SA22-7311

The steps for the planning, installation, and basic configuration of LoadLeveler are as follows:

1. Verify whether you have the supported version of AIX and PSSP installed on the nodes for installing LoadLeveler Version 2.1.
2. Plan how your nodes in the SP environment will be working in the LoadLeveler cluster. For example, which node will act as the Central Manager, and which node will be the executing node?
3. Plan where you want to have the various directories required for the LoadLeveler.
4. The LoadL_config and LoadL_admin files in the central manager node have to be replicated across all nodes in the cluster. This can be achieved using various methods. You will have to plan which method can be

followed in your environment. Some of the methods are discussed in Section 3.3.4.3, “Managing the configuration files within the cluster” on page 36.

5. Install the LoadLeveler product in all the nodes using installp.
6. Run the installation script on all the nodes.
7. Customize the administration and configuration files as planned in steps 2, 3, and 4.
8. Test LoadLeveler by submitting serial and parallel jobs.

3.3.1 Requirements

Before starting the installation, you have to perform the following preinstallation checks in order to avoid failures during installation:

1. Check whether you have the correct versions of AIX and PSSP installed on the nodes. Refer to the Table 1 for the supported AIX and PSSP versions for LoadLeveler Version 2.1.

Table 1. Supported AIX and PSSP versions for LoadLeveler Version 2.1

	PSSP						AIX			
LoadLeveler	**2.1**	**2.2**	**2.3**	**2.4**	**3.1**	**3.1.1**	**4.3.0**	**4.3.1**	**4.3.2**	**4.3.3**
2.1	N	N	N	N	Y	Y	N	N	Y	Y

2. Check whether you have sufficient disk space available in the /usr filesystem on all the nodes.

- To install the LoadL.full and LoadL.msg.lang filesets, you need a minimum of 21 MB of space in the /usr filesystem. An additional 2 MB of space will be required if you need to install the LoadL.html and LoadL.pdf documentation filesets.
- To install the LoadLeveler filesets in a submit-only machine, you need a minimum of 9 MB space in the /usr filesystem. An additional 2 MB of space will be required if you need to install the `LoadL.html` and `LoadL.pdf` documentation filesets.

3.3.2 Planning to configure LoadLeveler in an SP environment

Consider the following when planning to configure LoadLeveler in an SP environment:

1. Decide what user name, group name, user ID, and group ID you would like to have for LoadLeveler.
2. Decide the role of various nodes in the LoadLeveler cluster.

3. Decide the location of the LoadLeveler directory structure.

3.3.2.1 User and group IDs

LoadLeveler requires a common user name, group name, user ID, and group ID across all the nodes in the LoadLeveler cluster. By default, loadl is the name used for the LoadLeveler user and group. We also recommend that you use the default user and group named loadl.

If you want to use a user and group name other than loadl, you will have to use the new name in the appropriate places during the installation. For example, if you want to have the user name and group name be loadl2.1, you need to do the following:

- Create the user and group named loadl2.1 with its home directory as /u/loadl2.1.
- Copy the file /usr/lpp/LoadL/full/samples/LoadL.cfg to /etc directory using the following command:

  ```
  # cp /usr/lpp/LoadL/full/samples/LoadL.cfg /etc
  ```

- Edit the file /etc/LoadL.cfg so its contents look like this:

```
# cat /etc/LoadL.cfg
LoadLUserid = loadl2.1
LoadLGroupid = loadl2.1
LoadLConfig = /u/loadl2.1/LoadL_config
```

The user ID and group ID should also be the same across all nodes. There are three methods available to maintain a consistent user database across all nodes. This is discussed in detail in Chapter 12 of *The RS/6000 SP Inside Out*, SG24-5374.

Generally, in an SP environment, the SP File Collections facility provided by the PSSP software is used to maintain a consistent user and group database across all the nodes.

In the lab environment, we decided to have the default user and group ID loadl. The SP User Management and the File Collections have been enabled in order to maintain the uniformity of the user and group IDs across all nodes.

As seen from the splstdata -e output, the SP site environment data has the following attributes related to user management:

- usermgmt_config true

- filecoll_config true
- amd_config false

3.3.2.2 Planning the role of nodes

In the test environment, we had ten node SP systems. Initially, we used six nodes of the system for the LoadLeveler cluster. To discuss the example scenario (described in Part 2 of this book), we added the remaining four nodes in this cluster. You will see that we use all ten nodes when we discuss the example scenarios. We now need to decide the role of each node in our LoadLeveler cluster:

- Identify nodes for central manager and alternate central manager machines.
- Identify nodes for scheduling machines.
- Identify nodes for submit-only machines.
- Identify nodes for executing machines.

Table 2 describes the roles assigned in the test environment.

Table 2. The Role of nodes in our test environment

LoadLeveler element	The hostname of node
Central manager machine	sp4n01
Alternate Central Manager machine	sp4n06
Scheduling machine	sp4n05, sp4n15
Executing machine	sp4n01, sp4n05, sp4n06, sp4n07,sp4n08,sp4n15
Submit-only machine	sp4cws

We recommend that you not use the CWS in your LoadLeveler cluster as a computational resource.

The list of LoadLeveler filesets that comes as part of Version 2.1 is listed in Table 3.

Table 3. LoadLeveler file sets

File set	The function
LoadL.full	LoadLeveler (For nodes in the cluster)
LoadL.so	LoadLeveler (For Submit-only machine)
LoadL.msg.*lang*	LoadLeveler messages

File set	The function
LoadL.html	LoadLeveler HTML Pages
LoadL.pdf	LoadLeveler PDF Documentation

The minimum filesets required to be installed in the central manager, the alternate central manager, and the scheduling and executing machines are LoadL.full and LoadL.msg.lang.

The minimum filesets required to be installed in the submit-only node are LoadL.so and Loadl.msg.lang.

The documentation filesets, LoadL.html and LoadL.pdf, can be installed in one node or all the nodes as per your requirements.

3.3.2.3 LoadLeveler Directory Structure

LoadLeveler uses several files and directories. You should understand the purpose and usage of these files and directories before the installation. The directories are:

- Home directory

  The home directory for the use loadl is /u/loadl. The installation script creates the administration file and the global configuration file in this directory. These two files need to be identical across all the nodes in the LoadLeveler cluster.

- Release directory

  LoadLeveler products are installed in the /usr/lpp/LoadL/full directory. The LoadLeveler binaries and libraries reside in this directory.

  When you install PTFs for LoadLeveler, the PTFs have to be applied on all the nodes in the LoadLeveler cluster. If you want to avoid doing the PTF installations on all the nodes, you can share this directory from the central manager node by using AFS or NFS. In this case, you will have to create the symbolic links for the shared libraries.

  In our view, installing the LoadLeveler product on each node makes the administration tasks simpler than sharing the release directory. We recommend installing the LoadLeveler product on each node.

- Local directory

  LoadLeveler uses files that are private to each node in this directory. You can customize the keywords that are specific to a node in the local configuration file residing in this directory. In the test environment, we use the /u/loadl as the local directory for all the nodes.

LoadLeveler creates log, execute, and spool directories under this directory tree. LoadLeveler uses these directories as follows:

- **execute directory**- LoadLeveler uses this directory to store executables of jobs submitted from other machines.
- **log directory** - LoadLeveler uses this directory to store log files of daemons running on the node.
- **spool directory** - LoadLeveler keeps local job queue and checkpoint files in this directory.

Note

You should have at least 15 MB of free space in this local directory filesystem. The size of this local directory depends on your environment, such as the number of jobs and the size of logs. You need to estimate the required size and expand the filesystem if necessary.

Figure 2 on page 24 illustrates these directories and files.

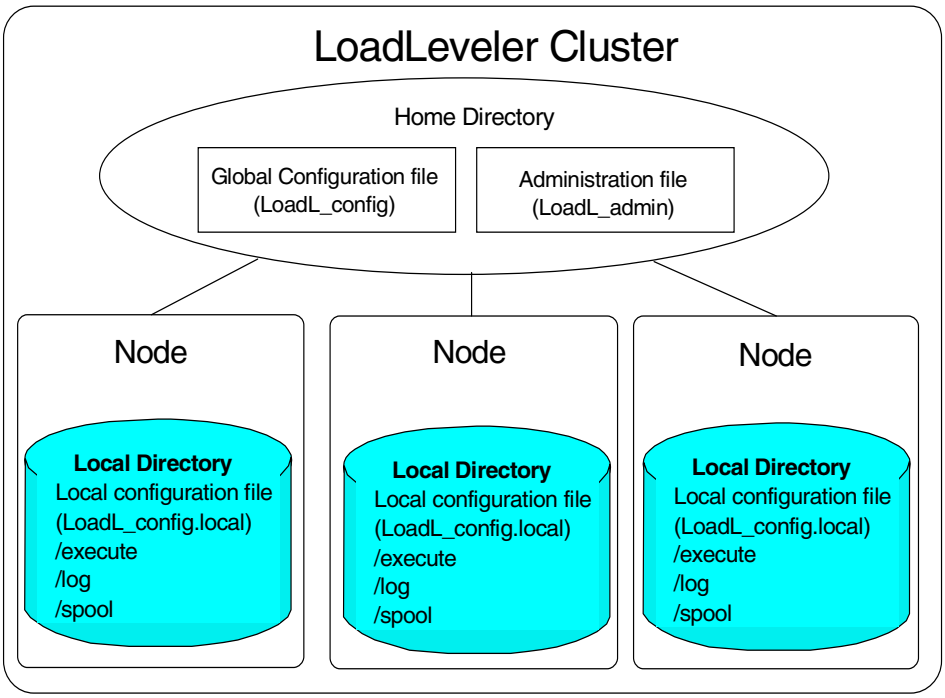


Figure 2. LoadLeveler directory and files

The following files should be common on all nodes in a LoadLeveler cluster:

- /etc/LoadL.cfg (This file is only required when you do not use default user ID and group ID)
- The administration file (LoadL_admin)
- The configuration file (LoadL_config)

It may be a good idea to configure the file collection on your RS/6000 SP for propagating these common files to nodes in your cluster. Perhaps, by sharing the home directory with NFS, you can keep these files common among nodes. Another option is using the `pcp` command when you update these files and copy to multiple nodes.

Each node maintains a local copy of the following files:

- A local configuration file (LoadL_config.local)
- Files in spool, execute, log directory

These files reside in the local directory of LoadLeveler. You can specify the same directory path with the local directory as with the home directory. However, you should keep the log, spool, and execute directories in a local file system in order to maximize performance; so, if you share the home directory among nodes in your LoadLeveler cluster, you should specify a different directory for the local directory.

3.3.2.4 General considerations

You can use the `llctl` command to control daemons on all nodes in your LoadLeveler cluster. The `llctl` command needs rsh privileges on all nodes in order to control daemons on remote nodes; so, you have to configure the environment to perform rsh privileges, such as $HOME/.rhosts and Kerberos.

When you have both Kerberos and the standard AIX authentication method in your RS/6000 SP system, the authentication of Kerberos is resolved first before the standard AIX system; so, if you do not want to get error messages from Kerberos, you need to add loadl user as the Kerberos principle and get the Kerberos ticket before performing this command.

3.3.3 Installation and configuration process

This section discusses the steps involved in installing the LoadLeveler in an SP environment. In the test environment, we decided to install the LoadLeveler on six nodes. The Installation of LoadLeveler on the nodes will be done from the CWS using the distributed shell and parallel commands that are available as part of the PSSP filesets.

The steps for installing the LoadLeveler in the nodes are as follows.

1. Log in as root in the CWS.

   You need to log in as the root user for creating user and group IDs in the CWS. Also, for executing the `installp` commands, one has to be a root user.

2. Create an SP group called loadl on the CWS using smit as follows:

   ```
   # smitty mkgroup
   ```

```
                                    Add a Group

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Group NAME                                         [loadl]
  ADMINISTRATIVE group?                               true
  Group ID                                           [2500]
  USER list                                          []
  ADMINISTRATOR list                                 []

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

In this example, we have given 2500 as the group ID. You can leave this field blank for system default.

3. Create an SP user named `loadl` on the CWS using smit as follows:

   ```
   # smitty spmkuser
   ```

Here, in the test environment, we have given 2500 as the user ID for the loadl user and its home directory as /u/loadl.

```
                                    Add a User

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* User NAME                                          [loadl]
  User ID                                            [2500]
  LOGIN user?                                         true
  PRIMARY group                                      [loadl]
  Secondary GROUPS                                   []
  HOME directory                                     [/u/loadl]
  Initial PROGRAM                                    []
  User INFORMATION                                   []


F1=Help          F2=Refresh       F3=Cancel        F4=List
F5=Reset         F6=Command       F7=Edit          F8=Image
F9=Shell         F10=Exit         Enter=Do
```

4. The next step is to propagate the loadl user and group ID to all the nodes in the SP. This can be done by executing the `supper` command from the dsh prompt on all the nodes as shown in the following screen.

```
# cat /tmp/loadlnodes
sp4n01
sp4n05
sp4n06
sp4n07
sp4n08
sp4n15
# export WCOLL=/tmp/loadlnodes
# dsh /var/sysman/supper update user.admin
sp4n01: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n01: File Changes:  3 updated, 0 removed, 0 errors.
sp4n05: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n05: File Changes:  3 updated, 0 removed, 0 errors.
sp4n06: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n06: File Changes:  3 updated, 0 removed, 0 errors.
sp4n07: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n07: File Changes:  3 updated, 0 removed, 0 errors.
sp4n08: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n08: File Changes:  3 updated, 0 removed, 0 errors.
sp4n15: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n15: File Changes:  3 updated, 0 removed, 0 errors.
```

In the test environment, we have selected six nodes that will be part of the LoadLeveler cluster. In order to install in these six nodes, it will be easy if we use these nodes as the working collective members for dsh. We created a

file, /tmp/loadlnodes, which contains the hostnames of only these six nodes, and set this as the WCOLL environment variable for dsh.

5. Create the loadl home directory, /u/loadl, on all the nodes. Also, change the file owner and group to loadl on all the nodes using the following commands:

```
# export WCOLL=/tmp/loadlnodes
# dsh mkdir /u/loadl
# dsh ls -ld /u/loadl
sp4n01: drwxr-xr-x   2 root     system      1024 Oct 31 15:51 /u/loadl
sp4n05: drwxr-xr-x   2 root     system      1024 Oct 31 15:51 /u/loadl
sp4n06: drwxr-xr-x   2 root     system      1024 Oct 31 15:51 /u/loadl
sp4n07: drwxr-xr-x   2 root     system      1024 Oct 31 15:51 /u/loadl
sp4n08: drwxr-xr-x   2 root     system      1024 Oct 31 15:51 /u/loadl
sp4n15: drwxr-xr-x   2 root     system      1024 Oct 31 15:51 /u/loadl
# dsh chown loadl.loadl /u/loadl
# dsh ls -ld /u/loadl
sp4n01: drwxr-xr-x   2 loadl    loadl        512 Oct 25 15:06 /u/loadl
sp4n05: drwxr-xr-x   2 loadl    loadl       1024 Oct 31 15:51 /u/loadl
sp4n06: drwxr-xr-x   2 loadl    loadl       1024 Oct 31 15:51 /u/loadl
sp4n07: drwxr-xr-x   2 loadl    loadl       1024 Oct 31 15:51 /u/loadl
sp4n08: drwxr-xr-x   2 loadl    loadl       1024 Oct 31 15:51 /u/loadl
sp4n15: drwxr-xr-x   2 loadl    loadl       1024 Oct 31 15:51 /u/loadl
```

6. In this step, we will install the LoadLeveler product on the nodes in our LoadLeveler cluster.

   We have to install the LoadL.full and LoadL.msg.lang filesets on the central manager, alternate central manager, and scheduling and executing nodes. In the test environment, we need to install sp4n01, sp4n05, sp4n06, sp4n07, sp4n08, and sp4n15 on the nodes.

   We do not plan to share the release directory; so, we have to install the fileset on each node. We copy the LoadLeveler filesets from the product CD to the CWS in the /psspimage/loadl directory and NFS export this directory to all the nodes for the installation.

   Install the LoadL.full and LoadL.msg.en_US filesets on all the nodes in the LoadLeveler cluster using the following commands:

```
# dsh mkdir -p /mntloadl
# dsh mount sp4en0:/psspimage/loadl /mntloadl
# dsh df | grep mntloadl
sp4n01: sp4en0:/psspimage/loadl    2007040    774496   62%      122     1% /mntloadl
sp4n05: sp4en0:/psspimage/loadl    2007040    774496   62%      122     1% /mntloadl
sp4n06: sp4en0:/psspimage/loadl    2007040    774496   62%      122     1% /mntloadl
sp4n07: sp4en0:/psspimage/loadl    2007040    774496   62%      122     1% /mntloadl
sp4n08: sp4en0:/psspimage/loadl    2007040    774496   62%      122     1% /mntloadl
sp4n15: sp4en0:/psspimage/loadl    2007040    774496   62%      122     1% /mntloadl
#
# dsh installp -acgNqwX -d /mntloadl LoadL.full LoadL.msg.en_US
```

Check the last part of the installp output for all the nodes to verify that the installation has gone through successfully. The output for all the nodes should look like the following:

```
sp4n15: Installation Summary
sp4n15: --------------------
sp4n15: Name                        Level           Part        Event       Result
sp4n15: -------------------------------------------------------------------------------
sp4n15: LoadL.full                  2.1.0.0         USR         APPLY       SUCCESS
sp4n15: LoadL.msg.en_US             2.1.0.0         USR         APPLY       SUCCESS
sp4n15: LoadL.full                  2.1.0.0         ROOT        APPLY       SUCCESS
```

Verify that the LoadLeveler filesets have been installed on all the nodes using the command shown in the following screen:

```
# export WCOLL=/tmp/loadlnodes
# dsh lslpp -la LoadL* | dshbak -c
HOSTS -------------------------------------------------------------------------
sp4n01             sp4n05             sp4n06             sp4n07
sp4n08             sp4n15
-------------------------------------------------------------------------------
  Fileset                      Level  State      Description
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  LoadL.full                 2.1.0.0  COMMITTED  LoadLeveler
  LoadL.msg.en_US            2.1.0.0  COMMITTED  LoadLeveler Messages - U.S.
                                                 English

Path: /etc/objrepos
  LoadL.full                 2.1.0.0  COMMITTED  LoadLeveler
```

7. In this step, we will install the submit-only component of the LoadLeveler. The filesets needed for a submit-only machine are LoadL.so and LoadL.msg.en_US. In the test environment, we have planned to have the CWS as the submit-only machine. The command to install the submit-only filesets in the CWS is:

   ```
   # installp -acgNqwX -d /psspimage/loadl LoadL.so LoadL.msg.en_US
   ```

   If you had planned to have one of the nodes, for example, sp4n08, as the submit-only machine, you can install using the following commands:

   ```
   # dsh -w sp4n08
   dsh > mount sp4en0:/psspimage/loadl /mntloadl
   dsh > installp -acgNqwX -d /mntloadl LoadL.so LoadL.msg.en_US
   ```

8. Change the ownership of all the executables in the /usr/lpp/LoadL so that they are owned by the loadl user ID. This excludes the LoadL_master file, which has to be owned by root. The command to accomplish this in all the nodes is:

   ```
   # export WCOLL=/tmp/loadlnodes
   # dsh chown -R loadl.loadl /usr/lpp/LoadL/full
   # dsh chown root.system /usr/lpp/LoadL/full/bin/LoadL_master
   ```

   The command to accomplish this in the submit-only machine, which, in our case, is the CWS, is:

   ```
   # chown -R loadl.loadl /usr/lpp/LoadL/so
   ```

   The next step is to run the LoadLeveler installation script, llinit, which completes the installation of LoadLeveler. This script needs to be executed on each node in the LoadLeveler cluster. You must run this script as the loadl user. The llinit determines the home directory by the value of the HOME environment variable; so, before executing this command, check that the environment variable, $HOME, for the loadl user is set to the home directory of the loadl.

   To perform this step from the CWS as loadl user, the loadl user needs authorization. To get authorization from Kerberos, one has to add loadl user as a new principle to Kerberos and get a Kerberos ticket.

   Create a new kerberos principle for user loadl using the following commands:

```
# kadmin
Welcome to the Kerberos Administration Program, version 2
Type "help" if you need it.
admin: ank loadl
Admin password:
Password for loadl:
Verifying, please re-enter Password for loadl:
loadl added to database.
admin: quit
#
#
# su - loadl
$k4init
Kerberos Initialization
Kerberos name: loadl
Password:
$
```

After getting the kerberos ticket, you can perform the `llinit` command on each node from the CWS.

The syntax of the llinit script is as follows:

llinit -local localdir -release releasedir -cm central_manager

where localdir is the full path name of the local directory; releasedir is the full path name of the release directory, and central_manager is the name of the central manager node.

The process that is performed by the llinit script includes the creation of the LoadL_admin and the LoadL_config files on the home directory and the editing of these files; so, if the home directory is shared among multiple nodes, we do not recommend that you perform llinit at the same time on multiple nodes.

The commands to initiate the llinit on all the nodes from the CWS are as follows:

```
# hostname
sp4cws
# su - loadl
$ export WCOLL=/tmp/loadlnodes
$ /usr/lpp/ssp/bin/dsh
dsh> /usr/lpp/LoadL/full/bin/llinit -local /u/loadl -release /usr/lpp/LoadL/full
 -cm sp4n01
sp4n01: /usr/lpp/LoadL/full/bin/llinit: creating directory "/u/loadl/spool".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: creating directory "/u/loadl/log".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: creating directory "/u/loadl/execute".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: set permission "700" on "/u/loadl/spool"
.
sp4n01: /usr/lpp/LoadL/full/bin/llinit: set permission "775" on "/u/loadl/log".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: set permission "1777" on "/u/loadl/execu
te".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: creating file "/u/loadl/LoadL_admin".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: creating file "/u/loadl/LoadL_config".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: creating file "/u/loadl/LoadL_config.loc
al".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: editing file /u/loadl/LoadL_config.
sp4n01: /usr/lpp/LoadL/full/bin/llinit: editing file /u/loadl/LoadL_admin.
sp4n01: /usr/lpp/LoadL/full/bin/llinit: creating symbolic link "/u/loadl/bin ->
/usr/lpp/LoadL/full/bin".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: creating symbolic link "/u/loadl/lib ->
/usr/lpp/LoadL/full/lib".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: creating symbolic link "/u/loadl/man ->
/usr/lpp/LoadL/full/man".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: creating symbolic link "/u/loadl/samples
 -> /usr/lpp/LoadL/full/samples".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: creating symbolic link "/u/loadl/include
 -> /usr/lpp/LoadL/full/include".
sp4n01: /usr/lpp/LoadL/full/bin/llinit: program complete.
```

The `llinit` creates sub-directories under the local directory and sets the appropriate permission to these directories.

9. Edit the.profile in the home directory of the loadl user to include the following entries to define the `PATH` and `MANPATH` environment variables:

```
PATH=$PATH:/usr/lpp/LoadL/full/bin:/u/loadl/bin:/usr/lpp/LoadL/so/bin
MANPATH=/usr/lpp/LoadL/full/man/$LANG:/usr/lpp/LoadL/full/man
```

3.3.3.1 Directory structure after installation

When you execute the `llinit` command, it initializes the following:

- It creates the subdirectories, spool, log, and execute, under the loadl's home directory, /u/loadl.
- It sets the required permissions for the newly-created directories.
- It creates the files, LoadL_admin, LoadL_config, and LoadL_config.local, under the /u/loadl directory.

- It edits the files LoadL_admin and LoadL_config based on the inputs given to the llinit command.
- It creates the required symbolic links under the /u/loadl directory.

The directory structure of the loadl home directory in Central Manager node, sp4n01, after executing the `llinit` command, will look like the following:

```
# dsh -w sp4n01 ls -l /u/loadl
sp4n01: -rw-------   1 loadl    loadl       1434 Nov 02 07:39 .sh_history
sp4n01: -rw-r--r--   1 loadl    loadl       8274 Nov 01 16:17 LoadL_admin
sp4n01: -rw-r--r--   1 loadl    loadl       7567 Nov 01 16:38 LoadL_config
sp4n01: -rw-r--r--   1 loadl    loadl        268 Nov 01 16:17 LoadL_config.local
sp4n01: lrwxrwxrwx   1 loadl    loadl         23 Nov 01 16:11 bin -> /usr/lpp/Lo
adL/full/bin
sp4n01: drwxrwxrwt   3 loadl    loadl        512 Nov 01 16:20 execute
sp4n01: lrwxrwxrwx   1 loadl    loadl         27 Nov 01 16:11 include -> /usr/lp
p/LoadL/full/include
sp4n01: lrwxrwxrwx   1 loadl    loadl         23 Nov 01 16:11 lib -> /usr/lpp/Lo
adL/full/lib
sp4n01: drwxrwxr-x   2 loadl    loadl        512 Nov 02 08:00 log
sp4n01: lrwxrwxrwx   1 loadl    loadl         23 Nov 01 16:11 man -> /usr/lpp/Lo
adL/full/man
sp4n01: lrwxrwxrwx   1 loadl    loadl         27 Nov 01 16:11 samples -> /usr/lp
p/LoadL/full/samples
sp4n01: drwx------   2 loadl    loadl        512 Nov 01 16:20 spool
```

The output of the loadl home directory on all the other nodes after executing the `llinit` command will look like:

```
# dsh -w sp4n05 ls -l /u/loadl
sp4n05: total 36
sp4n05: -rw-r--r--   1 loadl    loadl       8274 Nov 01 16:19 LoadL_admin
sp4n05: -rw-r--r--   1 loadl    loadl       7567 Nov 01 16:38 LoadL_config
sp4n05: -rw-r--r--   1 loadl    loadl        268 Nov 11 1998  LoadL_config.local
sp4n05: lrwxrwxrwx   1 loadl    loadl         23 Nov 01 16:11 bin -> /usr/lpp/Lo
adL/full/bin
sp4n05: drwxrwxrwt   3 loadl    loadl        512 Nov 01 16:20 execute
sp4n05: lrwxrwxrwx   1 loadl    loadl         27 Nov 01 16:11 include -> /usr/lp
p/LoadL/full/include
sp4n05: lrwxrwxrwx   1 loadl    loadl         23 Nov 01 16:11 lib -> /usr/lpp/Lo
adL/full/lib
sp4n05: drwxrwxr-x   2 loadl    loadl        512 Nov 01 16:50 log
sp4n05: lrwxrwxrwx   1 loadl    loadl         23 Nov 01 16:11 man -> /usr/lpp/Lo
adL/full/man
sp4n05: lrwxrwxrwx   1 loadl    loadl         27 Nov 01 16:11 samples -> /usr/lp
p/LoadL/full/samples
sp4n05: drwx------   2 loadl    loadl        512 Nov 01 16:20 spool
```

The screen output here only reflects the directory structure for the sp4n05 node. The other nodes will be the same as sp4n05.

3.3.4 Basic configuration

In this section, we will describe the basic minimum configuration required to start LoadLeveler. LoadLeveler needs the administration file and the configuration file to run. These files are located in the loadl home directory. The file LoadL_admin is the administration file, and LoadL_config is the configuration file. There is also a local configuration file, LoadL_config.local, in the local directory. This file is linked symbolically from the loadl home directory. During the installation process, the `llinit` command copies the default administration and configuration files to the LoadLeveler directories and edits some entries based on the information you entered as the llinit parameters.

There are many keywords that can be modified in these files to customize your requirements. The following sections explain the basic minimum configurations you need to know before starting LoadLeveler.

3.3.4.1 The Administration file

This file consists of machine, class, user, group, and adapter stanzas. For the basic configuration, you need to edit the machine stanza. Class, user group, and adapter stanzas are optional. We will discuss these stanzas in Section 3.4.1, “User, group, and class” on page 43.

For basic configuration, you need to define the role of the nodes in machine stanza. By creating this stanza, all the nodes in your LoadLeveler cluster and the submit-only machine can know which node is the central manager and which is the scheduling node; so, you need to edit this stanza to match the roles of machines in your LoadLeveler cluster.

The following screen output shows the machine stanzas defined in the LoadL_admin file for our test environment:

```
#############################################################################
# MACHINE STANZAS:
# These are the machine stanzas; the first machine is defined as
# the central manager.  mach1:, mach2:, etc. are machine name labels -
# revise these placeholder labels with the names of the machines in the
# pool, and specify any schedd_host and submit_only keywords and values
# (true or false), if required.
#############################################################################
sp4n01.msc.itso.ibm.com:        type = machine
                                central_manager = true
sp4n05.msc.itso.ibm.com:        type = machine
                                schedd_host = true
sp4n06.msc.itso.ibm.com:        type = machine
                                central_manager = alt
sp4n07.msc.itso.ibm.com:        type = machine
sp4n08.msc.itso.ibm.com:        type = machine
sp4n15.msc.itso.ibm.com:        type = machine
                                schedd_host = true
sp4en0.msc.itso.ibm.com:        type = machine
                                submit_only = true
```

In our LoadLeveler cluster, the central manager machine is sp4n01; the alternate central manager is sp4n06, and the scheduling machine is sp4n05 and sp4n15. The CWS sp4en0 is configured as the submit-only machine.

You can have only one node as the central manager. Therefore, there can be only one entry in the LoadL_admin file that has the stanza, central_manager = true, which identifies the central manager node.

The schedd_host=true stanza means that nodes sp4n05 and sp4n15 are scheduling nodes. These nodes will receive jobs submitted from the submit-only machines. Whether the schedd daemon should run on this node or not is not determined by this stanza. It is determined by the definition, SCHEDD_RUNS_HERE = true, in the LoadL_config configuration file.

3.3.4.2 The configuration file

There are two configuration files: The global configuration file (LoadL_config) and the local configuration file (LoadL_config.local). These two files have the same format and information. The global configuration file contains configuration information common to all nodes in the LoadLeveler cluster. The information of the local configuration file overrides the configuration in the global configuration file allowing specific information for an individual node.

The configuration file consists of statements that describe the keyword and the value. As part of the basic configuration, the minimum keywords that you should edit are as follows:

```
SCHEDD_RUNS_HERE = true or false
STARTD_RUNS_HERE = true or false
X_RUNS_HERE = true or false
```

These keywords define what LoadLeveler daemon runs on this machine. The `SCHEDD_RUNS_HERE`, `STARTD_RUNS_HERE`, `X_RUNS_HERE` keywords define whether schedd, startd, and the keyboard daemon run. The default value of these keywords is true. If you do not want to start the deamons on a specific node, you have to specify the keyword as false in the local configuration file.

In our example, we change the value of the keyword `X_RUNS_HERE` to false.

The role of the keyboard daemon is to monitor keyboard and mouse activity. Therefore, the daemon attempts to connect to X server on the machine. We do not start X server on our nodes. This means that we do not need to have the daemon running.

We also add the keyword entry `SCHEDD_RUNS_HERE = false` to the local configuration file on sp4n01, sp4n06, sp4n07, and sp4n08 so that the schedd daemon does not run on these nodes.

3.3.4.3 Managing the configuration files within the cluster

The LoadL_admin and LoadL_config files are to be maintained in all the nodes within the cluster. These files must remain the same in all nodes.

There are three different methods for managing the admin and config file common across all the nodes in the cluster. They are as follows:

1. Propagate the admin and config files to all the nodes in the cluster using the file collections that come with PSSP.
2. NFS exports the loadl home directory from the central manager node and mounts it on all the nodes in the cluster.
3. Copy the admin and config files to all the nodes in the cluster whenever you make changes to these files in the central manager using the `parallel copy` command, which comes as part of the PSSP.

The best method to implement in an SP environment will be to use the file collections.

File collection method

In our test environment, we use the CWS as the submit-only machine. The LoadL_admin and the LoadL_config files will be propagated from the CWS to all nodes in the cluster. For simplicity, we are adding the files to be propagated in the user.admin. If you do not want to add this to the

user.admin, you can create your own class of file collection for LoadLeveler. The steps to create a separate class for file collection are described in Section 5.3.5, “Creating a WLM configuration file collection” on page 118.

The steps to add the loadl config files in the existing file collection are as follows:

1. Log in as root in the CWS.
2. Edit the /var/sysman/sup/user.admin/list to add the following entries:

```
upgrade ./u/loadl/LoadL_admin
upgrade ./u/loadl/LoadL_config
```

3. Propagate the files to the nodes using the following commands:

```
# export WCOLL=/tmp/loadlnodes
dsh
dsh> /var/sysman/supper scan user.admin
dsh> /var/sysman/supper update user.admin
sp4n01: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n01: File Changes:  2 updated, 0 removed, 0 errors.
sp4n05: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n05: File Changes:  2 updated, 0 removed, 0 errors.
sp4n06: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n06: File Changes:  2 updated, 0 removed, 0 errors.
sp4n07: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n07: File Changes:  2 updated, 0 removed, 0 errors.
sp4n08: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n08: File Changes:  2 updated, 0 removed, 0 errors.
sp4n15: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n15: File Changes:  2 updated, 0 removed, 0 errors.
```

NFS mount method

In this method, you will be exporting the loadl home directory from the central manager machine to all the nodes in the cluster. The steps to implement are as follows:

1. Log in as root in the node CWS.
2. First, you need to start the NFS daemons and then export the loadl home directory on the NFS server. You can perform this task using the following commands:

```
# dsh -w sp4n01 mknfs -B
sp4n01: 0513-029 The portmap Subsystem is already active.
sp4n01: Multiple instances are not supported.
sp4n01: Starting NFS services:
sp4n01: 0513-059 The biod Subsystem has been started. Subsystem PID is 21698.
sp4n01: 0513-059 The nfsd Subsystem has been started. Subsystem PID is 21982.
sp4n01: 0513-059 The rpc.mountd Subsystem has been started. Subsystem PID is 24244.
sp4n01: 0513-059 The rpc.statd Subsystem has been started. Subsystem PID is 22552.
sp4n01: 0513-059 The rpc.lockd Subsystem has been started. Subsystem PID is 22954.
sp4n01: Completed NFS services.
sp4n01: 0513-095 The request for subsystem refresh was completed successfully.
# dsh -w sp4n01 mknfsexp -d /u/loadl -t rw -B
sp4n01: /u/loadl
sp4n01: Exported /u/loadl
#
```

3. Mount the loadl home directory on the NFS client in the LoadLeveler cluster. In our case, nodes sp4n05, sp4n06, sp4n07, sp4n08, and sp4n15 are the NFS clients.

   You can mount an NFS filesystem with the `mknfsmnt` command. By using this command, you can add the home directory to the /etc/filesystem and configure mounting automatically at system restart.

   You can also ensure the result of mounting using the `df` command. The following screen shows that /u/loadl of sp4n01 is available by mounting on nodes in the LoadLeveler cluster:

```
# dsh mknfsmnt -f /u/loadl -d /u/loadl -h sp4n01 -A -w bg
# dsh df | grep \/u\/loadl
sp4n05: sp4n01:/u/loadl       8192      7648    7%       72     8% /u/loadl
sp4n06: sp4n01:/u/loadl       8192      7648    7%       72     8% /u/loadl
sp4n07: sp4n01:/u/loadl       8192      7648    7%       72     8% /u/loadl
sp4n08: sp4n01:/u/loadl       8192      7648    7%       72     8% /u/loadl
sp4n15: sp4n01:/u/loadl       8192      7648    7%       72     8% /u/loadl
```

Parallel copy method

In this method, whenever you make any changes to the LoadL_admin or the LoadL_config file, you need to copy them manually to all the nodes in the cluster using the `pcp` command. The `pcp` command is available as part of PSSP. The steps for copying the config files from the central manager to all the nodes in the cluster are as follows:

1. Log in as root in the Central Manager node, sp4n01.

2. Execute the following commands to copy the LoadL_admin and LoadL_config files from the node, sp4n01, to all the nodes in the cluster.

```
# pcp -w sp4en0,sp4n05,sp4n06,sp4n07,sp4n08,sp4n15 \
/u/loadl/LoadL_admin /u/loadl/LoadL_admin
# pcp -w sp4en0,sp4n05,sp4n06,sp4n07,sp4n08,sp4n15 \
/u/loadl/LoadL_admin /u/loadl/LoadL_admin
```

3.3.5 Starting LoadLeveler

You can start the LoadLeveler daemons from any node in your LoadLeveler cluster using the user ID, loadl.

To start the daemons on all the nodes that are defined in the machine stanza, enter the command:

```
llctl -g start
```

When you run the `llctl` command, the screen output will look like the following:

```
$ llctl -g start
llctl: Attempting to start LoadLeveler on host sp4n01.msc.itso.ibm.com.
LoadL_master     2.1.0.0  rtrot3dh  98/10/22  AIX 4.3  2
10/27 19:08:05  CentralManager = sp4n01.msc.itso.ibm.com
llctl: Attempting to start LoadLeveler on host sp4n05.msc.itso.ibm.com.
spk4rsh: 0041-003 No tickets file found.  You need to run "k4init".
rshd: 0826-813 Permission is denied.
llctl: Attempting to start LoadLeveler on host sp4n06.msc.itso.ibm.com.
spk4rsh: 0041-003 No tickets file found.  You need to run "k4init".
rshd: 0826-813 Permission is denied.
llctl: Attempting to start LoadLeveler on host sp4n07.msc.itso.ibm.com.
spk4rsh: 0041-003 No tickets file found.  You need to run "k4init".
rshd: 0826-813 Permission is denied.
llctl: Attempting to start LoadLeveler on host sp4n08.msc.itso.ibm.com.
spk4rsh: 0041-003 No tickets file found.  You need to run "k4init".
rshd: 0826-813 Permission is denied.
llctl: Attempting to start LoadLeveler on host sp4n15.msc.itso.ibm.com.
spk4rsh: 0041-003 No tickets file found.  You need to run "k4init".
rshd: 0826-813 Permission is denied.
$
```

As mentioned in Section 3.3.2.4, “General considerations” on page 25, llctl uses rsh in order to start the daemons on other nodes. You may see a Kerberos error message in the screen output if the loadl user does not have a Kerberos ticket when performing the command. But, you can ignore these messages if you have set up your RS/6000 SP for standard AIX authentication for the `rsh` command. However, if you do not have permission

to perform rsh even on standard AIX authentication, llctl fails to start the LoadLeveler daemons on remote hosts.

The following is the output when we perform llctl after getting the Kerberos ticket:

```
$ llctl -g start
llctl: Attempting to start LoadLeveler on host sp4n01.msc.itso.ibm.com.
LoadL_master     2.1.0.0  rtrot3dh  98/10/22  AIX 4.3  2
10/27 19:09:41  CentralManager = sp4n01.msc.itso.ibm.com
llctl: Attempting to start LoadLeveler on host sp4n05.msc.itso.ibm.com.
LoadL_master     2.1.0.0  rtrot3dh  98/10/22  AIX 4.3  2
10/27 19:09:43  CentralManager = sp4n01.msc.itso.ibm.com
llctl: Attempting to start LoadLeveler on host sp4n06.msc.itso.ibm.com.
LoadL_master     2.1.0.0  rtrot3dh  98/10/22  AIX 4.3  2
10/27 19:09:43  CentralManager = sp4n01.msc.itso.ibm.com
llctl: Attempting to start LoadLeveler on host sp4n07.msc.itso.ibm.com.
LoadL_master     2.1.0.0  rtrot3dh  98/10/22  AIX 4.3  2
10/27 19:09:44  CentralManager = sp4n01.msc.itso.ibm.com
llctl: Attempting to start LoadLeveler on host sp4n08.msc.itso.ibm.com.
LoadL_master     2.1.0.0  rtrot3dh  98/10/22  AIX 4.3  2
10/27 19:09:45  CentralManager = sp4n01.msc.itso.ibm.com
llctl: Attempting to start LoadLeveler on host sp4n15.msc.itso.ibm.com.
LoadL_master     2.1.0.0  rtrot3dh  98/10/22  AIX 4.3  2
10/27 19:09:46  CentralManager = sp4n01.msc.itso.ibm.com
$
```

There may be a need to start the LoadLeveler in only one node. This can be done using the `-h` option of the `llctl` command. To start the LoadLeveler on the node, sp4n11, use the command:

```
$ llctl -h sp4n11 start
```

The `llctl` command starts the required LoadLeveler daemons on all the nodes based on the definition of the nodes in the LoadL_admin file. The daemons that are started and running in the central manager node can be verified using the command shown in the following screen:

```
# dsh -w sp4n01 ps -ef | grep LoadL
sp4n01:    loadl  9596 22486   0 16:02:21 -  0:04 LoadL_negotiator -f
sp4n01:    loadl 22486     1   0 16:02:21 -  0:00 /usr/lpp/LoadL/full/bin/LoadL_master
sp4n01:    loadl 24660 22486   3 16:02:21 -  0:55 LoadL_startd -f
```

The daemons that are started and running on scheduler nodes, sp4n05 and sp4n15, can be seen using the following command:

```
# dsh -w sp4n05,sp4n15 ps -ef | grep LoadL
sp4n05:    loadl 11516     1   0 16:02:22 -  0:00 /usr/lpp/LoadL/full/bin/LoadL_master
sp4n05:    loadl 11948 11516   1 16:02:22 -  0:18 LoadL_startd -f
sp4n05:    loadl 13072 11516   0 16:02:22 -  0:00 LoadL_schedd -f
sp4n15:    loadl 17688 23284   0 16:02:25 -  0:26 LoadL_startd -f
sp4n15:    loadl 21652 23284   0 16:02:25 -  0:00 LoadL_schedd -f
sp4n15:    loadl 23284     1   0 16:02:25 -  0:00 /usr/lpp/LoadL/full/bin/LoadL_master
#
```

The daemons that are started and running on the executing nodes can be verified using the following command:

```
# dsh -w sp4n06,sp4n07,sp4n08 ps -ef | grep LoadL
sp4n06:    loadl 12464 13138   1 16:02:23 -  0:17 LoadL_startd -f
sp4n06:    loadl 13138     1   0 16:02:23 -  0:00 /usr/lpp/LoadL/full/bin/LoadL_master
sp4n07:    loadl 12084     1   0 16:02:24 -  0:00 /usr/lpp/LoadL/full/bin/LoadL_master
sp4n07:    loadl 13470 12084   0 16:02:24 -  0:17 LoadL_startd -f
sp4n08:    loadl 12968     1   0 16:02:24 -  0:00 /usr/lpp/LoadL/full/bin/LoadL_master
sp4n08:    loadl 13520 12968   0 16:02:25 -  0:17 LoadL_startd -f
#
```

You can see that LoadL_master and LoadL_startd are running on all the nodes; LoadL_negotiator is running on sp4n01, which is the central manager, and LoadL_schedd is running on scheduling nodes sp4n05 and sp4n15.

If you use the default configuration file on your nodes, you may see LoadL_kbdd and LoadL_schedd running on all the nodes.

Verify the status of the LoadLeveler cluster using the `llstatus` command, and the output will look like the following:

```
# su - loadl
$ /usr/lpp/LoadL/full/bin/llstatus
Name                      Schedd  InQ Act Startd Run LdAvg Idle Arch      OpSys
sp4n01.msc.itso.ibm.com   Down      0   0 Idle     0 0.05     5 R6000     AIX43
sp4n05.msc.itso.ibm.com   Avail     0   0 Idle     0 0.00  9999 R6000     AIX43
sp4n06.msc.itso.ibm.com   Down      0   0 Idle     0 0.01  9999 R6000     AIX43
sp4n07.msc.itso.ibm.com   Down      0   0 Idle     0 0.00  9999 R6000     AIX43
sp4n08.msc.itso.ibm.com   Down      0   0 Idle     0 0.01  3534 R6000     AIX43
sp4n15.msc.itso.ibm.com   Avail     0   0 Idle     0 3.40  9999 R6000     AIX43

R6000/AIX43                 6 machines      0  jobs      0  running
Total Machines              6 machines      0  jobs      0  running

The Central Manager is defined on sp4n01.msc.itso.ibm.com
The following machine is marked SUBMIT_ONLY
sp4en0.msc.itso.ibm.com

All machines on the machine_list are present.
```

To confirm that your LoadLeveler can process your job, you have to prepare the job command file. You can see some sample files in the /usr/lpp/LoadL/full/samples directory.

You can submit a sample job command file, job1.cmd, with the `llsubmit` command. After submitting the `llsubmit` command, you can see the status of job queue with the `llq` command. The following is an example screen for testing the LoadLeveler:

```
$ llsubmit job1.cmd
llsubmit: The job "sp4n01.msc.itso.ibm.com.1" with 2 job steps has been submitte
d.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n01.1.0               loadl      10/28 04:55 I  50  No_Class
sp4n01.1.1               loadl      10/28 04:55 I  50  No_Class

2 job steps in queue, 2 waiting, 0 pending, 0 running, 0 held
```

You can see that two job steps are in the queue, No_Class, but they are not running.

To run the job steps, you need to add the Class keyword to accept No_Class jobs on these machines. The following entry shows accepting one No_Class job on this machine.

