Dino Quintero

IBM LoadLeveler to IBM Platform LSF Migration Guide

Introduction and overview

This IBM® Redpaper™ publication shows IBM Tivoli® Workload Scheduler (TWS) LoadLeveler® (LoadLeveler) users how to migrate their workloads to IBM Platform Load Sharing Facility (LSF®). This document does not provide a full description of LoadLeveler or LSF. For a more complete description of LSF, see *Administering IBM Platform LSF*, SC27-5302, and *IBM Platform LSF Configuration Reference*, SC27-5306.

LoadLeveler is a parallel job scheduling system that enables users to run more jobs in less time. It does so by matching each job's processing needs and priority with the available resources, thereby maximizing resource use. LoadLeveler also provides a single point of control for effective workload management, and supports high-availability configurations.

For additional information about LoadLeveler, see the following website:

http://www-01.ibm.com/software/tivoli/products/scheduler-loadleveler/

LSF is a powerful workload manager for demanding, distributed, and mission-critical high-performance computing (HPC) environments. When you want to address complex problems, simulation scenarios, extensive calculations, or anything else that needs compute power, and then run them as jobs, you can submit them using LSF.

For additional details about IBM Platform Computing software solutions, including LSF, see the following IBM Redbooks® publications:

- *IBM Platform Computing Solutions*, SG24-8073
- *IBM Platform Computing Integration Solutions*, SG24-8081

The LSF family consists of a suite of products that addresses many common client workload management requirements. LSF has a broad set of capabilities, one of the best in the industry. What differentiates LSF from many competitors is that all of LSF’s components are tightly integrated and fully supported.

For additional information about LSF, see the following website:

http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/index.html

For a use-case scenario using LSF, see *IBM Platform LSF Implementation Scenario in an IBM iDataPlex Cluster*, REDP-5004. This IBM Redpaper publication provides information to help clients move smoothly and efficiently from LoadLeveler-managed clusters to LSF-managed clusters. All LSF features mentioned in this paper are based on LSF V9.1.1.

LoadLeveler and LSF concepts and terminology

Figure 1 shows the concepts and terminology of the workload management solution.

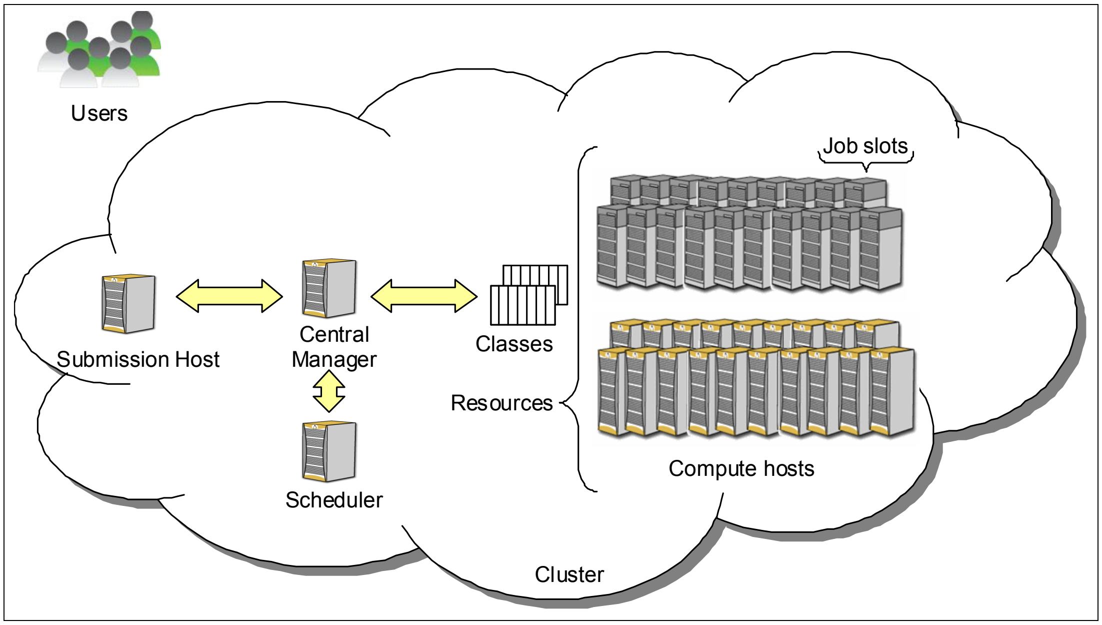


Figure 1 LoadLeveler concepts and terminology

Figure 2 shows the LSF concepts and terminology used within its workload management solution.

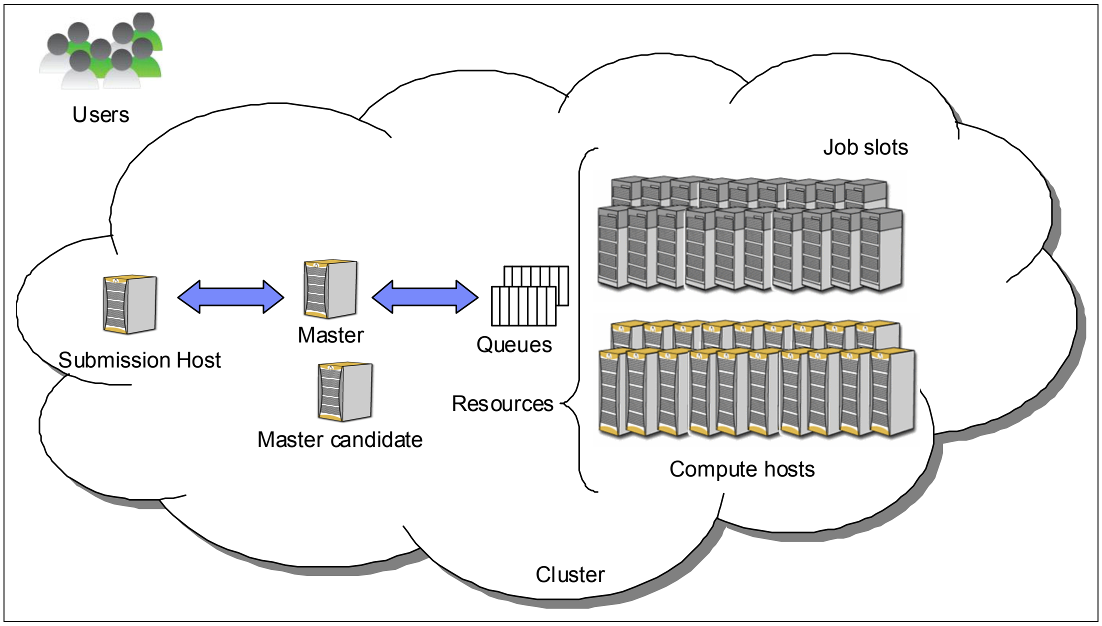


Figure 2   LSF concepts and terminology

Table 1 shows the terminology used in LoadLeveler and LSF.

Table 1   LoadLeveler and LSF terminology

LoadLeveler	Description	LSF
Cluster	A collection of networked resources (for example, compute hosts)	Cluster
Central manager host	Master host that controls the rest of the hosts in the cluster	Master host
Scheduler host	A host designated to identify candidate compute hosts that match job requirements	Master host
Compute host	A host within the cluster that submits and runs jobs	Compute host
Submission host	A host within the cluster that submits jobs (often referred to as a submit-only host)	Submission host
Resource	Shared host resources (for example, central processing unit (CPU), memory, and local temp storage)	Resource
Job	A unit of work run in the cluster	Job
Class	► A cluster-wide job “container”, where jobs wait until they are scheduled and dispatched to compute hosts ► Center of scheduling, with priorities and policies	Queue

LoadLeveler	Description	LSF
Job slot	► Basic allocation unit used in LSF ► Used to control concurrent running tasks per host ► Can be more than one per physical processor ► Can be more than one job slot per job	Job slot
Command file	Set of commands or directives that embody the job	Spool file
Requirements	Specifies criteria that a target job host must satisfy (for example, operating system, architecture, load, and machine)	Select (selection)
Preferences	Specifies how the target job hosts should be sorted	Order (ordering)
Resources	Specifies the expected (CPU and memory) resource consumption of the job on the job host	Rusage (resource usage)
Job type	Specifies the locality of a job (if or how a job should span across multiple hosts)	Span (job spanning)
User	A user account that has permission to submit jobs	User
Group	A group of undefined users used for control and accounting	Arbitrary group membership imported to LSF via egroup
	A group of predefined users used for control	Accounting done by the LSF user group
Account	Usage accounting label (tag)	Project
	A group of hosts that can be easily referenced as a unit	Host group
Job step	Two sub-units of work with different implementation in LSF and LoadLeveler (LoadLeveler is more like a chain of dependent jobs)	Job array index
Admin	A user account with permissions to perform all administrative operations in the cluster	Primary admin
	A user account with permissions to perform administrative operations on all jobs and queues in the cluster, but not to change `config` files	Cluster admin
	► Means to enable stakeholders to perform some administrative tasks ► Reduces cluster administration load	Delegated administrative rights
	A user account with administrative permissions limited to a specified queue	Queue admin

LoadLeveler	Description	LSF
	► A user account that has administrative permissions limited to controlling all jobs that are submitted by users who are members of a specified user group ► A user group administrator empowered to manage internal project and priority changes dynamically, such as modifying membership and fair share within the group	User group admin
	A user account that has administrative permissions limited to a specified compute host or host group	Host admin
	A definition of common application parameters for the same type of jobs (for example, resource limit, job control, pre-execution, post-execution, and so on)	Application profile
Grid	A federation of distributed, often heterogeneous computer resources from multiple administrative domains (for example, clusters) organized to reach a common goal	Grid

LoadLeveler to LSF migration

There are three main stages involved in migrating a LoadLeveler cluster to an LSF cluster:

- Set up a simple LSF cluster to shadow the LoadLeveler cluster.
- Migrate advanced features.
- Move users and users' jobs from LoadLeveler to LSF (job management).

The following sections provide detailed information about the migration from LoadLeveler to LSF.

Setting up a simple LSF cluster to shadow the LoadLeveler cluster

The migration from LoadLeveler to LSF begins with setting up an LSF cluster to shadow the LoadLeveler cluster. In the LSF cluster configuration, you define the following elements:

- Cluster name
- Cluster administrators
- Users
- User groups
- Hosts
- Host groups
- Queue names

All queues are first come, first served (FCFS). Table 2 illustrates a typical cluster setup, and illustrates how to map basic LoadLeveler configurations in `LoadL_config` and `LoadL_admin` to the corresponding LSF configurations.

Table 2 Mapping basic configurations

In `LoadL_config`:
`ARCH = i386`
`OPSYS = RedHat5`
	In `lsf.shared`:
	`Begin HostType`
	`TYPENAME`
	`LINUX86`
	`End HostType`

`LOADL_ADMIN = loadl root`	**In `lsf.cluster.<`*`cluster_name`*`>`:**
	`Begin   ClusterAdmins`
	`Administrators = lsfadmin`
	`End    ClusterAdmins`

`MACHINE_AUTHENTICATE = TRUE`	**In `lsf.cluster.<`*`cluster_name`*`>`:**
	`Set hosts IP range:`
	`Begin Parameters`
	`LSF_HOST_ADDR_RANGE=*.*.*.*`
	`FLOAT_CLIENTS_ADDR_RANGE=*.*.*.*`
	`FLOAT_CLENTS=10`
	`End Parameters`
	`In lsf.conf:`
	`LSF_AUTH_DAEMONS is unset`

`BASEDIR = /llconf/test/`
`MYPATH = /llconf/test/$(host)`
`RELEASEDIR = /opt/ibmll/LoadL/full`
`LOG = $(MYPATH)/log`
	In `lsf.conf`:
	`LSF_TOP=/home/user1/base`
	`LSF_MACHDEP=/home/user1/base/9.1`
	`LSF_LOGDIR=/home/user1/base/log`

In `LoadL_admin`:
`MASTER_STREAM_PORT = 9616`
`NEGOTIATOR_STREAM_PORT = 9614`
`SCHEDD_STREAM_PORT = 9605`
`STARTD_STREAM_PORT = 9611`
`COLLECTOR_DGRAM_PORT = 9613`
`STARTD_DGRAM_PORT = 9615`
`MASTER_DGRAM_PORT = 9617`
	In `lsf.conf`:
	`LSF_LIM_PORT=57869`
	`LSF_RES_PORT=46878`
	`LSB_MBD_PORT=56881`
	`LSB_SBD_PORT=46882`

`SCHEDULER_TYPE = BACKFILL`
`ACCT = A_ON A_DETAIL`
	`BACKFILL` is enabled on the queue level.
	Job accounting is supported automatically by LSF.

