IBM LoadLeveler
Administration and Installation Planning Guide

Document Number SH26-7220-02

January 26, 1994

Susann Dewey

Information Development
Dept. 34CA 634 Bldg. 202/2
Kingston, NY
Tie Line 8-695-4121
VNET: TADJ @ KGNVMC

IBM LoadLeveler
Administration and Installation Planning Guide

SH26-7220-02

IBM

IBM LoadLeveler
Administration and Installation Planning Guide

SH26-7220-02

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page iv.

First Edition, March 1994

This edition applies to Version 1 Release 1.3 of IBM LoadLeveler, (5765-145 for RISC System/6000 and 9076 Scalable POWERparallel Systems), (5765-227 for Sun), (5765-228 for SGI), (5765-287 for HP) and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for reader's comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
Department 34CA
Neighborhood Road
Kingston, NY
12401

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

Copyright International Business Machines Corporation 1993. All rights reserved. Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.. Contains licensed material — Property of IBM. Copyright 1986, 1987, 1988, 1989, 1990, 1991 by the Condor Design Team.

Contents

Notices iv
Trademarks and Service Marks iv
About This Book v
Who Should Use This Book v
How to Use This Book v
Overview of Contents v
Terms and Document Conventions vi
Related Publications vi
Ordering Additional Copies of This Book vi
Summary of New Functions vii
Parallel Job Submission Support vii
Job Accounting vii
Increased Support for Machines Running NQS vii
Submit-only Workstations vii
Additional Hardware Platforms Supported vii
Interactive Session Support viii

Chapter 1. Getting Started with LoadLeveler 1
How LoadLeveler Works 4

Chapter 2. Configuring and Administering LoadLeveler 9
Where to Begin? 10
Quick Set Up 10
Configuration File Structure and Syntax 11
Customizing the Configuration File 14
Customizing the Administration File 33
Checkpointing and Migrating Jobs During Execution 40
Checkpointing for the RISC System/6000 42
Checkpointing for the Sun 42
Checkpointing for the SGI 43
Checkpointing for the HP 44

Chapter 3. Working with the SP1 45
Scheduling Parallel Jobs to the SP1 45
Using Interactive Session Support 50

Chapter 4. Routing Jobs to NQS Machines 53

Chapter 5. Centralized Job Accounting 59
Accounting Files 59
Account Validation 59
Accounting Reports 60
Sample Job Accounting Scenario 61

Chapter 6. LoadLeveler Commands 63
The LoadLeveler Graphical User Interface 64
How to Read the Syntax Diagrams 67
llacctmrg - Collects machine history files 69
llctl - LoadLeveler Control 71
llfavorjob - Reorder System Queue by Job 73

llfavoruser - Reorder System Queue by User 75
llpreen - Search for Unexpected Files in LoadLeveler Directories 76
llsummary - Returns job resource information 77
llcancel - Cancel a Submitted Job 82
llhold - Hold or Release a Submitted Job 84
llprio - Change Priority of Submitted Job 87
llq - Query Job Status 89
llstatus - Query Machine Status 93
llsubmit - Submit a Job 97
NQS Scripts 98

Appendix A. Planning for Installation 99
Planning for Installation 99
List of Program Material 104
Statement of Program Support 105

Appendix B. Diagnosing LoadLeveler Problems 107
Getting Help from IBM 109

Appendix C. LoadLeveler Job Command File and Keywords 111
LoadLeveler Keywords 111

Appendix D. Sample Files 121
Sample Configuration File 121
Sample Administration File 125

Appendix E. LoadLeveler Daemons and Processes 127

Glossary of Terms and Abbreviations 135

Index 139

Notices

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 product, program, or service that does not infringe on any of IBM's intellectual property rights or other legally protectable rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

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 Commercial Relations, IBM Corporation, Purchase, NY 10577.

LoadLeveler incorporates Condor, which was developed at the University of Wisconsin, and uses it with the permission of its authors.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and other countries:

AIX	AIXwindows
IBM	RISC System/6000
LoadLeveler	Scalable POWERparallel Systems
9076 SP1	

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows:

Motif	Open Software Foundation Incorporated
NFS, Sun	Sun Microsystems Incorporated
HP	Hewlett-Packard Company
UNIX	Unix System Laboratories Incorporated
X Window System	Massachusetts Institute of Technology
CRAY	CRAY Research, Incorporated
SGI	Silicon Graphics, Incorporated
NQS, NQS Exec	Sterling Software Incorporated
AFS	Transarc Corporation

About This Book

This manual is designed to help you install and administer the LoadLeveler program product on your network. It describes how to tune LoadLeveler for increased efficiency in your environment and explains the various commands and files provided for use in the configuration process.

Who Should Use This Book

This manual is intended for those who are responsible for LoadLeveler system administration. The role of the system administrator normally includes the following tasks:

Installing LoadLeveler
Setting up configuration files
Writing expressions that daemons use to control access to the compute resources of the configured workstations
Maintaining LoadLeveler
Setting up the distributed environment for allocating batch jobs.

In order to perform the tasks described in this manual, the LoadLeveler system administrator must be an experienced user of UNIX** commands and be familiar with system management techniques such as SMIT, used in the AIX* environment. Knowledge of networking and NFS** or AFS** protocols will also prove helpful.

How to Use This Book

Overview of Contents

Chapter 1, “Getting Started with LoadLeveler” on page 1 describes LoadLeveler, its components, and how they work.

Chapter 2, “Configuring and Administering LoadLeveler” on page 9 is a comprehensive discussion about all the LoadLeveler configuration options available. Detailed descriptions of all LoadLeveler administration tasks are included.

Chapter 3, “Working with the SP1” on page 45 describes how to schedule batch parallel jobs to the SP1 and how to perform interactive login sessions.

Chapter 4, “Routing Jobs to NQS Machines” on page 53 describes LoadLeveler's role in routing jobs to NQS machines.

Chapter 5, “Centralized Job Accounting” on page 59 describes how to gather accounting data on jobs.

Chapter 6, “LoadLeveler Commands” on page 63 explains the LoadLeveler commands and describes the inputs and requirements for each. It also describes the LoadLeveler graphical user interface.

Appendix A, “Planning for Installation” on page 99 provides planning tasks to be completed before installing the LoadLeveler code.

Appendix B, “Diagnosing LoadLeveler Problems” on page 107 describes problem determination procedures.

Appendix C, “LoadLeveler Job Command File and Keywords” on page 111 describes the various keywords you can define when you submit a LoadLeveler job.

Appendix D, “Sample Files” on page 121 contains sample configuration and administration files.

Appendix E, “LoadLeveler Daemons and Processes” on page 127 describes the LoadLeveler program components.

A glossary and index are also included.

Terms and Document Conventions

Wherever the term UNIX appears in this manual, AIX, the IBM implementation of UNIX also applies.

This manual uses visual cues to help you locate and identify information quickly and easily. The following document conventions are used:

Type Style	Used for
bold	Command names, program names, file names, flag names, directory names, path names, keywords, and any text that you must type exactly as it appears.
italic	Parameters and variables. You can substitute the text for items shown in italic. For example, when you have to enter a variable called *filename* you must type the name of the appropriate file in place of *filename*. New terms are introduced in italic. Italic type is also occasionally used for emphasis.
monospace	Examples of input text and system prompts or responses that appear on the screen.

Related Publications

IBM LoadLeveler: General Information, GH26-7227

IBM LoadLeveler: User's Guide, SH26-7226

IBM LoadLeveler: Quick Reference Summary, SX23-1048

Ordering Additional Copies of This Book

To order additional copies of this book, request form number SH26-7220.

Summary of New Functions

Following is a brief description of the new functions contained in this release:

Parallel Job Submission Support

LoadLeveler now supports parallel job submission for the IBM 9076 SP1. By providing a parallel program written using the Application Programming Interface (API) of the IBM AIX Parallel Environment program product and by modifying your job command file to specify the number of parallel machines you will need, you can submit a parallel job to the IBM 9076 Scalable POWERparallel Systems (SP1).

Job Accounting

LoadLeveler provides accounting information on completed LoadLeveler jobs and stores this information in a file. This job information includes the job parameters, the amount of resources consumed by the job, as well as other job related data. Administrators can collect data on completed jobs submitted from one machine or on all completed jobs on all machines and merge this data into one central file. This feature can help keep track of the different types of jobs being processed, as well as how many resources each type of job requires for processing.

Increased Support for Machines Running NQS

In Release 1.0, LoadLeveler translated scripts containing NQS options into LoadLeveler scripts and submitted the jobs to a machine in the LoadLeveler cluster. Now with the increased support for machines running NQS, LoadLeveler can route any of your jobs (LoadLeveler jobs or non-LoadLeveler jobs) to machines outside of the LoadLeveler cluster that run NQS. This additional support allows you to use all of your machines that run NQS while at the same time taking advantage of LoadLeveler's other jobs scheduling features. In this way, you can include, for example, a CRAY in your pool of resources to be managed by LoadLeveler.

Submit-only Workstations

Your installation can now take advantage of the submit-only machine feature that allows workstations that are not part of the LoadLeveler pool to submit jobs to the pool for processing. In addition, you can use these workstations to cancel and query jobs. No LoadLeveler license fee is charged for these workstations.

Additional Hardware Platforms Supported

LoadLeveler now supports the Sun workstation from Sun Microsystems, Incorporated, the SGI workstation from Silicon Graphics, Incorporated, and the HP Apollo 9000 series 700 from Hewlett-Packard Company. In addition, the IBM RISC System/6000 and the IBM 9076 Scalable POWERparallel Systems (SP1) are also supported. The addition of the Sun, SGI, and HP platforms allows you to expand your network cluster into a more heterogeneous environment. LoadLeveler also plans to support additional platforms in the future.

Interactive Session Support

Interactive Session Support (ISS) distributes login and application sessions across the nodes of an SP1 and a cluster comprised of RISC System/6000 workstations by interfacing to a TCP/IP nameserver that converts a machine name to an internet address. With ISS, a specified machine name corresponds to specified nodes in the SP1. ISS polls the specified nodes regularly to determine which node is best suited to support a login session, and provides the internet address of this node to the nameserver. A user or application connects to the SP1 by specifying a generic SP1 or cluster node name which corresponds to the least-loaded node. In this way, ISS balances user logins across all the available nodes of an SP1 or cluster of RISC System/6000s.

Chapter 1. Getting Started with LoadLeveler

LoadLeveler is a batch job scheduling application that allows users to run more jobs in less time by matching their processing needs to available resources. As a job scheduler, LoadLeveler runs as a set of daemons on each node in the network. It provides a facility for building, submitting, and processing batch jobs quickly and efficiently in a dynamic environment.

Batch jobs can be either *serial* jobs or jobs that have been *parallelized* to take advantage of the parallel architecture of the IBM Scalable POWERparallel Systems* 9076 SP1* (SP1). The SP1 is a scalable, parallel, high-performance computing system that supports batch serial and parallel processing. Its software allows users to fully utilize the computing power of the SP1 architecture. Serial jobs are processed sequentially whereas, in this context, parallel jobs are jobs that have been coded using the Application Programming Interface (API) of the IBM AIX Parallel Environment* program product. Parallel batch jobs are as easy to schedule as are serial batch jobs. LoadLeveler just requires that users make a simple modification to their job command files and then schedules their parallel job to an SP1 machine.

NQS Support

With the increased support for machines outside of the LoadLeveler cluster that run the Network Queueing System (NQS), you can now run your LoadLeveler jobs or NQS scripts either on machines in the LoadLeveler cluster or on machines outside of the cluster. If you are submitting an NQS script to LoadLeveler for processing, LoadLeveler handles it in one of the following ways:

1. LoadLeveler translates the NQS options in the script to equivalent LoadLeveler commands and submits the job to a machine in the LoadLeveler cluster for processing.
2. LoadLeveler routes the NQS script to an NQS machine outside of the LoadLeveler cluster for processing.

LoadLeveler's approach to network job scheduling for UNIX batch jobs differs from the NQS approach. While the NQS-based approach involves managing pipe and batch queues, LoadLeveler lets you view the total computational capabilities as one single source. Therefore, job requirements are dynamically matched with the best available resources based on your preferences, globally defined scheduling policies, and priorities. For more information, see Chapter 4, “Routing Jobs to NQS Machines” on page 53.

Additional Platform Support

LoadLeveler schedules jobs on machines in the cluster that have the right operating system and architecture to provide the best turn-around possible. This cluster can consist of RISC System/6000s, an SP1, and the newly supported Sun Microsystems**, Silicon Graphics**, and HP** workstations.

In addition to supporting different hardware platforms, LoadLeveler now schedules a variety of types of jobs that can run on some or all of the hardware platforms supported.

Job Accounting

LoadLeveler stores accounting information on completed LoadLeveler jobs in a file. This job information includes the job parameters, the amount of resources consumed by the job, as well as other job related data. Administrators can collect data on jobs submitted from a single machine or on all completed jobs submitted from all machines and merge this data into one central file. This feature can help keep track of the different types of jobs being processed, as well as how many resources each type of job requires for processing.

When a job ends, LoadLeveler stores the information in a file named *history* in the spool directory of the machine that initially scheduled the job.

Resource information collected on the job is constrained by the capabilities of the wait 3 system call. Information for processes which fork child processes will include data for those child processes as long as the parent waits for the child process to end. In addition, LoadLeveler may not collect data for jobs which are not composed of simple parent/child processes. For example, if a LoadLeveler job invokes an rsh command to execute a function on another machine, the resources consumed on the other machine will not be collected as part of the LoadLeveler accounting data. For more information, see Chapter 5, “Centralized Job Accounting” on page 59.

Submit-only Workstations

Your installation can now take advantage of the submit-only machine feature that allows workstations that are not part of the LoadLeveler pool to submit jobs to the pool for processing. As an administrator, you need to add the executables for the submit, cancel, and query commands to the submit-only workstations. To do this, see the Memo to Users or the README file.

Personalized Workstations

Through LoadLeveler, all participating users can exploit the pooled idle compute power of a distributed network or a cluster. Users can also allow their workstations to participate in the LoadLeveler pool without degrading the performance of their local machines. They can do this by specifying when their personal machines are available to the pool. They can specify that their machine:

Always be available
Be available only between certain hours
Be available only when the keyboard and mouse are idle
Never be available.

Regardless of the availability of their machines, users can submit jobs designated by both class and priority to be dispatched to the available workstations and central servers within the LoadLeveler resource pool.

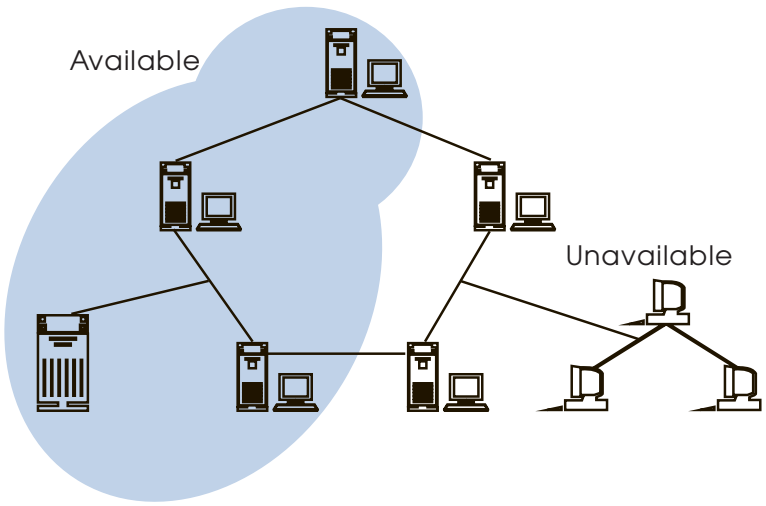


Figure 1. Individual Configurability in the LoadLeveler Pool

LoadLeveler determines which jobs to dispatch to certain machines by examining information such as the machine availability and current workload, and job requirements such as memory size and disk space. Machine availability is determined using availability policies recorded in configuration files. As a LoadLeveler system administrator, you can set these policies for the entire LoadLeveler pool by defining what information is in the *global* configuration file. Individual workstation users define their machines and resource availability policies to LoadLeveler in *local* configuration files. See "Customizing the Configuration File" on page 14 for descriptions of configuration files. Appendix D, "Sample Files" on page 121 shows a sample configuration file.

As the system administrator, you can define the LoadLeveler operating environment in an administration file, which contains machine, user, and job class definitions. See "Customizing the Administration File" on page 33 for a description of the administration file. Appendix D, "Sample Files" on page 121 shows a sample administration file.

How LoadLeveler Works

LoadLeveler processes jobs and monitors the workload by running the following daemons and processes:

LoadL_master referred to as the **master** daemon, this manages all daemons on its resident machine.

LoadL_schedd referred to as the **schedd** daemon, this manages batch submissions on its resident machine.

LoadL_startd referred to as the **startd** daemon, this accepts jobs for dispatch on its resident machine.

LoadL_starter spawned by **startd** daemon, the **starter** process manages a running job on server machine.

LoadL_shadow spawned by the **schedd** daemon, the **shadow** process communicates with the **starter** process to run a job on a server machine.

LoadL_kbdd referred to as the **keyboard** daemon, this monitors keyboard and mouse activity on its resident AIX machine. This daemon is not required on Sun machines.

LoadL_collector referred to as the **collector** daemon, this is the central collector of machine status from all machines in the LoadLeveler pool.

LoadL_negotiator referred to as the **negotiator** daemon, this is the central scheduler and collector of job status from all machines in the LoadLeveler pool.

Some of these components reside on every workstation or host machine in the LoadLeveler pool. Others reside only on the host designated as the central manager.

Central Manager In each LoadLeveler pool there is a machine known as the Central Manager. The Central Manager's principal function is to coordinate LoadLeveler related activities of all the machines in the pool. It maintains the status of all machines and jobs and makes decisions on where jobs should be run. It is important to remember that if the central manager should go down, there is no loss of job information. The jobs that are currently running will continue to run and the jobs that have been submitted but that are not running will be dispatched as soon as the central manager is restarted. In addition, you can continue to submit jobs to LoadLeveler. These jobs will be dispatched when the central manager is restarted.

When the central manager is restarted, it will reconstruct the machine status information from the **startd** daemon and job status information from the **schedd** daemon.

Refer to Appendix E, “LoadLeveler Daemons and Processes” on page 127 for more detailed information about LoadLeveler daemons and processes.

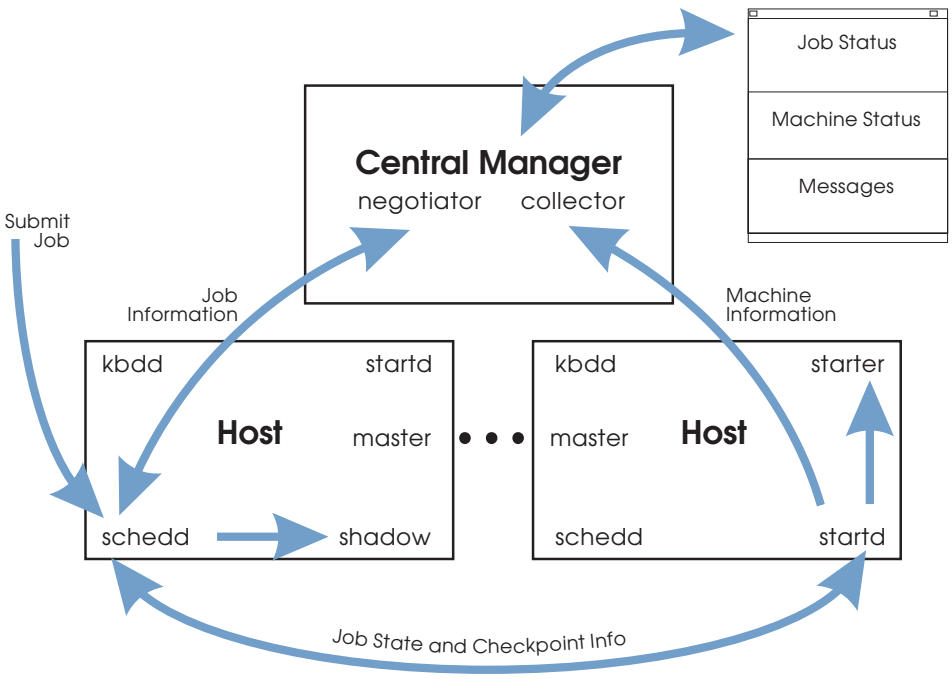
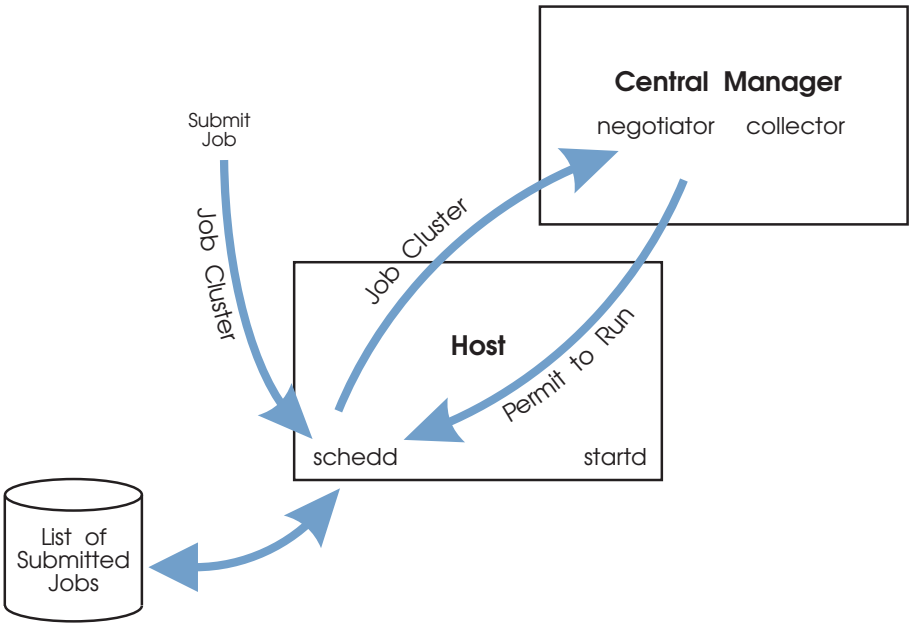


Figure 2. Information Flow through the LoadLeveler Pool

The LoadLeveler Job Cycle

The following sequence of events describes how the LoadLeveler daemons handle a batch job.