```
Class = {"No_Class"}
```

The following is the output after the class entry in the local configuration file on the sp4n15 node:

```
$ llsubmit job1.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.2" with 2 job steps has been submitt
$ lll
ksh: lll:  not found.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.2.0               loadl      10/28 15:03 R  50  No_Class     sp4n15
sp4n15.2.1               loadl      10/28 15:03 I  50  No_Class

2 job steps in queue, 1 waiting, 0 pending, 1 running, 0 held
$
```

You can see one job task running on the node, sp4n15.

3.3.6 Stopping LoadLeveler

The LoadLeveler daemons on a node can be stopped using the `llctl` command. This can only be done by the LoadLeveler administrator. The syntax for stopping the LoadLeveler in the node, sp4n01, is as follows:

```
$ llctl -h sp4n01 stop
llctl: Sent stop command to host sp4n01.msc.itso.ibm.com.
```

Use the following command to shut down the LoadLeveler cluster:

```
$ llctl -g stop
```

3.4 Managing LoadLeveler configuration

This section describes how to configure LoadLeveler for managing the workload on RS/6000 SP. The LoadLeveler administration file and the configuration file can be modified for your requirement. By modifying these files, you can manage the jobs in your RS/6000 SP.

3.4.1 User, group, and class

LoadLeveler manages jobs using the key parameters defined for user, group, and class. The job class of a job is determined by the user who submitted the job. An administrator can control the configuration for user, group, and class by modifying these stanzas in the administration file.

The following figure shows all the keywords related to the user, group, and class stanzas that you can modify in the administration file:

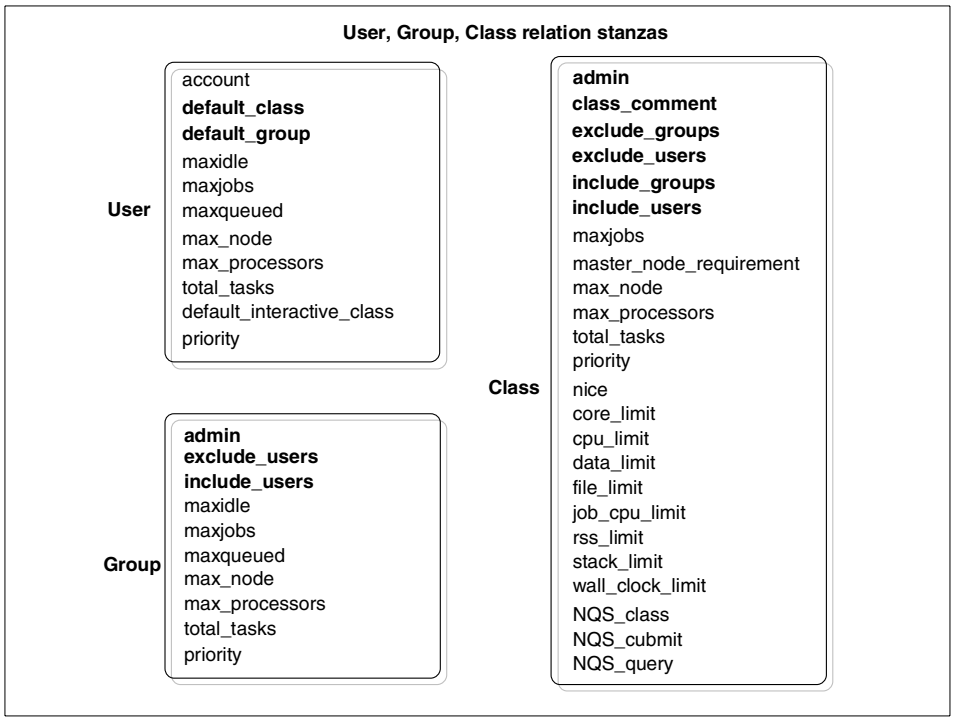


Figure 3. User, group, and class stanza keywords

You can manage LoadLeveler resources for user, group, and class with these keywords. We will discuss each keyword at first and show a sample configuration using the following keywords.

- default_class

  You can use this keyword to specify the default class for a job. A user can specify the name of a job class in the job command file with the class keyword. For example, if you want to submit a job to a specific class, say, `class_first`, you can add the following line in your job command file:

  ```
  # @ class = class_first
  ```

  If you do not specify the class in the job command file, the class you specify with this default_class keyword is used for the job class.

  If you do not specify this default_class in the administration file, the name of the default class is No_Class.

You can specify a list of class names as the value of this keyword. If you specify multiple default class names in the list, LoadLeveler selects a class that matches the resource requirement specified in the job command file.

- default_group

  You can specify the default group name to which a user can belong by assigning the name to this keyword in the user stanza. You can specify a group name when you submit a job in the job command file with the group keyword. If you do not specify the name in the job command file, the group name specified to this default_group keyword is used for jobs submitted by the user. If this keyword is not in the administration file, the default group name is No_group.

- admin

  You can use this keyword in the class and the group stanza to specify administrator names of a class and the group respectively. If you specify the names to this keyword in the class stanza, the administrators can hold, release, and cancel jobs in that class. If you specify the names in the group stanza, the administrators can hold, release, and cancel jobs that are submitted by the users in the group.

- exclude_users

  This keyword can be used in the class and group stanzas. If you specify the names to this keyword in the class stanza, the users are not permitted to submit jobs for that class. If you specify users in the group stanza with this keyword, the users are not permitted to belong to the group.

- include_users

  This keyword can be used in the class and the group stanza. If you specify the user names to this keyword in the class stanza, you can limit the users who can submit jobs for that class. If you specify the user names to this in the group stanza, you can limit the users who can belong to that group. The default is all users are included. Note that you should not specify exclude_users and include_users in the same stanza.

- exclude_groups

  This keyword can be used in the class stanza. If you specify group names to this keyword, the groups are not permitted to use this class as their job class.

- include_groups

  This keyword can be used in the class stanza. By specifying group names to this keyword, you can limit the groups that can use this class. The

default is all groups included. Note that you should not specify exclude_groups and include_groups in the same stanza.

- class_comment

  You can specify the description of the class to this keyword. You can see the description in the output of the `llclass` command.

We show an example of these user, group, and class keywords. We describe the following environment in our LoadLeveler cluster:

- We want to have two classes: class_first and class_second.
- We want to have two groups: group_para and group_serial.
- We have to manage four users: kannan1, bala1, bruno1, and tani1.

  Each user has the default class and the default group. As a default class, user kannan1 and bala1 use class_first, and bruno1 and tani1 use class_second. As the default group, kannan1 and bruno1 use group_para, and bala1 and tani1 use group_serial.
- The class class_first has include_users keyword specifying users, kannan1 and bala1 so that user bruno1 and tani1 cannot use class_first as their job class.
- The class_second class has an exclude_users keyword specifying a user, kannan1, so that user kannan1 cannot use class_second as his or her job class. The bala1 can use class_second if he or she specifies job class in the job command file.
- The group_para group has to have include_users keyword specifying users kannan1 and bruno1 so that bala1 and tani1 cannot belong to the group group_para.
- The group_serial group has to have exclude_users keyword specifying a user kannan1 so that kannan1 cannot belong to the group.
- We give administrator privilege to kannan1 for class_first and group_para.
- We give administrator privilege to bala1 for group_serial.

The following screen output of the admin file shows the keywords for the stanzas to describe the above environment:

```
############################################################################
# Sample class stanza
############################################################################
class_first:    type = class
                class_comment = "First Class Users"
                include_users = kannan1 bala1
                admin = kannan1

class_second:   type = class
                class_comment = "Second Class Users"
                exclude_users = kannan1
############################################################################
# Sample group stanz
############################################################################
group_para:     type = group
                admin = kannan1
                include_users = kannan1 bruno1
#
group_serial:   type = group
                admin = bala1
                exclude_users = kannan1
############################################################################
# Sample user stanza
############################################################################
kannan1:        type = user
                default_class = class_first
                default_group = group_para
#
bala1:          type = user
                default_class = class_first
                default_group = group_serial
#
bruno1:         type = user
                default_class = class_second
                default_group = group_para
#
tani1:          type = user
                default_class = class_second
                default_group = group_serial
```

To execute a job on a node, the node has to accept the jobs for that class. You need to define the Class keyword in the configuration file to accept the class. For example, if you add the following line to the local configuration file on the node, sp4n01, node sp4n01 can run two jobs of `class_first` and one job of `class_second`:

```
Class = {"class_first" "class_first" "class_second"}
```

So, you can use this Class keyword in the configuration file with stanzas in the administration file to specify who can use the node to run jobs and the number of jobs of that class. Therefore, if two jobs of kannan1 are running on

this node, the job submitted by bala1 as class_first cannot run on the same node until one of the jobs of kannan1 is completed.

In addition to the Class keyword, you can use the MAX_STARTERS keyword to specify how many jobs can run on a node. The MAX_STARTERS keyword can be defined in the configuration file. The keyword specifies the maximum number of jobs that can run simultaneously on a machine.

For example, if you want to limit the number of jobs running on a node to two and you want to limit the class of the jobs to `class_first` and `class_second`, you need to specify the following keywords in the local configuration file on the node:

```
Class = {"class_first" "class_first" "class_second"}
MAX_STARTER = 2
```

The possible combinations of jobs that can run on this node are:

- Two `class_first` jobs
- One `class_first` and one `class_second` job
- Only one `class_first` job, or only one `class_second` job

3.4.2 Maximum job requests from users

When multiple users want to use RS/6000 SP, you may want to control the number of jobs submitted by a user. If you do not control the number of jobs for a user, a user may dominate the workload on your RS/6000 SP and may prevent other users from running jobs. The keywords in bold print in Figure 4 on page 49 can be used for managing the job limits.

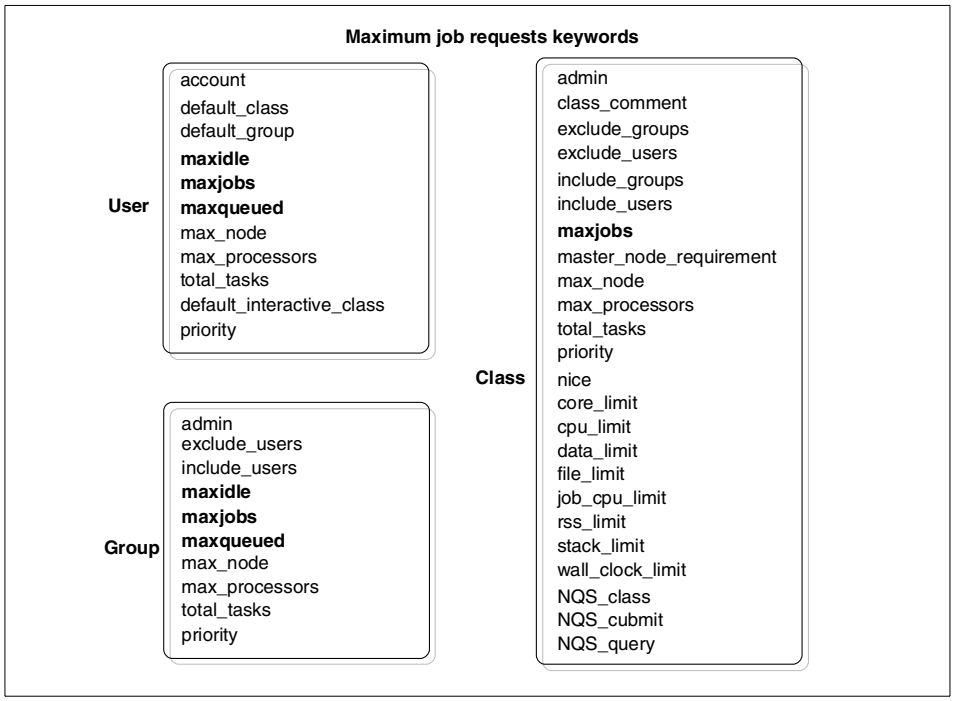


Figure 4. Maximum job requests keywords

To understand the keywords, `maxidle`, `maxjobs`, and `maxqueued`, you have to know the following job states placed by the negotiator after you submit a job:

- Idle

  When a job is placed in the idle state, this job is considered, by the negotiator daemon, to be running on a node or nodes; so, this idle state means that the negotiator daemon is searching nodes for dispatching this job for the user.

- Running

  This state means the job is running on a node or nodes.

- NotQueued

  When a job is placed in the NotQueued state, this job is not considered, by the negotiator daemon, to be running. That is, the negotiator daemon is not searching nodes for this job.

- Pending and Starting

  The Pending state and the Starting state are temporary states between the Idle state and the Running state. The Pending state means that the

negotiator daemon has selected nodes for the job and the job is in the process of starting by the schedd and startd daemons. The Starting state means the job has been dispatched to the target nodes and is in the process of starting on the nodes.

You can control the number of jobs that are placed in the preceding states. The following are the keyword definitions:

- **maxidle** - You can specify the maximum number of jobs placed in the Idle state with this keyword. This keyword can be used in the user or group stanza in the administration file. If you use this keyword in the user stanza, you can limit the number of idle jobs for a user. If you use this keyword in the group stanza, you can limit the number of idle jobs for the group. If you do not put this keyword in the administration file, there is no limit on the number of idle jobs.
- **maxjobs** - You can specify the maximum number of jobs placed in the Running state with this keyword. This keyword can be used in the user, group, and class stanzas in the administration file. If you use this keyword in the user stanza, you can limit the number of jobs the user can run at any time. If you use this keyword in the group stanza, you can limit the number of jobs that group can run at any time. If you use this keyword in the class stanza, you can limit the number of running jobs for that class. If you do not put this keyword in the administration file, the default value is -1. This means there is no limit on the number of running jobs.
- **maxqueued** - You can specify the maximum number of jobs that are placed in a queue, that is, this keyword controls the number of jobs in any of these states: Idle, running, pending, or starting. You can use this keyword in the user and group stanzas in the administration file. If you use this keyword in the user stanza, you can limit the maximum number of jobs allowed in the queue for the user. If you use this keyword in the group stanza, you can limit the maximum number of jobs for that group. The default is no limit on the number of jobs that can be placed in queue.

For a new job to be allowed into the job queue, the following conditions must be satisfied:

- The number of the user's job in the queue (in the Idle, Pending, Starting, and Running states) is less than the value of the maxqueued keyword in the user stanza.
- The number of jobs belonging to the user's group that are in the queue (in the Idle, Pending, Starting and Running states) is less than the value of the maxqueued keyword in the group stanza

- The number of jobs belonging to the user in the Idle state is less than the value of the maxidle keyword in the user stanza.
- The number of jobs belonging to the group in the Idle state is less than the value of the maxidle keyword in the group stanza.

If either of these conditions is not satisfied, the new job is placed in the NotQueued state until one of jobs in the queue changes the state and all of these conditions are satisfied.

Once a job is in the queue, the job will run if the following conditions are satisfied:

- The number of the user's jobs that are running on nodes is less than the value of the maxjobs keyword in the user stanza.
- The number of the group's jobs that are running on nodes is less than the value of the maxjobs keyword in the group stanza.
- The number of class jobs that are running on nodes is less than the value of the maxjobs keyword in the class stanza.

We use an example to describe the use of these keywords. We create an environment with the following characteristics in our LoadLeveler cluster:

- We want to limit the total number of running jobs that are submitted by the users in each group. We want to set this limit to 20.
- We want to limit the total number of jobs that are in the queue for each group. We want to set this limit to 40.
- For the users, kannan1 and bruno1, we want to limit the number of jobs that are in the Idle state to five.
- For the users, bala1 and tani1, we want to limit the number of running jobs to two and the number of jobs in queue to four.

To test this environment, you need the following modifications in the administration file:

```
#############################################################################
# Sample class stanza
#############################################################################
class_first:    type = class
                class_comment = "First Class Users"
                include_users = kannan1 bala1
                admin = kannan1
#
class_second:   type = class
                class_comment = "Second Class Users"
                exclude_users = kannan1
#############################################################################
# Sample group stanz
#############################################################################
group_para:     type = group
                admin = kannan1
                include_users = kannan1 bruno1
                maxjobs = 20
                maxqueued = 40
#
group_serial:   type = group
                admin = bala1
                exclude_users = kannan1
                maxjobs = 20
                maxqueued = 40
#############################################################################
# Sample user stanza
#############################################################################
kannan1:        type = user
                default_class = class_first
                default_group = group_para
                maxidle = 5
#
bala1:          type = user
                default_class = class_first
                default_group = group_serial
                maxjobs = 2
                maxqueued = 4
#
bruno1:         type = user
                default_class = class_second
                default_group = group_para
                maxidle = 5
#
tani1:          type = user
                default_class = class_second
                default_group = group_serial
                maxjobs = 2
                maxqueued = 4
```

3.4.3 Resource limits

You may want to manage resource limits allowed to use by a job step or a process running on RS/6000 SP nodes. AIX has the following user resource limit attributes:

fsize The file size limit that a user's precess can create or extend.

cpu The CPU time limit that a user's process can use.

data The data segment size limit to be used by a user's process.

stack The stack segment size limit to be used by a user's process.

core The core file size limit that a user's process can create

rss The resident size limit that a user's process can grow

You can see these user resource limits in the /etc/security/limits file on each node. AIX has a soft limit and a hard limit for each resource.

In addition to the above process resource limits, LoadLeveler manages two additional limits for managing resources for a job step.

You can specify these limits in the class stanza of the administration file. The following figure shows the keywords to set these limits. The limit keywords are expressed with bold characters:

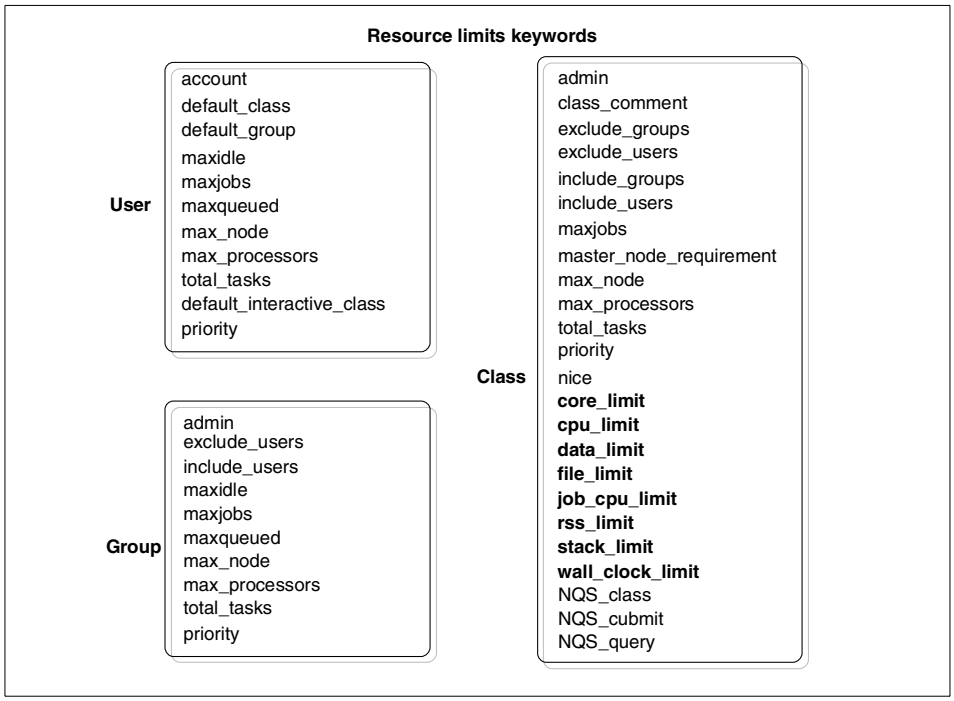


Figure 5. Resource limits keywords

The following are keyword definitions:

core_limit This keyword is for setting the value of the core file size limit (core) in the AIX user resource limit for a process invoked by a job.

cpu_limit This keyword is for setting the value of the CPU time limit (cpu) in the AIX user resource limit for a process invoked by a job.

data_limit This keyword is for setting the value of the data segment size limit (data) in the AIX user resource limit for a process invoked by a job.

file_limit This keyword is for setting the value of the file size limit (fsize) in the AIX user resource limit for a process invoked by a job.

job_cpu_limit This keyword is for setting the total CPU time limit to be used by all processes of a job step. This value is for the entire job step; so, if a job step forks five processes, the total

CPU time consumed by 5 processes cannot consume more than the value specified with this keyword.

rss_limit — This keyword is for setting the value of the resident set size limit (rss) in the AIX user resource limit for a process invoked by a job.

stack_limit — This keyword is for setting the value of the stack segment size (stack) in the AIX user resource limit for a process invoked by a job.

wall_clock_limit — This keyword is for setting the value for elapsed time that a job can be running.

You can specify the hard limit and the soft limit to each keyword using the following syntax:

```
limit_keyword = hardlimit,softlimit
```

To know the syntax and the units of these keywords in detail, refer to the book *IBM LoadLeveler for AIX - Using and Administering,* SA22-7311.

Note that LoadLeveler sets the user process limits by the value of core_limit, cpu_limit, data_limit, file_limit, rss_limit, and stack_limit; however, these limits are managed by AIX.

For the limits of job_cpu_time and wall_clock_limt, LoadLeveler manages the limits per job step. Table 4 shows the actions that occur when a job step limit is exceeded.

Table 4. The action when exceeding Job step limit

Type of job	Exceeding soft limit	Exceeding hard limit
Serial	SIGXCPU or SIGKILL issued	SIGKILL issued
Parallel(non-PVM)	SIGXCPU issued to both the user program and to the parallel daemon	SIGTERM issued
PVM	SIGXCPU issued to the user program	pvm_halt invoked to shut down PVM

We will now show example class stanzas to set resource limits for users in classes.

For class_first, we set the limits listed in Table 5.

Table 5. Limits for class_first

Resource keyword	Hard limit	Soft limit
job_cpu_limit	24 hours	12 hours
wall_clock_limit	24 hours	12 hours

For class_second, we set the limits listed in Table 6.

Table 6. Limits for class_second

Resource keyword	Hard limit	Soft limit
cpu_limit	3 hours	Not assign
data_limit	80 MB	60 MB
file_limit	512 MB	Not assign
stack_limit	80 MB	60 MB
job_cpu_limit	2 hours	1 hours
wall_clock_limit	3 hours	2 hours

Although we do not assign the soft limit for cpu_limit and file_limit, the soft limit is adjusted downward to equal the hard limit.

The following screen output shows the the class stanza keywords used in the admin file to set the limits.

```
#############################################################################
# Sample class stanza
#############################################################################
class_first:    type = class
                class_comment = "First Class Users"
                include_users = kannan1 bala1
                admin = kannan1
                job_cpu_limit = 24:00:00,12:00:00
                wall_clock_limit = 24:00:00,12:00:00
#
class_second:   type = class
                class_comment = "Second Class Users"
                exclude_users = kannan1
                cpu_limit = 00:30:00
                data_limit = 80mb,60mb
                file_limit = 512mb
                stack_limit = 80mb,60mb
                job_cpu_limit = 02:00:00,01:00:00
                wall_clock_limit = 03:00:00,02:00:00
```

A LoadLeveler user can specify these resource limits in the job command file with the same keywords. But, if the hard limit specified in the job command file is greater than the hard limit of the class, the hard limit of the class is used. Also, the hard limit specified in the job command file cannot be greater than the hard limit on the node limit set by the AIX user resource limit. If the hard limit specified in the job command file is greater than the limit on the node, the limit set by the AIX user limit is used.

3.4.4 Job priority

When you have multiple job requests to be processed on your RS/6000 SP, you may have to consider the job priority, that is, which job should be processed earlier and which job should be processed later. Job priority can be an important factor for the workload management on your RS/6000 SP.

Each job needs a different number of nodes; so, scheduling multiple jobs is not simple. However, the LoadLeveler's base priority for jobs in queue is first-in-first-out (FIFO) based on submission time. You can modify this job priority using the user, group, and class stanza keywords.

The user, group, and class stanzas have the priority keyword and the nice keyword. The nice keyword does not affect the LoadLeveler's scheduling; it affects the job priority on a node after dispatching the job.

These keywords can be seen with bold characters in Figure 6 on page 58.

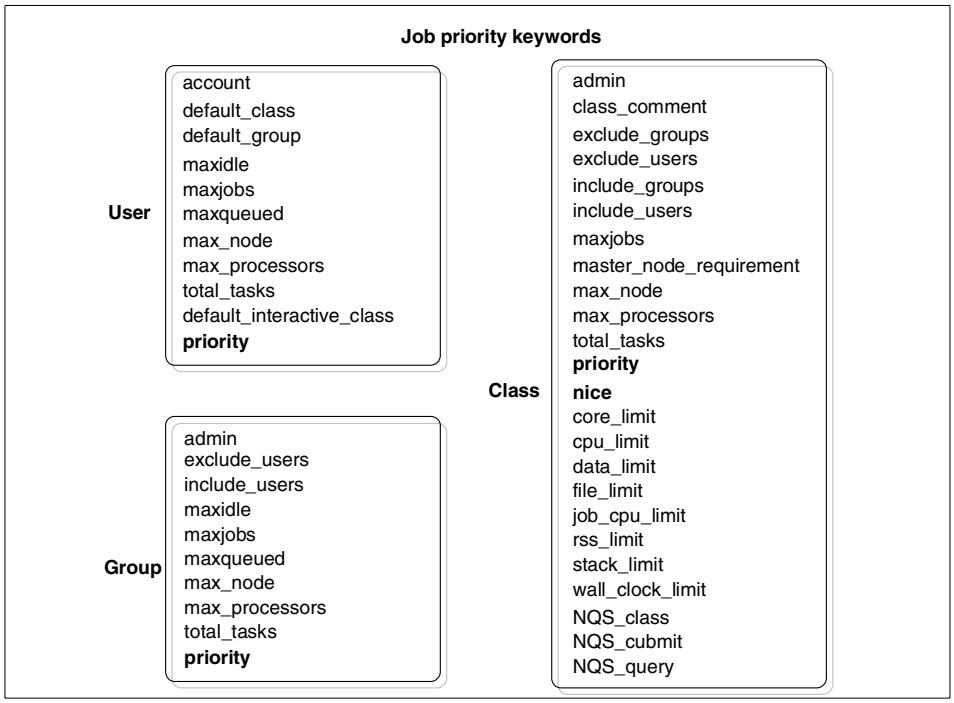


Figure 6. Job priority keywords

A list of the keywords and their definitions follows:

priority You can specify an integer as the value of this keyword in the user, group, and class stanzas. The value specified to this keyword in the user, group, and class stanzas is referred to as the value of UserSysprio, GroupSysprio, and ClassSysprio. UserSysprio, GroupSysprio, and ClassSysprio can be used in the configuration file to change the SYSPRIO keyword value that affects the dispatching priority. If you do not modify the configuration file, these values are not used. If you modify this value in the configuration file, it can affect the LoadLeveler's scheduling priority. The default value of this keyword is 0.

nice You can use this keyword in the class stanza. LoadLeveler increments the initial process priority on AIX using this value. The nice value can be one factor for a job priority on a node after dispatching a job. You can specify an integer from -20 to 20. The higher the number, the lower the priority. The default value is zero.

In addition to these keywords in the administration file, the SYSPRIO keyword in the configuration file and the user_priority keyword in the job command file also need to be considered. The LoadLeveler schedules jobs based on the adjusted system priority, which is determined by using both SYSPRIO (System priority) and user_priority (User priority).

SYSPRIO The value of SYSPRIO is determined by the definition in the configuration file. You can select the factor that affects the value of SYSPRIO in the configuration using the following keywords:

ClassSysprio The value of the priority keyword in the class stanza is referenced as this value.

GroupQueuedJobs The number of job steps currently in the queue submitted by the group, that is, the number of job steps that are in a Running, Starting, Pending, or Idle state.

GroupRunningJobs The number of job steps submitted by the group that are in a Running, Starting, or Pending state.

GroupSysprio The value of the priority keyword in the group stanza is referenced as this value.

GroupTotalJobs The total number of job steps associated with the group. The total number of job steps includes all job steps reported by the `llq` command.

QDate The difference between the UNIX date when the job step enters the queue and the UNIX date when the negotiator starts up. The larger the value, the later the job is submitted. The unit is in seconds.

UserPrio The value of user_priority keyword in the job command file is referenced as this value. This keyword is discussed later. You do not need to use this value.

UserQueuedJobs The number of job steps currently in the queue submitted by the user, that is, the number of job steps that are in a Running, Starting, Pending, or Idle state.

UserRunningJobs The number of job steps submitted by the user that are in a Running, Starting, or Pending state.

UserSysprio The value of the priority keyword in the class stanza is referenced as this value.

UserTotalJobs The total number of job steps associated with the user. Total job steps means all job steps reported by the `llq` command.

You can define the SYSPRIO with these factors. Jobs assigned higher SYSPRIO numbers are considered for dispatch before jobs with lower numbers.

The default definition of the SYSPRIO is the following:

```
SYSPRIO: 0 - (QDate)
```

This default definition creates a FIFO job queue based on submission time because jobs submitted at a later time have lower SYSPRIO than jobs submitted at an earlier time.

If you want to add the value specified in the user, group, and class stanza in the administration file as the factor in addition to the submitted time, you can define the SYSPRIO in the following way:

```
SYSPRIO: ClassSysprio + UserSysprio + GroupSysprio - (QDate)
```

If you want to give more weight to a user who does not have jobs running at the time, it may be good idea to add the number of running jobs in addition to the submitted time:

```
SYSPRIO: 0 - (UserRunningJobs * 10) - (QDate)
```

Note that SYSPRIO is calculated and assigned in the following cases:

- The new job is added to the queue.
- You change the job priority with a command.
- The NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL is elapsed. This keyword is defined in the configuration file.

The default value of NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL is zero; so, if you want to recalculate the SYSPRIO based on the current usage with the specific interval, you should set a non-zero value to this keyword in the configuration file.

user_priority When submitting a job, you can specify the value to this keyword as initial priority among your jobs in the job command file. This value only affects the order among the same users job steps in the same class. It does not affect the order of the job submitted by another user.

With SYSPRIO and the user_priority, LoadLeveler gives the adjusted system priority to jobs in the following way:

1. It first orders jobs by SYSPRIO
2. Among jobs that belong to the same user and the same class, change the order based on the user_priority. This is done by changing the order of the same user's jobs; so this does not affect another user's job.

We show an example of customizing the job priority. In the following example, we give the job priority based on the submission time, but we want to give higher priority to `class_first` jobs than `class_second` jobs in addition to the submission time. We also assign the nice value to jobs of class_first to run with higher priority on nodes. The following screen shows the sample administration file that describes this configuration.

```
############################################################################
# Sample class stanza
############################################################################
class_first:    type = class
                class_comment = "First Class Users"
                include_users = kannan1 bala1
                admin = kannan1
                job_cpu_limit = 24:00:00,12:00:00
                wall_clock_limit = 24:00:00,12:00:00
                priority = 100
                nice = -5
#
class_second:   type = class
                class_comment = "Second Class Users"
                exclude_users = kannan1
                cpu_limit = 00:30:00
                data_limit = 80mb,60mb
                file_limit = 512mb
                stack_limit = 80mb,60mb
                job_cpu_limit = 02:00:00,01:00:00
                wall_clock_limit = 03:00:00,02:00:00
                priority = 60
```

To reflect these keywords to the SYSPRIO, you need to modify the SYSPRIO in the configuration file. We show the modified SYSPRIO in the following screen:

```
#SYSPRIO: 0 - (QDate)
SYSPRIO: (ClassSysprio) - (QDate)
```

3.4.5 Machine priority

When you submit jobs to your RS/6000 SP, LoadLeveler selects nodes for executing jobs. This selection of nodes can also be complex because LoadLeveler has to deal with multiple types of jobs and multiple types of nodes.

When you think of load balancing on your RS/6000 SP environment, you may think you want to offer hints to LoadLeveler about which node should be selected first and which should be selected later. The negotiator assigns and manages a machine priority number to each node. This machine priority is based on the MACHPRIO keyword in the configuration file. You can modify this MACHPRIO keyword to reflect your criteria on node selection.

The nodes assigned higher `MACHPRIO` numbers are considered to run jobs before nodes with lower numbers.

In the configuration file, you can use the following keywords to define the MACHPRIO keyword:

LoadAvg The CPU load on the system is measured using the Berkeley one-minute load average. This load average is the average of the number of processes ready to run or waiting for disk I/O to complete.

Cpus The number of processors of the node.

Speed The relative speed of the node. You can use the speed keyword to specify this value in the machine stanza of the administration file. The default value is 1.

Memory The size of real memory in megabytes of the node.

Virtual Memory The size of available paging space in kilobytes on the node.

Disk The size of free disk space in kilobytes on the filesystem where the executables reside.

Custom Metric This value is determined by the CUSTOM_METRIC keyword in the configuration file. This allows you to give a relative priority number for each node.
You can specify an arbitrary number using the CUSTOM_METRIC keyword to give a relative priority number. Or by specifying an executable and any required arguments to CUSTOM_METRIC_COMMAND keyword in the configuration file, the exit code of this command is assigned to CUSTOM_METRIC. If this command does not exit normally, CUSTOM_METRIC is assigned a value of 1.

This command is forked every time when startd daemon updates the load information to the central manager. This interval is determined by the value of (POLLING_FREQUENCY * POLLS_PER_UPDATE). These values can be modified in the configuration file.

MasterMachPriority A value that is equal to 1 for nodes that are used as master nodes for parallel jobs. This value is equal to 0 for nodes that are not master nodes. In the administration file, you can specify master_node_exclusive = true if you want the node to only be used as a master node for parallel jobs.

When the negotiator assigns a job to a node, the negotiator adds a compensating value to LoadAvg of the node. This default value is 0.5. You can modify this value by specifying a value to NEGOTIATOR_LOADAVG_INCREMENT in the configuration file. If you use a MACHPRIO that is based on LoadAvg, because of this compensating value, the node's priority may be lower than other nodes immediately after a job is scheduled to the node.

We show a sample to assign the machine priority based on the number of processes on the process table. We create a shell script named numofproc.ksh and use the exit code of the numofproc.ksh. This shell script returns the number of processes by counting the lines of the output of the `ps` command as exit code. The following screen shows the numofproc.ksh shell script:

```
#!/bin/ksh
NUMOFPROC=`/usr/bin/ps -ef | wc -l`
(( NUMOFPROC=NUMOFPROC-1 ))
exit $NUMOFPROC
```

To reflect the exit code to MACHPRIO, you need to specify the following keywords in the configuration file:

```
CUSTOM_METRIC_COMMAND   = /u/loadl/numofproc.ksh

MACHPRIO: 0 - (1000 * LoadAvg) - (CustomMetric)
```

Note that all the machines need to have this configuration file and the shell script.

3.4.6 Parallel job

When you use the LoadLeveler to schedule parallel jobs, you need to use keywords that are associated with parallel jobs. In the job command file, you can request a specific adapter for your parallel tasks with the network keyword. To allow this, you need to modify the machine stanza and the adapter stanza in the administration file.

You need to add to the machine stanza with the following keywords:

- adapter_stanzas

  You need to specify using this keyword all adapter names available on this node. You can specify a adapter stanza name list to this keyword. Each adapter stanza name listed here links to an adapter in the adapter stanza.

- alias

  You can specify a blank-delimited list of machine names. In general, if your machine stanza name matches the hostname of the machine, you may not have to specify this keyword. If you specify the machine stanza name other than the hostname such as the hostname corresponding to the switch IP address, you need to specify the hostname using this alias keyword.

  The following screen shows the sample machine stanza we have created to add the adapter stanza:

```
sp4n01.msc.itso.ibm.com: type = machine
                         central_manager = true
                         adapter_stanzas = sp4sw01.msc.itso.ibm.com sp4n01.msc.itso.ibm.com
                         alias = sp4sw01.msc.itso.ibm.com
#
sp4n05.msc.itso.ibm.com: type = machine
                         schedd_host = true
                         adapter_stanzas = sp4sw05.msc.itso.ibm.com sp4n05.msc.itso.ibm.com
                         alias = sp4sw05.msc.itso.ibm.com
#
sp4n06.msc.itso.ibm.com: type=machine
                         adapter_stanzas = sp4sw06.msc.itso.ibm.com sp4n06.msc.itso.ibm.com
                         alias = sp4sw06.msc.itso.ibm.com
#
sp4n07.msc.itso.ibm.com: type = machine
                         adapter_stanzas = sp4sw07.msc.itso.ibm.com sp4n07.msc.itso.ibm.com
                         alias = sp4sw07.msc.itso.ibm.com
#
sp4n08.msc.itso.ibm.com: type = machine
                         adapter_stanzas = sp4sw08.msc.itso.ibm.com sp4n08.msc.itso.ibm.com
                         alias = sp4sw08.msc.itso.ibm.com
#
sp4n15.msc.itso.ibm.com: type = machine
                         schedd_host = true
                         adapter_stanzas = sp4sw15.msc.itso.ibm.com sp4n15.msc.itso.ibm.com
                         alias = sp4sw15.msc.itso.ibm.com
```

We create the adapter stanza for each adapter listed in the `adapter_stanzas` keyword in the machine stanza shown in the preceding screen.

The adapter stanza has the following keywords:

adapter_name Name of the network interface

interface_address The IP address for this interface.

interface_name The IP name corresponding to this adapter.

network_type You can specify the name of the network. This keyword defines the types of networks a user can specify with the network keyword in the job command file. For example, you can specify the name Ethernet for the ethernet work adapters.

switch_node_number This value is defined based on the value of the switch_node_number field in the Node class in the SDR.

You can use the `llextSDR` command to extract adapter information from the SDR. The `llextSDR` command creates adapter and machine stanzas based on

the SDR and writes the stanzas to standard output; so, you can create these stanzas easily by using the output of this command.

Figure 7 and Figure 8 on page 68 show the sample adapter stanzas. These stanzas are created using the output of `llextSDR` command.

```
sp4sw01.msc.itso.ibm.com: type = adapter
        adapter_name = css0
        network_type = switch
        interface_address = 192.168.14.1
        interface_name = sp4sw01.msc.itso.ibm.com
        switch_node_number = 0
#
sp4n01.msc.itso.ibm.com: type = adapter
        adapter_name = en0
        network_type = ethernet
        interface_address = 192.168.4.1
        interface_name = sp4n01.msc.itso.ibm.com
#
sp4sw05.msc.itso.ibm.com: type = adapter
        adapter_name = css0
        network_type = switch
        interface_address = 192.168.14.5
        interface_name = sp4sw05.msc.itso.ibm.com
        switch_node_number = 4
#
sp4n05.msc.itso.ibm.com: type = adapter
        adapter_name = en0
        network_type = ethernet
        interface_address = 192.168.4.5
        interface_name = sp4n05.msc.itso.ibm.com
#
sp4sw06.msc.itso.ibm.com: type = adapter
        adapter_name = css0
        network_type = switch
        interface_address = 192.168.14.6
        interface_name = sp4sw06.msc.itso.ibm.com
        switch_node_number = 5
#
sp4n06.msc.itso.ibm.com: type = adapter
        adapter_name = en0
        network_type = ethernet
        interface_address = 192.168.4.6
        interface_name = sp4n06.msc.itso.ibm.com
#
sp4sw07.msc.itso.ibm.com: type = adapter
        adapter_name = css0
        network_type = switch
        interface_address = 192.168.14.7
        interface_name = sp4sw07.msc.itso.ibm.com
        switch_node_number = 6
#
sp4n07.msc.itso.ibm.com: type = adapter
        adapter_name = en0
        network_type = ethernet
        interface_address = 192.168.4.7
        interface_name = sp4n07.msc.itso.ibm.com
```

Figure 7. The adapter stanza (1 of 2)

```
sp4sw08.msc.itso.ibm.com: type = adapter
        adapter_name = css0
        network_type = switch
        interface_address = 192.168.14.8
        interface_name = sp4sw08.msc.itso.ibm.com
        switch_node_number = 7
#
sp4n08.msc.itso.ibm.com: type = adapter
        adapter_name = en0
        network_type = ethernet
        interface_address = 192.168.4.8
        interface_name = sp4n08.msc.itso.ibm.com
#
sp4sw15.msc.itso.ibm.com: type = adapter
        adapter_name = css0
        network_type = switch
        interface_address = 192.168.14.15
        interface_name = sp4sw15.msc.itso.ibm.com
        switch_node_number = 14
#
sp4n15.msc.itso.ibm.com: type = adapter
        adapter_name = en0
        network_type = ethernet
        interface_address = 192.168.4.15
        interface_name = sp4n15.msc.itso.ibm.com
```

Figure 8. The adapter stanza (2 of 2)

By defining this adapter stanza, users can request css0 and en0 for their parallel tasks with the network keyword in the job command file.

There are some keywords you can define in your machine stanza in addition to the above keywords. These are optional for parallel jobs:

- machine_mode - You can specify the type of job this node can run. You can specify one of the following values:

  batch Specifies this node can be only used for batch jobs

  interactive Specifies this node can be only used for interactive jobs. Only POE is currently enabled to run interactively.

  general Specifies this node can be used for both batch jobs and interactive jobs. This is the default value.

- pvm_root - When you use PVM, you can use this keyword to specify the pathname where the PVM executables reside. The value of this keyword is used to set the environment variable $(PVM_ROOT) required by PVM.
- pool_list - By defining numbers to this keyword, you can specify to which pool this node belongs. Parallel jobs can use this pool number to request

for nodes belonging to a specific pool. This keyword provides compatibility with the function that was previously part of the Resource Manager.

In the user, group, and class stanza, there are some keywords that are used to manage parallel jobs. These keywords are not necessarily required, but you can use them to manage parallel jobs. Figure 9 shows these keywords with bold characters.

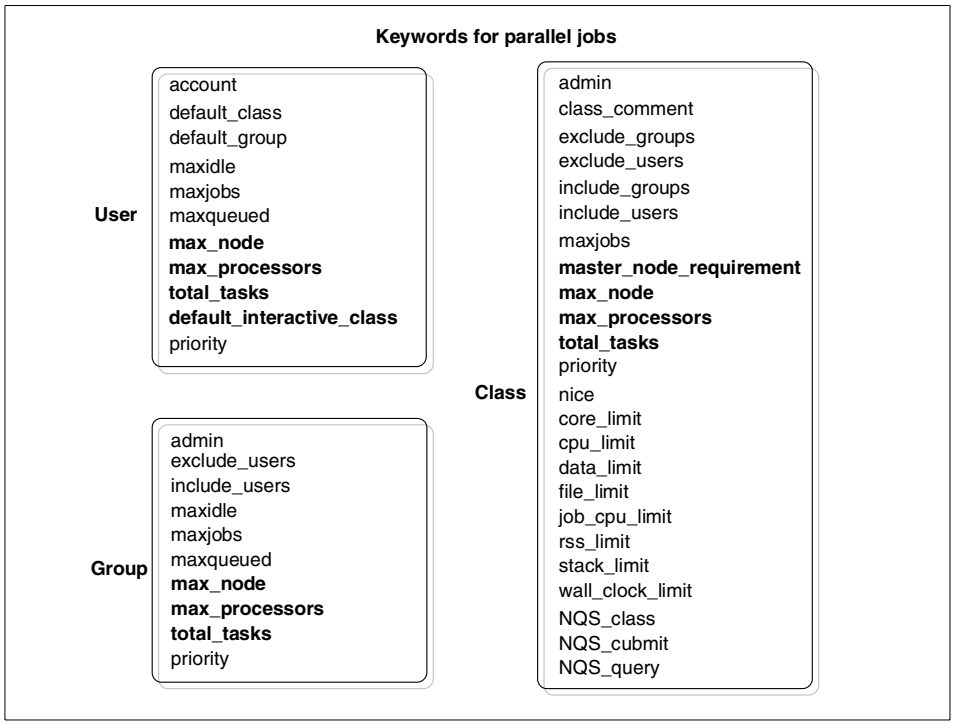


Figure 9. The keywords for parallel jobs

- max_node

  You can specify the maximum number of nodes a user can request for a parallel job. A user can specify the maximum number of nodes and the minimum number of nodes with the node keyword in the job command file when submitting a job. This max_node keyword restricts the maximum number a user can request with the node keyword. You can use this keyword in the user, group, and class stanzas. The default is -1, which means there is no limit.

- max_processors

You can specify the maximum number of processors a user can request for a parallel job in a job command file using the max_processors keyword. You can specify this keyword in the user, group, and class stanzas. The default is -1, which means there is no limit.

- total_tasks

  You can specify the maximum number of tasks a user can request for a parallel job in the job command file using total_task keyword. You can use this keyword in the user, group, and class stanzas. The default is -1, which means there is no limit.

- default _interactive_class

  You can specify the default class name to this keyword in the user stanza. This class name is assigned to the user when the user submits an interactive job and the user does not specify a class using the LOADL_INTERACTIVE_CLASS environment variable. You can specify only one default class name to this keyword.

- master_node_requirement

  This keyword is used in the class stanza. You can specify the value true or false to this keyword. If you specify true in the class stanza, LoadLeveler selects the node as the first node (called the master) that has the master_node_exclusive = true setting in the machine stanza for the parallel jobs in this class.

  Although we do not list the wall_clock_limit keyword here, you should specify the keyword for using the backfill scheduler in the class stanza. If you do not use the keyword in the class stanza, users who submit jobs must specify the wall clock time in their job command file. The backfill scheduler uses the wall_clock_limit to find short jobs that can be completed without delay while large jobs wait for multiple nodes.

We show an example class stanza for running parallel jobs. We want to create the following environment:

- We create a new class for parallel jobs that can be used by all users.
- A job in this class can request a maximum of five nodes.
- A job in this class can request a maximum of 15 tasks.
- Jobs have a 24 hour run time hard limit and a 12 hour run time soft limit.

The following is a class stanza to describe this environment:

```
class_parallel: type = class
                class_comment = "For job_type parallel"
                max_node = 5
                total_tasks = 15
                wall_clock_limit = 24:00:00,12:00:00
```

3.5 Using and managing LoadLeveler

In this section, we will discuss how to use LoadLeveler to submit jobs and manage them effectively within the given resources.

3.5.1 Submitting a job

In this section we will discuss how to submit jobs to LoadLeveler. LoadLeveler will not directly accept the executable file. The LoadLeveler job has to be defined by a job command file. Only after defining a job command file will a user be able to submit the job for scheduling and execution. The job command file almost resembles a shell script.

The job command file contains the LoadLeveler statement with LoadLeveler keywords that describe the job. The job command file can be built using a normal editor or using the Build a job window in the GUI. Once the job command file is prepared, the job can be submitted using the `llsubmit` command or from the xloadl GUI. By default, the name of a job command file ends in .cmd. However, this is not required.

The `llsubmit` command can be issued from any node in the LoadLeveler cluster or from the submit only nodes.

The list of LoadLeveler keywords that are available is shown in Figure 10 on page 72. The LoadLeveler keyword is case-insensitive. Some of the keywords are only required when you want to submit a parallel job. These are discussed in detail later in this chapter.

For now, using examples, we will discuss how to create the job command file for serial and parallel jobs.

3.5.1.1 Serial job command file

Figure 10 on page 72 shows the list of keywords that are available with LoadLeveler and the bold ones indicate few keywords used for submitting a serial job.

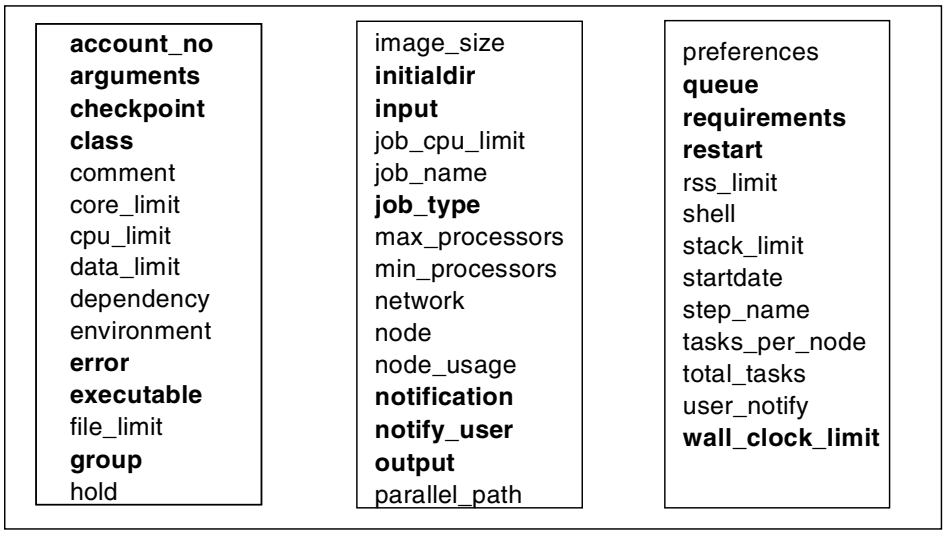


Figure 10. Keywords for submitting a serial job

The keywords are fully explained in the LoadLeveler publication manuals. For the description of each keyword, we suggest that you refer to the LoadLeveler publication, *LoadLeveler for AIX: Using and Administering*, SA22-7311.

For example, let us imagine that the user, bala, belonging to the group, staff, wants to submit an executable script called bala.sh as a serial job to the LoadLeveler. The home directory for user bala is /u/bala. The user bala has been authorized to submit the jobs in the class stanza for small. The job command file, bala.cmd, is edited using a vi editor, and its contents will look like the following:

```
#!/bin/ksh
# @ job_type = serial
# @ executable = bala.sh
## @ arguments =
# @ group = staff
# @ error   = bala.$(Host).$(Cluster).$(Process).err
# @ output  = bala.$(Host).$(Cluster).$(Process).out
# @ initialdir = /u/bala
# @ notify_user = bala
# @ notification = always
# @ checkpoint = no
# @ restart = no
# @ requirements =(Arch == "R6000") && (OpSys == "AIX43")
# @ class = small
# @ account_no = bala
# @ queue
```

Once the job command file is built, it can be submitted to LoadLeveler for execution. The command to submit is `llsubmit`. For our example, run the command `llsubmit bala.cmd` to submit the job to the LoadLeveler:

```
$ llsubmit bala.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.258" has been submitted.
```

The `llsubmit` command returns the job ID. In this case, the job has been submitted to the node, sp4n15, and the job number is 258. Now, to verify the status of the job you have just submitted using the `llq` command:

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.258.0             bala       11/11 08:23 R  50  small        sp4n07

1 job steps in queue, 0 waiting, 0 pending, 1 running, 0 held
```

The output here shows that your job has been submitted for execution on the node sp4n07. Now, to verify that the process has been started in the node, sp4n07, we issue the `ps` command. This can be done by logging on to the node, sp4n07, or by using the `dsh` command from the CWS. Here, the screen output is shown by using the `dsh` command from the CWS.

```
# dsh -w sp4n07
dsh> ps -eaf | grep bala
sp4n07:     bala 13560 15668   0 08:23:58      -  0:00 -ksh -c exec /u/loadl/exe
cute/sp4n15.msc.itso.ibm.com.258.0/bala.sh
sp4n07:     bala 15668 14498   0 08:23:57      -  0:00 LoadL_starter -p 4 -s sp4
n15.msc.itso.ibm.com.258.0
sp4n07:     bala 20176 13560   0 08:24:54      -  0:00 sleep 50
```

The LoadL-startd daemon has started the LoadL_starter process for the user, bala. This LoadL_starter process has forked a child process to execute the script bala.sh.