`MACHINE_UPDATE_INTERVAL = 30`	**In `lsf.cluster.<`*`cluster_name`*`>`:** `EXINTERVAL=30`

`FLOATING_RESOURCES =`
`FloatingLicenseX(5)`
`FloatingLicenseZ(2)`
`SCHEDULE_BY_RESOURCES = ConsumableCpus`
`LicenseA FloatingLicenseX`
`ConsumableMemory`
`PUBLISH_OBITUARIES = TRUE`
`OBITUARY_LOG_LENGTH = 25`
`RESTARTS_PER_HOUR = 12`
	In `lsf.shared`, `lsf.cluster.<`*`cluster_name`*`>` and `lsf.shared`:
	`Define static shared license resources`
	`FLoatingLicenseX and FloatingLicenseZ`

hosta.example.com: type = machine	**Define this host in `lsf.cluster.<cluster_name>` and `lsb.hosts`:** Begin Host HOSTNAME model type server r1m mem swp RESOURCES hosta.example.com ! ! 1 3.5 () () () End Host

classA: type = class
wall_clock_limit =00:10:00
	In `lsb.queues`:
	Begin Queue
	QUEUE_NAME = classA
	RUNLIMIT = 10:00
	USERS = user1
	End Queue

HPC_GROUP: type = group user1: type = user default_group = HPC_GROUP default_class = classA	**In `lsb.users`:** Begin UserGroup GROUP_NAME GROUP_MEMBER USER_SHARES HPC_GROUP (user1) () End UserGroup Begin User USER_NAME MAX_PEND_JOBS user1 10000 End User

Verifying cluster startup

If the cluster can be started up correctly, the next step is to convert some of the fundamental cluster resources from LoadLeveler to LSF, as shown in Example 1.

Example 1 Converting some fundamental cluster resources from LoadLeveler to LSF

```
$ llstatus
Name                     Schedd InQ  Act Startd Run LdAvg Idle Arch OpSys
hosta.example.com        Avail   0   0   Idle    0  0.05  0    i386 RedHat5.1

i386/RedHat5.1               1 machines      0  jobs      0  running tasks
Total Machines               1 machines      0  jobs      0  running tasks

The Central Manager is defined on hosta.example.com
The BACKFILL scheduler is in use

$ bhosts
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
Hosta               ok            -      8      0      0      0      0      0
```

Mapping resources

This section provides information about mapping LoadLeveler machines to LSF hosts. In LoadLeveler, machines are defined in the `LoadL_admin` file. A machine derives its attribute definition from its corresponding elements:

- Machine stanza
- Machine group stanza
- Machine sub-stanza
- Default machine stanza

To migrate machine definitions from LoadLeveler to LSF, you need to convert LoadLeveler stanza definitions for machines to LSF host configurations.

Example 2 shows a typical machine configuration sample. It will be used to demonstrate how to map LoadLeveler machines to LSF hosts.

Example 2   Typical machine configuration

```
$ vim LoadL_config
CENTRAL_MANAGER_LIST = hostb
RESOURCE_MGR_LIST = hostb
$ vim LoadL_admin
default: {
        type = machine_group
        schedd_runs_here = false
        schedd_host = false
xc3_mg1: {
        type = machine_group
        machine_list = hostb+1
        hostb: {
                type = machine
                schedd_runs_here = true
                schedd_host = true
        }
...
```

In Example 2 on page 8, `hostb` and `hostc` are the two machines in the cluster. The `hostc` machine is a compute node. The `hostb` machine also serves as the LoadLeveler central manager, and LSF scheduler daemon (`schedd`) host.

> **Tip:** Before migrating your LoadLeveler host configuration to LSF, read about the following items in *IBM Platform LSF Configuration Reference*, SC27-5306:
>
> - The **`LSF_MASTER_LIST`** parameter in the `lsf.conf` file
> - The host section in the `lsf.cluster.cluster_name` file

Example 3 shows the configuration converted from the previous LoadLeveler sample machine configuration.

Example 3   Converted LoadLeveler sample machine configuration

```
$ vim $LSF_ENVDIR/lsf.conf
LSF_MASTER_LIST=hostb
# NOTE: replace <cluster_name> with your defined name.
$ vim $LSF_ENVDIR/lsf.cluster.<cluster_name>
Begin   Host
HOSTNAME  model    type        server r1m  mem  swp  RESOURCES    #Keywords
hostb  !        !            1      3.5   ()   ()   (mg)
hostc  !        !            1      3.5   ()   ()   ()
End     Host
```

LSF does not provide a `machine_list` feature in LoadLeveler's machine group stanza to define a list of multiple machines quickly. You have to define them one at a time in the host section of the `lsf.cluster.<cluster_name>` file.

Machine groups

LSF does not support the LoadLeveler machine group feature. LSF provides host groups, host partitions, and compute units in the `lsb.hosts` file. These features are mostly for scheduling purposes instead of configuration simplification.

Users

Example 4 shows a typical LoadLeveler user configuration.

Example 4   LoadLeveler user

```
$ vim LoadL_admin
user1:        type = user
              default_class = classA
```

Example 5 shows the corresponding LSF user configuration.

Example 5   LSF user

```
$ vim lsb.users
Begin User
USER_NAME        MAX_PEND_JOBS
user1            10000
End User
```

User Groups

Example 6 shows a typical LoadLeveler user group configuration.

Example 6   LoadLeveler user group

```
$ vim LoadL_admin
HPC_GROUP:type = group
```

Example 7 shows the corresponding LSF user group configuration.

Example 7   LSF user group

```
$ vim lsb.users
Begin UserGroup
GROUP_NAME  GROUP_MEMBER  USER_SHARES
HPC_GROUP   (user1)         ()
End UserGroup
```

Job classes

A *job class* in LoadLeveler is equivalent to a *job queue* in the `lsb.queues` file. A typical LoadLeveler class definition is shown in Example 8.

Example 8   Typical LoadLeveler class definition

```
$ vim LoadL_config
...
CENTRAL_MANAGER_LIST = hostb
RESOURCE_MGR_LIST = hostb
...
$ vim LoadL_admin
...
short: {
        type = class
        class_comment = "Short job which less than 30 minutes."
        priority = 20
        …
}
...
urgent: {
        type = class
        class_comment = "Very urgent job which has relatively higher priority."
        priority = 100
        max_jobs = 4
        …
}
...
xc3_mg1: {
        type = machine_group
        machine_list = hostb+1
        max_starters = 32
        class = short(2) urgent(2) …
        …
}
...
```

Example 9 shows the corresponding LSF queue configuration.

Example 9   LSF queue configuration

```
# NOTE: replace <cluster_name> with your defined name.
$ vim $LSF_ENVDIR/lsbatch/<cluster_name>/configdir/lsb.queue
...
Begin Queue
QUEUE_NAME    = short
PRIORITY     = 20
HOSTS        = hostb hostc
HJOB_LIMIT   = 2
INTERACTIVE  = NO
DESCRIPTION  = Short job which less than 30 minutes.
End Queue

Begin Queue
QUEUE_NAME   = urgent
PRIORITY     = 100
HOSTS        = hostb hostc
HJOB_LIMIT   = 2
INTERACTIVE  = NO
DESCRIPTION  = Very urgent job which has relatively higher priority.
End Queue
...
$ vim $LSF_ENVDIR/ lsbatch/<cluster_name>/configdir/lsb.hosts
...
Begin Host
HOST_NAME MXJ   r1m     pg    ls    tmp  DISPATCH_WINDOW  # Keywords
hostb    32    ()      ()     ()     ()      ()
hostc    32    ()      ()     ()     ()      ()
End Host
...
```

```
$ vim $LSF_ENVDIR/ lsbatch/<cluster_name>/configdir/lsb.resources
...
Begin Limit
NAME = urgent_queue_limit
JOBS = 4
QUEUES = urgent
End Limit
...
```

Note: Some LoadLeveler class attributes can be covered by LSF features not defined in the `lsb.queues` file (for example, General Limits or Guaranteed service level agreement (SLA) in `lsb.resources`). For more information about these features, see *Administering IBM Platform LSF*, SC27-5302, and *IBM Platform LSF Configuration Reference*, SC27-5306.

Remember that after you make changes to the LSF configuration file, you must run the **`lsadmin reconfig`** command and the **`badmin reconfig`** command.

Migrating advanced features

After you complete this step, the LSF cluster should be able to satisfy the resource management and job scheduling policy needs.

Resource management

This section describes the resource management characteristics for LoadLeveler and LSF.

Resource enforcement

LSF provides comprehensive resource enforcement on all supported platforms: Intel and AMD x86-64, Intel, IBM POWER®, and Oracle SPARC. Example 10 shows how to configure LoadLeveler to support CPU and physical memory resource enforcement.

Example 10 LoadLeveler configuration to support CPU and memory resources

```
$ vim LoadL_config
...
SCHEDULE_BY_RESOURCES = ConsumableCpus ConsumableMemory
ENFORCE_RESOURCE_USAGE = ConsumableCpus ConsumableMemory
ENFORCE_RESOURCE_MEMORY = true
ENFORCE_RESOURCE_POLICY = share
...
$ vim job1.cmd
#! /bin/sh
...
# @ node = 2
# @ total_tasks = 2
# @ resources = ConsumableCpus(1) ConsumableMemory(800 mb)
# @ queue
...
$ llsubmit job1.cmd
```

The corresponding LSF conversion to support CPU and physical memory resource enforcement is shown in Example 11.

Example 11 LSF configuration to support CPU and physical memory resources

```
$ vim $LSF_ENVDIR/lsf.conf
...
LSB_MEMLIMIT_ENFORCE = y
...
$ bsub -n 2 -R "span[ptile=1]" -M 800 …
```

On these platforms, LSF supports either operating system-level generic job resource enforcement, or integrations with operating system functions, such as cpusets, cgroups, CPU affinity, and so on.

Process tracking

LoadLeveler supports process tracking functions on both IBM AIX® and Linux. When a job ends, its orphaned processes might continue to use or hold resources, thereby degrading system performance, or causing jobs to hang or fail.

With process tracking, LoadLeveler can cancel any processes (throughout the entire cluster) that are left behind when a job ends. When you run either `BACKFILL` or the application programming interface (API) scheduler, process tracking is required to accomplish preemption by the suspend method. Process tracking is optional in all other cases.

When process tracking is enabled, all child processes are stopped when the main process ends. These processes include any background or orphaned processes started in the following programs:

- Prolog
- Epilog
- User prolog
- User epilog

LSF provides process tracking on all supported platforms:

- Using process information collected by the LSF Process Information Manager (PIM) daemon running on each node
- Using more advanced features, such as cpusets and cgroups, if supported on your systems

Example 12 shows how to configure LoadLeveler to support process tracking.

Example 12 Configuration to support process tracking in LoadLeveler

```
$ vim LoadL_config
...
PROCESS_TRACKING = TRUE
PROCESS_TRACKING_EXTENSION = $(BIN)
...
```

On Linux platforms that support it, you can enable the LSF cgroup feature, as shown in Example 13.

Example 13 Enabling the LSF cgroup feature

```
$ vim $LSF_ENVDIR/lsf.conf
```

```
...
LSF_PROCESS_TRACKING=Y
...
```

LSF also provides job accounting information by default. There is no need to manually enable the job accounting. Job accounting information is logged by LSF to the `lsb.acct` file, and is visible through the **bacct** and **bhist** commands.

Scheduling policies

This section describes the scheduling policies for LoadLeveler and LSF.

Preemption

LoadLeveler has two distinct preemption methods:

- The **llpreempt** preemption command for manual preemption
- System-level preemption with a set of scheduling rules

LSF provides similar preemptive scheduling functionality. Example 14 shows the LoadLeveler preemption configuration.

