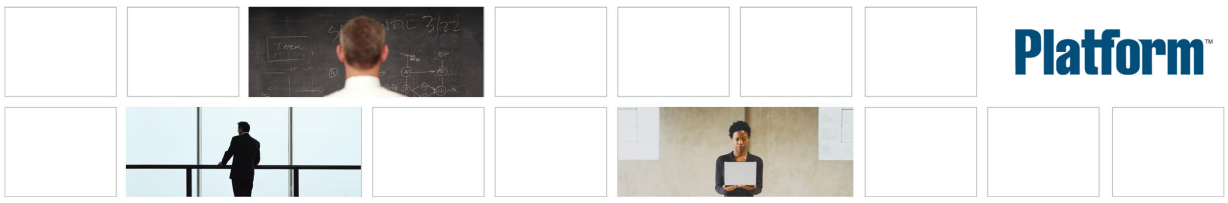
Using Platform LSF License Scheduler

Platform LSF License Scheduler
Version 7.0 Update 6
Release date: July 2009
Last modified: July 17, 2009



Contents

CHAPTER

1

Introduction

About Platform LSF License Scheduler

Platform LSF License Scheduler allows you to make policies that control the way software licenses are shared among different users in your organization. Platform LSF License Scheduler works with FLEXnet™ products to control and monitor license usage.

Distribute licenses intelligently

LSF License Scheduler intelligently distributes application licenses based on configurable policies to support

- Fairshare of licenses
- License preemption when license ownership is defined
- Awareness of LAN and WAN service domains for fault tolerance and failover
- Geographically distributed license servers
- Interactive and batch jobs running in LSF clusters

Complete projects faster

LSF License Scheduler enables faster project completion by

- Guaranteeing license availability for different projects and groups across physical locations
- Ensuring access to licenses so that license owners get their fair share of usage

Maximize your investment

LSF License Scheduler maximizes your investment in expensive application licenses by

- Optimizing usage of existing licenses
- Easing the configuration of a dynamic shared pool of licenses. Instead of assigning arbitrary shares of licenses to everyone, you can give more licenses to larger or more important projects
- Guaranteeing access to a minimum portion of licenses, no matter how heavily loaded the system is
- Controlling the distribution and scheduling of licenses among jobs it manages, without preventing users from checking out licenses directly
- Treating application licenses as another compute resource to be scheduled and managed

Terms

The following list provides brief descriptions of LSF License Scheduler terms that we use in this guide.

- allocation
- blcollect
- bld
- collector
- default license project
- failover host
- failover provision
- group ownership
- interactive job
- license project
- license server
- lmgrd
- non-shared license
- ownership
- preemption
- service domain
- shared license
- token

allocation

The distribution of license tokens between different LSF clusters.

Allocation takes place before you share the tokens. Allocation is a superset of project-based distribution policies. You allocate license tokens across clusters or between interactive and LSF jobs.

blcollect

The LSF License Scheduler daemon that queries FLEXnet licensing software for license usage. blcollect can collect information from `lmstat`.

By default, license information is collected from FLEXnet licensing on one host. You can distribute the license collection on multiple hosts by running the license information collection daemon, blcollect.

If the data from all your license servers is collected in one central location, mbatchd has to wait for the license usage information of all your license servers from the output of `lmstat`. With LSF License Scheduler, you can distribute the query to collect the information in parallel from each license server.

bld

The LSF License Scheduler batch daemon.

collector

The term used to describe the LSF License Scheduler daemon `blcollect` that queries FLEXlm for license usage information.

With LSF License Scheduler, you can distribute the query to collect the information in parallel from each license server. Run the license collectors on any machines you want. Each collector can query one or more license servers.

default license project

A license project that is not specified in job submissions, but uses license features that are managed by LSF License Scheduler.

All jobs requiring a license feature that is managed by LSF License Scheduler, and which are not submitted to a configured project for the feature, are treated as jobs submitted to the default project. However, if LSF_LIC_SCHED_STRICT_PROJECT_NAME=y in `lsf.conf` and you have not configured a default project in DISTRIBUTION parameter, the job is rejected rather than submitted to the default project.

failover host

A candidate LSF License Scheduler host that runs the LSF License Scheduler daemon (bld), and can take over license management if the LSF License Scheduler host fails or loses its connection to the network.

You can configure LSF License Scheduler for failover in a LAN or a WAN configuration by listing the hosts in order of their preferred candidacy.

failover provision

The configuration of LSF License Scheduler hosts to take over license management in case of a host failure or network breakdown.

LSF License Scheduler can be configured for failover provision in both LANs and WANs:

- In LSF, the LSF LIM daemon runs the LSF License Scheduler daemon (bld) on hosts you specify in an LSF host list.
- In LSF License Scheduler, you specify a candidate host list— hosts that can take over license management if the LSF License Scheduler host fails.

group ownership

An extra level of hierarchy added to the distribution of license tokens among license projects.

Optionally group your projects, then grant license ownership and shares to the whole group. Preemption between license projects only occurs if the whole group has used more licenses than it owns.

interactive job

A non-LSF job that is run by the LSF Task Manager (`taskman`) tool outside of LSF, but is scheduled by LSF License Scheduler.

You can allocate licenses for interactive jobs.

license project

A project you configure in LSF License Scheduler and which you associate with your job submissions.

You submit jobs to license projects. LSF License Scheduler then distributes tokens among the license projects based on the shares you define for each license project in distribution policies.

Use the `bsub -Lp` option to submit jobs to license projects.

Tip:

Although license projects are not the same as LSF projects, you can map your license project names to LSF project names for easier monitoring.

license server

Serves licenses to jobs requiring license features.

License Scheduler works with the FLEXlm license server. FLEXlm serves licenses—it does not schedule licenses. License Scheduler reserves licenses for you by distributing license tokens that you can use to check out your licenses from FLEXlm.

lmgrd

The main FLEXlm licensing daemon. Usually denoted by *port@host_name* and grouped into service domains inside License Scheduler.

non-shared license

A license that cannot be shared with other projects.

ownership

The right of a license project to use its licenses on demand, while still allowing License Scheduler to distribute the licenses to other license projects when the project is not using them.

preemption

Occurs when a project has to release a license it is using to a project that demands that license because it owns it.

Preemption only occurs when there are no free licenses.

Jobs using licenses that support job suspension release their tokens and automatically resume from where they were suspended. Jobs using licenses that do not support suspension are killed and restarted from the beginning.

service domain

A group of one or more FLEXlm license server hosts that serve licenses to LSF jobs.

You configure the service domain with the names and port numbers of the license server hosts that serve licenses to a network. For example, you can configure one service domain for Design Center A, and another service Domain for Design Center B. Both service domains can contain multiple license servers.

shared license

A license that, when free, can be distributed fairly among license projects.

You create distribution policies to share licenses among projects. Each license project is entitled to a minimum portion of the available licenses.

token

A license reservation that determines which job is dispatched next.

License Scheduler manages license tokens instead of controlling the licenses directly. After reserving licenses, jobs are dispatched, then the application that needs the license is started. The number of tokens available from LSF corresponds to the number of licenses available from FLEXlm, so if a token is not available, the job is not dispatched.

Architecture

LSF License Scheduler manages license tokens instead of controlling the licenses directly. Using LSF License Scheduler, jobs receive a license token before starting the application. The number of tokens available from LSF corresponds to the number of licenses available from FLEXlm, so if a token is not available, the job does not start. In this way, the number of licenses requested by running jobs does not exceed the number of available licenses.

When a job starts, the application is not aware of LSF License Scheduler. The application checks out licenses from FLEXlm in the usual manner.

Non-LSF jobs

Jobs that start outside of LSF do not receive a license token, but they can still check out a license. LSF automatically adjusts the total number of licenses managed to compensate for the licenses that have been taken by non-LSF jobs.

LSF ELIM not needed

Using License Scheduler, you do not need to configure custom resources or write an ELIM. LSF automatically sets up license tokens as LSF resources and makes ELIM redundant.

No lsf.shared setup

You do not need to define the license as a shared LSF resource in `lsf.shared`.

No lsf.cluster.*cluster_name* setup

You do not need to define the license as a shared LSF resource in `lsf.cluster.`*cluster_name*; just configure license projects, which you include in the distribution policy for that license.

No ELIM

Do not write an ELIM to monitor the license. With License Scheduler configured, LSF is aware of the actual license availability. If the job can receive a license token, it is guaranteed to receive an actual license when required.

LSF License Scheduler and host reliability

You can define a list of candidate hosts for LSF License Scheduler in case of a host failure. The LSF License Scheduler daemon (`bld`) runs on the candidate hosts and maintains a connection between each candidate and the LSF License Scheduler host.

Failover in a LAN

If the LSF License Scheduler host fails, the first candidate host listed in the License Scheduler host list takes over the license scheduling until the master host restarts. It must be running the LSF License Scheduler daemon.

Failover in a WAN

If License Scheduler is managing licenses in a WAN configuration, and the connection between sites breaks, a candidate LSF License Scheduler host manages license scheduling locally until the WAN connection returns.

Using License Scheduler in a WAN

The following examples illustrate the benefits of using License Scheduler to manage license tokens in a WAN. In these examples, the license server in Design Center A can only serve licenses to jobs from Design Center A. The license server in Design Center B, however, can serve licenses to jobs from both centers.

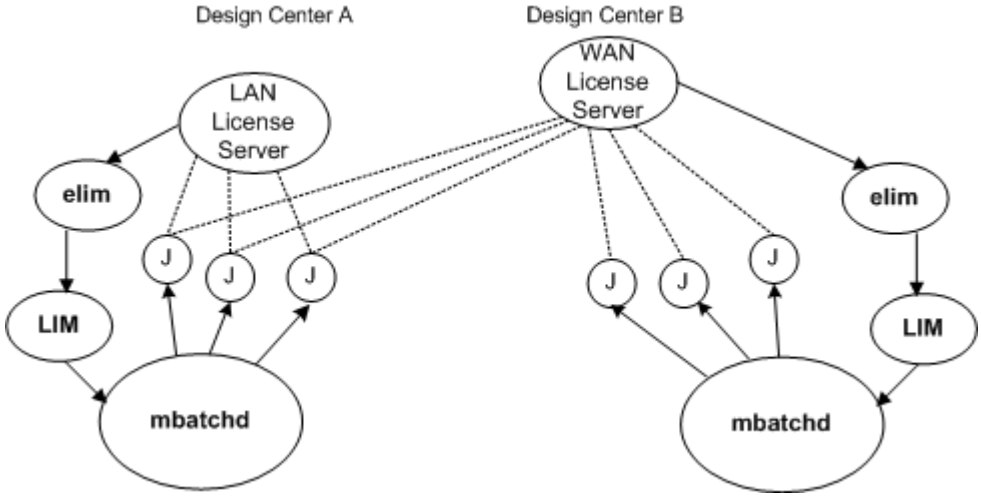


Figure 1: Two design centers without LSF License Scheduler

In this example

The `elim` collects license information from the FLEXlm license server host (LAN or WAN) and reports back to the LSF cluster master batch daemon (`mbatchd`) through `LIM`. When the LSF cluster starts jobs, the decision is based on license availability. The jobs check out the licenses directly from the server.

Interactive jobs check out licenses directly from the server without any scheduling controls.

This example shows two potential problems:

- Uncontrolled competition for license checkout can lead to a race condition that can result in job failure for some users.
- There is no way to balance license usage among multiple projects or multiple sites.

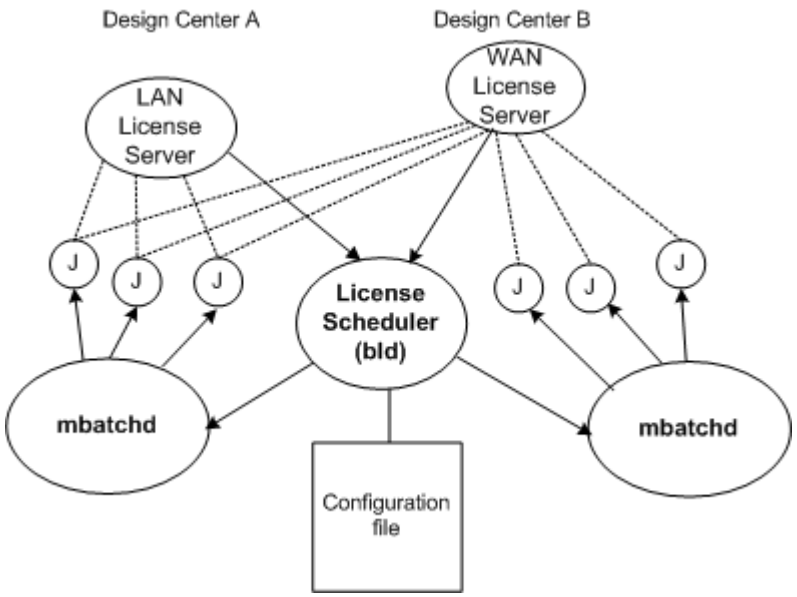


Figure 2: Two design centers with LSF License Scheduler

In this example

LSF License Scheduler collects license information from the FLEXlm license server host (LAN or WAN). The LSF cluster daemon (`mbatchd`) receives tokens from License Scheduler and starts jobs. The jobs check out the license directly from the server.

1. LSF License Scheduler collects the information related to licenses:
   - License availability and license usage from the FLEXlm license server hosts
   - License demand and license usage from LSF clusters and interactive users
2. Based on the information it collects, and on its scheduling and distribution policies, License Scheduler makes license distribution and preemption decisions.

Because License Scheduler distributes each license to only one license project, there is no race condition among multiple users. Because License Scheduler is a central point of control, scheduling policies can include multiple LSF clusters and non-LSF users.

LSF scheduling policies

With LSF License Scheduler, LSF gathers information about the licensing requirements of pending jobs to efficiently distribute available licenses. Other LSF scheduling policies are independent from LSF License Scheduler policies.

When starting a job, the basic LSF scheduling comes first.

- Assign a suitable LSF host before considering the requirements of any other resources, like licenses.

For example, a job must have a candidate LSF host on which to start before the LSF License Scheduler fairshare policy (for the license project this job belongs to) will apply.

- Other LSF fairshare policies are based on CPU time, run time, and usage. If LSF fairshare scheduling is configured, LSF determines which user or queue has the highest priority, then considers other resources. In this way, the other LSF fairshare policies have priority over LSF License Scheduler.

Offline behavior

- mbatchd

If `mbatchd` is offline while reconfiguring LSF or because of an unexpected failure of LSF software, tokens distributed to license projects in the unavailable cluster will be redistributed to other projects. When `mbatchd` comes back online, it immediately receives updated information about the number of tokens currently distributed to its projects in its cluster.

When LSF is reconfigured (`badmin reconfig`) the bld restarts. (Platform LSF Version 7 Update 5 onwards.)

- LSF License Scheduler

If `mbatchd` cannot contact LSF License Scheduler, it does not receive any updated information about the number of tokens dynamically distributed to the projects in its cluster, so it continues to run using the most recent data available.

LSF License Scheduler policies

LSF License Scheduler policies distribute license tokens among license projects, which you create and configure in the `lsf.licensescheduler` configuration file. The following cases describe how LSF License Scheduler policies can help license projects to share licenses.

Fair sharing

The License Scheduler distribution policy guarantees that each license project is entitled to a minimum portion of the available licenses.

Example

Create three LSF License Scheduler projects, and share the licenses equally. If one project does not need a license, another project can use it. In this case, dynamic redistribution of licenses maximizes utilization while enforcing the fair share policy.

Not all license projects in a cluster have a full workload at all times. Free licenses can be shared across projects, so that idle licenses for one project are available to other projects.

Round robin sharing

This example shows how to configure round robin sharing. Round robin sharing is required when there are fewer licenses than the number required by license projects. The policy can be configured in the same way as the fair sharing policy.

If the total number of licenses is smaller than the number required by license projects, and all projects need more licenses, then the projects take turns using the licenses.

Example

Create three LSF License Scheduler projects, and share the licenses equally. The projects take turns to use the license. The three license projects share one license.

Jobs having a run time of 5 to 10 minutes are pending for each of the three projects. The projects share the license based on round robin policy.

Preemption

This example shows how license ownership and preemption work.

License ownership gives license projects the right to use their licenses on demand, while still allowing LSF License Scheduler to distribute the licenses to other license projects when the owner is not using them.

Example

Create two LSF License Scheduler projects, `Lp1` and `Lp2`, and share the licenses, but grant ownership of the licenses to one of the projects (`Lp2`).

When `Lp2` has no work to be done, `Lp1`. can use the licenses. When `Lp2` has work to do, `Lp1` must return the license immediately to `Lp2`. The license utilization is always at the maximum, showing that all licenses are in use even while the license distribution policies are being enforced.