3.5.1.2 Parallel job command file

Figure 11 shows the list of keywords that are available with the LoadLeveler. The bold ones indicate a few of the keywords used for submitting a parallel job. The Parallel Operating Environment fileset ppe.poe is required to run the parallel job. The Parallel Operating Environment is a priced product of IBM on RS/6000 SP.

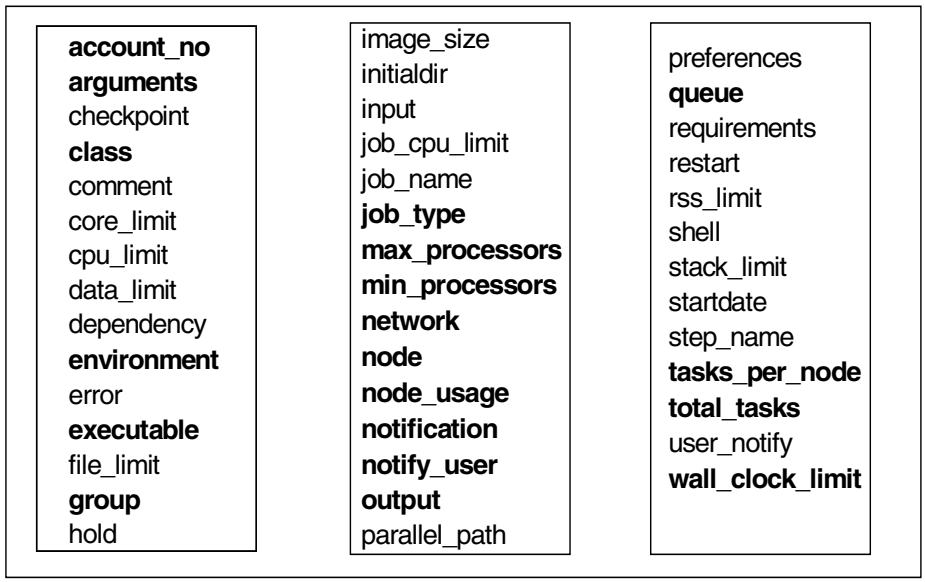


Figure 11. Keywords required to submit a Parallel job

At this point, let us take an example to build a parallel job command file. The user, bala, wants to submit a parallel job using POE. The name of the executable file is mpi_test. The user wants this job to be run on two nodes

using the css0 switch adapter. The job command file for submitting this parallel job to the LoadLeveler looks like the following:

```
### POE job : using switch (css0)

# @ job_type=parallel
# @ notification = always
# @ account_no = bala
# @ environment = COPY_ALL; MP_TIMEOUT=1200;
# @ error   = bala_mpi.$(Host).$(Cluster).$(Process).err
# @ output  = bala_mpi.$(Host).$(Cluster).$(Process).out
# @ wall_clock_limit = 600,500
# @ network.mpi = css0,shared,us
#  Beginning of step1
# @ step_name= step1
# @ node= 2
# @ total_tasks = 2
# @ executable = /bin/poe
# @ arguments = /u/bala/mpi_test     -ilevel 6 -labelio yes
# @ class = small
# @ queue
```

Once the job command file is ready, it can be submitted to the LoadLeveler using the `llsubmit` command. The name of the job command file created here is `bala_mpi.cmd`.

```
$ llsubmit bala_mpi.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.157" has been submitted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.214.0             bala       11/11 10:48 ST 50  small        sp4n07

1 job steps in queue, 0 waiting, 1 pending, 0 running, 0 held
```

The job bala_mpi.cmd is submitted to the scheduler sp4n05 and the job ID is sp4n05.214. This llq output shows that it is running on node sp4n07. This is a parallel job which runs on two nodes. This can be verified using the `llstatus` command. After submitting the parallel job, the `llstatus` command will look like the following:

```
$ llstatus
Name                     Schedd  InQ Act Startd Run LdAvg Idle Arch      OpSys
sp4n01.msc.itso.ibm.com  Down      0   0 Idle     0 0.01     3 R6000     AIX43
sp4n05.msc.itso.ibm.com  Avail     1   0 Idle     0 0.03  9999 R6000     AIX43
sp4n06.msc.itso.ibm.com  Down      0   0 Run      1 0.00  9999 R6000     AIX43
sp4n07.msc.itso.ibm.com  Down      0   0 Run      1 0.00  9999 R6000     AIX43
sp4n08.msc.itso.ibm.com  Down      0   0 Idle     0 0.26  9999 R6000     AIX43
sp4n15.msc.itso.ibm.com  Avail     0   0 Idle     0 3.00  9999 R6000     AIX43

R6000/AIX43                 6 machines      1  jobs      2  running
Total Machines              6 machines      1  jobs      2  running

The Central Manager is defined on sp4n01.msc.itso.ibm.com

The following machine is marked SUBMIT_ONLY
sp4en0.msc.itso.ibm.com

All machines on the machine_list are present.
```

From the `llstatus` command output, you can see that the job is running on two nodes: sp4n06 and sp4n07. Now, to see the process that has been started in the nodes, sp4n06 and sp4n07, use the `dsh` command from the CWS as shown in the next screen:

```
# dsh -w sp4n06,sp4n07 ps -eaf | grep bala
sp4n06:     bala  4576  4824   7 10:48:58      -  0:00 /etc/pmdv2
sp4n06:     bala  4824 13900   3 10:48:56      -  0:00 LoadL_starter -p 4 -s sp4
n05.msc.itso.ibm.com.214.0
sp4n06:     bala 15448  4576  61 10:48:59      -  0:00 /u/bala/mpi_test
sp4n07:     bala  5940 15860  96 10:48:59      -  0:01 /u/bala/mpi_test
sp4n07:     bala 14046  3558   6 10:48:56      -  0:00 LoadL_starter -p 4 -s sp4
n05.msc.itso.ibm.com.214.0
sp4n07:     bala 14232 14046   9 10:48:57      -  0:00 /u/loadl/execute/sp4n05.m
sc.itso.ibm.com.214.0/poe
sp4n07:     bala 15860 14046   4 10:48:58      -  0:00 /etc/pmdv2
```

From the output, you can see the LoadL_starter is started on both the nodes that are assigned for the parallel job. The pmdv2 is the POE process for executing the parallel job. The mpi_test is the executable parallel code that is running on both nodes.

3.5.2 Verifying job status

Once a job has been submitted to the LoadLeveler, the status of the job can be seen using the `llq` command. By default, the `llq` command gives the status of all the jobs in the queue. The following is the sample output of the `llq` command after submitting the serial job.

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.57.0              bala       11/8  12:07 R  50  small        sp4n05

1 job steps in queue, 0 waiting, 0 pending, 1 running, 0 held
```

There are few useful flags that can be used with the `llq` command. The most useful among them are -l and -s flags. The -l flag is used for getting more detailed status information about the job. For more details about the other flags, refer to the LoadLeveler publication, *LoadLeveler for AIX: Using and Administering*, SA22-7311.

The following is the example output for the `llq -l sp4n05.57.0` command, where `sp4n05.57.0` is the serial job that is submitted to LoadLeveler.

```
$ llq -l sp4n05.57.0
================ Job Step sp4n05.msc.itso.ibm.com.57.0 ================
        Job Step Id: sp4n05.msc.itso.ibm.com.57.0
           Job Name: sp4n05.msc.itso.ibm.com.57
          Step Name: 0
  Structure Version: 9
              Owner: bala
         Queue Date: Mon Nov  8 12:07:21 CST 1999
             Status: Running
      Dispatch Time: Mon Nov  8 12:07:21 CST 1999
    Completion Date:
    Completion Code:
      User Priority: 50
       user_sysprio: 80
      class_sysprio: 100
      group_sysprio: 0
    System Priority: -136188
          q_sysprio: -136188
      Notifications: Always
 Virtual Image Size: 1 kilobytes
         Checkpoint:
            Restart: no
     Hold Job Until:
                Cmd: bala.sh
               Args:
                Env:
                 In: /dev/null
                Out: bala.sp4n01.57.0.out
                Err: bala.sp4n01.57.0.err
```

Figure 12. Long listing of llq (Part 1 of 2)

```
Initial Working Dir: /u/bala
         Dependency:
       Requirements: (Arch == "R6000") && (OpSys == "AIX43")
        Preferences:
          Step Type: Serial
     Min Processors:
     Max Processors:
     Allocated Host: sp4n05.msc.itso.ibm.com
         Node Usage: shared
    Submitting Host: sp4n01.msc.itso.ibm.com
        Notify User: bala
              Shell: /bin/ksh
  LoadLeveler Group: staff
              Class: small
     Cpu Hard Limit: 72900 seconds
     Cpu Soft Limit: -1
    Data Hard Limit: -1
    Data Soft Limit: -1
    Core Hard Limit: -1
    Core Soft Limit: -1
    File Hard Limit: -1
    File Soft Limit: -1
   Stack Hard Limit: -1
   Stack Soft Limit: -1
     Rss Hard Limit: -1
     Rss Soft Limit: -1
Step Cpu Hard Limit: -1
Step Cpu Soft Limit: -1
Wall Clk Hard Limit: 1800 seconds
Wall Clk Soft Limit: -1
            Comment:
            Account: bala
         Unix Group: staff
   NQS Submit Queue:
   NQS Query Queues:
Negotiator Messages:
Adapter Requirement:

1 job steps in queue, 0 waiting, 0 pending, 1 running, 0 held
```

Figure 13. Long listing of llq (Part 2 of 2)

The -s flag is used to find out why the job is kept in the NotQueued, Idle or Deferred state. The following example shows the output of the `llq` command where the bold ones indicate the jobs that are in the idle state.

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.46.0              bala       11/8  11:42 R  50  small        sp4n08
sp4n15.101.0             bala       11/8  11:42 R  50  small        sp4n06
sp4n05.47.0              bala       11/8  11:42 R  50  small        sp4n05
sp4n15.102.0             bala       11/8  11:42 R  50  small        sp4n07
sp4n05.48.0              bala       11/8  11:42 R  50  small        sp4n01
sp4n15.103.0             bala       11/8  11:42 R  50  small        sp4n15
sp4n05.49.0              bala       11/8  11:44 I  50  small
sp4n15.104.0             bala       11/8  11:44 I  50  small
sp4n05.50.0              bala       11/8  11:44 I  50  small
sp4n15.105.0             bala       11/8  11:44 I  50  small
```

Let us consider one job, sp4n15.105.0, to find out why it cannot run at this point in time. To find the reasons, let us execute the `llq -s` command. The evaluation for job step, sp4n15.105.0, being kept in the idle state is given in the last few lines of the llq -s output, and it looks like this:

```
$ llq -s sp4n15.105.0
============================== EVALUATIONS FOR JOB STEP 105.0 ==================
============

SUMMARY

The class of this job step is : small. However, no machine in the LoadLeveler cl
uster can start a job step of this class at the present time.

ANALYSIS

   sp4n15.msc.itso.ibm.com : No initiator of job class = small is available on t
his machine at the present time.
   sp4n08.msc.itso.ibm.com : No initiator of job class = small is available on t
his machine at the present time.
   sp4n07.msc.itso.ibm.com : No initiator of job class = small is available on t
his machine at the present time.
   sp4n06.msc.itso.ibm.com : No initiator of job class = small is available on t
his machine at the present time.
   sp4n05.msc.itso.ibm.com : No initiator of job class = small is available on t
his machine at the present time.
   sp4n01.msc.itso.ibm.com : No initiator of job class = small is available on t
his machine at the present time.
```

In this example, user bala has submitted ten jobs in the job class small. The job class small is allowed to run only one job at a time in the nodes sp4n01, sp4n05, sp4n06, sp4n07, sp4n08, and sp4n15. This condition is defined by the LoadL_config.local file in each node by the Class keyword.

The llq output shows that one job of the class small is already running in nodes sp4n01, sp4n05, sp4n06, sp4n07, sp4n08, and sp4n15. These nodes cannot run anymore jobs submitted in the class small as defined in the LoadL_config.local; so, the job sp4n15.105.0 is kept in the idle state. The scheduler periodically checks for the availability of any node to run under this class. Once any node becomes free, the scheduler submits the job to that node for execution.

3.5.3 Changing a job's priority

The administrator or user may need to run a job at a higher or lower priority for various reasons. LoadLeveler gives you the option of changing the priority of a job that has been submitted to the scheduler. This can be done using the `llprio` command. The priority of a job ranges from 0 to 100, with the higher numbers corresponding to higher priorities. To change the priority, you need to supply a + (plus) or - (minus) followed by an incremental value.

The priority of a job can only be changed by the owner of the job or by the LoadLeveler administrator. Also, changing the priority of a running job has no significance. Changing the priority of a job is only significant for the jobs in the queue for dispatching.

Let us consider an example in which the user bala has submitted many jobs to LoadLeveler, and they are in the queue. The `llq` command output shows the jobs that are submitted to the LoadLeveler queue:

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.113.0             bala       11/8  15:03 R  50  small        sp4n07
sp4n05.59.0              bala       11/8  15:03 R  50  small        sp4n06
sp4n15.114.0             bala       11/8  15:03 R  50  small        sp4n05
sp4n05.60.0              bala       11/8  15:03 R  50  small        sp4n01
sp4n15.115.0             bala       11/8  15:03 I  50  small
sp4n05.61.0              bala       11/8  15:03 I  50  small

6 job steps in queue, 2 waiting, 0 pending, 4 running, 0 held
```

Now, the user bala wants the job sp4n05.61.0 to be run before sp4n15.115.0. This can be achieved either by increasing the priority of job sp4n05.61.0 or by decreasing the priority of job sp4n15.115.0. Here, the user has decided to increase the priority of job sp4n05.61 by 50 using the `llprio` command. The following output shows the job status after increasing the priority using the `llprio` command.

```
$ llprio +50 sp4n05.61.0
llprio: Priority command has been sent to the central manager.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.113.0             bala       11/8  15:03 R  50  small        sp4n07
sp4n05.59.0              bala       11/8  15:03 R  50  small        sp4n06
sp4n05.60.0              bala       11/8  15:03 R  50  small        sp4n01
sp4n15.114.0             bala       11/8  15:03 R  50  small        sp4n05
sp4n05.61.0              bala       11/8  15:03 I  100 small
sp4n15.115.0             bala       11/8  15:03 I  50  small

6 job steps in queue, 2 waiting, 0 pending, 4 running, 0 held
```

From the job status, you can see that job sp4n05.61.0 is scheduled before job sp4n15.115.0. Job sp4n05.61.0 will be the next job that will be scheduled for execution as soon as any of the nodes become free.

3.5.4 Holding/releasing a job

The administrator or user may need to hold some of the jobs from running for various reasons. For example, the user or administrator may feel that the job is not so urgent and that it can run during the weekend or overnight. In order to hold a job submitted in the queue, there is a LoadLeveler command, `llhold`, which will place the job in a temporary hold. This command will only affect the jobs that are not running. The owner of a job can put his or her job on hold, or the LoadLeveler administrator can put any job in the queue in a hold state.

For example, user bala has submitted few jobs to the LoadLeveler. Some of these jobs are already running and some of the jobs are in the queue. Now, bala has decided to hold one of the jobs, sp4n15.115.0, which is waiting in the queue. The screen output shows the commands that user bala will execute to monitor the status of the job whether it has moved to the hold state.

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.60.0              bala       11/8  15:03 R  100 small        sp4n01
sp4n15.113.0             bala       11/8  15:03 R  50  small        sp4n07
sp4n05.59.0              bala       11/8  15:03 R  50  small        sp4n06
sp4n15.114.0             bala       11/8  15:03 R  50  small        sp4n05
sp4n15.115.0             bala       11/8  15:03 I  60  small
sp4n05.61.0              bala       11/8  15:03 I  50  small

6 job steps in queue, 2 waiting, 0 pending, 4 running, 0 held
$ llhold sp4n15.115.0
llhold: Hold command has been sent to the central manager.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.60.0              bala       11/8  15:03 R  100 small        sp4n01
sp4n15.113.0             bala       11/8  15:03 R  50  small        sp4n07
sp4n05.59.0              bala       11/8  15:03 R  50  small        sp4n06
sp4n15.114.0             bala       11/8  15:03 R  50  small        sp4n05
sp4n05.61.0              bala       11/8  15:03 I  50  small
sp4n15.115.0             bala       11/8  15:03 H  60  small
```

The screen output here shows the state of job sp4n15.115.0, which was in the “I” (idle) state, has changed to the “H” (hold) state after the execution of the `llhold` command. You can also see that job sp4n05.61.0 is being pushed above in the queue.

The job that is kept in held state can be released using the `llhold -r` command. Only the LoadLeveler administrator can release other users’ jobs that are in the hold state. Now, let us try to release the job that we put on hold in the previous example. The screen output here shows the result of the `llhold` command:

```
$ llhold -r sp4n15.115.0
llhold: Hold command has been sent to the central manager.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.115.0             bala       11/8  15:03 R  60  small        sp4n05
sp4n05.61.0              bala       11/8  15:03 R  50  small        sp4n07

2 job steps in queue, 0 waiting, 0 pending, 2 running, 0 held
```

Immediately after releasing the hold on job sp4n15.115.0, it has started running on node sp4n05 since it is free. If there were no nodes available to run, this job would have been kept in the queue in an idle state.

3.5.5 Cancelling a job

The Jobs submitted to the LoadLeveler queue can be cancelled using the `llcancel` command. This command can be used to cancel running jobs and queued jobs. There are options available to cancel the jobs based on the user ID, hostname, queue host, cluster ID or process ID. The jobs submitted by other users can only be cancelled by the LoadLeveler administrator.

The screen output here shows the job status before and after executing the llcancel command to cancel job sp4n15.117.0.

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.116.0             bala       11/8  15:11 R  50  small        sp4n05
sp4n15.117.0             bala       11/8  15:11 R  50  small        sp4n01
sp4n05.64.0              bala       11/8  15:11 I  50  small
sp4n05.65.0              bala       11/8  15:11 I  50  small
sp4n15.118.0             bala       11/8  15:11 I  50  small

5 job steps in queue, 3 waiting, 0 pending, 2 running, 0 held
$ llcancel sp4n15.117.0
llcancel: Cancel command has been sent to the central manager.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.116.0             bala       11/8  15:11 R  50  small        sp4n05
sp4n05.64.0              bala       11/8  15:11 R  50  small        sp4n01
sp4n05.65.0              bala       11/8  15:11 I  50  small
sp4n15.118.0             bala       11/8  15:11 I  50  small

4 job steps in queue, 2 waiting, 0 pending, 2 running, 0 held
```

3.5.6 Verifying the node's status

The `llstatus` command is used to display the status of the nodes that are available in the LoadLeveler cluster. When you execute the `llstatus` command, it displays one line of information about each node in the LoadLeveler cluster. To get a long listing of the node status information, you can use the `llstatus` command with the -l option.

It is also possible to get a customized format of the status of the node using the -f option. The syntax to create a customized format is explained in detail in the manual *LoadLeveler for AIX: Using and Administering*, SA22-7311.

Sample output of the `llstatus` command is shown in the following screen:

```
$ llstatus
Name                      Schedd  InQ Act Startd Run LdAvg Idle Arch      OpSys
sp4n01.msc.itso.ibm.com   Down      0   0 Idle     0 0.00     0 R6000     AIX43
sp4n05.msc.itso.ibm.com   Avail     2   1 Run      2 0.23  9999 R6000     AIX43
sp4n06.msc.itso.ibm.com   Down      0   0 Run      2 0.16  9999 R6000     AIX43
sp4n07.msc.itso.ibm.com   Down      0   0 Run      2 0.08  9999 R6000     AIX43
sp4n08.msc.itso.ibm.com   Down      0   0 Run      2 0.16  9999 R6000     AIX43
sp4n15.msc.itso.ibm.com   Avail     0   0 Idle     0 3.00  9999 R6000     AIX43


R6000/AIX43                6 machines      2  jobs      8  running
Total Machines             6 machines      2  jobs      8  running

The Central Manager is defined on sp4n01.msc.itso.ibm.com

The following machine is marked SUBMIT_ONLY
sp4en0.msc.itso.ibm.com

All machines on the machine_list are present.
```

The `llstatus` command output here shows that there are six nodes in the LoadLeveler cluster. The scheduler is running on nodes sp4n05 and sp4n15. The startd daemon is running on all six nodes. There are two parallel jobs on sp4n05, and they are running on four nodes: sp4n05, sp4n06, sp4n07, and sp4n08. Each node is running two tasks; so, there are a total of eight tasks running in the LoadLeveler cluster.

The output also shows which node is acting as the Central Manager and which node is acting as the submit-only machine. In our case, sp4n01 is the central manager and sp4en0 is configured as the submit-only machine.

3.5.7 Central manager and alternate central manager

The main function of the Central Manager is to coordinate all LoadLeveler related activities within the LoadLeveler cluster. This node gathers node status and job information from all nodes in the cluster. This information is used to decide on which nodes the jobs in the queue can be executed. One may wonder what will happen to the jobs that are running when the central manager goes down. When the central manager goes down, the status of the jobs is as follows:

- Jobs executing on the other nodes will continue to run.
- Jobs waiting in the queue for execution will remain and get started when the Central Manager or its alternative come up.
- Users may continue to submit jobs from other nodes. These jobs will be in the queue and will get started when the Central Manager comes up.

- Job information is not lost. When the central manager comes up, you will be able to see all the running and waiting jobs that were in the queue.

The name of the Central Manager node can be seen at the end of the `llstatus` command output. In order to avoid the central manager being a single point of failure, we can designate any node in the cluster as the alternate central manager. The alternate central manager node will act as a standby for the central manager node.

Whenever the central manager node goes down or does not respond to the alternate central manager daemon due to network failures, the alternate central manager will start acting as the central manager. The alternate central manager will only become the central manager if there is no response from the central manager for the timeout period that is set by the keywords CENTRAL_MANAGER_HEARTBEAT_INTERVAL and CENTRAL_MANAGER_TIMEOUT in the configuration file.

In the test environment, we configured sp4n01 as the central manager node and sp4n06 as the alternate central manager node. Let us simulate a condition of bringing down the central manager and see how the alternate central manager behaves. The central manager can be brought down using the `llctl -h sp4n01 stop` command:

```
$ llctl -h sp4n01 stop
llctl: Sent stop command to host sp4n01.msc.itso.ibm.com.
```

After stopping the daemons in the central manager, if you immediately issue the `llq` or `llstatus` command to check the job or machine status, the system will report an error. The error will look like the following:

```
$ llq
11/09 15:45:11  llq: 2539-463 Cannot connect to sp4n01.msc.itso.ibm.com "LoadL_n
egotiator" on port 9614. errno = 22
11/09 15:45:11  llq: 2539-463 Cannot connect to sp4n06.msc.itso.ibm.com "LoadL_n
egotiator" on port 9614. errno = 22
11/09 15:45:11  llq: 2539-463 Cannot connect to sp4n01.msc.itso.ibm.com "LoadL_n
egotiator" on port 9614. errno = 22
llq: There is currently no job status to report.
$ /llstatus
11/09 15:45:14  llstatus: 2539-463 Cannot connect to sp4n01.msc.itso.ibm.com "Lo
adL_negotiator" on port 9614. errno = 22
11/09 15:45:14  llstatus: 2539-463 Cannot connect to sp4n06.msc.itso.ibm.com "Lo
adL_negotiator" on port 9614. errno = 22
11/09 15:45:14  llstatus: 2539-463 Cannot connect to sp4n01.msc.itso.ibm.com "Lo
adL_negotiator" on port 9614. errno = 22
llstatus: There is currently no machine status to report.
```

Node sp4n06, which has been defined as the alternate central manager, will be trying to communicate with the LoadLeveler daemons on node sp4n01. Since the daemons are not running, it will never get a response back from node sp4n01. Node sp4n06 will wait until the timeout period. Once the timeout period is reached, node sp4n06 will start the negotiator daemon and start contacting all the nodes in the cluster. This daemon will take some time to understand the status of all the nodes in the cluster. The time taken depends on the number of nodes in the cluster. If you try to run the `llstatus` command during this period, it may only show some nodes; there is no need to panic. The screen output here shows the result of the `llstatus` command after the alternate central manager has received all the nodes in the cluster.

```
$ llstatus
Name                     Schedd  InQ Act Startd Run LdAvg Idle Arch      OpSys

sp4n05.msc.itso.ibm.com  Avail     0   0 Idle     0 0.04  9999 R6000     AIX43
sp4n06.msc.itso.ibm.com  Down      0   0 Run      1 1.00  9999 R6000     AIX43
sp4n07.msc.itso.ibm.com  Down      0   0 Run      1 0.97  9999 R6000     AIX43
sp4n08.msc.itso.ibm.com  Down      0   0 Run      1 0.97  1429 R6000     AIX43
sp4n15.msc.itso.ibm.com  Avail     1   1 Idle     0 3.04  9999 R6000     AIX43

R6000/AIX43                5 machines      1  jobs      3  running
Total Machines             5 machines      1  jobs      3  running

The Central Manager is defined on sp4n01.msc.itso.ibm.com, but is unusable
Alternate Central Manager is serving from sp4n06.msc.itso.ibm.com

The following machine is marked SUBMIT_ONLY
sp4en0.msc.itso.ibm.com

The following machine is absent
sp4n01.msc.itso.ibm.com
```

It is clear from the output that the jobs that were running prior to bringing down the central manager continue to run.

3.5.8 Using the LoadLeveler GUI

LoadLeveler also provides a Motif-based GUI that can be used for building job command files and submitting and managing jobs. The GUI can be started using the following command:

```
/usr/lpp/LoadL/full/bin/xloadl &
```

To start the GUI on a submit-only machine, issue the following command:

```
/usr/lpp/LoadL/so/bin/xloadl_so &
```

Here, we will discuss the basic steps to start and use the LoadLeveler GUI. The step-by-step instructions for performing various tasks using the GUI are explained in detail in the publication *LoadLeveler for AIX: Using and Administering*, SA22-7311. We suggest you to refer to this manual for more details.

Figure 14 on page 88 shows the GUI that appears when xloadl_so is started on the submit-only machine. There are three windows that display the information regarding the jobs, machines, and messages.

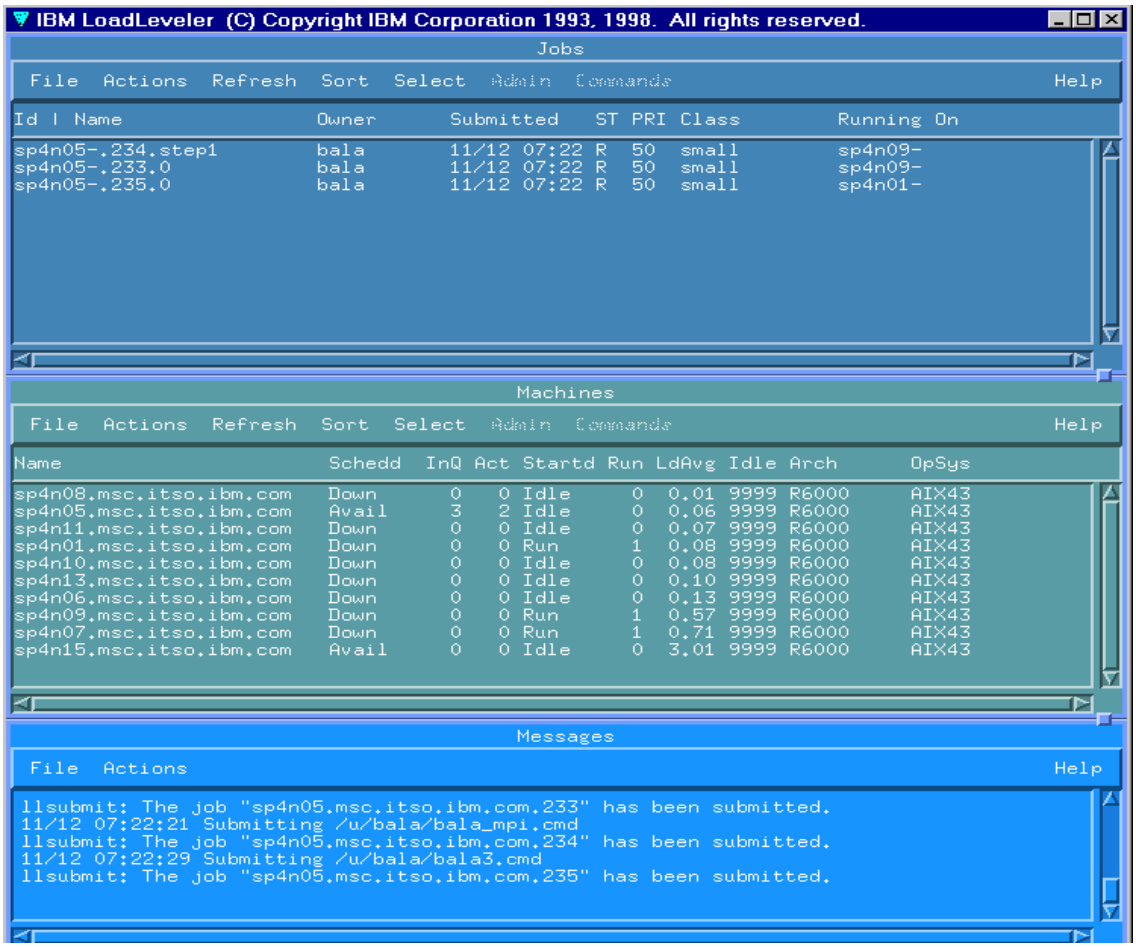


Figure 14. LoadLeveler GUI Main window

The job window gives the job status of all the jobs. This output will be the same as what you get when you run the `llq` command from the shell prompt. The menu bar in the jobs window has many options related to actions that can be performed on the jobs. By default, the job status screen will be refreshed every 120 seconds.

The machine window gives the machine status of all the nodes in the LoadLeveler cluster. The output will look the same as what you get when you run the `llstatus` command from the shell prompt. By default, the machine status will be refreshed every 60 seconds.

The message window gives the output of the commands you run from the job menu bar or the machines menu bar. The message window in Figure 14 on page 88 shows that three jobs were submitted.

3.5.8.1 Submitting a job from GUI

You can use the GUI to submit jobs. To submit a job, select the File menu in the job or machines window, and choose **Submit a Job**. This opens up the Submit a Job dialog as shown in Figure 15. Select the job command file from the list, and click on the **Submit** button to submit the job. Once you click on the submit button, the job is submitted to LoadLeveler, and the job ID will be displayed in the message window. The job and machine status may need to be refreshed in order to get the current status of the job and the nodes on which the job is running.

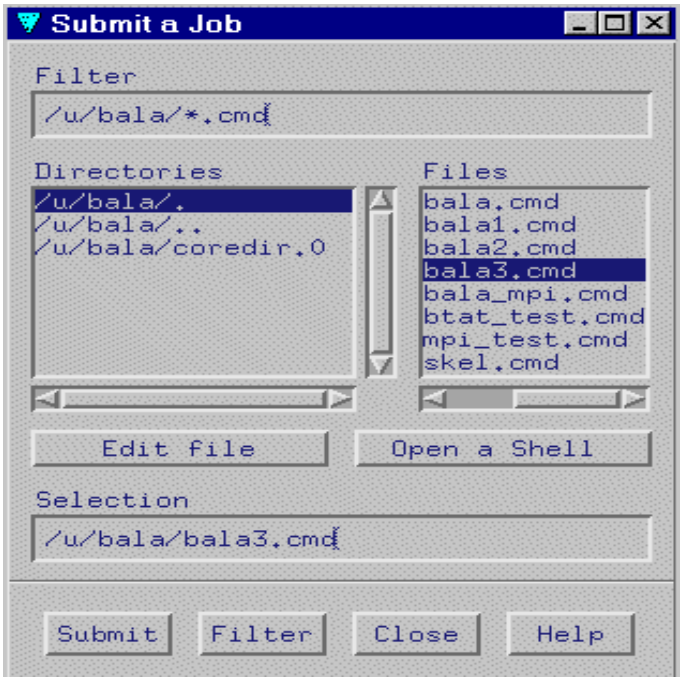


Figure 15. Submit a Job dialog

3.5.8.2 Building the job command file

The LoadLeveler GUI can be used to create the job command file. This is a very useful tool for first timers for building a LoadLeveler job command file. The GUI can be used to create job command files for serial, parallel, and PVM jobs. Let us take an example of creating a serial job command file to run

the script bala.sh. To create a job command file, select the **File** menu in the job status window and click on **Build a Job**. Now, you can choose the job type for building the job command file. For our example, to build a serial job file, click on **Serial**. This opens up the Build a Job dialog box as shown in Figure 16.

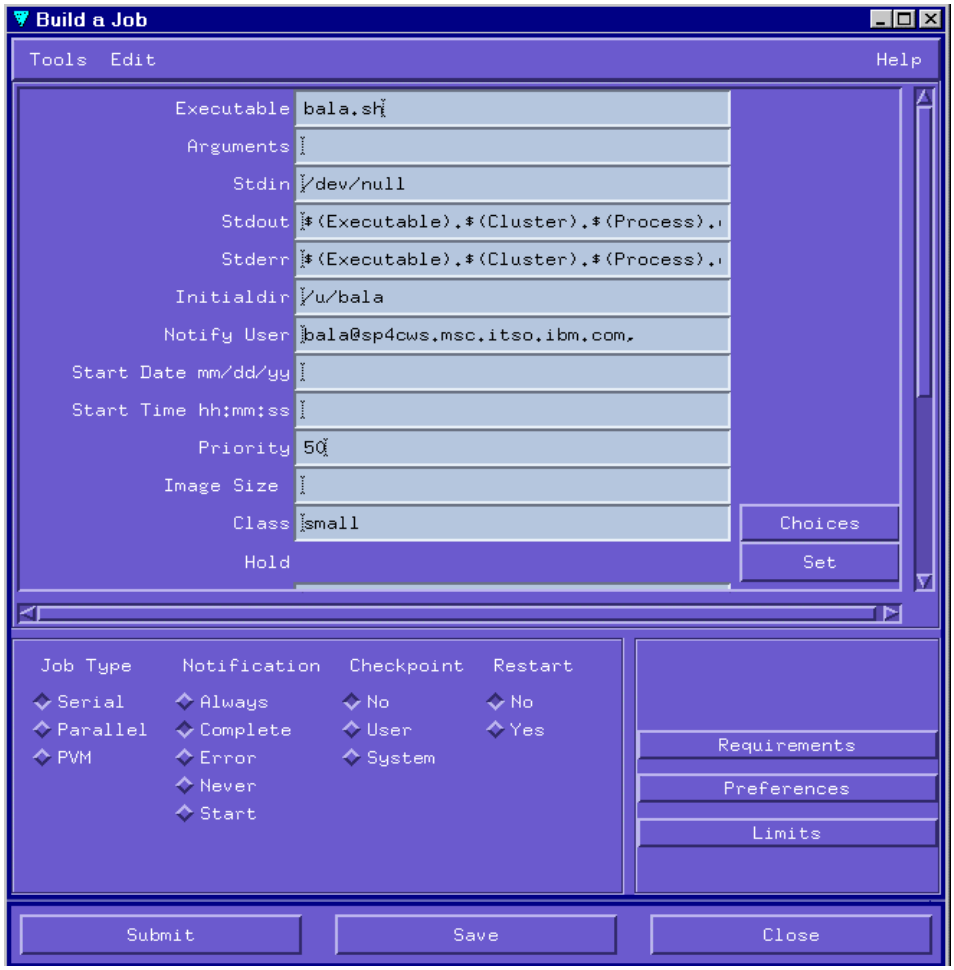


Figure 16. Build a serial job command file

The minimum fields that must be edited to submit the bala.sh script as a serial job are the executable and class fields. After editing these fields, click the **Submit** button to submit the job to LoadLeveler. If you want to save the job command file, click **Save**. This will open a dialog box for entering the

filename you want to save. The same method can be used for building the parallel jobs.

3.6 Checkpointing

The jobs submitted to the LoadLeveler have three types of completion. In a normal successful completion, job execution is aborted by LoadLeveler because wall clock time specified by the job is completed, or there is an abrupt end to the job due to node failure. In the case of a normal successful completion, no further action is necessary because the job is completed through normal execution. In the other two cases, the job has to be restarted by the user. In general, the job is restarted from the beginning by resubmitting the job to LoadLeveler. But, it would be ideal if the job was restarted from the point of failure either on the same node or on a different node. Restarting from the point of failure will save time from the user's point of view and resources from the administrator's point of view.

Checkpointing is the method by which you can periodically save the state of a job so that, if the job does not complete, it can be restarted from the saved state. LoadLeveler provides the facility for checkpointing. There are some keywords to be defined in the job command file to enable checkpointing and restart the job from the saved state.

A checkpoint can either be initiated by the system or by the user. System-initiated checkpointing is only available for serial jobs. User-initiated checkpointing is available for both serial and parallel jobs.

The serial jobs must use the LoadLeveler checkpoint API call to request for used initiated checkpointing. You can also enable both the user and system-initiated checkpoints for serial jobs.The system-initiated one automatically checkpoints your program at preset intervals.

The POE jobs have to use the parallel environment (PE) checkpointing API for requesting user initiated checkpoint. The LoadLeveler checkpoint API does not support checkpointing parallel jobs. The checkpointing for parallel PVM jobs is not supported.

The keywords for checkpointing are:

```
checkpoint = user_initiated | system_initiated | no
```

User-initiated
The user's application program determines when the checkpoint is to be taken. This type of checkpointing is available for both serial and parallel jobs.

System-initiated
The checkpoint is taken at administrator-defined intervals. This is defined by MIN_CKPT_INTERVAL and MAX_CKPT_INTERVAL in the LoadL_config file or in the LoadL_config.local file. The default values are 900 and 7200 seconds. The first checkpoint is taken at MIN_CKPT_INTERVAL. The second checkpoint will be taken at MIN_CKPT_INTERVAL * 2. The third checkpoint will be taken at twice the interval of the second checkpoint, that is, MIN_CKPT_INTERVAL *4. In this fashion, the checkpoint interval will be doubled with the previous checkpoint interval until it reaches the MAX_CKPT_INTERVAL.

The environment variables required for checkpointing are:

- CHKPT_STATE = enable | restart
- CHKPT_FILE
- CHKPT_DIR

The CHKPT_STATE environment variable allows you to enable checkpointing. The enable option enables the checkpointing. The restart option restarts the executable from an existing checkpoint. If you set the checkpoint equal to *no* in the job command file, no checkpoints will be taken regardless of the value of the CHKPT_STATE variable.

The CHKPT_FILE and CHKPT_DIR environment variables help you manage your checkpoint files. If you want the job to be restarted on a different node, you must define the directory in a global file system. If you define the directory in the local file system, the job can only be restarted in the same node in which it was running.

The following are a few points to keep in mind when you want to enable checkpoint:

- Set the CHKPT_FILE and CHKPT_DIR variables to a global file system so that you can migrate the job to a different node.
- Check that you have sufficient space in the CHKPT_DIR directory. LoadLeveler will try to reserve enough disk space for the checkpoint file when the job is started, but it is your responsibility to ensure that sufficient space is available.
- Set the checkpoint file size to maximum. This is required to ensure that the creation of the checkpoint file is not prevented due to system limits. This can be done by setting the file_limit to *unlimited* for your job class.
- The programs need to be compiled with one of the following supported compilers:

- For Fortran: xlf 5.1.1 or later releases
- For C and C++: xlC 3.6.x or Visual Age C, C++ (VAC++) 4.1, and later releases

- To enable user-initiated checkpointing, check that the programs are linked to the LoadLeveler libraries libchkrst.a and chkrst_wrap.o. To ensure that your checkpointing jobs are linked correctly, compile your programs using the compile script found in the bin directory of the LoadLeveler release directory.

3.7 LoadLeveler APIs

LoadLeveler provides Application Programming Interface (API) support for customers to develop site-specific LoadLeveler functions and commands. This allows the user's application programs to communicate with the LoadLeveler environment.

Let us try to understand the need for this API and how we can use it. LoadLeveler gives you various executables to submit, query, manage jobs, and generate reports about the usage of the resources in the LoadLeveler cluster. There may be cases in which some of the required functions in your customer environment cannot be met by these executables. In such situations, you can implement them by writing your own code using this API to interface with the LoadLeveler environment.

For example, the format of the accounting report generated using the `llsummary` command may not meet your requirements, and you may want to have the report generated in a different format. Generating the report in the required format can be done using two methods:

- Take the output using the `llsummary` command, which comes as part of LoadLeveler, and use the tools, such as grep and awk, to convert it to the required format.
- Write your own program using the GetHistory report generation subroutine to generate the reports.

Preparing the accounting report in the customized format is one example of using the LoadLeveler API. There may be many other requirements to submit, query, or manage jobs in your environment that may need to be implemented. In such situations, this API will be useful for creating your own executables to communicate with the LoadLeveler environment.

The LoadLeveler Version 2 Release 1 provides the following APIs, which can be used to integrate to the basic LoadLeveler functions:

- Accounting API
- Serial Checkpointing API
- Submit API
- Data Access API
- Parallel Job API
- Job control API
- Query API
- User Exits

There are a few sample programs available in the /usr/lpp/LoadL/full/samples directory for using the various API calls.

The llapi.h header file defines all the API data structures and subroutines. The libllapi.a library contains all the LoadLeveler API subroutines. These libraries are not thread-safe and, thus, should not be linked to the threaded applications. The syntax and usage of the subroutines is discussed in detail in the manual *LoadLeveler for AIX: Using and Administering*, SA22-7311.

Example of using the API

Now, we will write a program using the LoadLeveler API. This example queries and obtains the list of current jobs from the negotiator. It then prints the step ID and the name of the first allocated host. We edited the program llapi-test.c to implement this example, and the code looks like the following:

```
#include "llapi.h"

main(int argc,char *argv[])
{
  LL_element *queryObject=NULL, *job=NULL;
  int rc, num, err, state;
  LL_element *step=NULL, *machine = NULL;
  char *id=NULL, *name=NULL;

        /* Initialize the query for jobs */
        queryObject = ll_query(JOBS);

        /* I want to query all jobs */
        rc = ll_set_request(queryObject,QUERY_ALL,NULL,NULL);

        /* Request the objects from the Negotiator daemon */
        job = ll_get_objs(queryObject,LL_CM,NULL,&num,&err);

        /* Did we get a list of objects ? */
        if (job == NULL) {
                printf("  ll_get_objs returned a NULL object.\n");
                printf("  err = %d\n",err);
        }
        else {
                /* Loop through the list and process */
                printf(" RESULT: number of jobs in list = %d\n",num);
                while(job) {
                        rc = ll_get_data(job,LL_JobGetFirstStep, &step);
                        while (step) {
                            rc = ll_get_data(step,LL_StepID, &id);
                            rc = ll_get_data(step,LL_StepState,&state);
                            printf(" RESULT: step id: %s\n",id);
                            if (state == STATE_RUNNING) {
                               rc = ll_get_data(step,LL_StepGetFirstMachine, &machine);
                               rc = ll_get_data(machine,LL_MachineName, &name);

                               printf(" Running on 1st assigned host: %s.\n",name);
                               free(name);
                            }
                            else
                               printf("  Not Running.\n");
                            free(id);
                            rc=ll_get_data(job,LL_JobGetNextStep,&step);
                        }
                        job = ll_next_obj(queryObject);
                }
        }

        /* free objects obtained from Negotiator */
        rc = ll_free_objs(queryObject);

        /* free query element */
        rc = ll_deallocate(queryObject);
}
```

Once the file has been edited using your favorite editor, we need to compile and link this program using the `make` command. The Makefile to compile and link this program to the llapi library looks like:

```
*********************************************************************
#
#   5765-145 (C) COPYRIGHT IBM CORP 1993
#   5765-227 (C) COPYRIGHT IBM CORP 1993
#   5765-228 (C) COPYRIGHT IBM CORP 1993
#
#*********************************************************************
# Module Name     : Makefile for accounting api sample programs
#*********************************************************************

CC = cc

RELEASE_DIR=/usr/lpp/LoadL/full

LIB_DIR=$(RELEASE_DIR)/lib
INCLUDE_DIR=$(RELEASE_DIR)/include

CFLAGS =  -I$(INCLUDE_DIR)

LDFLAGS = -L$(LIB_DIR)  -lllapi

all: llapi-test

llapi-test: llapi-test.o
        $(CC) $(CFLAGS) -o llapi-test llapi-test.o  $(LDFLAGS)
```

Next, run the `make` command to create the executable llapi-test. Now, let us run this executable that we had created and see how this piece of code has communicated with the negotiator and returned the results. We can see the output of the `llq` command and the output of our test code llapi-test.

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.513.0             bala       11/17 09:11 R  50  small        sp4n05
sp4n05.711.0             bala       11/17 09:11 R  50  small        sp4n01
sp4n15.514.0             bala       11/17 09:12 I  50  small

3 job steps in queue, 1 waiting, 0 pending, 2 running, 0 held

$ ./llapi-test
 RESULT: number of jobs in list = 3
 RESULT: step id: sp4n15.msc.itso.ibm.com.514.0
  Not Running.
 RESULT: step id: sp4n15.msc.itso.ibm.com.513.0
 Running on 1st assigned host: sp4n05.msc.itso.ibm.com.
 RESULT: step id: sp4n05.msc.itso.ibm.com.711.0
 Running on 1st assigned host: sp4n01.msc.itso.ibm.com.
$
```

From this example, we can see how we captured the job status using the LoadLeveler API subroutines. Using the API, customers can customize the output formats. For example, each site would like to have a different style of accounting report generated from LoadLeveler; in such situations, you can use the accounting API.