Example 14 LoadLeveler preemption configuration in LoadLeveler

```
$ vim LoadL_config
...
PROCESS_TRACKING       = true
PREEMPTION_SUPPORT     = full
DEFAULT_PREEMPT_METHOD = su
PREEMPT_CLASS[priority]   = enough{normal}
...
$ vim normal.cmd
#! /bin/sh
...
# @ node = 2
# @ total_tasks = 4
# @ resources = ConsumableCpus(1) ConsumableMemory(800 mb)
# @ class = normal
# @ queue
...
$ vim priority.cmd
#! /bin/sh
...
# @ node = 2
# @ total_tasks = 4
# @ resources = ConsumableCpus(1) ConsumableMemory(800 mb)
# @ class = priority
# @ queue
...
$ llstatus -Lmachine -l | grep -i "^Max_Starters"
Max_Starters        = 4
Max_Starters        = 4

$ llsubmit normal.cmd
llsubmit: The job "hostb.clusters.com.20" has been submitted.

$ llsubmit normal.cmd
```

```
llsubmit: The job "hostb.clusters.com.21" has been submitted.

$ llsubmit normal.cmd
llsubmit: The job "hostb.clusters.com.22" has been submitted.

$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
----------------------- ---------- ----------- -- --- ------------ -----------
hostb.21.0            user1       1/15 20:50 R  50  normal       hostc
hostb.20.0            user1       1/15 20:50 R  50  normal       hostc
hostb.22.0            user1       1/15 20:52 I  50  normal

3 job step(s) in queue, 1 waiting, 0 pending, 2 running, 0 held, 0 preempted

$ llclass
Name                 MaxJobCPU     MaxProcCPU  Free   Max Description
                    d+hh:mm:ss     d+hh:mm:ss Slots Slots
-------------- -------------- -------------- ----- ----- ----------------------
No_Class             undefined      undefined     0     8
priority             undefined      undefined     0     8
normal               undefined      undefined     0     8
-------------------------------------------------------------------------------
"Free Slots" values of the classes "No_Class", "priority", "normal" are
constrained by the MAX_STARTERS limit(s).

$ llsubmit priority.cmd
llsubmit: The job "hostb.clusters.com.23" has been submitted.

$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
----------------------- ---------- ----------- -- --- ------------ -----------
hostb.21.0            user1        1/15 20:55 R  50  normal       hostc
hostb.23.0            user1        1/15 20:56 R  50  priority     hostc
hostb.22.0            user1        1/15 20:55 I  50  normal
hostb.20.0            user1        1/15 20:55 E  50  normal

4 job step(s) in queue, 1 waiting, 0 pending, 2 running, 0 held, 1 preempted
```

The corresponding LSF preemption configuration is shown in Example 15.

Example 15 LSF preemption configuration

```
$ vim $LSF_ENVDIR/lsf.conf
...
Begin Queue
QUEUE_NAME   = normal
PRIORITY     = 30
...
End Queue
...
Begin Queue
QUEUE_NAME    = priority
PRIORITY      = 43
PREEMPTION = PREEMPTIVE
...
End Queue
```

```
...
$ bhosts
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
hostb           ok              -      4      0      0      0      0      0
hostc           ok              -      4      0      0      0      0      0

$ bsub -q normal -n 4 -R "span[ptile=2]" /bin/sleep 1800
Job <393> is submitted to queue <normal>.

$ bsub -q normal -n 4 -R "span[ptile=2]" /bin/sleep 1800
Job <394> is submitted to queue <normal>.

$ bsub -q normal -n 4 -R "span[ptile=2]" /bin/sleep 1800
Job <395> is submitted to queue <normal>.

$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
393     user2   RUN    normal     hostb    hostc    *leep 1800 Jan 12 21:04
                                             hostc
                                             hostb
                                             hostb
394     user2   RUN    normal     hostb    hostc    *leep 1800 Jan 12 21:04
                                             hostc
                                             hostb
                                             hostb
395     user2   PEND   normal     hostb                *leep 1800 Jan 15 21:04

$ bsub -q priority -n 4 -R "span[ptile=2]" /bin/sleep 1800
Job <396> is submitted to queue <normal>.

$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
396     user2   RUN   priority   hostb    hostb    *leep 1800 Jan 12 21:06
                                             hostb
                                             hostc
                                             hostc
393     user2   RUN   normal     hostb    hostc    *leep 1800 Jan 12 21:04
                                             hostc
                                             hostb
                                             hostb
394     user2   SSUSP normal     hostb    hostc    *leep 1800 Jan 12 21:04
                                             hostc
                                             hostb
                                             hostb
395     user2   PEND  normal     hostb              *leep 1800 Jan 12 21:04
```

LoadLeveler supports the following types of preemption: `su`, `uh`, `sh`, `vc`, and `rm`. LSF provides comprehensive preemption policies based on job slots and resources. Administrators can also customize preemption signals or actions to cover site-specific requirements. See *Administering IBM Platform LSF*, SC27-5302, for more information about preemptive scheduling in LSF.

The LoadLeveler **`llpreempt`** command supports manual job control rather than system-level preemptive scheduling. In LSF, you can use several commands to achieve the same goal.

For example, the **llpreempt -m su job_id** command is equivalent to the **bstop job_id** command in LSF. Also, the **llpreempt -m vc job_id** command is equivalent to the **brequeue job_id** command in LSF. See the *IBM Platform LSF Command Reference*, SC27-5305, for more information about the **bstop**, **bresume**, and **brequeue** commands.

Fair share scheduling

See *Administering IBM Platform LSF*, SC27-5302, for more information about fair share scheduling in LSF.

Island scheduling

The *compute unit* concept in LSF partially matches the LoadLeveler *island scheduling* feature.

Affinity scheduling

LoadLeveler works with IBM Parallel Environment (PE) Runtime Edition to provide affinity support for parallel jobs. LoadLeveler also provides affinity support for sequential jobs.

LoadLeveler provides *resource set (RSET)-based* CPU and memory affinity on AIX systems. On x86 Linux, LoadLeveler provides CPU and memory affinity by working with PE Runtime Edition for parallel jobs, and by using Linux cpusets for sequential jobs.

On Linux, LSF supports both cpusets and CPU affinity for sequential jobs, and CPU affinity for parallel jobs.

Example 16 shows a typical affinity scheduling example for sequential jobs in LoadLeveler.

Example 16 LoadLeveler affinity scheduling configuration

```
$ vim LoadL_config
...
RSET_SUPPORT = RSET_MCM_AFFINITY
...

$ vim job1.cmd
#! /bin/sh
# @ job_type = serial
...
# @ task_affinity = cpu(1)
# @ queue
...

$ llsubmit job1.cmd
llsubmit: The job "hostb.clusters.com.29" has been submitted.

$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
---------------------- ---------- ----------- -- --- ------------ -----------
hostb.29.0             user1       1/15 21:45 R  50  normal       hostc
1 job step(s) in queue, 0 waiting, 0 pending, 1 running, 0 held, 0 preempted
$ ssh hostc lscgroup | grep cpuset
cpuset:/
cpuset:/hostb.29.0.tid-1
[user1@hostb]$ llstatus -M -h hostc
Machine               MCM details
-------------------- -------------------------------------------------
hostc.clusters.com
```

```
             MCMO
                Available Cpus :<  0-1 >(2)
                Used Cpus      :<  0 >(1)
                Adapters       :
                Total Tasks    :(1)
```

Example 17 shows the corresponding LSF configuration and job submission.

Example 17   LSF affinity scheduling configuration

```
$ vim $LSF_ENVDIR/lsbatch/user1_dev/configdir/lsb.hosts
...
Begin Host
HOST_NAME MXJ   r1m     pg    ls    tmp  DISPATCH_WINDOW  AFFINITY
evf2n02    !    ()      ()    ()     ()     ()                 (Y)
End Host
...
$ bsub -R "affinity[thread(1)]" -n 1 /bin/sleep 1800

$ bjobs -l | grep PID
                     PGID: 2019;  PIDs: 2019
$ taskset -pc 2019
pid 2019's current affinity list: 0
```

Note: For more information about support for high-performance networks, see the section on running Message Passing Interface (MPI) workloads through IBM PE Runtime Edition in *Administering IBM Platform LSF*, SC27-5302.

Move users and users' jobs from LoadLeveler to LSF

After the LSF cluster is configured and enabled with the job scheduling and management policies that you want, the next step is to migrate LoadLeveler job control and job command files to LSF job submission parameters.

After this step is completed, LoadLeveler users should be able to submit jobs to LSF, and the LSF cluster can be tested for production use. This section explains the job management characteristics in LoadLeveler and LSF.

Job submission

The LoadLeveler job submission command is **llsubmit**. The LoadLeveler **llsubmit** command requires that all of the job descriptions are defined in a job command file before submitting the command.

The LSF **bsub** command is the equivalent job submission tool. The LSF **bsub** command is more flexible. You can use it to specify job submission parameters, either using command-line options, or in a job submission pack file (by redirecting job submission files to the **stdin** of the **bsub** command).

See the *IBM Platform LSF Command Reference*, SC27-5305, for more details about the **bsub** command.

The following sections provide some examples of how to map LoadLeveler job submission attributes to LSF.

Execution environment

LoadLeveler uses the `# @ environment = xxxx` keyword to specify which environment variables need to be copied into the job's execution environment. LoadLeveler has an internal `COPY_ALL` variable that implies “to copy all environment variables.”

In LSF, by default, environment variables at job submission time will be set in the job execution environment. However, LSF provides mechanisms, such as job starters and external submission executable programs, to modify the job execution environment.

Task distribution

LoadLeveler supports the following types of task distribution, which could be requested within a job command file when submitting the job with the **llsubmit** command:

- By node
- By block
- By packing
- By host list
- By task geometry

Task distribution by node

The `# @ node`, `# @ tasks_per_node`, and `# @ total_tasks` keywords can be used in LoadLeveler to describe *by node* task distribution. The corresponding LSF **bsub** option for this feature is **bsub -R "span[...]"**.

Example 18 shows a typical *by node* job sample.

Example 18 Task distribution by node in LoadLeveler

```
$ vim job1.cmd
# @ job_type = parallel
# @ job_class = short
# @ node = 2
# @ tasks_per_node = 2
# @ executable = /usr/bin/poe
# @ arguments = /u/user1/bin/btat.a64r6 -ilevel 6 -pmdlog yes -d 60 -t 1
# @ wall_clock_limit = 00:30:00
# @ queue

$ llsubmit job1.cmd
```

The corresponding LSF **bsub** command is shown in Example 19.

Example 19 Task distribution by node in LSF

```
$ bsub -q short -n 4 -R "span[ptile=2]" -W 30 /usr/bin/poe /u/user1/bin/btat.a64r6
....
```

Task distribution by block

LSF does not provide a feature similar to *by block* task distribution in LoadLeveler. However, host groups and compute units might partially satisfy the requirements.

Task distribution by packing

The `# @ total_tasks` and `# @ blocking = unlimited` keywords can be used in LoadLeveler to describe *by packing* task distribution. By default, the LSF **bsub** command distributes job tasks by packing.

Example 20 shows a typical *by packing* job sample.

Example 20 Task distribution by packing in LoadLeveler

```
$ vim job3.cmd
# @ job_type = parallel
# @ job_class = short
# @ total_tasks = 4
# @ blocking = unlimited
# @ executable = /usr/bin/poe
# @ arguments = /u/user1/bin/btat.a64r6 -ilevel 6 -pmdlog yes -d 60 -t 1
# @ wall_clock_limit = 00:30:00
# @ queue

$ llsubmit job3.cmd
```

The corresponding LSF **bsub** command is shown in Example 21.

Example 21 Task distribution by packing in LSF

```
$ bsub -q short -n 4 -W 30 /usr/bin/poe /u/user1/bin/btat.a64r6 …
```

Task distribution by host list

The `# @ host_file` keyword is used to describe the *by host list* task distribution in LoadLeveler. The `host_file` keyword forces the scheduling system to make the exact allocation as specified in the host list.

In LSF, use compound resource requirements combined with job execution-time enforcement via PE Runtime Edition to achieve similar functionality.

Example 22 shows a typical *by host list* job sample.