License data example output

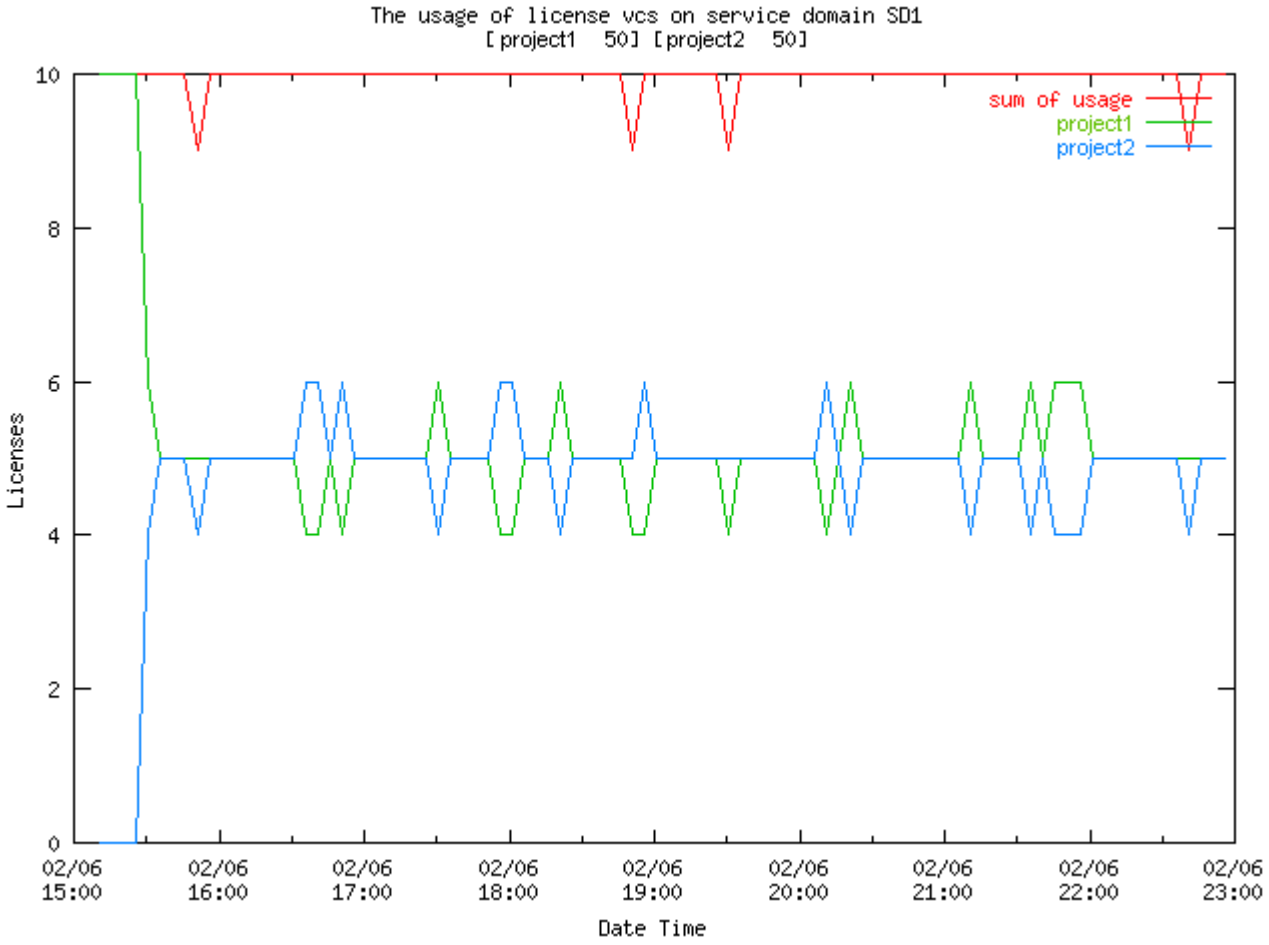


Figure 3: 50:50 fairshare between two license projects

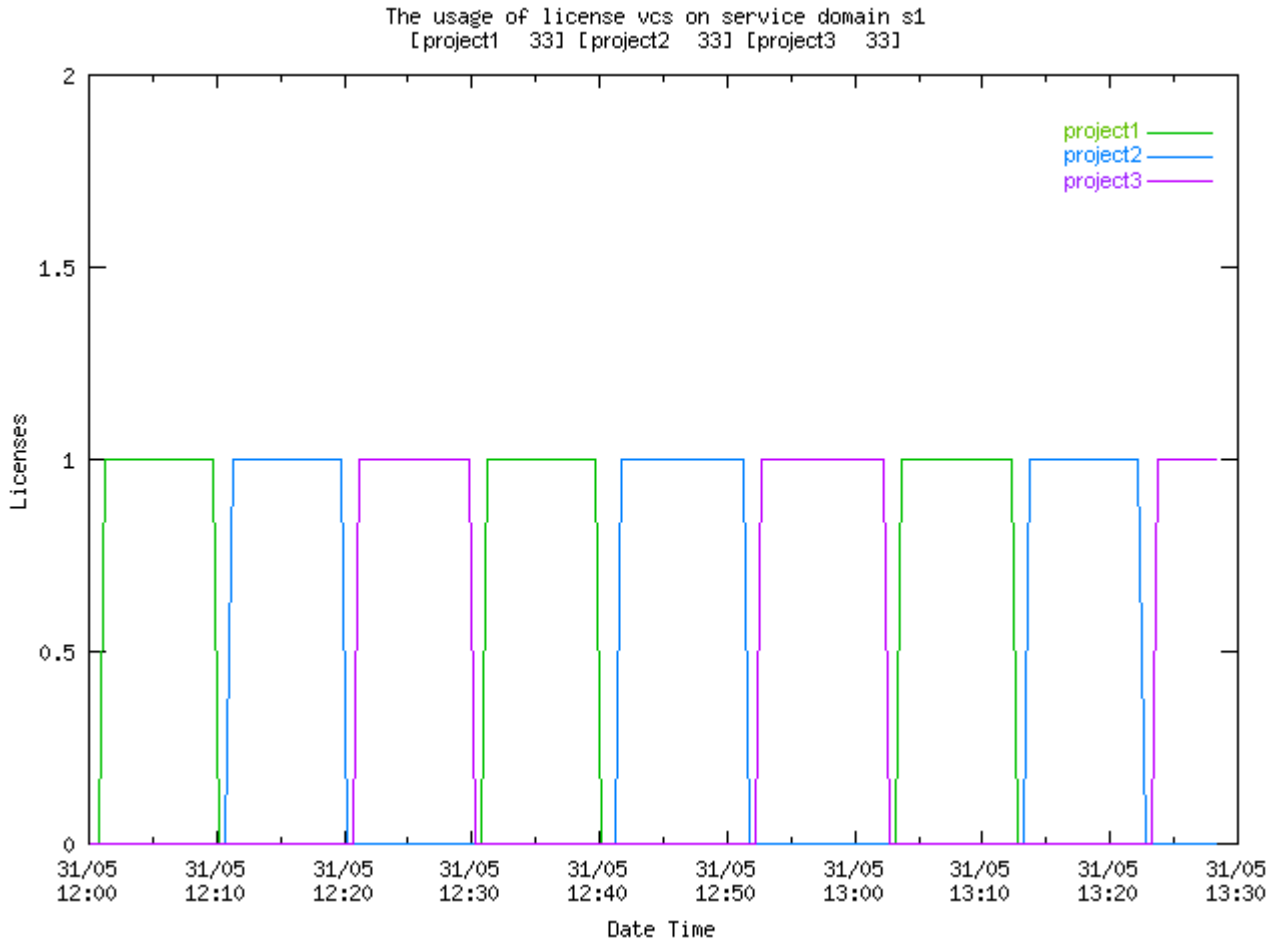


Figure 4: Round robin sharing of one license among three license projects

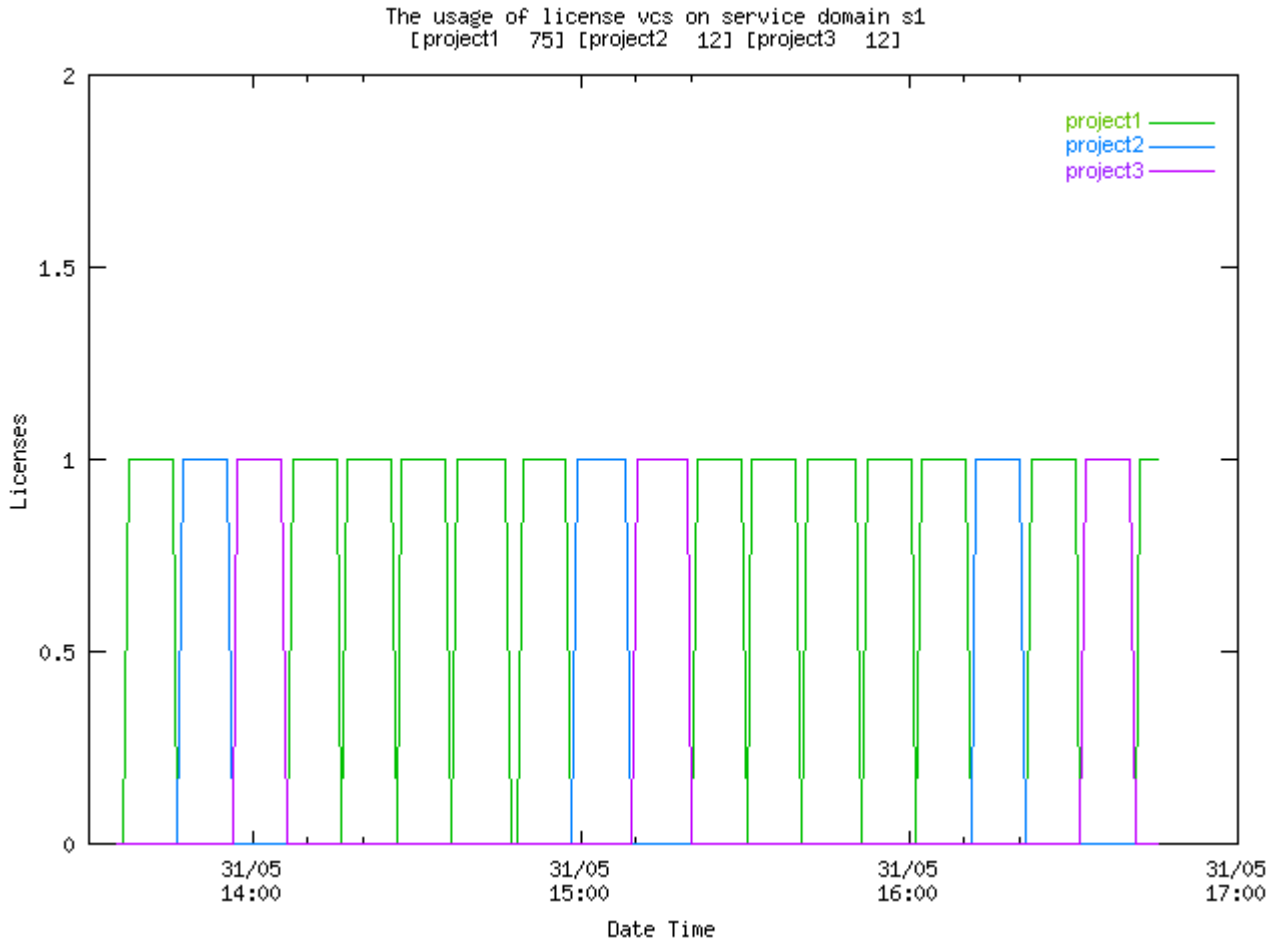


Figure 5: Round robin sharing of one license among three license projects

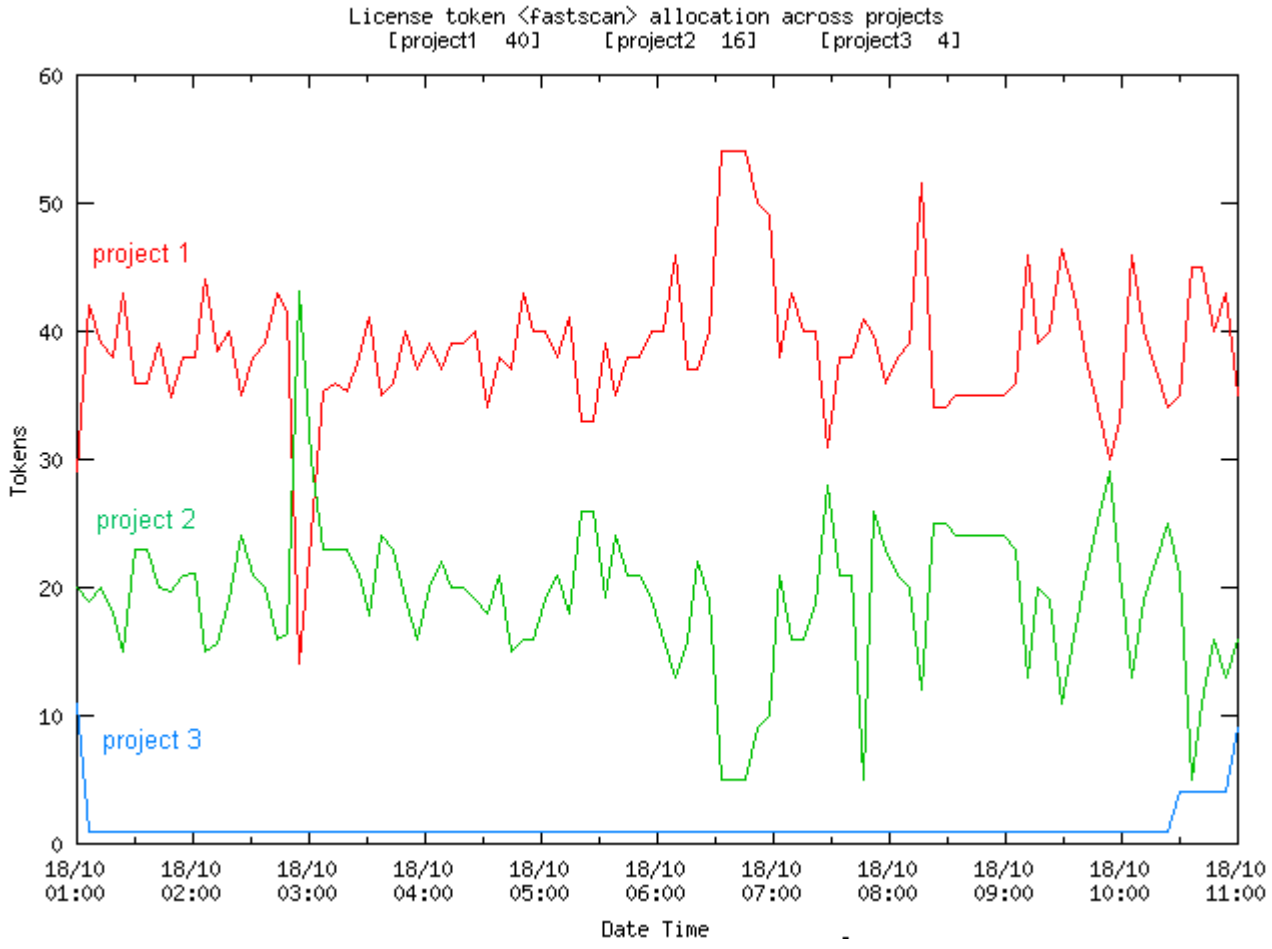


Figure 6: Load inversion

CHAPTER

2

Installing and Configuring Platform LSF License Scheduler

Install Platform LSF License Scheduler

1. Perform the pre-installation steps.
2. Choose an installation method:
   - Install LSF License Scheduler with LSF (UNIX)
   - Install LSF License Scheduler with LSF (Windows)
   - Install LSF License Scheduler standalone (UNIX)
   - Install LSF License Scheduler standalone (Windows)

Before you install

Platform LSF must be installed and running before installing LSF License Scheduler.

If you are installing LSF License Scheduler as a standalone product without Platform LSF, you do not need to modify LSF in preparation for License Scheduler. However, you still need to get an LSF License Scheduler license before installation.

1. Get an LSF License Scheduler license from Platform Computing:
   a) Send the host name and host identifier of the license server host to Platform at license@platform.com or to your LSF vendor.
   b) Check the LSF_LICENSE_FILE parameter in `lsf.conf` to locate the LSF license file.
   c) Add the LSF License Scheduler (lsf_license_scheduler) feature line to your existing LSF license file. For example:

```
FEATURE lsf_license_scheduler lsf_ld 7.000 1-jun-0003 1
3C0733892E1683812345 "Platform"
```

   d) For a permanent license, restart the LSF `lmgrd`.
2. Log on to any LSF host as root and use `lsid` to make sure the cluster is running. If you see the message "Cannot open `lsf.conf` file", the LSF_ENVDIR environment variable is probably not set correctly.

   To set your LSF environment:

   - For `csh` or `tcsh`:

     % **source LSF_TOP/conf/cshrc.lsf**
   - For `sh`, `ksh`, or `bash`:

     $ **. LSF_TOP/conf/profile.lsf**

What the LSF License Scheduler setup script does

- Finds the appropriate `lsf.conf` for the running cluster
- Copies the LSF License Scheduler files to your LSF directories:
- $LSF_ENVDIR
- $LSF_SERVERDIR
- $LSF_BINDIR
- LSF_MANDIR
- Finds the appropriate `lsf.cluster.`*cluster_name* file for the running cluster
- Creates the following additional directories:
- `LSB_SHAREDIR/`*cluster_name*`/db`

- `LSB_SHAREDIR/`*`cluster_name`*`/data`
- Sets your LSF License Scheduler administrators list in the lsf.licensescheduler file.
- Configures LSF to use License Scheduler

Install LSF License Scheduler with LSF (UNIX)

1. Log on as root to the installation file server host.

   You need to be able to write into the LSF_TOP directories.
2. Download, uncompress, and extract the LSF License Scheduler packages for the platforms you need from the directory `/license_scheduler_ls7_update5/`.

   For example, for x86 systems running Linux Kernel 2.4.x and compiled with glibc 2.3.x:

   ```
   ftp> get lsf7update5_licsched_linux2.4-glibc2.3-x86.tar.Z
   ```

   Make sure that you download the LSF License Scheduler distribution files to the same directory where you downloaded the LSF product distribution tar files.
3. Extract the distribution file.

   For example:

   ```
   # zcat lsf7update5_licsched_linux2.4-glibc2.3-x86.tar.Z | tar xvf -
   ```
4. Change to the extracted distribution directory.