Using the LoadLeveler API, you can develop your own scheduling applications that may be necessary for your environment

There are two such schedulers developed using the LoadLeveler API: EASY-LL is one of the schedulers developed jointly by IBM and the Cornell Theory Centre. The other one is the MAUI scheduler developed by the Maui High Performance Computing Center.

EASY-LL is a scheduling mechanism that provides FIFO scheduling for both large and small by means of a *backfill* algorithm. This algorithms allows small jobs to make use of the idle nodes without delaying the execution of the large jobs that need more nodes. The backfill algorithm has since been made available in the IBM LoadLeveler Version 2 Release 1. For more details about EASY-LL, refer to the Web site `http://www.tc.cornell.edu`.

The MAUI scheduler was developed to specifically address the job scheduling requirements of the Maui High Performance Computing Centre. This scheduler is designed to integrate fair share scheduling and backfill capabilities with LoadLeveler infrastructure. For more details about this scheduler, refer to the Web site `http://www.mhpcc.edu`.

3.8 LoadLeveler accounting

LoadLeveler has a feature to collect accounting information about the jobs that are run in the LoadLeveler cluster. This will be useful for analyzing the usage of all the nodes in the cluster. In some cases, the customer installations allow external customers or users to use the resources on a chargeable basis. In such situations, there is a requirement to find out the usage of the resources for charging purposes. The LoadLeveler accounting system can be enabled to collect the various resources used by all the users in the cluster.

The accounting information for a completed serial job is determined by the accumulated resources consumed by that job in the node. In case of a completed parallel job, the information is gathered by accumulating the resources from all the nodes that ran the job.

LoadLeveler can collect a variety of accounting information based on your requirements. The various types of job resource data that can be collected are:

- Job Resource data on serial parallel jobs
- Job Resource data based on the machines
- Job Resource data based on events
- Job Resource data based on user accounts

3.8.1 Configure LoadLeveler accounting

The steps to configure LoadLeveler accounting are as follows:

1. Log in as `root` user in the CWS.
2. To start collecting the accounting information, the following keywords have to be enabled in the LoadL_config file.

```
#
# Specify accounting controls
#
ACCT                    = A_ON A_DETAIL
ACCT_VALIDATION         = $(BIN)/llacctval
GLOBAL_HISTORY          = $(SPOOL)

#               LoadL_StartD Macros

JOB_LIMIT_POLICY        = 120
JOB_ACCT_Q_POLICY       = 300
```

The keyword ACCT = A_ON A_DETAIL where A_ON enables accounting and A_DETAIL collects detailed information.

The ACCT_VALIDATION keyword is the routine that performs validation. You can also supply your own validation routine by specifying this keyword in the configuration file. For example, if you need to track the use of the resources based on user account, the users must specify the account_no keyword in their job command file.

The GLOBAL_HISTORY keyword defines the directory in which the accounting data file needs to be collected. By default, this is defined in the spool subdirectory under the loadl's home directory. In the lab, the loadl's home directory is /u/loadl; so, the accounting data file is created under /u/loadl/spool.

JOB_LIMIT_POLICY and JOB_ACCT_Q_POLICY are used to control how often the startd daemon collects resource consumption data on running jobs and how often the job_cpu_limit is checked.

3. Propagate the LoadL_config file to all the nodes in the cluster using the command shown in the following screen:

```
# dsh -w sp4n01,sp4n05,sp4n06,sp4n07,sp4n08,sp4n15
dsh> /var/sysman/supper update user.admin
sp4n01: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n01: File Changes:  1 updated, 0 removed, 0 errors.
sp4n05: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n05: File Changes:  1 updated, 0 removed, 0 errors.
sp4n06: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n06: File Changes:  1 updated, 0 removed, 0 errors.
sp4n07: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n07: File Changes:  1 updated, 0 removed, 0 errors.
sp4n08: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n08: File Changes:  1 updated, 0 removed, 0 errors.
sp4n15: Updating collection user.admin from server sp4en0.msc.itso.ibm.com.
sp4n15: File Changes:  1 updated, 0 removed, 0 errors.
dsh>
```

4. To put the changes into effect, the LoadLeveler daemons have to reread the config files. This can be done using the following command:

```
# su - loadl
$ llctl -g reconfig
llctl: Sent reconfig command to host sp4n01.msc.itso.ibm.com.
llctl: Sent reconfig command to host sp4n06.msc.itso.ibm.com.
llctl: Sent reconfig command to host sp4n05.msc.itso.ibm.com.
llctl: Sent reconfig command to host sp4n07.msc.itso.ibm.com.
llctl: Sent reconfig command to host sp4n08.msc.itso.ibm.com.
llctl: Sent reconfig command to host sp4n15.msc.itso.ibm.com.
$
```

3.8.2 Collect accounting data

To create an accounting report, we first need to collect the data from all the nodes in the LoadLeveler cluster. This can be collected using the `llacctmrg` command. This command can only be executed by the LoadLeveler administrator. The history information is only collected and stored in the scheduling nodes. When you run the `llacctmrg` command without any options, you can see the history file received only from the scheduler nodes. In the test environment, the nodes sp4n05 and sp4n15 are the scheduling nodes. The history file available in these two nodes will be received and merged as one global history file. The output of the `llacctmrg` command will look like the following:

```
$ llacctmrg
11/15 10:07:16  llacctmrg: 2539-463 Cannot connect to sp4n01.msc.itso.ibm.com "L
oadL_schedd" on port 9605. errno = 79
llacctmrg: 2512-021 Cannot connect to LoadL_schedd on host sp4n01.msc.itso.ibm.c
om.
11/15 10:07:16  llacctmrg: 2539-463 Cannot connect to sp4n06.msc.itso.ibm.com "L
oadL_schedd" on port 9605. errno = 79
llacctmrg: 2512-021 Cannot connect to LoadL_schedd on host sp4n06.msc.itso.ibm.c
om.
llacctmrg: History transferred successfully from sp4n05.msc.itso.ibm.com (524304
 bytes).
11/15 10:07:17  llacctmrg: 2539-463 Cannot connect to sp4n07.msc.itso.ibm.com "L
oadL_schedd" on port 9605. errno = 79
llacctmrg: 2512-021 Cannot connect to LoadL_schedd on host sp4n07.msc.itso.ibm.c
om.
11/15 10:07:17  llacctmrg: 2539-463 Cannot connect to sp4n08.msc.itso.ibm.com "L
oadL_schedd" on port 9605. errno = 79
llacctmrg: 2512-021 Cannot connect to LoadL_schedd on host sp4n08.msc.itso.ibm.c
om.
llacctmrg: History transferred successfully from sp4n15.msc.itso.ibm.com (376212
 bytes).
```

So, instead of giving the `llacctmrg` command without any options, you can use the -h option with the hostlist of scheduler nodes. For example, to receive the history files from nodes sp4n05 and sp4n15, use the following command:

```
$ llacctmrg -h sp4n05 sp4n15
```

The global history file globalhist.<timestamp> will be created in the directory as defined by the GLOBAL_HISTORY variable in the LoadL_config file. The default is the spool directory /u/loadl/spool.

3.8.3 Generate accounting report

From the collected data, you can create the report using the `llsummary` command. This command has many flags for collecting the report in various formats. The default output includes summaries of the following data:

- The number of jobs and the total CPU usage on a per-user basis.
- The number of jobs and the total CPU usage per class.
- The number of jobs and the total CPU usage per group.
- The number of jobs and the total CPU usage per account number.

The default output without any option will look like the following:

```
$ llsummary globalhist.199911151007
        Name   Jobs   Steps        Job Cpu    Starter Cpu     Leverage
       tani1     18      24     0+00:03:08     0+00:00:24          7.8
        bala     28      28     0+01:51:11     0+00:00:13        513.2
      bruno1     15      21     0+01:39:30     0+00:00:12        497.5
     kannan1     16      17     0+02:45:05     0+00:00:16        619.1
       TOTAL     77      90     0+06:18:55     0+00:01:07        339.3

       Class   Jobs   Steps        Job Cpu    Starter Cpu     Leverage
class_parall     11      11     0+00:01:40     0+00:00:19          5.3
       small     28      28     0+01:51:11     0+00:00:13        513.2
class_second     22      34     0+01:40:57     0+00:00:16        378.6
 class_first     16      17     0+02:45:05     0+00:00:16        619.1
       TOTAL     77      90     0+06:18:55     0+00:01:07        339.3

       Group   Jobs   Steps        Job Cpu    Starter Cpu     Leverage
group_serial     18      24     0+00:03:08     0+00:00:24          7.8
       staff     13      13     0+00:00:08     0+00:00:04          2.0
    No_Group     15      15     0+01:51:02     0+00:00:09        740.2
  group_para     31      38     0+04:24:35     0+00:00:29        547.4
       TOTAL     77      90     0+06:18:55     0+00:01:07        339.3

     Account   Jobs   Steps        Job Cpu    Starter Cpu     Leverage
        NONE     60      73     0+06:06:57     0+00:01:00        366.9
        bala     17      17     0+00:11:57     0+00:00:06        119.5
       TOTAL     77      90     0+06:18:55     0+00:01:07        339.3
```

The definitions of the various fields follow:

Name The user ID that submitted the job to the LoadLeveler

Jobs The total number of jobs submitted by this user

Steps The total number of job steps submitted by this user

Job CPU The total CPU time consumed by this user

Starter CPU The Total CPU time taken by the LoadLeveler starter process for this user

Leverage The ratio of the job CPU to starter CPU

Class The class defined by the user in his job command file or the default class

Group The Users login group as defined in the job command file or the default group

Account The users account name as defined in the job command file or the default account

We can also generate the report for a particular user, class, group, unixgroup, or node. This can be done using the -u, -c, -g, -G, and -a options in the `llsummary` command.

The throughput of the system can be found using the option -r throughput in the `llsummary` command. For more details on the syntax and options of the `llsummary` command, refer to the manual *LoadLeveler for AIX: Using and Administering*, SA22-7311.

As we discussed in the previous section you can customize the accounting report specific to your requirement if the report generated by the various options of the llsummary command doesn't meet your requirement. Using the APIs you can write your own code using the `GetHistory` subroutine of the LoadLeveler API. There is a sample code and makefile given in the /usr/lpp/LoadL/full/samples/llphist directory on how to use this subroutine to generate the report.

Chapter 4. Secureway Network Dispatcher

Secureway Network Dispatcher Version 2.1 is a component of Websphere performance pack Version 2.0. Secureway Network Dispatcher is a load balancing tool for managing TCP/IP connections in a network of servers. Network Dispatcher can be configured to manage the TCP/IP sessions for a group of SP nodes. In this chapter, we will discuss the configuration of Network Dispatcher in an SP environment.

4.1 Network Dispatcher and RS/6000 SP overview

Secureway Network Dispatcher is a product for managing the TCP/IP connections in a network of servers. The RS/6000 SP environment can be viewed as a network of servers, thus, Network Dispatcher is very well suited for an SP environment. It can also be seen as the software counterpart of the RS/6000 SP hardware. In the domain of server consolidation, one the most attractive features of the RS/6000 SP is its scaleability, which allows a customer to add (or remove) nodes to an SP cluster. Secureway Network Dispatcher brings the software scalability that makes all the SP nodes look like a unique server, with only one IP address, that can provide a constantly-available service to the client. Secureway Network Dispatcher makes the hardware implementation of the SP environment transparent to the end-user.

From a network administration point of view, Secureway Network Dispatcher is aimed at providing traffic load balancing between different networks. Therefore, it fits perfectly into an SP environment where several networks are generally available: The service ethernet network, the SP switch, and the networks connecting the SP to the external world. Secureway Network Dispatcher provides a means of spreading the traffic load between all available networks to prevent network bottlenecks.

4.2 Architecture

Secureway Network Dispatcher is a tool for managing TCP/IP connections to a group of servers in a networked environment. The basic architecture of Network Dispatcher consists of a Network Dispatcher server, optional standby server, and a group of nodes in a cluster for routing the TCP/IP connections. All TCP/IP connections to this group of nodes will be routed through the Network Dispatcher server, which provides a virtual IP address to these nodes. In short, Network Dispatcher provides the virtual view to these nodes. The Secureway Network Dispatcher components are:

- Dispatcher
- Interactive Session Support (ISS)
- Content Based Routing (CBR)

Network Dispatcher manages the TCP/IP connections from the clients based on the current workload on these nodes. Network Dispatcher calculates the load on these nodes by the number of connections, server load information from ISS, or advisors.

For detailed information about features of Network Dispatcher, we recommend that you read the redbook *New and Improved : IBM WebSphere Performance Pack :Loadbalancing with IBM Secureway Network Dispatcher,* SG24-5858.

We also recommend that you read *SecureWay Network Dispatcher - User's Guide,* GC31-8496, which contains installation and configuration instructions as well as command and configuration file descriptions.

4.3 Installation and configuration

In this section, we will discuss the configuration of Secureway Network Dispatcher in an SP environment. We will also discuss the steps for planning and installing the dispatcher in SP nodes.

4.3.1 Packaging and requirements

Secureway Network Dispatcher is made of three components: The Dispatcher, ISS, and CBR. These components are packaged in several filesets. You only need to install the filesets corresponding to the components you need.

In the list below, <lang> must be replaced by the identifier of the locale you will use, such as en_US for U.S. English.

To install:

- The Dispatcher
  - intnd.admin.rte
  - intnd.nd.driver
  - intnd.nd.rte
  - intnd.ndadmin.rte
  - intnd.msg.<lang>.admin.rte

    - intnd.msg.<lang>.nd.rte
    - intnd.msg.<lang>.ndadmin.rte
  - ISS
    - intnd.admin.rte
    - intnd.issr.rte
    - intnd.issadmin.rte
    - intnd.msg.<lang>.admin.rte
    - intnd.msg.<lang>.issr.rte
    - intnd.msg.<lang>.issadmin.rte
  - CBR
    - intnd.admin.rte
    - intnd.cbr.rte
    - intnd.cbradmin.rte
    - intnd.msg.<lang>.admin.rte
    - intnd.msg.<lang>.cbr.rte
    - intnd.msg.<lang>.cbradmin.rte
  - Network Dispatcher Documentation
    - intnd.doc.en_US

The prerequisites for installing any of the Network Dispatcher components in an SP environment are:

- IBM AIX Version 4.2.1 or higher
- Java runtime environment (JRE) Version 1.1.6 or higher (Java.rte.bin, Java.rte.class, Java.rte.lib).
- Web Traffic Express (WTE) Version 2.0 if you are using the CBR component.

4.3.2 Planning to configure Network Dispatcher in SP

You need to plan the configuration of a Network Dispatcher Cluster for your SP environment. You must identify the nodes for configuring the Network Dispatcher server and standby or backup Network Dispatcher server. You also need to identify the cluster of nodes to be managed by the Network Dispatcher server. Also, identify the networks to be configured for routing the connections. Define the necessary route definitions if they are needed.

The roles that we planned for the nodes in our test environment are listed in Table 7.

Table 7. Role of nodes for Network Dispatcher configuration

Role	Hostname of the node
Network Dispatcher server	sp4n10
Network Dispatcher standby server	sp4n09
Network Dispatcher cluster nodes	sp4n11, sp4n13

The network configurations we planned for routing the connections are listed in Table 8.

Table 8. Network configuration

Network type	Function assigned
SP Ethernet	Incoming connections from client
SP switch	Network Dispatcher communication and routing the connections to nodes
Additional Ethernet	Outgoing traffic from nodes

We configured control workstation (CWS) and node15 (sp4n15) as routers to the external clients.

The logical view of the above scenario is represented in Figure 17 on page 107.

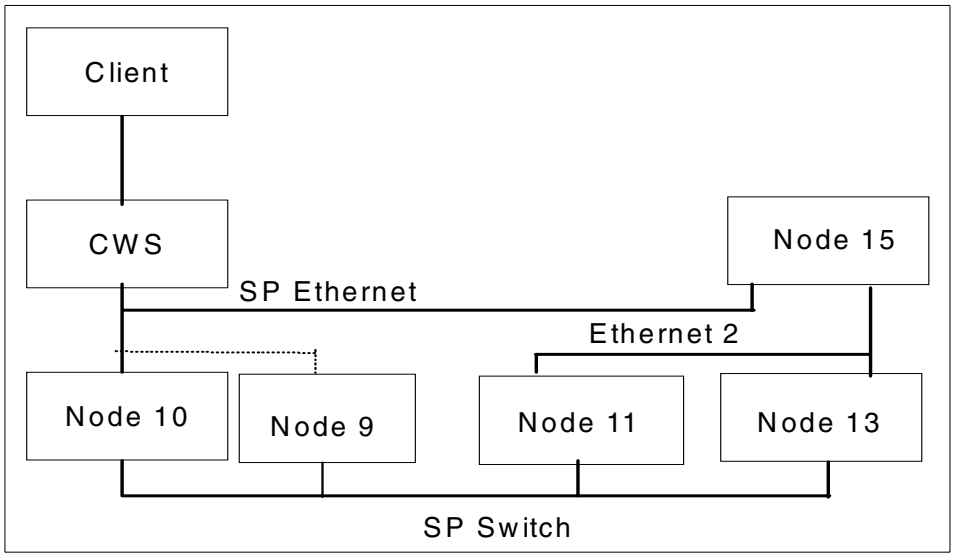


Figure 17. Hardware environment for Network Dispatcher installation

4.3.3 The installation and configuration process

This section describes the installation steps for configuring the Dispatcher in our SP environment.

1. Copy the filesets from installation media to CWS.

   Using the smitty `bffcreate` command, we copied the Network Dispatcher filesets to the /spdata/sys1/install/aix433/lppsource directory in CWS.

2. Install the Network Dispatcher filesets in node 10 (sp4n10).

```
# dsh -w sp4n10, mount sp4cws:/spdata/sys1/install/aix433/lppsource /mnt
# dsh -w sp4n10 installp -acXgd /mnt intnd.msg.en_US.admin.rte intnd.admin.rte
intnd.msg.en_US.issadmin.rte intnd.iss.rte intnd.issadmin.rte intnd.ndadmin.rte
intnd.msg.en_US.ndadmin.rte intnd.msg.en_US.iss.rte intnd.nd.driver \
intnd.nd.rte   intnd.msg.en_US.nd.rte
#
```

3. Start the Network Dispatcher server

   a. Log on to the sp4n10, and start the Network Dispatcher server using the `ndserver` command.

   b. Enter `#ndserver`

4. Verify, using the `ps` command, whether the java process has been started by the `ndserver` command.

```
# ps -ef|grep java
    root 21002     1  51 14:07:32  pts/0  0:03 java com/ibm/internet/nd/server/S
RV_ConfigServer 10099 10005 /usr/lpp/nd/dispatcher/logs/ /usr/lpp/nd/dispatcher/
configurations/ /usr/lpp/nd/dispatcher/bin/
```

5. Create the authentication keys, and start the executor.

```
#ndkeys create
#Key files have been created successfully
#ndcontrol executor start
Loaded kernel successfully.
```

6. Define the non-forwarding IP address.

   We set the non-forwarding address to the IP address node 10 on the SP ethernet adapter.

```
# ndcontrol executor set nfa 192.168.4.10
Executor field(s) successfully set.
```

7. Define the Network Dispatcher cluster.

   We now define a with the virtual IP address as 192.168.4.100 and assign to the new Network Dispatcher cluster called NDCLUSTER. The clients will connect to the NDCLUSTER using this IP address.

```
# ndcontrol cluster add NDCLUSTER
Cluster 192.168.4.100 has been added.
```

8. Add an alias on the Network Dispatcher server node.

   The NDCLUSTER address is aliased to the en0 interface of the Dispatcher machine as presented on Figure 18 on page 109.

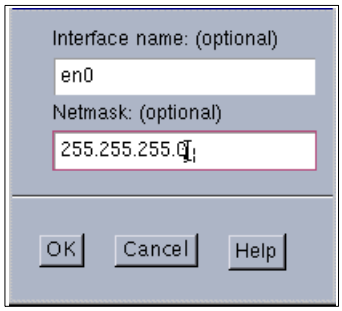


Figure 18. Aliasing NDCLUSTER on the SP ethernet adapter

9. Configure CWS as the Network Dispatcher management station.

   We configured CWS as the management station for Network Dispatcher administration. We installed the Network Dispatcher administration filesets (for Base Network Dispatcher) as well as the license key on the CWS. In addition, we installed the Network Dispatcher documentation:

```
# pwd
/spdata/sys1/install/aix433/lppsource
# installp -acXgd. intnd.admin.rte intnd.msg.en_US.admin.rte intnd.doc.en_US \
intnd.ndadmin.rte intnd.msg.en_US.ndadmin.rte
#
```

4.3.4 Configuring SP nodes to NDCLUSTER

By performing the steps described in the previous section, we installed and configured the Network Dispatcher server and created a new cluster called NDCLUSTER. Now, we can add nodes and services to this cluster. As we planned, we will add sp4n11 and sp4n13 to the NDCLUSTER. The nodes are configured in the Network Dispatcher cluster for a TCP/IP service. We define telnet service at port 23 and add nodes 11 and 13 for this service in the NDCLUSTER:

```
# ndcontrol port add 192.168.4.100:23
Port 23 successfully added to cluster 192.168.4.100.
ndcontrol port set 192.168.4.100:23 staletimeout 32000000
Port field(s) successfully set.
```

Nodes 11 and 13 are configured to serve telnet requests sent to port 23 of the Dispatcher. To ensure that traffic between the Dispatcher and the server flows

through the SP Switch, we use the switch address of these nodes as an argument of the `ndcontrol` command:

```
ndcontrol server add NDCLUSTER:23:sp4sw11+sp4sw13
Server 192.168.14.11 was added to port 23 of cluster 192.168.4.100.
Server 192.168.14.13 was added to port 23 of cluster 192.168.4.100.
```

The loopback device on server nodes 11 and 13 must be aliased to the cluster address NDCLUSTER so that the nodes accept incoming requests. We issued the following command in the control workstation:

```
# dsh -w sp4n11,sp4n13 ifconfig lo0 alias NDCLUSTER netmask 255.255.255.0
#
```

We defined the default routes in CWS and node 15 for managing the incoming and outgoing traffic.

The manager and the advisors are started with the default values. We present their use in Part 2, “Workload management sample scenarios” on page 133.

```
#ndcontrol manager start
The manager has been started.
#ndcontrol advisor start telnet 23
Advisor 'telnet' has been started on port 23.
# ndcontrol manager proportions 48 48 4 0
The proportions of the manager were set to: 48 48 4 0
```

In this configuration, the data flow is indicated by the numbered arrows as in Figure 19 on page 111. Requests from clients flow through the network (1 and 2) to the Dispatcher. They are forwarded on the SP switch (3) to the appropriate server (node 11 or 13), and the outgoing message is routed through the network (4, 5, and 6) to the client.

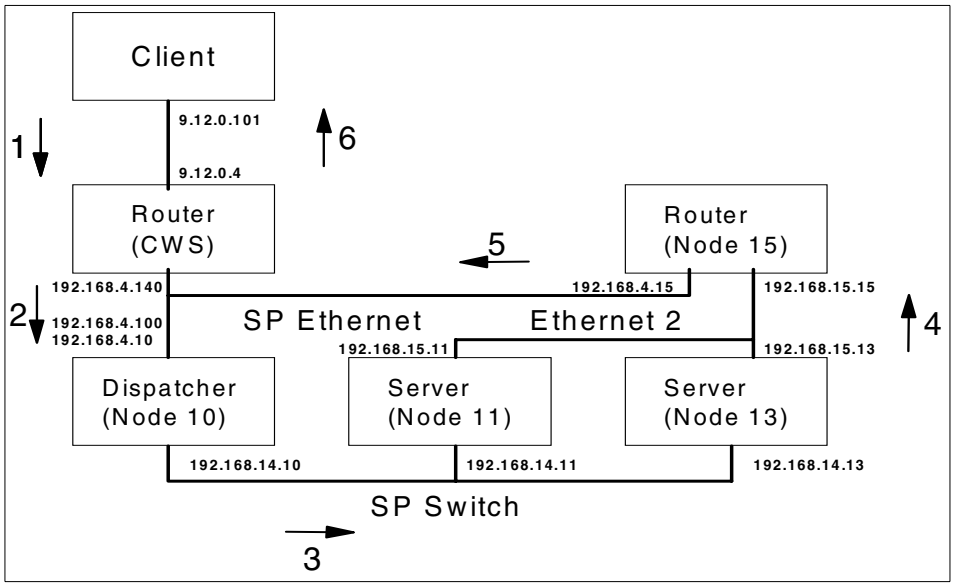


Figure 19. Data flow between client, dispatcher, and server

Figure 19 contains the IP addresses used for each network adapter in this configuration. The SP ethernet adapter address of node 10 is 192.168.4.10. It is also aliased to 192.168.4.100. This is the address used to represent the server cluster. The client will send their request to the 192.168.4.100 address also defined as NDCLUSTER.msc.itso.ibm.com.

To stop the server on the Dispatcher node, issue the following command on the dispatcher node.

```
#ndserver stop
```

4.3.5 Remarks about this configuration

In the configuration described above, the CWS is used as the software repository and also as a Network Dispatcher administrative station. We used three networks to manage the network traffic. We also made use of the SP switch taking advantage of its availability in our SP configuration, sharing the bandwidth with other applications that use the switch. If your configuration does not have an SP switch, you can, alternatively, use another network, such as ethernet or FDDI, instead of the switch.

4.4 Alternative configuration using the SP switch

Alternatively, if you want to use the SP switch bandwidth for outgoing traffic out of the SP nodes, you can configure the SP with Switch router, and the dispatcher can be configured to take advantage of the switch bandwidth for outgoing network communication. A sample configuration is described in Figure 20.

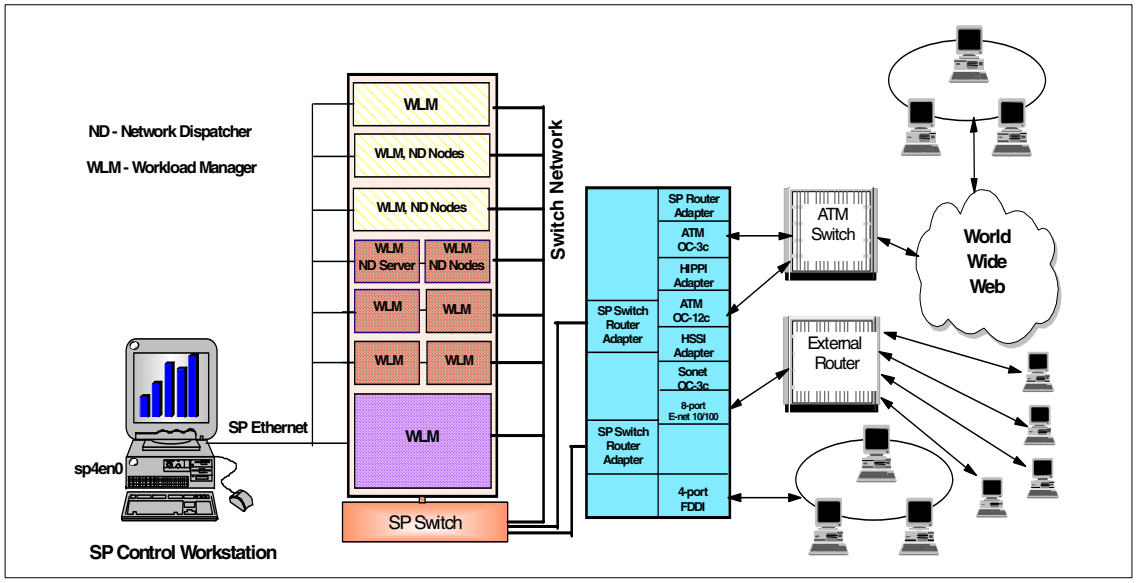


Figure 20. Network Dispatcher using SP switch router

4.5 Alternative configuration without SP switch

Another simple alternative configuration is to use one separate network other than the SP administrative ethernet for managing both incoming and outgoing traffic. This configuration does not require SP switch, and you can configure the Network Dispatcher to use this additional network to manage the network connections. A sample configuration is described in Figure 21 on page 113.

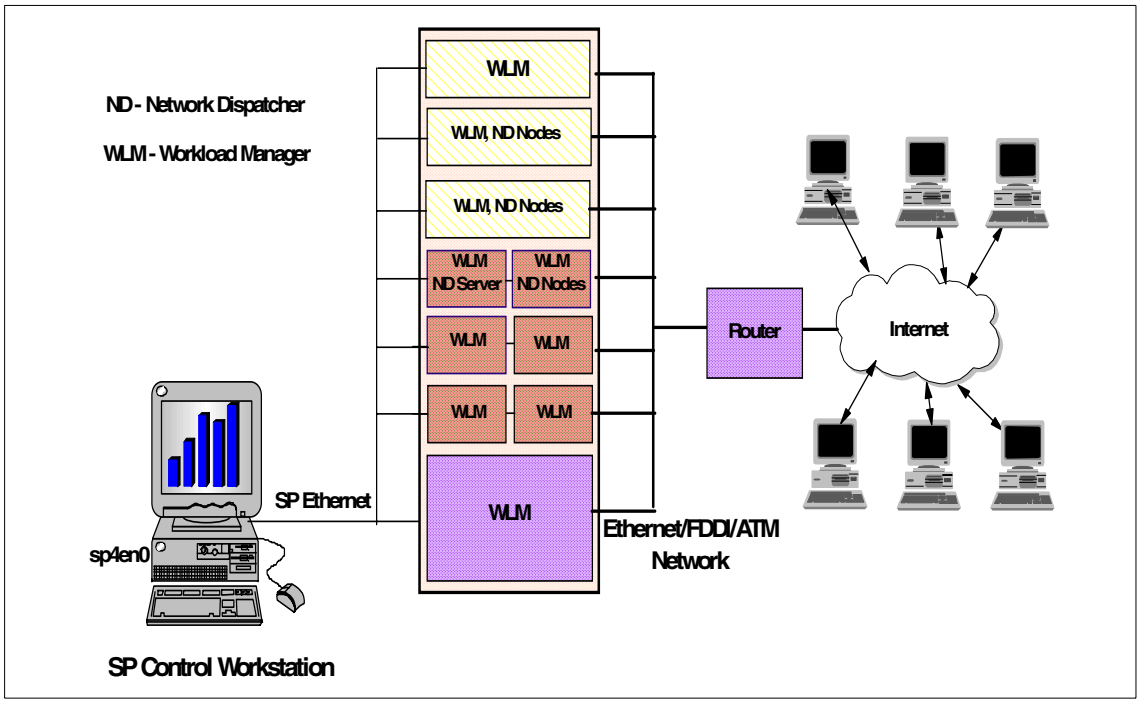


Figure 21. Network Dispatcher using Ethernet/FDDI/ATM networks

4.6 Related publications on Secureway Network Dispatcher

We have not discussed the features and functions of the Secureway Network Dispatcher in detail in this book. The detailed information about this product can be found in the following documents. We suggest that you read these documents before configuring the Dispatcher in SP.

The *SecureWay Network Dispatcher - User's Guide,* GC31-8496, contains installation and configuration instructions as well as command and configuration files description.

Part *New and Improved : IBM WebSphere Performance Pack :Loadbalancing with IBM Secureway Network Dispatcher,* SG24-5858, presents concrete examples of the use of Secureway Network Dispatcher in the general context of the IBM Websphere performance pack.

The Web site `http://www.ibm.com/software/network/dispatcher` contains more information on Secureway Network Dispatcher.

Chapter 5. AIX Workload Manager

AIX Workload Manager is a feature of AIX version 4.3.3. WLM is a tool to support the management of resources in an RS/6000 server. AIX workload manager can be configured to manage the workload within a node in an RS/6000 SP. In this chapter, we will focus on using WLM in an RS/6000 SP environment.

5.1 AIX Workload Manager and RS/6000 SP overview

The WLM component of AIX 4.3.3 is a very useful product for managing mixed workload situations in an RS/6000 server. Using WLM, you can control the allocation of resources, such as CPU and memory, to users and applications. This is achieved by grouping specific users, groups, or applications into WLM classes and regulating the resource allocations between these different classes using shares and limits. The resources are allocated to the classes as a percentage of CPU and memory. WLM allows you to define priorities between the classes. WLM allocates the resources based on the class configurations when jobs are executed.

RS/6000 SP is being used to consolidate multiple servers and applications; therefore, SP is increasingly being used to run more than one type of workload. In such situations, you need to configure the SP nodes to manage the mixed workloads within a node. As with any other RS/6000 server, WLM provides the support for managing the workloads within an SP node. The products, LoadLeveler and Secureway Network Dispatcher, can be used to schedule the jobs in multiple nodes in SP.

In this chapter, we will focus on configuring WLM on SP using PSSP tools, such as file collection. In Part 2, “Workload management sample scenarios” on page 133, we will describe a few scenarios on how to manage the workload in SP using WLM with LoadLeveler and Secureway Network Dispatcher.

5.2 AIX WorkLoad Manager architecture

WLM is a tool that helps administrators manage the resources in an RS/6000 server or a node in SP. The basic concepts of WLM are *classes* and *tiers*. The classes describe the rules and resource limits for allocating the CPU and Memory. The users, groups, or path names of the applications they execute, are defined using the class assignment rules. The tiers define the importance

of the class. WLM monitors the load on the system and allocates the resources to the jobs based on the class and tier defined by the administrator.

For a detailed description of WLM features and how to create a WLM configuration, we recommend that you read section 7.2 of the redbook *Server Consolidation on RS/6000*, SG24-5507.

We also recommend that you read about the WLM commands described in *AIX Version 4.3 Commands Reference*, SBOF-1877, for using the command line interface, and *Workload Manager Technical Reference* on the Web at `http://www.ibm.com/servers/aix/library` under Technical Publication section.

5.3 Installation and configuration

In this section, we will discuss on the installation and configuration of WLM in SP nodes. We will also discuss how to make use of PSSP tools, such as file collection, to create a consistent WLM configuration for a node set in SP. We will also discuss how to use the distributed shell feature to manage WLM configuration in SP.

5.3.1 Packaging and requirements

AIX Workload Manager is a part of the AIX 4.3.3 runtime. It is packaged with AIX 4.3.3 as a mandatory fileset bos.rte.control. This fileset is installed on the SP nodes with the base AIX 4.3.3. installation; therefore, there are no prerequisites for the installation of WLM in SP.

5.3.2 Planning to configure WLM in SP nodes

The SP supports multiple AIX versions to run on its nodes. The WLM is only installed on the nodes that run AIX 4.3.3. It is not necessary to configure WLM in all the nodes of SP that run AIX Version 4.3.3. The system administrator can select the nodes in which WLM configuration may be necessary. WLM configuration is required when you have to manage the mixed type of workload. You can select a subset or all the nodes in SP for configuring WLM if you need to manage different types of workloads in these nodes.

5.3.3 Installation and the configuration process

As we have just discussed, there are no installation steps for WLM installation; therefore, we will now discuss the configuration of WLM in SP nodes using the file collection methods of PSSP. Before we begin configuring WLM on SP nodes, we will verify the installation of the fileset, bos.rte.control,

in the CWS and SP nodes. This can be performed by issuing the following command in the control workstation:

```
lslpp -l bos.rte.control for CWS
dsh -W lslpp -l bos.rte.control for nodes
```

5.3.4 Basic Configuration

The configuration files of WLM are located in the /etc/wlm directory in each node where AIX 4.3.3 is installed and in the control workstation. The default configuration files are located in the /etc/wlm/standard directory. This configuration contains a class definition file for two classes: System and default. The other files in this directory are limits, shares, and rules. These files define the characteristics for the two classes defined in the class file. The current file in /etc/wlm is a symbolic link to the /etc/wlm/standard directory. The current file defines the active WLM configuration for that node. Initially, the current file is a link to the /etc/wlm/standard directory.

```
# cd /etc/wlm
# ls -al
total 32
drwxr-xr-x   3 root     system       512 Oct 20 11:54 .
drwxr-xr-x  27 root     system      7680 Oct 20 11:48 ..
lrwxrwxrwx   1 root     system        17 Oct 07 14:33 current -> /etc/wlm/standard
drwxr-xr-x   2 root     system       512 Oct 07 14:33 standard
# ls -al standard
total 48
drwxr-xr-x   2 root     system       512 Oct 20 11:55 .
drwxr-xr-x   6 root     system       512 Oct 20 11:55 ..
-rw-r--r--   1 root     system       423 Apr 27 14:51 classes
-rw-r--r--   1 root     system       430 Apr 27 14:51 limits
-rw-r--r--   1 root     system       496 Apr 27 14:51 rules
-rw-r--r--   1 root     system       404 Apr 27 14:51 shares
```

WLM configuration can be customized for your requirements in two ways: One method is to modify the files in the /etc/wlm/standard directory. The other method is to create a separate directory under /etc/wlm and copy the files from the /etc/wlm/standard directory and customize the files to your requirements.

To configure the WLM in SP, we define WLM configuration for the nodes in the control workstation, and then we can propagate them to the nodes using file collection. In this process, we can create a consistent WLM configuration in

SP nodes. In this case, the /etc/wlm directory in CWS consists of all WLM configuration files for SP nodes. To use file collection to manage WLM configuration, we define a separate file collection in CWS called wlmCollection.

5.3.5 Creating a WLM configuration file collection

Creating a new file collection for WLM configuration will help you independently manage the WLM configuration in SP. You can use file collection methods to distribute the changes to the WLM configuration.

5.3.5.1 Creating a file collection directory

First, we create a directory under /var/sysman/sup, called wlmCollection, to hold the file collection control files and copy the configuration files from the existing file collection sup.admin to the wlmCollection directory.

```
# cd /var/sysman/sup
# mkdir wlmCollection
# chown bin.bin wlmCollection
# cd /var/sysman/sup/sup.admin/
# ls -al
total 9
drwxr-xr-x   2 bin      bin           512 Oct 21 07:49 .
drwxr-xr-x   8 bin      bin           512 Oct 21 10:04 ..
-rw-r--r--   1 root     system       1877 Oct 20 10:48 host
-rwxr-xr--   1 bin      bin           219 Oct 19 11:09 list
-rwxr-xr--   1 bin      bin             0 Oct 19 11:09 lock
-rwxr-xr--   1 bin      bin             2 Oct 19 11:09 prefix
-rwxr-xr--   1 bin      bin             1 Oct 19 11:09 refuse
-rwxr-xr--   1 bin      bin             0 Oct 19 11:09 supperlock
# cp -p host lock prefix refuse supperlock ../wlmCollection
```

If sup.admin has been modified at your site, ensure that the copied prefix file contains only a “/” (slash) and that the copied refuse file is empty.

5.3.5.2 Creating a list file

We now edit the list file in this directory and define what files will be managed by wlmCollection. We defined the following rules to add entries to this file.

- The /etc/wlm/current link is not copied from the CWS to the node so that each node pointer to the current configuration is not changed by file collection updates.
- The backups (*.old) created by WLM are not propagated to the nodes.

- All other files in the /etc/wlm directory of the CWS are copied onto each node.

We add the following entries to the list file. We then create a link to the newly-created list in the /var/sysman/sup/lists.

```
# pwd
/var/sysman/sup/wlmCollection
# cat list
symlinkall
omit ./etc/wlm/current
omitany ./etc/wlm/*/*.old
upgrade ./etc/wlm
# ln -s /var/sysman/sup/wlmCollection/list /var/sysman/sup/lists/wlmCollection
```

5.3.5.3 Updating the file.collection file

We now define wlmCollection as a primary file collection by editing /var/sysman/file.collections and adding the last three lines in the following example. Refer to the *Parallel System Support Program for AIX - Administration Guide*, SA22-7348, for details about the syntax of these lines.

```
# pwd
/var/sysman
# cat file.collections
#
# File collection definitions
# sup.admin - file collection to manage file collections(sup)
primary sup.admin - /var/sysman - / EO power no
# user.admin - file collection to manage files needed to manage user
primary user.admin - / - / EO power no
# power_system - file collection to manage common files across systems
primary power_system  - /share/power/system/3.2 - / EO power no
# commons - file collection to manage common files across systems
secondary node.root power_system / - /share/power/system/3.2 EO power no
# wlmCollection - user defined file collection for managing wlm configurations
# of all SP nodes from the CWS.
primary wlmCollection - / - / EDO power no
#
```

5.3.5.4 Updating the .resident file

We now have to update the .resident file for each node on which the file collection will be propagated to add wlmCollection. After this update, /var/sysman/sup/.resident contains:

```
# pwd
/var/sysman/sup
# cat .resident
node.root 0
sup.admin 0
user.admin 0
wlmCollection 0
#
```

5.3.5.5 Update the sup.admin collection.

For each node on which the wlmCollection is to be propagated, we have to update the sup.admin collection. We perform this directly from the CWS using distributed shell:

```
# dsh -w sp4n09,sp4n10,sp4n11,sp4n13 /var/sysman/supper \
update sup.admin
#
```

5.3.5.6 Building the scan file

We now build a scan file to speed up later file collection processing using the `dsh` command to execute the `scan` command on each node from the CWS.

```
# dsh -w sp4n09,sp4n10,sp4n11,sp4n13 /var/sysman/supper \
scan wlmCollection
#
```

5.3.5.7 Installing the wlmCollection

We now install the wlmCollection file collection in the nodes where we will use WLM to manage the node resources.

```
# dsh -w sp4n09,sp4n10,sp4n11,sp4n13 /var/sysman/supper \
install wlmCollection
#
```

Using the distributed shell, we can now propagate the WLM configuration from control workstation to SP nodes. In this process, the files in the /etc/wlm directory in CWS are copied to the SP nodes. So far, we have discussed the basic WLM configuration and the file collection configuration required to distribute the WLM configuration files to SP nodes. We have not defined any

WLM classes for managing node resources. You can define such class configurations in CWS and use the file collection methods for distributing them to SP nodes. We will discuss these issues in the next section.

5.4 Managing the WLM configuration in SP

For managing the nodes in SP, we define the WLM configurations in the CWS. For your configurations, it may be necessary to define more than one WLM configuration. You can then assign a WLM configuration to a node or a group of nodes. In the following example, we want to manage two types of workloads in the SP nodes: Interactive and batch. We want to assign two nodes with higher resources for interactive jobs compared to batch jobs. Similarly, we want to assign two nodes for batch jobs with a greater share of resources.

5.4.1 Definition of user ID and groups

First, using the SP User Management on the CWS, we define the user IDs and groups that are to be managed through WLM.

We decided to create three groups:

- InterA for interactive users mostly using application A.
- InterB for interactive users mostly using application B.
- Batch for users submitting batch jobs.

And we created users with the SP User Management tools:

- InterA1 and InterA2 belonging to group InterA
- InterB1 and InterB2 belonging to group InterB
- Batch1 and Batch2 belonging to group Batch

```
#  mkgroup -'A' id='1000' InterA
#  mkgroup -'A' id='2000' InterB
#  mkgroup -'A' id='3000' Batch
#
```

```
# smitty spmkuser


Add a User

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* User NAME                                          [InterA1]
  User ID                                           [1001]
  LOGIN user?                                        true
  PRIMARY group [InterA]
  Secondary GROUPS                                  []
  HOME directory                                     [/home/InterA1]
  Initial PROGRAM                                   [/bin/ksh]
  User INFORMATION                                    [Interactive use of Application A]
```

5.4.2 Defining classes

In the scenario, we have to create two workload profiles, and we do not modify the default set of configuration files. Rather, we define one for each set of nodes, and we use the nodes profile as a mnemonic name for each configuration set.

The easiest way to create these configuration directories is to copy the standard configuration provided on the AIX distribution, and then modify it. We create two directories in /etc/wlm for InteractiveNodes and Batchnodes and copy the standard WLM property files from the /etc/wlm/standard directory:

```
# cd /etc/wlm
# ls -al
total 32
drwxr-xr-x   3 root     system       512 Oct 20 11:54 .
drwxr-xr-x  27 root     system      7680 Oct 20 11:48 ..
lrwxrwxrwx   1 root     system        17 Oct 07 14:33 current -> /etc/wlm/standard
drwxr-xr-x   2 root     system       512 Oct 07 14:33 standard
# mkdir InteractiveNodes
# mkdir BatchNodes
# cp -pr standard/* InteractiveNodes
# cp -pr standard/* BatchNodes
```

For customizing WLM, it is possible to use the command line interface, SMIT, or the Web-Based System Manager. WLM has a concept of *current* configuration. The current configuration is a pointer (a symbolic link) in /etc/

wlm to the directory of the configuration currently in use on the system. The SMIT interface allows you to work with the *current configuration* and not other configurations. We use the command line interface in the following sections. This is also useful for managing the WLM configuration in RS/6000 SP. We recommend that the reader refer to the redbook *Server Consolidation on RS/6000*, SG24-5507, for examples of using SMIT or the Web-Based System Manager.

5.4.2.1 Interactive node configuration

First, we create a description file for this configuration. This description file is optional, but we recommend that one be created in an SP environment because there may be many different workload profiles. This description will be useful when you later need to refer to this profile to modify it.

The description file is a free text ASCII file containing only one line of text. It is stored in /etc/wlm/InteractiveNodes.

```
# pwd
/etc/wlm/InteractiveNodes
# cat description
Profile to be used for nodes 9 and 10, running interactive applications.
#
```

We then create a new class, *interjobs*, for interactive jobs and a class, *batchjobs*, for batch. Since we define the configuration for the nodes where priority is given to interactive activity over batch jobs, we give a share level of 10 to interjobs, and we set batch jobs in tier 1 to ensure that processes in this class will not be scheduled as long as there are other jobs ready to run in the system. We also change the System class so that it has priority over all processes except those in the Interactive class.

```
# mkclass -a tier=0 -c shares=10 -m shares=10 -d InteractiveNodes interjobs
# mkclass -a tier=1 -c shares=1 -m shares=1 -d InteractiveNodes batchjobs
# chclass -c shares=2 -m shares=2 -d InteractiveNodes  System
```

As a result, we can see that the property files have been changed:

```
# more classes
System:
        description = ""
        tier   = 0

Default:

interjobs:
        tier   = 0

batchjobs:
        tier   = 1

# more limits
System:
        CPU    = 0%-100%
        memory = 1%-100%

# more shares
System:
        CPU    = 2
        memory = 2

interjobs:
        CPU    = 10
        memory = 10

batchjobs:
        CPU    = 1
        memory = 1
```

We now have to define the rules for assigning processes to classes. We choose a simple criteria: All processes belonging to groups InterA or InterB will be in class interjobs; all processes belonging to the group Batch will be in class batchjobs, and all others will be either in the System or Default class.