1 Submit a LoadLeveler job

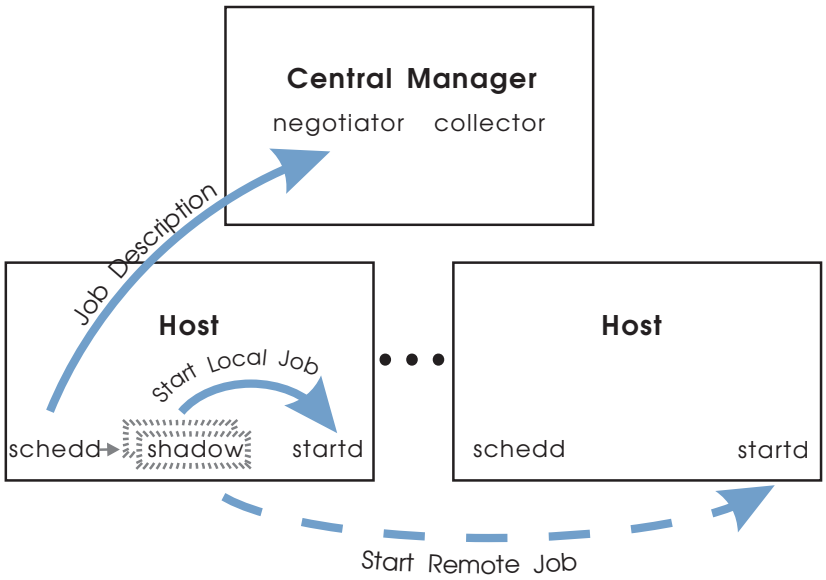


The LoadLeveler **submit** command sends the job to the **schedd** daemon. The **schedd** stores the job on disk and sends job description information to the **negotiator** daemon in the central manager machine.

2 Permit to run

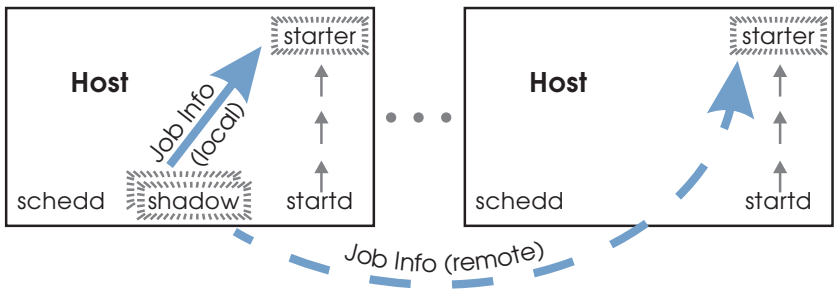
The **negotiator** daemon determines when and where the job can be run, sends the originating **schedd** a *permit to run*, and labels the job as pending. The **negotiator** does not indicate that the job is running on the selected machine until **schedd** responds.

3 Prepare to run



The **schedd** spawns a **shadow** process to handle the job. The **shadow** contacts the **startd** daemon on the target machine and requests that it start the job. **startd** creates two new ports for the job, one for logging and one for a checkpoint file. The **shadow** connects to both ports.

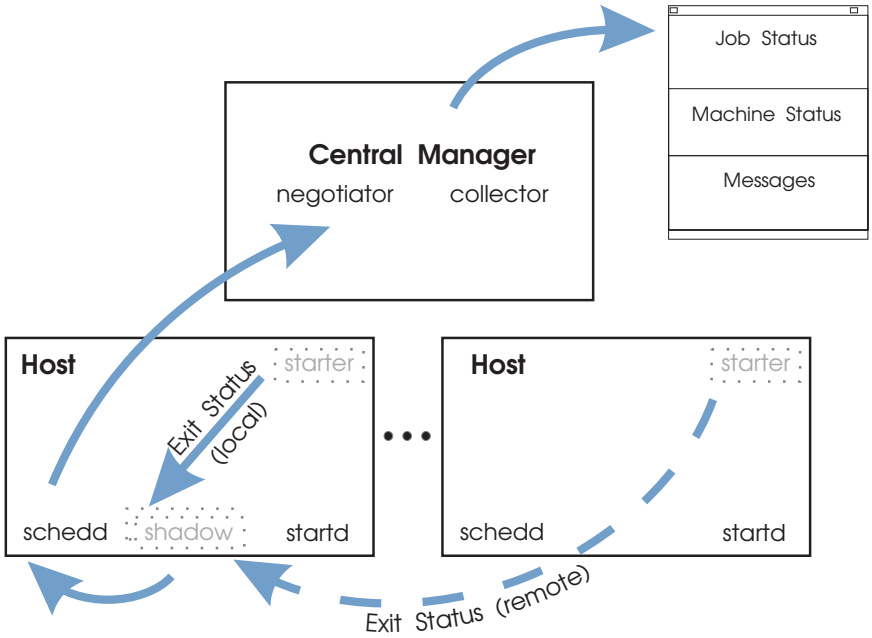
4 Initiate job



The **startd** spawns a **starter** process, closes the original socket, and awaits more work. The **shadow** sends the **starter** the job information and the executable. Then, the **shadow** notifies the **negotiator** that the job has been started and the **negotiator** marks the job as running. The **shadow** awaits termination of the job, listening on the two job sockets.

The **starter** forks itself and the parent awaits termination of the child, keeping its connection to the two job sockets. The **starter** child verifies permission to access the I/O files and execs itself into the job. Anything that comes across the logging socket is written to the **SHADOW_LOG**.

5 Complete job



The **startd** detects termination of its **starter** and updates its own status. When the job terminates, the **starter** parent returns exit status of the job to the **shadow**, which examines it and generates mail to the user. The **shadow** exits, sending status to the **schedd**, which, in turn, examines it and forwards it to the **negotiator**.

Chapter 2. Configuring and Administering LoadLeveler

Two of your main tasks as a system administrator are to configure and administer LoadLeveler. Configuring LoadLeveler involves modifying the global configuration file that specifies the terms (hours, job types, etc.) under which LoadLeveler can use machines for host services. This global configuration file is called **LoadL_config**. Users can have local configuration files on their personal machines to specify individual requirements as well. This local configuration file is called **LoadL_config.local**. The **LoadL_config.local** file is in the same format as **LoadL_config** and the information in this file overrides any information specified in **LoadL_config**.

Important

The **LoadL_config** must exist in the LoadLeveler home directory of the user **loadl**, either in actuality or symbolically linked. The **LoadL_config** file should contain configuration information common to all nodes in the LoadLeveler pool. "Customizing the Configuration File" on page 14 describes how to tailor this file to suit your needs.

The **LoadL_config.local** file is optional. It should be used to modify information on a local machine. Its full pathname is specified in the LoadL_config file by using the **LOCAL_CONFIG** keyword. See "Defining Paths" on page 22 for more information. "Customizing the Configuration File" on page 14 describes how to tailor this file to suit your needs. Appendix D, "Sample Files" on page 121 contains a sample configuration file.

Administering LoadLeveler involves modifying the administration file. This file is called **LoadL_admin**. This file contains user, class, and machine type stanzas to control user access to machines in the LoadLeveler pool and specifies where to run the central manager. The format of these stanzas is shown in "Customizing the Administration File" on page 33. LoadLeveler will not run unless the full pathname of the administration file is specified in the configuration file using the **ADMIN_FILE** keyword. See "Defining Paths" on page 22 for more information.

LoadLeveler assumes a common name space. This means that if there are multiple machines in a cluster, a user must have the same userid and login group id on both machines. For example, if there are two machines in your LoadLeveler cluster, MACH-1 and MACH-2. User john must effectively have the same userid and login group id in the **/etc/passwd** file on both machines. If user john has userid 1234 and login group id 100 on MACH-1, then user john must effectively have the same userid and login group id in **/etc/passwd** on MACH-2. This ensures that the **getuid** system call returns the same userid on both systems. This common name space is important because it ensures that the job that gets submitted gets run with the same groupid and userid of the person who submitted the job.

Where to Begin?

Setting up LoadLeveler should be a simple matter of defining machines, users, and the way in which they interact to process jobs quickly and efficiently. However, those with more experience in system administration and job scheduling will perform these administration tasks with greater ease and more confidence than those who may be new to such tasks. Therefore, three entry points are suggested for various levels of experience.

No matter what your level of experience, it will prove worthwhile to read all the information in this chapter at some time. It will help you to optimize LoadLeveler's performance regardless of the size of your network.

Expert: If you are very familiar with UNIX system administration and job scheduling, and have some idea how you want to distribute your workload, go to "Quick Set Up". Each step in this short procedure refers you to a detailed discussion of the task at hand. The sample configuration and administration files included in the samples subdirectory (and shown in Appendix D, "Sample Files" on page 121) will also provide valuable assistance.

Intermediate: If you are experienced in UNIX system administration but are unfamiliar with job scheduling systems, go to "Customizing the Configuration File" on page 14. In addition to describing the files and controls, this section provides background information that will help you to decide how to set up LoadLeveler.

Beginner: If your experience with UNIX system administration is limited, start at "Configuration File Structure and Syntax" on page 11. This section describes the coding conventions used in the LoadLeveler configuration and administration files.

Quick Set Up

If you are very familiar with UNIX system administration, and job scheduling, and don't want to read through this whole chapter but prefer to start up with only the essentials, perform the tasks listed here to get LoadLeveler up and running on your network quickly in a default configuration. Afterward, you can fine tune your configuration for greater efficiency when you become more familiar with the details of LoadLeveler.

The sample configuration and administration files included in the samples subdirectory (and shown in Appendix D, "Sample Files" on page 121) will also provide valuable assistance.

Step 1. Ensure that the installation procedure has completed successfully and that the configuration file, **LoadL_config**, exists in LoadLeveler's home directory. Refer to Appendix A, "Planning for Installation" on page 99

Step 2. Identify yourself as the LoadLeveler administrator with the **LOADL_ADMIN** macro in the **LoadL_config** file. Refer to "Defining LoadLeveler Administration" on page 19.

Step 3. Set LoadLeveler to accept all jobs by coding the following control statements in your (**LoadL_config**) file:

```
START: T
SUSPEND: F
```

With these settings, the CONTINUE, VACATE, and KILL control statements have no effect. See “Defining Job Management” on page 14 for more information on these control statements.

Step 4. Code these macros in your (**LoadL_config**) file as follows:

```
POLLS_PER_UPDATE = 6
SCHEDD_INTERVAL = 3
CLIENT_TIMEOUT = 3
NEGOTIATOR_INTERVAL = 3
MACHINE_UPDATE_INTERVAL = 3
```

These are baseline settings that provide a test environment which you can later adjust for more efficient operation. These are explained in “Defining Process Specific Keywords” on page 27.

Step 5. Ensure that a central manager has been defined in the administration file. If not, define a host machine to act as the LoadLeveler central manager by coding one machine stanza as follows in the **LoadL_admin** file. (Replace *hostname* with the actual name of the machine.)

```
hostname: type = machine    central_manager = true
```

See “Machine Stanzas” on page 39 for more information.

Step 6. Issue the following command for each machine to be included in the LoadLeveler resource pool. (Replace *hostname* with the actual name of the machine.)

Note: **llctl** uses rsh to start LoadLeveler on the target machine. Therefore, the administrator using **llctl** must have rsh authority on the target machine.

```
llctl -h hostname start
```

See “llctl - LoadLeveler Control” on page 71 for more information.

Configuration File Structure and Syntax

The information in both the **LoadL_config** and the **LoadL_config.local** files can either be in the form of a macro or in the form of a control function: **LoadL_config** and **LoadL_config.local**,

macros define constant string values to use throughout execution of the LoadLeveler program.

control functions define string values or arithmetic or Boolean functions to evaluate dynamically at run time.

Macros

Macro definitions take this form:

Keyword = *value*

Where *value* defines conditions, names files, or sets numeric values. For some macros, a predefined range of choices is shown. Default values are highlighted.

For example, a macro named **MYHOST** could define a host machine named *iron* as follows:

```
MYHOST = iron.ore.met.com
```

Macro keywords are identified by a dollar sign ($) and enclosed in parenthesis. Any use of the literal **$(MYHOST)** following the definition in the example above would result in the automatic substitution of **iron.ore.met.com** in place of **$(MYHOST)**.

Macro definitions may contain references, enclosed in parentheses, to previously defined macros, thus,

```
A = xxx

C = $(A)
```

is a legal set of macro definitions, and the resulting value of **C** is *xxx*. Note that **C** is actually bound to **$(A)**, not to its value, so that

```
A = xxx

C = $(A)

A = yyy
```

is also a legal set of macro definitions and the resulting value of **C** is *yyy*.

Control Functions

Function definitions take this form:

Keyword : *expression*

where expressions may contain constants, operators, and previously defined function names. Macros may also be used in expressions.

Functions take on the type of the expression to which they are bound.

Constants may be represented in the following values:

Boolean expressions that are evaluated as true or false
Signed integers
Floating point values
Strings enclosed in double quotes (" ").

Unlike macro keywords, control functions require no special characters (parenthesis or dollar sign) to identify them when invoked. Thus, the **LoadAvg** function is represented as **LoadAvg** when used in an expression.

Table 1 on page 13 lists the C operators supported by LoadLeveler.

Table 1. Control Function Operators (in order of precedence)	
Operator	**Associativity**
!	left to right
* /	left to right
- +	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
||	left to right

Functions that contain references to other functions are bound to that function's definition, not its current value, for example,

```
X : "xxx"

Y : "X"

X : "yyy"
```

results in **Y** being evaluated as *yyy* at run time.

Functions that contain macro invocations are bound to the current value of the macro, for example,

```
X = "xxx"

Y : "$(X)"

X = "yyy"
```

results in **Y** having a run time value of *xxx*:

LoadLeveler macros and control functions can be defined in the LoadLeveler configuration file and used to define the following types of keywords:

Administration Keywords, described in 19

Job Management Keywords, described in 14

Machine Description Keywords, described in 20

Pathname Keywords, described in 22

Process-Specific Keywords, described in 27.

Some of the macros are required by LoadLeveler. These macros are labelled *required* and must be defined. All other macros are optional and can be left to their default values.

Customizing the Configuration File

You can customize the global configuration file, called **LoadL_config**, to specify what information will be common across all nodes in the LoadLeveler pool.

Defining Job Management

The policies for running, suspending, and vacating LoadLeveler jobs on a machine are defined though a series of five key control functions that respond to environmental conditions you describe in the configuration file. These conditions are defined, following normal C conventions, in the **START**, **SUSPEND**, **CONTINUE**, **VACATE**, and **KILL** control functions.

The **START** control function is evaluated before a job is running. The remainder control functions are evaluated for jobs queued on a machine that are not necessarily running.

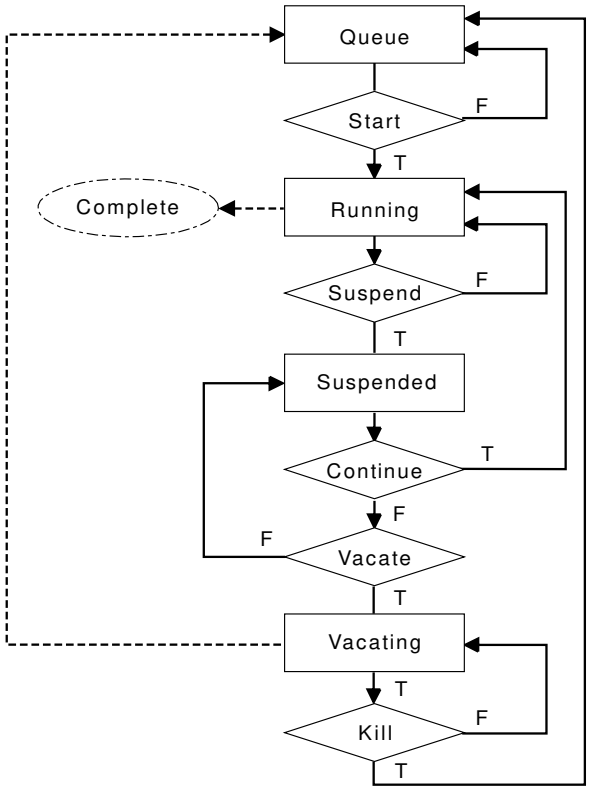


Figure 3. LoadLeveler Job Management through Five Key Control Functions

START: *expression that evaluates to true or false (T/F)*

Determines whether a machine is capable of running a LoadLeveler job. When **true**, LoadLeveler considers dispatching a job to the machine.

SUSPEND: *expression that evaluates to true or false (T/F)*

Determines whether running jobs should be suspended. When **true**,

LoadLeveler temporarily suspends jobs currently running on the machine. Suspended LoadLeveler jobs will either be continued or vacated.

CONTINUE: *expression that evaluates to true or false (T/F)*

Determines whether suspended jobs should continue execution. When **true**, suspended LoadLeveler jobs resume execution on the machine.

VACATE: *expression that evaluates to true or false (T/F)*

Determines whether suspended jobs should be vacated. When **true**, suspended LoadLeveler jobs are removed from the machine. If the jobs are checkpointable, LoadLeveler sends the latest checkpoint file back to the scheduling machine.

KILL: *expression that evaluates to true or false (T/F)*

Determines whether or not vacated jobs should be killed and replaced in the queue. It is used to remove a job that is taking too long to vacate. When **true**, vacated LoadLeveler jobs are removed from the machine with no attempt to take checkpoints.

Typically, machine load average, keyboard activity, time intervals, job class, and parallel machines, are used within these various expressions to dynamically control job execution.

Examine the examples of the various configuration files in the samples directory within the release directory. Samples are shown in Appendix D, “Sample Files” on page 121.

Predefined Macros

To write useful control functions, it is often necessary to access dynamic values which are only available at run time. LoadLeveler provides the following predefined macros and built-in variables for writing control functions in expressions:

Cpus

the number of CPU's installed. There is usually one CPU unless you have a multiprocessor.

CurrentTime

the **UNIX date**; the current system time, in seconds, since January 1, 1970, as returned by the time() function.

Disk

the free disk in megabytes on the filesystem where foreign checkpoints are stored.

domain or **domainname**

dynamically indicates the official name of the domain of the current host

machine where the program is running. Whenever a machine name can be specified or one is assumed, a domain name is assigned if none is present. For example, if a job is submitted to host machine **silver** and the domain of the submitting machine is **ore.met.com**, then **$(domain)** would have the value **ore.met.com**.

EnteredCurrentState

the value of **CurrentTime** when the current state (START, SUSPEND, etc) was entered.

host or **hostname**

dynamically indicates the official name of the host machine where the program is running. For example, if a job is submitted to host machine **gold**, then **$(host)** would have the value **gold**.

Initiator

the job initiators both available and in use.

KeyboardIdle

the number of seconds since the keyboard or mouse was last used. It also includes any telnet or interactive activity from any remote machine.

LoadAvg

the Berkeley one-minute load average.

Machine

the name of the current machine.

Memory

the physical memory installed on the machine in megabytes.

Qdate

the UNIX date the job entered the queue.

State

the state of the **startd**. Can be "None", "Busy", "Running", "Idle", "Suspend", "Flush", or "Drain".

Status

the status of the job. This is an integer code corresponding to Idle (I), Running (R), Completed (C), Removed (X), Pending (P), System_hold (S), User_hold (U), or Deferred (D).

tilde

the home directory for loadl. Dynamically defines the pathname of the LoadLeveler home directory.

User_Priority

the user defined priority of the job. The priority ranges from 0 to 100, with higher numbers corresponding to greater priority.

Time:

Note: You can use LoadLeveler across multiple time zones but unexpected results may occur.

tm_hour

the number of hours since midnight (0-23).

tm_min

number of minutes after the hour (0-59).

tm_sec

number of seconds after the minute (0-59).

tm_isdst

Daylight Savings Time flag: positive when in effect, zero when not in effect, negative when information is unavailable.

Date

tm_mday

the number of the day of the month (1-31).

tm_wday

number of days since Sunday (0-6).

tm_yday

number of days since January 1 (0-365).

tm_mon

number of months since January (0-11).

tm_year

the number of years since 1900 (0-99).

Creating New Variables

You can create your own variables using a combination of built-in variables and constants. The following examples can be found in the sample configuration file.

MINUTE = 60

defines the variable **MINUTE** and assigns to it the constant integer value 60.

HOUR = (60 * $(MINUTE))

defines the variable **HOUR** and assigns to it the constant integer value 360.

StateTimer = (CurrentTime - EnteredCurrentState)

defines the variable **StateTimer** and reassigns to it at each evaluation the number of seconds since the current state was entered.

BackgroundLoad = 0.3

defines the variable **BackgroundLoad** and assigns to it the floating point constant 0.3. This might be used as a noise factor indicating no activity.

HighLoad = 1.5

defines the variable **HighLoad** and assigns to it the floating point constant 1.5. This might be used as a saturation level at which no further jobs should be started.

KeyboardBusy = KeyboardIdle < (15 * $(MINUTE))

defines the variable **KeyboardBusy** and reassigns to it at each evaluation the Boolean value True or False, depending on whether the keyboard and mouse have been idle for fifteen minutes.

CPU_Idle = LoadAvg <= $(BackgroundLoad)

defines the variable **CPU_Idle** and reassigns to it at each evaluation the Boolean value True or False, depending on whether the Berkeley one-minute load average is equal or less than 0.3.

CPU_Busy = LoadAvg >= $(HighLoad)

defines the variable **CPU_Busy** and reassigns to it at each evaluation the Boolean value True or False, depending on whether the Berkeley one-minute load average is equal to or greater than the saturation level of 1.5.

See the configuration files in Appendix D, “Sample Files” on page 121 for additional examples.

Defining LoadLeveler Administration

Use this macro to define LoadLeveler system administrators:

LOADL_ADMIN = *user name(s) (required)*

where *user name(s)* is the name(s) of those individuals who will have administrative authority. These individuals will receive mail describing problems that are encountered by the **LoadL_master** daemon. These users are also able to invoke the administrator-only commands **llctl**, **llfavorjob**, and **llfavoruser**. They can also invoke the administrator-only graphical user interface functions.

Defining Job Accounting

Use this macro to turn accounting on and off:

ACCT = *flag (optional)*

The available flags are:

A_ON	Turns accounting data recording on. This is the default.
A_OFF	Turns accounting data recording off.
A_VALIDATE	Turns accounting validation on.

ACCT_VALIDATION = *$(RELEASEDIR)/llacctval (optional)*

Keyword used to identify the module that will be called to perform account validation. An installation may choose to replace the llacctval module with its own accounting routine by specifying the module in this keyword.

GLOBAL_HISTORY = *$(SPOOL) (optional)*
Keyword used to identify the directory that will contain the global history files produced by llacctmrg command when no directory is specified as a command argument.

Defining LoadLeveler Machine Operation

To define how machines in the LoadLeveler pool will be used, you need to specify machine description macros and machine description control functions. The machine description macros follow:

Machine Description Macros

Use these macros to tell LoadLeveler how the machines in the pool are to be used:

ARCH = *string (required)*

Indicates the standard architecture of the system. This definition must be consistent among machines of the same type and must be executed the same in the requirements or preferences statements in job command files. The administrator defines the character string for each architecture.

Class = { *class set***}| {"No_Class"}**

Specifies what class of jobs may be run on the machine. See “Defining Job Class” on page 24 for more information.

DEFAULT_JOB_CLASS = *class name*|**No_Class** *(required)*
Specifies the default job class when none is found in either (1) the job command file or (2) the administration file, in that search order. See “Defining Job Class” on page 24 for more information.

POOL_ID = *number (optional)*

Number identifies the batch sub-pool as defined in the **/etc/SP1/node_types** configuration file of the SP1.

MACHINE_AUTHENTICATION = true| **false**

Specifies whether machine validation is performed. When set to **true**, LoadLeveler only accepts connections from machines specified in the administration file. When set to **false**, LoadLeveler accepts connections from any machine.

MAX_STARTERS = *number (required)*

Specifies the maximum number of jobs that can run simultaneously. If not specified, the default is the number of elements in class set. In the parallel context, MAX_STARTERS is the number of jobs that can be initiated from a **startd**. See "Limiting Simultaneous Jobs" on page 25 for more information.