   For example:

   ```
   # cd lsf7update5_licsched_linux2.4-glibc2.3-x86
   ```
5. Run the `setup` script as root:

   ```
   # ./setup
   ```
6. Enter y (yes) to confirm that the path to `lsf.conf` is correct.

   To enter a path to a different `lsf.conf`, type n (no) and specify the full path to the `lsf.conf` file you want to use.
7. Enter y to confirm that the path to `lsf.cluster.`*`cluster_name`* is correct.

   To enter a path to a different `lsf.cluster.`*`cluster_name`* file, type n (no) and specify the full path to the `lsf.cluster.`*`cluster_name`* file you want to use.
8. Enter y to confirm that you want to use the LSF Administrators list for License Scheduler with LSF.

   To enter a different list of administrators for License Scheduler, enter a space-separated list of administrator user names. You can change your License Scheduler administrators list later, if desired.

Install LSF License Scheduler with LSF (Windows)

1. Log on as an OS administrator to the Windows client.

   You need to be able to write into the LSF_TOP directories.
2. Download the Platform License Scheduler Windows Client distribution package for the platforms you need from the directory `/license_scheduler_ls7_update5/`.
3. Extract the distribution file.

The distribution package contains binary files for the Platform License Scheduler user commands (`*.exe`) and the LSF License Scheduler configuration file (`lsf.licensescheduler`).

4. Copy all user command binary files to the `%LSF_BINDIR%` directory in your Windows host.
5. Copy the LSF License Scheduler configuration file (`lsf.licensescheduler`) to the `%LSF_ENVDIR%` directory in your Windows host and edit the file to match your LSF License Scheduler master host configuration.

Install LSF License Scheduler standalone (UNIX)

1. Log on as root to the installation file server host.

   You need to be able to write into the LSF_TOP directories.
2. Download, uncompress, and extract the LSF License Scheduler packages for the platforms you need from the directory `/license_scheduler_ls7_update5/`.

   For example, for x86 systems running Linux Kernel 2.4.x and compiled with glibc 2.3.x:

   ```
   ftp> get lsf7update5_licsched_linux2.4-glibc2.3-x86.tar.Z
   ```

   Make sure that you download the LSF License Scheduler distribution files to the same directory where you downloaded the LSF product distribution tar files.
3. Extract the distribution file.

   For example:

   ```
   # zcat lsf7update5_licsched_linux2.4-glibc2.3-x86.tar.Z | tar xvf -
   ```
4. Change to the extracted distribution directory.

   For example:

   ```
   # cd lsf7update5_licsched_linux2.4-glibc2.3-x86
   ```
5. Edit the `setup.config` file and set the parameters you need for installation. You must specify LS_TOP and LS_ADMIN.

   Do not edit `setup.config` if you are installing LSF License Scheduler to work with Platform LSF.
6. Run the `setup` script as root:

   ```
   # ./setup
   ```
7. Enter y to confirm that you want to use the administrator list in `setup.config` for a standalone License Scheduler installation.

   To enter a different list of administrators for License Scheduler, enter a space-separated list of administrator user names. You can change your License Scheduler administrators list later, if desired.

Install LSF License Scheduler standalone (Windows)

1. Log on as an OS administrator to the Windows client.
2. Create a new top-level directory for LSF License Scheduler with a directory structure that is similar to the LSF_TOP directory structure, including `bin`, `conf`, and `etc` subdirectories.

   For example, create `C:\LS7.0` as your top-level LSF License Scheduler directory with the following subdirectories:

```
C:\LS7.0\bin
```

```
C:\LS7.0\conf
```

```
C:\LS7.0\etc
```

3. Download the Platform License Scheduler Windows Client distribution package for the platforms you need from the directory `/license_scheduler_ls7_update5/`.
4. Extract the distribution file.

   The distribution package contains binary files for the Platform License Scheduler user commands (`*.exe` files) and the LSF License Scheduler configuration file (`lsf.licensescheduler` file).
5. Copy all user command binary files to the `bin` subdirectory in your Windows client.

   For example, copy all user command binary files to the `C:\LS7.0\bin` directory.
6. Copy the `lsf.licensescheduler` and `lsf.conf` files to the `conf` subdirectory in your Windows client.

   For example, copy the `lsf.licensescheduler` and `lsf.conf` files to the `C:\LS7.0\conf` directory.
7. Set the `%LSF_BINDIR%` environment variable to the `bin` subdirectroy in your Windows client.

   For example, set `%LSF_BINDIR%` to `C:\LS7.0\bin`.
8. Set the `%LSF_ENVDIR%` environment variable to the `conf` subdirectroy in your Windows client.

   For example, set `%LSF_ENVDIR%` to `C:\LS7.0\conf`.
9. Set the `%LSF_SERVERDIR%` environment variable to the `etc` subdirectroy in your Windows client.

   For example, set `%LSF_SERVERDIR%` to `C:\LS7.0\etc`..

Configure LSF License Scheduler

LSF License Scheduler automatically distributes unused licenses to the projects that need them. Your configured distribution policies take effect when the system is fully loaded and there is competition for resources. You can update the license distribution policies and change the share allocations at any time. You can also update the configuration to add new licenses or new projects.

Configuration files

The LSF License Scheduler configuration files are located in $LSF_ENVDIR.

Tip:

See the Platform LSF Reference for details about the LSF License Scheduler configuration parameters.

lsf.licensescheduler

The `lsf.licensescheduler` file contains the LSF License Scheduler configuration information, including the license distribution policies, which describe how many license features are controlled by LSF and how the licenses are to be shared or owned in the event of competition among projects.

The following sections are required:

- Parameters — License Scheduler configuration parameters
- Projects — lists the License Scheduler projects
- Clusters — lists the clusters that can use License Scheduler
- ServiceDomain — defines License Scheduler service domains as groups of physical license server hosts that serve a specific network
- Feature — defines license distribution policies for application license features

The ProjectGroup section is optional — defines hierarchical relationships among projects.

Use a line continuation character "\" to continue a line

lsf.conf

Parameters in `lsf.conf` that start with LSF_LIC_SCHED are relevant to both LSF and License Scheduler:

- LSF_LIC_SCHED_HOSTS — LIM starts the License Scheduler daemon (`bld`) on candidate License Scheduler hosts.

Caution:

You cannot use LSF_LIC_SCHED_HOSTS if your cluster was installed with UNIFORM_DIRECTORY_PATH or UNIFORM_DIRECTORY_PATH_EGO. *Do not* set UNIFORM_DIRECTORY_PATH or UNIFORM_DIRECTORY_PATH_EGO for new or upgrade installations. They are for backwards compatibility only.

- LSF_LIC_SCHED_PREEMPT_REQUEUE — requeues a job whose license is preempted by License Scheduler. The job will be killed and requeued instead of suspended.
- LSF_LIC_SCHED_PREEMPT_SLOT_RELEASE — releases the slot of a job that is suspended when the its license is preempted by License Scheduler.

- LSF_LIC_SCHED_PREEMPT_STOP — uses job controls to stop a job that is preempted. When this parameter is set, a UNIX SIGSTOP signal is sent to suspend a job instead of a UNIX SIGTSTP.
- LSF_LIC_SCHED_STRICT_PROJECT_NAME—enforces strict checking of the License Scheduler project name upon job submission. If the project named is misspelled (case sensitivity applies), the job is rejected.

License Scheduler uses the following LSF parameters:

- LSB_SHAREDIR — directory where the job history and accounting logs are kept for each cluster
- LSF_LICENSE_FILE — one or more demo or FLEXnet-based permanent license files used by LSF
- LSF_LICENSE_ACCT_PATH — location for the license accounting files, including the license accounting files for LSF Family products
- LSF_LOG_MASK — logging level of error messages for LSF daemons
- LSF_LOGDIR — LSF system log file directory

When you change your configuration

After making any change to `lsf.licensescheduler`:

1. Use `bld -C` to test for configuration errors.
2. Run `bladmin reconfig all` to reconfigure LSF License Scheduler and make the changes take effect.

After making any change to `lsf.conf` or other LSF configuration files:

1. Use `bld -C` to test for configuration errors.
2. Run `bladmin reconfig all` to reconfigure LSF License Scheduler and make the changes take effect.
3. Run badmin mbdrestart to restart mbatchd.

Note:

When LSF is reconfigured (`badmin reconfig`) bld a restarts. (Platform LSF Version 7 Update 5 onwards.)

Example basic configuration

The following example configures one cluster with two license servers. The policy schedules licenses for one application feature using a fairshare distribution for two projects.

Log on as the primary License Scheduler administrator and edit the License Scheduler configuration in LSF_CONFDIR/`lsf.licensescheduler` to

- Configure parameters
- Configure projects
- Configure clusters
- Configure service domains
- Configure license features
- Configure hierarchical project groups (optional)

Configure parameters

HOSTS

HOSTS=*host_name_1 ... host_name_n ...*

List the License Scheduler hosts, including License Scheduler candidate hosts.

- *hostname_1* is the most preferred host for running LSF License Scheduler.
- *hostname_n* is the least preferred host for running LSF License Scheduler.

By default, the HOSTS parameter is set to the LSF_MASTER_LIST during installation with LSF. The License scheduler daemon (`bld`) can only start on the hosts listed in the HOSTS parameter. The first host is the primary license scheduler, and the other hosts are failover backups.

Specify a fully qualified host name such as hostX.mycompany.com. You can omit the domain name if all your License Scheduler clients run in the same DNS domain.

LM_STAT_INTERVAL

LM_STAT_INTERVAL=*seconds*

Specify the frequency in seconds to collect data from FLEXnet servers.

LM_STAT_INTERVAL defines a time interval between calls that License Scheduler makes to collect license usage information from FLEXnet.

During License Scheduler installation, the interval is set to 60 seconds.

LMSTAT_PATH

LMSTAT_PATH=*path*

Specify the full path to the location of the FLEXnet command `lmstat`.

For example, if `lmstat` is located in `/etc/flexlm/bin`:

```
LMSTAT_PATH=/etc/flexlm/bin
```

Tip:

If the lmstat command is not included in the flexlm/bin directory, you will find it packaged with your LSF distribution in LSF_SERVERDIR.

Example

```
Begin Parameters
HOSTS=hostA hostB hostC
LMSTAT_PATH=/etc/flexlm/bin
LMSTAT_INTERVAL=30
End Parameters
```

Configure projects

The Projects section in the `lsf.licensescheduler` file lists the names of all license projects.

If you want to specify a distribution policy for a feature, you must define the associated license projects in the Projects section. For example, define license projects Lp1 and Lp2 in the Projects section:

```
Begin Projects
PROJECTS
Lp1
Lp2
End Projects
```

Use the defined projects to specify a distribution policy in the Feature section:

```
Begin Feature
NAME = AppY
DISTRIBUTION = LanServer1(Lp1 10/10 Lp2 5/5)
End Feature
```

Projects without priority

The following Projects section in `lsf.licensescheduler` defines three license project names without an associated priority:

```
Begin Projects PROJECTS Lp1 Lp2 Lp3 End Projects
```

Projects with priority

If you want to use the project priority feature with license ownership policies, add a PRIORITY column and assign a priority to each project (a higher value represents a higher priority). This overrides the default behavior; instead of preempting in order the projects are listed under PROJECTS based on the accumulative usage of each project, the projects are preempted according to the specified priority from lowest to highest.

```
Begin Projects
PROJECTS       PRIORITY
Lp1             3
Lp2             1
Lp3             2
default         0
End Projects
```

When 2 projects have the same priority number configured, the first listed project has higher priority, like LSF queues.

Priority of default project

If not explicitly configured, the default project has the priority of 0. You can override this value by explicitly configuring the default project in Projects section with the chosen priority value.

Configure clusters

If you run LSF License Scheduler in a WAN configuration, you must configure the Clusters section of the `lsf.licensescheduler` file. You do not need to configure the Clusters section when you run License Scheduler in a single cluster.

After installing License Scheduler and starting `bld` on all your candidate hosts in each cluster, configure the Clusters section in the cluster that contains the WAN license server. The following example describes a WAN with two clusters named cluster1 and cluster2.

Example

```
Begin Clusters
CLUSTERS
cluster1
cluster2
End Clusters
```

Configure service domains

You must configure a service domain for LSF License Scheduler. The service domain is a group of one or more FLEXnet license server hosts that serve licenses to LSF jobs. The service domain is used when you define a policy for sharing software licenses among your projects.

You can configure multiple service domains for LSF license Scheduler. The `lsf.licensescheduler` file comes with example configurations.

Keep the following requirements in mind:

- If a FLEXnet license server host is not part of an LSF License Scheduler service domain, its licenses are not managed by License Scheduler (the license distribution policies you configure in LSF do not apply to these licenses and usage of these licenses does not influence LSF scheduling decisions).
- License Scheduler assumes that any license in the service domain is available to any user who can receive a token from License Scheduler. Therefore, every user associated with a project specified in the distribution policy must meet the following requirements:
- The user is able to make a network connection to every FLEXnet license server host in the service domain.
- The user environment is configured with permissions to check out the license from every FLEXnet license server host in the service domain.
- To use LSF License Scheduler tokens, a job submission must specify the `-Lp` (license project) option. The project must be defined for the requested feature in `lsf.licensescheduler`.

The ServiceDomain section in `lsf.licensescheduler` defines the LSF License Scheduler service domain.

In the simplest case, the service domain consists of one FLEXnet license server host. In this example, the service domain is named DesignCenterA, and consists of one FLEXnet license server host, hostA. FLEXnet uses port number 1700 on this host.

Example

```
Begin ServiceDomain NAME=DesignCenterA LIC_SERVERS=((1700@hostA)) End
ServiceDomain
```

NAME

Choose a name for the service domain. You will use this name when you configure the distribution policies.

LIC_SERVERS

Specify all the FLEXnet license server hosts that make up the service domain. Specify the host name of each host and its FLEXnet port number.

Use one set of parentheses to enclose the entire list, and one more set around each host:

```
LIC_SERVERS=((1700@hostA)(1700@hostB))
```

If you have only one host, use a double set of parentheses:

```
LIC_SERVERS=((1700@hostA))
```

If you have redundant FLEXnet license server hosts, the parentheses are used to group the three hosts that share the same `license.dat` file:

```
LIC_SERVERS=((1700@hostD 1700@hostE 1700@hostF))
```

If FLEXnet uses a port from the default range, you can specify the host name only:

LIC_SERVERS=((@hostA))

LIC_COLLECTOR

Specify a name for the license collector daemon. You can use any name, but you must specify the same name when you start the license collector daemon.

Specify one LIC_COLLECTOR for each service domain:

```
Begin ServiceDomain NAME=DesignCenterA LIC_SERVERS=((1700@hostA 1700@hostB
1700@hostC)) LIC_COLLECTOR=CenterA End ServiceDomain
```

```
Begin ServiceDomain NAME=DesignCenterB LIC_SERVERS=((1888@hostD)
(1888@hostE)) LIC_COLLECTOR=CenterB End ServiceDomain
```

Configure license features

Define a feature section to create a distribution policy for each licensed feature managed by License Scheduler:

FLEX_NAME

Optional. Defines the feature name—the name used by FLEXnet to identify the type of license. You only need to specify this parameter if the License Scheduler token name is not identical to the FLEXnet feature name.

FLEX_NAME allows the NAME parameter to be an alias of the FLEXnet feature name.

default

A reserved keyword that represents the default License Scheduler project if the job submission does not specify a project (`bsub -Lp`) (does not apply if LSF_LIC_SCHED_STRICT_PROJECT_NAME=y in `lsf.conf` and you have not configured a default project for the required feature).

Example

```
Begin Feature
```

```
FLEX_NAME=201-AppZ
NAME=AppZ201
DISTRIBUTION=LanServer1(Lp1 1 Lp2 1 default 1)
End Feature
```

Configure hierarchical project groups (optional)

For detailed instructions on configuring a project group hierarchy, see Hierarchical Fairshare among Project Groups.

Start LSF License Scheduler

You can configure LSF to start the License Scheduler daemon (`bld`) on the License Scheduler host as well as on candidate License Scheduler hosts that can take over license distribution in the case of a network failure. The LSF LIM daemon starts `bld` automatically.

1. Log on as the primary LSF administrator.
2. Set your LSF environment:
   - For `csh` or `tcsh`:

     `% source LSF_TOP/conf/cshrc.lsf`
   - For `sh`, `ksh`, or `bash`:

     `$ . LSF_TOP/conf/profile.lsf`
3. In `LSF_CONFDIR/lsf.conf`, specify a space-separated list of hosts for the `LSF_LIC_SCHED_HOSTS` parameters:

   ```
   LSF_LIC_SCHED_HOSTS="hostname_1 hostname_2 ... hostname_n"
   ```

   Where:

   hostname_1, *hostname_2*, ..., *hostname_n* are hosts on which the LSF LIM daemon starts the LSF License Scheduler daemon. The order of the host names is ignored.

   Note:

   Set the `LSF_LIC_SCHED_HOSTS` parameter to the same list of candidate hosts you used in the `lsf.licensescheduler HOSTS` parameter. The `LSF_LIC_SCHED_HOSTS` parameter is not used in any other function.
4. Run `lsadmin reconfig` to reconfigure the LIM.
5. Use `ps -ef` to make sure that `bld` is running on the candidate hosts.
6. Run `badmin mbdrestart` to restart `mbatchd`.

Start LSF License Scheduler standalone

To start LSF License Scheduler standalone, run `blstartup`.