Example 22 Task distribution by host list in LoadLeveler

```
$ vim job2.cmd
# @ job_type = parallel
# @ job_class = short
# @ host_list = hosts
# @ executable = /usr/bin/poe
# @ arguments = /u/user1/bin/btat.a64r6 -ilevel 6 -pmdlog yes -d 60 -t 1
# @ wall_clock_limit = 00:30:00
# @ queue

$ cat hosts
hostb(2)
hostc(2)

$ llsubmit job2.cmd
```

Task distribution by task geometry

The `# @ task_geometry` keyword is used to describe the *by task geometry* task distribution. The corresponding LSF feature is supported by the **`LSB_PJL_TASK_GEOMETRY`** environment variable.

See the section on running MPI workloads through PE Runtime Edition in *Administering IBM Platform LSF*, SC27-5302, for more information.

Example 23 shows a typical *by task geometry* job sample.

Example 23 Task distribution by task geometry job sample

```
$ vim job5.cmd
# @ job_type = parallel
# @ job_class = short
# @ task_geometry = {(0) (1,2,3)}
# @ executable = /usr/bin/poe
# @ arguments = /u/user1/bin/btat.a64r6 -ilevel 6 -pmdlog yes -d 60 -t 1
# @ wall_clock_limit = 00:30:00
# @ queue

$ llsubmit job5.cmd
```

The corresponding LSF **bsub** command is shown in Example 24.

Example 24 Task distribution by task geometry in LSF

```
$ setenv LSB_PJL_TASK_GEOMETRY "{(0) (1,2,3)}"
$ bsub -network "type=sn_single:mode=us:instance=1:protocol=mpi" -q short -n 6 -R
"span[ptile=3]" -W 30 /usr/bin/poe …
```

Requesting network resources

LoadLeveler uses the `# @ network` keyword to request network resources for parallel jobs. The **bsub -network** option in LSF supports similar functionality. See the *IBM Platform LSF Command Reference*, SC27-5305*,* and the man pages, for more information about the **bsub -network** option.

Example 25 is a typical job sample with a network statement.

Example 25 Requesting network resources in LoadLeveler

```
$ vim job5.cmd
# @ job_type = parallel
...
# @ network.mpi = sn_single,,US,,instances=1
# @ queue

$ llsubmit job5.cmd
```

The corresponding LSF **bsub** command is shown in Example 26.

Example 26 Requesting network resources in LSF

```
$ bsub … -network "type=sn_single:mode=us:instance=1:protocol=mpi" …
```

Controlling jobs by resource limits

Both LoadLeveler and LSF provide job execution environment control *by limits*, as shown in Table 3.

Table 3 Job execution environment control

LoadLeveler `limit` keyword	LSF `bsub` option
as_limit	-v

LoadLeveler `limit` keyword	LSF `bsub` option
`core_limit`	`-C`
`cpu_limit`	`-c` with `LSF_JOB_CPULIMIT=n`
`data_limit`	`-D`
`file_limit`	`-F`
`jopb_cpu_limit`	`-c`
`locks_limit`	Not supported
`memlock_limit`	Not supported
`nofile_limit`	Not supported
`nproc_limit`	`-T`
`rss_limit`	`-M`
`stack_limit`	`-S`
`wall_clock_limit`	`-W`

Job classes

LoadLeveler uses the `# @ class` keyword to enable users to select the class for the job. The LSF **`bsub -q`** option specifies the corresponding functionality.

Example 27 shows a typical job sample with a *job class*.

Example 27 Selecting a class for the job in LoadLeveler

```
$ vim job5.cmd
# @ job_type = parallel
...
# @ class = short
# @ queue

$ llsubmit job5.cmd
```

The corresponding LSF **`bsub`** command is shown in Example 28.

Example 28 Selecting a class for the job in LSF

```
$ bsub … -q short …
```

Resource requirements

LoadLeveler supports filtering machines *by requirement expression* with the `# @ requirements` keyword. In LSF, the **`bsub`** command **`-R "select[…]"`** resource requirements option supports similar functionality.

Example 29 shows a typical *by requirement expression* job sample.

Example 29 Filtering machines using a requirement expression in LoadLeveler

```
$ vim job5.cmd
# @ job_type = parallel
...
# @ requirements = (Feature == "feature_a")
```

```
...
# @ queue

$ llsubmit job5.cmd
```

The corresponding LSF **bsub** command is shown in Example 30.

Example 30 Filtering machines in LSF

```
$ bsub … -R "select[feature_a]" …
```

Resource preferences

LoadLeveler supports ordering candidate machines *by preference expression* with the `# @ preferences` keyword. The LSF **bsub** command has the **-R "order[…]"** resource requirement specification option to support similar functionality.

Example 31 shows a typical *by preference expression* job sample.

Example 31 Resource preferences with LoadLeveler

```
$ vim job5.cmd
# @ job_type = parallel
...
# @ preferences = (Feature == "feature_a")
...
# @ queue

$ llsubmit job5.cmd
```

The corresponding LSF **bsub** command is shown in Example 32.

Example 32 Resource preferences in LSF

```
$ bsub … -R "order[feature_a]" …
```

Job control

This section describes the job control commands.

Canceling jobs

The **llcancel** command in LoadLeveler is equivalent to the **bkill** command in LSF. See the *IBM Platform LSF Command Reference*, SC27-5305, and the man pages, for more information about the **bkill** command.

Table 4 shows a summary of how the **llcancel** and the **bkill** command options map for these two commands.

Table 4 The llcancel and bkill command options

LoadLeveler **llcancel**	LSF **bkill**
`-f` *<hostlist>*	Not supported
`-u userlist`	`-u ...`
`-h hostlist`	Not supported
<joblist>	`jobId ...`

The LSF **bkill** command is more flexible than the LoadLeveler **llcancel** command for job control, because it provides more options to select target jobs.

Modify jobs

The **llmodify** command in LoadLeveler is equivalent to the **bmod** and **bswitch** commands in LSF. See the *IBM Platform LSF Command Reference*, SC27-5305, and the man pages, for more information about the **bmod** and **bswitch** commands.

Table 5 is a summary of how the available options map for these two commands.

Table 5 LoadLeveler and LSF modify jobs commands

LoadLeveler llmodify	LSF bmod
`-c consumable_cpus`	`-N`
`-m consumable_memory`	`-M`
`-W wclimit_add_min`	`-W`
`-C job_class`	`-q`
`-a account_no`	Not supported because LSF does not have `account_no`.
`-s q_sysprio`	Not supported because LSF does not have a system priority of the job step.
`-p preempt|nopreempt`	Not supported because LSF does not have a similar preemptible soft flag on the job.
`-k keyword=value` The following values for keywords are valid: ► `account_no` ► `bg_connectivity` ► `bg_node_configuration` ► `bg_block` ► `bg_rotate` ► `bg_shape` ► `bg_size` ► `bg_requirements` ► `class` ► `cluster_option` ► `consumableCpus` ► `consumableMemory` ► `node_resources` ► `preemptible` ► `resources` ► `sysprio` ► `wclimit_add` ► `dstg_resources` ► `wall_clock_limit` ► `startdate`	For those `bg_*` keywords related to IBM Blue Gene®, The LSF **bmod** command does not support similar functionality. The `cluster_option` keyword is related to LoadLeveler multicluster. The `dst_resources` and `node_resources` keywords map to `bmod -R “usage[...]`. The account_no and class methods are alternative methods in LoadLeveler. LSF can support them as is. For other keywords, LSF does not have similar concepts, and does not support them.

Holding and releasing jobs

The LoadLeveler **llhold** command holds and releases an idle job. The corresponding LSF commands are **bstop** and **bresume**. After a job is submitted, the **bsub -H** option also keeps the job in a `PENDING` state until the **bresume** command is issued. LSF also supports threshold scheduling based on stop and suspend thresholds defined in the `lsb.hosts` file.

See the *IBM Platform LSF Command Reference*, SC27-5305, and the man pages, for more information about the **bstop** and **bresume** commands. See the *IBM Platform LSF Configuration Reference*, SC27-5306, for more information about the `lsb.hosts` file.

Example 33 shows a typical job hold and release sample.

Example 33 Holding a releasing a job with LoadLeveler

```
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
----------------------- ---------- ----------- -- --- ------------ -----------
hostb.14.0             user1      1/14 18:35 I  50  small
1 job step(s) in queue, 1 waiting, 0 pending, 0 running, 0 held, 0 preempted

$ llhold hostb.14.0
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
----------------------- ---------- ----------- -- --- ------------ -----------
hostb.14.0            user1       1/14 18:35 H  50  small

1 job step(s) in queue, 0 waiting, 0 pending, 0 running, 1 held, 0 preempted

$ llhold -r hostb.14.0
$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
----------------------- ---------- ----------- -- --- ------------ -----------
hostb.14.0            user1       1/14 18:35 I  50  small

1 job step(s) in queue, 1 waiting, 0 pending, 0 running, 0 held, 0 preempted
```

The corresponding LSF **bsub** command is shown in Example 34.

Example 34 Holding a releasing a job with LSF

```
$ bjobs 276
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME SUBMIT_TIME
276     user2    PEND  normal     hostb                *leep 1800 Jan 14 18:41

$ bstop 276
Job <276> is being stopped

$ bjobs 276
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME SUBMIT_TIME
276     user2    PSUSP normal     hostb                *leep 1800 Jan 14 18:41

$ bresume 276
Job <276> is being resumed

$ bjobs 276
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME SUBMIT_TIME
276     user2    PEND  normal     hostb                *leep 1800 Jan 14 18:41
```

Checkpoint and restart

LSF provides a checkpoint framework to support operating system-level checkpoints, user-level checkpoints, and application-level checkpoints on applicable platforms. Parallel job checkpoint and restart is supported if this application-level functionality is available.

Job dependencies

Example 35 shows the creation of a two-step job. The second job step starts after waiting for the first job step to complete.

Example 35 Creating a two-step job with LoadLeveler

```
#!/bin/ksh
# @ step_name           = step1
# @ job_type            = serial
# @ executable          = /bin/sleep
# @ arguments           = 30
# @ error               = error.$(jobid)
# @ output              = output.$(jobid)
# @ wall_clock_limit    = 18:00
# @ class               = small
# @ queue

# @ step_name           = step2
# @ dependency          = (step1 == 0)
# @ executable          = /bin/sleep
# @ arguments           = 18000
# @ error               = error.$(jobid)
# @ output              = output.$(jobid)
# @ wall_clock_limit    = 18:00
# @ class               = small
# @ queue

[loadl@hosta ~]$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
----------------------- ---------- ----------- -- --- ------------ -----------
hosta.5.0             loadl       2/22 08:03 R  50  small        hosta
hosta.5.1             loadl       2/22 08:03 NQ 50  small

2 job step(s) in queue, 0 waiting, 0 pending, 1 running, 1 held, 0 preempted

[loadl@hosta ~]$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
----------------------- ---------- ----------- -- --- ------------ -----------
hosta.5.1             loadl       2/22 08:03 R  50  small        hosta
hosta.5.0             loadl       2/22 08:03 C  50  small

1 job step(s) in queue, 0 waiting, 0 pending, 1 running, 0 held, 0 preempted
```

The corresponding LSF **bsub** command is show in Example 36.

Example 36 Creating a two step-job in LSF with the bsub command

```
bash-3.2$ cat step1.jf
c#!/bin/csh
#BSUB -q normal
#BSUB -J step1
```

```
sleep 180;
bash-3.2$ cat step2.jf
#!/bin/csh
#BSUB -q normal
#BSUB -J step2
#BSUB -w done(step1)
sleep 180;
bash-3.2$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
111     user1   RUN   normal     delpe05.eng delpe05.eng step1      Feb 21 21:55
112     user1   PEND  normal     delpe05.eng             step2      Feb 21 21:55
bash-3.2$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
112     user1   PEND  normal     delpe05.eng             step2      Feb 21 21:55
bash-3.2$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
112     user1   RUN   normal     delpe05.eng delpe05.eng step2      Feb 21 21:55
```

Prolog and epilog

The prolog and epilog functions are useful for monitoring and controlling job execution.

LoadLeveler can configure two sets of prolog and epilog functions in a global configuration file that serves all jobs. The LSF *pre-execution and post-execution* feature corresponds to the LoadLeveler *prolog and epilog* feature.

See *Administering IBM Platform LSF*, SC27-5302, for more information about pre-execution and post-execution processing.