RESERVED_SWAP = *number*
SHADOW_SIZE_ESTIMATE = *number*

Each job that LoadLeveler executes remotely has an associated shadow process running on the machine to which it is submitted. LoadLeveler attempts to limit the number of jobs run concurrently to keep these shadow processes from exhausting all the swap space on the machine the job is submitted to.

In addition it will also try to leave the amount specified by the **RESERVED_SWAP** (in megabytes) keyword free for any non-LoadLeveler jobs that may try to start. To facilitate this, LoadLeveler uses the value specified by the **SHADOW_SIZE_ESTIMATE** (in kilobytes) keyword as an estimate of the swap space consumed by a single shadow process. LoadLeveler refines this estimate at run time if necessary.

START_DAEMONS = true| false

One example of why you might want to set this macro to **false** is if you wanted to designate the node to be used exclusively for interactive work or if you wanted to remove it from participating in the cluster.

X_RUNS_HERE = true|false

If you are not using a Sun machine, set **X_RUNS_HERE** to **true** if you are running X windows. This will start up the keyboard daemon. Set **X_RUNS_HERE** to false if you not running X windows.

If you are using a Sun machine, ignore this keyword. There will be no keyboard daemon running on this platform. Instead, X events are monitored by the **startd** polling /**dev**/**kbd** and /**dev**/**mouse**.

SCHEDD_RUNS_HERE = true| false
STARTD_RUNS_HERE = true| false

These macros specify which daemons run on the host. If you do not want to run any daemons (machine is *not* to be used as a LoadLeveler host), then you must code the **START_DAEMONS** keyword **false**. If you do not want to run the **SCHEDD**, **STARTD**, or **KBDD** daemons, then you must specify **false** with the appropriate keyword. **X_RUNS_HERE** should be set to **true** only on UNIX systems running X.

USE_X_IDLE_EXTENSION = true| **false**

If your X server supports the XGetIdleTime protocol extension, set this keyword to **true**.

If you are using a Sun machine, ignore this keyword.

Machine Description Control Functions

OpSys : $(OPSYS)

indicates the operating system on the host where the program is running. This value is automatically determined and need not be defined in the configuration file.

Arch : $(ARCH)

indicates the standard architecture system.

Machine : $(hostname)

indicates the name of the current host. See "Defining Job Management" on page 14 for information about the **hostname** macro.

String-valued functions are provided for programs that may want to utilize these control functions to evaluate them at run time.

Defining Paths

Use these macros to tell LoadLeveler where files and directories are located.

Administration File

ADMIN_FILE = *pathname (required)*

points to the administration file containing user, class, and machine list stanzas. For example,

```
ADMIN_FILE = $(tilde)/admin.list
```

Refer to "Customizing the Administration File" on page 33 for information.

Local Configuration File

LOCAL_CONFIG = *pathname*

defines the pathname of the optional (but recommended) local configuration file containing information specific to a node in the LoadLeveler network. If you are using a distributed filel system like NFS or AFS, two examples are:

```
LOCAL_CONFIG = $(tilde)/$(hostname).LoadL_config.local
LOCAL_CONFIG = $(tilde)/LoadL_config.$(hostname).$(domain)
```

If you are using a local file system, an example is:

```
LOCAL_CONFIG = /var/LoadL/LoadL_config.local
```

See "Predefined Macros" on page 15 for information about the **tilde**, **hostname**, and **domain** macros.

Local Directory

The following subdirectories reside in the local directory. It is possible that the local directory and the loadl's home directory are the same.

EXECUTE = *local directory* /**execute** *(required)*

defines the local directory to store the executable checkpoints of jobs submitted by other machines.

LOG = *local directory* /**log** *(required)*

defines the local directory to store log files. It is not necessary to keep all the log files created by the various LoadLeveler daemons and programs in one directory but you will probably find it convenient.

SPOOL = *local directory* /**spool** *(required)*

Defines the local directory where LoadLeveler keeps the local job queue and checkpoint files, as well as:

HISTORY = $(SPOOL)/history *(required)*

defines the pathname where a file containing the history of local LoadLeveler jobs is kept.

Release Directory

RELEASEDIR = *release directory (required)*

defines the directory where all the LoadLeveler software resides. Most installations use /**usr**/**local**/**loadl** or the home directory, /**u**/**loadl**.

The following subdirectories reside in the release directory:

BIN = $(RELEASEDIR)/bin *(required)*

defines the directory where LoadLeveler binaries are kept.

LIB = $(RELEASEDIR)/lib *(required)*

defines the directory where LoadLeveler libraries are kept.

NQS_DIR = *NQS directory (optional)*

defines the directory where NQS commands **qsub**, **qstat**, and **qdel** reside. The default is /**usr**/**bin**.

Specifying Where LoadLeveler Jobs Run

LoadLeveler allows you to define how machines are to be used for host services. Use the **Class** keyword to set the number of different job classes. Use the **MAX_STARTERS** keyword to limit the number of simultaneous jobs for the entire LoadLeveler system or for individual machines.

Defining Job Class

The local configuration file provided with LoadLeveler sets the class keyword to **No_Class**. This means that, by default, each machine is initially configured to run one job of a class defined as **No_Class**. Similarly by default, each LoadLeveler job has its class defined as **No_Class**. In order for a LoadLeveler job to run on a machine, the machine must have a vacancy for the class of that job. If the machine is configured for only one **No_Class** job and a LoadLeveler job is already running there, then no further LoadLeveler jobs are started on that machine until the current job completes.

Machines are permitted multiple classes. These classes are specified in the configuration file (either the LoadL_config.local or LoadL_config file).

The following examples show how to define job classes for machines:

Example 1

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

This is the default. The machine will only run LoadLeveler jobs that have either defaulted to, or explicitly requested class **No_Class**. A LoadLeveler job with class **CPU_bound**, for example, would not be eligible to run here. Only one LoadLeveler job at a time will run on the machine. See “Limiting Simultaneous Jobs” on page 25.

Example 2

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

The machine will only run jobs that have either defaulted to, or explicitly requested class **No_Class**. A maximum of two LoadLeveler jobs are permitted to run simultaneously on the machine. See “Limiting Simultaneous Jobs” on page 25.

Example 3

```
Class = { "No_Class" "Small" "Medium" "Large" }
```

The machine will only run a maximum of four LoadLeveler jobs that have either defaulted to, or explicitly requested **No_Class**, **Small**, **Medium**, or **Large** class. A LoadLeveler job with class **IO_bound**, for example, would not be eligible to run here. See “Limiting Simultaneous Jobs” on page 25.

Example 4

```
Class = { "B" "B" "D" }
```

The machine will run only LoadLeveler jobs that have explicitly requested class **B** or **D**. Up to three LoadLeveler jobs may run simultaneously: two of class **B** and one of class **D**. A LoadLeveler job with class **No_Class**, for example, would not be eligible to run here. See “Limiting Simultaneous Jobs” on page 25.

Designating Machine Features

You can specify unique characteristics for any machine using the *feature* keyword. When evaluating job submissions, LoadLeveler compares any required features specified in the job command file to those specified using this keyword.

FEATURE = { " *string***" }**

any characteristics to be used to match jobs with machines.

Limiting Simultaneous Jobs

MAX_STARTERS, specified in the configuration file (either **LoadL_config** or **LoadL_config.local**) sets the maximum number of jobs that can run. It is equal to the minimum of the MAX_STARTERS and the number of classes in the class statement. If no value is set for MAX_STARTERS, the default is the number of classes in the class statement.

For example, if the LoadL_config or the LoadL_config.local file contained these statements:

```
Class = { "A" "B" "B" "C" }
MAX_STARTERS = 2
```

the machine would run a maximum of two LoadLeveler jobs simultaneously. In the example above, the possible combinations of LoadLeveler jobs are:

A and B
A and C
B and B
B and C
A, B, or C, running alone.

Prioritizing the LoadLeveler Job Queue

Each job submitted to LoadLeveler is assigned a system priority number, based on the evaluation of the **SYSPRIO** keyword expression in the configuration file. The LoadLeveler system priority number is assigned when the central manager adds the new job to the queue of jobs eligible for dispatch. Once assigned, the LoadLeveler system priority number for a job is never changed. Jobs assigned higher **SYSPRIO** numbers are considered for dispatch before jobs with lower numbers.

You can use the **Qdate**, **ClassSysprio**, and **UserSysprio** keywords to define the **SYSPRIO** keyword expression:

Qdate same as **CurrentTime**, the **UNIX date**; the current system time, in seconds, since January 1, 1970, as returned by the time() function.

is the UNIX date and time of job submission.

UserSysprio is the priority of the user who submitted the job, as defined in a user stanza in the administration file by the administrator.

ClassSysprio is the priority for the class of the job, defined in a class stanza in the administration file by the administrator. The default is 0.

If **ClassSysprio** and **UserSysprio** are not to be used in the **SYSPRIO** expression, then there is no need to define them in the administration file.

The following example creates a FIFO job queue:

```
SYSPRIO :   - (Qdate)
```

This example factors in all sources of priority:

```
SYSPRIO : (ClassSysprio   5) + (UserSysprio   2) - (Qdate   1)
```

Recording and Controlling Log Files

The LoadLeveler daemons and processes keep log files according to the specifications in the configuration file. A number of macros are used to describe where the logs are maintained and how much information they record. These macros, shown in Table 2, are repeated in similar form to specify the pathname of the log file, its maximum length, and the debug flags to be used.

"Controlling Debugging Output" on page 27 describes the events that can be reported through logging controls.

Table 2. Log Control Statements

Daemon/ Process	**Log File** *(required)* **(See note 1)**	**Max Length** *(required)* **(See note 2)**	**Debug Control** *(required)* **(See note 3)**
Master	**MASTER_LOG =** *path*	**MAX_MASTER_LOG =** *bytes*	**MASTER_DEBUG =** *flags*
Schedd	**SCHEDD_LOG =** *path*	**MAX_SCHEDD_LOG =** *bytes*	**SCHEDD_DEBUG =** *flags*
Startd	**STARTD_LOG =** *path*	**MAX_STARTD_LOG =** *bytes*	**STARTD_DEBUG =** *flags*
Starter	**SHADOW_LOG =** *path*	(See note 4)	**STARTER_DEBUG =** *flags*
Shadow	**SHADOW_LOG =** *path*	**MAX_SHADOW_LOG =** *bytes*	**SHADOW_DEBUG =** *flags*
Negotiator	**NEGOTIATOR_LOG =** *path*	**MAX_NEGOTIATOR_LOG =** *bytes*	**NEGOTIATOR_DEBUG =** *flags*
Collector	**COLLECTOR_LOG =** *path*	**MAX_COLLECTOR_LOG =** *bytes*	**COLLECTOR_DEBUG =** *flags*
Kbdd	**KBDD_LOG =** *path*	**MAX_KBDD_LOG =** *bytes*	**KBDD_DEBUG =** *flags*

Notes:

1. When coding the *path* for the log files, it is not necessary that all LoadLeveler daemons keep their log files in the same directory, however, you will probably find it a convenient arrangement.
2. There is a maximum length, in bytes, beyond which the various log files cannot grow. Each file is allowed to grow to the specified length and is then saved to an **old file**. The **old** files are overwritten each time the log is saved, thus the maximum space devoted to logging for any one program will be twice the maximum length of its log file. The default length is 64KB.

   You can also specify that the log file be started anew with every invocation of the daemon by setting the **TRUNC** statement to **true** as follows:

   TRUNC_MASTER_LOG_ON_OPEN = true| **false**
   TRUNC_STARTD_LOG_ON_OPEN = true| **false**
   TRUNC_SCHEDD_LOG_ON_OPEN = true| **false**
   TRUNC_KBDD_LOG_ON_OPEN = true| **false**
   TRUNC_SHADOW_LOG_ON_OPEN = true| **false**
   TRUNC_COLLECTOR_LOG_ON_OPEN = true| **false**
   TRUNC_NEGOTIATOR_LOG_ON_OPEN = true| **false**

3. The debug flags, described in “Controlling Debugging Output” on page 27, are normally used only by installers and debuggers. The default error logging, obtained by leaving the right side of the debug control statement null, will be sufficient for most installations.
4. Since data logged by the **starter** process is recorded in the **SHADOW_LOG**, no maximum length keyword exists for the **starter**.

Controlling Debugging Output

Flags control the level of debugging output logged by each program.

The available flags are:

D_TERMLOG	Sends logging information to **stdout** rather than the log file.
D_CKPT	Logs various steps in the checkpointing process.
D_XDR	Logs calls to read and write by the xdr routines.
D_EXPR	Logs steps in parsing and evaluating control expressions.
D_PROC	Logs information about jobs being started remotely.
D_JOB	Logs job requirements and preferences when making decisions regarding whether a particular job should run on a particular machine.
D_MACHINE	Logs machine control functions and variables when making decisions regarding starting, suspending, resuming, and aborting remote jobs.
D_FULLDEBUG	Logs details about most actions performed by each daemon but doesn't log as much activity as setting all the flags.

For example,

```
SCHEDD_DEBUG = D_CKPT    D_XDR
```

causes the scheduler to log information about checkpointing user jobs and exchange xdr messages with other LoadLeveler daemons. These flags will primarily be of interest to LoadLeveler implementers and debuggers.

Defining Process Specific Keywords

You will find that in the sample configuration files provided with LoadLeveler, certain keyword definitions are grouped according to the daemons or processes that use them. Many of these have already been described but are shown again here in the order they appear in the sample configuration file in Appendix D, “Sample Files” on page 121.

Kbdd Daemon Keywords

The following **kbdd** keywords may be specified in your configuration file. Keep in mind that this daemon does not run on Sun machines.

KBDD = *pathname*

location of **kbdd** executable, **LoadL_kbdd**. See “The Kbdd Daemon” on page 132 for more information.

KBDD_LOG = *pathname*

location of log. See “Recording and Controlling Log Files” on page 26.

MAX_KBDD_LOG = *bytes*

limit of log size. See “Recording and Controlling Log Files” on page 26.

KBDD_DEBUG = *flags*

Error logging controls. See “Controlling Debugging Output” on page 27.

X_RUNS_HERE = true|**false**

Specifies whether the **kbdd** daemon runs on host for UNIX systems running X.

USE_X_IDLE_EXTENSION = true| **false**

If your X server supports the XGetIdleTime protocol extension, set this required item to true.

Preen Keywords

PREEN detects unexpected files in the LoadLeveler home subdirectories. This executes daily at 3:30 AM. You can specify the following **PREEN** keyword in your configuration file:

PREEN = *command*

where *command* is the location of the executable and the specified option. For example,

PREEN = /u/var/llpreen -r

See 76 for more information.

Startd Daemon Keywords

The following **startd** keywords may be specified in your configuration file:

STARTD = *pathname*

location of **startd** executable, **LoadL_startd**. See “The Startd Daemon” on page 130 for more information.

STARTD_LOG = *pathname*

location of log. See “Recording and Controlling Log Files” on page 26.

MAX_STARTD_LOG = *bytes*

limit of log size. See “Recording and Controlling Log Files” on page 26.

STARTD_DEBUG = *flags*

Error logging controls. See “Controlling Debugging Output” on page 27.

POLLING_FREQUENCY = *number*

where *number* specifies the frequency, in seconds, with which the **startd** daemon evaluates the load on the local machine and decides whether to suspend, resume, or abort remote jobs by evaluating the START, SUSPEND, CONTINUE, VACATE, and KILL control functions. On machines that run X, this is also the minimum interval at which the **kbdd** daemon will report keyboard or mouse activity to the **startd** daemon. A value of 5 should offer sufficient reporting under most operating conditions.

POLLS_PER_UPDATE = *n*

where *n* specifies how often the **startd** daemon updates the central manager. Due to the communication overhead, it is impractical to do this with the frequency defined by the **POLLING_FREQUENCY** keyword. Therefore, the **startd** daemon only updates the central manager every *n*th local update. Change **POLLS_PER_UPDATE** when changing the **POLLING_FREQUENCY**. A value of 24 should be sufficient.

KBD_DEVICE = *filename*

where *filename* specifies the keyboard device file. This keyword only applies to Sun machines and is used by the **startd** when monitoring X events. The directory /**dev**/ is assumed. The default is **kbd**.

MOUSE_DEVICE = *filename*

where *filename* specifies the mouse device file. This keyword only applies to Sun machines and is used by the **startd** when monitoring X events. The directory /**dev**/ is assumed. The default is **mouse**.

Schedd Daemon Keywords

The following **schedd** daemon keywords may be specified in your configuration file:

SCHEDD = *pathname*

location of **schedd** executable, **LoadL_schedd**. See “The Schedd Daemon” on page 128 for more information.

SCHEDD_LOG = *pathname*

location of log. “Recording and Controlling Log Files” on page 26.

MAX_SCHEDD_LOG = *bytes*

limit of log size. See “Recording and Controlling Log Files” on page 26.

SCHEDD_DEBUG = *flags*

Error logging controls. See “Controlling Debugging Output” on page 27.

SCHEDD_INTERVAL = *number*

where *number* specifies the interval, in seconds, at which the **schedd** daemon checks the local job queue and updates the **collector** daemon. A value of 120 should be sufficient.

Collector Daemon Keywords

The following **collector** daemon keywords may be specified in your configuration file.

COLLECTOR = *pathname*

location of **collector** executable, **LoadL_collector**. See "The Collector Daemon" on page 132.

COLLECTOR_LOG = *pathname*

location of log. See "Recording and Controlling Log Files" on page 26.

MAX_COLLECTOR_LOG = *bytes*

limit of log size. See "Recording and Controlling Log Files" on page 26.

COLLECTOR_DEBUG = *flags*

Error logging controls. See "Controlling Debugging Output" on page 27.

CLIENT_TIMEOUT = *number*

where *number* specifies the maximum time, in seconds, that the **collector** awaits a response from a process on a TCP/IP connection before assuming the host machine is down. A value of 30 should be sufficient.

MACHINE_UPDATE_INTERVAL = *number*

where *number* specifies the time period, in seconds, during which machines must report to the central manager. Machines that do not report in this number of seconds are considered *down*. This number should be larger than the POLLING_FREQUENCY * POLLS_PER_UPDATE and larger than the **schedd** interval. A value of 150 should be sufficient.

Negotiator Daemon Keywords

The following **negotiator** daemon keywords may be specified in your configuration file.

NEGOTIATOR = *pathname*

location of the **negotiator** executable, **LoadL_negotiator**. See"The Negotiator Daemon" on page 132 .

NEGOTIATOR_LOG = *pathname*

location of the log. See "Recording and Controlling Log Files" on page 26.

MAX_NEGOTIATOR_LOG = *bytes*

limit of log size. See “Recording and Controlling Log Files” on page 26.

NEGOTIATOR_DEBUG = *flags*

Error logging controls. See “Controlling Debugging Output” on page 27.

NEGOTIATOR_INTERVAL = *number*

where *number* specifies the interval, in seconds, at which the **negotiator** daemon updates the status of jobs in the LoadLeveler pool and negotiates with machines that are available to run jobs.

MAX_JOBS_RUNNING = *number*

where *number* specifies the maximum number of jobs a single user can run simultaneously on all machines in the LoadLeveler pool. This value is superseded by the limit set in the administration file, which serves as a default. A value of 15 should be sufficient for a small configuration, but this depends on the number of users and the types of jobs they run. See “User Stanzas” on page 34.

Shadow Process Keywords

The following **shadow** process keywords may be specified in your configuration file.

SHADOW = *pathname*

location of the **shadow** executable, **LoadL_shadow**. See “The Shadow Process” on page 129 for more information.

SHADOW_LOG = *pathname*

location of log. See “Recording and Controlling Log Files” on page 26.

MAX_SHADOW_LOG = *bytes*

limit of log size. See “Recording and Controlling Log Files” on page 26.

SHADOW_DEBUG = *flags*

Error logging controls. See “Controlling Debugging Output” on page 27.

SHADOW_LOCK = *pathname*

where *pathname* identifies the shadow lock file used to prevent loggers from multiple shadows from writing over each other's output. It is recommended that **SHADOW_LOCK** be located in a local file system of a machine.

Starter Process Keywords

The following **starter** process keywords may be specified in your configuration file.

STARTER = *pathname*

location of **starter** executable, **LoadL_starter**. See “The Starter Process” on page 131 for more information.

STARTER_DEBUG = *flags*

Error logging controls. See “Controlling Debugging Output” on page 27.

MIN_CKPT_INTERVAL = *number*
MAX_CKPT_INTERVAL = *number*

where *number* specifies a period, in seconds, between checkpoints taken for running jobs. The time between checkpoints will be increased after each checkpoint within these limits as follows: the first checkpoint is taken after a period of time equal to the **MIN_CKPT_INTERVAL** has passed. LoadLeveler waits *twice as long* (**MIN_CKPT_INTERVAL** X 2) before taking the second checkpoint, and *twice again* as long (**MIN_CKPT_INTERVAL** X 4) before taking the third checkpoint. LoadLeveler continues to double this period until the the value of **MAX_CKPT_INTERVAL** has been reached, where it stays for the remainder of the job.

A minimum value of 900 (15 minutes) and a maximum value of 7200 (2 hours) are the defaults.

Master Daemon Keywords

The following **master daemon** keywords may be specified in your configuration file.

MASTER = *pathname*

location of **master** executable, **LoadL_master**. See “The Master Daemon” on page 127 for more information.

MASTER_LOG = *pathname*

location of log. See “Recording and Controlling Log Files” on page 26.

MAX_MASTER_LOG = *bytes*

limit of log size. See “Recording and Controlling Log Files” on page 26.

MASTER_DEBUG = *flags*

Error logging controls. See “Controlling Debugging Output” on page 27

PUBLISH_OBITUARIES = true| false

where **true** specifies that the **master** daemon send mail to the administrator(s) identified by **LOADL_ADMIN** keyword, when any of the daemons it manages dies abnormally.

OBITUARY_LOG_LENGTH = *number*

where *number* specifies the number of lines from the *tail* to be appended to the log that the central manager mails to **LOADL_ADMIN** when one of the daemons dies. A value of 25 should be sufficient.

RESTARTS_PER_HOUR = *number*

where *number* specifies how many times the **master** daemon attempts to restart a daemon that dies abnormally. Because one or more of the daemons may be unable to run due to a permanent error, the **master** only attempts $(RESTARTS_PER_HOUR) restarts within a 60 minute period. Failing that, it sends mail to the administrator(s) identified by the **LOADL_ADMIN** keyword and exits.

Customizing the Administration File

Any hostname or machine name specified in the administration file must be a name which can be resolved to an address. This name is referred to as an interface name because the name can be used for a program to interface with the machine. Generally, interface names match the hostname but they do not have to.

The only requirement for the administration file is that you specify a machine stanza identifying the central manager. For example:

```
hostname: type = machine    central manager = true
```

where *hostname* is the name of the machine.

Entries in the administration file are made in a stanza format, consisting of a control function followed by macro definitions:

*label***: type =** *type*

keyword1 = *value1*
keyword2 = *value2*
keyword3 = *value3*.

where *label* is a machine name, a class name, or a user name, depending on the *type* of stanza. There are three different type stanzas in the administration file. Each type, with its associated *keywords*, is described on the following pages:

type = user (See “User Stanzas” on page 34)

type = class (See “Class Stanzas” on page 35)

type = machine (See “Machine Stanzas” on page 39)

Administration File Structure and Syntax

Generally, stanzas are constructed as follows:

Each stanza is identified by its label. The label specifies the name of the user, class, or machine described in the stanza.
Each stanza must have a *type* field that specifies it as a user, class, or machine stanza.