There is no command to modify the assignment of rules. This can be done using a text editor or wsm. We edited the /etc/wlm/InteractiveNodes/rules file and added three rules (lines starting with interjobs and batchjobs):

```
# pwd
/etc/wlm/InteractiveNodes
# cat rules
* IBM_PROLOG_BEGIN_TAG
* This is an automatically generated prolog.
*
* bos43N src/bos/etc/wlm/rules 1.1
*
* Licensed Materials - Property of IBM
*
* (C) COPYRIGHT International Business Machines Corp. 1999
* All Rights Reserved
*
* US Government Users Restricted Rights - Use, duplication or
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
*
* IBM_PROLOG_END_TAG
* class resvd user group application
System    -   root   -        -
interjobs -   -     InterA    -
interjobs -   -     InterB    -
batchjob  -   -     Batch     -
Default   -   -      -        -
#
```

5.4.2.2 Batch Node class

For the batch nodes, we create same *interjobs* and *batchjobs* classes, but, in this configuration, priorities are inverted: Batchjobs get the highest priority. However, to ensure that the response time of interactive users would not be too long, we keep all classes in the same tier. We assign a minimum of 50 percent of CPU and memory resources to batch jobs. We keep the same assignment rules that exist in the InteractiveNodes configuration.

```
# mkclass -d BatchNodes interjobs
# mkclass -c shares=10 -c min=50 -m shares=10 -m min=50 -d BatchNodes batchjobs
# chclass -c shares=2 -m shares=2 -d BatchNodes  System
# cp -p /etc/wlm/InteractiveNodes/rules /etc/wlm/BatchNodes/rules
#
```

5.4.3 Updating WLM configuration to nodes

The WLM configurations defined in the previous section are created in the control workstation. These configurations should be propagated to the SP nodes using the wlmCollection file collection.

5.4.3.1 Using the distributed shell command

We will use the PSSP `dsh` command to copy the files to the SP nodes. We perform this command on the four nodes of our SP configuration:

```
# dsh -w sp4n09,sp4n10,sp4n11,sp4n13 /var/sysman/supper \
update wlmCollection
```

5.4.3.2 Using crontab

We can also define the `supper update` command in the crontab in these nodes to perform the update at regular intervals. We modify the crontab entry on each node with the following entry:

```
10 * * * *  /var/sysman/supper update sup.admin user.admin node.root \
wlmCollection 1>/dev/null 2>/dev/null
```

5.4.4 Starting WLM

WLM is activated with the `wlmcntrl` command. Usually, on production systems, WLM will be started during system initialization by a `wlmcntrl` command added to the file */etc/inittab*. The system administration tools, SMIT and WebSM, offer menu selections to either start WLM immediately or at boot time by adding a WLM entry to the *inittab*, or both.

The WLM entry will be added at the end of the /etc/inittab file; so, starting WLM will be the last step in the system initialization. System administrators might want to start WLM before starting their main applications so that the resource utilization of these applications is controlled by WLM from the start. They would then have to move the WLM entry *up* in the *inittab* file.

One thing to remember when doing this is that the application files referenced in the assignment rules must be accessible when WLM is started. Otherwise, those applications that cannot be accessed will be ignored, and the assignment rules will be incomplete. To avoid this problem, WLM should be started after all the file systems containing application files (both local and remote if applicable) are mounted.

5.4.4.1 Activating WLM at boot time

We use the `dsh` and `mkitab` commands to modify the /etc/inittab file of each node. We install the InteractivesNodes configuration on nodes 9 and 10 and the BatchNodes configuration on nodes 11 and 15.

```
# dsh -w sp4n09,sp4n10 "mkitab -i rc 'wlm:2:once:wlmcntrl  \
-d /etc/wlm/InteractivesNodes > /dev/console 2>&1'"
## dsh -w sp4n11 , sp4n15 "mkitab -i rc 'wlm:2:once:wlmcntrl  \
-d /etc/wlm/BatchNodes > /dev/console 2>&1'"
```

Now, WLM will start at the next reboot of each node.

5.4.4.2 Using the command line

It is not necessary to reboot the node to activate the WLM configuration. We can use the command line interface to start the WLM after the configurations are copied using the file collection methods.

```
# dsh -w sp4n09,sp4n10 "wlmcntrl -d /etc/wlm/InteractivesNodes"
## dsh -w sp4n11,sp4n15 "wlmcntrl -d /etc/wlm/BatchNodes"
#
```

WLM is now active on our four nodes, with a different configuration on each of them.

5.4.5 Verifying the WLM

It is possible to verify WLM activity with the `wlmcntrl -q` command, but we recommend that you use the `wlmstat` command that provides more information and returns an error message if WLM is not started.

```
# dsh -w sp4n09,sp4n10,sp4n11 wlmstat
sp4n09:              Name CPU MEM
sp4n09:     Unclassified   1   6
sp4n09:           System  11  14
sp4n09:          Default   0   0
sp4n09:        interjobs   0   0
sp4n09:        batchjobs   0   0
sp4n10:              Name CPU MEM
sp4n10:     Unclassified   0  31
sp4n10:           System   0   0
sp4n10:          Default   0   0
sp4n10:        interjobs   0   0
sp4n10:        batchjobs   0   0
sp4n11:              Name CPU MEM
sp4n11:     Unclassified   2  58
sp4n11:           System   0   1
sp4n11:          Default   0   0
sp4n11:        interjobs   0   0
sp4n11:        batchjobs   0   0
#
```

Now, we have created a simple CPU-intensive ksh script (merely a loop performing 100000 additions and one file copy). We start this script four times with user ID InterA1 and five times with user ID Batch1 on nodes 9 and 11. We can then use the `wlmstat` command to visualize the effect of the different configuration scripts:

```
# dsh -w sp4n09,sp4n11
dsh> wlmstat
sp4n09:              Name CPU MEM
sp4n09:     Unclassified   0   6
sp4n09:           System   1  21
sp4n09:          Default   0   0
sp4n09:        interjobs  56   0
sp4n09:        batchjobs  43   0
sp4n11:              Name CPU MEM
sp4n11:     Unclassified   3  44
sp4n11:           System   2  12
sp4n11:          Default   0   0
sp4n11:        interjobs  27   0
sp4n11:        batchjobs  68   1
dsh>
```

Node 9 is using the InteractivesNodes configuration that favors user InterA1 while node 11 runs the BatchNodes configuration in favor of user Batch1. Even if there are more instances of the program run by Batch1, we see that,

on Node 9, user InterA1 gets more resources than user Batch1. And, of course, the situation is reversed on node 11.

It may be surprising to see that, even with the “extreme” options used in configuring InteractivesNodes, Batch1 gets 43 percent of node 9’s CPU. The difference between the resources allocated to InterA1 and Batch1 may seem minor when compared to the differences between the parameters used in the configuration. This may be due to the fact that the program is not just performing a CPU activity. It also performs file copy (disk I/O), and, when all InterA1 processes are waiting for the I/O to complete, they cannot use the CPU; so, WLM allocates it to the Batch1 processes. This explains how even if Batch1 processes are in Tier 1, and not Tier 0, they get some resources.

Actions on the wlm parameters do not have immediate consequences. It takes some time for the system to stabilize to the load equilibrium defined by the new parameters. We, therefore, recommend that you monitor your system for some time after each change of parameters before drawing any conclusions about the consequences of the change.

5.4.6 Changing classes properties in a configuration

We can change the initial classes configuration by modifying the class parameters. Here we only modify the shares and limits of existing classes in a configuration currently in use.

We first log on to the CWS and modify the InteractivesNodes configuration. We set the shares limit of the interjobs class to 97.

```
# chclass -m shares=97 -c shares=97 -d InteractivesNodes interjobs
#
```

We then propagate the modification to all nodes using file collections.

```
# dsh -w sp4n09,sp4n10,sp4n11 /var/sysman/supper scan wlmCollection
# dsh -w sp4n09,sp4n10,sp4n11 /var/sysman/supper update wlmCollection
sp4n09: Updating collection wlmCollection from server sp4en0.msc.itso.ibm.com.
sp4n09: File Changes:  4 updated, 0 removed, 0 errors.
sp4n10: Updating collection wlmCollection from server sp4en0.msc.itso.ibm.com.
sp4n10: File Changes:  4 updated, 0 removed, 0 errors.
sp4n11: Updating collection wlmCollection from server sp4en0.msc.itso.ibm.com.
sp4n11: File Changes:  4 updated, 0 removed, 0 errors.
#
```

We update the wlm current configuration on node 9:

```
# dsh -w sp4n09 wlmcntrl -d InteractivesNodes -u
#
```

5.4.7 Changing priorities of the currently used classes

It is possible to load another configuration if it contains the same classes as the current configuration. This is useful if, for example, you want to have a set of priorities for daytime to favor interactive users and a set of priorities for nighttime to favor other users. In our scenario, node 9 is initially managed with InteractiveNodes configuration and node 11 with BatchNodes configuration. We load the BatchNodes configuration on node 9 using the update (-u) option of wlmcntrl, and then we measure the result:

```
# dsh -w sp4n09 wlmcntrl -d BatchNodes -u
# dsh -w sp4n09,sp4n11 wlmstat
sp4n09:             Name CPU MEM
sp4n09:     Unclassified   0   6
sp4n09:           System   0  21
sp4n09:          Default   0   0
sp4n09:        interjobs  20   0
sp4n09:        batchjobs  79   0
sp4n11:             Name CPU MEM
sp4n11:     Unclassified   2  44
sp4n11:           System   0  13
sp4n11:          Default   0   0
sp4n11:        interjobs  22   0
sp4n11:        batchjobs  75   1
#
```

Now that node 9 and node 11 are using the same wlm configuration, the resource allocation becomes similar on both nodes.

5.4.8 Stopping WLM

To stop WLM on the nodes, issue the following command:

```
#dsh -w sp4n09 ,sp4n10, sp4n11,sp4n15 wlmcntrl -o
```

5.5 Other publications related to AIX Workload Manager

We recommend that you refer to the following documents for additional information about AIX Workload Manager.

Details about the WLM commands can be found in *AIX Version 4.3 Commands Reference*, SBOF-1877.

The Workload Manager Technical Reference can be found on the Web at `http://www.ibm.com/servers/aix/library` under the Technical Publication section.

The Web-based System Manager online help offers detailed descriptions of the WLM activities that can be performed from the wsm Graphical User Interface.

Section 7.2 of *Server Consolidation on RS/6000*, SG24-5507 contains an introduction to AIX Workload Manager.

Chapter 4 of *Parallel System Support Program for AIX - Administration Guide*, SA22-7348, describes the use of File Collections

Part 2. Workload management sample scenarios

Chapter 6. Managing serial batch jobs

This chapter discusses how to distribute workload to nodes in the RS/6000 SP when you want to execute multiple serial jobs.

6.1 Scenario description

Suppose a user has multiple serial jobs to perform. We will cover the following two test cases:

1. Executing batch jobs that have no dependency on each other:

   If the multiple batch jobs have no dependency on each other, you can execute these jobs simultaneously using multiple nodes. For executing jobs at the same time, you need to know how to configure the environment for dispatching these jobs to nodes in your RS/6000 SP.

2. Executing batch jobs that have dependency on each other:

   You may have batch jobs that have dependency on each other. For example, you may need to execute batch job B after job A is completed successfully. In this case, you need to know how to manage this dependency in your environment.

6.2 Tool choice

You can use LoadLeveler to perform this scenario. By using LoadLeveler, you can use multiple nodes as a single system image and the LoadLeveler distributes workload to nodes for executing multiple batch jobs. We use the following environment:

- LoadLeveler V2.1 and AIX V4.3.3 on 10 nodes in the RS/6000 SP
- We define all 10 nodes as the executing nodes in LoadLeveler cluster
- We define two nodes as the scheduling nodes in LoadLeveler cluster

6.3 Considerations for the executing environment

To use multiple nodes in a LoadLeveler cluster, you need to create a common user space for your users, that is, a user has to have the same user ID and group ID on nodes in the LoadLeveler cluster to perform jobs. In our environment, we created the user ID using the SP user management function and propagate the ID with the file collection to create this common user space.

You need to consider the input and the output for jobs. The job should be able to read the input file from every executing node in the LoadLeveler cluster. You can configure a global file system, such as NFS, to share the directory that is needed by the jobs for I/O. We used NFS for the user's home directory. The user submits jobs from the home directory and gets the output from this directory.

6.4 Executing batch jobs that have no dependency on each other

If you have multiple jobs with no dependency, you can execute jobs simultaneously using multiple nodes. You can manage the workload for performing the serial jobs by using the class statement in the config file. You can control the number of jobs running on each node; this results in the distribution of jobs to multiple nodes while workload for the serial jobs is limiting on each node.

6.4.1 The administration file

We define four classes in our LoadLeveler cluster. We use one of those four classes, named `class_second`, for executing 20 serial jobs by user tani1. The following is the class stanza in the administration file:

```
small:          type = class
                priority = 100
                include_users=bala
#
class_first:    type = class
                class_comment = "First Class Users"
                include_users = kannan1 bala1
                admin = kannan1
                job_cpu_limit = 24:00:00,12:00:00
                wall_clock_limit = 24:00:00,12:00:00
                priority = 100
                nice = -5
#
class_second:   type = class
                class_comment = "Second Class Users"
                exclude_users = kannan1
                cpu_limit = 00:30:00
                data_limit = 80mb,60mb
                file_limit = 512mb
                stack_limit = 80mb,60mb
                job_cpu_limit = 02:00:00,01:00:00
                wall_clock_limit = 03:00:00,02:00:00
                priority = 60
#
class_parallel: type = class
                class_comment = "For job_type parallel"
                max_node = 5
                total_tasks = 15
                wall_clock_limit = 24:00:00,12:00:00
```

The `class_second` has some keywords for the resource limit and the priority. Tani1's jobs should be expected within these limits.

The following screen shows the definition of user `tani1` in the user stanza:

```
tani1:          type = user
                default_class = class_second
                default_group = group_serial
                maxjobs = 40
                maxqueued = 80
```

This user's default class is `class_second`. So, if this user does not specify a class name in the command file, jobs submitted by this user belong to the `class_second` class. This user can run 40 jobs simultaneously and have 80 jobs in the job queue.

6.4.2 The configuration file

We want to control workload by limiting the maximum number of jobs that can run on each node.

We define the following `Class` keyword in the local configuration file on each node:

```
Class = { "small" "small" "small" "small" "class_first" "class_first"
"class_second"
"short_limit" "long_limit" "class_parallel" }
```

This setting means each node can execute one `class_second` job; so, the user tani1 can execute 10 jobs at the same time using 10 nodes in our RS/6000 SP environment, assuming other jobs are not using `class_second`. By this definition, you can distribute the workload to multiple nodes when multiple jobs are submitted simultaneously. You can also control the workload of each class job on each node and prevent the class from dominating the workload on each node.

6.4.3 Job and the job command file

We use a simple korn shell script for each job. The shell script needs a specific process time before exiting the script. We can specify the process time as the first parameter of the shell script. The following is the shell script we use for this scenario:

```
$ cat sec_spend.ksh
#!/bin/ksh
echo $2 $3 $1 `date "+%m/%d/%Y %H:%M:%S"` `hostname` >> $2.pre.out

sleep $1

echo $2 $3 $1 `date "+%m/%d/%Y %H:%M:%S"` `hostname` >> $2.post.out

exit 0
$
```

For performing 20 jobs, we create two job command files: sen1_all_1.cmd and sen1_all_2.cmd. Each job command file has 10 job steps. In each job

step, we perform the shell script sec_spend.ksh with a different process time parameter. Table 9 shows the process time for each job step.

Table 9. The process time of job steps

Job step	Process time in seconds
step0	120
step1	110
step2	100
step3	90
step4	80
step5	70
step6	60
step7	50
step8	40
step9	30

The same job step in the two job command files will have the same process time. The difference between two job command files is the name of the output file that is specified as the second parameter of the shell script. The following is one of two job command files:

```
#  Submit 10 Serial Jobs
#
# @ job_name = sen1_all_1
# @ step_name = step0
# @ executable = sec_spend.ksh
# @ arguments = 120 sen1_all_1 step0
# @ wall_clock_limit = 125
# @ queue
# @ step_name = step1
# @ executable = sec_spend.ksh
# @ arguments = 110 sen1_all_1 step1
# @ wall_clock_limit = 115
# @ queue
# @ step_name = step2
# @ executable = sec_spend.ksh
# @ arguments = 100 sen1_all_1 step2
# @ wall_clock_limit = 105
# @ queue
# @ step_name = step3
# @ executable = sec_spend.ksh
# @ arguments = 90 sen1_all_1 step3
# @ wall_clock_limit = 95
# @ queue
# @ step_name = step4
# @ executable = sec_spend.ksh
# @ arguments = 80 sen1_all_1 step4
# @ wall_clock_limit = 85
# @ queue
# @ step_name = step5
# @ executable = sec_spend.ksh
# @ arguments = 70 sen1_all_1 step5
# @ wall_clock_limit = 75
# @ queue
# @ step_name = step6
# @ executable = sec_spend.ksh
# @ arguments = 60 sen1_all_1 step6
# @ wall_clock_limit = 65
# @ queue
# @ step_name = step7
# @ executable = sec_spend.ksh
# @ arguments = 50 sen1_all_1 step7
# @ wall_clock_limit = 55
# @ queue
# @ step_name = step8
# @ executable = sec_spend.ksh
# @ arguments = 40 sen1_all_1 step8
# @ wall_clock_limit = 45
# @ queue
# @ step_name = step9
# @ executable = sec_spend.ksh
# @ arguments = 30 sen1_all_1 step9
# @ wall_clock_limit = 35
# @ queue
```

In each step, we specify step_name, executable, arguments, wall_clock_time, and queue keyword. For example, the following command is performed in step0 with executable, arguments, and queue keywords:

```
sec_spend.ksh 120 sen1_all_1 step0
```

The `wall_clock_time` is important. If you underestimate the time, the job is killed by LoadLeveler before completing the process. Therefore, you need to specify a larger time than the real process time. If you only perform serial jobs in your LoadLeveler cluster, you do not need to overestimate, but, when performing serial jobs and parallel jobs simultaneously, estimating adequate time may be an important factor for the efficient functionality of the backfill scheduler. This will be discussed in another scenario where we will perform both serial jobs and parallel jobs in a LoadLeveler cluster.

6.4.4 Submitting jobs to the LoadLeveler

Before submitting jobs, you should confirm the node status of nodes in your LoadLeveler cluster with the `llstatus` command. The output of the `llstatus` command is shown in the following screen:

```
$ llstatus
Name                      Schedd  InQ Act Startd Run LdAvg Idle Arch      OpSys
sp4n01.msc.itso.ibm.com   Down      0   0 Idle     0 0.08   720 R6000     AIX43
sp4n05.msc.itso.ibm.com   Avail     0   0 Idle     0 0.01  9999 R6000     AIX43
sp4n06.msc.itso.ibm.com   Down      0   0 Idle     0 0.02  9999 R6000     AIX43
sp4n07.msc.itso.ibm.com   Down      0   0 Idle     0 0.00  9999 R6000     AIX43
sp4n08.msc.itso.ibm.com   Down      0   0 Idle     0 0.02  9999 R6000     AIX43
sp4n09.msc.itso.ibm.com   Down      0   0 Idle     0 0.00  1300 R6000     AIX43
sp4n10.msc.itso.ibm.com   Down      0   0 Idle     0 0.01  2433 R6000     AIX43
sp4n11.msc.itso.ibm.com   Down      0   0 Idle     0 0.03  1184 R6000     AIX43
sp4n13.msc.itso.ibm.com   Down      0   0 Idle     0 0.00   725 R6000     AIX43
sp4n15.msc.itso.ibm.com   Avail     0   0 Idle     0 3.00  9999 R6000     AIX43

R6000/AIX43                10 machines      0  jobs      0  running
Total Machines             10 machines      0  jobs      0  running

The Central Manager is defined on sp4n01.msc.itso.ibm.com

The following machine is marked SUBMIT_ONLY
sp4en0.msc.itso.ibm.com

All machines on the machine_list are present.
```

This output shows two scheduler daemons running on sp4n05 and sp4n15. The startd daemon is running on every executing node. At this time, no job is running.

You can also find the available class information by using the `llclass` command, and you can find the current queue information by using the `llq` command. The following is the screen output when we perform the `llclass` and `llq` commands before submitting jobs:

```
$ llclass
Name              MaxJobCPU      MaxProcCPU   Free  Max   Description
                  d+hh:mm:ss     d+hh:mm:ss   Slots Slots

short_limit     -1             -1             10    10
long_limit      -1             -1             10    10
small           -1             -1             40    40
class_parallel  -1               0+20:15:00   10    10    For job_type parallel
class_second      0+02:00:00     0+00:30:00   10    10    Second Class Users
class_first       1+00:00:00   -1             20    20    First Class Users
$ llq
llq: There is currently no job status to report.
```

We can see that class_second has 10 free slots in this output. That means 10 class_second jobs can run in the LoadLeveler cluster at this time, and we can see that no jobs are placed in the queue by the output of the `llq` command.

After confirming the current status, we submit two job command files, sen1_all_1.cmd and sen1_all_2.cmd, with the `llsubmit` command. The following is the output when we submit the command:

```
$ llsubmit sen1_all_1.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.169" with 10 job steps has been submitted
.
$ llsubmit sen1_all_2.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.120" with 10 job steps has been submitted
.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.169.0             tani1      11/9  14:06 R  50  class_second sp4n07
sp4n15.169.9             tani1      11/9  14:06 R  50  class_second sp4n15
sp4n15.169.1             tani1      11/9  14:06 R  50  class_second sp4n09
sp4n15.169.2             tani1      11/9  14:06 R  50  class_second sp4n05
sp4n15.169.3             tani1      11/9  14:06 R  50  class_second sp4n06
sp4n15.169.4             tani1      11/9  14:06 R  50  class_second sp4n10
sp4n15.169.5             tani1      11/9  14:06 R  50  class_second sp4n13
sp4n15.169.6             tani1      11/9  14:06 R  50  class_second sp4n11
sp4n15.169.7             tani1      11/9  14:06 R  50  class_second sp4n01
sp4n15.169.8             tani1      11/9  14:06 R  50  class_second sp4n08
sp4n05.120.2             tani1      11/9  14:07 I  50  class_second
sp4n05.120.3             tani1      11/9  14:07 I  50  class_second
sp4n05.120.4             tani1      11/9  14:07 I  50  class_second
sp4n05.120.5             tani1      11/9  14:07 I  50  class_second
sp4n05.120.1             tani1      11/9  14:07 I  50  class_second
sp4n05.120.6             tani1      11/9  14:07 I  50  class_second
sp4n05.120.0             tani1      11/9  14:07 I  50  class_second
sp4n05.120.7             tani1      11/9  14:07 I  50  class_second
sp4n05.120.8             tani1      11/9  14:07 I  50  class_second
sp4n05.120.9             tani1      11/9  14:07 I  50  class_second

20 job steps in queue, 10 waiting, 0 pending, 10 running, 0 held
```

In the screen output, you can see the queue status from the output of the `llq` command after submitting the jobs. We can see that 10 job steps defined in a job command file are handled by a scheduling node, and each job command file is handled by the different scheduling node. The ID is represented by the hostname of the scheduling node, the assigned job number, and the step number. The 10 job steps are dispatched to 10 nodes. On each node, only one job step is running as the class_second job. The other 10 jobs are placed in the idle state to wait in the queue.

The following screen output shows the queue status we can get with the `llq` command after some time has elapsed:

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.169.1             tani1      11/9  14:06 R  50  class_second sp4n09
sp4n15.169.0             tani1      11/9  14:06 R  50  class_second sp4n07
sp4n15.169.2             tani1      11/9  14:06 R  50  class_second sp4n05
sp4n15.169.3             tani1      11/9  14:06 R  50  class_second sp4n06
sp4n05.120.0             tani1      11/9  14:07 R  50  class_second sp4n15
sp4n05.120.5             tani1      11/9  14:07 R  50  class_second sp4n10
sp4n05.120.1             tani1      11/9  14:07 R  50  class_second sp4n08
sp4n05.120.2             tani1      11/9  14:07 R  50  class_second sp4n01
sp4n05.120.3             tani1      11/9  14:07 R  50  class_second sp4n11
sp4n05.120.4             tani1      11/9  14:07 R  50  class_second sp4n13
sp4n05.120.6             tani1      11/9  14:07 I  50  class_second
sp4n05.120.9             tani1      11/9  14:07 I  50  class_second
sp4n05.120.7             tani1      11/9  14:07 I  50  class_second
sp4n05.120.8             tani1      11/9  14:07 I  50  class_second
sp4n15.169.5             tani1      11/9  14:06 C  50  class_second
sp4n15.169.4             tani1      11/9  14:06 C  50  class_second
sp4n15.169.6             tani1      11/9  14:06 C  50  class_second
sp4n15.169.7             tani1      11/9  14:06 C  50  class_second
sp4n15.169.8             tani1      11/9  14:06 C  50  class_second
sp4n15.169.9             tani1      11/9  14:06 C  50  class_second

14 job steps in queue, 4 waiting, 0 pending, 10 running, 0 held
```

In the screen output we see that six job steps are completed; 10 job steps are running, and four jobs steps are still waiting. In this way, LoadLeveler dispatches the next job successively when one of the jobs that was running is completed and the node is available to execute the class job. This results in using nodes efficiently without leaving nodes unused while controlling the workload of each node.

6.4.5 Submitting a small job command file

In the preceding example, we created two job command files with ten job steps each. In this subsection, we divide the preceding two job command files into 20 job command files that have one job step each. The jobs we want to perform are the same as in the preceding subsection. What we want to know is the difference between submitting a job command file that contains multiple job steps and submitting multiple job command files that each contain single job steps.

The following is one of the 20 job command files we created for this purpose:

```
$ cat sen1_one_0.cmd
# @ step_name = step0
# @ executable = sec_spend.ksh
# @ arguments = 120 sen1_one_1 step0
# @ wall_clock_limit = 125
# @ queue
$
```

Each job command file has a different value for the entry of the `arguments` keyword in order to give different parameters to each job command file.

For submitting 20 job command files, we have to perform `llsubmit` commands 20 times; so, we create the shell script to perform the `llsubmit` command 20 times as shown in the following screen:

```
$ cat sen1_one_all.ksh
#!/bin/ksh
llsubmit sen1_one_0.cmd
llsubmit sen1_one_1.cmd
llsubmit sen1_one_2.cmd
llsubmit sen1_one_3.cmd
llsubmit sen1_one_4.cmd
llsubmit sen1_one_5.cmd
llsubmit sen1_one_6.cmd
llsubmit sen1_one_7.cmd
llsubmit sen1_one_8.cmd
llsubmit sen1_one_9.cmd
llsubmit sen1_one_0.cmd
llsubmit sen1_one_1.cmd
llsubmit sen1_one_2.cmd
llsubmit sen1_one_3.cmd
llsubmit sen1_one_4.cmd
llsubmit sen1_one_5.cmd
llsubmit sen1_one_6.cmd
llsubmit sen1_one_7.cmd
llsubmit sen1_one_8.cmd
llsubmit sen1_one_9.cmd
$
```

With this `sen1_one_all.ksh` shell script, we submit 20 job command files. You can see the queue status after submitting 20 job command files using the script in the following screen:

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.132.0             tani1      11/9  14:27 R  50  class_second sp4n09
sp4n15.183.0             tani1      11/9  14:27 R  50  class_second sp4n08
sp4n15.182.0             tani1      11/9  14:27 R  50  class_second sp4n13
sp4n05.133.0             tani1      11/9  14:27 R  50  class_second sp4n07
sp4n05.134.0             tani1      11/9  14:27 R  50  class_second sp4n05
sp4n15.184.0             tani1      11/9  14:27 R  50  class_second sp4n11
sp4n05.135.0             tani1      11/9  14:27 R  50  class_second sp4n10
sp4n15.186.0             tani1      11/9  14:27 R  50  class_second sp4n15
sp4n05.136.0             tani1      11/9  14:27 R  50  class_second sp4n01
sp4n15.185.0             tani1      11/9  14:27 R  50  class_second sp4n06
sp4n15.187.0             tani1      11/9  14:27 I  50  class_second
sp4n05.137.0             tani1      11/9  14:27 I  50  class_second
sp4n05.138.0             tani1      11/9  14:27 I  50  class_second
sp4n15.188.0             tani1      11/9  14:27 I  50  class_second
sp4n05.139.0             tani1      11/9  14:27 I  50  class_second
sp4n05.140.0             tani1      11/9  14:27 I  50  class_second
sp4n15.190.0             tani1      11/9  14:27 I  50  class_second
sp4n15.189.0             tani1      11/9  14:27 I  50  class_second
sp4n05.141.0             tani1      11/9  14:27 I  50  class_second
sp4n15.191.0             tani1      11/9  14:27 I  50  class_second

20 job steps in queue, 10 waiting, 0 pending, 10 running, 0 held
$
```

You can see the difference in the ID columns. Each job command file is assigned a job ID with a step ID 0. And you can see the behavior by which the LoadLeveler distributes the submission to multiple scheduling nodes equally. The nodes sp4n05 and sp4n15 have same number of jobs to schedule. But there is no difference about dispatching job steps to nodes. The LoadLeveler distributes the first 10 jobs to 10 nodes with job step units.

If you want to manage job steps as a group in the queue, we recommend that you write multiple steps in a job command file. You can manage each job step with the `llprio` and `llcancel` commands by specifying hostname.job-id.step-id in the following way:

```
llprio +30 sp4n15.310.9
llcancel sp4n15.310.9
```

You can also manage all the job steps within a job as a group by using the commands without specifying the job step ID:

```
llprio +30 sp4n15.310
llcancel sp4n15.310
```

The following output shows the job status after submitting two job command files that include five job steps each:

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.270.1             tani1      11/12 17:18 R  50  class_second sp4n07
sp4n05.270.0             tani1      11/12 17:18 R  50  class_second sp4n11
sp4n05.270.2             tani1      11/12 17:18 R  50  class_second sp4n05
sp4n05.270.4             tani1      11/12 17:18 R  50  class_second sp4n06
sp4n05.270.3             tani1      11/12 17:18 R  50  class_second sp4n08
sp4n15.313.0             tani1      11/12 17:18 R  50  class_second sp4n01
sp4n15.313.4             tani1      11/12 17:18 R  50  class_second sp4n15
sp4n15.313.1             tani1      11/12 17:18 R  50  class_second sp4n10
sp4n15.313.2             tani1      11/12 17:18 R  50  class_second sp4n13
sp4n15.313.3             tani1      11/12 17:18 R  50  class_second sp4n09

10 job steps in queue, 0 waiting, 0 pending, 10 running, 0 held
$ llcancel sp4n15.313.2
llcancel: Cancel command has been sent to the central manager.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.270.1             tani1      11/12 17:18 R  50  class_second sp4n07
sp4n05.270.0             tani1      11/12 17:18 R  50  class_second sp4n11
sp4n05.270.2             tani1      11/12 17:18 R  50  class_second sp4n05
sp4n05.270.4             tani1      11/12 17:18 R  50  class_second sp4n06
sp4n05.270.3             tani1      11/12 17:18 R  50  class_second sp4n08
sp4n15.313.0             tani1      11/12 17:18 R  50  class_second sp4n01
sp4n15.313.4             tani1      11/12 17:18 R  50  class_second sp4n15
sp4n15.313.1             tani1      11/12 17:18 R  50  class_second sp4n10
sp4n15.313.3             tani1      11/12 17:18 R  50  class_second sp4n09
sp4n15.313.2             tani1      11/12 17:18 CA 50  class_second

9 job steps in queue, 0 waiting, 0 pending, 9 running, 0 held
$ llcancel sp4n05.270
llcancel: Cancel command has been sent to the central manager.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.313.1             tani1      11/12 17:18 R  50  class_second sp4n10
sp4n15.313.0             tani1      11/12 17:18 R  50  class_second sp4n01
sp4n15.313.4             tani1      11/12 17:18 R  50  class_second sp4n15
sp4n15.313.3             tani1      11/12 17:18 R  50  class_second sp4n09
sp4n15.313.2             tani1      11/12 17:18 CA 50  class_second

4 job steps in queue, 0 waiting, 0 pending, 4 running, 0 held
$
```

You can see from the screen output that you can cancel each job step with the `llcancel` command by specifying the step ID, and you can cancel the job step as a group by using the command without specifying the step ID.

It is easier and more convenient to describe multiple job steps in a job command file than to prepare multiple job files.

Also, if your jobs have dependency, you can manage them by describing them in a job command file. This is discussed in the following section.

6.5 Executing batch jobs with dependency on each other

In some cases, you may need to perform batch jobs sequentially because of the dependency between jobs. You can specify the dependency between job steps in the job command file by using the dependency keyword. In this section, we submit a job command file that has the dependency keyword to LoadLeveler, and we show how LoadLeveler manages these dependencies.

We have seven job steps to perform. We can perform steps 0, 1, 2, 4, and 5 5 simultaneously, but step 3 should be performed after steps 0, 1, and 2 are completed successfully with exit code 0. Also, job step 6 should be performed after steps 4 and 5 are completed successfully with exit code 0. We can perform steps 3 and 6 simultaneously.

The job command file is shown in the following screen:

```
$ cat depend1.cmd
#  Dependency cmd
#
# @ job_name = depend_1
# @ step_name = step0
# @ executable = sec_spend_exit.ksh
# @ arguments = 120 depend_1 step0 0
# @ wall_clock_limit = 125
# @ queue
# @ step_name = step1
# @ executable = sec_spend_exit.ksh
# @ arguments = 110 depend_1 step1 0
# @ wall_clock_limit = 115
# @ queue
# @ step_name = step2
# @ executable = sec_spend_exit.ksh
# @ arguments = 100 depend_1 step2 0
# @ wall_clock_limit = 105
# @ queue
# @ dependency = ( step0 == 0 ) && ( step1 == 0 ) && ( step2 == 0 )
# @ step_name = step3
# @ executable = sec_spend_exit.ksh
# @ arguments = 90 depend_1 step3 0
# @ wall_clock_limit = 95
# @ queue
# @ step_name = step4
# @ executable = sec_spend_exit.ksh
# @ arguments = 80 depend_1 step4 0
# @ wall_clock_limit = 85
# @ queue
# @ step_name = step5
# @ executable = sec_spend_exit.ksh
# @ arguments = 70 depend_1 step5 0
# @ wall_clock_limit = 75
# @ queue
# @ dependency = ( step4 == 0 ) && ( step5 == 0 )
# @ step_name = step6
# @ executable = sec_spend_exit.ksh
# @ arguments = 60 depend_1 step6 0
# @ wall_clock_limit = 65
# @ queue
```

You can see dependency keywords in job steps 3 and 6. The dependency keyword in step 3 means that step 3 is only performed when the exit codes of steps 0, 1, and 2 are all equal to 0. The keyword in step 6 means that step 6 is only performed when the exit codes of step 4 and step 5 are equal to 0.

The following screen output shows the queue status after submitting the job command file shown in the preceding screen:

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.142.1             tani1      11/9  14:53 R  50  class_second sp4n06
sp4n05.142.0             tani1      11/9  14:53 R  50  class_second sp4n09
sp4n05.142.2             tani1      11/9  14:53 R  50  class_second sp4n11
sp4n05.142.5             tani1      11/9  14:53 R  50  class_second sp4n13
sp4n05.142.4             tani1      11/9  14:53 R  50  class_second sp4n08
sp4n05.142.3             tani1      11/9  14:53 NQ 50  class_second
sp4n05.142.6             tani1      11/9  14:53 NQ 50  class_second

7 job steps in queue, 0 waiting, 0 pending, 5 running, 2 held
$
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.142.1             tani1      11/9  14:53 R  50  class_second sp4n06
sp4n05.142.0             tani1      11/9  14:53 R  50  class_second sp4n09
sp4n05.142.2             tani1      11/9  14:53 R  50  class_second sp4n11
sp4n05.142.6             tani1      11/9  14:53 R  50  class_second sp4n08
sp4n05.142.3             tani1      11/9  14:53 NQ 50  class_second
sp4n05.142.5             tani1      11/9  14:53 C  50  class_second
sp4n05.142.4             tani1      11/9  14:53 C  50  class_second

5 job steps in queue, 0 waiting, 0 pending, 4 running, 1 held
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.142.6             tani1      11/9  14:53 R  50  class_second sp4n08
sp4n05.142.3             tani1      11/9  14:53 R  50  class_second sp4n11
sp4n05.142.2             tani1      11/9  14:53 C  50  class_second
sp4n05.142.1             tani1      11/9  14:53 C  50  class_second
sp4n05.142.0             tani1      11/9  14:53 C  50  class_second
sp4n05.142.4             tani1      11/9  14:53 C  50  class_second
sp4n05.142.5             tani1      11/9  14:53 C  50  class_second

2 job steps in queue, 0 waiting, 0 pending, 2 running, 0 held
```

The first output of the `llq` command shows that steps 0, 1, 2, 4, and 5 are running simultaneously on different nodes, and steps 3 and 6 are placed in the Not Queued (NQ) state because these step have the dependency keyword.

The second output of the `llq` command shows that steps 4 and 5 have been completed and step 6 is running on node sp4n08. However, steps 0, 1, and 2 are still running. Step 3 is still placed in the Not Queued state.

The third output shows that steps 0, 1, and 3 have been completed; so, step 3 has started running.

These results show that the LoadLeveler dispatches jobs to nodes to process in parallel if the job steps do not have the dependency, but, if you specify the dependency in the job command file, the LoadLeveler manages the dependency and dispatches the job when the dependency is satisfied.

We will now show how the LoadLeveler manages job steps if the dependency cannot be satisfied. We tested the case in which step 5 returns the value 1 as exit code. The following screen shows the queue status after step 5 completes with exit code 1:

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.193.0             tani1      11/9  15:15 R  50  class_second sp4n09
sp4n15.193.1             tani1      11/9  15:15 R  50  class_second sp4n05
sp4n15.193.2             tani1      11/9  15:15 R  50  class_second sp4n06
sp4n15.193.3             tani1      11/9  15:15 NQ 50  class_second
sp4n15.193.6             tani1      11/9  15:15 NR 50  class_second
sp4n15.193.5             tani1      11/9  15:15 C  50  class_second
sp4n15.193.4             tani1      11/9  15:15 C  50  class_second

4 job steps in queue, 0 waiting, 0 pending, 3 running, 1 held

$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.193.3             tani1      11/9  15:15 R  50  class_second sp4n09
sp4n15.193.6             tani1      11/9  15:15 NR 50  class_second
sp4n15.193.0             tani1      11/9  15:15 C  50  class_second
sp4n15.193.1             tani1      11/9  15:15 C  50  class_second
sp4n15.193.4             tani1      11/9  15:15 C  50  class_second
sp4n15.193.2             tani1      11/9  15:15 C  50  class_second
sp4n15.193.5             tani1      11/9  15:15 C  50  class_second

1 job steps in queue, 0 waiting, 0 pending, 1 running, 0 held
$ llq
llq: There is currently no job status to report.
```

The first output shows that steps 4 and 5 have been completed, but step 6 is placed in the NR state. The NR state stands for the Not Run state; so, step 6 does not run because the dependency is not satisfied.

The second output shows that step3 is running because the dependency is satisfied for step3.

The third output shows that the NR state does not remain in the queue after all other steps in the job command file are completed.

Chapter 7. Managing parallel jobs

This chapter describes some scenarios for managing parallel jobs. The parallel jobs are written with the Message Passing Interface (MPI) library on Parallel Operating Environment (POE) V2.4. We show an example of submitting the POE jobs with the LoadLeveler V2.1.

7.1 Scenario description

We discuss the following scenarios:

- Executing multiple size parallel jobs

  You may have multiple parallel jobs, and each job may request a different number of resources and different times for processing the job. In this scenario, we prepare three parallel jobs of different sizes and show how LoadLeveler schedules these jobs.

- Executing multiple parallel jobs using a communication subsystem

  You can request a communication adapter and a communication subsystem mode for a job by specifying the request in the job command file. We show how LoadLeveler schedules the jobs that specify the network requests.

- Interactive POE Job

  When you perform a POE Job interactively, you can manage resource information with LoadLeveler. POE contacts LoadLeveler to know the resource and manages the job. In this scenario, we will show how LoadLeveler manages the job.

7.2 Tools choice

We discuss submitting parallel jobs with the following products:

- PSSP V3.1.1 and AIX V4.3.3
- LoadLeveler V2.1
- Parallel Environment (PE) for AIX V2.4

Parallel Environment for AIX V2.4 includes the Parallel Operating Environment (POE) as one of its component. In an earlier release of PE, POE relied on the SP Resource Management to perform job management functions. This function keeps track of available or allocated nodes and loading the switch tables for programs performing User Space

communications. LoadLeveler V2.1 provides the resource management function. POE can get node and adapter information from LoadLeveler. For the User Space communication, LoadLeveler is able to load and unload to the Job Switch Resource Table (JSRT) and provide access to the JSRT to POE; so, LoadLeveler V2.1 works as the job management system for POE V2.4.

The benefits of using LoadLeveler V2.1 with POE V2.4 on RS/6000 SP include:

- LoadLeveler V2.1 supports multiple processes that use user space protocols with the SP switch. Each SP switch adapter has four adapter windows available for using user space protocol.
- LoadLeveler Version 2.1 allows a job step to request the network resources required for a task to run both the Message Passing Interface (MPI) and the Low-level Application Programming Interface (LAPI). MPI runs in both IP or user space mode, but LAPI can only run in user space mode.
- The new scheduler, called the Backfill scheduler, provides a backfill capability for scheduling multiple size parallel jobs. The Backfill scheduler schedules small jobs while waiting for the start time of any large job requiring many nodes. This scheduler allocates the nodes more efficiently by dispatching smaller jobs while waiting for the resources for large jobs.
- LoadLeveler V2.1 works with POE V2.4 for interactive jobs.
- LoadLeveler V2.1 allows multiple tasks of a parallel job to run on the same node.

7.3 Environment for processing parallel jobs

You need to have a common user name space of the same kind as when processing serial jobs. Refer to Section 6.3, "Considerations for the executing environment" on page 135, for information on creating a common user name space.

You have to edit the machine stanza and the adapter stanza in the administration file for users to request the network adapter in the job command file. Refer to Section 3.4.6, "Parallel job" on page 64, for information on the environment of parallel jobs.

We create a class for processing parallel jobs in our environment. All users can use this class. The following are the definitions for the classes in the admin file:

```
class_parallel: type = class
                class_comment = "For job_type parallel"
                max_node = 5
                total_tasks = 15
                wall_clock_limit = 24:00:00,12:00:00
                cpu_limit = 20:15:00
```

All parallel jobs discussed in this chapter will use this class.

7.4 Executing multiple size parallel jobs

LoadLeveler V2.1 has two types of scheduler: The default scheduler and the Backfill scheduler. If you do not change the default configuration in the configuration file, the Backfill scheduler is selected with the SCHEDULER_TYPE keyword. The following screen displays the scheduler selection part of the configuration file:

```
# For Backfill Scheduler
SCHEDULER_API= NO
SCHEDULER_TYPE = BACKFILL
# For default scheduler
# SCHEDULER_API= NO
# SCHEDULER_TYPE =
# For external scheduler
# SCHEDULER_API= YES
# SCHEDULER_TYPE =
```

In the screen output, the Backfill scheduler is selected. If you comment out SCHEDULER_TYPE = BACKFILL and activate the SCHEDULER_TYPE = statement in the file, the default scheduler is activated. You can use the job control API to use an external scheduler for site-specific requirements. The SCHEDULER_API = YES is used to enable the job control API.

In this section, we discuss how these schedulers schedule varying sizes of parallel jobs. The scheduler has to allocate multiple numbers of nodes depending on the size of parallel jobs. If you have various sizes of parallel jobs, scheduling is a complex task.

We prepare the following job command files for submitting multiple sizes parallel jobs:

- n4t110.cmd

This parallel job needs 4 nodes. We estimate the elapsed time for this job will be running at 110 sec. We set the value 110 to wall_clock_limit keyword in the job command file. The following is the output screen of n4t110.cmd file with cat command:

```
$ cat n4t110.cmd
### Small Parallel Job 2 node 1 minute
### need 4 node
### wall clock limit 110 sec
# @ job_type=parallel
# @ environment = COPY_ALL; MP_TIMEOUT=2000;
# @ error   = btat_test.$(Host).$(Cluster).$(Process).err
# @ output  = btat_test.$(Host).$(Cluster).$(Process).out
# @ wall_clock_limit = 110,110
# @ network.mpi = css0,not_shared,ip
# @ node = 4
# @ tasks_per_node = 1
# @ executable = /bin/poe
# @ arguments = /u/tani1/SCENARIO_PARA/btat -d 90 -t 1 -m 1000 -v -labelio yes
# @ class = class_parallel
# @ queue
```

- n5110.cmd

  This parallel job needs five nodes. We estimate the elapsed time for this job will be running at 110 seconds. We set the value of 110 to the wall_clock_limit keyword in the job command file. The following is the screen output of the n5t110.cmd file with the `cat` command:

```
$ cat n5t110.cmd
### Small Parallel Job 2 node 1 minute
### need 5 node
### wall clock limit 110 sec
# @ job_type=parallel
# @ environment = COPY_ALL; MP_TIMEOUT=2000;
# @ error   = btat_test.$(Host).$(Cluster).$(Process).err
# @ output  = btat_test.$(Host).$(Cluster).$(Process).out
# @ wall_clock_limit = 110,110
# @ network.mpi = css0,not_shared,ip
# @ node = 5
# @ tasks_per_node = 1
# @ executable = /bin/poe
# @ arguments = /u/tani1/SCENARIO_PARA/btat -d 90 -t 1 -m 1000 -v -labelio yes
# @ class = class_parallel
# @ queue
```

- n2t60.cmd

  This parallel job needs two nodes. We estimate the elapsed time for this job will be running at 60 seconds. We set a value of 60 to the

wall_clock_limit keyword in the job command file. The following is the screen output of the n2t60.cmd file with the `cat` command:

```
$ cat n2t60.cmd
### Small Parallel Job 2 node 1 minute
### need 2 node
### wall clock limit 60 sec
# @ job_type=parallel
# @ environment = COPY_ALL; MP_TIMEOUT=2000;
# @ error   = btat_test.$(Host).$(Cluster).$(Process).err
# @ output  = btat_test.$(Host).$(Cluster).$(Process).out
# @ wall_clock_limit = 60,60
# @ network.mpi = css0,not_shared,ip
# @ node = 2
# @ tasks_per_node = 1
# @ executable = /bin/poe
# @ arguments = /u/tani1/SCENARIO_PARA/btat -d 30 -t 1 -m 1000 -v -labelio yes
# @ class = class_parallel
# @ queue
```

For executing these parallel jobs, we suppose that we only have six nodes, and each node can execute only one job task for the class_parallel job. The following screen is the output of the `llclass` command:

```
$ llclass
Name              MaxJobCPU      MaxProcCPU   Free  Max   Description
                  d+hh:mm:ss     d+hh:mm:ss   Slots Slots

interactive     -1             -1             2     2
inter_class     -1             -1             2     2
short_limit     -1             -1             10    10
small           -1             -1             37    37
long_limit      -1             -1             10    10
class_parallel  -1               0+20:15:00   6     6     For job_type parallel
class_second      0+02:00:00     0+00:30:00   10    10    Second Class Users
class_first       1+00:00:00   -1             20    20    First Class Users
$ llq
llq: There is currently no job status to report.
```

You can see that we have six slots for the class_parallel job in our LoadLeveler cluster. That is, if a job is submitted with the n5t110.cmd file and is in run state, other jobs cannot run simultaneously.

The another configuration for scheduling jobs is the job priority determined by the SYSPRIO keyword in the configuration file. Refer to Section 3.4.4, "Job priority" on page 57, to get the detail information of this keyword.

Our configuration of the keyword throughout this chapter is the following default setting:

```
SYSPRIO: 0 - (QDate)
```

This means the job priority is assigned based on FIFO.

7.4.1 With the Backfill scheduler (Case 1)

We will discuss how the Backfill scheduler schedules jobs of various sizes in this section with the job command files described in the preceding section.

The jobs are submitted in the following order:

1. n4n110.cmd
2. n5t110.cmd
3. n2t60.cmd
4. n2t60.cmd

The following screen output shows the status of the jobs after they are submitted:

```
$ llsubmit n4t110.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.627" has been submitted.
$ llsubmit n5t110.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.628" has been submitted.
$ llsubmit n2t60.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.629" has been submitted.
$ llsubmit n2t60.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.424" has been submitted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.627.0             tani1      11/16 13:09 R  50  class_parall sp4n08
sp4n05.629.0             tani1      11/16 13:10 R  50  class_parall sp4n07
sp4n05.628.0             tani1      11/16 13:09 I  50  class_parall
sp4n15.424.0             tani1      11/16 13:10 I  50  class_parall

4 job steps in queue, 2 waiting, 0 pending, 2 running, 0 held
```

In this screen output, you can see that the scheduler receives and assigns the job ID to each job. The `n4t110.cmd` command is submitted first and assigned the ID sp4n05.627.0. The n5t110.cmd job is submitted second and assigned the ID sp4n05.628.0. The n2t60.cmd job is submitted third and assigned the ID sp4n05.629.0. The n2t60.cmd job is submitted again as the fourth job and assigned the ID sp4n15.424.0. The output of the `llq` command

shows that both sp4n05.627.0 and sp4n05.629.0 are currently running. That is, the Backfill scheduler scheduled the execution of sp4n05.629.0 before sp4n05.628.0 and used two nodes for sp4n05.629.0 in addition to four nodes for sp4n05.627.0. The scheduler can estimate the maximum time for job completion of n2t60.cmd at 60 seconds with the wall_clock_limit; so, the job can run and complete the task while waiting for the the n4t110.cmd job to complete.

The following screen output shows the result of the `llq` command that we can get after some intervals:

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.627.0             tani1      11/16 13:09 R  50  class_parall sp4n08
sp4n05.628.0             tani1      11/16 13:09 I  50  class_parall
sp4n15.424.0             tani1      11/16 13:10 I  50  class_parall

3 job steps in queue, 2 waiting, 0 pending, 1 running, 0 held
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.628.0             tani1      11/16 13:09 R  50  class_parall sp4n07
sp4n15.424.0             tani1      11/16 13:10 I  50  class_parall

2 job steps in queue, 1 waiting, 0 pending, 1 running, 0 held
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.424.0             tani1      11/16 13:10 R  50  class_parall sp4n07

1 job steps in queue, 0 waiting, 0 pending, 1 running, 0 held
$
```

This screen output shows that `sp4n05.629.0` completed while `sp4n05.627.0` was still running. The two nodes becametemporarily idle; however, the scheduler did not dispatch the `sp4n15.424.0` job to two nodes before dispatching `sp4n05.628.0`. The scheduler waits for the completion of job `sp4n05.627.0` and dispatches `sp4n05.628.0` next.