1. Verify your configuration.
   a) Use `blinfo -D` to see the FLEXnet license server hosts in the service domains:

   blinfo -D

   ```
   SERVICE_DOMAIN LIC_SERVERS

   DesignCenterA (1700@hostA)
   ```

   b) Use `blstat` to make sure License Scheduler is collecting data from FLEXnet, and to see license usage and distribution information. Check the TOTAL line for non zero values:

   blstat

   ```
   FEATURE: vcsruntime

   SERVICE_DOMAIN: DesignCenterA

   TOTAL_INUSE: 2 TOTAL_RESERVE: 1 TOTAL_FREE: 13 OTHERS:1
   ```

```
PROJECT SHARE INUSE RESERVE FREE DEMAND
projectA 60.0 % 0 1 9 n
projectB 40.0 % 2 0 4 y
```

c) Use `bhosts -s` to verify that LSF and License Scheduler are communicating. The `bhosts -s` output shows the features configured in License Scheduler as LSF resources:

bhosts -s

```
RESOURCE TOTAL RESERVED LOCATION
p1_f1 4.0 0.0 hostA
hostB
hostF
p1_f2 4.0 0.0 hostA
hostB
hostG
```

Submit jobs

Run jobs using LSF

When you submit an LSF job, you must:

- Reserve the license using the resource requirement usage section (`bsub -R "rusage..."` option)

Tip:

You cannot successfully reserve a license using `bsub -R "select"`.

- Specify the license token name (same as specifying a shared resource)
- Specify a license project name with the `bsub -Lp` option

Projects

The project must be a valid license project configured in the `lsf.licensescheduler` file. If your usage section specifies a feature that you configured in the `lsf.licensescheduler` file, and you do not submit your job to a license project, the job is submitted to the `default` license project unless LSF_LIC_SCHED_STRICT_PROJECT_NAME=y in `lsf.conf` and you have not configured a default project for the required feature.

Tip:

Use the `blstat` command to view information about the `default` license project.

Example 1:

```
% bsub -R "rusage[AppB=1]" -Lp Lp1 myjob
```

This submits a job called `myjob` to license project `Lp1` and requests one `AppB` license.

Example 2:

```
% bsub -R "rusage[AppB=1]" myjob
```

This submits a job called `myjob` to the `default` license project (unless LSF_LIC_SCHED_STRICT_PROJECT_NAME=y in `lsf.conf` and you have not configured a default project for the required feature) and requests one `AppB` license.

Update resource requirements

If your queue or job starter scripts request a license that is managed by an LSF ELIM, you need to update the job submission scripts to request that license using the license token name.

Optimize expensive licenses

You can optimize the usage of expensive licenses in two ways:

- Configure the resource requirements order string for sorting selection at job submission
- Configure a host list at the queue level

Tip:

If the resource requirement string is configured both at the queue level and at job submission, the queue level configuration is ignored.

Configuration of order string

bsub -R "select[type==any] order[*resource_name*] rusage[*token_name*=1]" -Lp *license_project_name job_name*

For example:

bsub -R "select[type==any] order[cpuf] rusage[feature1=1]" -Lp Lp1 my_jobname

You can sort by other factors such as `swp`, `ut`, and `r1m` in your order string.

Configuration at the queue level

Configure the resource requirement string and host list in `lsb.queues`:

```
Begin Queue
QUEUE_NAME = fastqueue
PRIORITY = 10
HOSTS = hostA+1 hostB+2 hostC hostD+1
RES_REQ = select[type==any] rusage[feature1=1]
End Queue
```

Configure your host list based on desirability. This depends on your own knowledge of the capability of each host. In this example, you decide that:

- `hostB` is the fastest
- `hostA` and `hostD` are the second fastest
- `hostC` is the slowest

After configuring `lsb.queues`, use `badmin reconfig`, then submit jobs:

```
% bsub -q fastqueue -Lp Lp1 my_jobname
```

CHAPTER

3

Controlling License Distribution

Understanding distribution policies

The most important part of LSF License Scheduler is license distribution. The license distribution policy determines how licenses are shared among projects. Whenever there is competition, the configured share assignment determines the portion of licenses each project is entitled to.

You can use different methods of distribution of License Scheduler policies:

- Fairshare helps you ensure that all projects receive the share of license tokens they are entitled to.
- Ownership lets high priority projects preempt licenses on demand, but share them when not required.
- Non-shared licenses are not shared with other projects. When not in use by the project, they are always only available to that project.

Share assignments

Whenever there are licenses to spare, license projects can get as many tokens as they need. The share assignment is defined in the policy, but is ignored.

Whenever a project is using all its licenses and needs more, License Scheduler attempts to assign additional licenses to it. This is possible if another project is not using all its assigned licenses.

The total number of licenses managed by License Scheduler depends on the following:

- The number of active license servers in the service domain
- The number of licenses checked out by non-License Scheduler users
- The number of new licenses that are added
- The number of licenses that expired

The License Scheduler distribution policy entitles each license project to a minimum portion of the available licenses.

The share assignment in the License Scheduler distribution policy determines what portion of the total licenses is assigned to each project.

For example, create three License Scheduler projects, and share the licenses equally. If one project does not need a license, another project can use it.

```
(projectA 1 projectB 1 projectC 1)
```

License Scheduler distributes the licenses evenly to each project. If you have 264 licenses, each project gets 88 licenses.

Not all license projects in a cluster have a full workload at all times. Free licenses can be shared across projects, so that idle licenses for one project are available to other projects.

Example of share assignments with demand

The following illustrates how licenses are shared according to the share assignment ratio when there is demand.

The AppZ feature has a total of 120 tokens.

AppZ has the following configuration:

```
Begin Feature

NAME = AppZ

DISTRIBUTION = LanServer(A 1 B 1)

End Feature
```

Initially, before demand is introduced, the projects have the following token distribution:

Project	Used, Reserved, and Free	Demand
A	70	0
B	0	0

Because this feature controls 120 tokens and only 70 are used, 50 tokens are free.

If both projects A and B demand 100 tokens:

Project	Used, Reserved, and Free	Demand
A	70	100
B	0	100

The total number of shared tokens is calculated as the number of shared tokens consumed by P1 + shared tokens consumed by P2 + free tokens.

```
70 + 0 + 50 = 120 shared tokens
```

Both A and B have a share assignment of 1 configured, which means they each deserve the same number of tokens. In this case, they each deserve 60 (half of 120) tokens.

Because project A is already using 70 shared tokens, it is allocated no additional free tokens.

Because project B is using 0 shared tokens and it deserve 60 shared tokens, all 50 free tokens are allocated to project B.

After token distribution, the token usage and demand situation is as follows:

Project	Used, Reserved, and Free	Demand
A	70	100
B	50	50

License ownership

License ownership gives license projects the right to use their licenses on demand, while still allowing License Scheduler to distribute the licenses to other projects when the owner is not using them.

Whenever there is competition, the configured share assignment determines the portion of licenses each license project is entitled to. Whenever there are licenses to spare, the license distribution policy still defines which projects can receive tokens, but the share assignment is ignored (the specified projects can receive as many tokens as they need).

How license ownership and preemption work

In some cases, licenses can be shared among license projects, but one project has priority. This project is the owner of the licenses. When there is competition between the owner and other projects, the licenses should become available to the owner immediately.

License Scheduler only distributes tokens for free licenses. By default, if all licenses are in use, a license has to be released before License Scheduler can provide a license token to another license project. Therefore, a project that is entitled to a license may have to wait for another project's running job to finish. There is no way to predict how long this takes.

If license ownership is configured, the owner should never have to wait to use the owned licenses. If a license project requires a license and is entitled to it by right of ownership, but there are no licenses free, License Scheduler preempts a running job in order to take a license away from another project.

License projects that own some licenses can participate in license sharing with projects that do not own any. A service domain can include both owned and unowned licenses.

When a license project's share assignment is more than the number of licenses it owns, and there is competition, the project is entitled to use its configured ownership entitlement. Preemption can occur only while the project is not yet using the specified number of owned licenses and no free licenses are available. Once the project is using the number of licenses it owns, License Scheduler waits for licenses to become free and then distributes additional tokens until the project is using its fair share.

The jobs that are preempted by LSF are automatically resumed by LSF as licenses become available.

You can enable LSF to release the job slot of a suspended job when License Scheduler preempts the license from the job.

Note:

For License Scheduler to give a license token to another license project, the applications must be able to release their licenses upon job suspension.

Preemption when JOB_CONTROLS are defined

If the LSF administrator has defined JOB_CONTROLS in `lsb.queues` so that preemption uses the signal SIGTSTP, they must also define LIC_SCHED_PREEMPT_STOP=Y in `lsf.conf` for License Scheduler preemption to work.

How LSF License Scheduler selects projects for preemption

When a project needs to preempt a license token currently in use by other projects, License Scheduler determines which license token to preempt and which project gets the preempted token. It is possible for a project to use more licenses than it owns—this is an *overfed project*. It is also possible for a project to use fewer licenses than it owns—this is an *underfed project*. License Scheduler typically preempts license tokens from the overfed projects and gives these preempted tokens to the underfed projects.

License Scheduler determines the order in which overfed and underfed projects are selected for preemption by looking at the accumulative `inuse` of each project—the time in which the project has been overfed or underfed.

You can manually override this behavior by adding a PRIORITY column to the `Projects` section of `lsf.licensescheduler` and assigning a priority to each project. This enables License Scheduler to look at the project priority along with the hierarchical fairshare. All the projects in the hierarchy are assigned a priority for preemption ordering.

If no hierarchical project groups are defined, the default project configuration is flat. The priority of a project has nothing to do with its position in the hierarchy. Project priority values can be compared between all leaf nodes.

You can also minimize the overall number of preempted jobs by enabling job list optimization. When you set the parameter ENABLE_MINJOB_PREEMPTION=Y in the `Feature` section of `lsf.licensescheduler`, License Scheduler preempts the minimum number of jobs needed to obtain the required licenses. For example, for a job that requires 10 licenses, License Scheduler preempts one job that uses 10 or more licenses rather than 10 jobs that each use one license.

Overfed projects by priority

If there is more than one overfed project, License Scheduler preempts tokens from these projects in the following order:

- If there are fewer license tokens than the project owns, License Scheduler preempts license tokens from the lowest priority project.
- If there are more licenses than the project owns, License Scheduler preempts license tokens from projects according to the order projects are listed in the Projects section.

Underfed projects by priority

If there is more than one underfed project, License Scheduler assigns the preempted tokens to these projects in the following order:

- If there are fewer licenses than the project owns, License Scheduler gives the preempted license tokens to the highest priority project
- If there are more licenses than the project owns, License Scheduler gives the preempted license tokens to projects according to the order projects are listed in the Projects section.

How LSF preemption and License Scheduler preemption coexist

Jobs belonging to a license project that has ownership in License Scheduler can trigger preemption even when no more slots are available in LSF. Configured together with LSF_LIC_SCHED_PREEMPT_SLOT_RELEASE, license job preemption works together with LSF slot-based preemption.

Example

Project `proj1` has ownership of 3 of the license `AppX`.

MXJ = 5, and LSF_LIC_SCHED_PREEMPT_SLOT_RELEASE=Y is configured in `lsf.conf`.

5 jobs are submitted and started using `AppX`, in `proj2`. Then 2 jobs are submitted to `proj1`, and pend waiting for a `AppX` license token. Although the slots are full, the request is sent to License Scheduler, which recognizes the ownership and preempts 2 jobs in `proj2`. The jobs are suspended, both their licenses and slots are released, and the 2 jobs in `proj1` can run.

Maximum preemption limits

Both LSF jobs and taskman jobs using licenses managed by License Scheduler can be preempted. To ensure lower priority jobs are not preempted too many times, maximum preemption time limits can be enabled with LS_ENABLE_MAX_PREEMPT.

For LSF jobs the parameter MAX_JOB_PREEMPT sets the maximum number of times a job can be preempted. MAX_JOB_PREEMPT can be defined in `lsb.params`, `lsb.queues`, or `lsb.applications`, with the application setting overriding the queue setting and the queue setting overriding the cluster-wide `lsb.params` definition.

License Scheduler taskman job preemption limits are controlled by the parameter LS_MAX_TASKMAN_PREEMPT in `lsf.licensescheduler`.

How LSF License Scheduler calculates shares and owned licenses

To configure license ownership, edit the DISTRIBUTION parameter in the `Feature` section of `lsf.licensescheduler`. Each license project's share assignment is defined by specifying the project name followed by its share:

license_project_name number_shares

To indicate ownership, the share assignment is followed by a slash and the number of licenses owned by that license project. The two numbers cannot be compared because they are different units—the ownership figure is always a fixed number of licenses, while share assignment can represent a ratio or percentage of the total instead of an actual number of licenses.

license_project_name number_shares/number_licenses_owned F

Example

```
DISTRIBUTION=LanServer1(Lp1 1 Lp2 2/6)
```

This example allows `Lp1` to use one third of the available licenses and `Lp2` to use two thirds of the licenses. However, `Lp2` is always entitled to six licenses, and can preempt other license projects to obtain the licenses immediately if they are needed. If the projects are competing for a total of 12 licenses, `Lp2` is entitled to eight (six on demand, and two more as soon as they are free). If the projects are competing for only six licenses in total, `Lp2` is entitled to all of them, and `Lp1` can only use licenses when `Lp2` does not need them.

Total licenses

The total number of licenses managed by License Scheduler depends on whether all license servers in the domain are active, and whether licenses have been checked out by non-LSF users. The number of licenses available to LSF changes when licenses are added to any license server in the domain and when licenses expire.

Share assignment

The share assignment determines what fraction of the total licenses is assigned to each license project.

The formula for converting a number of shares to a number of licenses is:

```
(shares assigned to a project) x (total number of licenses)
-----------------------------------------------------------        (sum of all shares assigned to
all projects)
```

The number of shares assigned to a license project is only meaningful when you compare it to the number assigned to other projects, or to the total number of shares.

Example

- `(Lp1 1 Lp2 1)`

In this example, you configure a 1:1 ratio. No matter how many licenses you have, LSF assigns half to each license project. If you have 264 licenses, LSF assigns 132 licenses to each.

- `(Lp1 1 Lp2 1 Lp3 1)`

In this example, you add another license project, and assign it one share. Now LSF evenly distributes one third of the available licenses to each project. If you have 264 licenses, each project receives 88 licenses.

Percentage of share

If you set `LS_FEATURE_PERCENTAGE=Y` in `lsf.licensescheduler`, you configure license ownership in percentages instead of absolute numbers. When not combined with hierarchical projects, affects DISTRIBUTED and NON_SHARED_DISTRIBUTION values only. When using hierarchical projects, percentage is applied to OWNERSHIP, LIMITS, and NON_SHARED values.

Example 1

```
Begin Feature

LS_FEATURE_PERCENTAGE = Y

DISTRIBUTION = LanServer (p1 1 p2 1 p3 1/20)

...

End Feature
```

The service domain LanServer shares licenses equally among three License Scheduler projects. P3 is always entitled to 20% of the total licenses, and can preempt another project to get the licenses immediately if they are needed.

Example 2

Hierarchical project groups:

```
Begin ProjectGroup
GROUP       SHARES    OWNERSHIP    LIMITS    NON_SHARED
(R (A p4))  (1  1)      ()          ()         ()
(A (B p3))  (1  1)      (- 10)      (- 20)     ()
(B (p1 p2)) (1  1)      (30 -)      ()         (- 5)
End ProjectGroup
```

Project p1 owns 30% of the total licenses, and project p3 owns 10% of total licenses. P3's LIMITS is 20% of total licenses, and p2's NON_SHARED is 5%.

Non-shared licenses

If you have a project that is so important that you must always guarantee the availability of a license token to the project, you can distribute non-shared licenses to a project. These licenses cannot be shared with other projects.

Some license agreements do not allow license sharing. These can be distributed to license projects as non-shared licenses.

Default projects

Default projects are projects that are not specified in job submission (for example, with `bsub -Lp`), but use license tokens that are managed by License Scheduler.

If you do not want the default project to get shares of license tokens, you do not need to define a default project in the distribution policy for a feature. If you do configure the default project in a policy, it receives its assigned share of the license tokens. All jobs requiring a license feature that is managed by License Scheduler, and are not submitted to a configured project for the feature, are treated as jobs submitted to the default project unless `LSF_LIC_SCHED_STRICT_PROJECT_NAME=y` in `lsf.conf` and you have not configured a default project in DISTRIBUTION parameter, in which case the job is rejected.

Configuring distribution policies

Configure all LSF License Scheduler distribution policies in the `lsf.licensescheduler` file.

Configure a separate Feature section for each license feature. For each license feature, specify the service domain and the distribution policy. Distribution is indicated by specifying the license project name and share assignment.

Checking configuration

To see the distribution policies configured by the License Scheduler administrator, run `blinfo` with no options.

Ownership and sharing

When you configure the ownership and sharing of the licenses for a feature in `lsf.licensescheduler`, specify the following for each project:

- The number of license tokens that are owned and can be shared with other projects
- The number of license tokens that are owned and not shared with other projects
- The number of shares of the license feature that are assigned to each project

Tip:

Shared licenses can be borrowed while they are not required by their owners. Non-shared licenses cannot be borrowed from their owners.

Configure shared licenses without ownership

In `lsf.licensescheduler`, configure the feature with two license projects.

In the following example, each project is entitled to 50% of the total number of license tokens.

```
Begin Feature
NAME=AppZ
DISTRIBUTION=LanServer1(Lp1 1 Lp2 1)
End Feature
```

Initially, one project may be using most of the tokens. Later the other project may require tokens. As tokens become available, License Scheduler distributes them according to the ratio of the shares.

Configure shared licenses with ownership

In `lsf.licensescheduler`, configure the feature with two license projects.