User prolog and epilog

The LSF *user prolog and epilog* support is simpler than that in LoadLeveler. Each job can have its own prolog and epilog specified with the **bsub -E** and **-Ep** options. See the **bsub** man page for more detail.

Example 37 shows a typical *user prolog and epilog* sample.

Example 37 LoadLeveler user prolog and epilog

```
$ vim LoadL_config
...
JOB_USER_PROLOG = /u/user1/bin/uprolog.sh
JOB_USER_EPILOG = /u/user1/bin/uepilog.sh
...

$ llsubmit job1.cmd
```

The corresponding LSF **bsub** command is shown in Example 38.

Example 38 LSF pre-execution and post-execution options

```
$ bsub -E /u/user1/bin/uprolog.sh -Ep /u/user1/bin/uepiog.sh …
```

System prolog and epilog

LSF provides queue-level prolog and epilog functions, and application-level prolog and epilog functions, which are configured by the administrator. See the sections on the `lsb.queues` file and the `lsb.applications` file in the *IBM Platform LSF Configuration Reference*, SC27-5306, for more information about configuring queues and application profiles.

Example 39 shows a typical *system prolog and epilog* sample.

Example 39 LoadLeveler system prolog and epilog

```
$ vim LoadL_config
...
JOB_PROLOG = /u/user1/bin/sprolog.sh
JOB_EPILOG = /u/user1/bin/sepilog.sh
...

$ llsubmit job1.cmd
```

The corresponding LSF **bsub** command is shown in Example 40.

Example 40 LSF system prolog and epilog

```
$ vim $LSF_ENVDIR/lsbatch/<cluster_name>/configdir/lsb.queues
...

Begin Queue
QUEUE_NAME = short
...
PRE_EXEC  = /u/user1/bin/sprolog.sh
POST_EXEC = /u/user1/bin/sepilog.sh
...
End Queue
...
```

Reservation

LoadLeveler reservation is equivalent to LSF advance reservation (AR). LSF creates slot-based AR, but LoadLeveler is node-based.

To reserve a set of hosts for job execution, LSF provides functions similar to LoadLeveler. The basic flow of using the reservation feature is almost the same in both systems:

1. Create a reservation.
2. Submit a job to the reservation and activate the reservation (a job is scheduled within the reservation).
3. Modify the reservation.
4. Clean up the job after the reservation expires.

See *Administering IBM Platform LSF*, SC27-5302, for more information about AR in LSF.

Creating a reservation

LoadLeveler has the **llmkres** command to create the reservation. The corresponding LSF command is **brsvadd**. For more information about the **brsvadd**, command, see *IBM Platform LSF Command Reference*, SC27-5305, and the man pages.

A typical reservation creation contains the following requests:

- Time frame
- Resources to be reserved
- Jobs to be run within that reservation

The **`llmkres`** command has the **`-t`**, **`-x`**, **`-d`**, and **`-e`** options to specify the time frame. They specify the following information:

- Start time (date and time)
- Flexible reservation
- Duration (in minutes)
- Expiration

The following examples specify reservations with different time frames.

Example 41 creates a one-time reservation to reserve one host for one hour.

Example 41   LoadLeveler one-time reservation

```
$ vim LoadL_admin
...
default: {
        type = user
        …
        max_reservations = 4
        max_reservation_duration = 120
        …
}

default: {
        type = machine_group
        …
        reservation_permitted = true
}

$ llmkres -t "01/10/2013 18:00" -d 60 -n 1
llmkres: The reservation hostb.clusters.com.6.r has been successfully made.

$ LL_RES_ID=hostb.6.r llsubmit job1.cmd
```

The corresponding LSF **`bsub`** command for a one-time reservation is shown in Example 42.

Example 42   LSF one-time reservation

```
$ vim $LSF_ENVDIR/lsbatch/<cluster_name>/configdir/lsb.resources
...
Begin ResourceReservation
NAME  = testPolicy
USERS = user1
HOSTS = hostb hostc
TIME_WINDOW = 00:00-24:00
End ResourceReservation
...

$ brsvadd -n 1 -R "select[slots > 0]" -b "2013:1:10:18:00" -e "2013:1:10:19:00" -u
user1
Reservation user1#5 is created

$ bsub -U user1#5 -n 1 /bin/sleep 1800
```

Example 43 creates a recurring reservation to reserve one host at 7:00 a.m. every Monday morning.

Example 43 Creating a recurring reservation in LoadLeveler

```
$ llmkres -t "00 07 * * 1" -d 60 -n 1 -e "02/01/2013 18:00"
llmkres: The reservation hostb.clusters.com.9.r has been successfully made.
```

The corresponding LSF **bsub** command for a recurring reservation is shown in Example 44.

Example 44 Creating a recurring reservation in LSF

```
$ brsvadd -n 1 -R "select[slots > 0]" -u user1 -t "1:7:0-1:8:0"
Reservation user1#6 is created
```

The **llmkres** command provides options **-n**, **-h**, **-j**, **-f**, **-c**, and **-F** to help specify which resources to be reserved. LSF provides options **-n**, **-m**, and **-R**. See *IBM Platform LSF Command Reference*, SC27-5305, for more information about these command options.

For the **llmkres** command **-n**, **-h**, and **-F** options, LSF does not provide exactly the same functions, because the resource unit of LSF reservation is *slots*, but the LoadLeveler unit is *host*. A workaround is to carefully calculate the slots number from the expected number of hosts, because the maximum slot number is mostly a static value that is configured by the administrator.

For the **llmkres -j** and **-f** options, LSF provides **-n**, **-M**, and **-R** options to achieve almost the same functions by combining them together. The general idea of the **llmkres -j** and **-f** options is to tell the scheduler to select resources based on the resource requirements from a job. The LSF **brsvadd -R** option provides the same LoadLeveler functions and flexibility after understanding the resource requirement conversion from a LoadLeveler job to an LSF job.

For more information, see the description about the **bsub** command in *IBM Platform LSF Command Reference*, SC27-5305.

Submitting a job to a reservation

When a reservation is created, LoadLeveler provides the **llbind** command for users to bind their jobs to a reservation. The LoadLeveler **llsubmit** command interprets the **LL_RES_ID** environment variable before accepting the job submit request, so that the new job can be bound to an existing reservation.

LSF provides the **bsub -U** option for users submitting a job to create a reservation that is similar to the LoadLeveler **llsubmit** command and **LL_RES_ID** approach. This achieves almost the same functionality to bind a job to a reservation.

Modifying a reservation

LoadLeveler provides the **llchres** command to help the user modify a reservation, and LSF provides the **brsvmod** command. The options for these commands are almost the same as the reservation creation command options.

Removing a reservation

To remove a reservation, LoadLeveler provides the **llrmres** command, and LSF provides the **brsvdel** command. A typical usage of these commands is to remove a specific reservation identified by a reservation ID. The only difference is the reservation ID naming syntax.

You can simply run a reservation query command to confirm before any future modification operations. The LoadLeveler reservation query command is **llqres**, and the LSF command is **brsvs**.

Accounting

In addition to using IBM Platform Analytics to obtain accounting information for finished jobs, you can use the **bhist** and **bacct** commands in LoadLeveler.

Job command file keyword reference

Each LoadLeveler job command directive is shown in Table 6 with a comment, along with the associated LoadLeveler environment variable (if any) available at run time. This is followed by the LSF equivalent, if any.

The input to the **bsub** command in LSF can be an executable shell script in which the **bsub** command-line options are embedded for **bsub** to parse. The job directives can also be passed to **bsub** as command-line arguments, or through the **bsub** interactive command prompt.

Table 6 LoadLeveler job command reference

LL job command file directive	LL environment variable	LL comment	LSF bsub job spool script directive	LSF comment
#! /bin/ksh		Optional, but a shell interpreter typically turns this command file into a shell script. Any script type is supported. For example, perl/ksh/csh.	#! /bin/ksh	
# [comments text]		Optional.	# [comments text]	Optional.
# @ job_name = [string]	**LOADL_JOB_NAME**	Optional.	#BSUB -J [string]	
# @ step_name = [string]	**LOADL_STEP_NAME** **LOADL_STEP_ID**	Optional.		Optional.
# @ dependency = [dep]		Optional.		
# @ initialdir = [path to directory where job should run]	**LOADL_STEP_INITDIR**	Optional. The directory must exist and be accessible, or else the job will get rejected.		The working directory is the current directory.
# @ executable = [job command file]	**LOADL_STEP_COMMAND**	Optional.		Specified in the job spool script.
# @ arguments = [string of arguments]	**LOADL_STEP_ARGS**	Optional.		Specified in the job spool script on the executable command line.
# @ input = [name of input file]	**LOADL_STEP_IN**	Optional.	#BSUB -i [stdin file]	

LL job command file directive	LL environment variable	LL comment	LSF `bsub` job spool script directive	LSF comment
`# @ output = [stdout file (for example: csout.stdout)]`	**`LOADL_STEP_OUT`**	Optional.	`#BSUB -o [stdout file (for example, csout.stdout)]`	
`# @ error = [stderr file (for example: csout.stderr)]`	**`LOADL_STEP_ERR`**	Optional.	`#BSUB -e [stderr file (for example, csout.stderr)]`	

`# @ class = [one of supported classes]`	**`LOADL_STEP_CLASS`**	Optional.
		Default= `unlimited`.
			`#BSUB -P [ ]`
			`#BSUB -J [ ]`	

`# @ environment = [semicolon-separated list of environment vars to be available to job]`		Optional. **`COPY_ALL`** copies all of the environment variables.	`export [env-var=value]`	
`# @ requirements = [Boolean expr consisting of: { Feature|Speed|Machine |OpSys|Arch}]`	**`LOADL_REQUIREMENTS`**	Optional. Default `OpSys/Arch` from the submitting machine.	`#BSUB -R [requirement]`	Optional static resources.
`# @ preferences = [Boolean expr consisting of: { Feature|Speed|Machine }]`		Optional.	`#BSUB -R [requirement]`	Optional.
`# @ resources =`		Suggested. Default=(see class default).	`#BSUB -R [requirement]`	Optional.
`# ConsumableCpus(min)`		Optional. Minimum number of processors.	`#BSUB -n min[,max]`	Minimum number of processors, maximum number of processors.
`#`		Optional line-continuation.		Optional line-continuation.
`# ConsumableMemory (min [units])`		Optional. Units default= megabytes (MB).	`#BSUB -R "rusage[mem=nn]"`	Optional. Units default= MB.
`# Memory(min [units])`		Optional. Units default=MB. Physical memory >= count.		
`# @ shell = [optional login shell of job, for example, bin/ksh]`		Optional. Default=user's password file entry.		

LL job command file directive	LL environment variable	LL comment	LSF `bsub` job spool script directive	LSF comment

`# @ notification = [One of: always|error|start|never|complete]`		Optional. Default=`complete`, and email has job_name or job ID as the subject line.	`#BSUB -B # start`
			`#BSUB -o [file] # never`
			`#BSUB -N # complete`
				Optional. Default = send all `stdout/stderr` and job information to the email set in the `lsf.conf` file.

`# @ notify_user = [user-ID or email address where email is sent]`		Optional. Default=*submitting -user-ID@ submitting-host name*.	`#BSUB -u [alternate user-ID]`	Optional. Default= *submitting-user-ID@ submitting-host name*. Can specify only an IBM AFS™ ID, not an email address. Set in `lsf.conf`.
`# @ startdate`		Optional.		Optional.
`# @ account_no = [valid value]`	**`LOADL_STEP_ACCOUNT`**	Required. Enforced by Grid submit filter.	`#BSUB -P [valid value]`	
`# @ group = [valid value]`	**`LOADL_STEP_GROUP`** **`LOADL_GROUP_NAME`**	Suggested.	`#BSUB -G [valid value]`	Required.

`# @ cpu_limit = [hardlimit[,softlimit]]`
`# @ data_limit = [hardlimit[,softlimit]]`
`# @ file_limit = [hardlimit[,softlimit]]`
`# @ core_limit = [hardlimit[,softlimit]]`
`# @ rss_limit = [hardlimit[,softlimit]]`
`# @ stack_limit = [hardlimit[,softlimit]]`
`# @ job_cpu_limit = [hardlimit[,softlimit]]`
		Optional.	`#BSUB -c or -n [limit]`
			`#BSUB -D [limit]`
			`#BSUB -F [limit]`
			`#BSUB -C [limit]`
			`#BSUB -M [limit]`
			`#BSUB -S [limit]`
			`#BSUB -c [limit]`
				Optional.