The wall clock limit of `sp4n15.424.0` is 60 seconds, and that of `sp4n05.627.0` is 110 seconds. If `sp4n15.424.0` is dispatched prior to `sp4n05.628.0`, the `sp4n05.628.0` may need to wait to finish the `sp4n15.424.0` to get five nodes.

7.4.2 With the Backfill scheduler (Case 2)

We submit the same jobs in the same order as the ones described in the preceding subsection. Since we know the real elapsed time to complete each job from the first experiment, we change the wall clock limit to the small size of jobs that need two nodes. We renamed the job command file and changed the value of the `wall_clock_limit` in the following way:

```
$ cat n2t40.cmd
### Small Parallel Job 2 node 1 minute
### need 2 node
### wall clock limit 40 sec
# @ job_type=parallel
# @ environment = COPY_ALL; MP_TIMEOUT=2000;
# @ error   = btat_test.$(Host).$(Cluster).$(Process).err
# @ output  = btat_test.$(Host).$(Cluster).$(Process).out
# @ wall_clock_limit = 40,40
# @ network.mpi = css0,not_shared,ip
# @ node = 2
# @ tasks_per_node = 1
# @ executable = /bin/poe
# @ arguments = /u/tani1/SCENARIO_PARA/btat -d 30 -t 1 -m 1000 -v -labelio yes
# @ class = class_parallel
# @ queue
```

The bold character shows the difference between the n2t40.cmd and the n2t60.cmd. We submit the n2t40.cmd job command file as the alternative to n2t60.cmd.

The following is the output when we submitted job command files:

```
$ llsubmit n4t110.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.425" has been submitted.
$ llsubmit n5t110.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.426" has been submitted.
$ llsubmit n2t40.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.630" has been submitted.
$ llsubmit n2t40.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.631" has been submitted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.425.0             tani1      11/16 13:14 R  50  class_parall sp4n07
sp4n05.630.0             tani1      11/16 13:14 R  50  class_parall sp4n08
sp4n15.426.0             tani1      11/16 13:14 I  50  class_parall
sp4n05.631.0             tani1      11/16 13:14 I  50  class_parall

4 job steps in queue, 2 waiting, 0 pending, 2 running, 0 held
```

In the screen output, you can see that the n4t110.cmd job is submitted first and assigned the ID of sp4n15.625.0. The n5t110.cmd job is submitted second and assigned the ID of sp4n15.626.0. The n2t40.cmd job is submitted third and assigned the ID of sp4n05.630.0. The n2t40.cmd job is submitted again as the fourth job and assigned the ID of sp4n05.631.0. The output of the `llq` command shows that both sp4n15.625.0 and sp4n05.630.0 are running in our LoadLeveler cluster.

The following screen shows the `llq` command after some intervals:

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.425.0             tani1      11/16 13:14 R  50  class_parall sp4n07
sp4n05.631.0             tani1      11/16 13:14 R  50  class_parall sp4n08
sp4n15.426.0             tani1      11/16 13:14 I  50  class_parall

3 job steps in queue, 1 waiting, 0 pending, 2 running, 0 held
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.426.0             tani1      11/16 13:14 R  50  class_parall sp4n08

1 job steps in queue, 0 waiting, 0 pending, 1 running, 0 held
$
```

The first output shows that, while executing `sp4n15.425.0`, the job `sp4n05.631.0` was allocated two nodes. The last submitted job was `sp4n15.426.0`. The wall clock limit of `sp4n05.630.0` and `sp4n05.631.0` is 40 seconds, and that of `sp4n15.425.0` is 110 seconds. Thus, the maximum total time for processing both jobs that require two nodes is shorter than the latter job that requires four nodes.

You can see the following characteristics in these scenarios:

- The Backfill scheduler backfills the small size jobs if the required resource is matched to the small size jobs while the large jobs wait for the resources.
- Although the Backfill scheduler backfills the small size of jobs, the Backfill scheduler honors the job priority. The starting of the highest priority job is never delayed.
- The Backfill scheduler knows the maximum processing time by the value of the wall_clock_time keyword.

If you want your job to run quickly, you should submit the job with fewer node requests and shorter wall clock limit time. Note that overestimating the time is a disadvantage for your job from this point of view; however, underestimating results in the termination of your job before it completes.

7.4.3 With the default scheduler

In this section, we will discuss how the default scheduler schedules the jobs when we submit the same job command files we used with the Backfill scheduler.

To use the default scheduler, we chang the SCHEDULER_TYPE keyword entry in the admin file:

```
# For Backfill Scheduler
# SCHEDULER_API= NO
# SCHEDULER_TYPE = BACKFILL
# For default scheduler
SCHEDULER_API= NO
SCHEDULER_TYPE =
# For external scheduler
# SCHEDULER_API= YES
# SCHEDULER_TYPE =
```

We submit the same job command file in the same order as we used with the Backfill scheduler. You can see the output that resulted when we submitted to the default scheduler in the following screen:

```
$ llq
llq: There is currently no job status to report.
$ llsubmit n4t110.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.624" has been submitted.
$ llsubmit n5t110.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.625" has been submitted.
$ llsubmit n2t60.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.422" has been submitted.
$ llsubmit n2t60.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.626" has been submitted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.624.0             tani1      11/16 13:03 R  50  class_parall sp4n07
sp4n05.625.0             tani1      11/16 13:03 I  50  class_parall
sp4n15.422.0             tani1      11/16 13:03 I  50  class_parall
sp4n05.626.0             tani1      11/16 13:03 I  50  class_parall

4 job steps in queue, 3 waiting, 0 pending, 1 running, 0 held
```

The job IDs are assigned to jobs. The default scheduler does not schedule small job `sp4n15.422.0` although two nodes are available while executing `sp4n05.624.0` with four nodes.

The following is the screen output of the `llq` command after some interval:

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.625.0             tani1      11/16 13:03 R  50  class_parall sp4n06
sp4n15.422.0             tani1      11/16 13:03 I  50  class_parall
sp4n05.626.0             tani1      11/16 13:03 I  50  class_parall

3 job steps in queue, 2 waiting, 0 pending, 1 running, 0 held
```

This output shows that the second job the scheduler dispatched was `sp4n05.625.0`. That is, when executing the first job with four nodes, two nodes were idle, and jobs with two node requirements were waiting in the queue. As you can see, the Backfill scheduler is the preferred one, especially for executing jobs in these simple scenarios.

7.5 Executing multiple parallel jobs specifying network types

You can specify the network requirement for your parallel jobs with the network keyword in the job command file. In this scenario, we show how LoadLeveler allocates the network resource for parallel jobs depending on the request and how this network request affects the execution of parallel jobs.

To allow users to execute multiple class_parallel jobs, we changed the entry of the Class keyword in the local configuration file. Now, we have four nodes that can execute six class_parallel jobs. The following is the output of the `llclass` command:

```
$ llclass
Name             MaxJobCPU      MaxProcCPU   Free  Max   Description
                 d+hh:mm:ss     d+hh:mm:ss   Slots Slots

interactive     -1             -1            2     2
inter_class     -1             -1            2     2
short_limit     -1             -1            10    10
small           -1             -1            37    37
long_limit      -1             -1            10    10
class_parallel  -1               0+20:15:00  24    24    For job_type parallel
class_second      0+02:00:00     0+00:30:00  10    10    Second Class Users
class_first       1+00:00:00   -1            20    20    First Class Users
```

You can see that 24 slots are available for class_parallel jobs.

7.5.1 User space shared mode

At first, we run multiple parallel jobs written with the MPI library. A job needs four nodes, and one task runs on each node. We want to communicate with user space mode through the SP switch. We share the switch adapter with tasks of other job steps. The following screen output shows the job command file to submit this job:

```
$ cat us_share_css0.cmd
# @ job_type=parallel
# @ environment = COPY_ALL; MP_TIMEOUT=2000;
# @ error   = btat_test.$(Host).$(Cluster).$(Process).err
# @ output  = btat_test.$(Host).$(Cluster).$(Process).out
# @ wall_clock_limit = 180,180
# @ network.mpi = css0,shared,us
# @ node = 4
# @ tasks_per_node = 1
# @ executable = /bin/poe
# @ arguments = /u/tani1/SCENARIO_PARA/btat -d 40 -t 1 -m 1000 -v -labelio yes
# @ class = class_parallel
# @ queue
```

The **network** keyword means that we need a css0 adapter that can be shared with other tasks and communicate with MPI using user space mode. The **node** keyword means we need four nodes for this job, and **tasks_per_node** means we run one task on each node.

We submit this job command file six times. The following screen output shows the result when we submitted the job command file:

```
$ llsubmit us_share_css0.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.660" has been submitted.
$ llsubmit us_share_css0.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.661" has been submitted.
$ llsubmit us_share_css0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.456" has been submitted.
$ llsubmit us_share_css0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.457" has been submitted.
$ llsubmit us_share_css0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.458" has been submitted.
$ llsubmit us_share_css0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.459" has been submitted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.660.0             tani1      11/16 16:40 R  50  class_parall sp4n09
sp4n05.661.0             tani1      11/16 16:40 R  50  class_parall sp4n09
sp4n15.456.0             tani1      11/16 16:40 R  50  class_parall sp4n09
sp4n15.457.0             tani1      11/16 16:40 R  50  class_parall sp4n09
sp4n15.458.0             tani1      11/16 16:40 I  50  class_parall
sp4n15.459.0             tani1      11/16 16:40 I  50  class_parall

6 job steps in queue, 2 waiting, 0 pending, 4 running, 0 held
```

The screen output shows that after six job command files are submitted, four jobs are running and two jobs are waiting in the job queue. The following screen output shows the result of the `ps` command using distributed shell on the CWS to see tasks running on each node:

```
# dsh -a ps -ef | grep btat | grep -v grep | awk '{print $1,$2,$6,$9}'
sp4n06: tani1 16:40:16 /u/tani1/SCENARIO_PARA/btat
sp4n06: tani1 16:40:18 /u/tani1/SCENARIO_PARA/btat
sp4n06: tani1 16:40:15 /u/tani1/SCENARIO_PARA/btat
sp4n06: tani1 16:40:12 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:40:15 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:40:12 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:40:16 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:40:18 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:40:18 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:40:14 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:40:12 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:40:16 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:40:16 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:40:12 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:40:15 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:40:18 /u/tani1/SCENARIO_PARA/btat
#
```

You can see four tasks running on each node. For user space mode communication, four adapter windows can be allocated on each node. The LoadLeveler allows you to run up to four user space tasks per node simultaneously.

7.5.2 Non-shared user space mode

We change the entry of the network keyword in the job command file for requesting LoadLeveler for dedicated use of the switch adapter. You can see this change in the following screen output:

```
$ cat us_notshare_css0.cmd
# @ job_type=parallel
# @ environment = COPY_ALL; MP_TIMEOUT=2000;
# @ error   = btat_test.$(Host).$(Cluster).$(Process).err
# @ output  = btat_test.$(Host).$(Cluster).$(Process).out
# @ wall_clock_limit = 180,180
# @ network.mpi = css0,not_shared,us
# @ node = 4
# @ tasks_per_node = 1
# @ executable = /bin/poe
# @ arguments = /u/tani1/SCENARIO_PARA/btat -d 40 -t 1 -m 1000 -v -labelio yes
# @ class = class_parallel
# @ queue
```

not_shared means that the LoadLeveler reserves the adapter for this job and the adapter is not shared with other jobs that use this adapter with user space mode. If another job uses the adapter with IP mode, the job can use this adapter.

We submit the job command file four times. The following screen shows the output that resulted after we submitted the jobs:

```
$ llq
llq: There is currently no job status to report.
$ llsubmit us_notshare_css0.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.662" has been submitted.
$ llsubmit us_notshare_css0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.460" has been submitted.
$ llsubmit us_notshare_css0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.461" has been submitted.
$ llsubmit us_notshare_css0.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.663" has been submitted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.662.0             tani1      11/16 16:43 R  50  class_parall sp4n09
sp4n15.460.0             tani1      11/16 16:43 I  50  class_parall
sp4n15.461.0             tani1      11/16 16:43 I  50  class_parall
sp4n05.663.0             tani1      11/16 16:43 I  50  class_parall

4 job steps in queue, 3 waiting, 0 pending, 1 running, 0 held
$
```

In this case, only one job step can run. Other jobs are waiting for the job to complete, and you can see the output of the `ps` command with dsh on the CWS in the following screen output:

```
# dsh -a ps -ef | grep btat | grep -v grep | awk '{print $1,$2,$6,$9}'
sp4n06: tani1 16:43:28 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:43:29 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:43:28 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:43:28 /u/tani1/SCENARIO_PARA/btat
```

7.5.3 IP mode over a switch in shared mode

If you want to use the switch with IP mode, you can issue a request to LoadLeveler by changing the network keyword. You can see the change in the job command file from the following screen output in bold.

```
$ cat ip_share_css0.cmd
# @ job_type=parallel
# @ environment = COPY_ALL; MP_TIMEOUT=2000;
# @ error   = btat_test.$(Host).$(Cluster).$(Process).err
# @ output  = btat_test.$(Host).$(Cluster).$(Process).out
# @ wall_clock_limit = 180,180
# @ network.mpi = css0,shared,ip
# @ node = 4
# @ tasks_per_node = 1
# @ executable = /bin/poe
# @ arguments = /u/tani1/SCENARIO_PARA/btat -d 40 -t 1 -m 1000 -v -labelio yes
# @ class = class_parallel
# @ queue
```

The job command file requests the css0 adapter in IP mode that is shared with other job steps.

The following screen shows the output that resulted after we submitted the job command file five times:

```
$ llq
llq: There is currently no job status to report.
$ llsubmit  ip_share_css0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.466" has been submitted.
$ llsubmit  ip_share_css0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.467" has been submitted.
$ llsubmit  ip_share_css0.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.665" has been submitted.
$ llsubmit  ip_share_css0.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.666" has been submitted.
$ llsubmit  ip_share_css0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.468" has been submitted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.466.0             tani1      11/16 16:48 R  50  class_parall sp4n07
sp4n15.467.0             tani1      11/16 16:48 R  50  class_parall sp4n07
sp4n05.665.0             tani1      11/16 16:48 R  50  class_parall sp4n09
sp4n05.666.0             tani1      11/16 16:48 R  50  class_parall sp4n09
sp4n15.468.0             tani1      11/16 16:48 R  50  class_parall sp4n09

5 job steps in queue, 0 waiting, 0 pending, 5 running, 0 held
```

You can see that multiple job steps (more than four) can run if you use IP mode, and, in the following screen output, you can see that multiple tasks are running on each node:

```
# dsh -a ps -ef | grep btat | grep -v grep | awk '{print $1,$2,$6,$9}'
sp4n06: tani1 16:48:35 /u/tani1/SCENARIO_PARA/btat
sp4n06: tani1 16:48:41 /u/tani1/SCENARIO_PARA/btat
sp4n06: tani1 16:48:43 /u/tani1/SCENARIO_PARA/btat
sp4n06: tani1 16:48:41 /u/tani1/SCENARIO_PARA/btat
sp4n06: tani1 16:48:39 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:48:41 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:48:39 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:48:43 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:48:41 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:48:35 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:48:43 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:48:35 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:48:41 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:48:41 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:48:39 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:48:35 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:48:41 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:48:41 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:48:43 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:48:39 /u/tani1/SCENARIO_PARA/btat
#
```

7.5.4 Using ethernet with IP in shared mode

You can request a network interface when you use the IP mode. The following is the job command file that requests the en0 interface in shared mode:

```
$ cat ip_share_en0.cmd
# @ job_type=parallel
# @ environment = COPY_ALL; MP_TIMEOUT=2000;
# @ error   = btat_test.$(Host).$(Cluster).$(Process).err
# @ output  = btat_test.$(Host).$(Cluster).$(Process).out
# @ wall_clock_limit = 180,180
# @ network.mpi = en0,shared,ip
# @ node = 4
# @ tasks_per_node = 1
# @ executable = /bin/poe
# @ arguments = /u/tani1/SCENARIO_PARA/btat -d 40 -t 1 -m 1000 -v -labelio yes
# @ class = class_parallel
# @ queue
```

The following screen shows the output that resulted after we submitted the job command file. You can see that multiple jobs are running simultaneously:

```
$ llq
llq: There is currently no job status to report.
$ llsubmit ip_share_en0.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.669" has been submitted.
$ llsubmit ip_share_en0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.471" has been submitted.
$ llsubmit ip_share_en0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.472" has been submitted.
$ llsubmit ip_share_en0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.473" has been submitted.
$ llsubmit ip_share_en0.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.670" has been submitted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.669.0             tani1      11/16 16:54 R  50  class_parall sp4n09
sp4n15.471.0             tani1      11/16 16:54 R  50  class_parall sp4n09
sp4n15.472.0             tani1      11/16 16:54 R  50  class_parall sp4n06
sp4n15.473.0             tani1      11/16 16:54 R  50  class_parall sp4n06
sp4n05.670.0             tani1      11/16 16:54 ST 50  class_parall sp4n06

5 job steps in queue, 0 waiting, 1 pending, 4 running, 0 held
```

You can also see that multiple tasks are running on each node in the following screen output:

```
# dsh -a ps -ef | grep btat | grep -v grep | awk '{print $1,$2,$6,$9}'
sp4n06: tani1 16:54:37 /u/tani1/SCENARIO_PARA/btat
sp4n06: tani1 16:54:42 /u/tani1/SCENARIO_PARA/btat
sp4n06: tani1 16:54:35 /u/tani1/SCENARIO_PARA/btat
sp4n06: tani1 16:54:43 /u/tani1/SCENARIO_PARA/btat
sp4n06: tani1 16:54:45 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:54:42 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:54:35 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:54:45 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:54:43 /u/tani1/SCENARIO_PARA/btat
sp4n07: tani1 16:54:37 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:54:36 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:54:45 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:54:43 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:54:37 /u/tani1/SCENARIO_PARA/btat
sp4n08: tani1 16:54:42 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:54:42 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:54:37 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:54:45 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:54:43 /u/tani1/SCENARIO_PARA/btat
sp4n09: tani1 16:54:35 /u/tani1/SCENARIO_PARA/btat
```

7.5.5 Using ethernet with IP not-shared mode

You can specify **not_shared** usage when you request an interface with IP mode. The following screen shows the job command file with network keyword entry for not_shared usage:

```
$ cat ip_notshare_en0.cmd
# @ job_type=parallel
# @ environment = COPY_ALL; MP_TIMEOUT=2000;
# @ error   = btat_test.$(Host).$(Cluster).$(Process).err
# @ output  = btat_test.$(Host).$(Cluster).$(Process).out
# @ wall_clock_limit = 180,180
# @ network.mpi = en0,not_shared,ip
# @ node = 4
# @ tasks_per_node = 1
# @ executable = /bin/poe
# @ arguments = /u/tani1/SCENARIO_PARA/btat -d 40 -t 1 -m 1000 -v -labelio yes
# @ class = class_parallel
# @ queue
```

We submitted this job command file five times. You can see the output screen when we submitted the job command files in the following:

```
$ llsubmit ip_notshare_en0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.469" has been submitted.
$ llsubmit ip_notshare_en0.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.667" has been submitted.
$ llsubmit ip_notshare_en0.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.668" has been submitted.
$ llsubmit ip_notshare_en0.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.470" has been submitted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.469.0             tani1      11/16 16:51 R  50  class_parall sp4n09
sp4n05.667.0             tani1      11/16 16:51 I  50  class_parall
sp4n05.668.0             tani1      11/16 16:51 I  50  class_parall
sp4n15.470.0             tani1      11/16 16:51 I  50  class_parall

4 job steps in queue, 3 waiting, 0 pending, 1 running, 0 held
```

You can see that only one job is running, and other jobs are waiting in the job queue.

7.6 Interactive POE

When you want to perform a POE job interactively using userspace, you need to define default_interactive_class in the administration file. LoadLeveler uses this class name for your interactive POE job. The following modification is required in the user stanza used in this scenario:

```
tani1:          type = user
                default_class = class_second
                default_group = group_serial
                default_interactive_class = class_parallel
                maxjobs = 40
                maxqueued = 80
```

You can specify the pool_list keyword in the machine stanza. By specifying the pool_list, LoadLeveler manages the pool list and allocates the nodes for your POE job. If you define the pool_list in the machine stanza, you can perform your POE job without the hostlist file by specifying the pool with the option. To read about how to create a hostlist file, refer to the book *Parallel Environment for AIX: Operation and Use*, SC28-1979.

The following is the machine stanza we use in this scenario. Three nodes are defined as pool 1 with the **pool_list** keyword:

```
sp4n06.msc.itso.ibm.com:        type=machine
                                central_manager = alt
                                adapter_stanzas = sp4n06_en0 sp4n06_css0
                                spacct_excluse_enable = false
                                pool_list = 1

sp4n07.msc.itso.ibm.com:        type = machine
                                adapter_stanzas = sp4n07_en0 sp4n07_css0
                                spacct_excluse_enable = false
                                pool_list = 1

sp4n08.msc.itso.ibm.com:        type = machine
                                adapter_stanzas = sp4n08_en0 sp4n08_css0
                                spacct_excluse_enable = false
                                pool_list = 1
```

When you perform POE interactively, you can specify your resource requirements with environment variables or flags of the poe command. You can see these flags by entering the poe command with -h:

```
poe -h
```

The following is the screen output when we executed the POE job interactively:

```
$ export MP_EUIDEVICE=css0
$ export MP_EUILIB=us
$ poe mpi_test -rmpool 1 -procs 3 -infolevel 4 2>&1 | more
```

In this case, we specified the css0 for network adapter and user space mode for communication protocol as environment variables. We specified pool 1 with the -rmpool flag and the number of nodes with the -procs flag. The -infolevel flag determines the level of message reporting for debug purposes.

The following is the screen output that resulted when we executed the `poe` command:

```
INFO: DEBUG_LEVEL changed from 0 to 2
D1<L2>: ./host.list file did not exist
D1<L2>: mp_euilib = us
D1<L2>: node allocation strategy = 1
INFO: 0031-364  Contacting LoadLeveler to set and query information for interact
ive jo
b
D1<L2>: Job Command String:
#@ job_type = parallel
#@ environment = COPY_ALL
#@ requirements = (Pool == 1)
#@ node = 3
#@ total_tasks = 3
#@ node_usage = not_shared
#@ network.mpi = css0,not_shared,us
#@ class = class_parallel
#@ queue

INFO: 0031-380  LoadLeveler step ID is sp4n15.msc.itso.ibm.com.646.0
D1<L2>: Job key assigned by LoadLeveler = 943044330
INFO: 0031-119  Host sp4n06.msc.itso.ibm.com allocated for task 0
INFO: 0031-119  Host sp4n08.msc.itso.ibm.com allocated for task 1
INFO: 0031-119  Host sp4n07.msc.itso.ibm.com allocated for task 2
D1<L2>: Spawning /etc/pmdv2 on all nodes
D1<L2>: Socket file descriptor for task 0 (sp4n06.msc.itso.ibm.com) is 6
D1<L2>: Socket file descriptor for task 1 (sp4n08.msc.itso.ibm.com) is 7
D1<L2>: Socket file descriptor for task 2 (sp4n07.msc.itso.ibm.com) is 8
```

In the output, you can see that POE contacts the LoadLeveler and creates the job command file. The LoadLeveler receives and assigns the job ID for the job. You can check the status with the `llq` command. The following screen shows the output of the `llq` command:

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.646.0             tani1      11/19 14:45 R  50  class_parall sp4n06

1 job steps in queue, 0 waiting, 0 pending, 1 running, 0 held
```

The job is running as the class_parallel job. In the following screen, you can see that three tasks are running on three nodes:

```
# dsh -w sp4n06,sp4n07,sp4n08 ps -ef | grep mpi
sp4n06:    tani1  6382 14286  77 14:45:31      -  4:23 mpi_test
sp4n07:    tani1  3568 12954  78 14:45:31      -  4:23 mpi_test
sp4n08:    tani1 13252 15376 115 14:45:31      -  4:23 mpi_test
#
```

POE contacts the LoadLeveler to get the resource information and manages the resource and the job. Using LoadLeveler, you can manage the workload - not only the batch jobs but also the interactive POE jobs.

Chapter 8. Managing application build

This chapter discusses the distribution of workload for the compilation and linking of multiple source programs and the creation of the executable in an SP environment.

8.1 Scenario description

In an application development environment, there will be a number of program files to be compiled and linked to create the executable file. Compiling a program is an extremely CPU- and memory-intensive job. Generally, we prepare the build using the `make` command, which compiles the programs one by one in a serial fashion and then links the object modules to get the executable. This is really a time-consuming process.

At this point, we would like to show an example of how we can use the LoadLeveler to submit this compilation of the programs in parallel across all nodes in the LoadLeveler cluster. This way, the time taken to build an application can be reduced.

For the sample scenario, we are using the C sample files that come as part of the PESSL software for building the executable. These files are available in the directory /usr/lpp/pessl.rte.common/example/c. This particular sample program has the following source programs: diffusion.c, fourier.c, main.c, and scalemod.c. There is a Makefile file given to compile and create the executable using the `make` command. The `make` command compiles the four source programs serially and then links the object files to build the executable.

Here, we will first show how we can submit the make command as a LoadLeveler job; then, we will show how we can do the compilation of the four source programs in parallel in four nodes in the SP using LoadLeveler.

8.2 Tool choice

To perform a parallel build across all nodes in the cluster, we need the following environment in all the nodes in which compilation needs to be done.

- LoadLeveler installed and configured as part of the LoadLeveler cluster.
- The required version of AIX, PSSP, and POE is installed on all the nodes.
- The required compiler version is installed on all the nodes, and it has the licence to do the compilation.

- The other dependent software and libraries that are required for the compilation of this program are installed and made available.
- The source code and header files required for compilation are accessible to all the nodes in the LoadLeveler cluster.

For our sample scenario in the lab, we used the following environment:

- LoadLeveler V2.1, AIX V4.3.3, PSSP V3.1.1, and POE V2.4 are installed on ten nodes in the RS/6000 SP.
- We defined all ten nodes as the executing nodes in the LoadLeveler cluster.
- We defined two nodes as the scheduling nodes in the LoadLeveler cluster.
- We installed PESSL on all the nodes. The sample program we are using here requires the pessl libraries.

8.3 Configuring the build environment

We copied the sample files from the example directory, /usr/lpp/pessl.rte.common.example/c, to tani1's home directory. The home directory of the user tani1 is /u/tani1, and it is exported from the CWS to all the nodes. This sample PESSL program has four C source programs to be compiled.

8.3.1 LoadLeveler administration file

We created a class called `build`, and the user `tani1` is being permitted to use this class. This is configured in the LoadL_admin file, and it will look like the following:

```
# CLASS STANZAS:
build:          type = class
                priority = 100
                include_users=bala tani1
                total_tasks = 100
                max_node = 16
```

8.3.2 LoadLeveler local configuration file

We installed the C compiler and the PESSL software in the nodes sp4n01, sp4n05, sp4n06, sp4n07, and sp4n08. The sample make program has four source programs that we want to get that compiled in any of these nodes. We added the `build` class for the class keyword in the local configuration file in each of these nodes. The contents of LoadL_config.local will look like the following:

```
Class = { "build" "small" "class_first" "class_first" "class_second" "short_limi
t" "long_limit" }
```

8.3.3 Creating an executable

Creating an executable file from the source program can be defined as a two-step process. The first step is to compile all the source programs, and the second step is to link them to create the executable. The second step of linking the object modules cannot be done unless the first step is completed without errors.

We also need to implement the same thing in the LoadLeveler as two steps for creating the executable. In LoadLeveler terms, we can say that step0 does the compilation, and step1 does the linking of the object modules. We also need to tell the LoadLeveler that step1 is dependent on the successful completion of step0.

8.3.4 Submitting a compilation job to the LoadLeveler

In the normal environment, to compile a C program, we use the `cc <options> <filename>` command to get the object code. Now, we will see how we can submit this as a LoadLeveler job. As we mentioned earlier, LoadLeveler cannot directly accept the executable to submit a job, and it needs to be given as a job command file. We wrote a script that is used to create the job command file. The contents of the `llxlc` script look like the following:

```
$ cat llxlc
echo "#!/bin/ksh" > compile.cmd
echo "# @ initialdir = /u/tani1/SCENARIO1/make" >> compile.cmd
echo "# @ error   = $4.err ">> compile.cmd
echo "# @ output  = $4.out "    >> compile.cmd
echo "# @ executable = /usr/bin/cc" >> compile.cmd
echo "# @ arguments = "  $*  >> compile.cmd
echo "# @ class = build" >> compile.cmd
echo "# @ queue"  >> compile.cmd
llsubmit compile.cmd
```

The `llxlc` script creates a job command file: `compile.cmd`. This job command file will have the minimum required keywords that are required to submit a job.

- The `initialdir` keyword names your working directory for this job.

- The *error* and *output* keywords define the name of the output and error files to be created. Here we are creating files with the same name of the source program file with the .err and .out extensions respectively.
- The *class* field tells you the class of node on which this job must run.
- The *executable* is given as the `cc` command.
- The *arguments* received along with llxlc are passed as the arguments to cc.
- The last line in the script submits the job compile.cmd to the LoadLeveler using the `llsubmit` command.

Let us take one example of a C program compiled using the normal `cc` and the same when we use the `llxlc` script to submit it to the LoadLeveler.

Compile using cc

```
$ cc -c -g -O3 fourier.c
```

When you give the cc from the shell prompt, it does the compilation on the same node.

Compile using llxlc

Let us compile the same program using llxlc here and see the difference.

```
$ llxlc -c -g -O3 fourier.c

llsubmit: The job "sp4n15.msc.itso.ibm.com.647" has been submitted.
```

When you use the llxlc script with the same arguments as given to the `cc` command, the llxlc script creates the job command file `compile.cmd` and submits the job to the LoadLeveler.

The contents of the `compile.cmd` file look like the following:

```
$ cat compile.cmd
#!/bin/ksh
# @ initialdir = /u/tani1/SCENARIO1/make
# @ error   = fourier.c.err
# @ output  = fourier.c.out
# @ executable = /usr/bin/cc
# @ arguments =  -c -g -O3 fourier.c
# @ class = build
# @ queue
```

Now, the LoadLeveler will schedule the job in any of the nodes with the class build based on the availability of the node that is free to run jobs of this class build.

The steps that we have seen so far are for creating one job command file for compiling one source code. Normally, there would be multiple source programs compiled in an application. In a normal case, we create the Makefile, which defines all the source files to be compiled and files to be linked to create the final executable. After editing the Makefile, we use the `make` command, which takes the input from the Makefile to process the targets defined in the Makefile.

Now, let us see how we can use the `make` command and the Makefile in our scripts to achieve the same results by submitting them as multiple jobs to the LoadLeveler. In order to use this default Makefile for submitting jobs to LoadLeveler, we need to make some minor modifications to the default Makefile. The changes required for the Makefile are shown in the following screen:

```
#*******************************************************************
#*    LICENSED MATERIALS - PROPERTY OF IBM                         *
#*    "RESTRICTED MATERIALS OF IBM"                                *
#*                                                                 *
#*    5765-422                                                     *
#*    (C) COPYRIGHT IBM CORP. 1995. ALL RIGHTS RESERVED.           *
#*                                                                 *
#*    U.S. GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION   *
#*    OR DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH   *
#*    IBM CORP.                                                    *
#*******************************************************************

#CC = /usr/lpp/xlC/bin/cc
#CC = /usr/vac/bin/cc (Comment this line for Parallel Build)
# Add the next line for Parallel Build
CC = llxlc

CFLAGS =  -O3
OBJS = main.o scalemod.o diffusion.o fourier.o

#.c.o: (Comment this line for parallel Build)
# Add the next two lines for Parallel Build
compile:$(OBJS)
$(OBJS):
        $(CC)  -c -g $(CFLAGS) $<

diffus:
mpcc -o diffus $(OBJS) -lpessl -lblacs -lm -lessl

clean:
        rm -f *.o diffus core *.lst

main.o: main.c diffus.h  parameter.h
diffusion.o: diffusion.c diffus.h parameter.h
fourier.o: fourier.c diffus.h parameter.h
scalemod.o: scalemod.c diffus.h parameter.h
```

The first change is to replace the line `CC = /usr/vac/bin/cc` with `CC = llxlc`. The `llxlc` is the script, which we created for creating the job command file. The second change is to have a target defined to only perform the compilation. This is highlighted and shown in the Makefile here.

Now, when we execute the `make` command with the target as compile, it invokes the `llxlc` script along with the arguments instead of `cc`. The `make` command will invoke the `llxlc` script for each compilation of source program as defined in the Makefile. The `llxlc` script, in turn, creates a job command file for each compilation and submits it as a job to the LoadLeveler; so, if you had defined four source programs to be compiled in the Makefile, it means it will create four jobs to be submitted to the LoadLeveler.

Let us execute the `make` command for our example and see how it submits the job to the LoadLeveler:

```
$ make compile
        llxlc  -c -g -O3 main.c
llsubmit: The job "sp4n05.msc.itso.ibm.com.841" has been submitted.
        llxlc  -c -g -O3 scalemod.c
llsubmit: The job "sp4n15.msc.itso.ibm.com.658" has been submitted.
        llxlc  -c -g -O3 diffusion.c
llsubmit: The job "sp4n05.msc.itso.ibm.com.842" has been submitted.
        llxlc  -c -g -O3 fourier.c
llsubmit: The job "sp4n15.msc.itso.ibm.com.659" has been submitted.
Target "compile" is up to date.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.841.0             tani1      11/22 09:39 R  50  build        sp4n08
sp4n15.658.0             tani1      11/22 09:39 R  50  build        sp4n07
sp4n05.842.0             tani1      11/22 09:39 R  50  build        sp4n06
sp4n15.659.0             tani1      11/22 09:39 R  50  build        sp4n05

4 job steps in queue, 0 waiting, 0 pending, 4 running, 0 held
$
```

From the screen output, you can see that the four programs, main.c, scalemod.c, diffusion.c, and fourier.c, are submitted as four jobs to the LoadLeveler using the `make` command with the target `compile`. The job status when seen using the `llq` command shows that the four jobs are running in the four nodes, `sp4n05`, `sp4n06`, `sp4n07`, and `sp4n08`. Now, let us see the processes that are running in these four nodes. This can be seen using the `dsh` command from the CWS, and the output looks like the following:

```
# dsh -w sp4n05,sp4n06,sp4n07,sp4n08
dsh> ps -eaf | grep tani1
sp4n05:    tani1  5490 13926   0 09:39:21      -  0:00 /u/loadl/execute/sp4n15.m
sc.itso.ibm.com.659.0/cc -c -g -O3 fourier.c
sp4n05:    tani1  7458  5490   0 09:39:21      -  0:00 xlcentry -D_AIX -D_AIX32
-D_AIX41 -D_AIX43 -D_IBMR2 -D_POWER -qlanglvl=extended -qnoro -qnoroconst -g -O3
 -ofourier.o fourier.c /tmp/xlcqwfxia /tmp/xlcqwfxib /dev/null fourier.lst fouri
er /tmp/xlcqwfxic
sp4n05:    tani1 13926  6892   0 09:39:20      -  0:00 LoadL_starter -p 4 -s sp4
n15.msc.itso.ibm.com.659.0
sp4n06:    tani1  6328  4740   0 09:39:19      -  0:00 LoadL_starter -p 4 -s sp4
n05.msc.itso.ibm.com.842.0
sp4n06:    tani1  8422 14276   0 09:39:21      -  0:00 xlcentry -D_AIX -D_AIX32
-D_AIX41 -D_AIX43 -D_IBMR2 -D_POWER -qlanglvl=extended -qnoro -qnoroconst -g -O3
 -odiffusion.o diffusion.c /tmp/xlcY9nYqa /tmp/xlcY9nYqb /dev/null diffusion.lst
 diffusion /tmp/xlcY9nYqc
sp4n06:    tani1 14276  6328   0 09:39:20      -  0:00 /u/loadl/execute/sp4n05.m
sc.itso.ibm.com.842.0/cc -c -g -O3 diffusion.c
sp4n07:    tani1 11880  5494   0 09:39:19      -  0:00 LoadL_starter -p 4 -s sp4
n15.msc.itso.ibm.com.658.0
sp4n07:    tani1 12834 11880   0 09:39:20      -  0:00 /u/loadl/execute/sp4n15.m
sc.itso.ibm.com.658.0/cc -c -g -O3 scalemod.c
sp4n07:    tani1 13104 12834   0 09:39:20      -  0:00 xlcentry -D_AIX -D_AIX32
-D_AIX41 -D_AIX43 -D_IBMR2 -D_POWER -qlanglvl=extended -qnoro -qnoroconst -g -O3
 -oscalemod.o scalemod.c /tmp/xlcXnm9ia /tmp/xlcXnm9ib /dev/null scalemod.lst sc
alemod /tmp/xlcXnm9ic
sp4n08:    tani1  3406 15454   0 09:39:19      -  0:00 /u/loadl/execute/sp4n05.m
sc.itso.ibm.com.841.0/cc -c -g -O3 main.c
sp4n08:    tani1 12900  3406   0 09:39:20      -  0:00 xlcentry -D_AIX -D_AIX32
-D_AIX41 -D_AIX43 -D_IBMR2 -D_POWER -qlanglvl=extended -qnoro -qnoroconst -g -O3
 -omain.o main.c /tmp/xlcQ7duUa /tmp/xlcQ7duUb /dev/null main.lst main /tmp/xlcQ
7duUc
sp4n08:    tani1 15454  5720   0 09:39:18      -  0:00 LoadL_starter -p 4 -s sp4
n05.msc.itso.ibm.com.841.0
```

The LoadLeveler will get the job executed on different nodes in the LoadLeveler cluster based on the class and workload of the nodes. In this way, we can do the parallel compilation across multiple nodes and balance the workload across all the nodes.

8.3.5 Submitting the Build Job to the LoadLeveler

Once all the source programs are compiled, we can perform the linking of objects to create the executable. This has to be executed only on one node. This can also be submitted as one serial job to the LoadLeveler so that the LoadLeveler can schedule this job to a node that is free. The normal method of linking is with the `make` command as follows:

```
$ make diffus
       mpcc -o diffus main.o scalemod.o diffusion.o fourier.o -lpessl -lblacs -
lm -lessl
```

When you issue the `make` command from the shell prompt, it executes in the same node. Now, let us do the same thing by submitting it as a job to the LoadLeveler. Edit the job command file, `diffus.cmd`, to submit the linking of all object modules using the `make` command. The contents of the job command file will look like the following:

```
$ cat diffus.cmd
#!/bin/ksh
# @ initialdir = /u/tani1/SCENARIO1/make
# @ error   = diffus.err
# @ output  = diffus.out
# @ executable = /usr/bin/make
# @ arguments = diffus
# @ class = build
# @ queue
```

Submit the `diffus.cmd` job to the LoadLeveler using the `llsubmit` command, and check the job status using the `llq` command to find the node on which the job is executed.

```
$ llsubmit diffus.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.849" has been submitted.

$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.849.0             tani1      11/22 12:14 R  50  build        sp4n08

1 job steps in queue, 0 waiting, 0 pending, 1 running, 0 held
```

From the screen output shown here, you can see the job is running on the node `sp4n08`.

So far, we have shown how to submit the compilation and linking of object modules as two different steps to get the final executable code. But, normally, we do the same in one step by simply issuing the `make` command once. This two step process can be done using one `llsubmit` command.

For this, we should create a job command file with two steps: The first step in the job command file should perform the compilation of all the source programs, and the second step should do the linking of all the object modules to make the executable. When you do it this way, all the jobs will be submitted to the LoadLeveler immediately. But, we cannot perform the linking unless the compilation is finished successfully. In other words, the second step of linking the object modules should only start after the first step has finished successfully.

This can be implemented in the LoadLeveler job command file using the `hold` keyword. The first step in the job command file will be step0, and the second step will be step1. We will define the job command file for step1 as being held by the user. This will keep the job on hold by the user until he or she releases the hold.

The user can monitor whether all his or her compilation jobs are successful and whether the object files are created. Once the user confirms that all the compilations are successful, he or she can release the hold for this job using the `llhold -r` command. As soon as the hold on the job is released, the LoadLeveler will submit the job for execution on any node that is free.

The contents of the job command file `build.cmd` will look like the following:

```
$ cat build.cmd
#!/bin/ksh
# @ initialdir = /u/tani1/SCENARIO1/make
# @ error   = build.err
# @ output  = build.out
# @ step_name = step0
# @ executable = /usr/bin/make
# @ arguments = compile
# @ class = build
# @ queue
# @ step_name = step1
# @ hold = user
# @ executable = /usr/bin/make
# @ arguments = diffus
# @ class = build
# @ queue
```

Now, let us submit this job to the LoadLeveler and see how it works.

```
$ llsubmit build.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.824" with 2 job steps has been submit
ted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.641.0             tani1      11/19 14:33 R  50  build        sp4n05
sp4n05.825.0             tani1      11/19 14:33 R  50  build        sp4n06
sp4n15.642.0             tani1      11/19 14:33 R  50  build        sp4n08
sp4n05.826.0             tani1      11/19 14:33 R  50  build        sp4n07
sp4n05.824.1             tani1      11/19 14:33 H  50  build
sp4n05.824.0             tani1      11/19 14:33 C  50  build

5 job steps in queue, 0 waiting, 0 pending, 4 running, 1 held
```

From the screen output, you can see that the four source programs are running while one job is kept on hold. The job that is kept on hold is the linking job. After some time, the user can use the `llq` command to check whether all four jobs have completed.

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.824.1             tani1      11/19 14:33 H  50  build
sp4n05.824.0             tani1      11/19 14:33 C  50  build

1 job steps in queue, 0 waiting, 0 pending, 0 running, 1 held
```

The user can also check that all his or her object modules were created successfully. Now the user can release the job that is on hold using the `llhold -r` command.

```
$ llhold -r sp4n05.824.1
llhold: Hold command has been sent to the central manager.

$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.824.1             tani1      11/19 14:33 R  50  build        sp4n01
sp4n05.824.0             tani1      11/19 14:33 C  50  build

1 job steps in queue, 0 waiting, 0 pending, 1 running, 0 held
```

Immediately after executing the release command, the job was submitted to the node, sp4n01, and the executable, diffus, was created.

This way, one can speed up the build process in an SP environment using the LoadLeveler.

Chapter 9. Managing workload using checkpointing

In this chapter, we will discuss how to perform checkpointing of serial and parallel jobs. The types of checkpointing that can be enabled and the associated keywords are discussed in detail in Section ”3.6” Checkpointing” on page 91.