- Lp1 owns 5 licenses.
- Lp1 is entitled to three quarters of the total number of license tokens.
- Lp2 is entitled to one quarter of the total number of license tokens.

```
Begin Feature
NAME=AppZ
DISTRIBUTION=LanServer1(Lp1 3/5 Lp2 1)
End Feature
```

Lp1 is guaranteed 5 tokens and can preempt them from Lp2 if necessary. As tokens become available, License Scheduler distributes them according to their shares (3:1).

Configure non-shared licenses

In `lsf.licensescheduler`, configure the feature with two license projects.

- Lp1 owns five licenses:
- Three licenses that can be shared
- Two licenses that cannot be shared
- Lp1 is entitled to three quarters of the total number of license tokens.
- Lp2 is entitled to one quarter of the total number of license tokens.

Do this by adding a non-shared distribution parameter to the Feature section.

```
Begin Feature
NAME=AppZ
DISTRIBUTION=LanServer1(Lp1 3/5 Lp2 1)
NON_SHARED_DISTRIBUTION=LanServer1(Lp1 2)
End Feature
```

Lp1 can receive five tokens on demand.

- The two non-shared licenses it owns are always available to or being used by Lp1.
- If necessary, Lp1 can preempt the three shared licenses it owns from Lp2.

As tokens become available, License Scheduler distributes them by their shares. For example, if the total number of licenses is 60, and Lp1 is guaranteed five owned licenses (three shared and preemptable, two not shared), it is entitled to another 40 licenses according to its share (3:1).

Use `blinfo -a` to display NON_SHARED_DISTRIBUTION information:

Configure default projects

You can optionally configure default projects for a feature.

Default includes all projects that have not been defined in the PROJECTS section of `lsf.licensescheduler`. Jobs that belong to projects that are defined in `lsf.licensescheduler` do not get a share of the tokens when the project is not explicitly defined in the distribution.

Use blinfo to display license usage information about jobs that you submit without specifying a project. They can be viewed under the default project.

Default projects behavior

- You submit a job requiring a feature that is managed by License Scheduler, but you do not submit it to any projects configured in `lsf.licenseschduler` (and you have LSF_LIC_SCHED_STRICT_PROJECT_NAME=n in `lsf.conf`).

For example, you configure your Feature section as follows:

```
Begin Feature
NAME=AppZ
DISTRIBUTION=LanServer1(Lp1 2 Lp2 1)
End Feature
```

If you submit a job that requires a license for AppZ without specifying a project, your job will pend, even if projects Lp1 and Lp2 are not using or waiting for licenses.

- You configure a default project for a feature in a policy that also has other projects defined. For example:

```
Begin Feature
NAME=AppZ
DISTRIBUTION=LanServer1(Lp1 2 Lp2 1 default 1)
End Feature
```

If you submit a job that requires a license for AppZ without specifying a project, your job runs if there are free licenses, which are distributed according to the shares specified in the policy. In this example, jobs that do not specify a project receive a quarter of the license tokens.

- You can only configure a default project for a feature that has no other projects defined in the policy. For example:

```
Begin Feature
NAME=AppZ
DISTRIBUTION=LanServer1(default 1)
End Feature
```

If you submit a job that requires a license for AppZ without specifying a project, your job runs if there are free licenses. The only advantage of using this configuration is the ability to view usage information.

Configure reserved license preemption

In `lsf.licensescheduler`, you can configure a feature so that its license is preemptable when reserved or already in use by other projects. Do this by adding PREEMPT_RESERVE=Y to the Feature section.

For example,

```
Begin Feature
NAME=AppZ
DISTRIBUTION=LanServer1(Lp1 1/5 Lp2 1)
PREEMPT_RESERVE=Y
End Feature
```

If no licenses are available and project Lp1 is using fewer than 5 licenses, project Lp1 preempts a reserved license from project Lp2 and checks out the license from the license server.

Tip:

When PREEMPT_RESERVE is set, the project preempts either a reserved license or a license that is in use by another project. By default, reserved licenses are not preemptable.

Preempting reserved licenses

With License Scheduler, licenses are reserved before it is actually necessary to check them out from the license server.

When you submit a job that requires a license feature at the conclusion of its run, it could take a long time before the license actually is checked out by the application requiring it. While the job runs, the license is reserved even though it is not in use. The job has reserved this license.

At the same time, you may want to submit a shorter job that requires the same license. You can preempt the reserved license if all of the following criteria are true:

- No licenses are available for the job
- You submit the job to a project that owns a minimum number of licenses

- The project is using fewer licenses than the number of licenses it owns
- You have configured License Scheduler to allow preemption of reserved licenses

Aliasing license token names (FLEX_NAME)

Normally, license token names should be the same as the FLEXnet feature names, as they represent the same license. However, LSF does not support names that start with a number, or names containing a dash or hyphen character (-), which may be used in the FLEXnet feature name.

For these feature names, you must set both NAME and FLEX_NAME in the Features section of `lsf.licensescheduler`. FLEX_NAME is the actual FLEXnet feature name, and NAME is an arbitrary license token name you choose.

Example

```
Begin Feature
FLEX_NAME=201-AppZ
NAME=AppZ201
DISTRIBUTION=LanServer1(Lp1 1 Lp2 1)
End Feature
```

Viewing available licenses

License Server collects license feature information from physical servers and merges this data together into a service domain. After the merging, the individual license server information is retained, and you can view this information together with the physical server information.

Use the `blstat` command to display dynamic system status, and to display information about the actual license distribution and use for each project.

The licenses in use have been checked out from FLEXnet by your projects. Free licenses and licences reserved by a project have not yet been checked out from FLEXnet.

The total number of licenses could change as licenses expire, or are added. As non-LSF users check out licenses, the OTHERS count in `blstat` should increase and the TOTAL_FREE count will decrease. The number of licenses for each project changes whenever LSF redistributes license tokens among competing projects.

Viewing license server and license feature information passed to jobs

To view the license server associated with the license features, use `blstat -S`. This displays the license servers used by each service domain allocated to the license features.

The license server information for each license feature is stored in the LS_LICENSE_SERVER_*feature* environment variable.

blstat -S

```
FEATURE: feature1
SERVICE_DOMAIN: domain1
SERVERS     INUSE  FREE
 server1      1      0
 server2      0      1
 TOTAL        1      1
SERVICE_DOMAIN: domain2
SERVERS     INUSE  FREE
 server3      1      0
 TOTAL        1      0
```

This shows that the license feature feature1 is assigned to server1 and server2 in the domain1 service domain and server3 in the domain2 service domain. A job uses the feature1 license feature when the job is submitted with "rusage [feature1=1]" as the rusage string.

Viewing workload distribution information

Use `blstat -s` to display license usage. Workload distributions are defined by WORKLOAD_DISTRIBUTION in `lsf.licensescheduler`. If there are any distribution policy violations, `blstat` marks these with an asterisk (*) at the beginning of the line.

blstat -s

```
FEATURE: p1_f2
SERVICE_DOMAIN: app_1 TOTAL_LICENSE: 10
LSF_USE LSF_DESERVE LSF_FREE   NON_LSF_USE NON_LSF_DESERVE  NON_LSF_FREE
     0       10         10          0           0                0
FEATURE: p1_f1
SERVICE_DOMAIN: app_1 TOTAL_LICENSE: 5
LSF_USE LSF_DESERVE LSF_FREE   NON_LSF_USE NON_LSF_DESERVE  NON_LSF_FREE
     0       5          5           0           0                0
```

blinfo -a

Use `blinfo -a` to display WORKLOAD_DISTRIBUTION information:

```
% blinfo -a
FEATURE       SERVICE_DOMAIN  TOTAL  DISTRIBUTION
g1            LS              0      [p1, 50.0%] [p2, 50.0%]
                                     WORKLOAD_DISTRIBUTION
                                     [LSF 66.7%, NON_LSF 33.3%]
```

Viewing workload distribution information

View sorted license feature information (blinfo -o and blstat -o)

Use the `-o` option of blinfo and `blstat` to sort license feature information alphabetically, by total licenses, or by available licenses.

Use `-o blstat` alone or with options `-Lp`, -t, `-D`, `-G`, `-s`, `-S`. The values of "total licenses" and "licenses available" are calculated differently when `blstat -o` is used with different options:

- Options `-Lp`, -t, `-D`, `-G`: Total licenses means the sum of licenses that are allocated to LSF workload from all the service domains configured to supply licenses to the feature. Licenses borrowed by non-LSF workload are subtracted from this sum.
- Options `-s`, `-S`: All the licenses (supplied by the license vendor daemon) from all the service domains configured to supply licenses to that feature.

Use `-o` with `blinfo` alone or with options `-a` and `-t`. You can only sort alphabetically (`-o alpha`) or by total licenses (`-o total`), because `blinfo` does not display information about available licenses. The command `blinfo -o` is not supported in combination with `-Lp`, -p, `-D`, `-G`.

The value of total licenses is calculated using the number of licenses LSF workload deserves from all service domains that supply licenses to the feature, regardless of whether non-LSF workload has borrowed licenses from LSF workload.

Examples

The blstat -o alpha

```
FEATURE: p1_5
 SERVICE_DOMAIN: LanServer
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 6    OTHERS: 0
       PROJECT          SHARE     INUSE     RESERVE   FREE   DEMAND
         B             50.0 %       0          0       3       n
        p1             25.0 %       0          0       1       n
        p2             25.0 %       0          0       2       n
FEATURE: p1_12
 SERVICE_DOMAIN: LanServer
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 7    OTHERS: 0
  PROJECT                 SHARE     INUSE     RESERVE   FREE       DEMAND
  default              33.3 %     0          0        3          n
     p1                33.3 %     0          0        2          n
     p2                33.3 %     0          0        2          n
 SERVICE_DOMAIN: LanServer1
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 10   OTHERS: 0
  PROJECT                 SHARE     INUSE     RESERVE   FREE       DEMAND
    p1                 50.0 %     0         0         5         n
    p2                 50.0 %     0         0         5         n
FEATURE: myjob10
 SERVICE_DOMAIN: LanServer
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 9    OTHERS: 0
  PROJECT                 SHARE     INUSE     RESERVE   FREE       DEMAND
  p4                    50.0 %       0           0      5          n
  p3                    25.0 %       0           0      2          n
  p1                    12.5 %       0           0      1          n
  p2                    12.5 %       0           0      1          n
```

Sorting with -o total is based on the total number of licenses. In the following example, the total licenses for p1_12, myjob10, p1_5 are 17(7+10), 9, and 6. p1_12 which is the feature with largest number of "total licenses" comes first.

The the total licenses value for p1_5 in this example is 6 not 10 (6+4), because the '4' in the OTHERS field shows that non-LSF workload has borrowed 4 licenses from the LSF allocation.

blstat -o total

```
FEATURE: p1_12
 SERVICE_DOMAIN: LanServer
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 7    OTHERS: 0
  PROJECT                  SHARE     INUSE     RESERVE    FREE       DEMAND
  default                  33.3 %      0          0        3           n
  p1                       33.3 %      0          0        2           n
  p2                       33.3 %      0          0        2           n
 SERVICE_DOMAIN: LanServer1
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 10   OTHERS: 0
  PROJECT                  SHARE     INUSE     RESERVE    FREE       DEMAND
  p1                       50.0 %      0          0        5           n
  p2                       50.0 %      0          0        5           n
FEATURE: myjob10
 SERVICE_DOMAIN: LanServer
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 9    OTHERS: 0
  PROJECT                  SHARE     INUSE     RESERVE    FREE       DEMAND
  p4                       50.0 %      0          0        5           n
  p3                      25.0 %       0          0        2           n
  p1                      12.5 %       0          0        1           n
  p2                      12.5 %       0          0        1           n
FEATURE: p1_5
  SERVICE_DOMAIN: LanServer
      TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 6    OTHERS: 4
    PROJECT                 SHARE     INUSE     RESERVE    FREE       DEMAND
      B                     50.0 %      0          0        3           n
      p1                    25.0 %      0          0        1           n
      p2                    25.0 %      0          0        2           n
```

The following example specifies service domains using the -D option. The values of total licenses for myjob10, p1_12, p1_5 are 9, 7, 6, so myjob10 is listed first.

blstat -o total -D LanServer

```
FEATURE: myjob10
 SERVICE_DOMAIN: LanServer
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 9    OTHERS: 0
  PROJECT                  SHARE     INUSE     RESERVE    FREE       DEMAND
  p4                       50.0 %    0         0          5          n
  p3                       25.0 %    0         0          2          n
  p1                       12.5 %    0         0          1          n
  p2                       12.5 %    0         0          1          n
FEATURE: p1_12
 SERVICE_DOMAIN: LanServer
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 7    OTHERS: 0
  PROJECT                  SHARE     INUSE     RESERVE    FREE       DEMAND
  default                  33.3 %    0         0          3          n
  p1                       33.3 %    0         0          2          n
  p2                       33.3 %    0         0          2          n
    FEATURE: p1_5
      SERVICE_DOMAIN: LanServer
      TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 6    OTHERS: 4
      PROJECT              SHARE     INUSE     RESERVE    FREE       DEMAND
      B                    50.0 %    0         0          3          n
      p1                   25.0 %    0         0          1          n
      p2                   25.0 %    0         0          2          n
```

In the following example, the number of licenses available for p1_5, p1_12 and myjob10 are 6, 5(0+5),4. p1_5 is the feature with largest number of available licenses, so it is listed first.

blstat -o avail

```
FEATURE: p1_5
 SERVICE_DOMAIN: LanServer
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 6    OTHERS: 0
  PROJECT                  SHARE     INUSE     RESERVE   FREE      DEMAND
  B                        50.0 %       0         0        3          n
  p1                       25.0 %       0         0        1          n
  p2                       25.0 %       0         0        2          n
FEATURE: p1_12
 SERVICE_DOMAIN: LanServer
 TOTAL_INUSE: 7    TOTAL_RESERVE: 0    TOTAL_FREE: 0    OTHERS: 0
  PROJECT                  SHARE     INUSE     RESERVE   FREE      DEMAND
  default                  33.3 %       0         0        0          n
  p1                       33.3 %       7         0        0          n
  p2                       33.3 %       0         0        0          n
 SERVICE_DOMAIN: LanServer1
 TOTAL_INUSE: 5    TOTAL_RESERVE: 0    TOTAL_FREE: 5    OTHERS: 0
  PROJECT                  SHARE     INUSE     RESERVE   FREE      DEMAND
  p1                       50.0 %       5         0        0          n
  p2                       50.0 %       0         0        5          n
FEATURE: myjob10
 SERVICE_DOMAIN: LanServer
 TOTAL_INUSE: 5    TOTAL_RESERVE: 0    TOTAL_FREE: 4    OTHERS: 0
  PROJECT                  SHARE     INUSE     RESERVE   FREE      DEMAND
  p4                       50.0 %       0         0        4          n
  p3                       25.0 %       0         0        0          n
  p1                       12.5 %       5         0        0          n
  p2                       12.5 %       0         0        0          n
```

blinfo shows the following feature configuration and workload distribution:

blinfo

```
FEATURE    SERVICE_DOMAIN  TOTAL    DISTRIBUTION
p1_5       LanServer         6      [B, 50.0%] [p1, 25.0%] [p2, 25.0% / 1]
p1_12      LanServer         7      [default, 33.3%] [p1, 33.3%] [p2, 33.3%]
p1_12      LanServer1        10     [p1, 50.0% / 1] [p2, 50.0% / 6]
myjob10    LanServer         9      [B, 50.0%] [p1, 25.0%] [p2, 25.0% / 1]
```

In the following example, the value for the TOTAL column is always the number of licenses the feature should get from the service domain, no matter if some of the licenses have been borrowed by non-LSF workload.

blinfo -o "total"

```
FEATURE     SERVICE_DOMAIN   TOTAL   DISTRIBUTION
p1_12       LanServer          7     [default, 33.3%] [p1, 33.3%] [p2, 33.3%]
p1_12       LanServer1        10     [p1, 50.0% / 1] [p2, 50.0% / 6]
myjob10     LanServer          9     [B, 50.0%] [p1, 25.0%] [p2, 25.0% / 1]
p1_5        LanServer          6     [B, 50.0%] [p1, 25.0%] [p2, 25.0% / 1]
```

The `blinfo` options '-A' and '-C' do not supply information about "total licenses" and "licenses available", so only '-o alpha' is supported with `blinfo` -A and `-C. -o "total"` is ignored with `'-A'` and `'-C'` and the output is the same as using `-o "alpha"`.

Configure feature groups

Feature groups view, list, and control groups of features instead of each individual feature.

In `lsf.licensescheduler`, configure a `FeatureGroup` section, listing the license features associated with that license. For example:

```
Begin FeatureGroup
NAME = Synposys
FEATURE_LIST = ASTRO VCS_Runtime_Net Hsim Hspice
End FeatureGroup

Begin FeatureGroup
NAME = Cadence
FEATURE_LIST = Encounter NCSim NCVerilog
End FeatureGroup
```

Note:

- Each feature group must have a unique name.
- The feature names in FEATURE_LIST must already be defined in `Feature` sections.
- Feature names cannot be repeated in the FEATURE_LIST of one feature group.
- The FEATURE_LIST cannot be empty.
- Different feature groups can have the same features in their FEATURE_LIST.

View license feature group information (blinfo -g and blstat -g)

When FEATURE_LIST is configured for a group of license features in `lsf.licensescheduler`, you can run `blinfo -g` and `blstat -g` to see information about the features configured for the specified feature groups.

When you specify feature names with `-t`, features in the feature list defined by `-t` and features in feature list of feature groups are both displayed.

Feature groups listed with `-g` but not defined in `lsf.licensescheduler` are ignored.

The command `blstat -g` can be used alone or with options `-Lp`, `-t`, `-D` ,`-G'`, `-s`. The option `-g` is not supported together with `-S`, since `blstat -S` displays information for all the features.