`# @ wall_clock_limit = [hardlimit[,softlimit]]`		Suggested units=seconds or HH:MM:SS. Default=(see class default).	`#BSUB -W [limit]`	Suggested.
`# @ max_processors`		Optional.		

LL job command file directive	LL environment variable	LL comment	LSF bsub job spool script directive	LSF comment
# @ min_processors		Optional.		
# @ blocking		Optional.		
# @ checkpoint		Optional.		
# @ ckpt_dir		Optional.		
# @ ckpt_file		Optional.		
# @ ckpt_time_limit		Optional.		
# @ comment = [text]	**LOADL_COMMENT**	Optional.		
# @ hold		Optional.		
# @ image_size		Optional.		
# @ job_type = [serial | parallel]	**LOADL_STEP_TYPE**	Optional. Default=serial, which includes any shared-memory (thread-/fork-based) parallelism.	#BSUB -R "span[hosts=1]" #BSUB -R "span[hosts=n]"	We assume that hosts=1 is not the default, so must be specified for serial jobs. Handled by esub.
			#BSUB -E [presubmit command]	Optional.
# @ network		Optional.		
# @ node		Optional.		
# @ node_usage		Optional.		
# @ parallel_path		Optional.		
# @ restart		Optional.		
# @ restart_from_ckpt		Optional.		
# @ restart_on_same_node		Optional.		
# @ task_geometry		Optional.		
# @ tasks_per_node		Optional.		
# @ total_tasks		Optional.		
# @ user_priority	**LOADL_STEP_NICE**	Optional.		
# @ queue		Required.		
		Optional blank lines.		Optional blank lines.
[shell commands to be run]		Optional, but commonly used. Must follow the LoadLeveler # @ queue statement.	[shell commands to be run]	Optional, but commonly used.

Submitting an LSF job spool script

A command file must be passed via the stdin file to the **bsub** command. LSF command-line options can be passed to **bsub** by embedding them as comments in the job spool script to be run, using the following syntax:

```
#BSUB-J JobName
#BSUB-q unlimited
```

To submit the job spool script containing the LSF options, use the following command:

```
bsub < lsf_script.sh
```

Note: The **bsub** script is similar to the job command file passed into LoadLeveler **llsubmit**. However, these input files differ:

- The input file to **llsubmit** is a LoadLeveler job command file containing directives for **llsubmit** to parse. The input file to **llsubmit** can optionally be a shell script (or a job command file packaged as an executable shell script).
- The input to **bsub** can be an executable shell script in which **bsub** command-line options are embedded for **bsub** to parse. Alternatively, the job directives can be passed to **bsub** as command-line arguments, or via the **bsub** interactive command prompt.

Mapping LoadLeveler to LSF

The following sections provide more detailed information about mapping LoadLeveler commands, concepts, and components to LSF:

- LoadLeveler command file to LSF job spool script (two examples)
- LoadLeveler commands to LSF commands
- LoadLeveler concepts to LSF concepts
- LoadLeveler job states to LSF job states
- LoadLeveler resources, requirements, and preferences to LSF resources
- LoadLeveler requirements to LSF resource selection (select string)

Mapping a LoadLeveler job command file to an LSF job spool script (1 of 2)

Table 7 shows the map of a LoadLeveler job command file to an LSF job spool script.

Table 7 LoadLeveler to LSF job command mapping table

LoadLeveler job command file directive	LSF bsub job spool script directive	Comment
#@ shell = /bin/ksh	#!/bin/ksh	
# Invoke: llsubmit ll-job-command-example.sh	# Invoke: bsub < lsf-job-spool-example.sh	
#@ job_name = my_cte_check	#BSUB -J my_cte_check	
#@ initialdir = $HOME/public/ll	cd $HOME/public/lsf	

LoadLeveler job command file directive	LSF `bsub` job spool script directive	Comment
`#@ environment = $CTEPATH; $PATH`	# LoadLeveler `environment` equivalent is inherently supported: all environment variables are available when this spool script is run.	`$CTEPATH` is inherited from the shell where the job spool script is run.

`#@ output = batch_test.$(jobid)_$(stepid).out`
`#@ error = batch_test.$(jobid)_$(stepid).err`
	`#BSUB -o batch_test.%J_%I.out`
	`#BSUB -e batch_test.%J_%I.err`
		LSF warning: if `-o` (or `-N`) is not specified, `stdout/stderr` is sent by email.
		Note: if `-o` is specified, `stdout/stderr` will be redirected to a file, and no email will be sent.

#@ notification = error	`#BSUB -N` # LoadLeveler `notification` equivalent is not available in LSF: `-B` and `-N` send job information upon job dispatch (exit).	LSF information: By default, an email will be sent when a batch job completes or exits. The mail includes a job report with the following information: ► LSF job information, such as CPU, process, and memory usage. ► Standard output/error of the job. ► If `-N` is specified, only the LSF job information will be sent in email, at job exit. ► If `-B` is specified, a notice will be emailed at job dispatch.
`#@ notify_user = user1@example.com`	# LoadLeveler `notify_user` equivalent is not available in LSF: `-u [user-ID]` supports an alternative user-ID, not email address.	LSF information: `#BSUB -u [user-ID]` enables you to specify an alternative user-ID (AFS ID), but not an email address.
`#@ class = hour`	`#BSUB -q hour`	
`#@ job_type = serial`	# LoadLeveler `job_type` is reflected in the `-R` span (locality) expression shown in the below cell. Assume that `hosts=1` is required for serial jobs.	Note that `serial` is the LoadLeveler default, so it is usually not specified. LoadLeveler `serial` implies only one compute node for a job step, which is the most common type for electronic design automation (EDA) tools.

LoadLeveler job command file directive	LSF bsub job spool script directive	Comment
#@ requirements = (Arch == "x86") && (OpSys == "Linux26") && (Feature == "RH56")	#BSUB -R "select[(type==X86_64) && (rh56)]" -R "span[hosts=1]" -R "rusage[mem=1]" -n 1	For AIX: #BSUB -R "select[(type==IBMAIX64) && (AIX61)]" -R "span[hosts=1]" -R "rusage[mem=1]" -n 1
#@ resources = ConsumableCpus (1) ConsumableMemory (1)	# LoadLeveler resources are reflected in the -n and -R rusage expressions shown in the above cell.	
#@ account_no = EDA	#BSUB -P EDA	
#@ group = enablement	#BSUB -G enablement	
#@ comment = "check size of AFS volume at CTE root"	#BSUB -Jd "check size of AFS volume at CTE root"	
#@ queue	The # LoadLeveler queue equivalent is inherent in directing this spool script as stdin to the **bsub** command.	

OsType=`uname`
if [[$OsType = 'Linux']]
then
fsLoc=/usr/afsws/bin
else
fsLoc=/usr/afsws/bin
fi
sleep 1000
$fsLoc/fs lq $CTEPATH
	OsType=`uname`
	if [[$OsType = 'Linux']]
	then
	fsLoc=/usr/afsws/bin
	else
	fsLoc=/usr/afsws/bin
	fi
	sleep 1000
	$fsLoc/fs lq $CTEPATH	

Mapping a LoadLeveler job command file to an LSF job spool script (2 of 2)

Table 8 shows the map between a LoadLeveler job command file and an LSF job spool script.

Table 8 LoadLeveler to LSF job commands mapping

Description	LoadLeveler job command file directive	LSF bsub job spool script directive	Comment
Shell interpreter, if command file is executable	#@ shell = /bin/ksh		
Environment variables available at job run time	#@ environment = ENVIRONMENT=BATCH	Export ENVIRONMENT=BATCH	
Job step runtime profile	#@ job_type = serial	#BSUB -R "span[hosts=1]"	LoadLeveler serial implies only one compute node for a job step, which is the most common type for EDA tools. Note that serial is the LoadLeveler default.

Description	LoadLeveler job command file directive	LSF bsub job spool script directive	Comment
Job identifier number	`$(job_id)`	`%I`	
Job identifier name	`#@ job_name = batch-test`	`#BSUB -J batch-test`	

`stdout`
`stderr`
	`#@ output = $(job_name).log`
	`#@ error = $(job_name).log`
		`#BSUB -o batch_test.%J_%I.out`
		`#BSUB -e batch_test.%J_%I.err`	

For job usage tracking	`#@ account_no = EDA`	`#BSUB -P EDA`	Required in Grid, enforced by `Sub_Filter`.
Job max elapsed time	`#@ wall_clock_limit = 1:00:00`	`#BSUB -W 60`	One hour (hh:mm:ss) = 60 minutes.
Job class or queue	`#@ class = batch`	`#BSUB -q batch`	
	`#@ notification = never`		
	`#@ resources = ConsumableCpus (min)`	`#BSUB -n min, max`	
	`#@ resources = ConsumableMemory (min)`		
	`#@ requirements`		
	`#@ requirements`		
	`#@ requirements`		
	`#@ queue`		Required in LoadLeveler.
	`myjob`		Executable to run.

LoadLeveler and LSF commands reference

Table 9 shows the LoadLeveler and LSF commands reference.

Table 9 LoadLeveler and LSF command reference

Command	Notes
llacctmrg	LSF has a different accounting record management mechanism. No LSF equivalent.
llbgctl	A Blue Gene-related command. No LSF equivalent.
llbgstatus	A Blue Gene-related command. No LSF equivalent.
llbind	**bmod**
llcancel	**bkill**
llchres	**brsvmod**
llckpt	**bchkpnt**
llclass	**bqueues**
llclusterauth	A LoadLeveler internal command. No LSF equivalent.

Command	Notes
llconfig	**badmin, lsadmin**
llctl	**badmin, lsadmin**
lldbupdate	A LoadLeveler internal command. No LSF equivalent.
llfavorjob	LSF uses a different method to calculate the relative job priority between the system and the job queue. LSF provides the **bmod** command to support adjusting the job priority dynamically.
llfavoruser	LSF uses a different method to calculate the relative job priority between the system and the job queue. LSF provides the **bmod** command to support adjusting the job priority dynamically.
llfs	LoadLeveler fair share scheduling command. No LSF equivalent.
llhold	**bstop**
llinit	**lsfinstall**
llmigrate	LSF does not provide job migration based on the checkpoint and restart function.
llmkres	**brsvadd**
llmodify	**bmod**
llmovejob	LoadLeveler multi-cluster command. No LSF equivalent.
llmovespool	LSF uses a different mechanism to maintain job-persistent data. It has its own way to recover the master node failure.
llpreempt	**bstop**, **bresume**, **brequeue**, and **bkill**
llprio	LSF uses a different method to calculate the job's relative priority between the system and the job queue. The **llprio** command is specific to LoadLeveler's framework, which LSF does not provide. LSF provides the **bmod** command to support adjusting the job priority dynamically.
llq	**bjobs**
llqres	**brsvs**
llrmres	**brsvdel**
llrun	The **llrun** command is an internal command for third-party MPI to run with LoadLeveler. LSF uses its own command with third-party MPI integration libraries, such as MPICH2, OpenMPI, and so on.
llrunscheduler	A LoadLeveler internal command. No LSF equivalent.
llstatus	**bhosts**
llsubmit	**bsub**
llsummary	The LSF **bacct** command provides similar functions to generate accounting report.

Command	Notes
`lltrace`	The LSF **`bhist`** command could be used to support the **`lltrace`** command **`-j`** option job trace functions. They are not exactly the same, but the purpose of these two commands is to help trace the job during its lifecycle.
`llxcatimp`	LSF does not support integration with the Extreme Cloud Administration Toolkit (xCAT).

LoadLeveler and LSF concept reference

Table 10 shows LoadLeveler concepts mapped to LSF.