9.1 Scenario description

Checkpoint can either be initiated by the system or by the user. System-initiated checkpointing is only available for serial jobs. User-initiated checkpointing is available for both serial and parallel jobs. For user-initiated checkpointing, you should have to insert the checkpoint subroutines in the source program at required intervals. You can also enable both the user- and system-initiated checkpoints for serial jobs.The system-initiated one will automatically checkpoint your program at preset intervals.

Here, we will take a sample C program and demonstrate how we can use the system-initiated and user-initiated checkpointing.

9.2 Tools choice

To initiate user- and system-initiated checkpointing, the following criteria must be met in the SP nodes.

- LoadLeveler installed and configured as part of the LoadLeveler cluster
- The required version of AIX, PSSP, and POE installed on all the nodes
- The required version of C or Fortran compiler installed in any one of the nodes for compiling

9.3 Testing environment

The sample C program is available under the home directory of user bala. The home directory of the user bala is /u/bala, and it is NFS exported from the CWS to all the nodes. File collection is enabled, and user management is done from the CWS.

9.4 System-initiated serial job checkpointing

Here, we will illustrate, by means of an example, how system-initiated checkpointing works.

9.4.1 Configuring the LoadL_config file for checkpointing

In the case of system-initiated checkpointing, the system will checkpoint at periodic intervals as defined by the administrator in the LoadL_config file. The keywords are MIN_CKPT_INTERVAL and MAX_CKPT_INTERVAL. For our test environment, we defined the following values in the LoadL_config file.

```
MIN_CKPT_INTERVAL    = 60

MAX_CKPT_INTERVAL    = 300
```

9.4.2 Writing a sample C program for testing Checkpoint

We wrote a sample C program that counts to five and goes to sleep for five seconds. It resumes, counts to five again, and goes to sleep. This program is in a continuous loop.

The sample C program for testing our system-initiated checkpointing looks like this:

```
$ cat no_ckpt.c
#include <stdio.h>

main()
{
        int i, rc = 99;

        for ( i = 0 ; ; i++ ) {
                fprintf( stdout, "i = %d\n", i );
                fflush( stdout );

                if ( i % 5 == 0 ) {
                        fprintf( stdout, "Counting\n" );
                        fprintf( stdout, "Sleeping 5 seconds...\n" );
                        fflush( stdout );
                        sleep( 5 );
                        fprintf( stdout, "Resume.\n" );
                        fflush( stdout );
                }
        }
}
```

For system-initiated checkpoints, there are no checkpoint related subroutines required in the source program. But, the serial jobs have to be linked with the libchkrst.a and chkrst_wrap.o LoadLeveler libraries. To ensure that checkpointing is working properly, we suggest that you always use the scripts provided by LoadLeveler for compiling and linking the object modules. These scripts are available in the /usr/lpp/LoadL/full/bin directory. For the C source

program, no_ckpt.c, we used the `crxlc` script to create the `no_ckpt` executable. The command to create the executable looks like this:

```
$ /usr/lpp/LoadL/full/bin/crxlc no_ckpt no_ckpt.c
```

9.4.3 Creating a job command file with checkpoint enabled

The next step is to create a job command file to submit the executable `no_ckpt` as a LoadLeveler job. We will enable system-initiated checkpointing in the job command file. The job command file sys-ckpt.cmd for system-initiated checkpointing with the executable as `no_ckpt` will look like this:

```
#
#     Testing system initiated checkpointing using a c program

# @ EXE         = no_ckpt
# @ TC          = sysckpt
#
#     Execute step
#
# @ initialdir = /u/bala/check
# @ step_name    = $(TC)
# @ error   = $(EXE).$(Cluster).$(Process).err
# @ output  = $(EXE).$(Cluster).$(Process).out
# @ class = small
# @ notification = always
# @ executable   = $(EXE)
# @ environment = COPY_ALL
# @ requirements = (Machine == {"sp4n08.msc.itso.ibm.com" "sp4n09.msc.itso.ibm.c
om"})
# @ environment = CHKPT_STATE=enable; CHKPT_FILE=$(EXE).ckpt; CHKPT_DIR=/u/bala/
check
# @ checkpoint   = system_initiated
# @ restart      = yes
# @ queue
```

There are five keywords defined in the job command file that are related to checkpoint. They are highlighted in the screen output.

- The `checkpoint` keyword indicates that it has to perform system-initiated checkpointing.
- The `restart = yes` keyword indicates that the job has to be restarted after a failure. This keyword has to be set to `yes` for system-initiated checkpointing. If you set it to `no`, the LoadLeveler, while submitting the job, will force it to `yes`.
- The Environment variable, `CHKPT_STATE = enable`, enables checkpointing.

- The name of the checkpoint file that has to be created by the system is defined by the `CHKPT_FILE` keyword. In our case, we defined it to be executable.ckpt.
- The directory where this checkpoint files to be created is defined by the keyword `CHKPT_DIR`.

9.4.4 Testing the system-initiated serial checkpoint

Having created the executable and the job command file, we will now test the checkpoint program. The user bala has logged in to one of the nodes to perform the test. The following are the steps to test the system-initiated checkpointing.

1. Submit the job to the LoadLeveler using the `llsubmit` command:

```
$ llsubmit sys-ckpt.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.863" has been submitted.
```

2. Check the status of the job to find out on which node the job is scheduled and running using the `llq` command.

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.863.0             bala       11/23 13:55 R  50  small        sp4n05

1 job steps in queue, 0 waiting, 0 pending, 1 running, 0 held
```

   From the output, you can see that the job ID is `sp4n05.863.0`, and it is running on the node `sp4n05`.

3. Check the CHKPT_DIR for the checkpoint files that have been created by the system. In the job command file, we defined that the CHKPT_DIR is /u/bala/check, and the files are to be created with the extension `.ckpt`.

```
$ ls -l *.ckpt*
-rw-r-----   1 bala     staff     217029 Nov 23 14:35 no_ckpt.ckpt
-rw-r-----   2 bala     staff     217029 Nov 23 14:30 no_ckpt.ckptjDnyUa
-rw-r-----   2 bala     staff     217029 Nov 23 14:30 no_ckpt.ckptjDnyUb
```

   From the output, you can see that three checkpoint files are created at this time. These checkpoint files will be updated at periodic intervals as defined in the configuration file.

4. Check the output file to see the current status of the sample program. The `no_ckpt.out` output file looks like this:

```
$ tail no_ckpt.863.0.out
Counting
Sleeping 5 seconds...
Resume.
i = 131
i = 132
i = 133
i = 134
i = 135
Counting
Sleeping 5 seconds...
```

   From the output, we can see that the program has counted up to 135 and then gone to sleep for five seconds.

5. Now, we will force the node to fail. In the labs, we forced the node sp4n05 to fail by giving the `reboot -q` command.

6. Check the llq and llstatus outputs. It will show as if the job is still running. There is a keyword, `MACHINE_UPDATE_INTERVAL`, defined in the LoadL_config file. This keyword defines the timeout period before which the nodes must report to the central manager. If there is no response within this period, the central manager will mark this machine as down. The default value for this keyword is 300 seconds.

7. After 300 seconds, if you see the llstatus for the node, you can see that it would have been marked as *down* by the central manager. Verify the output file to see to what count the job has completed. The screen here shows the job and machine status after 300 seconds. The tail of the output file is also shown to indicate the end of the job when node sp4n05 failed.

```
$ llstatus | grep sp4n05
sp4n05.msc.itso.ibm.com   Down      1   1 Down     1 0.01  9999 R6000     AIX43

$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.863.0             bala       11/23 13:55 R  50  small        sp4n05

1 job steps in queue, 0 waiting, 0 pending, 1 running, 0 held
$ tail *.out
Counting
Sleeping 5 seconds...
Resume.
i = 866
i = 867
i = 868
i = 869
i = 870
Counting
Sleeping 5 seconds...
```

From the screen output you can see that the node sp4n05 has been set as down and the job has finished counting upto 870 before the node sp4n05 failed.

8. Now, let us assume that node sp4n05 is not going to come up for some time and we want to run this job in a different node. For this, we will first issue the `llcancel` command to cancel this job on node sp4n05. When you issue this command, the job will be put into a Remove Pending (RP) state.

```
$ llcancel sp4n05.863.0
llcancel: Cancel command has been sent to the central manager.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.863.0             bala       11/23 13:55 RP 50  small

0 job steps in queue, 0 waiting, 0 pending, 0 running, 0 held
```

9. To resubmit the job from the checkpoint, we need to modify the keyword, CHKPT_STATE = restart, instead of the keyword, enable, in the job command file.
10. After modifying the job command file, resubmit the job to LoadLeveler using the `llsubmit` command. Check the job status to determine on which node the job is now scheduled.

```
$ llsubmit sys-ckpt.cmd
llsubmit: The job "sp4n15.msc.itso.ibm.com.680" has been submitted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n15.680.0             bala       11/23 14:23 R  50  small        sp4n07
sp4n05.863.0             bala       11/23 13:55 RP 50  small

1 job steps in queue, 0 waiting, 0 pending, 1 running, 0 held
```

From the screen output, you can see that the job has been submitted in node `sp4n07` and the job ID is `sp4n15.680.0`

11. Now, we need to check whether the job has started from the saved state in the checkpoint file or whether it has started from the beginning. This can be checked by comparing the end of the output file, no_ckpt.863.0.out, and the start of the file no_ckpt.680.0.out. Here, we will show the tail output of no_ckpt.863.0.out and the head output of the file no_ckpt.680.0.out.

```
$ tail *863*.out
Counting
Sleeping 5 seconds...
Resume.
i = 866
i = 867
i = 868
i = 869
i = 870
Counting
Sleeping 5 seconds.

$ head *680*.out
[YOU HAVE NEW MAIL]
i = 781
i = 782
i = 783
i = 784
i = 785
Counting
```

From the screen output, you can see that the last count in the first submitted job is 870, and the start count in the job that was resubmitted is 781. The reason is that the checkpoint is being done at periodic intervals.

When resubmitted, the job will only start from the previous checkpoint state. This indicates that the system has checkpointed after counting to 780, and so, when we resubmitted, the job has started at 781.

9.5 User-initiated serial job checkpointing

In the case of user-initiated checkpointing, the system will not checkpoint at predefined intervals as defined in the configuration file. Instead, users have to implement, in their source program, where they need to checkpoint. For this, users have to use the `ckpt` subroutine in their source programs in the places where they want to save the state of the program.

The procedure to resubmit the job using the checkpoint file is the same as we perform for system-initiated checkpoint. Here, we do not intend to repeat the same. As far as the procedure is concerned, there are two differences when compared to system-initiated checkpoint; so, here, we will discuss only those two points. They are:

1. The user has to code his or her program to call the `ckpt` subroutine as and when he or she requires to save the state. Here, we have written a small C program that uses the `ckpt` subroutine:

```
#include <stdio.h>

main()
{
        int i, rc = 99;

        for ( i = 0 ; ; i++ ) {
                fprintf( stdout, "i = %d\n", i );
                fflush( stdout );

                if ( i % 5 == 0 ) {
                        fprintf( stdout, "Start checkpoint...\n" );
                        fflush( stdout );
                        ckpt();
                        fprintf( stdout, "Completed ckpt ....\n" );
                        fprintf( stdout, "Sleeping 3 seconds...\n" );
                        fflush( stdout );
                        sleep( 3 );
                        fprintf( stdout, "Resume.\n" );
                        fflush( stdout );
                }
        }
}
$
```

2. In the job command file, the checkpoint has to be set for `user_initiated`. The job command file for the `user_initiated` checkpoint looks like the following:

```
#     Testing user initiated checkpointing using a c program

# @ EXE         = ckpt_test
# @ TC          = userckpt
#
#     Execute step
#
# @ step_name    = $(TC)
# @ error   = $(EXE).$(Cluster).$(Process).err
# @ output  = $(EXE).$(Cluster).$(Process).out
# @ class = small
# @ notification = always
# @ executable   = $(EXE)
# @ environment = COPY_ALL
# @ environment = CHKPT_STATE=restart; CHKPT_FILE=$(EXE).ckpt; CHKPT_DIR=/u/bala
/check
# @ initialdir = /u/bala/check
# @ checkpoint   = user_initiated
# @ restart      = yes
# @ queue
$
```

9.6 System- and user-initiated checkpointing

LoadLeveler supports both user- and system-initiated checkpoints. In order to enable both, you need to specify the checkpoint keyword to `system_initiated` in your job command file and also have your program written using the appropriate `ckpt` subroutine call. This way, both system and user will initiate to save the state of the job as defined in the configuration file and whenever the `ckpt` subroutine is called from the program. The checkpointing of parallel jobs can be done by the user-initiated method. For more information on how to use it in parallel programs, refer to the Parallel Environment Version 2.4 documentation.

Chapter 10. Workload Management using LoadLeveler and WLM

The IBM LoadLeveler product can be used to schedule the user jobs in an SP environment. LoadLeveler selects the nodes based on the job description defined by the user. System resources, such as CPU and memory, can be allocated to the LoadLeveler jobs using the new AIX feature WLM. We can use LoadLeveler to schedule the jobs and WLM to allocate the resources to a job. This chapter discusses the distribution of the workload using LoadLeveler and WLM in an SP environment.

10.1 Scenario description

Let us consider a scenario in which we have an SP being used to execute parallel and serial jobs in both interactive and batch mode. The jobs are being submitted by the LoadLeveler. There will be both the serial and parallel jobs running in the nodes. The resources used by the parallel job in each node may vary as per the AIX scheduler based on the number of jobs running in the node. In a typical parallel environment, the parallel jobs communicate among themselves using the message passing interface. In such an environment, it is better if we can assign equal resources to each node. In this scenario, the customer would like to have a mechanism by which the parallel jobs submitted to the nodes get equal priority from the scheduler and the virtual memory manager in the node.

From the machine status, LoadLeveler will determine what physical resources are available on each node in the cluster, and it will submit the jobs accordingly in the nodes. The class keyword in the local configuration file in the node defines which class of jobs can be run in that node and how many jobs can be run at a time. Once the job has been submitted to a node for execution, the LoadLeveler has no role to play about the usage of the CPU and memory resources in that node. When we have multiple jobs running in the node, the priority at which a job will execute is decided by the AIX scheduler.

The AIX Workload Manager (WLM) provides the system administrator greater control over how the scheduler and virtual memory manager allocate resources to process. This is specific to one node.

As you can see, LoadLeveler can be used to manage the jobs across multiple nodes, whereas the WLM can be used to manage the execution of jobs within the node. With this in mind, we will see how we can use both of them to ensure that the parallel jobs get more priority than the serial jobs.

10.2 Tool choice

To perform workload management using both LoadLeveler and WLM, the following environment crieteria must be met in all the nodes:

- LoadLeveler is installed and configured in all the nodes
- Workload manager is installed and configured in all the nodes

10.3 Testing environment

There are two users: bala and tani1. The user bala is considered to be a parallel job user, whereas the user tani1 is a serial job user. We want to assign the user bala with more CPU and memory resources than the serial jobs submitted by user tani1. The configurations required for LoadLeveler and WLM to achieve this in an SP environment are discussed in the following sections in more detail.

10.4 LoadLeveler configuration

We presume here that the LoadLeveler is installed and running on all the nodes. Here, we will show the configuration defined for the users bala and tani1.

LoadLeveler configuration criteria for user bala are as follows:

- The user bala is configured as a LoadLeveler user. The user bala is allowed to submit the jobs in the job class small.
- The LoadL_config.local file in all the nodes has the small class defined for the class keyword.

The LoadLeveler configurations for user tani1 are as follows:

- The user, tani1, is configured as a LoadLeveler user. The user, tani1, is allowed to submit the jobs in the class_second job class.
- The LoadL_config.local file in all the nodes has the class_second defined for the class keyword.

10.5 WorkLoad manager configuration

In this section, we will discuss the workload manager configuration required for our scenario. For the procedure to install and configure the Workload Manager, refer to Section 5.3.3, “Installation and the configuration process” on page 116.

For the scenario described, the Workload Manager configuration rules will look like the following:

Class

We will create two classes: Parallel and serial. The parallel class will be configured as tier0 and the serial class as tier1. This indicates that the processes running in the parallel class will have more priority for the resources than the serial class. This is configured in the classes file. The contents of the classes file will look like this:

```
# cat /etc/wlm/parallel/classes
parallel:
        tier   = 0

serial:
        tier   = 1
System:

Default:
```

Limits

Here, we will define the limits for the parallel and serial classes. Let us define the maximum CPU limits for parallel classes as 75 percent and those for the serial classes as 25 percent. The contents of the limits file will look like the following:

```
# cat /etc/wlm/parallel/limits
parallel:
        memory = 1%-75%
        CPU    = 1%-75%
serial:
        memory = 1%-25%
        CPU    = 1%-25%
System:
        memory = 1%-100%
        CPU    = 1%-100%
```

Shares

We will define the number of shares for the parallel class as two and the number for the serial class as one. The contents of the shares file for this configuration will look like the following:

```
# cat /etc/wlm/parallel/shares
parallel:
        CPU    = 2
        memory = 2
serial:
        CPU    = 1
        memory = 1
```

Rules

In order for WLM to classify processes into the two new classes, we must provide a set of class assignment rules. Class assignment is done by comparing attributes of the process, such as user ID, group ID and/or the pathname of the application executed with the values in the assignment rules file. The first match determines to which class the process will be assigned.

In our test environment, we would like to have the process running with the user ID bala come under the class parallel and the processe running with the user ID tani1 to come under the class serial. The contents of the rules file for our test environment will look like the following:

```
# cat /etc/wlm/parallel/rules
* class resvd user group application
parallel        -       bala    -       -
serial  -       tani1   -       -
System   -    root   -    -
Default  -    -      -    -
```

In order for the configuration to be similar across all the nodes in the SP cluster, we configured the Workload Manager in the CWS and enabled file collections to propagate the files to all the nodes in the cluster. Refer to Section 5.3.3, “Installation and the configuration process” on page 116for the procedure to configure file collection. Propagate the files to all the nodes in the cluster from the dsh prompt using the following commands:

```
#dsh -a

dsh>/var/sysman/supper update wlmCollection
```

Next, we need to reinitialize the WLM to look into the changed configuration rules. This can be done using the following commands:

```
#dsh -a

dsh> wlmcntrl -u
```

10.6 Serial and parallel jobs for testing

For testing the scenario, we have created one serial job and one parallel job. The serial job will be submitted by the user tani1, and the parallel job will be submitted by the user bala. Since we wanted to capture the screen and show the process running on different nodes, we restricted ourselves to two nodes. This was done by adding the keyword requirements in the job command file.

The serial job is an executable file called serial submitted using llsubmit by the user tani1. The job command file for the serial job to be submitted by user tani1 looks like this:

```
# cat serial.cmd
#!/bin/ksh
# @ job_type = serial
# @ executable = serial
# @ error   = tani1.$(Host).$(Cluster).$(Process).err
# @ output  = tani1.$(Host).$(Cluster).$(Process).out
# @ initialdir = /u/tani1
# @ notify_user = tani1
# @ notification = always
# @ requirements =(Arch == "R6000") && (OpSys == "AIX43")
# @ requirements = (Machine == {"sp4n05.msc.itso.ibm.com" "sp4n06.msc.itso.ibm.c
om"})
# @ class = class_second
# @ queue
```

The parallel job is an executable btat_test, which has to be passed as an argument to the executable /bin/poe. This parallel job will run on the two nodes sp4n05 and sp4n06. There will be four tasks running on each node. The job command file for the parallel job to be submitted by user bala looks like the following:

```
# cat parallel.cmd
# @ job_type=parallel
# @ notification = error
# @ account_no = bala
# @ environment = COPY_ALL; MP_TIMEOUT=2000;
# @ requirements = (Machine == {"sp4n05.msc.itso.ibm.com" "sp4n06.msc.itso.ibm.c
om"})
# @ error   = btat_test.$(Host).$(Cluster).$(Process).err
# @ output  = btat_test.$(Host).$(Cluster).$(Process).out
# @ wall_clock_limit = 6000,5000
# @ network.mpi = css0,shared,us
# @ node = 2
# @ tasks_per_node = 4
# @ executable = /bin/poe
# @ arguments = /u/bala/btat  -d 400 -m 1000000  -v -ilevel 6 -labelio yes
# @ class = small
# @ queue
```

10.7 Testing the scenario

To test the scenario, we will first submit the serial job from the user, tani1, and allow the CPU and memory resources allocated to this serial class job. In order to have better CPU load on the system, we will submit the same job four times so that each node runs two serial jobs. The screen output here shows the submission of the serial job by the user tani1.

```
$ llsubmit serial.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.935" has been submitted.
$ llsubmit serial.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.936" has been submitted.
$ llsubmit serial.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.937" has been submitted.
$ llsubmit serial.cmd
lllsubmit: The job "sp4n05.msc.itso.ibm.com.938" has been submitted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.935.0             tani1      11/30 13:52 R  50  class_second sp4n05
sp4n05.936.0             tani1      11/30 13:53 R  50  class_second sp4n05
sp4n05.937.0             tani1      11/30 13:53 R  50  class_second sp4n06
sp4n05.938.0             tani1      11/30 13:53 ST 50  class_second sp4n06

4 job steps in queue, 0 waiting, 1 pending, 3 running, 0 held
```

Now, let us see the resources taken by the user tani1 using the `wlmstat` command. In order to check the status on both nodes at the same time, we used the distributed shell feature of SP from the CWS. The output of the `wlmstat` command on nodes sp4n05 and sp4n06 looks like the following:

```
# dsh -w sp4n05,sp4n06
dsh> wlmstat
sp4n05:             Name CPU MEM
sp4n05:     Unclassified   3  36
sp4n05:           System   1   9
sp4n05:          Default   0   1
sp4n05:         parallel   0   1
sp4n05:           serial  97   1
sp4n06:             Name CPU MEM
sp4n06:     Unclassified   3  31
sp4n06:           System   1  10
sp4n06:          Default   0   0
sp4n06:         parallel   0   1
sp4n06:           serial  97   1
```

From the `wlmstat` output, we can see that there are no processes other than the serial class job that is running on both nodes, and it is consuming 97 percent of CPU resources. This 97 percent of CPU is the processes running by the user tani1. The processes that are running under this serial class can be seen using the `ps` command. The screen here shows the processes for which the resources are allocated by the AIX scheduler and virtual memory manager to the class serial.

```
dsh>  ps -ae -o pid,user,class,pcpu,vsz,wchan,args | grep serial
sp4n05: 15746    tani1               serial  44.7   820        - /u/loadl/execut
e/sp4n05.msc.itso.ibm.com.936.0/serial
sp4n05: 16884    tani1               serial  45.0   820        - /u/loadl/execut
e/sp4n05.msc.itso.ibm.com.935.0/serial
sp4n05: 18324    tani1               serial   0.0   348    EVENT -ksh
sp4n06:  4376    tani1               serial  46.4   820        - /u/loadl/execut
e/sp4n05.msc.itso.ibm.com.938.0/serial
sp4n06: 15266    tani1               serial  46.7   820        - /u/loadl/execut
e/sp4n05.msc.itso.ibm.com.937.0/serial
dsh>  ps -ae -o pid,user,class,pcpu,vsz,wchan,args | grep serial
sp4n05: 15746    tani1               serial  46.9   820        - /u/loadl/execut
e/sp4n05.msc.itso.ibm.com.936.0/serial
sp4n05: 16884    tani1               serial  47.1   820        - /u/loadl/execut
e/sp4n05.msc.itso.ibm.com.935.0/serial
sp4n05: 18324    tani1               serial   0.0   348    EVENT -ksh
sp4n06:  4376    tani1               serial  46.2   820        - /u/loadl/execut
e/sp4n05.msc.itso.ibm.com.938.0/serial
sp4n06: 15266    tani1               serial  46.3   820        - /u/loadl/execut
e/sp4n05.msc.itso.ibm.com.937.0/serial
```

Now, let us submit the parallel job from user bala and monitor how CPU and memory resources are allocated by the AIX scheduler and virtual memory manager. The following screen shows the submission of the parallel job by user bala.

```
$ llsubmit parallel.cmd
llsubmit: The job "sp4n05.msc.itso.ibm.com.939" has been submitted.
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
sp4n05.935.0             tani1      11/30 13:52 R  50  class_second sp4n05
sp4n05.936.0             tani1      11/30 13:53 R  50  class_second sp4n05
sp4n05.937.0             tani1      11/30 13:53 R  50  class_second sp4n06
sp4n05.938.0             tani1      11/30 13:53 R  50  class_second sp4n06
sp4n05.939.0             bala       11/30 13:54 R  50  small        sp4n06

5 job steps in queue, 0 waiting, 0 pending, 5 running, 0 held
```

This parallel job is a two-node job with four tasks in each node. Since, in the job command file, we defined the requirements for this job to run only in the nodes sp4n05 and sp4n06, the job is scheduled by the LoadLeveler to the nodes sp4n05 and sp4n06. Let us now check the resource allocation, which is done using the `wlmstat` command:

```
dsh> wlmstat
sp4n05:             Name CPU MEM
sp4n05:     Unclassified   3  36
sp4n05:           System   1  10
sp4n05:          Default   0   1
sp4n05:         parallel  75  17
sp4n05:           serial  19   0
sp4n06:             Name CPU MEM
sp4n06:     Unclassified   3  31
sp4n06:           System   1  10
sp4n06:          Default   0   0
sp4n06:         parallel  77  17
sp4n06:           serial  18   1
```

The screen output here shows the CPU resource allocated to the parallel class on nodes sp4n05 and sp4n06 are 75 percent and 77 percent respectively, whereas the CPU resource allocated to the serial job has come down from 97 percent to 19 percent and 18 percent respectively.

Now, let us see which processes are running under the parallel class using the `ps` command. The following screen shows the processes for which the resources are allocated by the AIX scheduler and virtual memory manager to the class parallel.

```
dsh> ps -ae -o pid,user,class,pcpu,vsz,wchan,args | grep parallel
sp4n05:  6942     bala             parallel   0.0   468        - /etc/pmdv2
sp4n05: 14404     bala             parallel   0.0   468        - /etc/pmdv2
sp4n05: 14666     bala             parallel   0.0   468        - /etc/pmdv2
sp4n05: 15580     bala             parallel  14.9  9308        * /u/bala/btat -d
 400 -m 1000000 -v
sp4n05: 16274     bala             parallel   0.0   468        - /etc/pmdv2
sp4n05: 17068     bala             parallel  14.9  9308        * /u/bala/btat -d
 400 -m 1000000 -v
sp4n05: 17308     bala             parallel  16.4  9308        * /u/bala/btat -d
 400 -m 1000000 -v
sp4n05: 17428     bala             parallel  14.9  9376        * /u/bala/btat -d
 400 -m 1000000 -v
sp4n05: 18326     bala             parallel   0.0   348    EVENT -ksh
sp4n06:  2370     bala             parallel   0.0   468        - /etc/pmdv2
sp4n06:  4746     bala             parallel  16.4  9436        * /u/bala/btat -d
 400 -m 1000000 -v
sp4n06:  7484     bala             parallel  16.4  9432        * /u/bala/btat -d
 400 -m 1000000 -v
sp4n06: 12724     bala             parallel  16.4  9436        * /u/bala/btat -d
 400 -m 1000000 -v
sp4n06: 13058     bala             parallel   0.0   976        - /u/loadl/execut
e/sp4n05.msc.itso.ibm.com.939.0/poe -d 400 -m 1000000 -v
sp4n06: 13822     bala             parallel   0.0   468        - /etc/pmdv2
sp4n06: 14472     bala             parallel  14.9  9384        * /u/bala/btat -d
 400 -m 1000000 -v
sp4n06: 15492     bala             parallel   0.0   468        - /etc/pmdv2
sp4n06: 16532     bala             parallel   0.0   468        - /etc/pmdv2
```

From the screen output, you can see that only the tasks that are running under the user bala are classified under the WLM class parallel.

Chapter 11. Managing users

This scenario will describe the use of Secureway Network Dispatcher and PSSP features to manage groups of users with different needs accessing the same service in a server consolidation environment.

11.1 Scenario description

We assume that we have to provide a computer environment that meets the following requirements:

- Two groups of users are using workstation applications executing on four hosts.
- They access the hosts using telnet.
- Each group has a different priority.
- Under normal conditions, each group of users is confined to a subset of the hosts and must be prevented from accessing the other hosts.
- The implementation must be flexible so that, when load conditions change, it is possible to easily change the allocation of user group to the available hosts or to add hosts.

11.2 Tool choice

To fulfill these requirements, we use the following hardware and software tools:

- RS/6000 SP with four nodes
- PSSP User Management to share the AIX user and group between all SP nodes
- PSSP access control (spacs_cntrl) to limit each user's access to the subset of nodes he or she is allowed to use
- Dispatcher to allocate each user request to the server able to process the request

11.3 Environment configuration

All users access the services using telnet. The Dispatcher is unable to look into the content of a telnet IP packet to identify which user is sending the request; therefore, the Dispatcher cannot rely only on the port number used by clients to make a routing decision, and an extra decision criteria needs to

be used in addition to the basic port-based algorithm. We present two ways of solving this problem:

1. Rule-based decision

   If the users are grouped by range of IP addresses, the Dispatcher can be configured to use the IP address of the incoming requests to route it to the appropriate server. This could be the case when, for example, all members of a development team are using workstations connected to the same subnet, and all members of a marketing department are connected to a subnet in another building, and they want to share the resource of an SP cluster.

2. Multiple clusters

   If rule-based decision is not possible, another possibility is to create two clusters on the same Dispatcher. Each group is assigned one cluster so that each group has the impression of having its own (virtual) server. The two clusters can then share the SP nodes, but this is transparent to the users.

Telnet servers do not accept incoming requests on several ports. For applications that are able to listen to several ports, there is another solution to this problem. Rather than defining a multiple cluster, you can define one cluster managing requests on two ports associated with the same application. Each group of users is allocated one of the two ports. We do not present this solution in detail here.

11.3.1 Hardware platform

With either a rule-based choice or multiple clusters, we use the same hardware configuration.

We reuse the configuration presented in Chapter 4, “Secureway Network Dispatcher” on page 103, with the addition of one extra SP node as server. This configuration is presented in Figure 22 on page 209.

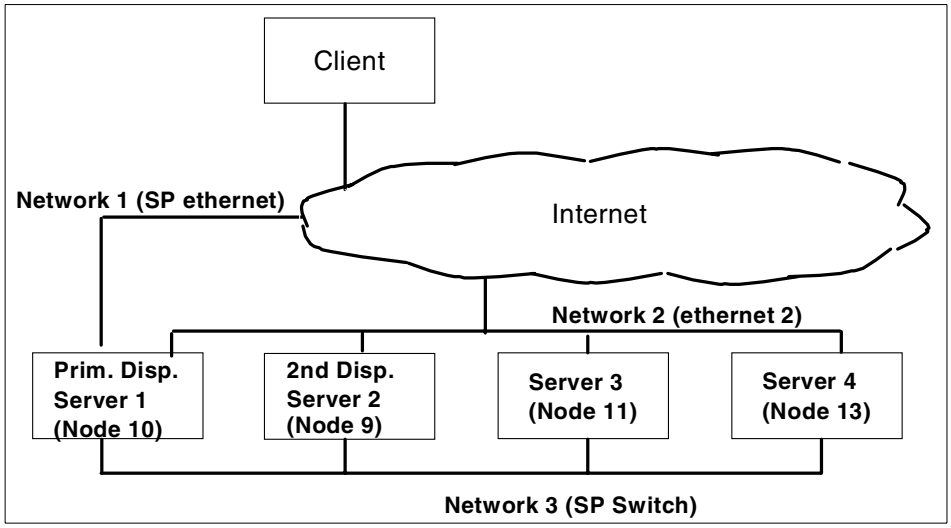


Figure 22. Secureway Network Dispatcher configuration

11.3.2 Dispatcher configuration

The Dispatcher configuration is different for the ruled-based choice and multiple cluster solutions. In this section, we only present the installation steps specific to these configurations. Refer to 4.2, “Architecture” on page 103, for general Dispatcher installation instructions.

11.3.2.1 Configuration for rule-based choice

We assume that users of one group are all working from workstations with IP addresses in the range 9.12.0.0 to 9.12.0.100 and that users in the other group use addresses in the range 9.12.0.101 to 9.12.0.255.

Dispatcher is installed on Server 1.

We created a cluster with the address 192.168.4.100. We configured the Port 23 for load balancing the telnet connections. Other ports are not configured. We created the rules to define our requirement. The following screen output describes the Dispatcher commands we used to test this scenario:

```
# ndcontrol cluster add 192.168.4.100
Cluster 192.168.4.100 has been added.
# ndcontrol port add 192.168.4.100:23
Port 23 successfully added to cluster 192.168.4.100.
# ndcontrol server add NDCLUSTER:23:sp4sw09+sp4sw10+sp4sw11+sp4sw13
Server 192.168.14.09 was added to port 23 of cluster 192.168.4.100.
Server 192.168.14.10 was added to port 23 of cluster 192.168.4.100.
Server 192.168.14.11 was added to port 23 of cluster 192.168.4.100.
Server 192.168.14.13 was added to port 23 of cluster 192.168.4.100.
ndcontrol rule add 192.168.4.100:23:RuleMktg type ip priority 42 beginrange \
9.12.0.101 endrange 9.12.0.255
ndcontrol rule useserver 192.168.4.100:23:RuleMktg 192.168.14.9+192.168.14.10

ndcontrol rule add 192.168.4.100:23:RuleDevl type ip priority 52 beginrange \
9.12.0.0 endrange 9.12.0.100
ndcontrol rule useserver 192.168.4.100:23:RuleDevl 192.168.14.11+192.168.4.13

ndcontrol rule add 192.168.4.100:23:RuleOther type true priority 62 \
beginrange 0 endrange 0
```

The first rule indicates that all client requests coming from an address in the range 9.12.0.100-255 will use server 9 and 10. The second rule routes users in the other range to use servers 11 and 13. The last rule, which is always true, is evaluated only if the request does not match any of the previous two rules. If a request comes from an address outside the ranges specified in the first two rules, this request is dropped.

The manager and the advisors are activated. The manager proportions are set to 40, 40, 20, 0 so that the input from the telnet advisor is also used by the dispatching algorithm.:

```
ndcontrol manager start manager.log 10004
ndcontrol manager proportions 40 40 20 0

ndcontrol advisor start Telnet 23 Telnet_23.log
```

11.3.2.2 Configuration for multiple clusters

We decide that users in one group use cluster 192.168.14.100 and that the other group points to cluster 192.168.14.101.

Dispatcher is installed on Server 1 (node 10).

The two clusters are created and configured (aliased) on the same adapter of node 10;

```
ndcontrol cluster add 192.168.4.100
ndcontrol cluster configure 192.168.4.100
ndcontrol cluster add 192.168.4.101
ndcontrol cluster configure 192.168.4.101
```

Port 23 is configured for load balancing of telnet connections on both clusters in the same command. Other ports are not configured.

```
ndcontrol port add 192.168.4.100+192.168.4.101:23
ndcontrol port set 192.168.4.100+192.168.4.101:23 staletimeout 32000000
```

Servers 1 and 2 are assigned to port 23 of cluster 101 while servers 3 and 4 are assigned to cluster 100:

```
ndcontrol server add 192.168.4.100:23:192.168.14.11+192.168.14.13
ndcontrol server add 192.168.4.101:23:192.168.14.9+192.168.14.10
```

The manager and the advisors are activated. The manager proportions are set to 40, 40, 20, 0 so that the input from the telnet advisor is also used by the dispatching algorithm.:

```
ndcontrol manager start manager.log 10004
ndcontrol manager proportions 40 40 20 0

ndcontrol advisor start Telnet 23 Telnet_23.log
```

The loopback device on server nodes 9, 11, and 13 must be aliased to both cluster addresses. In initial conditions, each node will belong to only one cluster, but aliasing both clusters will later allow dynamic reallocation of nodes to the other cluster if needed.

```
# ifconfig lo0 alias 192.168.4.100 netmask 255.255.255.0
# ifconfig lo0 alias 192.168.4.101 netmask 255.255.255.0
```

11.3.3 Configuration of users, groups, and applications

User and group management is implemented the same way as for ruled-based choices or multiple clusters.

Users belong to one of three groups:

- devl contains devl1 and devl2.

- mktg consists of mktg1 and mktg2.

In normal operating mode, groups are given access to each server based on the following table:

Table 10. Server access by group

	Server 1	Server 2	Server 3	Server 4
Mktg	yes	yes	no	no
Devl	no	no	yes	yes

PSSP User Management is used to manage groups and user IDs, which are, therefore, defined on all four nodes. It is then possible to dynamically reallocate users to other servers, if necessary, by new load conditions if a node is down or for any other reason.

When telnetting to the cluster address, users will always be routed to the nodes they are allowed to use by the cluster administrator. However, if a user discovers the real IP address of a node that he or she is not allowed to use, they can telnet directly to this address rather than using the cluster address, and, since we used the SP User Management, he or she would be able to log on to the node. Therefore, PSSP access control (spacs_cntrl) is configured to prevent user access to unauthorized nodes. A simple way of using the PSSP access control is to use the `spacs_cntrl` command for each user. For example, we enable user devl1 to log onto nodes 9 and 10, and we prevent this user from logging on to nodes 11 and 13:

```
# dsh -w sp4n09,sp4n10 /usr/lpp/ssp/bin/spacs_cntrl unblock devl1
# dsh -w sp4n11,sp4n13 /usr/lpp/ssp/bin/spacs_cntrl block devl1
#
```

In a real RS/6000 SP environment, it is likely that there will be too many users defined in the system to manage them one by one. In this case, we recommend that you use the file collection option or the NIS group options to manage users by group rather than individually. Chapter 5 of *Parallel System Support Program for AIX - Administration Guide* ,SA22-7348, provides detailed information about the possibilities offered by spacs_cntrl.

11.4 Results

The Dispatcher and the servers are now ready to serve client requests. Using the ndadmin GUI, we can check all settings of the Dispatcher. Figure 23 and

Figure 24 present the manager settings in a rule-based configuration and the advisor measured load in a multicluster environment.

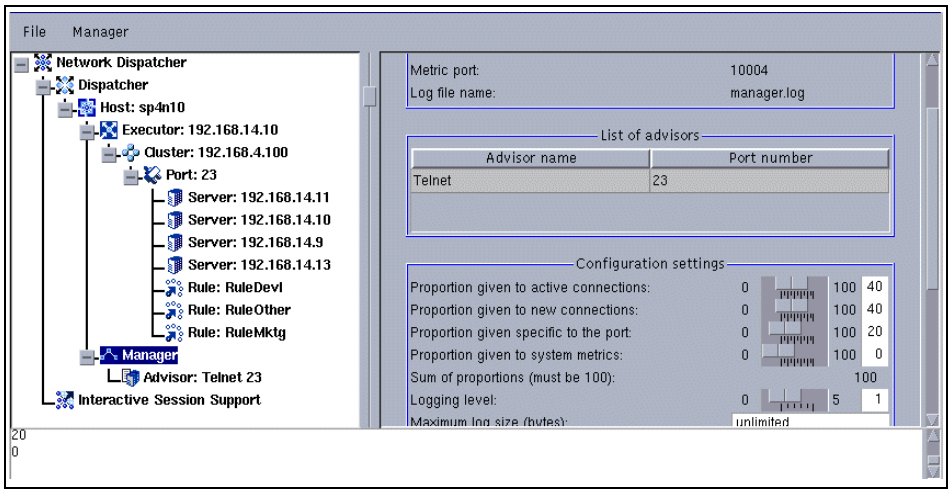


Figure 23. Manager window in rule-based configuration.

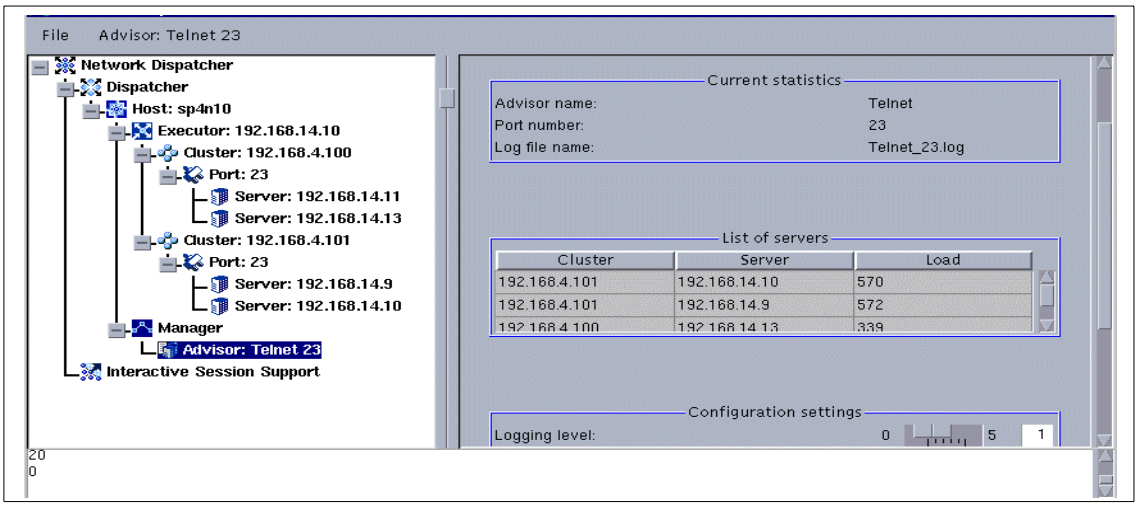


Figure 24. Advisor window in a multicluster environment

To check that load balancing is performed as planned, we select **port 23** on the Dispatcher GUI with the right mouse button, and click **Monitor**. From a client workstation, we telnet to the cluster four times. In the Monitor window,

we can verify that the new connections are allocated to the two nodes we are allowed to use (See Figure 25).

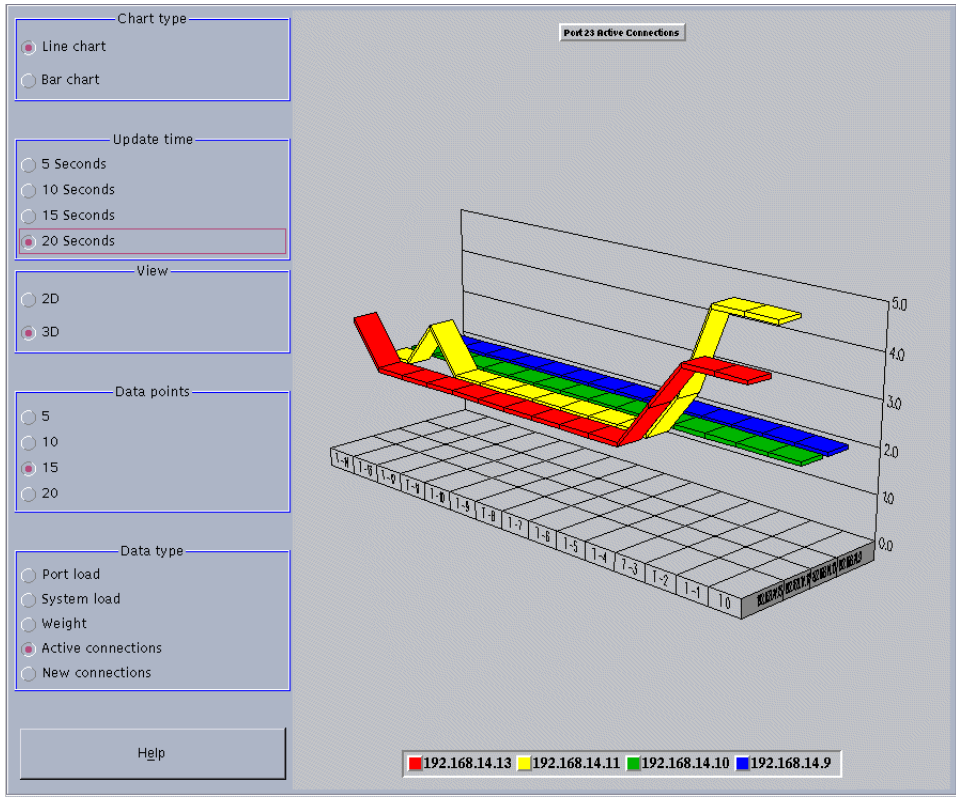


Figure 25. Monitoring connections

11.5 Closing comments

When using Dispatcher to manage a group of users logging into a pool of SP nodes, we recommend that you share the groups' and users' AIX definitions between all the nodes so that users can log onto any nodes rather than limiting these definitions to the nodes where the users are initially authorized to log in. This may help you take advantage of the dynamic reconfiguration possibilities of the RS/6000 SP and of Dispatcher.

Chapter 12. Workload management for Web server

The RS/6000 SP is one of the most powerful Web servers on the market today. The SP provides very high scalability and availability to such Web servers. The multiple node SP configuration with Secureway Dispatcher software provides a load-balanced environment for managing Web servers. The dispatcher software can be configured to manage the client connections to the SP nodes based on node load. In this chapter, we will discuss the workload management scenario with SP nodes functioning as the Web server.

12.1 Scenario description

In this scenario, we consider a computing environment with the following requirements:

- The Web server is configured on multiple nodes in SP.
- The client connections are to be routed to the nodes to balance the workload.

12.2 Tools choice

To test this scenario, we use the following hardware and software tools:

- An RS/6000 SP with four nodes
- The IBM HTTP server
- IBM Secureway Network Dispatcher 2.1

12.3 Configuring the test environment configuration

We will now define the test environment. We use four SP nodes for this test scenario. We will configure the Network dispatcher server in one node, and we will use the other three nodes as a Web server. We will define a shared file system for the Web documents. We will use the default methods for managing the client connections and we will also use the http advisor shipped with the dispatcher to manage the connections.

12.3.1 Hardware and network configuration

The purpose of this scenario is to balance the Web client connections to the SP nodes; so, we will create a simple network configuration between the nodes. This is illustrated in Figure 26 on page 216.

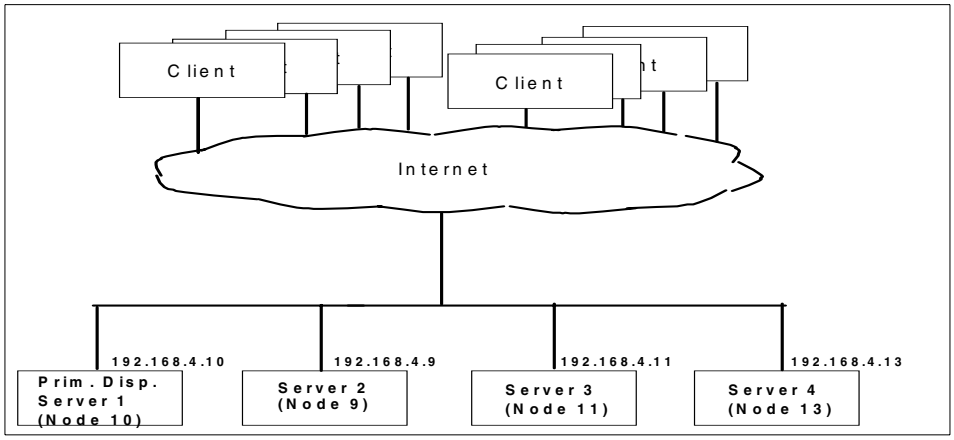


Figure 26. Simple network configuration

We assign the roles listed in Table 11 to the nodes.

Table 11. Role of nodes for ND configuration

Role	Hostname of the node
ND server	sp4n10
ND cluster nodes	sp4n09, sp4n11, sp4n13

We will use only one network interface for network communication. We do not use a switch interface in this scenario.

Table 12. The hostname and IP address for the test environment

Hostname	IP address
sp4n10	192.168.4.10
sp4n09	192.168.4.9
sp4n11	192.168.4.11
sp4n13	192.168.4.13

We will install and configure the IBM HTTP server software in these three server nodes. The three nodes will act together as the Web server for the external clients. The external clients will see the three nodes as a single system through the cluster address defined in the dispatcher configuration.

12.4 Installation and configuration of IBM HTTP server

To configure the Web server on the SP nodes, we need to install the IBM http server software on all four nodes. First, we copied the IBM HTTP server product Version 1.3.6.0 filesets into the control workstation. The following filesets are required to install the IBM HTTP server. We copied these filesets into the /psspimage/http directory in the control workstation.