Multiple stanzas of the same label are allowed, but not merged. Multiple stanzas with the same label and type, however, are not recommended except when they are labelled **default**. A **default** stanza is replaced by a subsequent **default** stanza of the same type.

Default stanzas specify the default values to be used for stanzas of the same type that do not have all fields specified. Any value may be defaulted. You can specify a separate default stanza for LoadLeveler to maintain for each of the three stanza types.

Default stanzas are not directly added to any internal structures but are used in the building of internal structures whenever incomplete stanzas are encountered at configuration time.

New lines are ignored, which means that the various components of a stanza may cross line boundaries or be included on the same line.

White space is ignored, other than to delimit keyword identifiers. This eliminates confusion between tabs and spaces at the beginning of lines.

A pound sign (#) is used to identify a comment and may appear anywhere on the line. All characters following the pound sign on that line are ignored.

User Stanzas

The information specified in a user stanza defines the characteristics of that user. User stanzas are not required.

User stanzas are specified as follows:

user

is the name of the user.

type = user

where **user** identifies the stanza as user information.

priority = *number*

where *number* can be any integer from 0 to 100, inclusive. If not specified, the default is zero (0). The number specified for priority is referenced as **UserSysprio** in the configuration file. **UserSysprio** can be used in the assignment of job priorities. If the variable **UserSysprio** does not appear in the SYSPRIO expression in the configuration file, the priority numbers for users specified here in the administration file have no effect. See “Prioritizing the LoadLeveler Job Queue” on page 25 for more information about the **UserSysprio** keyword.

default_class = *class name*

where *class name* is the default value assigned to jobs submitted by the user for which no **class** statement appears in the LoadLeveler job stream. If no **default_class** is specified in the user stanza, or if there is no user stanza at all, then jobs submitted by the user without a **class** statement are assigned to the **default_job_class** that appears in the configuration file. Additionally, if no **default_job_class** appears in the configuration file, then an error occurs when a user attempts to submit a job.

maxjobs = *number*

where *number* is a the maximum number of jobs a single user can run at any time. This value prevents the user from flooding the LoadLeveler run queue. If no **maxjobs** is specified in the user stanza, or if there is no user stanza at all, the maximum jobs that can be simultaneously run by the user is defined in the **MAX_JOBS_RUNNING** value in the configuration file. If no value is found, or the limit found is zero (0), then no limit is placed on the number of jobs that can simultaneously run for the user. Regardless of the limit set to running jobs, there is no limit to the number of jobs a user can submit.

account =*list*

where *list* is a blank delimited list of account numbers that identifies the account numbers a user may use when submitting jobs.

Here is an example of a user stanza:

```
# Define user stanzas
fred:    type = user
   priority = 1
   default_class = class_a
   maxjobs = 15
```

Class Stanzas

The information in a class stanza defines characteristics for that class. Class stanzas are not required.

Class stanzas are specified as follows:

class

is the name of the job class.

type = class

identifies the stanza as class information.

priority = *number*

where *number* can be any integer from 0 to 100, inclusive. If not specified, the default is zero (0). The number specified for priority is referenced as **ClassSysprio** in the configuration file. **ClassSysprio** can be used in the assignment of job priorities. If the variable **ClassSysprio** does not appear in the SYSPRIO expression, then the priority specified here in the administration file is ignored. See “Prioritizing the LoadLeveler Job Queue” on page 25 for more information about the **ClassSysprio** keyword.

include_users = *list*

where *list* identifies the users who are permitted to submit jobs of that *class name*. If provided, this list limits users to those on the list. Do not specify both a list of included users and a list of excluded users. The syntax of this list is a space delimited list of user names. Only one of these may be used for any class.

exclude_users = *list*

where *list* identifies the users who are *not* permitted to submit jobs of that *class name*. Do not specify both a list of included users and a list of excluded users. The syntax of this list is a space delimited list of user names. Only one of these may be used for any class.

Setting Limits: When specifying the following keywords in the administration file: **cpu_limit**, **data_limit**, **core_limit**, **file_limit**, **stack_limit**, and **rss_limit**, only a *hardlimit* may be defined. When a job reaches a *hardlimit*, it is terminated.

When a job reaches a *softlimit*, it receives a signal and is allowed to save information before stopping gracefully. A *softlimit* must be defined by the user in the job command file (see Appendix C, "LoadLeveler Job Command File and Keywords" on page 111).

If no *hardlimit* is specified by the user in the job command file, then the limit defined here for the job class is used.

If no *hardlimit* is defined for the job class in the administration file, then the limit set on the machine that runs the job is used.

If the resulting *hardlimit* is less than the *softlimit* specified in the job command file, then the *softlimit* is lowered to the *hardlimit*.

Any *hardlimit* specified by the user in the job command file must be greater than or equal to the specified *softlimit*, and less than or equal to the limit set by the administrator here in the class stanza of the administration file.

If no limit is defined in the administration file and the *hardlimit* defined by the user in the job command file is greater than the limit on the executing machine, then the *hardlimit* is set to the machine limit.

cpu_limit = *hardlimit*

These values specify the maximum CPU time to be used by the job, expressed as *hours:minutes:seconds.fraction*. Fractions are rounded to seconds. The following strings can also be used:

rlim_infinity	represents the largest positive number.
unlimited	has same effect as **rlim_infinity**.
copy	use the limit currently active when the job is submitted.

Some examples:

cpu_limit = 12:56:21	*hardlimit* = 12 hours 56 minutes 21 seconds
cpu_limit = 56:21	*hardlimit* = 56 minutes 21 seconds
cpu_limit = 1:03	*hardlimit*= 1 minute 3 seconds
cpu_limit = unlimited	*hardlimit* = 2, 147, 483, 647 seconds (X'7FFFFFFF')
cpu_limit = rlim_infinity	*hardlimit* = 2, 147, 483, 647 seconds (X'7FFFFFFF')

cpu_limit = copy current CPU *hardlimit* value on the submitting machine.

data_limit = *hardlimit*

Specifies the maximum size data segment to be used by the submitted job, expressed as *integer.fraction units*, where integer and fraction represent strings of up to eight digits. The units may be:

b bytes

w words

kb kilobytes (2^{10} bytes)

kw kilowords (2^{10} words)

mb megabytes (2^{20} bytes)

mw megawords (2^{20} words)

gb gigabytes (2^{30} bytes)

If no units are specified, then bytes are assumed.

The same shorthand strings can be used to specify an infinite limit or copy the limit from the submitting machine as described for the **cpu_limit** parameter.

Some examples:

data_limit = 125621 *hardlimit* = 125621 bytes

data_limit = 5621kb *hardlimit* = 5621 kilobytes

data_limit = 2mb *hardlimit* = 2 megabytes

data_limit = 2.5mw *hardlimit* = 2.5 megwords

data_limit = unlimited *hardlimit* = 2,147,483,647 bytes (X'7FFFFFF')

data_limit = rlim_infinity *hardlimit* = 2,147,483,647 bytes (X'7FFFFFF')

data_limit = copy copy data *hardlimit* value from submitting machine.

core_limit = *hardlimit*

Specifies the maximum size of a core file, expressed in the same format as the **data_limit**. The same shorthand strings that>were described for **cpu_limit** may be used to specify infinity or to copy limits from the submitting machine. The *hardlimit* parameter is interpreted the same way as explained for **cpu_limit**.

file_limit = *hardlimit*

Specifies the maximum size a file can be created, expressed in the same format as the **data_limit**. The same shorthand strings that>were described for **cpu_limit** may be used to specify infinity or to copy limits from the submitting machine. The *hardlimit* parameter is interpreted the same way as explained for **cpu_limit**.

stack_limit = *hardlimit*

Specifies the maximum size for a stack, expressed in the same format as the **data_limit**. The same shorthand strings that were described for **cpu_limit** may be used to specify infinity or to copy limits from the submitting machine. The *hardlimit* parameter is interpreted the same way as explained for **cpu_limit**.

rss_limit = *hardlimit*

Specifies the maximum resident set size, expressed in the same format as the **data_limit**. The same shorthand strings that>were described for **cpu_limit** may be used to specify infinity or to copy limits from the submitting machine. The *hardlimit* parameter is interpreted the same way as explained for **cpu_limit**.

Specify the following keywords in the administration file when you want jobs submitted to this class to be routed to NQS machines:

NQS_class = true|**false**

When **true**, any job submitted to this class will be routed to an NQS machine.

NQS_submit = *name*

Where *name* is the name of the NQS pipe queue to which the job will be routed. When the job is dispatched to LoadLeveler, LoadLeveler will invoke the **qsub** command using the name of this queue.

NQS_query = *queue names*

Where *queue names* is a blank delimited list of queue names (including host names if necessary) to be used with the **qstat** command to monitor the job and with the **qdel** command to cancel the job.

For more information on routing jobs to machines running NQS, refer to Chapter 4, “Routing Jobs to NQS Machines” on page 53.

The following is an example of a class stanza:

```
# Define class stanzas

class_a:                        # first class
   type = class
   priority = 1
   exclude_users = green bill2 judy
```

Here is an example of a class stanza with all possible limits specified:

```
Metals:
   type = class
   cpu_limit = 72:  :
   data_limit = 2 97152b
   corelimit = 2 48kb
   file_limit = 2mb
   stack_limit = 52488w
   rss_limit = 512kw
```

See Appendix D, “Sample Files” on page 121 for more examples.

Machine Stanzas

The information in a machine stanza defines the characteristics of that machine. A minimum of one machine stanza must be specified, identifying the central manager where both the **negotiator** and **collector** daemons run.

If the **MACHINE_AUTHENTICATION** keyword is coded **True** in the configuration file, then a machine stanza is required for each node that LoadLeveler includes in the pool.

Machine stanzas are specified as follows:

machine name

is the name of the machine.

type = machine

identifies the stanza as machine information.

central_manager = true| **false**

where **true** designates this machine as the LoadLeveler central manager host, where both the **negotiator** and **collector** daemons run, and **false** specifies that no central manager runs on this machine. Only one machine stanza should have this keyword defined as **true** or you may see unexpected results.

schedd_host = true| **false**

where **true** designates this as a public submission machine, used to help submit-only machines access LoadLeveler hosts that run LoadLeveler jobs. Submit-only machines do not run LoadLeveler jobs.

submit_only = true| **false**

where **true** designates this as a submit-only machine. If set to **true**, set the **central_manager** and **schedd_host** to **false**.

Here is an example of a machine stanza:

```
# Define machine default stanza
#
bronze:
        type = machine
        central_manager = true      # central manager runs here
        schedd_host = false
```

Checkpointing and Migrating Jobs During Execution

Checkpointing is a method of periodically saving the state of a job so that, if for some reason, the job does not complete, it can be restarted from the saved state.

At checkpoint time, a checkpoint file is created on the executing or target machine. The checkpoint file contains the text data, stack segments, register contents, and the status of the open files at the time of the checkpoint.

Some system calls have been changed to allow LoadLeveler to save certain state related information. For example, “open()” records all files used by the process so that at checkpoint time the current file pointer can be saved and then restored at restart time.

For some processes, it is impossible to obtain or recreate the state of the process. For this reason, users should only checkpoint jobs whose states are simple enough to checkpoint and recreate. A job that is long-running, computation-intensive, and runs as a single process, is an example of a job well suited for checkpointing.

Any checkpointing job should not use the following system calls in order to prevent unpredictable results from occurring:

- Administrative (for example, audit, swapqry)
- Signals
- Fork
- Dynamic loading
- Shared memory
- Semaphores
- Messages
- Internal timers
- Set user/group id

Another limitation of checkpointing jobs is file I/O. Since individual write calls are not traced, the file recovery scheme requires that all file updates be *idempotent*. In other words, all I/O operations, when repeated, must yield the same result. A job that opens all files as read only can be checkpointed. A job that writes to a file and then reads the data back may also be checkpointed. An example of I/O that could cause unpredictable results is reading, writing, and then reading again the same area of a file.

The system administrator controls the frequency of checkpointing by specifying the **MIN_CKPT_INTERVAL** and **MAX_CKPT_INTERVAL** in seconds in the configuration file. The first checkpoint occurs at the **MIN_CKPT_INTERVAL**. For subse-

quent checkpoints the interval between successive checkpoints is doubled until the **MAX_CKPT_INTERVAL** is reached. For example, given **MIN_CKPT_INTERVAL=1000** and **MAX_CKPT_INTERVAL=7500**, checkpoints will be taken at the following intervals: 1000, 2000, 4000, 7500, 7500, ... The default **MIN_CKPT_INTERVAL** value is 900 (15 minutes). The default **MAX_CKPT_INTERVAL** value is 7200 (120 minutes). The system administrator can set these keyword values globally, so that all machines in the cluster have the same value, or he can specify a different value for each machine.

As discussed earlier, all of your checkpointing jobs must be linked with the LoadLeveler libraries and LoadLeveler checks to make sure the job is linked correctly. When submitting a job to a machine other than the RISC System/6000, LoadLeveler does not recognize the executable. This means that you can submit checkpointing jobs only to machines of the same type. For example, you can submit checkpointing jobs from a Sun to a Sun, from a RISC System/6000 to RISC System/6000, from an SGI to another SGI machine, or from an HP to another HP machine.

A checkpoint does not automatically occur when a job is vacated. When a job is vacated, the most recent checkpoint file taken before the job was vacated will be sent back to the scheduling machine.

In the case of a target or executing machine failure, upon machine restart the checkpoint file is read and rescheduled for execution on an available machine.

The LoadLeveler administrator can manually force jobs enabled for checkpointing on a machine to be checkpointed using the **llctl** flush command. Note that no new jobs are redispatched to this host until the **resume** option is issued on the **llctl** command.

Parallel jobs cannot be checkpointed.

Checkpointing for the RISC System/6000

To support checkpointing, LoadLeveler has altered the UNIX system call mechanism by providing a special version of the C library (libc.a). Therefore, you must link the program with the LoadLeveler libraries. Linking instructions vary slightly from language to language. Jobs that run without checkpoint do not require relinking. In AIX, your LoadLeveler administrator, as part of the installation process, should have activated the following commands to allow you to do the compile/link:

llcc (for use with C)

llxlf (for use with FORTRAN)

C Example

```
llcc -o MYPROG.ckpt MYPROG.c
```

FORTRAN Example

```
llxlf -o MYPROG.ckpt MYPROG.f
```

Checkpointing for the Sun

When submitting a checkpoint job to a Sun machine, link your programs with LoadLeveler's version of crt0.o, and LoadLeveler's libckpt.a and libloadl.a. You can find these in the **/usr/lpp/LoadL/sunosnfs/lib** directory. Since the UNIX compilers automatically link in crt0.o, you must link your program by calling ld directly, substituting "loadl_rt0.o" for "crt0.o". You must also tell the linker to use static linking, since LoadLeveler does not support the use of shared libraries.

C Example

```
ld -o a.out -dc -e start -Bstatic /usr/lpp/LoadL/sunosnfs/lib/loadl_rt .o
a.o b.o c.o /usr/lpp/LoadL/sunosnfs/lib/libckpt.a
/usr/lpp/LoadL/sunosnfs/lib/libloadl.a /usr/lib/libc.a
```

FORTRAN Example

```
ld -o a.out -dc -dp -e start -Bstatic -u _MAIN_ -u _main -X
/usr/lpp/LoadL/sunosnfs/lib/loadl_rt .o /usr/lang/SC1. /cg87/_crt1.o
-L/usr/lang/SC1. /cg87 -L/usr/lang/SC1.  a.o b.o c.o -lF77 -lm
/usr/lpp/LoadL/sunosnfs/lib/libckpt.a
/usr/lpp/LoadL/sunosnfs/lib/libloadl.a -lc
```

Checkpointing for the SGI

When checkpointing jobs to an SGI machine, link your programs with LoadLeveler's version of crt1.o (loadl_rt0.o) and LoadLevler's libckpt.a and libloadl.a. You can find these in the directory **/usr/lpp/LoadL/sginfs/lib**. Since IRIX compilers automatically link in crt1.o, you must link your programs by calling ld directly, substituting "loadl_rt0.o" for "crt1.o".

C Example

Compiling:

cc -c test01.c

Linking:

/usr/bin/ld -o test01 -G 8 -g0 -nocount /usr/lpp/LoadL/sginfs/lib /loadl_rt0.o -count test01.o -nocount /usr/lpp/LoadL/sginfs/lib/libckpt.a /usr/lpp/LoadL/sginfs/lib/libloadl.a -lrpcsvc -lsun -lm -lc /usr/lib/crtn.o

FORTRAN Example

Compiling:

f77 -c test01.f

Linking:

/usr/bin/ld -o test01 -Xnobsschange -G 8 -g0 -nocount /usr/lpp/LoadL/sginfs/lib/loadl_rt0.o -count test01.o -nocount /usr/lpp/LoadL/sginfs/lib/libckpt.a /usr/lpp/LoadL/sginfs/lib/libloadl.a -lF77 -lm -lU77 -lI77 -lisam -lmpc -lrpcsvc -lsun -lc /usr/lib/crtn.o

Checkpointing for the HP

When checkpointing jobs to an HP-UX system, link your programs with LoadLeveler's version of crt0.o (loadl_rt0.o) and LoadLevler's libckpt.a and libloadl.a. You can find these in the directory /usr/lpp/LoadL/hpnfs/lib. Since HP-UX compilers automatically link in crt0.o, you must link your programs by calling ld directly, substituting "loadl_rt0.o" for "crt0.o".

C Example

Compiling:

cc -c test01.c

Linking:

/bin/ld -o test01 /usr/lpp/LoadL/hpnfs/lib/loadl_rt0.o test01.o -a archive /usr/lpp/LoadL/hpnfs/lib/libckpt.a /usr/lpp/LoadL/hpfs/lib/libloadl.a -lm -lc

FORTRAN Example

Compiling:

f77 -c test01.f

Linking:

/bin/ld -o test01 /usr/lpp/LoadL/hpnfs/lib/loadl_rt0.o test01.o -a archive /usr/lpp/LoadL/hpnfs/lib/libckpt.a /usr/lpp/LoadL/hpfs/lib/libloadl.a -lcl -lisamstub -lc

Chapter 3. Working with the SP1

Users can now use LoadLeveler to schedule parallel batch jobs they have written using the IBM Parallel Environment Library* to the 9076 SP1*. They can also create interactive login session on the SP1. This chapter discusses how to perform these two tasks. Before continuing, be sure you are familiar with the IBM 9076 Scalable POWERparallel Systems library, specifically, the *IBM 9076 Scalable POWERparallel Systems: Administration Guide*.:

IBM 9076 Scalable POWERparallel Systems: Planning and Installation.

IBM 9076 Scalable POWERparallel Systems: Maintenance Information Manual.

IBM 9076 Scalable POWERparallel Systems: Administration Guide.

Note: When purchasing an SP1, the LoadLeveler program is provided with no additional charge.

The SP1 is IBM's initial offering in a family of scalable technical computing systems for both serial and parallel computing. Its software allows users to run combinations of interactive, batch, serial, and parallel jobs.

Interactive jobs are run in the foreground of the SP1 and may require user input as the job progresses.

Batch jobs are run in the background of the SP1 and do not require user input during processing. Batch jobs free the user allowing him to work on other tasks.

Serial jobs are processed sequentially and can be run in either interactive or batch mode.

Parallel jobs are broken down and processed in parts and can be run in either interactive or batch mode.

The SP1 consists of a single system frame of eight to 16 RISC System/6000 nodes. This configuration can be scaled upward, in increments of two, by connecting additional frames. Central control for the SP1 is provided by a customer-supplied control workstation and a file server allows administrators to manage files. The following is an example of the basic SP1 configuration. One SP1 frame, one control workstation, and one file server allow users to run both parallel and serial jobs in either interactive or batch mode.

Scheduling Parallel Jobs to the SP1

If your organization has an SP1, users can schedule both parallel and serial jobs on it. This section discusses how to schedule parallel jobs on the SP1. If you want to schedule serial batch jobs on the SP1, you need to install the LoadLeveler **startd** daemon on every node that you want to run serial batch jobs. You can install the **startd** on nodes that will run parallel jobs but that means that serial and parallel jobs may be running together on that node and performance may be affected.

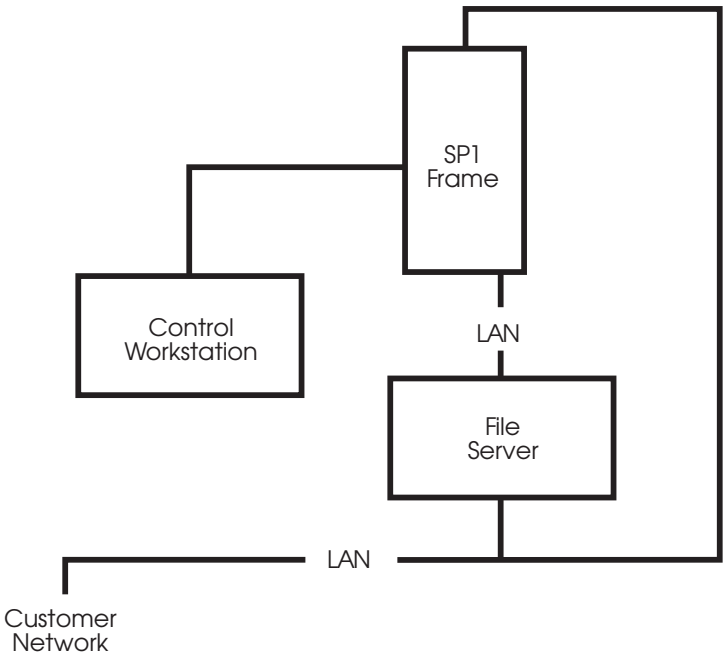


Figure 4. Basic SP1 Configuration.

Prior to running parallel jobs on the SP1, the SP1 system administrator designates which nodes in the frame will run parallel jobs and which will run serial jobs. The nodes that will run parallel jobs are controlled by the *Resource Manager* which is delivered with the SP1. The Resource Manager is a program that determines the amount and type of resources a parallel job needs.

The group of nodes that will run parallel jobs is called a parallel pool. Since parallel jobs can be run interactively or submitted to LoadLeveler (batch jobs), the parallel pool is divided into batch and interactive pools in order to cater to the two different job types. Every node on the SP1 that will be running serial batch jobs must have a LoadLeveler **startd** daemon on it. These **startd** daemons on the SP1 that have access to the parallel pool, poll the Resource Manager for information on the number of parallel nodes available for use. It reports this number, as part of its status, to the **negotiator** daemon which uses this information to schedule the parallel job. The **startd** tells the Resource Manager to reserve the user-specified number of parallel nodes to run the parallel job. The **startd** then initiates the LoadLeveler job which invokes the *Partition Manager* which picks up the reserved nodes. The Partition Manager is a set of subroutines that are linked into the parallel program and run the program on the parallel nodes.