Table 10 LoadLeveler and LSF concept reference

Description	LoadLeveler concept	LSF concept
Set of commands or directives that embody the job	Job command file	Job spool script
Shared CPU, memory, temporary disk space, and software license	(Compute) resource	(Compute) resource
Pool of computer resources	Cluster	Cluster
Single host with resources available in the cluster	Machine, host, and compute node	Host and computer
Group of hosts with resources available in the cluster	Not applicable	Host group
Host-based resources	Resources	Resource usage (rusage)
Host or platform unique capability or characteristic	Requirement and preferences (OpSys, Arch, Feature, Speed, and Machine)	Static resource selection and ordering (select and order)
Batch workload	Job (**`llsubmit`** and other **`ll*`** commands)	Job (**`bsub`** and other **`b*`** commands)
Interactive workload	Not applicable	Task (**`lsrun`** and other **`ls*`** commands)
Unit of work with resource requirements	Batch job	Batch job (%J) or interactive task
Sub-unit of work	Step	Job array index (%I)
► Cluster-wide job container ► Center of scheduling, with priority and policies ► Where jobs wait until they are scheduled and dispatched to hosts	Class	Queue
Fixed unit of potential work in a container	Slot	Slot
User	User	User
Group of predefined users, used for control and accounting	Not applicable	User group
Group of undefined users, used for control and accounting	Group	

Description	LoadLeveler concept	LSF concept
Usage accounting label	Account	Project
Cluster administrator	Cluster administrator	▶ Primary cluster administrator ▶ Secondary cluster administrator
Queue administrator	Cluster administrator	▶ Cluster administrator ▶ Queue administrator
User group administrator	Cluster administrator	▶ Cluster administrator ▶ User group administrator
Host group administrator	Cluster administrator	▶ Cluster administrator ▶ Host group administrator
Delegated administrative rights: ▶ Reduce administration load ▶ Remove need for administrator to manage internal project and priority changes ▶ Empower project managers and line of business owners ▶ Dynamically modify membership and fair share within the group	Not applicable	Delegated administration: ▶ Secondary cluster administrator ▶ Queue administrator ▶ User group administrator ▶ Host group administrator
..	Submit	Submit
Cluster central management host	Central manager and negotiator	Cluster workload manager (CWM)
Cluster central manager processes	`LoadL_negotiator`	▶ Cluster workload manager process ▶ **`mbatchd`** ▶ Master load information manager (LIM) process
Cluster job schedulers	Scheduler and job manager	
Cluster job scheduler processes	`LoadL_schedd`	**`mbschd`**
Dispatch processes		**`mbatchd`**
Host manager	Master	Host workload manager (HWM)
Host manager processes	`LoadL_master`	HWM process
Host monitor	Master	Subordinate LIM process information manager (PIM)
Host monitor processes	▶ `LoadL_master` ▶ Sentinel (Grid cron)	▶ Subordinate LIM process ▶ PIM process
Host job starter processes	▶ `LoadL_startd` (master) ▶ `LoadL_starter` (per job)	▶ **`sbatchd`** (master) ▶ **`sbatchd`** (child, per job)
Job monitor	▶ `LoadL_starter` ▶ Monitor (Grid)	**`sbatchd`** (child)
Job terminator	Terminator (Grid)	
Submit-only host	Submit-only host	Client host and submit-only client
Submit filter		

Description	LoadLeveler concept	LSF concept
Placeholder job for user’s direct-login interactive work	`getllmach.pl` (Grid)	
Backfill scheduler	Backfill scheduler	Backfill scheduler
Scheduler of many small jobs as a single job	Not applicable	Session scheduler
Common application parameters (such as `pre-exec`, `post-exec`, `resource limit`, and so on) for jobs of the same type		Application profile
► Resource reservation ► Advance reservation	► Resource reservation ► Advance reservation	► Resource reservation ► Advance reservation
CPU and memory load manager and enforcer	`LoadMgr` (Grid)	

Mapping LoadLeveler commands to LSF

Table 11 shows LoadLeveler commands mapped to LSF.

Table 11 Mapping LoadLeveler commands to LSF

Description	LoadLeveler command	LSF command	Comments
► Submit a command-file or job spool script for execution. ► Submit an executable program for execution. ► Submit via interactive command prompt.	► **`llsubmit ll-command-file.sh*`** ► Not applicable ► Not applicable	► **`bsub < lsf-spool-script.sh*`** ► **`bsub -i infile -o outfile -e errfile command`** ► **`bsub`**	Command can be any executable, such as a binary file or script.
Note: LSF behaves differently than LoadLeveler in parsing the shell commands that are combined with the directives. LSF requires that the job spool-script be available on the remote execution host in the same directory path as it is on the submission host. The shell commands in the LSF spool script are not parsed and captured at job-submission time, so are not sent to the remote execution host. The spool-script must be on the remote execution host at the time that the job is started. For example, if `lsf-spool-script.sh` is in the `/tmp` directory on job-submission host `hostA` (where the **`bsub`** command is run), the contents of `lsf-spool-script.sh` will not be visible for execution on remote-execution host `hostB`.			
► Show status of running and pending jobs. ► Display historical information about jobs.	► **`llq`** ► (Not applicable: use LLweb Memory Chart)	► **`bjobs`** ► **`bhist`**	
Cancel jobs.	**`llcancel`**	**`bkill`** (**`bdel`**)	
List available job classes.	**`llclass`**	**`bqueues`**	
Display information about users and groups.		**`busers`**	
Display accounting information about unfinished jobs.		**`bacct`**	
Display load on compute nodes.	**`llstatus`**	**`bhosts`**	
► Suspend a job. ► Resume a job.	► **`llhold`** ► **`llhold -r`**	► **`bstop`** ► **`bresume`**	

Description	LoadLeveler command	LSF command	Comments
Modify submission options for a submitted job.	**llmodify**	**bmod**	
Switch an unfinished job from one queue to another.		**bswitch**	
Preempt (suspend) a running job.	**llpreempt**		
Display **stdout** and **stderr** for an unfinished job.		**bpeek**	
Display cluster name and status.	**llstatus | grep Cluster**	**lsclusters**	
Display information about available hosts.	**llstatus**	**lsload, lshosts**	
Display information about available system features or resources.	Not applicable	**lsinfo**	
► Make an advance reservation for a job. ► Change an advance reservation for a job. ► Query status of advance reservation for a job. ► Bind job steps to run under an advance reservation.	► **llmkres** ► **llchres** ► **llqres** ► **llbind > llsubmit**		
Launch GUI.	**xloadl** (no longer supported)		
Identify cluster name and central manager.	**llstatus**	**lsid**	

The input files to **llsubmit** and **bsub** are similar, but differ in intent:

- The input file to **llsubmit** is a LoadLeveler job command file containing directives for **llsubmit** to parse. The input file to **llsubmit** can optionally be a shell script (or a job command file packaged as an executable shell script).
- The input to **bsub** can be an executable shell script in which **bsub** command-line options are embedded for **bsub** to parse. Alternatively, the job directives can be passed to **bsub** as command-line arguments, or via the **bsub** interactive command line.

Mapping LoadLeveler job states to LSF

Figure 3 shows the LoadLeveler job states.

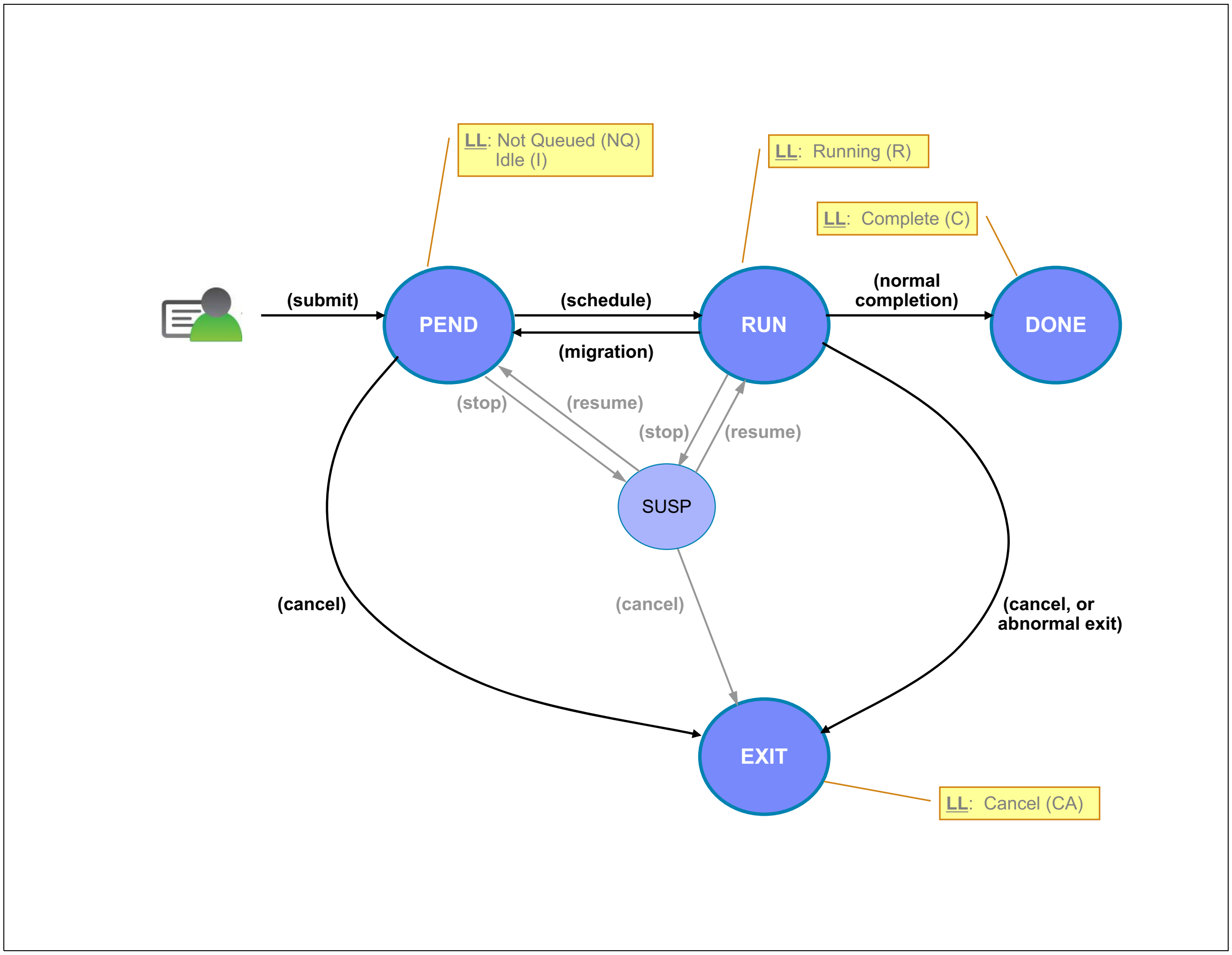
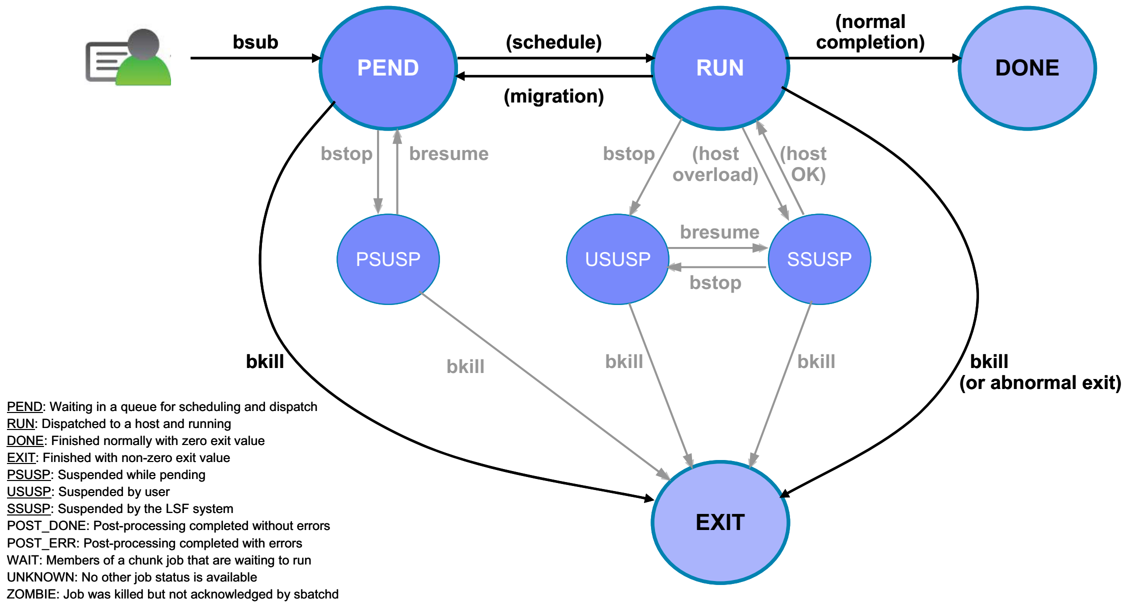