```
http_server.admin          1.3.6.0  IBM HTTP Server Administration
http_server.base.rte       1.3.6.0  IBM HTTP Server Base Run-Time
http_server.base.source    1.3.6.0  IBM HTTP Server Source Code
http_server.frca           1.3.6.0  IBM HTTP Server Fast Response
http_server.html           1.3.6.0  IBM HTTP Server Documentation
http_server.man.en_US      1.3.6.0  IBM HTTP Server Manual Pages
http_server.modules.mt     1.3.6.0  IBM HTTP Server MT Module
http_server.modules.snmp   1.3.6.0  IBM HTTP Server SNMP Module
http_server.msg.en_US.admin
http_server.msg.en_US.html
http_server.base.rte       1.3.6.0  IBM HTTP Server Base Run-Time
```

We created a user, admuser, and a new group, admingrp . We designated admuser as the administrator for managing the HTTP server.

We installed the HTTP product on the nodes using the `dsh` utility as described below.

```
#dsh -w sp4n09,sp4n11,sp4n13
#dsh mkdir -p /mnthttp
#dsh mount sp4en0:/psspimage/http /mnthttp
#dsh installp -acgNqwX -d /mnthttp http_server.base http_server.html http_server.man_en
```

After the installation, the HTTP server product files are located in the /usr/HTTPServer directory in all the four nodes in our test environment. The binary files are located in the bin directory and the http config files are located in the conf directory. The default documentation directory is located in the htdocs directory.

12.4.1 Basic configuration

To configure the nodes as a Web server, we need to make few changes to the default httpd.conf file in the conf directory. We changed values to the following variables in this configuration file.

```
User admuser
Group admingrp
ServerName sp4n09.msc.itso.ibm.com
<Directory /usr/HTTPServer/htdocs>
DocumentRoot /usr/HTTPServer/htdocs
```

We identified the administrative user and the group in the configuration file. We started modifying the httpd.conf file with node 9 in our configuration; so, the ServerName variable in the configuration file was set to sp4n09.msc.itso.ibm.com. The domain name in the test environment is msc.itso.ibm.com. We made the same changes in nodes 11 and 13. To start the Web server, we need to define the documentation directory were the Web documentation is located. The default documentation directory is /usr/HTTP/htdocs directory. This directory is located in all four nodes in the test environment. We started the http server by issuing following command in the control workstation:

```
# dsh -w sp4n09,sp4n11,sp4n13 /usr/HTTPServer/bin/apachectl start
```

We observed that the http processes were started in these nodes. We connected to this server from a client machine using Netscpae browser. We connected to node 9 and saw the HTTP server page on the browser . We tested the connection for node 10 and node 13, and we got the same html page on the screen.

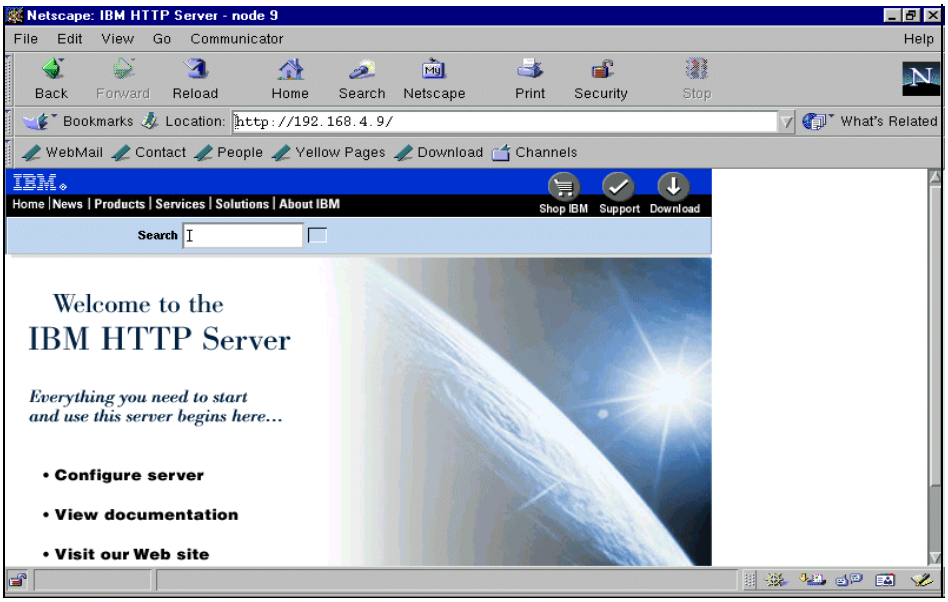


Figure 27. Web document obtained from http server on Node 9

12.4.2 Creating a common documentation directory

In the basic configuration, we configured three http servers on the three SP nodes. The documentation directory is located in the /usr/HTTPServer/htdocs directory for all the servers. In this configuration, we have three http servers serving the same Web document because the content of the directory, /usr/HTTPServer/htdocs, is same. But, this directory is unique to each node and since they have the same contents , we observe the same page irrespective of the node we connect from the client machine. To access this document, the client can connect to any one of the nodes in this cluster. Now, we can configure the three independent Web servers as a cluster Web server using the IBM Secureway Network Dispatcher. Using Dispatcher, we can assign a single cluster address for all three servers, and we can define the http service for port 80 in Dispatcher configuration. In this case, we need to make sure that the content of the document directory is the same in all the nodes. This can be achieved in SP by NFS file systems, multiple copies of independent file systems managed by file collections methods, or by using the GPFS file system.

NFS file system

To maintain the content of the document directory of the Web server using NFS file system , you need to configure an NFS file server in your network

configuration. You need to mount this file system on all the nodes. In this case, any changes to the documents are automatically seen by the http servers, and they serve the same content. The document is accessed over the network from the NFS server for serving a client request.

File collection methods
You can define a file collection for the document directory, and the PSSP file collection methods will replicate the files at regular intervals. In this method, the document directory will be local to each node. The contents of the document directory can be modified in a master node, and it will be replicated to other nodes using PSSP file collection methods. You can define a separate file collection for managing the Web server document directory. Please refer to Section 5.3.3, “Installation and the configuration process” on page 116, for information on how to create a file collection. In this case, the document will be served from the local disk for servicing a Web client request.

GPFS file system
Another alternative in an SP environment is to use a GPFS file system to maintain a document directory. GPFS is a parallel file system; the document directory is maintained by the GPFS servers. In this case, the document directory is a striped file system on all the participating nodes. To configure a GPFS file system, refer to the redbooks *GPFS: A Parallel File system*, SG24-5165, and *Sizing and Tuning of GPFS*, SG24-5610, for information on sizing and performance tuning of the GPFS file system.

You can choose any of the methods just described to create a common document directory for the Web server cluster. You can use the SP switch to improve the network performance for data access.

12.4.3 Configuring the dispatcher

We configured the ND server on sp4n10, and sp4n09, sp4n11, and sp4n13 as http server nodes for load balancing. We followed the procedures described in Chapter 4, “Secureway Network Dispatcher” on page 103, for configuring the ND cluster. The cluster IP address was defined as 192.168.4.100. The configuration steps are defined in the following screen output:

```
ndcontrol executor start
ndcontrol executor set nfa 9.12.0.4
ndcontrol cluster add 192.168.4.100
ndcontrol port add 192.168.4.100:80
ndcontrol port set 192.168.4.100:80 staletimeout 32000000
ndcontrol server add 192.168.4.100:80:192.168.4.9
ndcontrol server add 192.168.4.100:80:192.168.4.13
ndcontrol server add 192.168.4.100:80:192.168.4.11
ndcontrol cluster configure 192.168.4.100 lo0 255.255.255.0
```

We configured port 80 for the http service and added the three servers in this configuration.

Next, we connected to the Web server using the cluster address while the dispatcher was routing the connections to one of the three servers based on the load on each node. The clients will see the cluster a single web server and you can add or remove the nodes in the cluster with out shutting down the cluster. The dispatcher provides different ways to load balance the connections and we will discuss them in the next chapter.

In this chapter we have only discussed how to configure a Web server and balance the workload in a multi-node SP configuration. We have not discussed the various ways the Web server can be configured to improve performance. You can refer to the following documentation for detailed information on these topics:

- *IBM HTTP Server Powered by Apache on RS/6000*, SG24-5132
- *IBM Websphere Performance Pack: Caching and Filtering with IBM Web Traffic Express*, SG24-5859.

Chapter 13. Managing online interactive workloads

Managing with interactive workload in SP is a challenging task for system administrators. Online users always want to connect to a node with the smallest current workload in order to get better response time. It is difficult for the SP administrators and users to find out which node to log in or connect to in order to get better response time. Also, the system administrators want to balance the workload across the SP nodes to utilize the nodes effectively. In an SP environment, it is possible that some nodes will get a very high load while other nodes get lesser or moderate loads.

Earlier, the interactive sessions were managed by the Interactive Sessions Support (ISS) component of the LoadLeveler. ISS was based on DNS implementation and had limited features. The IBM Secureway Network Dispatcher product has many features for managing the interactive sessions. As we discussed in Part 1, the secureway Network Dispatcher manages the TCP/IP connections in a network of servers, and SP can be viewed as a network of servers. With Network Dispatcher, we will be able manage the user connections and balance the workload across the SP nodes.

In this section, we will discuss the management of interactive sessions in SP. We will discuss two ways of managing the interactive sessions. First, we will use the default loadbalancing methods used by ND for routing the connections. In the second method, we will use the custom advisors of ND and provide the node load information using AIX workload manager.

13.1 Scenario description

In this scenario, we consider a computing environment with the following requirements:

- Users create many interactive TCP/IP connections to SP from the client terminals.
- The user connections need to be automatically routed to a least loaded SP node.
- WLM is configured in SP nodes for managing the workload at each node

13.2 Tools choice

To manage user connections and balance the load on the nodes, we chose the following tools :

- IBM Secureway Network Dispatcher Version 2.1

- AIX Workload Manager
- File collection tools of PSSP
- Java development tools for writing custom advisors for ND

13.3 Configuring the test environment

We will define the following test environment. The SP environment we used in the lab has 10 nodes. For this test scenario, we will use only four nodes. We will configure Network Dispatcher and WLM in this environment. We will also define the users' sessions and show how the TCP/IP sessions can be managed using Network Dispatcher. We will also define the Loadbalancing methods for the nodes using WLM. We will write custom advisors to gather the node workload information and use this information to balance the workload in the SP environment.

13.3.1 Hardware and network configuration

The purpose of this scenario is to balance the client connections to the SP nodes; so, we will create simple network configuration between the nodes. This illustrated in the Figure 22.

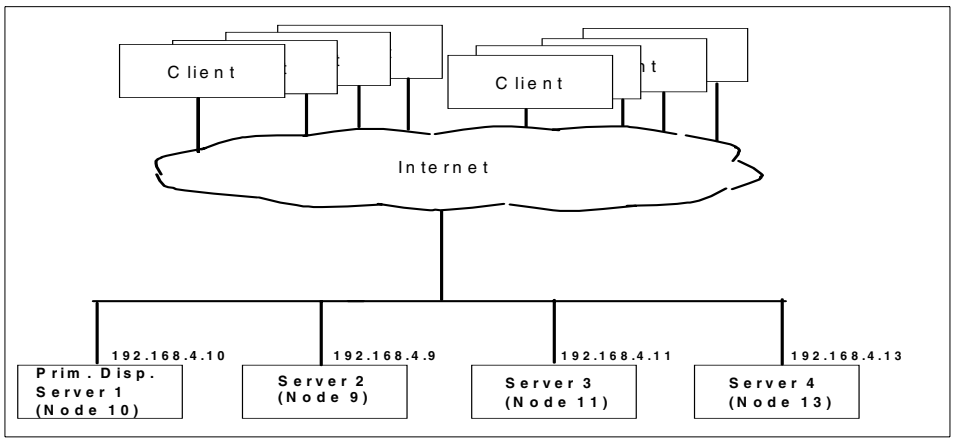


Figure 28. Simple "one-network" configuration.

We assign the roles listed in Table 13 to the nodes.

Table 13. Role of nodes for ND configuration

Role	Hostname of the node
ND server	sp4n10

Role	Hostname of the node
NDcluster nodes	sp4n09, sp4n11, sp4n13

We will use only one network interface for network communication. We do not use a switch interface in this scenario.

Table 14. Hostname and IP address of nodes for the test environment

Hostname	IP address
sp4n10	192.168.4.10
sp4n09	192.168.4.9
sp4n11	192.168.4.11
sp4n13	192.168.4.13

The user sessions we consider here are TCP/IP connections to the SP nodes. Usually, these connections can be any application that is serviced on a TCP/IP port. Most of the client/server applications use a specific TCP/IP port for communication. We can configure these applications in one or more SP nodes, and the Network Dispatcher can be configured to manage the connections to these applications. The application can be a simple telnet service, and users can log on to any node in SP to run their applications. The application can also be a large parallel database server, Web server, parallel file system, and so on. In these scenarios, we can manage the user TCP/IP connections by routing them to a node based on the current workload on the nodes. We can configure nodes with WLM to manage the workload at nodes and we can use Network Dispatcher to distribute the connections.

We test the loadbalancing scenario with TCP/IP application Telnet. We did not create a particular user application environment during this project but the procedure followed here can be applicable to most user environments.

13.3.2 Network dispatcher configuration

We configured the ND server on sp4n10, sp4n09, sp4n11, and sp4n13 as application nodes for load balancing. To configure the ND cluster, we followed the procedures described in Figure 4 on page 103. The cluster IP address was defined as 192.168.4.100. The configuration steps are defined in the following screen output:

```
ndcontrol executor start
ndcontrol executor set nfa 9.12.0.4
ndcontrol cluster add 192.168.4.100
ndcontrol port add 192.168.4.100:23
ndcontrol port set 192.168.4.100:23 staletimeout 32000000
ndcontrol server add 192.168.4.100:23:192.168.4.9
ndcontrol server add 192.168.4.100:23:192.168.4.13
ndcontrol server add 192.168.4.100:23:192.168.4.11
ndcontrol cluster configure 192.168.4.100 lo0 255.255.255.0
```

13.3.3 Managing the user Telnet sessions

For managing user workload in this scenario, the telnet connections, we used the following three methods:

- Default weighted round robin method
- Fixed weight method
- Using Manager and Advisors

13.3.3.1 Default weighted round robin method

The dispatcher is currently configured with three nodes to accept the connections at port 23 for telnet service. We now issue the `telnet` command to the cluster address, telnet 192.168.4.100, from the client terminals, and the dispatcher routes the connection to any of the three nodes. At this point, dispatcher load balances the client connections based on a weighted round robin method. In this method, the user telnet connections are forwarded to the nodes in a round robin fashion. In this method, the default dispatcher configuration sets equal weights to the nodes in the cluster.

13.3.3.2 Fixed weight method

The SP configurations can have different types of nodes, thus, the power of these nodes can differ greatly. It would be appropriate to set different weights to these nodes based on their power. The SP configurations in our lab environment had mixed types of nodes. Node 9 in this cluster is a four CPU PowerPC 332 Mhz node, and nodes 11 and 13 were single CPU 66 Mhz nodes. Node 9 is more powerful than nodes 11 and 13. To manage the workload with mixed nodes, we use the fixed weight method. In this method, we define the weight of node 9 as 10 times that of nodes 11 and 13. We set this weight using the following commands:

```
ndcontrol server set 192.168.4.100:23:192.168.4.13 weight 1
ndcontrol server set 192.168.4.100:23:192.168.4.11 weight 1
ndcontrol server set 192.168.4.100:23:192.168.4.9 weight 10
```

We have now created telnet connections to the cluster, and the dispatcher routes the connections to the nodes based on the weights defined for the nodes. In this scenario, the connections were routed to the nodes in a 1:10 ratio as shown in Figure 29.

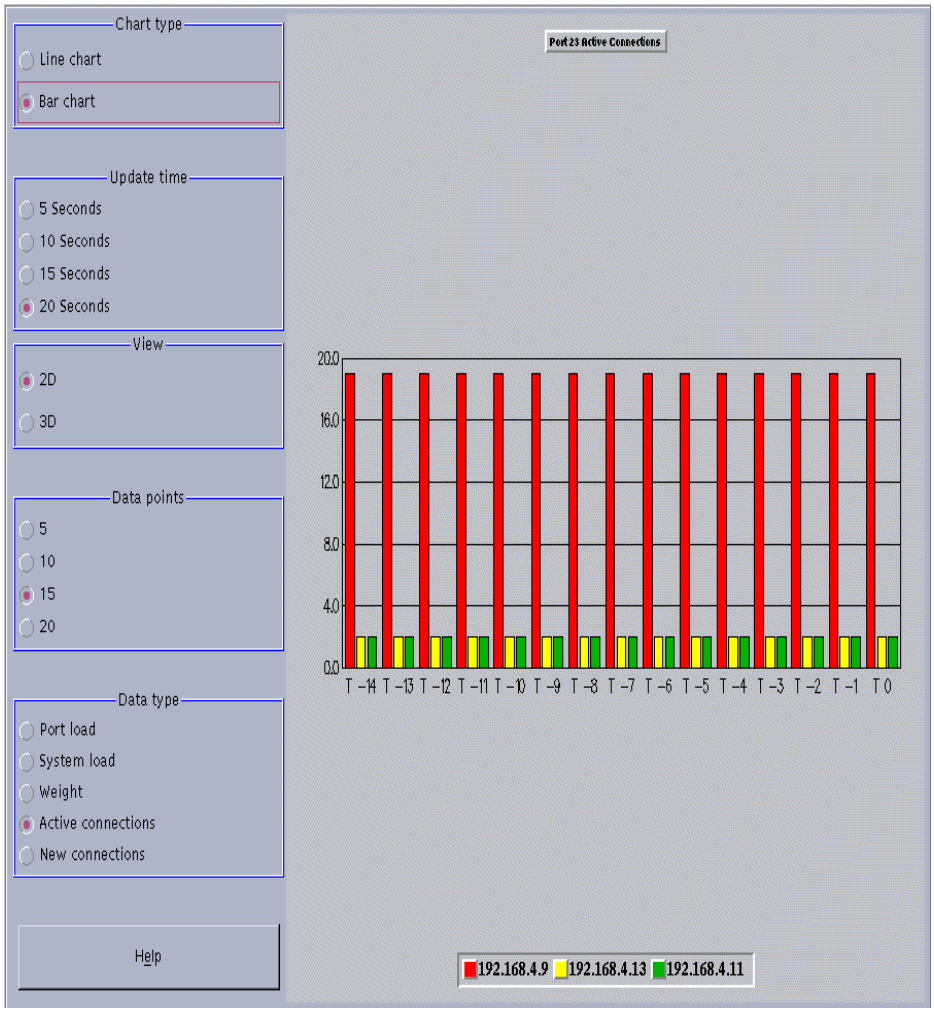


Figure 29. Managing the connections using fixed weight for nodes

13.3.3.3 Using managers and advisors

In the previous two methods, the workload is balanced based on the static weights defined in the Dispatcher configuration. These weights are defined manually by the administrators, and the dispatcher cannot automatically adjust these values based on the current load in the nodes. While these two methods provide certain workload balancing functions based on the weights defined for the nodes, the weights are not adjusted based on the current workload at each node. The Manager feature of dispatcher dynamically sets weights to the nodes in the cluster based on the internal counters in the executor, feedback from advisors, and load information from a system monitoring tool, such as ISS. The manager dynamically calculates the weights based on proportions assigned to the four factors, such as active connections, new connections, advisor information, and system monitoring tools. Based on the proportions settings, the manager feature of dispatcher recalculates the weights to each node in the cluster. The proportions define the importance of the parameter in calculating the weights. The default values are 50 percent for active connections, 50 percent for new connections and zero percent for advisors and system monitoring tools. You can change the proportions to suit your configuration, but the sum of the proportions should be equal to 100. There are certain guidelines for setting these proportions, and they are described in the Network Dispatcher user guide.

At this point in time, we set the proportions to the default values to 50, 50, 0, and 0. The following steps describe how to start the manager and set the proportions in dispatcher configuration. The screen output also shows the current load on the nodes.

```
#ndcontrol manager start
The manager has been started.
# ndcontrol manager proportions 50 50 0 0
The proportions of the manager were set to: 50 50 0 0
```

```
-------------------------------------------------------------------------------------------------
---
|   192.168.4.100|   WEIGHT  |   ACTIVE %  50   |    NEW %  50    |   PORT %   0  | SYSTEM %   0 |
-------------------------------------------------------------------------------------------------
---
| PORT:    23    | NOW | NEW |  WT  |  CONNECT  |  WT  |  CONNECT  |  WT  |  LOAD  |  WT  |  LOAD  |
-------------------------------------------------------------------------------------------------
---
|    192.168.4.13|  14 |  14 |   18 |         0 |   10 |         0 |   10 |      0 |-9999 |     -1 |
|    192.168.4.11|  14 |  14 |   18 |         0 |   10 |         0 |   10 |      0 |-9999 |     -1 |
|    192.168.14.9|   5 |   5 |    0 |        11 |   10 |         0 |   10 |      0 |-9999 |   Down |
-------------------------------------------------------------------------------------------------
---
| PORT TOTALS:   |  33 |  33 |      |        11 |      |         0 |      |      0 |      | -1002 |
-------------------------------------------------------------------------------------------------
```

We created 10 telnet connections, and the connections were distributed as shown in Table 15.

Table 15. With default manager configuration

Node name	Number of connections
SP4n09	2
SP4n11	3
SP4n13	5

Using custom advisors

The advisor feature of the dispatcher sends a TCP request to the nodes and measure the actual client response time for a particular port. These results are fed to the manager for weight calculations. The importance of the advisor in recalculating the weights is defined by the proportions. The advisor function is optional in dispatcher. But, it is recommended that you use the advisor. There are advisors currently available with the Network Dispatcher for certain TCP protocols. You have the option of writing the custom advisors. You can write a custom advisor to provide precise load information for the nodes in the cluster. The custom advisors are to be written and compiled using java Version 1.1. The custom advisors can be written in such a way as to receive the load information from the server application and feed it to the manager for calculating the weights. At this time, you can set the appropriate importance to the advisor proportions for the manager to calculate the weights. In this case, you can set higher proportions for the advisors. It is also recommended to set at least 20 percent of the proportions to each active connection and new connection parameter.

We modified the sample custom advisor code shipped with the dispatcher to receive the load information from the server. We used a separate control port to receive the load information from the server. On the server side, we wrote a simple java code to send the current load information from the server using the control port. We executed this server code on all three servers. We used AIX Workload manager to collect the current load information on the server. To start, we configure the AIX Workload manager on the servers.

WLM configuration on the nodes

We created a user group, telnet, and we allowed telnet access to the users of this group on the three nodes. We created a new WLM configuration directory, /etc/wlm/session, in all the nodes. We also defined a WLM class, tnintjobs, and we defined telnet group to this class. The following is the WLM configuration used in this scenario.

```
WLM CLASS Configuration
System:
        description = ""
        tier   = 0
Default:
tnintjobs:
        description = "Interactive jobs called by telnet users"
        tier   = 0
httpjobs:
        description = "HTTP Service"
        tier   = 0
WLM Rules configuration

* class resvd user group application
tnintjobs -   -    telnet     -
httpjobs -   -       -    /usr/HTTPServer/bin/httpd
System    -   root   -        -
Default   -   -      -        -
```

We activated this WLM configuration using the `wlmcntrl -d /etc/wlm/session` command.

For this scenario, we defined the load on the nodes as a measure of cpuload. We wrote two shell scripts to gather the load information on the system.

The `CPULOAD` script measures the current cpuload on each node.

```
# cat CPULOAD
wlmstat -c |awk '{sum = sum + $2 ; print sum}' | tail -1 |awk '{ print  $1*100+1 }'
```

The WLMLOAD23 script measures the current cpuload on each node defined by the tnintjobs class.

```
# cat WLMLOAD23
wlmstat -l tnintjobs -c |grep tnintjobs |awk '{ print $2/75 * 10000 }' | cut -f 1 -d .
```

The shell scripts used here use the wlmstat command to measure the load on the system based only on the cpuload on the nodes. You can write your own scripts to derive the load on the nodes specific to your configuration.

We put these commands in the cron table on each node so that they execute at regular intervals. The following are the crontab entries.

```
5 * * * *  /tmp/CPULOAD 1>/tmp/cpuload.out
5 * * * *  /tmp/WLMLOAD23 1>/tmp/wlmload23.out
```

The output of these scripts is created in two files in the /tmp directory. These output files read by the simple java application to forward the load information to the dispatcher custom advisor.

Java application on the nodes

We wrote the following piece of java code to read the load information from the output files created by the previous shell scripts to the custom advisor using a control port. This application was developed in Java Version 1.1.

The execution procedure for this program is as follows:

Java server < control port > < file >

<control port> is the number of the port in which the custom advisor will receive the load information.

<file> is the name of the file from which the node load information should be read by this application.

For general cpuload information, for all applications, we use the cpuload.out file. For the cpuload created by the telnet application, we use the wlmload23.out file. The number 23 signifies the TCP/IP port for telnet service.

```
import java.io.*;
import java.net.*;
import java.util.*;

public class server
{
        public static void main(String args[]) throws Exception
        {
          if (args.length != 2)
              {
               System.out.println("\nUsage : java server <port> <file name>");
               System.out.println("        port      - an integer where the server listens");
               System.out.println("        file name - input file");
               System.exit(0);
              }
              ServerSocket serverSocket = null;
              Socket clientSocket = null;
              int port = Integer.parseInt(args[0]);
              String command= args[1];
              serverSocket = new ServerSocket(port);
              while (true)
              {
              clientSocket = serverSocket.accept();
              serviceClient(clientSocket,command);
              }
        }

        public static void serviceClient(Socket client,String command)
        {
          DataInputStream inbound = null;
          PrintStream outbound = null;

         try{
         inbound = new DataInputStream(new BufferedInputStream(client.getInputStream()));
         outbound = new PrintStream(new BufferedOutputStream(client.getOutputStream(),1024),
                                    false);

         tring inputLine=inbound.readLine();
         //System.out.println(inputLine);
         int j=0,count=1;
         String s = null;
         BufferedReader br = new BufferedReader(new FileReader(command));
         s = new String(br.readLine());
         //System.out.println("Data: " + s);
         outbound.println(s);
         outbound.flush();
         outbound.close();
         inbound.close();
         client.close();
         br.close();
         }
         catch(Exception e){
         //System.out.println(e.getMessage());
         }
        }
}
```

Custom advisor code

We modified the sample custom advisor code shipped with the dispatcher product for our requirement. The two specific changes are:

- The custom advisor works in replace mode to pass the load information from the server.
- We use a control port other than the TCP/IP port used by the dispatcher for servicecing the client requests. In this case we use the control port number 8823. This defined in the variable ADV_DEF_CONTROLPORT = 8823.

We named this custom advisor port23. We followed the ND naming conventions for the custom advisors.

We started the dispatcher server and the manager. We started this custom advisor using the following command:

```
ndcontrol advisor start port23  23
```

We started the java application on the nodes by issuing following command:

```
java server 8823 /tmp/wlmload23.out
```

At this point in time, the custom advisor was receiving the cpuload information for telnet class users.

```
/**
 * ADV_port23:  The Network Dispatcher custom advisor for port 23
 *
 *
 * This class defines a sample custom advisor for Network Dispatcher.  Like all
 * advisors, this custom advisor extends the function of the advisor base, called ADV_Base.
 * It is the advisor base that actually performs most of the advisor's functions,
 * such as reporting  loads back to the Network Dispatcher for use in the
 * Network Dispatcher's weight algorithm.  The advisor base also performs socket connect
 * and close operations and provides send and receive
 * methods for use by the advisor.  The advisor itself is used only for
 * sending and receiving data to and from the port on the server being advised.
 * The TCP methods within the advisor base
 * are timed to calculate the load.  A flag within the constructor in the ADV_base
 * overwrites the existing load with the new load returned from the advisor if desired.
 *
 * Note: Based on a value set in the constructor,  the advisor base supplies
 * the load to the weight algorithm at specified intervals.  If the actual
 * advisor has not completed so that it can return a valid load, the advisor base uses
 * the previous load.
 *
 * NAMING
 *
 * The naming convention is as follows:
 *
 *  - The file must be located in the Network Dispatcher base directory.
 *    The defaults for the directory vary by operating system:
 *
 *    - NT  - \Program Files\nd\dispatcher
 *    - AIX - /usr/lpp/nd/dispatcher
 *    - Solaris - /opt/nd/dispatcher
 *      within the subdirectory of lib\CustomAdvisors.
 *
 * - The Advisor name must be preceded with "ADV_".  The advisor can
 *    be started with only the name, however; for instance, the "ADV_port23"
 *    advisor can be started with "port23".
 *
 * - The advisor name must be in lowercase.
 *
 *  With these rules in mind, therefore, this port23 is referred to as:
 *
 *             <base directory>/lib/CustomAdvisors/ADV_port23.class.
 *
 *
 * Advisors, as with the rest of Network Dispatcher, must be compiled with Java 1.1.5.
 * To ensure access to Network Dispatcher classes, make sure that the ibmnd.jar
 * file (located in the lib subdirectory of the base directory) is included in the system's
 * CLASSPATH.
 *
 *
```

```
* Methods provided by ADV_Base:
 *
 * - ADV_Base (Constructor):
 *
 *   - Parms
 *     - String sName = Name of the advisor
 *     - String sVersion = Version of the advisor
 *     - int iDefaultPort = Default port number to advise on
 *     - int iInterval = Interval on which to advise on the servers
 *     - String sDefaultLogFileName = Unused.  Must be passed in as "".
 *     - boolean replace = True - replace the load value being calculated by the advisor
 *       base False - add to the load value being calculated by the advisor base
 *     - Return
 *     - Constructors do not have return values.
package CustomAdvisors;
import java.io.*;
import java.net.*;
import com.ibm.internet.nd.advisors.*;
public class ADV_port23 extends ADV_Base implements ADV_MethodInterface
{
  String COPYRIGHT = "(C) Copyright IBM Corporation 1997, All Rights Reserved.\n";

  static final String  ADV_NAME              = "port23";
  static final int     ADV_DEF_ADV_ON_PORT   = 23;
  static final int     ADV_DEF_INTERVAL      = 7;
  static final int     ADV_DEF_CONTROLPORT   = 8823;

  // Note: Most server protocols require a carriage return ("\r") and line feed ("\n")
  //       at the end of messages.  If so, include them in your string here.
  static final String  ADV_SEND_REQUEST      =
    "hello";

  /**
   * Constructor.
   *
   * Parms:  None; but the constructor for ADV_Base has several parameters that must be
   * passed to it.
   *
   */
  public ADV_port23()
  {
    super( ADV_NAME,
                  "2.0.0.0-03.27.98",
           ADV_DEF_ADV_ON_PORT,
           ADV_DEF_INTERVAL,
           "",     // not used
           true);
    super.setAdvisor( this );
  }
```

```
  /**
   * ADV_AdvisorInitialize
   *
   * Any Advisor-specific initialization that must take place after the advisor
   * base is started.
   * This method is called only once and is typically not used.
   */
  public void ADV_AdvisorInitialize()
  {
    return;
  }
public int getLoad(int iConnectTime, ADV_Thread caller)
  {
        int iRc;
        int iLoad = ADV_HOST_INACCESSIBLE;  // -1
        Socket SoServer = null;
// Send tcp request
        iRc = caller.send(ADV_SEND_REQUEST);
        if (iRc >= 0)
        {
                // Perform a receive
                StringBuffer sbReceiveData = new StringBuffer("");
                iRc = caller.receive(sbReceiveData);
if (iRc >= 0)
                {
                        String Sserver = caller.getCurrentServer();
                        try
                        {
                        SoServer =  new Socket(Sserver,ADV_DEF_CONTROLPORT);
                        PrintStream outbound = new PrintStream(
                            SoServer.getOutputStream());
                        DataInputStream inbound = new DataInputStream(
                            SoServer.getInputStream());
                        outbound.println("Hello");
                        outbound.flush();
                        String fromServer;
                        while ((fromServer = inbound.readLine()) != null)
                        iLoad = Integer.parseInt(fromServer.trim());
                        inbound.close();
                        }
                        catch(Exception e){
                        iLoad = 0;
                        }
                        try
                        {
                        SoServer.close();
                        }
                        catch(Exception e){}
}
        }
        return iLoad;
  }

} // End - ADV_port23
```

Test 1

To test how the telnet connections are routed to the nodes based on the load information from the nodes, we created initial cpuloads on two nodes: sp4n11 and sp4n13. To do this, we ran a program under the user, telnet1, who is a member of the user group telnet. We measured the cpuload created by this user using the WLMLOAD23 script on the nodes. This script reported a load factor of 12133 in sp4n13 and 12533 in sp4n11. We did not create any load on sp4n09. We set the manager proportions as 20, 20, 60, and 0. That is, 20 percent each for active connections and new connections and 60 percent for advisors. We obtained the following report from the dispatcher manager function.

```
-------------------------------------------------------------------------------------------------
192.168.4.100|   WEIGHT  |   ACTIVE %  20   |     NEW %  20    |   PORT %  60  |  SYSTEM %   0 |
-------------------------------------------------------------------------------------------------
| PORT:    23    | NOW | NEW |  WT  |  CONNECT  |  WT  |  CONNECT  |  WT  |  LOAD  |  WT  |  LOAD  |
-------------------------------------------------------------------------------------------------
|    192.168.4.13|   7 |   7 |  15 |         0 |    8 |         1 |    5 |  12133 |-9999 |     -1 |
|    192.168.4.11|   7 |   7 |  15 |         0 |    8 |         1 |    5 |  12533 |-9999 |     -1 |
|    192.168.14.9|  13 |  13 |   0 |        12 |   12 |         1 |   19 |      0 |-9999 |   Down |
-------------------------------------------------------------------------------------------------
| PORT TOTALS:   |  27 |  27 |     |        12 |      |         3 |      |  24666 |      |  -1002 |
-------------------------------------------------------------------------------------------------
```

Now, we create 10 telnet sessions, and the connections are routed to the nodes in the following way:

Table 16. With propotions setting 20, 20, 60, 0

Node name	Number of connections
Sp4n13	3
sp4n11	2
sp4n09	5

As we see in Table 16, the least loaded node received the most connections.

Now, we change the manager proportions to 0, 0, 100, 0. This is done to assign the maximum weight to advisor load information. Again, we create ten telnet connections to the cluster, and the connections are routed as listed in Table 17.

Table 17. With manager proportion 0, 0, 100, 0

Node name	Number of connections
SP4n11	0

Node name	Number of connections
SP4n13	0
SP4n09	10

As we see from the above results, all the new connections were routed to a node with the least of the cpu loads.

Test 2

In this test, we did not create any initial cpuload on the nodes, and we started with no load on the nodes. The WLMLOAD23 script returned 0 for all the nodes. We set the manager proportions to 0, 0, 100, and 0 setting maximum weight to the advisor input. We initiated the telnet sessions to the cluster. The first connection was routed to node sp4n13. We logged on as telnet1 user and started a CPU-intensive job on this node. We waited for the cronjob WLMLOAD23 script to calculate the cpuload for the tnintjobs class. The WLMLOAD23 script returned a cpuload factor of 12533 to the advisor once the cron job was completed. The advisor returned load 12533 for node 13 and 0 for sp4n11 and sp4n09. This started the second command to the cluster, and, this time, the connection was routed to sp4n11. Again, we logged on to the node as telnet1 user and started the CPU-intensive job on this node; we waited for the cron job to complete, and the advisor-returned cpuload factor for this node was 12133. Next, we issued the third telnet connection, and, this time, the dispatcher routed this connection to sp4n09. We also logged on as telnet1 user and issued the same CPU-intensive job on this node. After waiting for the cronjob to complete, we got the following report from the manager:

```
-------------------------------------------------------------------------------------------------------
|   192.168.4.100|   WEIGHT  |   ACTIVE %   0   |     NEW %   0    |   PORT % 100  | SYSTEM %   0  |
-------------------------------------------------------------------------------------------------------
| PORT:    23    | NOW | NEW |  WT  |  CONNECT  |  WT  |  CONNECT  |  WT  |  LOAD  |  WT  |  LOAD  |
-------------------------------------------------------------------------------------------------------
|    192.168.4.13|   3 |   3 |   11 |         1 |    2 |         1 |    3 |  12133 |-9999 |     -1 |
|    192.168.4.11|   1 |   1 |    8 |         2 |    0 |         1 |    1 |  12533 |-9999 |     -1 |
|    192.168.14.9|  20 |  24 |   10 |        13 |   26 |         0 |   24 |   3333 |-9999 |   Down |
-------------------------------------------------------------------------------------------------------
| PORT TOTALS:   |  24 |  28 |      |        16 |      |         2 |      |  27999 |      |  -1002 |
-------------------------------------------------------------------------------------------------------
```

The cpuload returned for node 9 was 3333. While we used the same CPU-intensive job in all the nodes, the load factor for node 9 was less compared to nodes 11 and 13. This is because node 9 was a different type of node than nodes 11 and 13. Node 9 was a four-CPU 332 Mhz node while nodes 11 and 13 were weaker 66 Mhz nodes. At this point in time, we see node 9 as the

least loaded node, and, so, as expected, all subsequent telnet connections were routed to this node until the load on this node became nearly equal to the load on the other two nodes.

13.3.4 Using multiple applications

The dispatcher software can be used to manage the load on SP if more than one application is running on SP nodes. To load balance the multiple applications in SP nodes, we can use both dispatcher and AIX workload manager. While AIX workload manager can manage the resources of the individual nodes, the dispatcher can route the TCP/IP connections for the application to the nodes with the least load. To manage the load using custom advisors, you need to write one custom advisor per port. You also need to configure the WLM to allocate the resources for the applications in the nodes. We tested such an environment using two applications: Telnet and HTTP.

To do this, we configured three nodes, sp4n09, sp4n11, and sp4n13, as Web servers accessing the same Web document. This Web document was seen by the three servers through an NFS file system. You can also configure a GPFS in SP to perform this function. We defined the following limits in the WLM configuration file in each node:

```
# cat limits
tnintjobs:
        memory = 1%-75%
        CPU    = 1%-75%
httpjobs:
        memory = 1%-25%
        CPU    = 1%-25%
System:
        memory = 1%-100%
        CPU    = 1%-100%
```

We have assigned 75 percent of the resources to telnet jobs and 25 percent of the resources to the Web server. This is defined as `tnintjobs` and `httpjobs` in the limits file of WLM session configuration directory. We wrote another custom advisor similar to the custom advisor we wrote for the telnet session with control port 8880. We created following script file using the WLM command for gathering load information for http jobs. The script was called WLMLOAD80. The number 80 signifies the port number for http.

```
# cat WLMLOAD80
wlmstat -l httpjobs -c |grep httpjobs |awk '{ print $2/25*10000 }'
| cut -f 1 -d .
```

We added this job to the crontab as we did in the previous test to collect the load information at regular intervals. We redirected the output of this script to the /tmp/wlmload80.out file. This file will be the input file for the Java application to send the load information to the advisor. Since we now have two custom advisors for port 23 for telnet and port 80 for http, we need to start two java applications on each node to read the data from the file and send to the custom advisor; so, we issued the following commands on each node:

```
Java server 8823 /tmp/wlmload23.out
Java server 8880 /tmp/wlmload80.out
```

The two files, `wlmload23.out` and `wlmload80.out`, provide the load information, and these files are updated at regular intervals by the cronjob. We also set the dispatcher manager propotions to 0, 0, 100, and 0. In this setting, the advisor load information will get the maximum weight while calculating the load information.

In this configuration, we observed that the telnet connections were routed to the nodes based on the load information for tnintjobs, and the http connections were routed to the nodes based on the load information for httpjobs. The load information for these two applications was obtained using the WLM command `wlmstat`. Also, the node resources were allocated to these jobs as defined in the limits file of the WLM configuration.

13.3.5 Remarks on using the custom advisors and WLM

When using custom advisors, the compiled code should be placed in the /usr/lpp/nd/dispatcher/lib/Custom advisors directory. For every port that you want to manage, you need to write one customer advisor code in Java Version 1.1 and change the port number, control port number, and the request string specific to that port. You need not write any java code to run on the server, but you need to execute the server code on each node. You have to run this server code for every port in order to pass load information to the dispatcher. You have to specify a one input file per port created by a WLM script. You also need to write a shell script to calculate the workload information and place it in a cronjob to execute at frequent intervals. When setting the proportions for the manager to 0, 0, 100, 0, the workload is balanced entirely based on the load information provided by the Custom Advisors. The dispatcher load balancing algorithms are not used with these settings. As we mentioned earlier, the dispatcher software recommend to set the new and active connections proportions to, at least, 20,20. It is recommended to start with the 20, 20, 60, 0 settings for the manager and gradually change these proportions based on the load balancing pattern for your specific application and system configurations. You can use WLM to allocate the resources within a node.

Appendix A. Special notices

This publication is intended to help IBM customers, Business Partners, IBM System Engineers and other RS/6000 SP specialists who are involved in workload management projects in SP. The information in this publication is not intended as the specification of any programming interfaces that are provided by product LoadLeveler Version 2 and Release 1 and AIX Version 4 Release 3 and modification 3 and Secureway Network Dispatcher Version 2 Release 1 . See the PUBLICATIONS section of the IBM Programming Announcement for LoadLeveler Version 2 Release 1 and AIX Version 4 Release 3 and modification 3 and Secureway Network Dispatcher Version 2 Release 1 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM

assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	AS/400
eNetwork	GPFS
Home Director	IBM
LoadLeveler	Netfinity
NetView	PESSL
POE	POWER 3
POWERparallel	PowerPC
PSSP	RS/6000
SecureWay	SP
System/390	WebSphere

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other

countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Appendix B. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

B.1 IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 247.

- *GPFS: A Parallel File System*, SG24-5165
- *IBM HTTP Server Powered by Apache on RS/6000*, SG24-5132
- *IBM Websphere Performance Pack: Caching and Filtering with IBM Web Traffic Express*, SG24-5859
- *Load Balancing for eNetwork Communications Servers,* SG24-5305
- *New and Improved : IBM WebSphere Performance Pack: Loadbalancing with IBM Secureway Network Dispatcher*, SG24-5858
- *RS/6000 Performance Tools in Focus*, SG24-4989
- *RS/6000 Scalable POWERparallel System: Scientific and Technical Computing Overview*, SG24-4541
- *RS/6000 SP Performance Tuning*, SG24-5340
- *Server Consolidation on RS/6000*, SG24-5507
- *Sizing and Tuning GPFS*, SG24-5610
- *The RS/6000 SP Inside Out*, SG24-5374

B.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at http://www.redbooks.ibm.com/ for information about all the CD-ROMs offered, updates, and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849

CD-ROM Title	Collection Kit Number
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr Format)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

B.3 Other resources

These publications are also relevant as further information sources:

- *LoadLeveler for AIX: Using and Administering*, SA22-7311
- *Parallel Environment for AIX: Operation and Use Vol.1*, SC28-1979
- *PSSP: Administration Guide*, SA22-7348
- *SecureWay Network Dispatcher for AIX, Solaris, and Windows*, GC31-8496

The following publication is product documentation and must be purchased with the software product:

- *IBM LoadLeveler for AIX Version 2 Release 1, Installation Memo*, GI10-0642

B.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- `http://www.tc.cornell.edu`
- `http://www.mhpcc.edu`
- `http://www.ibm.com/software/network/dispatcher`
- `http://www.ibm.com/servers/aix/library`
- `http://www.ibm.com/servers/aix/library`

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	Order Number	Quantity

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number _____

☐ Credit card number _____

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

ACL. Access Control List.

AFS. Andrew File System.

AIX. Advanced Interactive Executive.

API. Application Programming Interface.

ARP. Address Resolution Protocol.

BSD. Berkeley Software Distribution.

CBR. Context Base Routing.

CPU. Central Processing Unit.

CSS. Communication Subsystem.

CWS. Control Work Station.

DFS. Distributed File System.

DNS. Domain Name Server.

ESSL. Engineering and Scientific Subroutine Library.

FIFO. First in first out.

FTP. File Transfer Protocol.

GPFS. General Parallel File System.

IBM. International Business Machines.

IP. Internet Protocol.

ISS. Interactive Session Support.

ITSO. International Technical Support Organization.

LAN. Local Area Network.

LAPI. Low Level Application Programming Interface.

LPP. Licensed Program Product.

MB. Megabytes.

MPI. Message Passing Interface.

MPL. Message Passing Library.

MPP. Massive Parallel Processors.

ND. Network Dispatcher.

NFS. Network File System.

PE. Parallel Environment.

PESSL. Parallel Engineering and Scientific Subroutine Library.

POE. Parallel Operating Environment.

PSSP. Parallel System Support Program.

PTF. Problem Temporary Fix.

PVM. Parallel Virtual Machine.

SDR. System Data Repository.

SMIT. System Management Interface Tool.

SMP. Symmetrical Multiprocessing.

SP. RS/6000 SP.

TCP. Transmission Control Protocol.

WCOLL. Working Collection.

WLM. Work Load Manager.

WEBSM. Web-based System Manager.

WTE. Web Traffic Express.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

S

T

U

V

W

X

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at http://www.redbooks.ibm.com/
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number **Redbook Title**	SG24-5522-00 Workload Management: SP and Other RS/6000 Servers
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	O Very Good O Good O Average O Poor
Please identify yourself as belonging to one of the following groups:	O Customer O Business Partner O Solution Developer O IBM, Lotus or Tivoli Employee O None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. http://www.ibm.com/privacy/yourprivacy/

SG24-5522-00
Printed in the U.S.A.

Workload Management: SP and Other RS/6000 Servers

SG24-5522-00