Figure 5. Basic LoadLeveler/SP1 Configuration

Note that all of the nodes that are running the **startd** can run serial jobs up to MAXSTARTERS. Although not shown in this picture, the **startd** can also run on node 1 that is running the Resource Manager. Only nodes 2 and 4 can run parallel jobs. Keep in mind that a parallel job, in this discussion, is a serial job that invokes the Partition Manager. Node 2 can access parallel machines from pool 1 and node 4 can access the parallel machines from pool 2 (as indicated with the number appearing in the parenthesis). You can install the **startd** on nodes eight - 16 (the parallel nodes) but that means that serial and parallel jobs may be running together on that node and performance may be affected.

The remainder of this chapter lists the tasks the LoadLeveler administrator needs to perform to allow users to schedule parallel jobs.

Task 1: Designate SP1 Batch Parallel Nodes

The task of dividing up the SP1 nodes into parallel and serial nodes is made up of several steps:

Step 1: Determine if the Resource Manager is Running: To determine if the Resource Manager is running, review the following table:

If the SP1:	Then:
Has the high-performance switch and it has been initialized	The Resource Manager is running. Go to step 2 of task 1.
Does not have an initialized high-performance switch	The Resource Manager is not running and needs to be started. Continue with step 1 of task 1.
Note: The optional high-performance switch connects all the nodes in the SP1. Redundant connections between processors and frames guarantees the concurrency and high-availability of communications, making it ideal for parallel application execution. If you install the switch in your SP1, you must include one switch for each frame.	

To start the Resource Manager:

Add the start command for the Resource Manager to the **/etc/inittab** file on the control workstation. The start command is:

```
resd:2:boot:/usr/bin/resd -c
```

Add the name of the SP1 nodes to the **/etc/SP1/node_list** file on the control workstation.

Add the following line to the **/etc/inittab** file on the control workstation:

```
resd:2:boot:/usr/bin/resd -c
```

Add the following to the **/etc/SP1/node_list** file on the control workstation:

- The external host name of each node in the format node_name:.
- The external host name of the control workstation in the format node_name:.

Step 2: Configure the Resource Manager:

1. Edit the **/etc/SP1/node_types** file.
2. Add an entry for each node in your SP1 that you want to use for parallel processing. The format is:

   node_name:pool_type:sub_pool_id

   node_name — External host name of the nodes.

   pool_type — 1 = batch, 0 = interactive.

   Only designate your processor nodes for batch parallel processing (pool_type = 1) when using LoadLeveler.

   sub_pool_id — An integer identifying the sub-pool for this node within its pool_type.

   There is only one interactive sub-pool. Assign all interac-

tive nodes a sub_pool_type 0. Batch parallel nodes can be divided into sub-pools based on LoadLeveler criteria.

For example, an entry in the **node_types** file might be:

```
f 1n 1.kgn.ibm.com:1:2
```

f 1n 1.kgn.ibm.com Specifies node's external host name
1 Designates this node for batch use
2 Specifies batch sub-pool number.

In the following example of a **node_types** file, there are two batch sub-pools and one interactive pool:

```
f 1n 1.hpssl.kgn.ibm.com: :
f 1n 2.hpssl.kgn.ibm.com: :
f 1n 3.hpssl.kgn.ibm.com: :
f 1n 4.hpssl.kgn.ibm.com: :
f 1n 5.hpssl.kgn.ibm.com: :
f 1n 6.hpssl.kgn.ibm.com: :
f 1n 7.hpssl.kgn.ibm.com: :
f 1n 8.hpssl.kgn.ibm.com: :
f 1n 9.hpssl.kgn.ibm.com:1:1
f 1n1 .hpssl.kgn.ibm.com:1:1
f 1n11.hpssl.kgn.ibm.com:1:1
f 1n12.hpssl.kgn.ibm.com:1:1
f 1n13.hpssl.kgn.ibm.com:1:1
f 1n14.hpssl.kgn.ibm.com:1:1
f 1n15.hpssl.kgn.ibm.com:1:2
f 1n16.hpssl.kgn.ibm.com:1:2
```

Note: If you need to change the Resource Manager configuration to alter the number of nodes designated as interactive and batch or if some nodes are no longer accessible on the Ethernet, you must change the **node_types** file. The Resource Manager reads this file every time it receives a client request. Once you change this file, the new node designations are available to the Resource Manager daemon on the next client request.

To automate this process, you can establish files with different sets of node designations and use **crontabs** to copy these files to **node_types** on a scheduled basis.

Task 2: Update the Local Configuration File

After determining which machines will run the Partition Manager that starts the parallel jobs, update the local configuration file for each of those machines. To do this, set the POOL_ID keyword equal to the SP1 batch sub-pool to which this machine should have access. If a valid POOL_ID is not defined in the configuration file, the **startd** daemon will not accept parallel jobs.

Sample Parallel Job Submission Scenario

The following sample scenario walks you through the process of submitting a parallel job to the SP1. This scenario assumes the following:

The name of the parallel job is my_parallel_job.

The end user has already created the job command file that specifies the keyword **parallelmachines** to specify the number of parallel machines needed on the SP1. The parallel job requires eight parallel machines.

You have one SP1 that has 16 nodes in it. You want eight of these nodes to run parallel jobs and the remaining eight to run serial jobs.

You do not have an initialized high-performance switch.

Task 1: Designate SP1 Batch Parallel Nodes

Follow these steps to designate eight nodes to run parallel jobs:

Step 1: Start the Resource Manager

Add the start command for the Resource Manager to the **/etc/inittab** file on the control workstation. The start command is:

```
resd:2:boot:/usr/bin/resd -c
```

Add the SP1 host names to the **/etc/SP1/node_list** file on the control workstation.

Add the following line to the **/etc/inittab** file on the control workstation:

```
resd:2:boot:/usr/bin/resd -c
```

Add the following to the **/etc/SP1/node_list** file on the control workstation:

- The external host name of each node in the format node_name:.
- The external host name of the control workstation in the format node_name:.

Step 2: Configure the Resource Manager: To configure the Resource Manager, edit the **/etc/SP1/node_types** file by adding an entry for each node in your SP1 that you want to use for parallel processing. In this case, you want eight of the SP1 nodes to be used for parallel processing:

```
f 1n 1.kgn.ibm.com:1:2
f 1n 2.kgn.ibm.com:1:2
f 1n 3.kgn.ibm.com:1:2
f 1n 4.kgn.ibm.com:1:2
f 1n 5.kgn.ibm.com:1:2
f 1n 6.kgn.ibm.com:1:2
f 1n 7.kgn.ibm.com:1:2
f 1n 8.kgn.ibm.com:1:2
```

Task 2: Update the Local Configuration File

Now that you have determined which nodes will run the parallel jobs, update the local configuration file for each of the **startd** machines that you want to run parallel jobs by setting the POOL_ID keyword equal to the SP1 batch sub-pool to which this machine has access. Each local configuration file should have the following line:

```
POOL_ID = 2
```

Task 3: Submit the Parallel Jobs

As an administrator, you can add the **parallelmachines** keyword to your local configuration file to eliminate the possibility of users submitting serial jobs to a machine designated to run parallel jobs. For example, in the START expression:

```
START: parallelmachines>
```

In order to support batch-parallel jobs, users must add the **parallelmachines** keyword to their job command file. This keyword specifies the number of parallel machines needed by the user to run the parallel job. Invoke the executable as you

normally would from the command line. You still need to pass the number of tasks needed by your program. This is where the Partition Manager picks up the nodes LoadLeveler has reserved. This environment variable must be set less than or equal to the number of parallel machines specified with the keyword **parallelmachines**. In this example, the parallel job requires eight parallel nodes to run. Therefore, the job command file would look similar to:

```
#
# @ input = job.in
# @ output = job.out
# @ error = job.err
# @ parallelmachines = 8
# @ queue
      my_parallel_job -procs 8 -resd yes -hfile NULL
```

Task 4: Obtain Status of Parallel Jobs

Both end users and LoadLeveler administrators can obtain status of parallel jobs in the same way as they obtain status of serial jobs - either by using the **llq** command or by viewing the Jobs window on the graphical user interface. As an administrator, you can create a class for parallel jobs. Then, when users check the status of their parallel jobs, they can identify their parallel jobs by the class field on the Jobs window. The user only sees the **startd** job which is running the Partition Manager. LoadLeveler does not know about the jobs spawned from the Partition Manager.

LoadLeveler monitors the **startd** that is running the Partition Manager that in turn, is running the parallel job on the SP1 nodes. If LoadLeveler loses contact with SP1, the Partition Manager recognizes this and terminates the parallel job.

Task 5: Cancel Parallel Jobs

Administrators and users can cancel parallel jobs using the LoadLeveler **llcancel** command. All they need to know is the LoadLeveler job id for the parallel job. Once they submit their request to cancel the job, LoadLeveler forwards their request to the **startd** which in turn, forwards the request to the Partition Manager. The Partition Manager cancels the job.

Using Interactive Session Support

Interactive Session Support (ISS) distributes user and application login sessions across the nodes of an SP1 in a manner transparent to the user or the application. ISS interfaces to a standard TCP/IP nameserver that has authority over the nodes in the server pool; the nameserver converts a machine name into an internet address. Network routers use this number to connect the user to the correct remote machine. For example, machine name *sp1* is resolved by the local nameserver to the internet address, 1.2.3.4. This nameserver is referred to in this book and in sample setup files as the ISS nameserver, or as the *iss_server*.

With ISS, there is a special machine name that refers to a group of nodes in the SP1. When a user or application wants to connect to the SP1, a connection is made to the generic SP1 node name. Before establishing a connection to the

specified remote machine, the client machine consults its nameserver (which is connected to the ISS server) for the internet address of the machine to which this name refers. On a scheduled basis, ISS polls designated nodes, determines which node is best suited to support another login session, and provides the internet address of this node to the nameserver; this address is associated with the SP1 special machine name.

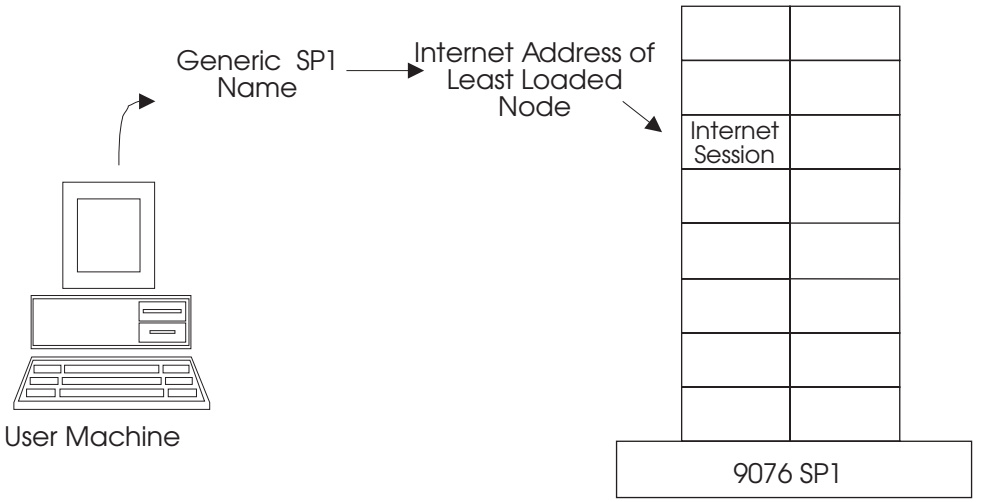


Figure 6. Internet login sessions.

You can configure and tune ISS to respond to the specific environment in terms of frequency of access, total number of users, and types of access: short queries, long-running MIS queries, CPU-intensive loads, and so on. Typically, ISS provides the internet address of the least-loaded node in the group to the requesting user or application. ISS monitors and assesses the group's activity metrics (as configured), and equitably distributes user and application connection requests across the server nodes in the group.

ISS also supports multiple generic names and their associated (possibly overlapping) groups of nodes in the SP1. You may, for example, dedicate four nodes of a 16-node SP1 to long-running MIS-type queries which are balanced according to a *load average* statistic, while the remaining 12 nodes are used for short queries and update transactions which might be balanced using a simple *round-robin* scheduling algorithm. The node groups associated with configured generic names, and the measurement metric and algorithms used to make the login load balancing decisions are all accessible to you who can change and tune the system by modifying the ISS configuration file, **config.data**.

Note: **config.data** does not have to be called by that name. You may have a variety of configuration files for different workloads and server group definitions.

Note: For more information on the following topics, refer to the README file provided with LoadLeveler:

config.data file

Starting and Using ISS

Stopping ISS

Chapter 4. Routing Jobs to NQS Machines

Users can now submit jobs to LoadLeveler and have them routed to a machine outside of the LoadLeveler cluster that runs NQS. LoadLeveler supports COSMIC NQS version 2.0 and other versions of NQS that support the same commands and options and produce similar output for those commands.

The following diagram illustrates a typical environment that allows users to have their jobs routed to machines outside of LoadLeveler for processing:

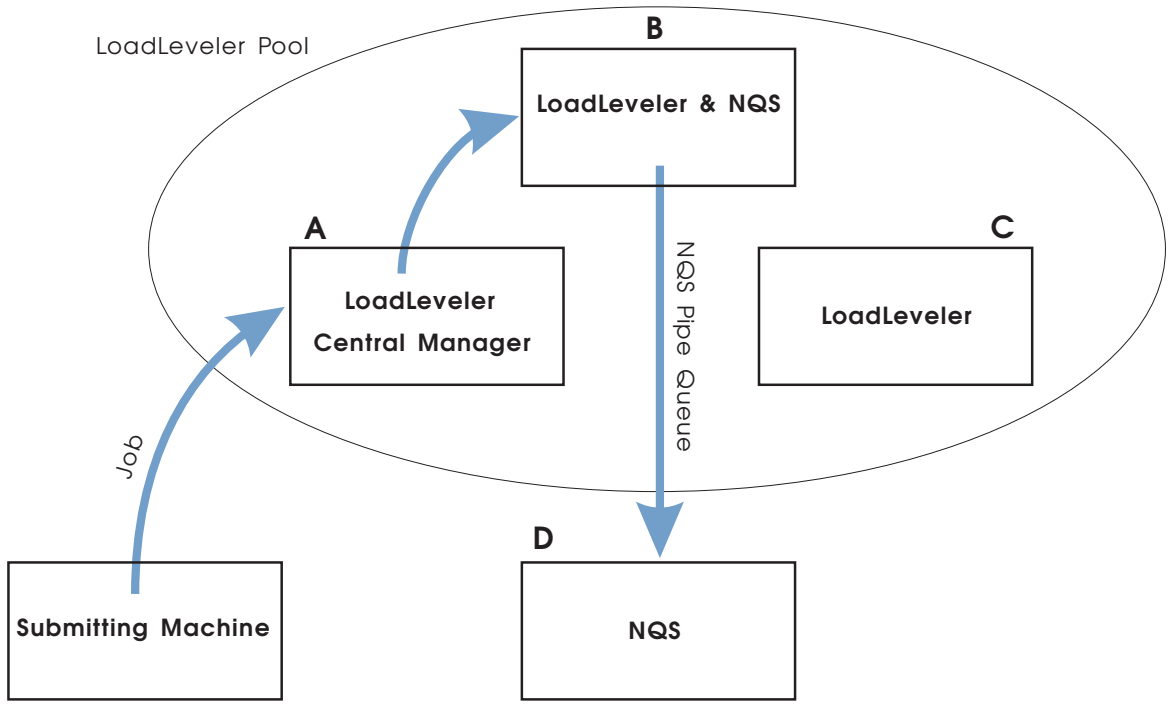


Figure 7. Environment illustrating jobs being routed to NQS machines.

As the diagram illustrates, machines A, B, and C, are members of the LoadLeveler pool. Machine A has the central manager running on it and machine B has both LoadLeveler and NQS running on it. Machine C is a third member of the pool. Machine D is outside of the cluster and is running NQS.

When a user submits a job to LoadLeveler, machine A, that runs the central manager, schedules the job to machine B. LoadLeveler running on machine B routes the job to machine D using NQS. Keep this diagram in mind as you work through the administrator tasks listed on the next page.

Task 1: Setup the NQS Environment

Setting up the NQS environment involves the following:

Install NQS on each node an NQS class is defined

In the previous diagram, this would be machine B.

Create an NQS pipe queue on the LoadLeveler machine whose destination is the NQS batch queue on the machine designated to run the NQS jobs.

In the previous diagram, you would create the NQS pipe queue on machine B.

Create an NQS batch queue on the machine designated to run the NQS jobs.

In the previous diagram, this would be machine D.

Task 2: Designate Machines to Which Jobs Will be Routed

To designate a machine to which your jobs will be routed, your first job is to set up a special class in the **LoadL_admin** file. You do this by adding the following class definitions to the file:

NQS_class = true | **false**

When this flag is set to **true**, any job submitted to this class will be routed to an NQS machine.

NQS_submit = *name*

the name of the NQS pipe queue to which the job will be routed. When the job is dispatched by LoadLeveler, LoadLeveler will invoke the **qsub** command using the name of the this queue.

NQS_query = *queue names*

A blank delimited list of queue names (including host names if necessary) to be used with the **qstat** command to monitor the job and **qdel** to cancel the job.

You can set up multiple classes to access different machines.

Your next step in this task is to modify the local configuration file on the machines that you want to accept this class of jobs.

Next, add the **NQS_DIR** keyword to the **LoadL_admin** file:

NQS_DIR = *NQS directory*
defines the directory where NQS commands **qsub**, **qstat**, and **qdel** reside. The default is /**usr**/**bin**.

Sample Routing Jobs to NQS Machines Scenario

The following example walks you through the process of setting up your environment to route jobs to machines that run NQS.

Assume Figure 7 on page 53 depicts your environment. You have three machines in the cluster named A, B, and C. Outside of the cluster, you have machine D running NQS.

After completing task 1 to setup your NQS environment, modify the **LoadL_admin** file by defining the class **D_CLASS** including the following stanzas:

```
D_CLASS:
  type = class
  NQS_class = true
  NQS_submit = pipe_a
  NQS_query = queue@D
```

Then, modify the **LoadL_config.local** on the machine(s) that you want to accept this class of jobs. In this example, you would modify machine B's **LoadL_config.local** file. To do this, add a class statement similar to:

```
CLASS = {"D_CLASS" "a" "b" ....}
```

where D_CLASS is the name of the class of jobs that will be routed to the machines that run NQS, and a and b are names of additional classes.

Submitting NQS Jobs

After you perform the previous tasks, users can route their jobs to machines running NQS using the **llsubmit** command. The job command file must specify the **class** keyword. For example:

```
class = D_CLASS
```

The job command file must also contain the shell script to be submitted to the NQS node. NQS accepts only shell scripts, binaries are not allowed. All options in the command file pertaining to scheduling the job will be used by LoadLeveler to schedule the job. When the job is dispatched to the node running the specified NQS class, the LoadLeveler options pertaining to the runtime environment are converted to NQS options and the job is submitted to the specified NQS queue.

LoadLeveler command file options are used as follows:

arguments	error message generated and job not submitted
checkpoint	error message generated and job not submitted
class	used only for LoadLeveler scheduling
core_limit	converted to **-lc** option
cpu_limit	converted to **-lt** option
data_limit	converted to **-ld** option
environment	if COPY_ALL is specified, the option is converted to **-x**, otherwise error message generated and job not submitted
error	converted to **-e**
executable	error message generated and job not submitted
file_limit	converted to **-lf** option
hold	used only for LoadLeveler scheduling
image_size	error message generated and job not submitted
initialdir	error message generated and job not submitted
input	error message generated and job not submitted
notification	If the option specified is
	always converted to **-mb**and **-me** options
	error converted to **-me** option

	start	converted to **-mb** option
	never	ignored
	complete	converted to **-me** option
notify_user	converted to **-mu** option	
output	converted to **-o** option	
preferences	used only for LoadLeveler scheduling	
queue	places one copy of job in the LoadLeveler queue	
requirements	used only for LoadLeveler scheduling	
restart	If the option specified is	
	yes	ignored
	no	converted to **-nr** option
rss_limit	converted to **-lw** option	
shell	converted to **-s** option	
stack_limit	converted to **-ls** option	
start_date	used only for LoadLeveler scheduling	
user_priority	used only for LoadLeveler scheduling	

Users can also submit an NQS script. In this case, any NQS options in the script are used to schedule the job and once dispatched by LoadLeveler, the file is sent to NQS unmodified.

LoadLeveler schedules these jobs the same as it schedules other jobs. When the job is dispatched, LoadLeveler determines whether or not it is running in an NQS class. If it is, an NQS command **qsub** is issued.

LoadLeveler monitors the job by periodically invoking a **qstat** command. A **qstat** command is first issued for the pipe queue on the local host. If the request id is not found, a **qstat** is issued for each queue listed in the NQS_query class keyword. If the request id is still not found, starter marks the job as complete.

When a job is sent to an NQS class, **llsubmit** saves the following environment variables:

HOME

LOGNAME

MAIL

PATH

SHELL

TZ

USER

When LoadLeveler dispatches the job, these environment variables are installed so that they are available to **qsub**. **llsubmit** also saves the name of the current directory (pwd) and the current value of the user file create mask (umask).

Obtaining Status of NQS Jobs

Users can obtain status of NQS jobs in the same way as they obtain status of LoadLeveler jobs - either by using the **llq** command or by viewing the Jobs window on the graphical user interface. The users can identify the NQS jobs by the class field on the Jobs window.

LoadLeveler monitors the job until **qstat** shows the job is no longer in either queue. If LoadLeveler loses contact with NQS, a message is written to the shadow log file and the jobs associated with those queues will be considered complete.

NQS does not provide job accounting. Therefore, the only accounting information LoadLeveler will have is the total time for the job.

LoadLeveler will not send mail when the job completes. The LoadLeveler notification option is translated to the appropriate NQS flag (me or mb) and NQS will send the mail.

Cancelling NQS Jobs

Users can cancel NQS jobs using the LoadLeveler **llcancel** command. All they need to know is the LoadLeveler job id for the NQS job. Once they submit their request to cancel the job, LoadLeveler forwards their request to the appropriate node and a **qdel** will be issued for the job for the queue listed in the the **NQS_submit** and **NQS_query** keywords.

Chapter 5. Centralized Job Accounting

LoadLeveler provides accounting information on completed LoadLeveler jobs. Job resource information is collected for a LoadLeveler job using the UNIX wait3 system call. When the job ends, this information is stored in a file named *history* in the spool directory of the machine that initially scheduled this job, the **schedd** machine. In addition to job resource information, job information, such as job parameters, as well as information regarding resources consumed by the **shadow** and **starter**, are also stored. Data on parallel jobs are not collected in the *history* files except for data on the processes which initiate the parallel jobs.

Resource information collected on the LoadLeveler job are constrained by the capabilities of the wait3 system call. Information for processes which fork child processes will include data for those child processes as long as the parent process waits for the child process to terminate. Complete data may not be collected for jobs which are not composed of simple parent/child processes. For example, if you have a LoadLeveler job which invokes an rsh command to execute a function on another machine, the resources consumed on the other machine will not be collected as part of the LoadLeveler accounting data.

Accounting Files

LoadLeveler accounting uses the following types of files:

The local history file which is local to each **schedd** machine, is where job resource information is first recorded. These files are usually named *history* and are located in the spool directory of each **schedd** machine, but you may specify an alternate name and/or directory with the **HISTORY** keyword in either the global or local configuration file.