Figure 3   LoadLeveler job states

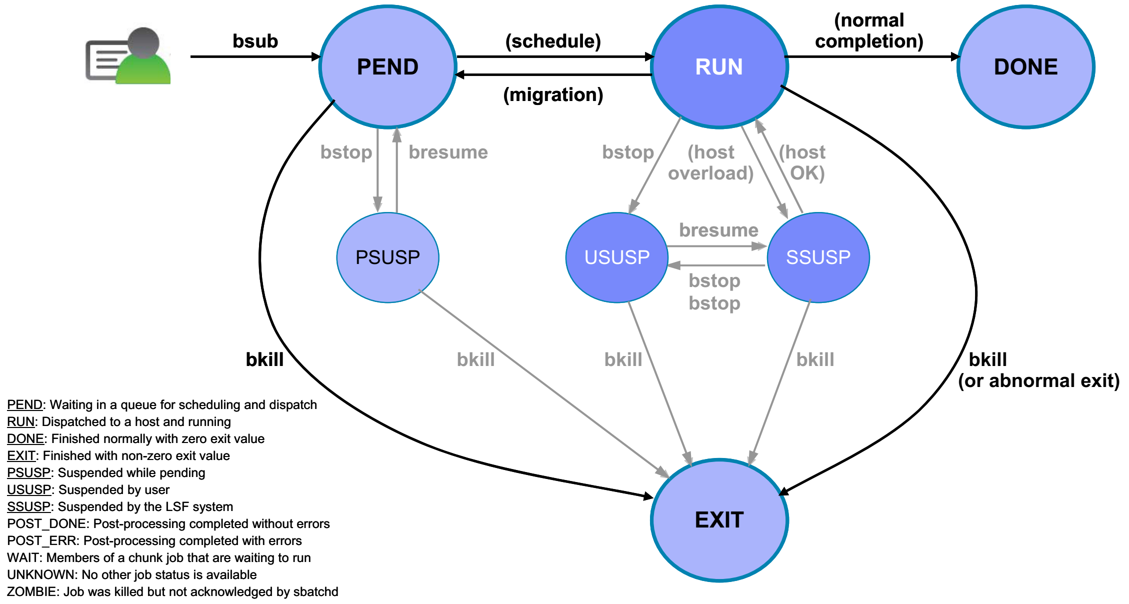
Figure 4 shows the LSF job states (queues).



PEND: Waiting in a queue for scheduling and dispatch
RUN: Dispatched to a host and running
DONE: Finished normally with zero exit value
EXIT: Finished with non-zero exit value
PSUSP: Suspended while pending
USUSP: Suspended by user
SSUSP: Suspended by the LSF system
POST_DONE: Post-processing completed without errors
POST_ERR: Post-processing completed with errors
WAIT: Members of a chunk job that are waiting to run
UNKNOWN: No other job status is available
ZOMBIE: Job was killed but not acknowledged by sbatchd

Figure 4 LSF job states (queues)

Figure 5 shows the LSF job states (hosts).



PEND: Waiting in a queue for scheduling and dispatch
RUN: Dispatched to a host and running
DONE: Finished normally with zero exit value
EXIT: Finished with non-zero exit value
PSUSP: Suspended while pending
USUSP: Suspended by user
SSUSP: Suspended by the LSF system
POST_DONE: Post-processing completed without errors
POST_ERR: Post-processing completed with errors
WAIT: Members of a chunk job that are waiting to run
UNKNOWN: No other job status is available
ZOMBIE: Job was killed but not acknowledged by sbatchd

Figure 5 LSF job states (hosts)

Table 12 shows, in bold, the most commonly observed LoadLeveler and LSF states. *Pending* is considered a temporary intermediate LoadLeveler job state.

The `llq` command summary appears in this order:

1. Number of job steps in query
2. Number waiting
3. Number pending
4. Number running
5. Number held
6. Number preempted

Table 12 Common LoadLeveler states

LoadLeveler state	LoadLeveler abbreviations	Treated like LoadLeveler	Counted as this in LoadLeveler `llq` summary	LSF state	LSF abbreviations	Treated like LSF
Canceled	**CA**	Stopped		Exited	EXIT	
Checkpointing	CK	Running				PEND
Completed	**C**	Stopped		Done	DONE	
Complete pending	CP	Stopped	Pending			EXIT
Deferred	D	Idle	Held			
Idle	**I**	Idle	Waiting	Pending	PEND	
Not queue	**NQ**	Idle	Held			PEND
Not run	NR					PEND
Pending	P	Running	Pending	Pending	PEND	
Preempted	E	Running	Preempted			USUSP
Preempt pending	EP	Running	Pending			USUSP
Rejected	X	Idle	Held			
Reject pending	XP	Idle	Pending			
Removed	RM	Stopped				EXIT
Remove pending	**RP**	Stopped	Pending			EXIT
Resume pending	MP	Running	Pending			RUN
Running	**R**	Running	Running	Running	RUN	
Starting	**ST**	Running	Running			RUN
System hold	**S**	Idle	Held	System suspend	SSUSP	
Terminated	TX	Stopped		Exited	EXIT	
User and system hold	HS	Idle	Held			

LoadLeveler state	LoadLeveler abbreviations	Treated like LoadLeveler	Counted as this in LoadLeveler `llq` summary	LSF state	LSF abbreviations	Treated like LSF
User hold	H	Idle	Held	► Pending suspend ► User suspend	► PSUSP ► USUSP	
Vacated	V	Idle	Held			
Vacate pending	VP	Idle	Pending			

Mapping LoadLeveler directives to LSF resources

LoadLeveler provides three directives to specify the system resource requirements for job execution:

- Requirements (**`requirements`**)
- Preferences (**`preferences`**)
- Resources (**`resources`**)

LSF provides six resource requirement specifications for job execution:

- Selection (**`select`**)
- Ordering (**`order`**)
- Resource usage (**`rusage`**)
- Job spanning (**`span`**)
- Same host type (**`same`**)
- Compute unit (**`cu`**)

Table 13 shows the LSF resources that encompass all of the system resource requirements defined by LoadLeveler requirements, preferences, and resources.

Table 13 LoadLeveler requirements, preferences, and resources

Description	LoadLeveler	LSF
Specifies the characteristics a host must have to match the resource requirement	**`requirements`**	**`select`**
Indicates how the hosts that meet the selection criteria should be sorted	**`preferences`**	**`order`**
Specifies the expected resource consumption of the job	**`resources`**	**`rusage`**
Indicates the locality of a distributed parallel job (if and how a parallel job should span across multiple hosts)		**`span`**
Indicates that all processes of a parallel job must run on the same type of host		**`same`**
Specifies computational unit requirements for spreading a job over the cluster		**`cu`**

Mapping LoadLeveler requirements to LSF resource selection (select string)

The following code sample shows the LSF resource requirements selection string syntax:

```
select[selection_string] order[order_string] rusage[usage_string [, usage_string]
[|| usage_string]...] span[span_string] same[same_string] cu[cu_string]
```

LSF provides for numerous system resources, built-in and external (custom), in two types, as shown in Table 14:

- Load indexes (dynamic)
- Static resources

Use the following notes to read and understand Table 14:

- Asterisk (*) indicates that the command is not supported in a job command file, and is for administration purposes only.
- Bold indicates an LSF built-in resource.
- Italics indicate an external or customized command.

Table 14 LoadLeveler and LSF system resources

Host resource	LoadLeveler `requirements`	LSF `select` load indices	LSF `select` static resources
Chip architecture	**Arch** - x86 - R6000	Not applicable	**type** (HostType TYPENAME) - X86_64 - IBMAIX64
Chip type and implementation	Not applicable Closest: Standardizing some of the non-common, site-specific LoadLeveler feature definitions	Not applicable	**model** (HostModel MODELNAME) - Xeon_X7560_2266 - Xeon_E7-8837_2666 - POWER5_x_1650
System type	Not applicable Closest: **Type-Model-Speed** in HWDB	Not applicable	**architecture** (HostModel ARCHITECTURE) - IBM_7145-AC1-2266 - IBM_7143-AC1-2666 - IBM_9115-505-1650
System performance factor	**Speed** Also known as **Perf**	Not applicable	**cpuf** (HostModel CPUFACTOR)
System operating system	**OpSys** - Linux26 - AIX61	Not applicable	Not applicable See **resource** (in the cell below)
System trait or chip trait	**Feature** - x86 x86_64 - POWER5 - HT - RH5 RH56 - AIX61 TL03 - GSA - *dedicated_batch* - *submit_only*	Not applicable	**resource** (Resource RESOURCENAME) - ht - rh5 rh56 - aix-6100-03-02 - gsa - *batch_only* - *submit_only*
Name	**Machine**	Not applicable	**hname**
Status	Not applicable	Not applicable	**status**

Host resource	**LoadLeveler `requirements`**	**LSF `select` load indices**	**LSF `select` static resources**
Can run remote jobs	Not applicable	Not applicable	**`server`** (boolean)
Execution priority	**`MACHPRIO*`**	Not applicable	**`rexpri`**
Number of CPUs	**`Cpus*`**	Not applicable	**`ncpus`**
Number of physical processors		Not applicable	**`nprocs`**
Number of cores per physical processor		Not applicable	**`ncores`**
Number of threads per processor core		Not applicable	**`nthreads`**
Number of local disks		Not applicable	**`ndisks`**
Maximum RAM memory available to users	**`Memory`**	Not applicable	**`maxmem`** (MB)
Maximum available swap space	**`VirtualMemory*`**	Not applicable	**`maxswp`** (MB)
Maximum available space in the `/tmp` file system		Not applicable	**`maxtmp`** (MB)
Maximum available space in the `/data` file system		Not applicable	*`maxdata`* (MB)
Run-queue depth		**`r15s`** (number of processes averaged over 15 seconds)	Not applicable
Run-queue depth		**`r1m`** (number of processes averaged over one minute) Also known as **`cpu`**	Not applicable
Run-queue depth	**`LoadAvg*`**	**`r15m`** (number of processes averaged over 15 minutes)	Not applicable
CPU utilization		**`ut`** (percentage averaged over one minute)	Not applicable
Paging activity		**`pg`** (pages-in plus pages-out per second)	Not applicable
Disk I/O		**`io`** (kilobytes per second averaged over one minute)	Not applicable

Host resource	**LoadLeveler `requirements`**	**LSF `select` load indices**	**LSF `select` static resources**
Logins		**ls** (number of users)	Not applicable
Idle time	**`Idle*`**	**it** (minutes)	Not applicable
Available space in the `/tmp` file system		**`tmp`** (MB)	Not applicable
Available swap space		**`swp`** (MB) Also known as **swap**	Not applicable
Available memory	**`FreeRealMemory*`**	`mem` (MB)	Not applicable
Available space in the `/data` file system		*`data`* (MB)	Not applicable
Other metric	*`CustomMetric*`*		

Authors

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Dino Quintero is a complex solutions project leader and IBM Senior Certified IT Specialist with the ITSO in Poughkeepsie, NY. His areas of knowledge include enterprise continuous availability, enterprise systems management, system virtualization, technical computing, and clustering solutions. He is currently an Open Group Distinguished IT Specialist. Dino holds a Master of Computing Information Systems degree and a Bachelor of Science degree in Computer Science from Marist College.

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

Ella Buslovich
International Technical Support Organization, Poughkeepsie Center

Bill S McMillan
IBM UK

Jim Smith
IBM Canada

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Obtain more information about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks
- Follow us on Twitter:

  http://twitter.com/ibmredbooks
- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm
- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

This document, REDP-5048-00, was created or updated on September 18, 2013.

Send us your comments in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AFS™
AIX®
Blue Gene®
IBM®
LoadLeveler®
LSF®
POWER®
Redbooks®
Redpaper™
Redbooks (logo) ®
Tivoli®

The following terms are trademarks of other companies:

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.