The command `blinfo '-g'` can be used alone or with options `-a`, `-t`, `-C`, and `-A`.

Examples:

For the following feature group configurations in `lsf.licensescheduler`:

```
Begin Feature
NAME = feature1
DISTRIBUTION = LanServer(default 1 p1 1 p2 1/10)
End Feature
```

```
Begin Feature
NAME = feature2
DISTRIBUTION = LanServer(p1 2  p2 1/5)
End Feature

Begin Feature
NAME = feature3
DISTRIBUTION = LanServer(p1 1  p2 1/5)
End Feature

Begin Feature
NAME = feature4
DISTRIBUTION = LanServer(p1 1  p2 1)
End Feature

Begin FeatureGroup
NAME = myFeatureGroup1
FEATURE_LIST = feature1 feature2
End FeatureGroup

Begin FeatureGroup
NAME = myFeatureGroup2
FEATURE_LIST = feature2 feature3
End FeatureGroup
```

blstat-g "myFeatureGroup1"

Shows information for feature1 and feature2 in descending alphabetic order.

blstat-g "myFeatureGroup2"

Shows information for feature3 and feature3 in descending alphabetic order.

blstat -t "feature3 feature4" -g "myFeatureGroup1 myFeatureGroup2 FeatureGroup3"

Shows information for feature1, feature2, feature3, feature4 in descending alphabetic order. Information for each feature is displayed only once. Feature group "FeatureGroup3" is ignored because it is not defined in `lsf.licensescheduler`.

CHAPTER

4

Failover Provisioning

Failover provisioning for LANs

You can configure LSF License Scheduler for enhanced performance, easy organization, and reliable license distribution.

You only need one host to run LSF License Scheduler, but you can configure your site for a failover mechanism with multiple candidate hosts to take over the scheduling in case of a failure. This configuration can be used in a local network or across multiple sites in a wider network.

LAN example

A design center contains the following hosts configuration in a LAN:

- `lsf.conf` in Design Center A

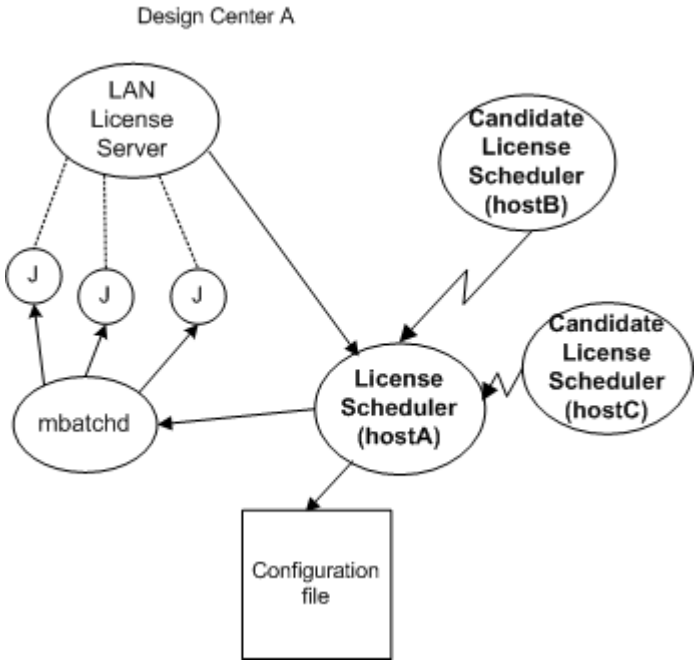
```
LSF_LIC_SCHED_HOSTS="hostA.designcenter_a.com hostB.designcenter_a.com hostC.designcenter_a.com"
```

- `lsf.licensescheduler` in Design Center A

```
HOSTS=hostA.designcenter_a.com hostB.designcenter_a.com hostC.designcenter_a.com
```

The LSF LIM daemon starts the LSF License Scheduler daemon (`bld`) on each host in the LSF_LIC_SCHED_HOSTS list.

Each host in the LSF_LIC_SCHED_HOSTS list is a potential LSF License Scheduler candidate in Design Center A and is running the `bld` daemon, but only one host becomes the LSF License Scheduler host.



In this example

- hostA.designcenter_a.com is the LSF License Scheduler host, and the remaining hosts are candidate hosts running the `bld` daemon, ready to take over the management of the licenses in case of a network failure
- Each host contains the list of candidate hosts in memory

- Each candidate License Scheduler host communicates with the LSF License Scheduler host, License Scheduler (hostA)
- If the LSF License Scheduler host fails, each candidate host checks to see if a more eligible host is running the LSF License Scheduler daemon. If not, it becomes the failover host and inherits the communication links that existed between the original LSF License Scheduler host and each candidate host. In this example, if License Scheduler on hostA fails, Candidate License Scheduler hostB is the next most eligible host, and takes over the license scheduling.

Failover provisioning for WANs

Similar to LANs, you can configure your site for a failover mechanism across multiple sites in a wide network.

You only need one host to run the LSF License Scheduler, but you can configure your site for a failover mechanism with multiple candidate hosts to take over the scheduling in case of a failure.

License scheduling across sites can be streamlined because LSF License Scheduler supports service provisioning during breaks in wide area network connections. This allows you to run LSF License Scheduler from one host that controls license scheduling across multiple sites.

Configure and start LSF License Scheduler in a WAN

In a WAN configuration:

1. As the root user, install LSF License Scheduler on each cluster in the WAN configuration and select one cluster to be the cluster.
2. In the cluster that contains the WAN license server, log on as the primary License Scheduler administrator.
3. Edit the following items in LSF_CONFDIR/lsf.licensescheduler:
4. Specify a space-separated list of hosts for the HOSTS parameter:

   HOSTS=hostname_1 hostname_2 ... hostname_n

   Where:

   hostname_1 is the most preferred host for running LSF License Scheduler.

   hostname_n is the least preferred host for running LSF License Scheduler.
5. In the Clusters section, specify the names of the clusters in the WAN.

   For example:

   ```
   Begin Clusters
   CLUSTERS
   design_SJ
   design_BOS
   End Clusters
   ```
6. In the cluster that contains the WAN license server, as the LSF primary administrator, edit LSF_CONFDIR/lsf.conf. Lines that begin with # are comments:

   Specify a space-separated list of hosts for the LSF_LIC_SCHED_HOSTS parameter:

   LSF_LIC_SCHED_HOSTS="*hostname_1 hostname_2 ... hostname_n*"

   Where:

   hostname_1, *hostname_2*, ..., *hostname_n* are hosts on which the LSF LIM daemon starts the LSF License Scheduler daemon (bld).

   The first host listed in the HOSTS list will be the default master License Scheduler host for the WAN.

   The order of the host names in LSF_LIC_SCHED_HOSTS is ignored.
7. In the other clusters in the WAN:
8. Configure the LSF_LIC_SCHED_HOSTS parameter in lsf.conf with a local list of candidate hosts.
9. Configure the HOSTS parameter in the Parameters section lsf.licensescheduler with the following list of hosts:

   - Start the list with the same list of candidate hosts as the HOSTS parameter in the cluster that contains the WAN license server.

- Continue the list with the local cluster's list of hosts from the LSF_LIC_SCHED_HOSTS parameter in lsf.conf.

10. In the cluster that contains the WAN license server and the other clusters in the WAN, run the following commands:
    1. Run `bld -C` to test for configuration errors.
    2. Run `bladmin reconfig` to configure LSF License Scheduler.
    3. Run lsadmin reconfig to reconfigure LIM.
    4. Use ps -ef to make sure that bld is running on the candidate hosts.
    5. Run badmin reconfig to reconfigure mbatchd.

Tip:

Although the bld daemon is started by LIM, bld runs under the account of the primary License Scheduler administrator. If you did not configure the LIM to automatically start the `bld` daemon on your License Scheduler hosts, run `LSF_BINDIR/blstartup` on each host to start the bld daemon.

WAN example

A design center contains the following hosts configuration in a WAN:

LIM starts bld on the following hosts:

- lsf.conf in Design Center A

```
LSF_LIC_SCHED_HOSTS="hostA1.designcenter_a.com hostA2.designcenter_a.com
hostA3.designcenter_a.com"
```

- lsf.conf in Design Center B

```
LSF_LIC_SCHED_HOSTS="hostB1.designcenter_b.com hostB2.designcenter_b.com
hostB3.designcenter_b.com"
```

License Scheduler candidate hosts are listed in the following order of preference:

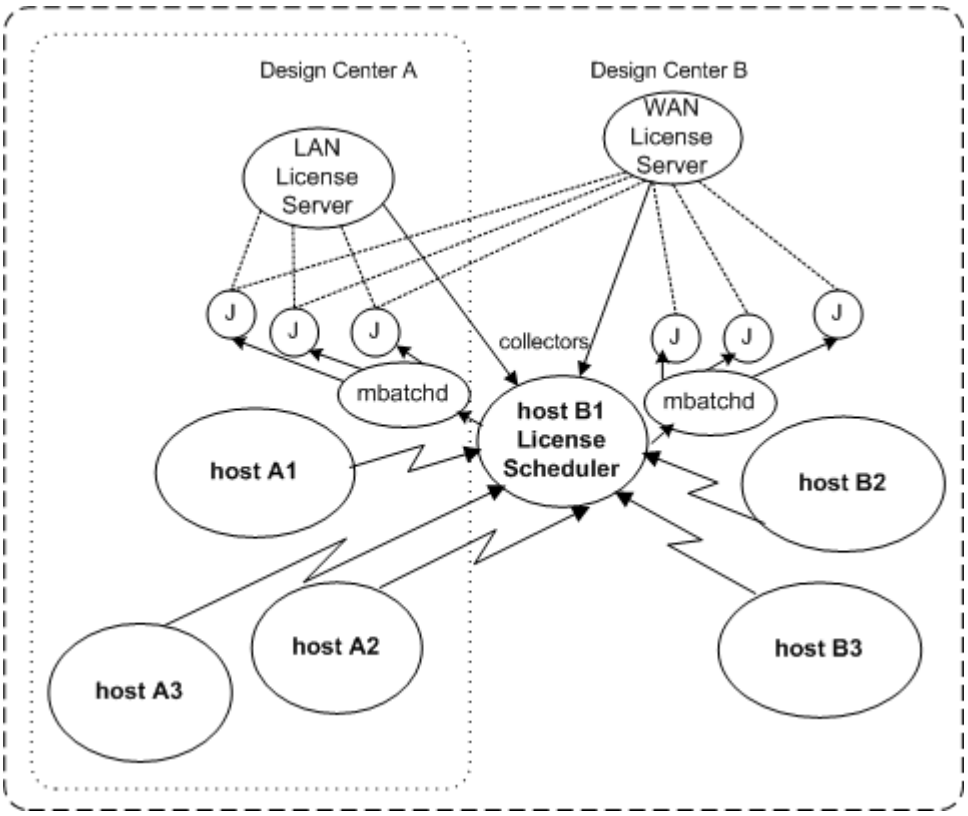
- lsf.licensescheduler in Design Center A

```
HOSTS=hostB1.designcenter_b.com hostB2.designcenter_b.com hostA1.designcenter_a.com
hostA2.designcenter_a.com hostA3.designcenter_a.com
```

- lsf.licensescheduler in Design Center B

```
HOSTS=hostB1.designcenter_b.com hostB2.designcenter_b.com hostB3.designcenter_b.com
```

The following diagram shows hostB1.designcenter_b.com, the License Scheduler host for the WAN containing Design Center A and Design Center B.



How it works

The LSF LIM daemon starts the LSF License Scheduler daemon (bld) on each host listed in LSF_LIC_SCHED_HOSTS in Design Center A and Design Center B.

Each host in the HOSTS list in Design Center A is a potential LSF License Scheduler candidate in Design Center A and is running the bld daemon, but only one host becomes the LSF License Scheduler host—the first host in the HOSTS list that is up and that is running the bld daemon. Similarly, the License Scheduler host in Design Center B is the first host in the HOSTS list that is up and that is running the bld daemon.

License Scheduler manages the licenses in Design Center A and Design Center B as follows:

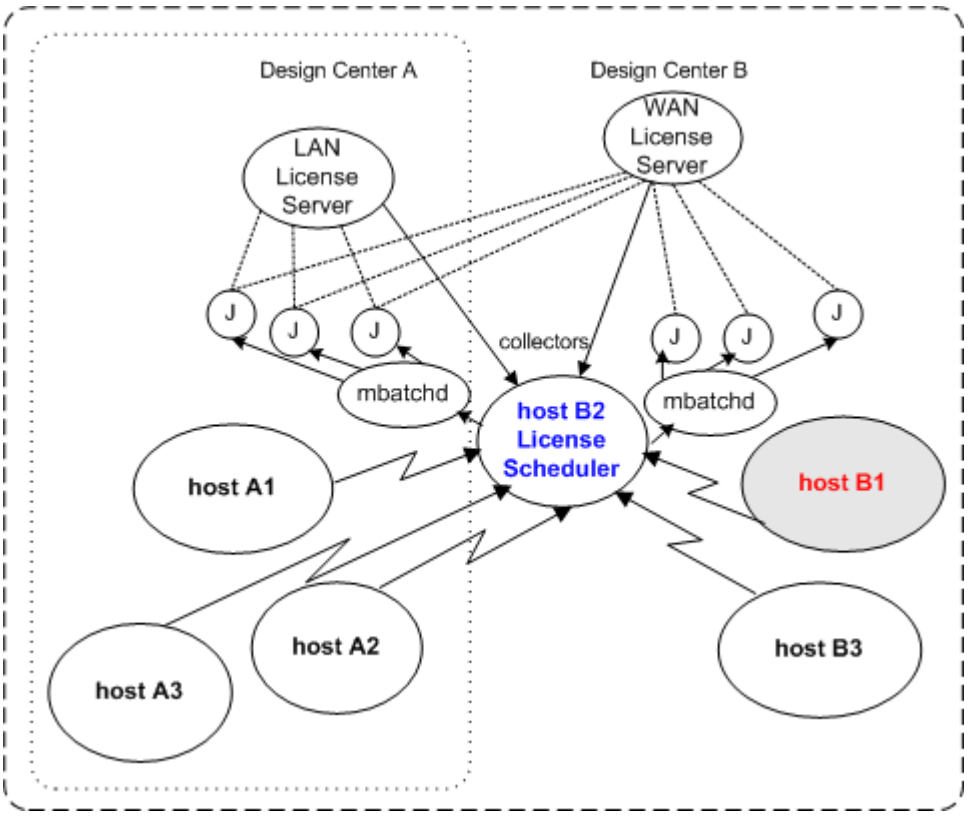
Both design centers list hostB1.designcenter_b.com at the top of their HOSTS lists. hostB1.designcenter_b.com is the License Scheduler host for Design Center A and for Design Center B. The rest of the hosts in both design centers remain on standby as candidate License Scheduler hosts. License Scheduler manages the license scheduling across the WAN connection.

Service provisioning at the host and network levels

In the above example configuration, there are two potential points of failure:

- Host failure:

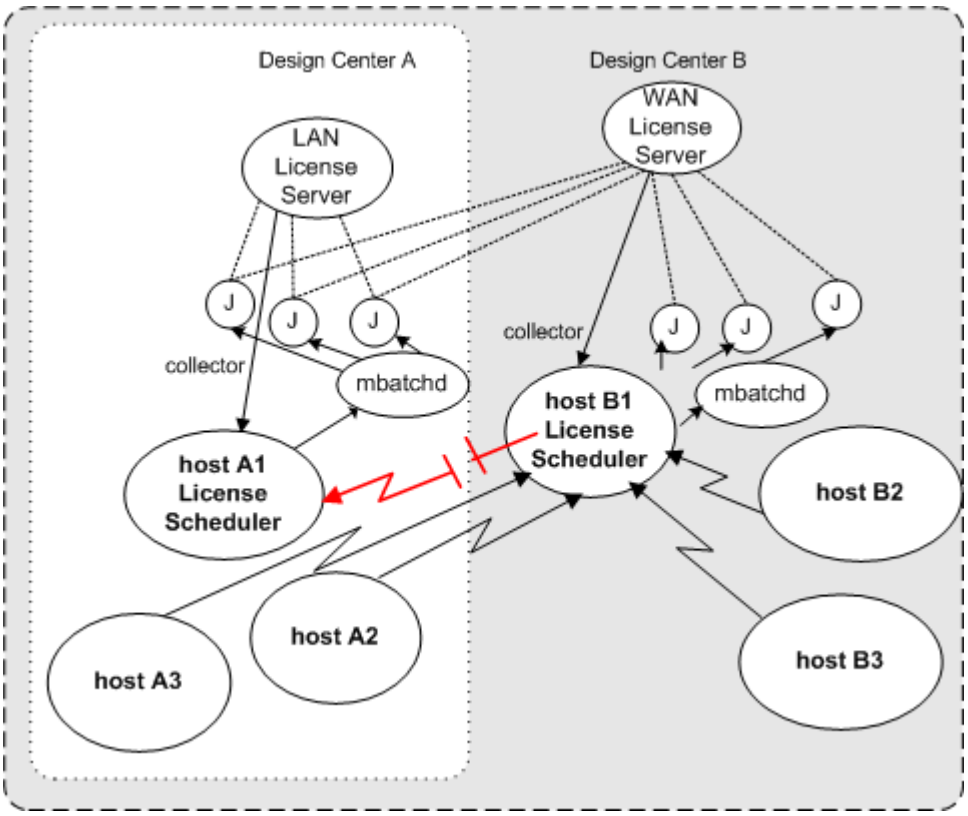
If hostB1.designcenter_b.com fails, and bld stops running, a candidate License Scheduler host must take over the license management. The next host on the HOSTS list in both Design Center A and Design Center B is hostEB2designcenter_b.com. License Scheduler fails over to this host if it is up and running.