The global history file is a combination of the history files from some or all of the machines in the LoadLeveler cluster merged together. The command **llacctmrg** is used to collect files together into a global file. As the files are collected from each machine, the local history file for that machine is reset to contain no data. The file is named *globalhist.yymmdd*. You may specify the directory in which to place the file when you invoke the **llacctmrg** command or you can specify the directory with the **GLOBAL_HISTORY** keyword in the configuration file. The default value set up in the sample configuration file is the local spool directory.

Account Validation

If your installation is interested in keeping track of resources used on an account basis, you can require all users to specify an account number in their job command files. This account number is specified with the **account_number** keyword and is explained in detail in Appendix C, “LoadLeveler Job Command File and Keywords” on page 111. LoadLeveler validates this account number by comparing it against a list of account numbers specified for the user in the user stanza in the administration file.

Account validation is under the control of the **ACCT** keyword in the configuration file. The routine which performs the validation is called **llacctval** but you can override this by using the **ACCT_VALIDATION** keyword in the configuration file.

Accounting Reports

You can produce two types of reports from either the local or global history file and you can also specify a range on which to report. See “llsummary - Returns job resource information” on page 77 for information on producing either a short or long report. The short report is the default report produced. The short version displays the following:

Number of jobs, Total CPU usage, per user.

Number of jobs, Total CPU usage, per class.

Number of jobs, Total CPU usage, per group.

Number of jobs, Total CPU usage, per account number.

The long version of the report displays all of the data collected for every job. See the **llsummary** command, “llsummary - Returns job resource information” on page 77,
for examples of the short and long versions of the report.

Programming Interface

Programmers may write C programs to process records in either a local or the global history file. The function **GetHistory** obtains an accounting record from the history file and invokes a caller suppled routine, passing the routine the address of an **llacct** structure. The **GetHistory** function reads every record of the file, specified by the first passed argument. Each record is read into an **llacct** structure and passed to the function passed as the second argument. The parameters are:

file Name of the history file to process.

function Points to the function to call to process each history record.

You need to link your accounting programs with the LoadLeveler accounting libraries which you can find in the **lib** subdirectory of the release directory, usually **/usr/lpp/LoadL/nfs** or **/usr/lpp/LoadL/afs**.

Include header (llacct.h) can be found in the include subdirectory of the release directory, usually **/usr/lpp/LoadL/nfs** or **/usr/lpp/LoadL/afs**.

The syntax is:

```
#include <stdio.h>
#include <sys/resource.h>
#include <sys/types.h>
#include <sys/time.h>
#include "llacct.h"

void
GetHistory (file, function)
char  file;
int function();
```

A sample program called **llphist** is provided in the samples/phist subdirectory of the release directory, usually **/usr/lpp/LoadL/nfs** or **/usr/lpp/LoadL/afs**. **llphist** invokes **GetHistory** to print every record in the history file.

Sample Job Accounting Scenario

The following sample scenario walks you through the process of collecting account data. You can perform all of the steps or just the ones that apply to your situation.

Task 1: Update the Configuration File

Edit the configuration file according to the following table:

Edit this keyword:	To:
GLOBAL_HISTORY	Specify a directory in which to place the global history files.
ACCT	Turn accounting and account validation on and off.
ACCT_VALIDATION	Specify the account validation routine.
Note: See "Defining Job Accounting" on page 20 for more information on these keywords.	

Task 2: Merge Multiple Files Collected From Each Machine Into One File

You can accomplish this step using either the **llacctmrg** command or the graphical user interface:

Using **llacctmrg**

See "llacctmrg - Collects machine history files" on page 69 for the syntax of this command.

Using the Graphical User Interface

Select a machine from the Machines window

Select **Admin** → **Collect Account Data...** from the Machines window.

A window appears prompting you to enter a directory name where the file will be placed. If no directory is specified, the directory specified with the **GLOBAL_HISTORY** keyword in the global configuration file is the default directory.

Press **OK**

The window closes and you return to the main window.

Task 3: Report Job Information on all the Jobs in the History File

You can accomplish this step using either the **llsummary** command or the graphical user interface:

Using **llsummary**

See "llsummary - Returns job resource information" on page 77 for the syntax of this command.

Using the Graphical User Interface

Select **Admin** → **Create Account Report...** from the Machines window.

A window appears prompting you to enter the following information:

- A short or long version of the output. The short version is the default version.
- A start and end date for the report. If no date is specified, the default is to report all of the data in the report.
- The name of the input data file.
- The name of the output data file. This is the same as stdout.

Press **OK**

The window closes and you return to the main window. The report appears in the Messages window if no output data file was specified.

Task 4: Using Account Numbers and Setting Up Account Validation

1. Edit the user stanza in the administration file. The format of this stanza is:

   account = list

   where list is a blank delimited list of account numbers a user may use when submitting jobs.

2. Instruct users to associate an account number with their job.

   Using the job command file

   Add the **account_no** keyword to the job command file. See Appendix C, "LoadLeveler Job Command File and Keywords" on page 111 for details.

   Using the Graphical User Interface

   Select **File** → **Build a Job** from the main window.

   The Build a Job window appears.

   Type the account number in the **account_no** field on the Build a Job window.

   Press OK

   The window closes and you return to the main window.

3. Specify the **ACCT_VALIDATION** keyword in the configuration file that identifies the module that will be called to perform account validation. The default module is called **llacctval**. You can replace this module with your installation's own accounting routine by specifying a new module with this keyword.

Chapter 6. LoadLeveler Commands

LoadLeveler provides two types of commands: those that are available to all users of LoadLeveler, and those that are reserved for use by LoadLeveler administrators. The administrator commands can operate on the entire LoadLeveler job queue and all machines configured. The user commands affect only those jobs submitted by that user.

Some user commands, such as **llhold**, include functions that can only be performed by an administrator identified by the **LOADL_ADMIN** keyword in the configuration file. Refer to “Defining LoadLeveler Administration” on page 19 for more information.

Administrator Commands

llacctmrg	Collects all individual machine history files together into a single file.	See "llacctmrg - Collects machine history files" on page 69
llctl	LoadLeveler Control: use to request action such as starting and stoping daemons on one or more machines in the LoadLeveler pool.	See "llctl - LoadLeveler Control" on page 71
llfavorjob	Reorder System Queue by Job: use to raise one or more jobs to the highest priority, or to restore original priority (unfavor).	See "llfavorjob - Reorder System Queue by Job" on page 73
llfavoruser	Reorder System Queue by User: use to raise job(s) submitted by one or more users to the highest priority, or to restore original priority (unfavor).	See "llfavoruser - Reorder System Queue by User" on page 75
llpreen	Locates and can detects unexpected files in the LoadLeveler directories.	See "llpreen - Search for Unexpected Files in LoadLeveler Directories" on page 76
llsummary	Returns resource information regarding completed jobs.	See "llsummary - Returns job resource information" on page 77

User Commands

llcancel	Cancel a Submitted Job: use to remove one or more LoadLeveler jobs.	See "llcancel - Cancel a Submitted Job" on page 82
llhold	Hold or Release a Submitted Job: Administrators can use this command to move jobs into and out of system hold. All users and administrators can use it to move jobs in and out of user hold.	See "llhold - Hold or Release a Submitted Job" on page 84
llprio	Change Priority of a Submitted Job: use to adjust the priority of one or more jobs in the LoadLeveler queue.	See "llprio - Change Priority of Submitted Job" on page 87
llq	Use to query status on LoadLeveler jobs.	See "llq - Query Job Status" on page 89
llstatus	Query Machine Status: use to request information about one or more machines in the LoadLeveler pool.	See "llstatus - Query Machine Status" on page 93
llsubmit	Submit a Job: use to submit job(s) to the LoadLeveler central manager.	See "llsubmit - Submit a Job" on page 97

The LoadLeveler Graphical User Interface

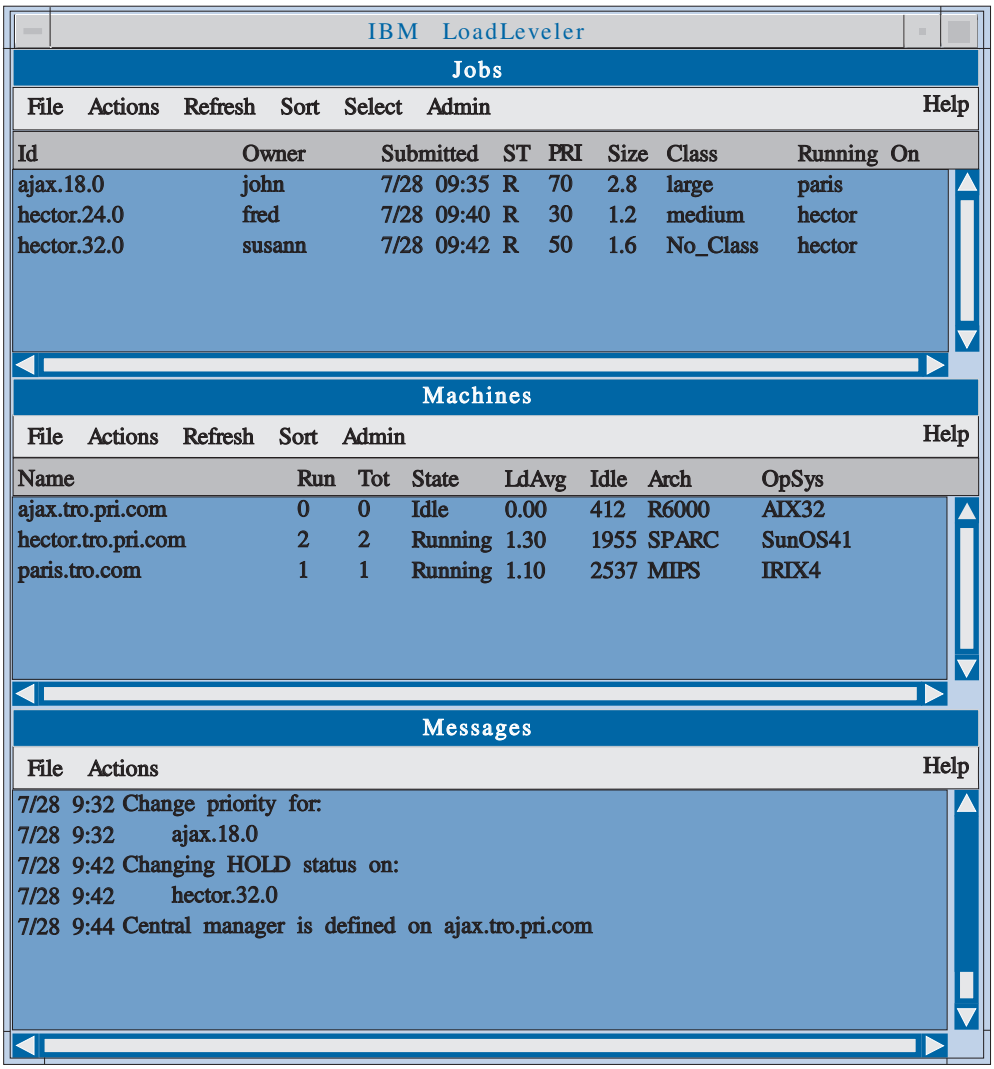


Figure 8. The LoadLeveler GUI

If you have AIXwindows* or the X Window System** installed, you will probably find the LoadLeveler Motif**-based graphical user interface (GUI) faster and easier to use than command line entry. This interface allows all LoadLeveler commands to be executed by button selection with pull-down menus where input or supplemental information is required.

Three panes are presented in the LoadLeveler GUI: one for job status with pull-down menus for job-related commands, one for machine status with pull-down menus for machine-related commands, and one for messages and logs. Facilities are provided for sorting and selecting status in a variety of ways.

To invoke the LoadLeveler GUI, enter:

```
xloadl
```

You access the administrative commands that act on jobs through the **Admin** pulldown menu in the jobs pane of the GUI. In order to cover the range of possibilities provided on the command line, two choices are offered on the pulldown for each of the commands that affect the system queue.

You can access the administrative commands that act on the LoadLeveler hardware resource pool through the **Admin** pulldown menu in the Machines pane of the GUI. These choices correspond to the range of command options available in the **llctl** command.

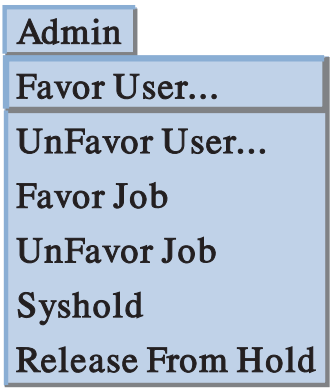


Figure 9. Administrator Commands: Job Pulldown

The LoadLeveler GUI differs from the Motif standard in some ways:

No accelerators or mneumonics appear on the menu bars.

The cursor must be located in the target field for data entry.

Submerged windows do not necessarily rise to the top when refreshed.

The GUI resources are defined in the **Xloadl** resource file. For information on customizing your GUI, refer to Appendix B of the LoadLeveler User's Guide.

How to Read the Syntax Diagrams

Purpose

For each LoadLeveler command, a diagram shows you how to enter the required information correctly on the command line. These diagrams show:

Which flags can be entered

Which flags take parameters

Which flags can and cannot be entered together

Where you must enter flags or parameters and where you have a choice

Where you can repeat flag and parameter sequences.

The following discussion explains how to interpret the syntax diagrams. It begins with an example diagram that shows all of the conventions used in the LoadLeveler commands. Each item in the diagram is explained.

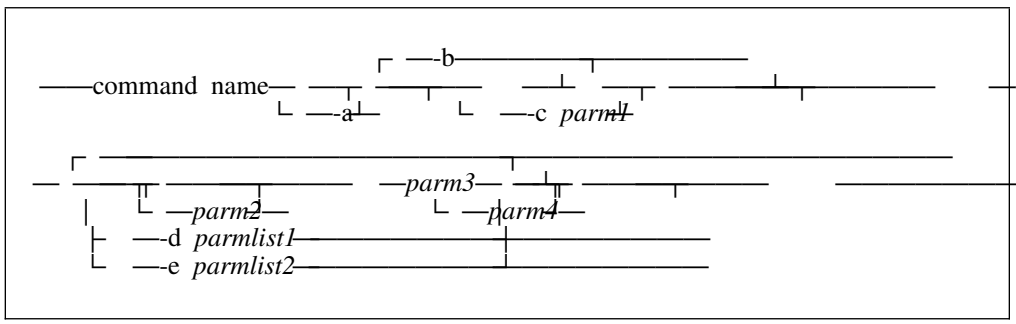
Diagram items that must be entered literally are shown in normal typeface. These items include the command name, all flags, and literal characters.

Variable items appear in italic. These items include parameters that follow flags and parameters that the command reads, such as machine names, user names, and job identifiers.

Required items appear on the central line of the command path.

Default items appear above the central line.

Optional choices are listed below the central line.



where

-a is an optional flag. (Optional flags appear below the line.)

-b is a default flag (defaults appear above the line) that will automatically be in effect unless optional flag **-c** is specified.

-c *parm1* is an optional flag that can be specified instead of flag **-b**. The **-c** flag takes a parameter and requires that information be specified where *parm1* appears.

The single arrow indicates command continuation on the next line.

parm2 is an optional parameter.

parm3 is required if you do not specify the **-d** or **-e** flags.

parm4 is an optional parameter.

-d *parmlist1* is required if you do not specify *parm3* or **-e**. If you specify the **-d** flag, it takes a list of parameters delineated by blanks.

-e *parmlist2* is required if you do not specify *parm3* or **-d**. If you specify the **-e** flag, it takes a different list of parameters, delineated by blanks.

You can specify *parm2*, *parm3*, and *parm4* one or more times, in addition to **-d** and **-e**. This is indicated by the arrow that points back to where you first chose the path of *parm3*, **-d**, or **-e**. You can return to choose another of these options or enter the command as is, since no further options or parameters appear on the path.

Examples

Following are examples of how to enter the command based on the syntax diagram.

```
command name parm3 -d parmitem
command name -a parm3 parm3 -e parmitem parmitem
command name -a -b parm2 parm3 parm2 parm3 -d parmitem parmitem
command name -a -c parm1 parm3 parm4 parm3 -d parmitem parmitem
command name -c parm1 parm3 parm4 parm3 parm2 parm3 -e parmitem parmitem
```

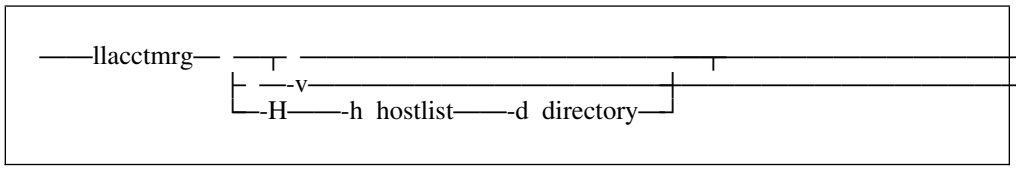
llacctmrg - Collects machine history files

Administrator Command

Purpose

Collects all individual machine history files together into a single file specified as a parameter. The default is to collect data from all the machines identified in the LoadL_admin file but you can also specify a list of machines as a parameter to gather information on a subset of machines.

Syntax



where

-H provides a short usage message.

-v outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

-h *hostlist* a blank delimited list of hosts from which to collect data. The default is all machines listed in LoadL_admin.

-d *directory* directory to hold the new global history file. If not specified, the directory specified in the **GLOBAL_HISTORY** keyword in the configuration file is used.

Usage Notes

llacctmrg collects all individual machine accounting files together into a single file. It defaults to collecting information from all machines identified in the LoadL_admin file. You can also get information on a subset of machines by providing a list of machines as a parameter to the command.

When the **llacctmrg** command ends, accounting information regarding the job is stored in a file called *globalhist.YYMMDDHHMM*. *YY* indicates the year, *MM* indicates the month, *DD* indicates the day, *HH* indicates the hour, and *MM* indicates the minute. Information such as the amount of resources consumed by the job as well as other job related data is stored in this file.

Data on processes which fork child processes will be included in the file only if the parent process waits for the child process to end. Therefore, complete data may not be collected for jobs which are not composed of simple parent/child processes. For example, if a LoadLeveler job invokes an rsh command to execute some function on another machine, the resources consumed on the other machine will not be collected as part of the accounting data.

Data on parallel jobs is not collected in the history files, except for the process which initiates the parallel job.

Examples

The following example collects data from machines named mars and pluto.

```
llacctmrg -h mars pluto
```

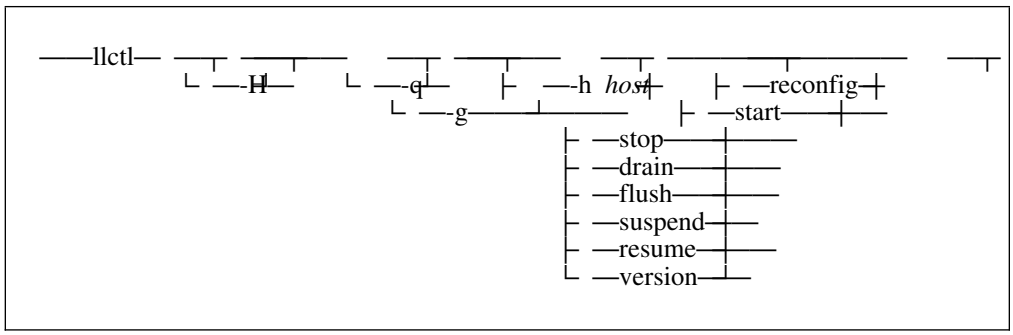
llctl - LoadLeveler Control

Administrator Command

Purpose

Controls LoadLeveler daemons on all members of the LoadLeveler pool. This command sends a message to the central manager requesting that action be taken on the members of the LoadLeveler pool.

Syntax



where the flags have the following meanings:

-H provides a short usage message.

-q specifies quiet mode: print no messages other than error messages.

-h *host* indicates that the command should apply to only this host in the LoadLeveler pool. If neither **-h** nor **-g** is specified, the default is the machine on which the **llctl** command is issued.

-g indicates that the command should apply globally to all machines in the LoadL_admin file.

Flags must be specified before any of the following keywords are specified.

recycle stops all LoadLeveler daemons and restarts them.

reconfig forces all daemons to reread the configuration files.

stop stops the LoadLeveler daemons on the specified host.

start starts LoadLeveler on the specified host. You must have rsh privileges to start LoadLeveler on a remote host.

drain allows no more LoadLeveler jobs to begin running on this host.

flush terminates running jobs on this host and sends them back to the system queue to await redispatch. No new jobs are redispatched to this host until **resume** is issued. Forces a checkpoint if jobs are enabled for checkpointing.

suspend	suspends all jobs on this host.
resume	resumes all jobs on this host.
version	displays version and release data at the screen.

Examples

This example stops LoadLeveler on the host named *iron*:

```
llctl -h iron stop
```

This example restarts the LoadLeveler daemons on all members of the LoadLeveler pool as defined in the machine stanzas of the administration file:

```
llctl -g start
```

This example causes the LoadLeveler daemons on host *iron* to re-read the configuration files, which may contain new configuration information for the *iron* host:

```
llctl -h iron reconfig
```

llfavorjob - Reorder System Queue by Job

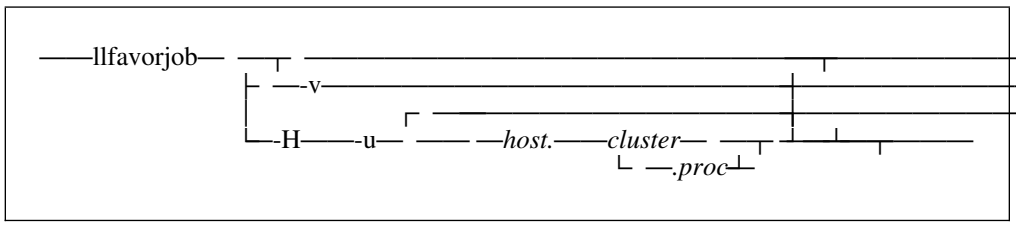
Administrator Command

Purpose

Sets specified jobs to the highest system priority regardless of the current setting of the job priority or SYSPRIO. Jobs already running are not affected but when jobs are vacated jobs are returned to the queue, the job gets re-ordered with the new priority.

This command also *unfavors* previously favored job(s), restoring the original priority, when you specify the **-u** flag.

Syntax



where

-H provides a short usage message.

-u unfavors previously favored jobs, requeuing them according to their original priority levels.

-v outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

host. is the name of the host to which the job was submitted (delimited by dot). The default is the local host.

cluster is the cluster id assigned to the job by LoadLeveler when it was submitted using the **llsubmit** command.

.proc (delimited by dot) is the process number given to the job by LoadLeveler when it was submitted using the **llsubmit** command. The default is to include all members of the cluster.

Usage Notes

If more than one job is affected by this command, then the jobs are ordered by time of submission.

Favoring remains in effect until reversed.

Examples

This example assigns jobs 12.4 on host machine *iron* and 8.2 on *zinc* the highest priorities in the system, with the first job submitted being the highest.

```
llfavorjob iron.12.4 zinc.8.2
```

This example unfavors jobs 12.4 on host machine *iron* and 8.2 on host machine *zinc*.

```
llfavorjob -u iron.12.4 zinc.8.2
```

llfavoruser - Reorder System Queue by User

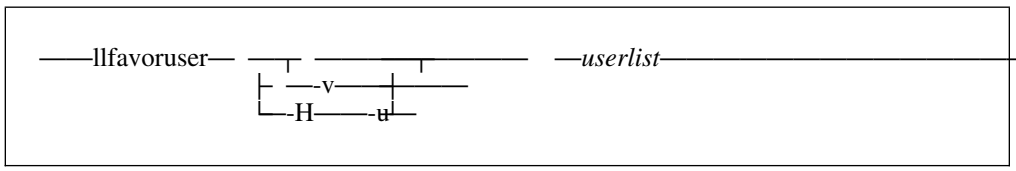
Administrator Command

Purpose

Sets a user's job(s) to the highest priority in the system, regardless of the current setting of the job priority, or SYSPRIO. Jobs already running are not affected.

This command also *unfavors* the user's job(s), restoring the original priority, when you specify the **-u** flag.

Syntax



where

-H provides a short usage message.

-u unfavors previously favored users, requeuing their job(s) according to their original priority level(s).

If **-u** is **not** specified, the user's job(s) are favored.

-v outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

userlist is a blank-delimited list of users whose jobs are given the highest priority.

Usage Notes

If more than one user is affected by this command, then the jobs are ordered by time of submission. This command remains in effect until reversed (with the **-u** option). Until then, a favored user's jobs will continue to be dispatched with the highest priority.

Examples

This example grants highest priority to all queued jobs submitted by users ellen and fred according to their time of submission.

```
llfavoruser ellen fred
```

This example unfavors all queued jobs submitted by users ellen and fred.

```
llfavoruser -u ellen fred
```

llpreen - Search for Unexpected Files in LoadLeveler Directories

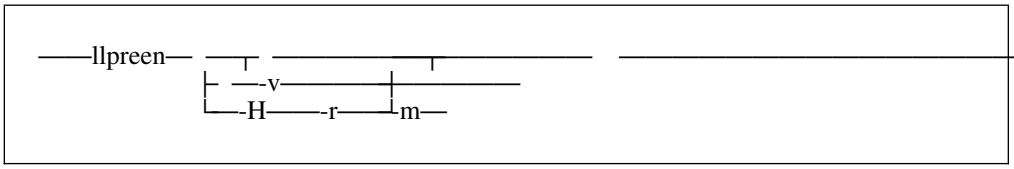
Administrator Command

Purpose

Alerts the administrator of unexpected files in the LoadLeveler spool, log, and execute directories and removes them, if so ordered.

This command is run automatically by LoadLeveler everyday at 3:30 AM. This time cannot be changed.

Syntax



where

-H provides a short usage message.

-r removes any unexpected files.

-m notifies the administrator through the mail of the presence of unexpected files.

-v outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

Usage Notes

When neither **-r** nor **-m** is specified, the command issues a message for each unexpected file detected.

Examples

This example removes unexpected files and alerts the administrator.

```
llpreen -r -m
```

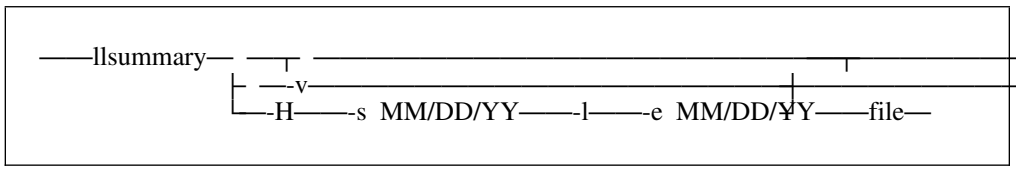
llsummary - Returns job resource information

Administrator Command

Purpose

Returns job resource information regarding completed jobs.

Syntax



Parameters

where

-H provides a short usage message

-v outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

-s specifies the start date for accounting data to be included in this report. The format for entering the date is MM/DD/YY where MM is month, DD is day, and YY is year. The default is to report all the data in the report.

-l specifies that the long form of output will be displayed.

-e specifies the end date for accounting data to be included in this report. The format for entering the date is MM/DD/YY where MM is month, DD is day, and YY is year. The default is to report all the data in the report.

file the file containing the accounting data. If not specified, the default is the local history file on the machine from which the command was issued.

Usage Notes

This command provides both a short and long version of output to standard output:

The Short Listing

The short listing, generated when **-l** is *not* specified with **llsummary**, includes summaries of the data contained in the input history file:

Number of jobs, Total CPU usage, per user.

Number of jobs, Total CPU usage, per class.

Number of jobs, Total CPU usage, per group.

Number of jobs, Total CPU usage, per account number.

The following is an example of the short version:

```
    Name      Jobs       Shadow  Cpu      Starter  Cpu            Job  Cpu     Leverage
 brels4d       1        + :  :19        + :  :21        + :1 :58          16.4
 tmorgan       23        + :  :24        + :  :26        + : 4:26           5.3
    TOTAL      33        + :  :43        + :  :46        + :15:24           1 .4

   Class     Jobs       Shadow  Cpu      Starter  Cpu            Job  Cpu     Leverage
No_Class       1        + :  :19        + :  :21        + :1 :58          16.4
      LL       23        + :  :24        + :  :26        + : 4:26           5.3
    TOTAL      33        + :  :43        + :  :46        + :15:24           1 .4

   Group     Jobs       Shadow  Cpu      Starter  Cpu            Job  Cpu     Leverage
   staff       1        + :  :19        + :  :21        + :1 :58          16.4
     usr       23        + :  :24        + :  :26        + : 4:26           5.3
    TOTAL      33        + :  :43        + :  :46        + :15:24           1 .4

 Account     Jobs       Shadow Cpu      Starter Cpu            Job Cpu     Leverage
    ac1        1        + :  :19        + :  :21        + :1 :58          16.4
 acct1 1       23        + :  :24        + :  :26        + : 4:26           5.3
    TOTAL      33        + :  :43        + :  :46        + :15:24           1 .4
```

where Name - is a userid submitting jobs.

Class - is a class specified or defaulted for the jobs.

Group - is the user's login group.

Account - is an account number specified for the jobs.

Jobs - is a count of the total number of jobs submitted by this user, class, group, or account.

Shadow CPU - is the total cpu time consumed by LoadLeveler shadow processes on behalf of the user jobs.

Starter CPU - is the total cpu time consumed by LoadLeveler starter processes on behalf of the user jobs.

Job CPU - is the total cpu time consumed by user's jobs.

Leverage - is the ratio of user CPU to LoadLeveler (shadow and starter) CPU.

The Long Listing

The long listing, returned by **llsummary** when the **-l** option is specified, includes these fields:

llsummary

```
================== Job seismic.nic.kingston.ibm.com 1.  =======
      Structure Version: 6
                  Owner:  tmorgan
                  Group:  staff
             Account No: ac1
             Queue Date: Mon Apr 19 13:17:59 1993
                 Status:  Completed
        Completion Date: Mon Apr 19 13:18:27 1993
          User Priority: 75
           user_sysprio:
         class_sysprio:
       System Priority:
              q_sysprio:
         Notifications:  Always
    Virtual Image Size: 1 kilobytes
            Checkpoint:  no
       Hold Job Until: (no hold)
                  Cmd:  datproc_seq
                 Args:
                  Env: LOADL_CORESIZE = 1 24
                   In:  /dev/null
                  Out:  datproc_seq.1. .out
                  Err:  datproc_seq.1. .err
    Initial Working Directory: /u/tmorgan/seapsu
            Requirements: (Arch == "R6   ") & (OpSys == "AIX32") &
                (Class == "No_Class")
             Preferences:
      Shadow User Time:      + : :
    Shadow System Time:      + : : 1
     Shadow Total Time:      + : : 1
          Shadow  maxrss:
          Shadow    ixrss:
           Shadow  idrss:
           Shadow  isrss:
          Shadow  minflt:
          Shadow  majflt:
           Shadow  nswap:
         Shadow  inblock:
         Shadow  oublock:
         Starter  msgsnd:
           Shadow  msgrcv:
       Shadow  nsignals:
            Shadow  nvcsw:
           Shadow  nivcsw:
     Starter User Time:      + : :
   Starter System Time:      + : : 1
    Starter Total Time:      + : : 1
         Starter  maxrss:
         Starter    ixrss:
          Starter  idrss:
          Starter  isrss:
         Starter  minflt:
         Starter  majflt:
          Starter  nswap:
        Starter  inblock:
        Starter  oublock:
         Starter  msgsnd:
         Starter  msgrcv:
```

```
Starter  nsignals:
    Starter  nvcsw:
   Starter  nivcsw:
     Job User Time:      + :1 :2
   Job System Time:      + :  :18
    Job Total Time:      + :1 :38
         Job  maxrss:
         Job   ixrss:
          Job  idrss:
          Job  isrss:
         Job  minflt:
         Job  majflt:
          Job  nswap:
        Job  inblock:
        Job  oublock:
          Job  msgsnd:
          Job  msgrcv:
       Job  nsignals:
           Job  nvcsw:
          Job  nivcsw:
     Running host: parpia.kgn.ibm.com
  Submitting host: seismic.nic.kingston.ibm.com
       Notify User: tmorgan@seismic.nic.kingston.ibm.com
             Shell:  /bin/ksh
             Class:  No_Class
```

New fields which are not described in the **llq** command description are:

Account No	account number specified in job command file
Group	login group name
maxrss	Maximum resident set sized utilized.
ixrss	Amount of shared memory used.
idrss	Amount of unshared memory used.
isrss	Integral unshared stack used.
minflt	# Page faults (re-claimed).
majflt	# Page faults (I/O required).
nswap	# times swapped out.
inblock	# times file system performed input.
oublock	# times file system performed output.
msgsnd	# of IPC messages sent.
msgrcv	# of IPC messages received.
nsignals	# of signals delivered.
nvcsw	# of context switches due to voluntarily giving up processor.
nivcsw	# of involuntary context switches.

Examples

The following example requests summary reports (short listing) of all the jobs submitted on your machine between the days of September 12, 1993 and October 12, 1993:

```
llsummary -s  9/12/93 -e 1 /12/93
```

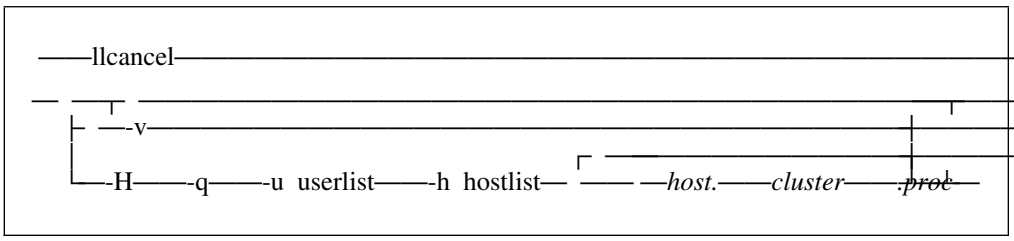
llcancel - Cancel a Submitted Job

User Command

Purpose

Cancels one or more jobs from the LoadLeveler queue.

Syntax



Parameters

where

-H provides a short usage message.

-q specifies quiet mode: print no messages other than error messages.

-v outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

host. is the name of the host to which the job was submitted (delimited by dot). The default is the local host.

cluster is the cluster id assigned to the job when it was submitted using the **llsubmit** command.

.*proc* (delimited by dot) is the proc number assigned to the job when it was submitted using the **llsubmit** command. The default is to include all members of the cluster.

-u *userlist* is a blank-delimited list of users. When used with the **-h** option, only the user's jobs monitored on the host machines in the *hostlist* are cancelled. When used alone, only the user's jobs monitored on the **schedd** machine are cancelled.

-h *hostlist* is a blank-delimited list of hosts. All jobs monitored on host machines in this list are cancelled. When issued with the **-u** option, the *userlist* is used to further select jobs for cancellation.

Usage Notes

The **-u** or **-h** flags override the *host.cluster.proc* parameters.

When the **-h** flag is specified by a non-administrator, all jobs submitted from the machines in *hostlist* by the user issuing the command are cancelled.

When the **-h** flag is specified by an administrator, all jobs monitored on machines in *hostlist* are canceled, unless the **-u** is also specified, in which case all jobs both submitted by users in *userlist* and monitored on machines in *hostlist* are cancelled.

When cancelling a job from a submit-only host, you must specify the host name of the machine that scheduled the job. For example, if you submitted the job from machine A, a submit-only host, and machine B, a scheduling machine, scheduled the job to run, you must specify machine B's host name in the cancel command.

Examples

This example cancels the job bronze.18.3.

```
llcancel bronze.18.3
```

This example cancels all members of cluster id 8 submitted to host machine gold.

```
llcancel gold.8
```

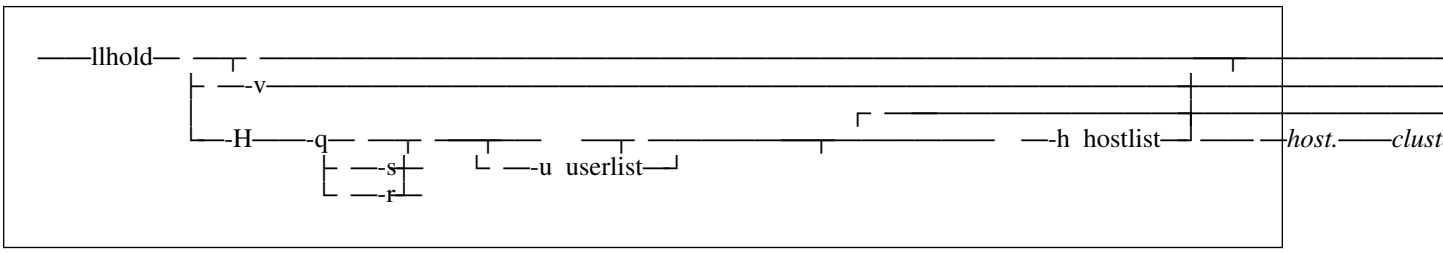
llhold - Hold or Release a Submitted Job

User Command

Purpose

Places jobs in user hold or system hold and releases jobs from both types of hold. Users can only move their own jobs into and out of user hold. Only LoadLeveler administrators move jobs into and release them from system hold.

Syntax



where

-H provides a short usage message.

-q specifies quiet mode: print no messages other than error messages.

-v outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

-s puts job(s) in system hold. Only a system administrator can use this option.

If neither **-s** nor **-r** is specified, LoadLeveler puts the job(s) in user hold.

-r releases a job from hold. A job in user hold is released unless it is also in system hold, where it remains. A job in system hold is released unless it is also in user hold, where it remains.

Only an administrator can release jobs from system hold. Only an administrator or the owner of a job can release it from user hold.

If neither **-s** nor **-r** is specified, LoadLeveler puts the job(s) in user hold.

-u *userlist* is a blank-delimited list of users. When used with the **-h** option, only the user's jobs monitored on the host machines in the *hostlist* are held or released. When used alone, only the user's jobs monitored on the **schedd** machine are held or released.

-h *hostlist* is a blank-delimited list of hosts. All jobs monitored on host machines in this list are held or released. When issued with the **-u** option, the *userlist* is used to further select jobs for holding or releasing.

When issued by a non-administrator, this option only acts upon jobs that user has submitted to the hosts in *hostlist*.

When issued by an administrator, all jobs monitored on the hosts are acted upon unless the **-u** option is also used. In that case, the *userlist* is also part of the selection process, and only jobs both submitted by users in *userlist* and monitored on the host machines in the *hostlist* are acted upon.

*host***.** is the name of the host to which the job was submitted (delimited by dot). The default is the local host.

If the job was submitted from a submit-only machine, this is the name of the **schedd** machine that sent the job to the negotiator.

cluster is the cluster id assigned to the job when it was submitted using the **llsubmit** command.

.*proc* (delimited by dot) is the proc number assigned to the job by LoadLeveler when it was submitted using the **llsubmit** command. The default is to include all members of the cluster.

Usage Notes

Only jobs in the system queue can be operated on with this command.

To insure a job is released from both system hold and user hold, the administrator must issue the command with **-r** specified to release it from system hold. The administrator or the submitting user can reissue the command to release the job from user hold.

This command will fail if:

a non-administrator attempts to move a job into or out of system hold.

a non-administrator attempts to move a job submitted by someone else into or out of user hold.

Examples

This example places jobs 23.0 and 19.1 on the local host in user hold.

```
llhold 23. 19.1
```

This example releases jobs 23.0, 19.1 and 20.3 on the local host from a hold state.

```
llhold -r 23. 19.1 2 .3
```

This example places all jobs from users abe, barbara, and carol2 in system hold.

```
llhold -s -u abe barbara carol2
```

This example releases from a hold state all jobs on host machines bronze, iron, and steel.

```
llhold -r -h bronze iron steel
```

This example releases from a hold state all jobs on host machines bronze, iron, and steel that smith submitted.

```
llhold -r -u smith -h bronze iron steel
```

llprio - Change Priority of Submitted Job

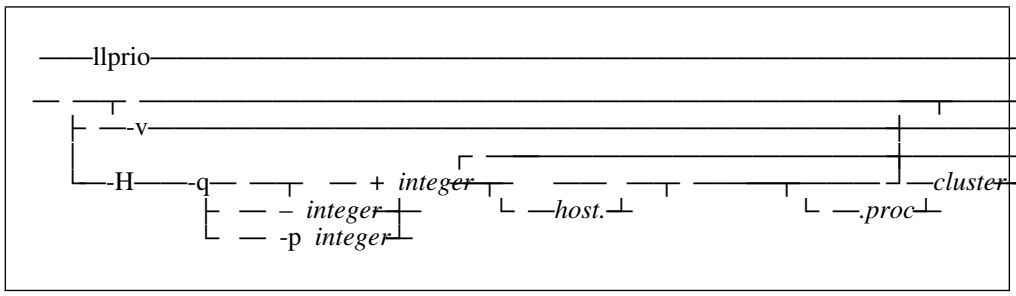
User Command

Purpose

Changes the priority of one or more jobs in the LoadLeveler queue. It does not affect running jobs. You can adjust the priority by supplying a **+** (plus) or **-** (minus) immediately followed by an *integer*value. The priority of a job ranges from 0 to 100 inclusively, with higher numbers corresponding to greater priority. The default priority is 50. Only the owner of a job or the system administrator can change the priority of that job.

Note: **llprio** does not preempt a job that is running, even if its priority is lower than other jobs.

Syntax



Parameters

where

-H provides a short usage message.

-q specifies quiet mode: print no messages other than error messages.

-v outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

+/– *integer* operates on the current priority of the job, making it higher (closer to execution) or lower (further from execution) by adding or subtracting the value of *integer*.

-p *integer* is the new absolute value for priority. The valid range is 0-100 (inclusive) where 0 is the lowest possible priority and 100 is highest.

host. is the name of the host to which the job was submitted (delimited by dot). The default is the local host.

If the job was submitted from a submit-only machine, this is

	the name of the machine where the **schedd** daemon that sent the job to the negotiator resides.
cluster	is the cluster id assigned to the job when it was submitted using the **llsubmit** command.
.*proc*	(delimited by dot) is the proc number assigned to the job when it was submitted using the **llsubmit** command.

Usage Notes

Priority changes resulting in a value less than 0 become 0.

Priority changes resulting in a value greater than 100 become 100.

Any change to a job priority applied by a user is relative only to *that user's other jobs*. If you have three jobs enqueued, you can reorder those three jobs with **llprio** but the result does not affect jobs submitted by other users, regardless of their priority and position in the queue.

Although you can use **llprio** to move one job ahead of others you have submitted, you can only overtake jobs of an equal or lower class priority, as set by the system administrator.

Examples

This example raises the priority of job 4.1 submitted to machine bronze by a value of 25.

```
llprio +25 bronze.4.1
```

This example sets the priority of job 18.4 submitted to machine silver to 100, the highest possible value.

```
llprio -p 1   silver.18.4
```

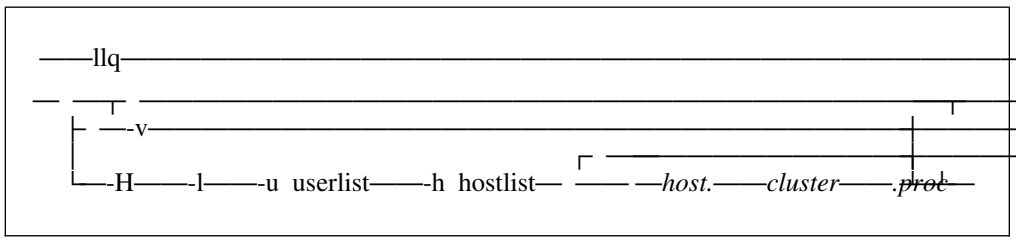
llq - Query Job Status

User Command

Purpose

Returns information about jobs in the LoadLeveler job queue. See “Output” for a description of the information generated by this command.

Syntax



Parameters

where

-H provides a short usage message.

-l specifies that a long listing be generated for each job for which status is requested. Fields included in the long listing are shown in “Output”. Only the owner of a job and the administrator can use this option.

If **-l** is *not* specified, then the standard listing, described in 90 is generated.

-v outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

*host***.** is the name of the host to which the job was submitted (delimited by dot). The default is the local host.

If the job was submitted from a submit-only machine, this is the name of the machine where the **schedd** daemon that sent the job to the negotiator resides.

cluster is the cluster id assigned to the job when it was submitted using the **llsubmit** command.

.*proc* (delimited by dot) is the proc number assigned to the job when it was submitted using the **llsubmit** command. The default is to include all members of the cluster.

-u *userlist* is a blank-delimited list of users. When used with the **-h** option, only the user's jobs monitored on the host machines in the *hostlist* are queried. When used alone, only the user's jobs monitored on the **schedd** machine are queried.

-h *hostlist* is a blank-delimited list of hosts. All jobs monitored on host machines in this list are queried. When issued with the **-u** option, the *userlist* is used to further select jobs for querying.

Usage Notes

If the **-u** or **-h** are not specified, and if no *cluster* is specified, then all jobs are queried.

The **-u** and **-h** options override the *cluster* parameters.

Examples

This example generates a long listing for job 8.2 submitted to host machine *gold*.

```
llq -l gold.8.2
```

This example generates a standard listing for all members of job cluster 12 submitted to the local host.

```
llq 12
```

Output

Standard Listing

The standard listing, generated when **-l** is *not* specified with **llq**, includes the following fields:

Id job identifier presented as *host.cluster.proc* number.

Owner userid of the job submitter.

Submitted date and time of job submission.

ST current job status, where:

R indicates job is running.

I indicates job idle. Job has been submitted, dispatched, vacated, or never run.

P indicates job pending. Job has been dispatched to a server but is not yet running.

C indicates job completed. This only occurs when multiple jobs have been submitted from a job command file.

X indicates job removed. This only occurs when multiple jobs have been submitted from a job command file or when the **llcancel** command is issued while the **schedd** host is down.

S indicates job has been put in system hold. See “llhold - Hold or Release a Submitted Job” on page 84.

H indicates job has been put in user hold. See “llhold - Hold or Release a Submitted Job” on page 84.

PRI user priority of the job, where the values are defined in the **llprio** command. See “llprio - Change Priority of Submitted Job” on page 87.

Size size of the executable submitted, in megabytes.