- Network failure:

If the network connection between Design Center A and Design Center B breaks, Design Center A can no longer communicate with the hosts in Design Center B, so hostB1.designcenter_b.com and hostB2.designcenter_b.com are no longer candidate license scheduling hosts for Design Center A. The next candidate host for Design Center A is hostA1.designcenter_a.com. License management then runs locally in Design Center A on hostA1.designcenter_a.com. In Design Center B, hostB1.designcenter_b.com continues to run License Scheduler, but only manages the local network as long as the wide area network connection is down.

The local License Scheduler host, hostA1.designcenter_a.com, checks for a heartbeat from hostB1.designcenter_b.com at regular intervals, then returns license management back to hostB1.designcenter_b.com when the network connection returns.


Design Center A
Design Center B
LAN License Server
WAN License Server
J
J
J
collector
mbatchd
J
J
J
collector
mbatchd
host A1 License Scheduler
host B1 License Scheduler
host B2
host A2
host A3
host B3

Other failover provisioning

blcollect failover

It is possible for the host running the `blcollect` daemon to fail in such a way that information from a license server could stop flowing to License Scheduler. Platform Computing offers a method for allowing `blcollect` daemons to be restarted on the collector host if they die, and starting additional `blcollect` daemons on remote hosts if the primary collector host dies. Contact your Platform Computing account representative to learn more.

Use BLC_HEARTBEAT_FACTOR in the Parameters section of `lsf.licensescheduler` to enable `bld` to detect `blcollect` failure. Define the number of times that `bld` receives no response from a license collector daemon (`blcollect`) before `bld` resets the values for that collector to zero. Each license usage reported to `bld` by the collector is treated as a heartbeat. The default is 3.

FLEXnet integration failover

Though LSF and License Scheduler are designed to failover and failback gracefully, and Platform Computing provides a tool for reconfiguring the FLEXnet options file and restarting the license server automatically on failover or failback to preserve the integrity of the License Scheduler solution. Contact your Platform Computing account representative to learn more.

CHAPTER

5

Advanced Topics

Distributing license collection

You can improve performance by distributing license collection across your site.

If the `lmstat` data from all your license servers is collected in one central location, `mbatchd` has to wait for the license usage information to come in from all your license servers, and this can result in poor performance in the flow of information.

With License Scheduler, you can distribute the query to collect the information in parallel from each license server. Run the license collectors on any machines you want. Each collector can query one or more license servers. You can also set the time interval between queries for each collector.

Caution:

Do not run more than one license collector per service domain. License Scheduler closes connections to multiple collectors in one service domain.

To distribute license collection, run the command `blcollect` to manually launch the license information collection daemon.

To distribute your license information collection

1. Log on as the primary License Scheduler administrator.
2. In `lsf.licensescheduler`, specify the name of the license information collector in the ServiceDomain section:

   Specify one LIC_COLLECTOR for each service domain:

   ```
   Begin ServiceDomain
   NAME=DesignCenterA
   LIC_SERVERS=((1700@hostA)(1700@hostB))
   LIC_COLLECTOR=lic_collector_name
   LM_STAT_INTERVAL=seconds
   End ServiceDomain
   ```

   Where:

   - *lic_collector_name* is the name you specify for license information collection for a service domain. You can use any name, but you must use the same name when you start the license information collector daemon.
   - *seconds* is the time interval between queries made by the collector. The collector for this service domain will use the specified value rather than the global LM_STAT_INTERVAL value defined in the Parameters section.
3. Log on to any host.
4. If the host is not in the shared file system used by LSF, make sure you set your environment to the directory containing `lsf.licensescheduler`.

   For example:

   ```
   setenv LSF_ENVDIR "/mydir/conf"
   ```
5. Run the following command for each service domain you define in `lsf.licensescheduler`:

   blcollect -m "*host_list*" **-p** *lic_scheduler_port* **-c** *lic_collector_name*

   Where:

- *host_list*

  Specifies a space-separated list of License Scheduler candidate hosts to which license information is sent. Use fully qualified host names.
- *lice_scheduler_port*

  Corresponds to the License Scheduler listening port, which is set in `lsf.licensescheduler`.
- *lic_collector_name*

  Specifies the name of the license collector you set for `LIC_COLLECTOR` in the service domain section of `lsf.licensescheduler`.

  For example:

  blcollect -m "hostD.designcenter_b.com hostA.designcenter_a.com" -p 9581 -c CenterB

  A file named `collectors/CenterB` is created in your `LSF_WORKDIR`.

  Note:

  If you do not specify a license collector name in an LSF License Scheduler service domain, the master bld host starts a default blcollect.

Managing licenses for different application versions

If you use more than one version of an application, you can specify the version you prefer together with a legacy version if the preferred version is not available. Use the OR (||) expression to reserve your license with an alternative license choice in the rusage string of your resource requirement.

- Only use the OR expression when both versions of the license will work with the application.
- Configure the resource requirements usage string with the licenses in descending order of preference.

Configuration of usage string

bsub -R "rusage[token_name1=1||token_name2=1]" -Lp license_project_name job_name

Examples:

You are running appA version 1.5 and appA version 2.0.1. The license key for version 2.0.1 is backward compatible with version 1.5, but the license key for version 1.5 does not work with version 2.0.1.

- If you can run your job using either version of the application, try to reserve appAv201 for your job. If it is not available, you can use appAv15.

  ```
  bsub -R "rusage[appAv201=1||appAv15=1]" -Lp Lp1 myjob
  ```

- Do not use the OR expression if your job can only run on one version of the application:

  ```
  bsub -R "rusage[appAv201=1]" -Lp Lp1 myjob
  ```

- If different versions of an application require different system resources, you can specify other resources in your rusage string.

bsub -R "rusage[mem=20:appAv201=1||mem=20:swap=50:appAv15=1]" -Lp Lp1 myjob

Group license ownership

Group license ownership lets you distribute license features to license projects. Defining groups is optional. A license project should only belong to one group.

License Scheduler first balances license distribution at the group level based on group license ownership. The total number of licenses owned by all group members is the number of licenses owned by the group. License Scheduler then balances license distribution among license projects.

With group license ownership, projects can trigger preemption *either* when the project is using fewer licenses than it owns (the project is underfed) *or* when the group to which the project belongs is using fewer licenses than the group owns (the group is underfed).

Examples of preemption with group license ownership

The following tables show changes in preemption behavior based on group license ownership of a total of 20 licenses.

Project license ownership only

License project	Licenses owned	Licenses used
Lp1	5	6
Lp2	5	0
Lp3	5	7

Group license ownership

Group	License projects	Project licenses owned
GroupA	Lp1	5
	Lp2	5
GroupB	Lp3	5
	Lp4	5
Group	License projects	Project licenses owned

Project license ownership within a group

Group	License projects	Project licenses owned
GroupA	Lp1	5
	Lp2	5
GroupB	Lp3	5
	Lp4	5
Group	License projects	Project licenses owned

Configuration

Use the GROUP parameter in the Feature section of the `lsf.licensescheduler` file to define groups and their members.

Example

```
Begin Feature NAME = AppY DISTRIBUTION = LanServer1(Lp1 5/5 Lp2 5/5 Lp3 5/5 Lp4 5/5) GROUP = GroupA
(Lp1 Lp2) GroupB (Lp3 Lp4) End Feature
```

In this example, Lp1 and Lp2 belong to the group GroupA. Lp3 and Lp4 belong to the GroupB group.

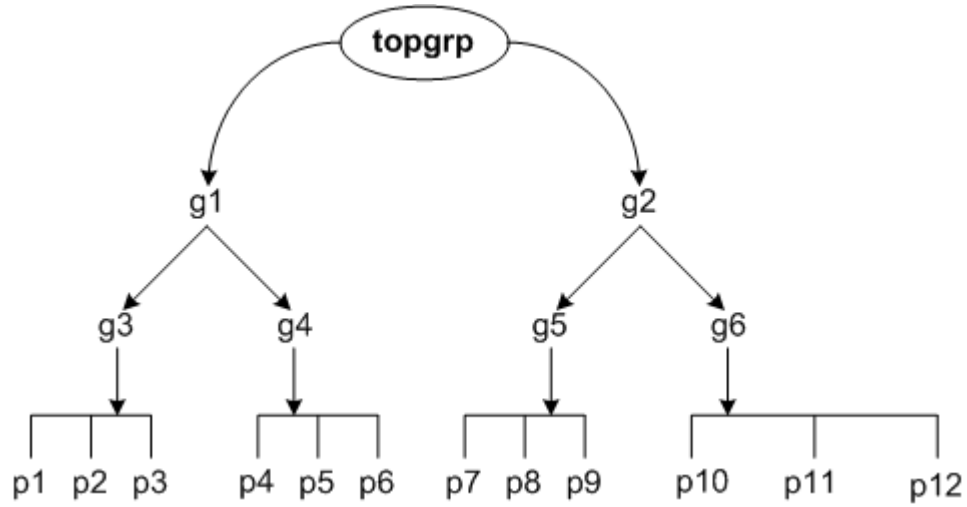
Hierarchical fairshare among project groups

Project groups pool multiple service domains together and treat them as one source for licenses, and distribute them in a hierarchical fairshare tree. The leaves of the policy tree are the license projects that jobs can belong to. Each project group in the tree has a set of values, including shares and limits.

License ownership can only be configured at the leaf level; that is, on individual license projects. Ownership of a given internal node equals to sum of the ownership of all of its direct children.

The grouping of projects is per feature. Each feature has its own hierarchical group, but features can share the same hierarchy. The hierarchical scheduling is done per feature across service domains. This is the difference between hierarchical and non-hierarchical scheduling, which is done per feature and per service domain. With a hierarchical grouping of projects defined, service domains become license containers, free or in use.

Example project group configuration



The following project group configuration in `lsf.licensescheduler` implements this hierarchy:

```
Begin ProjectGroup
GROUP                SHARES    OWNERSHIP   LIMITS     NON_SHARED
(topgrp (g1 g2))     (1 1)     (4 4)       (10 10)    (4 4)
(g1 (g3 g4))         (1 1)     (0 4)       (10 10)    (0 4)
(g2 (g5 g6))         (1 1)     (2 2)       (- 5)      (2 2)
(g3 (p1 p2 p3))      (1 1 2)   ()          (3 4 5)    ()
(g4 (p4 p5 p6))      (1 1 1)   (1 1 1)     ()         (- 3 0)
(g5 (p7 p8 p9))      (1 1 1)   (2 - 2)     ()         (1 - 1)
(g6 (p10 p11 p12))   (1 1 1)   (2 2 2)     (4 4 4)    (1 0 1)
End ProjectGroup
```

```
Begin Feature
NAME = AppZ
GROUP_DISTRIBUTION = topgrp
SERVICE_DOMAINS = LanServer WanServer
End Feature
```

In this example a dash (-) denotes the default value, as in LSF. This default value depends on the column, for example by default OWNERSHIP should be 0, whereas LIMITS should be infinity (unlimited). A dash is not allowed in the SHARES column.

Since the service domains are no longer used in the distribution policies, they only need to be listed under the SERVICE_DOMAINS parameter in each Feature section.

Maximum token limit in project groups

By default, License Scheduler distributes all available tokens if possible. Project group configuration enables you to set hard limits on available tokens so that some tokens may not be distributed even if they are available.

In the following example, a total of 6 licenses are available:

```
Begin ProjectGroup
GROUP       SHARES OWNERSHIP LIMITS NON_SHARED
(Root(A B)) (1 1)   ()        ()     ()
(A (c d))   (1 1)   ()        (1 1)  ()
(B (e f))   (1 1)   ()        ()     ()
End ProjectGroup
```

When there is no demand for license tokens, License Scheduler only allocates 5 tokens according to the distribution. License Scheduler gives 3 tokens to group A and 3 tokens to group B, but project c and project d are limited to 1 token each, so 1 token will not be allocated within group A. As more demand comes in for project e and project f, the unallocated tokens are distributed to group B.

Shared and non-shared licenses

Normally, the total number of non-shared licenses should be less than the total number of license tokens available. License tokens may not be available to project groups if the total non-shared licenses for all groups is greater than the number of shared tokens available.

For example, feature p4_4 is configured as follows, with a total of 4 tokens:

```
Begin Feature
NAME =p4_4
# total token value is 4
GROUP_DISTRIBUTION=final
SERVICE_DOMAINS=LanServer
End Feature
```

The correct configuration is:

```
GROUP             SHARES    OWNERSHIP   LIMITS      NON_SHARED
(final (G2 G1))   (1 1)     (2 0)          ()          (2 0)
(G1 (AP2 AP1))    (1 1)     (1 1)          ()          (1 1)
```

Project group configuration like the following is valid, but could cause tokens not to be available for the leaf-level projects of group G1.

```
Begin ProjectGroup
GROUP           SHARES  OWNERSHIP   LIMITS    NON_SHARED
(final(G1 G2))  (1 1)   (2 2)          ()        (2 2)
(G1(AP2 AP1 ))  (1 1 )  (1 1)          ()        (1 1)
End ProjectGroup
```

The total non-shared tokens is 6, but the total available is 4, which can cause non-shared license tokens not to be available. License scheduler satisfies the non-shared configuration first, so it gives 2 tokens to group G2 and 2 tokens to group G1. No tokens are left, and the non-shared configuration for projects AP2 and AP1 is not satisfied.

For projects defined with NON_SHARED_DISTRIBUTION, you must assign the project OWNERSHIP an equal or greater number of tokens defined in the DISTRIBUTION line.

Viewing information about project groups

Use `blstat -G` to view the hierarchical dynamic license information:

blstat -G

```
FEATURE: p1_f1
SERVICE_DOMAINS:
TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 5    OTHERS: 0
SHARE_INFO_FOR:  /topgrp
GROUP/PROJECT           SHARE   OWN  INUSE RESERVE FREE   DEMAND
```

```
g2                      100.0 %   4     0    0        4     0
SHARE_INFO_FOR:  /topgrp/g2
GROUP/PROJECT           SHARE   OWN  INUSE RESERVE FREE   DEMAND      p3                      50.0
%   0     0    0        2     0
p4                      50.0 %   0     0    0        2     0
FEATURE: p1_f2
SERVICE_DOMAINS:
TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 10   OTHERS: 0
SHARE_INFO_FOR:  /topgrp
GROUP/PROJECT           SHARE   OWN  INUSE RESERVE FREE   DEMAND
g2                      100.0 %   4     0    0        4     0
SHARE_INFO_FOR:  /topgrp/g2
GROUP/PROJECT           SHARE   OWN  INUSE RESERVE FREE   DEMAND
p3                      50.0 %   0     0    0        2     0
p4                      50.0 %   0     0    0        2     0
```

blinfo -G

Use `blinfo -G` to view the hierarchical configuration:

blinfo -G

```
GROUP                 SHARES    OWNERSHIP   LIMITS     NON_SHARED
(topgrp (g1 g2))      (1 1)     (4 4)          (10 10)    (4 4)
(g1 (g3 g4))          (1 1)     (2 4)          (10 10)    (0 4)
(g2 (g5 g6))          (1 1)     (2 2)          (- 5)      (2 2)
(g3 (p1 p2 p3))       (1 1 2)   ()          (3 4 5)    ()
(g4 (p4 p5 p6))       (1 1 1)   (1 3 1)     ()         (- 3 0)
(g5 (p7 p8 p9))       (1 1 1)   (2 - 2)     ()         (1 - 1)
(g6 (p10 p11 p12))    (1 1 1)   (2 2 2)     (4 4 4)    (1 0 1)
```

Use `blinfo -G` to view hierarchical project group priority information.

blinfo -G

```
GROUP            SHARES    OWNERSHIP   LIMITS   NON_SHARED  PRIORITY
(root (A B C))    (1 1 1)      ()          ()          ()          (3 2 0)
(A (P1 D))        (1 1)      ()          ()          ()          (3 5)
(B (P4 P5))       (1 1)      ()          ()          ()          ()
(C (P6 P7 P8))    (1 1 1)      ()          ()          ()          (8 3 0)
(D (P2 P3))        (1 1)     ()          ()          ()          (2 1)
```

Defining flat project priority

If no hierarchical project groups are defined, the default project configuration is flat. The priority of a project has nothing to do with its position in the hierarchy. Project priority values can be compared between all leaf nodes.

To configure flat project priority in `lsf.licensescheduler`, set the PRIORITY column in the Project section. For example:

```
Begin Projects
PROJECTS    PRIORITY
P1             2
P2             3
P3             5
P4             1
…              …
P8             7
End Projects
```

Configure tree priority to define priorities for all nodes in the hierarchy

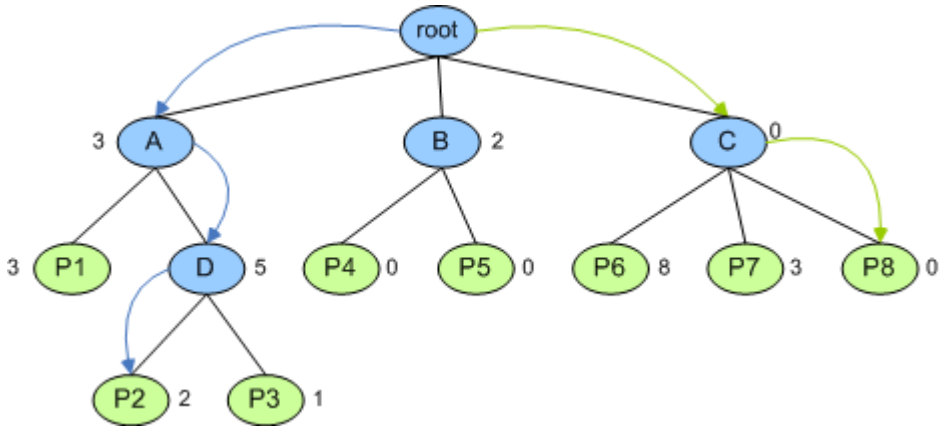
To configure tree priority in `lsf.licensescheduler`, specifiy PRIORITY in the ProjectGroup configuration section. For example:

```
Begin ProjectGroup
GROUP             SHARES   OWNERSHIP  LIMITS  NON_SHARED PRIORITY
(root(A B C))     (1 1 1)  ()         ()         ()      (3 2 -)
```

```
(A (P1 D))        (1 1)    ()        ()        ()      (3 5)
(B (P4 P5))       (1 1)    ()        ()        ()      ()
(C (P6 P7 P8))    (1 1 1)  ()        ()        ()      (8 3 0)
(D (P2 P3))       (1 1)    ()        ()        ()      (2 1)
End ProjectGroup
```

By default, priorities are evaluated from top to bottom. The priority of a given node is first decided by the priorities of its parent nodes. The values are only comparable between siblings.