Class job class.

Running On if running, the machine the job is running on. This is blank when the job is not running.

The Long Listing

The long listing, returned by **llq** when the **-l** option is specified, includes these fields:

Owner userid of the job submitter.

Queue Date date and time job was received by LoadLeveler.

Status status of the job, where:

- *Running* indicates job is running on the designated server.
- *Idle* indicates job idle. Job has been submitted and dispatched but was vacated or never run.
- *Pending* indicates job pending. Job has been dispatched to a server but is not yet running.
- *Completed* indicates job completed. This only occurs when multiple jobs have been submitted from a job command file.
- *Removed* indicates job removed. This only occurs when multiple jobs have been submitted from a job command file.
- *System Hold* indicates job has been put in system hold. See "llhold - Hold or Release a Submitted Job" on page 84.
- *User Hold* indicates job has been put in user hold. See "llhold - Hold or Release a Submitted Job" on page 84.

Completion Date date and time job was completed.

User Priority priority of the job, as specified by the user.

user_sysprio user system priority of the job, where the value is defined in the configuration and administration files.

class_sysprio class priority of the job, where the value is defined in the configuration and administration files.

System Priority overall system priority of the job, where the value is defined in the configuration and administration files.

q_sysprio submission date priority of the job.

Notifications notification status for the job, where:

- *always* notification is sent through mail for all four jobs status categories below.
- *complete* notification is sent through mail only when job completes.
- *error* notification is sent through mail only when job terminates abnormally.

never	notification is never sent.
start	notification is sent through mail only upon starting or restarting job.
Virtual Image Size	of the executable that was submitted.
Checkpoint	checkpoint status (yes or no)
Hold Job Until	job is not dispatched before this date and time.
Cmd	name of the executable that was submitted.
Args	arguments that were passed to the executable.
Env	environment variables to be set before executable runs.
In	file to be used for stdin.
Out	file to be used for stdout.
Err	file to be used for stderr.
Init Working Directory	directory from which the job is run. The relative directory from which the stdio files are accessed, if appropriate.
Requirements	job requirements as specified when the job was submitted.
Preferences	job preferences as specified when the job was submitted.
Running Host	name of the machine where job is running.
Submitting Host	name of machine to which job is submitted the job.
Notify User	user to be notified by mail of job status.
Shell	shell to be used when job is run.
Class	job class as specified when job was submitted.

The following appear only when they have been specified in the job command file.

CPU limit	CPU soft and/or hard limit as specified when job was submitted.
data limit	data hard and soft limits as specified when job was submitted.
core limit	core hard and soft limits as specified when job was submitted.
file limit	file hard and soft limits as specified when job was submitted.
stack limit	stack hard and soft limits as specified when job was submitted.
RSS limit	RSS hard and soft limits as specified when job was submitted.

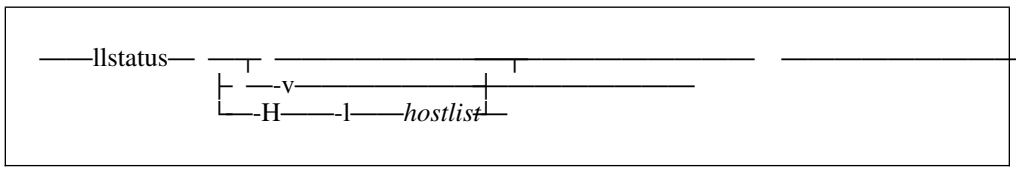
llstatus - Query Machine Status

User Command

Purpose

Returns status information about machines in the LoadLeveler pool. It does not provide status on any NQS machine.

Syntax



Parameters

where

-H provides a short usage message.

-l specifies that a long listing be generated for each machine for which status is requested. The fields included in the long listing are described in 94. If **-l** is *not* specified, the standard list, described below, is generated.

-v outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

hostlist is a blank-delimited list of host machines for which status is requested.

Usage Notes

If no *hostlist* is specified, all machines are queried.

If you have more than a few machines configured for LoadLeveler, consider redirecting the output to a file when using the **-l** flag.

Each machine periodically updates the central manager with a snapshot of its situation. Since the information returned by using **llstatus** is a collection of such snapshots, all taken at varying times, the total picture may not be completely consistent.

Examples

This example requests a long status listing for host machines silver and gold machines.

```
llstatus -l silver gold
```

Output

The Standard Listing

The standard listing, generated by **llstatus** when **-l** is *not* specified, includes the following fields:

Name hostname of the machine.

Run number of jobs submitted to this machine that are running on this machine or else where in the cluster.

Total total number of jobs submitted to this machine that are currently in queue.

State state of the machine, where:

- *Busy* Maximum number of jobs are running on this machine.
- *Down* LoadLeveler is not running on this machine. A machine obtains this state if the daemons on that machine have not been able to report their status to the central manager. This may occur because the machine is actually down or there is a network failure.
- *Drain* running jobs continue but no additional jobs are started on this machine.
- *Flush* all jobs on this machine are flushed. No new jobs will be accepted.
- *Idle* machine is running no jobs.
- *None* LoadLeveler is running on this machine but no jobs can be run.
- *Running* machine is running one or more jobs and is capable of running more.
- *Suspend* machine has suspended all jobs due to a system suspend.

LdAvg Berkeley one-minute load average on this machine.

Idle number of seconds since keyboard or mouse activity in a login session was detected. Highest number displayed is 9999.

Arch hardware architecture of machine as listed in configuration file.

OpSys operating system on this machine.

The Long Listing

The long listing, returned by **llstatus** when the **-l** option is specified, includes these fields:

Name hostname of the machine.

OpSys operating system on this machine.

Arch hardware architecture of machine as listed in configuration file.

Machine fully qualified name of the machine.

START the expression, defined following C conventions in the configuration file, that evaluates to true or false (T/F). This determines whether jobs can be started on this machine.

SUSPEND the expression, defined following C conventions in the configuration file, that evaluates to true or false (T/F). This determines whether running jobs should be suspended on this machine.

CONTINUE the expression, defined following C conventions in the configuration file, that evaluates to true or false (T/F). This determines whether suspended jobs are continued on this machine.

VACATE the expression, defined following C conventions in the configuration file, that evaluates to true or false (T/F). This determines whether suspended jobs are vacated on this machine.

KILL the expression, defined following C conventions in the configuration file, that evaluates to true or false (T/F). This determines whether vacated jobs are killed on this machine.

SYSPRIO actual expression that determines overall system priority of the job, where the value is defined in the configuration file.

Users number of different users with jobs submitted to this machine.

Running number of jobs submitted to this machine that are running.

Idle number of jobs submitted to this machine that are not running.

ScheddAvail flag indicating if machine is running a **schedd** daemon (0=no, 1=yes).

StartdAvail flag indicating if machine is running a **startd** daemon (0=no, 1=yes).

State state of the machine, where:

- *Busy* Maximum number of jobs are running on this machine.
- *Down* LoadLeveler is not running on this machine. A machine obtains this state if the daemons on the machine have not been able to report their status to the central manager. This may occur because the machine is actually down or there is a network failure.
- *Drain* running jobs continue but no additional jobs are started on this machine.
- *Flush* all jobs on this machine are flushed. No new jobs will be accepted.
- *Idle* machine is running no jobs.
- *None* LoadLeveler is running on this machine but no jobs can be run.
- *Running* machine is running one or more jobs.

Suspend machine has suspended all jobs due to a system suspend.

Entered Current State
date and time when machine state was set.

Virtual Memory available swap space, in kilobytes, on this machine.

Memory physical memory, in megabytes, on this machine.

Disk available space, in kilobytes (less 512KB) in LoadLeveler's execute directory on this machine.

Keyboard Idle number of seconds since last keyboard or mouse activity.

Cpus number of CPUs on this machine.

LoadAvg Berkeley one-minute load average on machine.

Subnet TCP/IP subnet that this machine resides on.

Class set of available classes (those not currently in use) on this machine as defined in the configuration file.

Initiators set of all classes on this machine, both those in use and those not in use.

Feature set of features available on this machine.

Subnet TCP/IP subnet that this machine resides on.

Max_Jobs_Running
maximum number of jobs that any single user can run simultaneously on this machine.

Max_Starters maximum number of jobs that can run simultaneously on this machine.

Config Time Stamp
date and time of last (re)configuration.

time stamp date and time the central manager last received a status update from this machine.

llsubmit - Submit a Job

User Command

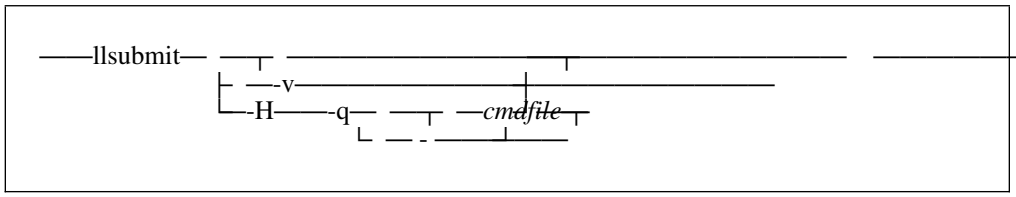
Purpose

Submits a job to LoadLeveler to be dispatched based upon job requirements in the job command file.

You can submit both LoadLeveler jobs and NQS jobs. To submit NQS jobs, the job command file must contain the shell script to be submitted to the NQS node.

Users with **uid** or **gid** equal to 0 are not allowed to issue the **llsubmit** command.

Syntax



Parameters

Where

-H	provides a short usage message.
-q	specifies quiet mode: print no messages other than error messages.
-v	outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
cmdfile	is the name of the job command file containing LoadLeveler commands.
-	specifies that LoadLeveler commands that would normally be in the job command file, are read from stdin. When entry is complete, press Ctrl-D to end the input.

Examples

In this example, a command file named *qtrlyrun.cmd* is submitted.

```
llsubmit qtrlyrun
```

NQS Scripts

Scripts originally written for NQS that contain NQS options are acceptable to LoadLeveler. The options are mapped as closely as possible to the features provided by LoadLeveler, but the exact function is not always available. NQS options map to LoadLeveler as follows:

a	startdate
e	error
ke	ignored
ko	ignored
lc	core_limit
ld	data_limit
lf	file_limit
lm	rss_limit
lM	ignored
ln	ignored
ls	stack_limit
lt	cpu_limit
lT	ignored
lv	ignored
lw	ignored
mb	notification (always)
me	notification (complete)
mu	notify_user
nr	restart = no
o	output
p	user_priority
q	class
r	ignored
re	ignored
ro	ignored
s	shell
x	environment = copyall
z	suppresses messages but not mail

Appendix A. Planning for Installation

This section discusses how to plan for your installation. For specific instructions how to install LoadLeveler, see the Memo to Users or the README file. Before continuing, read the following paragraph to briefly familiarize yourself with the installation process.

First, the **llinit** command creates the **loadl** home directory in $HOME. Ensure that your HOME environment variable is set to **loadl**'s home before running **llinit**. Then, the **llinit** command places the log, execute, and spool directories under the local directory and the administration and local configuration files in **loadl**'s home directory. You may, however, change these locations by changing the associated paths in the global configuration file. The global configuration file must reside in **loadl**'s home directory.

Planning for Installation

Planning for installation requires two main tasks; ensuring you have the correct software and hardware installed and allocating the correct amount of disk space for directories.

Software and Hardware Requirements

Your LoadLeveler cluster can be made up of the any combination of RISC System/6000, Sun, SGI, and HP workstations and the SP1. In addition, you may have one or more submit-only workstations. The software and hardware requirements listed below apply to all platforms unless specifically stated otherwise.

Note: The customer is responsible for licensing and installing these products and integrating them with LoadLeveler.

Software Requirements

Requirement	RISC System/6000	Sun	SGI	HP
AIXWindows Environment Version 1 Release 2 (program number 5601-257). (Only required if using the graphical user interface)	√			
AIX Version 3 Release 2	√			
IRIX 4.0.5 or later compatible release of IRIX			√	
SunOS 4.1.2 or later compatible releases		√		
OpenWindows 3.0 with a patch for its X11/NeWS server. The Patch_Id is I00444-37. (Only required if using the graphical user interface)		√		
HP-UX 9.01 or later compatible releases				√

Hardware Requirements

The machine on which you run the GUI to LoadLeveler should have an all-points-addressable 1284 x 1024 pixel color display.

Other Requirements

Have **rsh** or **remsh** privileges on all machines that you plan on using **llctl** to start.

If the release directory will be shared among machines in the cluster using NFS, you must

- Either allow all files to be world-readable (by default), or the file system must be exported with root permission to all hosts that will be running LoadLeveler.
- When mounting the filesystem, the flag -nosuid must not be used.

LoadLeveler log files should be stored locally at the nodes for best performance.

Disk Space Requirements

After installation, certain directories require specific amounts of disk space.

For the RISC System/6000

Directory	Bytes
For AFS: release directory (/usr/lpp/LoadL/afs)	21MB
For NFS: release directory (/usr/lpp/LoadL/nfs)	19MB
local directory	15MB
home directory	no limits unless same as release or local directory
info directory (/usr/lpp/info/data/LoadL)	3MB
For submit-only machine release directory (/usr/lpp/LoadL/nfs_so) and/or release directory (/usr/lpp/LoadL/afs_so)	2MB

For the SGI

Directory	Bytes
release directory (/usr/lpp/LoadL/sginfs)	19MB
local directory	15MB
home directory	no limits unless same as release or local directory
For submit-only machine release directory (/usr/lpp/LoadL/sginfs_so)	2MB

For the Sun

Directory	Bytes
release directory (/usr/lpp/LoadL/sunosnfs)	18MB
local directory	15MB
home directory	no limits unless same as release or local directory
For submit-only machine release directory (/usr/lpp/LoadL/sunosnfs_so)	2MB

For the HP

Directory	Bytes
release directory (/usr/lpp/LoadL/hpnfs)	19MB
local directory	15MB
home directory	no limits unless same as release or local directory
For submit-only machine release directory (/usr/lpp/LoadL/hpnfs_so)	2MB

Note: Ordinarily, you will have only one release directory for either the AFS or NFS version of LoadLeveler.

The AFS version of LoadLeveler should be used if user's jobs require AFS file access. Otherwise, NFS version 2 of LoadLeveler may be used.

The local directory must be unique to each machine in the cluster and must not be shared by any other machines in the cluster.

After Installation

The installation process builds a release directory and deposits the LoadLeveler code in this directory. Then, the **llinit** command creates three directories that are subordinate to the local directory. You specify the local directory in the **llinit** command. If you don't specify a local directory, these three directories will be subordinate to the home directory. These three directories are:

log

spool

execute

After creating the directories, **llinit** sets the permissions for each of these directories. The directories will have the following permissions:

Directory	World-readable	World-writeable
../log	√	
../spool	√	√
../execute	√	√

After installation, the following product directories reside on disk. Note, both AFS and NFS versions are shown for completeness. The installation process creates only those directories required to service the LoadLeveler options specified with the installp command.

For the RISC System/6000

Directory	Description
/usr/lpp/LoadL/afs/bin and/or /usr/lpp/LoadL/nfs/bin	Part of the release directory containing daemons, commands, and other binaries
/usr/lpp/LoadL/afs/lib and/or /usr/lpp/LoadL/nfs/lib	Part of the release directory containing product libraries and sample stanzas
/usr/lpp/LoadL/afs/man and/or /usr/lpp/LoadL/nfs/man	Part of the release directory containing man pages
/usr/lpp/LoadL/afs/samples and/or /usr/lpp/LoadL/nfs/samples	Part of the release directory containing sample administration and configuration files and sample jobs
local directory	Contains log, spool, and execute directories for each machine in the cluster
home directory	Contains administration and configuration files, and symbolic links to the release directory
info directory (/usr/lpp/info/data/LoadL)	Contains LoadLeveler documentation.

For the Sun

Directory	Description
/usr/lpp/LoadL/sunosnfs/bin	Part of the release directory containing daemons, commands, and other binaries.
/usr/lpp/LoadL/sunosnfs/lib	Part of the release directory containing product libraries and sample stanzas
/usr/lpp/LoadL/sunosnfs/man	Part of the release directory containing man pages
/usr/lpp/LoadL/sunosnfs/samples	Part of the release directory containing sample administration and configuration files and sample jobs
local directory	Contains log, spool, and execute directories for each machine in the cluster
home directory	Contains administration and configuration files, and symbolic links to the release directory

For the SGI

Directory	Description
/usr/lpp/LoadL/sginfs/bin	Part of the release directory containing daemons, commands, and other binaries
/usr/lpp/LoadL/sginfs/lib	Part of the release directory containing product libraries and sample stanzas
/usr/lpp/LoadL/sginfs/man	Part of the release directory containing man pages
/usr/lpp/LoadL/sginfs/samples	Part of the release directory containing sample administration and configuration files and sample jobs
local directory	Contains log, spool, and execute directories for each machine in the cluster
home directory	Contains administration and configuration files, and symbolic links to the release directory

For the HP

Directory	Description
/usr/lpp/LoadL/hpnfs/bin	Part of the release directory containing daemons, commands, and other binaries
/usr/lpp/LoadL/hpnfs/lib	Part of the release directory containing product libraries and sample stanzas
/usr/lpp/LoadL/hpnfs/man	Part of the release directory containing man pages
/usr/lpp/LoadL/hpnfs/samples	Part of the release directory containing sample administration and configuration files and sample jobs
local directory	Contains log, spool, and execute directories for each machine in the cluster
home directory	Contains administration and configuration files, and symbolic links to the release directory

For the Submit-Only

Directory	Description
For the RISC System/6000 /usr/lpp/LoadL/afs_so/bin and/or /usr/lpp/LoadL/nfs_so/bin	Part of the release directory containing commands.
For the Sun /usr/lpp/LoadL/sunosnfs_so/bin	
For the SGI /usr/lpp/LoadL/sginfs_so/bin	
For the HP /usr/lpp/LoadL/hpnfs_so/bin	
For the RISC System/6000 /usr/lpp/LoadL/afs_so/man and/or /usr/lpp/LoadL/nfs_so/man	Part of the release directory containing man pages
For the Sun /usr/lpp/LoadL/sunosnfs_so/man	
For the SGI /usr/lpp/LoadL/sginfs_so/man	
For the HP /usr/lpp/LoadL/hpnfs_so/man	
For the RISC System/6000 /usr/lpp/LoadL/afs_so/samples and/or /usr/lpp/LoadL/nfs_so/samples	Part of the release directory containing sample administration and configuration files.
For the Sun /usr/lpp/LoadL/sunosnfs_so/samples	
For the SGI /usr/lpp/LoadL/sginfs_so/samples	
For the HP /usr/lpp/LoadL/hpnfs_so/samples	
home directory	Contains administration and configuration files.

List of Program Material

The distribution material contains all of the basic machine-readable material for LoadLeveler. There are no optional machine-readable materials.

The following is the list of basic program publications for LoadLeveler:

IBM LoadLeveler: Quick Reference (SX23-1048)

IBM LoadLeveler: Administration and Installation Planning (SH26-7220)

IBM LoadLeveler: User's Guide (SH26-7226)

IBM LoadLeveler: Licensed Program Specification for the RISC System/6000 and the SP1 (GH26-7222)

IBM LoadLeveler: Licensed Program Specification for the Sun (GH26-7232)

IBM LoadLeveler: Licensed Program Specification for the SGI (GH26-7233)

IBM LoadLeveler: Licensed Program Specification for the HP (GH23-0040)

Statement of Program Support

The distribution tape contains the most complete and accurate code and examples. If you incur any problems and want to report them, apply this tape before contacting the Simplified Support Interface to IBM (SSITI).

If you encounter problems when using LoadLeveler, you should consult Appendix B for procedures to isolate your problem and report any suspected code defects.

Program service for Version 1 Release 1.1 will be available until discontinued by IBM upon six months written notice.

Appendix B. Diagnosing LoadLeveler Problems

Should any of these messages or symptoms appear, use the procedures that follow to investigate and eliminate the cause of the problem. Refer to “Recording and Controlling Log Files” on page 26 for information on setting up error logs.

Daemon LoadL_master Will Not Run

If the **master** daemon will not run, go to the node where **LoadL_master** will not start and type from the command line:

```
LoadL_master -t
```

This will generate messages that may help to diagnosis the problem. In addition, ensure that:

1. the release directory is properly specified in the configuration files.
2. the administration file exists and is properly defined in the configuration file.
3. the central manager is correctly defined in the administration file.
4. the log directories are correctly defined in the configuration file.
5. the permissions are set as follows:
   a. the **spool** and **execute** subdirectories are set to 777
   b. the **log** subdirectory is set to 775.
6. the **LoadL_master** setuid is a root executable.

Job Submission Rejected

The local directory may be out of space. The administrator should increase the LoadLeveler local file system or purge jobs from the local spool.

Queued LoadLeveler Jobs Will Not Run

If either a serial or parallel job is not running:

1. Allow a few minutes for the central manager to be updated and then the job may be dispatched. The information the central manager has about machines and jobs may not be current. By allowing a few minutes for the central manager to be updated, the job may be dispatched. This time limit (a few minutes) depends upon the polling frequency set in the **LoadL_config** file. The default polling frequency is five minutes.
2. On the GUI, compare the job requirements to the machine details, especially **Arch**, **OpSys**, and **Class**. Ensure that the spelling matches.
3. Examine the configuration files (both **LoadL_config** and **LoadL_config.local**) to determine the **START** control function expression used by LoadLeveler to start a job. As a problem determination measure, set the START and SUSPEND values, as shown in this example:

   ```
   START: T
   SUSPEND: F
   ```

If a parallel job is not running, check that the:

1. Number of parallel machines requested in the job command file are available. The number of parallel machines requested must be less than or equal to the

number of parallel machines in at least one of the batch parallel sub-pools associated with the SP1 **startd** machines.

2. **parallelmachines** keyword is being correctly used in the job command file.
3. Local configuration file of the **startd** SP1 machine and make sure that the POOL_ID is valid. A valid POOL_ID is one that exists in the node_types file.
4. SP1 Resource Manager is running on the primary node, if the switch is operational, or on the control node.

llstatus Indicates that a Machine is Down but llq Indicates a Job is Running on The Machine

If a machine fails while a job is running on the machine, the central manager does not change the status of any job on the machine. When the machine comes back up the central manager will be updated.

Insufficient Swap Space

Determine how much swap space is available using the UNIX **lsps -a** command. By default, each LoadLeveler job requires 5M bytes, unless specified in either of the configuration files (**LoadL_config** or **LoadL_config.local**) with the **RESERVED_SWAP = nnnn** keyword.

Warning: Error Receiving Job Data. The Query May be Incomplete.

1. Ensure that you have a common name space, so that each userid name and number effectively match in /**etc**/**passwd** on all participating LoadLeveler machines.
2. There may be an incompatibility between LoadLeveler installation levels.
3. **Note. Taking the following action is quite severe and will cause the loss of any queued jobs.** Stop LoadLeveler, go into the local directory spool directory and remove all files, then restart LoadLeveler

Getting Help from IBM

Should you require help from IBM in resolving a LoadLeveler problem, you can get assistance by calling IBM Support. Before you call, be sure you have the following information:

1. Your access code (customer number).
2. The LoadLeveler product number for the:

   RISC System/6000 and SP1 is 5765-145

   Sun is 5765-227