The following figure illustrates the example configuration:



The priority of each node is shown beside the node name. If priority is not defined, by default is set to 0 (nodes P4 and P5 under node B).

To find the highest priority leaf node in the tree License Scheduler traverses the tree from root to node A to node D to project P2.

To find the lowest priority leaf node in the tree, License Scheduler traverses the tree from root to node C to project P8.

When two nodes have the same priority, for example, projects P4 and P5, priority is determined by accumulated inuse usage at the time the priorities are evaluated.

When a leaf node in branch A wants to preempt a token from branch B or C, branch C is picked because it has a lower priority than branch B.

How preemption happens in a project group hierarchy

License preemption in License Scheduler takes place when no more free tokens are available, and there is at least one underfed project and at least one overfed project. When PRIORITY is configured in the project group, License scheduler can preempt license tokens at leaf nodes across the branches in the group hierarchy configuration.

By default, when no PRIORITY is configured in the project group, License Scheduler only preempts within a particular branch for tokens to preempt. This approach works like a flat project configuration—projects are underfed only when they have need and their inuse tokens are less than their owned tokens. Projects are overfed when their inuse tokens are greater than their owned tokens.

Configure hierarchical project group preemption

Configure hierarchical project group preemption two ways:

- Top-down—License Scheduler always tries to preempt tokens from the projects that are furthest away in the hierarchy first. This balances the token ownership from top to bottom. Top-down preemption does not support ENABLE_MINJOB_PREEMPTION.

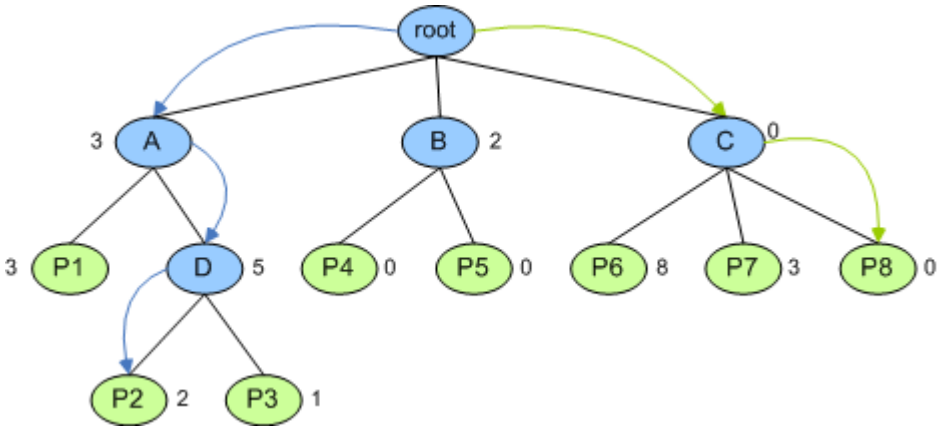
To enable top-down preemption, configure priorities for projects in the project group hierarchy. Top-down preemption is the default when priorities are configured in a project group hierarchy.

- Bottom-up—License Scheduler always tries to preempt tokens from the closest projects in the hierarchy first. This balances token ownership from bottom to top. Bottom-up preemption does not conform to GROUP preemption in a flat project group configuration. Bottom-up preemption does not support ENABLE_MINJOB_PREEMPTION.

To enable bottom-up preemption, configure priorities for projects in the project group hierarchy, and specify LS_PREEMPT_PEER=Y in the Parameters section of `lsf.licensescheduler`.

Hierarchical project group preemption examples

The following configuration examples use the ownership configuration illustrated in the following figure:



Top-down configuration

- Needs from a leaf node with ownership not satisfied:

P1: inuse=0

P2: inuse=2

P3: inuse=0

P4: inuse=2

If P1 needs one token, it will preempt one token from P4 instead of P2.

- Needs from a leaf node without ownership or ownership are already satisfied:

P1: inuse=2

P2: inuse=0

P3: inuse=0

P4: inuse=2

If P2 needs one token, it can trigger preemption and get one token from P4. Because the ownership of its grandparent node (group A owns 3) is not satisfied.

- Add limits to the tree. Assume limits defined on the internal nodes are same as ownership: group B limits=2, group A limits=3. Needs from a leaf node with ownership not satisfied:

P1: inuse=0

P2: inuse=2

P3: inuse=0

P4: inuse=2

When P1 needs one token, it can only preempt from P2 because the limit of group B is 2 to satisfy the total limits in that branch.

- Add limits to the tree. Assume limits defined on the internal nodes are same as ownership: group B limits=2, group A limits=3. Needs from a leaf node without ownership or ownership are already satisfied:

P1: inuse=2

P2: inuse=0

P3: inuse=0

P4: inuse=2

If P2 needs one token, now because of limits, P2 can no longer preempt any tokens.

Bottom-up configuration

- Needs from a leaf node with ownership not satisfied:

P1: inuse=0

P2: inuse=2

P3: inuse=0

P4: inuse=2

If P1 needs one token, it will preempt from P2 (instead of P4 as in the top-down configuration). This balances token preemption within its own sub-tree first.

- Needs from a leaf node without ownership or ownership are already satisfied:

P1: inuse=2

P2: inuse=0

P3: inuse=0

P4: inuse=2

If P2 needs one token, it can preempt a token form P4 because the ownership of its grandparent (group A) has not been satisfied yet.

- Add limits to the tree. Assume limits defined on the internal nodes are same as ownership: group B limits=2, group A limits=3. Needs from a leaf node with ownership not satisfied:

P1: inuse=0;

P2: inuse=2;

P3: inuse=0;

P4: inuse=2;

If P1 needs one token, it will preempt from P2, the closest leaf node, where the limits of this subtree have already been met.

- Add limits to the tree. Assume limits defined on the internal nodes are same as ownership: group B limits=2, group A limits=3. Needs from a leaf node without ownership or ownership are already satisfied:

P1: inuse=2

P2: inuse=0

P3: inuse=0

P4: inuse=2

If P2 needs one token, it can no longer get tokens from the other branches because the limits of its parent node B have already been reached.

Configuring multiple administrators

The primary License Scheduler admin account must have write permissions in the LSF working directory of the primary LSF admin account.

The administrator account uses a list of users that you specified when you installed LSF License Scheduler. Edit this parameter if you want to add or change administrators. The first user name in the list is the primary License Scheduler administrator. By default, all the working files and directories created by License Scheduler are owned by the primary License Scheduler account.

1. Log on as the primary License Scheduler administrator.
2. In `lsf.licensescheduler`, edit the ADMIN parameter if you want to change the License Scheduler administrator. You can specify multiple administrators separated by spaces.

   For example:

   ```
   ADMIN = lsfadmin user1 user2 root
   ```
3. Run `bld -C` to test for configuration errors.
4. Run `bladmin reconfig all` to make the changes take effect.

Allocating license shares to clusters and interactive jobs

With License Scheduler, you can allocate shares of license features across clusters and between LSF jobs and interactive jobs run through the LSF Task Manager (`taskman`).

- You can globally enable a share of all license features for interactive tasks.
- You can configure the allocation of license shares to:

Change the share number between clusters for a feature

Limit the scope of license usage and change the share number between LSF jobs and interactive tasks for a feature

Enabling a share of licenses for interactive tasks

To globally enable one share of the licenses for interactive tasks, you must set the ENABLE_INTERACTIVE parameter in lsf.licensescheduler.

In lsf.licensescheduler, edit the Parameters section:

```
Begin Parameters
...
ENABLE_INTERACTIVE = y
...
End Parameters
```

Tip:

By default, ENABLE_INTERACTIVE is not set. License Scheduler allocates licenses equally to each cluster and does not distribute licenses for interactive tasks.

Configuring allocation for specific features

To specify ALLOCATION for a specific feature, you set the ALLOCATION keyword in the Features section of lsf.licensescheduler. This feature ignores the global setting of the ENABLE_INTERACTIVE parameter because ALLOCATION is configured for the feature.

In lsf.licensescheduler, edit the Features section:

```
Begin Feature
NAME = AppX
DISTRIBUTION = LanServer1 (Lp1 1)
ALLOCATION = Lp1 (Cluster1 1 Cluster2 1 interactive 1)
End Feature
```

Default ALLOCATION setting

ALLOCATION is not configured. The ENABLE_INTERACTIVE parameter is not set.

Each cluster receives one share. Interactive tasks receive no shares.

Example

For two clusters and 12 licenses,

```
Begin Feature
NAME = AppX
DISTRIBUTION = LanServer (Lp1 1)
End Feature
```

Six licenses are allocated to each cluster. No licenses are allocated to interactive tasks.

Changing the ALLOCATION configuration

You can edit the default ALLOCATION configuration, resulting in the following scenarios. Each example contains two clusters and 12 licenses of a specific feature.

Example 1

ALLOCATION is not configured. The ENABLE_INTERACTIVE parameter is not set.

```
Begin Parameters
...
ENABLE_INTERACTIVE = n
...
End Parameters
Begin Feature
NAME = AppX
DISTRIBUTION = LanServer (Lp1 1)
End Feature
```

Six licenses are allocated to each cluster. No licenses are allocated to interactive tasks.

Example 2

ALLOCATION is not configured. The ENABLE_INTERACTIVE parameter is set.

```
Begin Parameters
...
ENABLE_INTERACTIVE = y
...
End Parameters
```

```
Begin Feature
NAME = AppX
DISTRIBUTION = LanServer (Lp1 1)
End Feature
```

Four licenses are allocated to each cluster. Four licenses are allocated to interactive tasks.

Example 3

In the following example, the ENABLE_INTERACTIVE parameter does not affect the ALLOCATION configuration of the feature.

ALLOCATION is configured. The ENABLE_INTERACTIVE parameter is set.

```
Begin Parameters
...
ENABLE_INTERACTIVE = y
...
End Parameters
Begin Feature
NAME = AppY
DISTRIBUTION = LanServer (Lp1 1)
ALLOCATION = Lp1(cluster1 1 cluster2 0 interactive 1)
End Feature
```

The ENABLE_INTERACTIVE setting is overridden for feature AppY. Licenses are shared equally between cluster1 and interactive tasks. Six licenses of AppY are allocated to cluster1. Six licenses are allocated to interactive tasks.

Example 4

In the following example, the ENABLE_INTERACTIVE parameter does not affect the ALLOCATION configuration of the feature.

ALLOCATION is configured. The ENABLE_INTERACTIVE parameter is not set.

```
Begin Parameters
...
ENABLE_INTERACTIVE = n
...
End Parameters
```

```
Begin Feature
NAME = AppZ
DISTRIBUTION = LanServer (Lp1 1)
ALLOCATION = Lp1(cluster1 0 cluster2 1 interactive 2)
End Feature
```

The ENABLE_INTERACTIVE setting is ignored for feature AppZ. Four licenses of AppZ are allocated to cluster2. Eight licenses are allocated to interactive tasks.

Enable a share of licenses for interactive tasks

1. To globally enable one share of the licenses for interactive tasks, you must set the `ENABLE_INTERACTIVE` parameter in `lsf.licensescheduler`.

   In `lsf.licensescheduler`, edit the `Parameters` section:

   ```
   Begin Parameters
   ...
   ENABLE_INTERACTIVE = y
   ...
   End Parameters
   ```

   By default, ENABLE_INTERACTIVE is not set. License Scheduler allocates licenses equally to each cluster and does not distribute licenses for interactive tasks.

Configure allocation for specific features

1. To specify ALLOCATION for a specific feature, you set the ALLOCATION keyword in the Features section of lsf.licensescheduler. This feature ignores the global setting of the ENABLE_INTERACTIVE parameter because ALLOCATION is configured for the feature.

   In `lsf.licensescheduler`, edit the `Features` section:

   ```
   Begin Feature
   NAME = AppX
   DISTRIBUTION = LanServer1 (Lp1 1)
   ALLOCATION = Lp1 (Cluster1 1 Cluster2 1 interactive 1)
   End Feature
   ```

Default ALLOCATION setting

`ALLOCATION` is not configured. The `ENABLE_INTERACTIVE` parameter is not set.

Each cluster receives one share. Interactive tasks receive no shares.

Example

For two clusters and 12 licenses,

```
Begin Feature
NAME = AppX
DISTRIBUTION = LanServer (Lp1 1)
End Feature
```

Six licenses are allocated to each cluster. No licenses are allocated to interactive tasks.

Application integrations

The examples described in this section demonstrate how you can integrate LSF License Scheduler with your applications. Contact Platform Computing Professional Services for specific integration projects.

When should integration be considered?

Applications with licenses that come with several features require some configuration and are candidates for integration with LSF License Scheduler. The application users may not know which license features they must check out. A script is used to determine which license features require LSF License Scheduler tokens.

Integration requirements

- The applications must work with Platform LSF or LSF Task Manager (`taskman`)
- The applications should be able to release their licenses upon job suspension
- LSF License Scheduler is configured to preempt low priority jobs with higher priority jobs
- For job submission, the -Lp and -R options are required
- License projects must be defined in the lsf.licensescheduler file

Integration steps

1. Write a script that determines which license features are needed for a job. The script receives the license requirements from application options or input files, where available, and outputs the information to the bsub command using the following format:

   ```
   AppLicense=1:Feature=1
   ```
2. Configure the application features and policy rules in the lsf.licensescheduler file.
3. Update wrapper scripts to call the script that outputs the feature information to the bsub command. Alternatively, show application users how to submit their jobs with their required features in the bsub command.

Integration example: Synopsys Design Compiler licenses

Synopsys® Design Compiler® users run the Perl script with a .scr input file.

The Perl script, dc_features.pl is added to the LSF_BINDIR.

To use the Perl script in bash shell:

```
features=`dc_features.pl [optional parameters] -f synthesis.scr [optional parameters]`
if [ $? = 0 ] ; then
Project="Lp1"
bsub -Lp $Project -R "rusage[$features]" dc_shell ... -f synthesis.scr ...
fi
```

Contact Platform Professional Services for usage in other shells or in Windows.

Job submission example:

```
bsub -Lp Lp1 -R "rusage[$features]" dc_shell -f afile.scr ...
```

Integration example: MSC Nastran licenses

MSC Nastran users run the Perl script with a .bdf input file.

The Perl script, nastran_features.pl is added to the LSF_BINDIR.

To use the Perl script in C-shell:

```
set features=`nastran_features.pl [optional parameters] input.bdf batch=no
...
`
if ( $status == 0 ) then
Project="Lp2"
   bsub -Lp $Project -R "rusage[$features]" nastran ... input.bdf batch=no ...
endif
```

Contact Platform Professional Services for usage in other shells or in Windows.

Job submission example:

```
bsub -Lp Lp2 -R "rusage[$features]" nastran -f afile.bdf batch=no ...
```

License usage enforcement with FLEXnet

Depending on how each application uses the licenses, License Scheduler manages and enforces license usage in different ways. License Scheduler has two levels of classifications for applications depending on how the application uses the license features and whether these license features are known at the start of the job.

License Scheduler and FLEXnet work with both classes of applications.

Class A and B applications

Class A applications are those where all license features needed to run its jobs are known before the start of the job, and these features are used for the entire job.

Class B applications are those where all license features needed to run the job are known before the start of the job, but not all of these features are used all the time. The period of time that the license features are not in use are known to License Scheduler, so additional requests for the unused license features can be handled appropriately using the Class A request process.

Class A and B license requests

1. The user application makes a license usage request to the Platform LSF cluster.
2. LSF sends a query to License Scheduler to see if the license token can be given to the application.
3. When License Scheduler grants permission, LSF gives authorization to the user application.
4. The user application sends a request to FLEXnet to check out a license.

License resource duration

If you specify DYNAMIC=Y in the `lsf.licensescheduler` Feature section, you must specify a duration in an rusage resource requirement for the feature. This enables License Scheduler to treat the license as a dynamic resource and prevents License Scheduler from scheduling tokens for the feature when they are not available, or reserving license tokens when they should actually be free.

For example, feature p1_2 is configured with DYNAMIC=Y:

```
Begin Feature
NAME = p1_2
DISTRIBUTION= Lan1 (a 1 b 1 c 1 default 1)
DYNAMIC=Y
End Feature
```

A job is submitted requesting license feature p1_2 with a duration of 2 minutes:

```
taskman -R "rusage[p1_2=1:duration=2]" myjob [1] 7141
```

myjob is granted the license token it requests:

```
blstat -t p1_2
FEATURE: p1_2
SERVICE_DOMAIN: Lan1
TOTAL_INUSE: 1    TOTAL_RESERVE: 0    TOTAL_FREE: 1    OTHERS: 0
PROJECT                SHARE   OWN  INUSE RESERVE FREE   DEMAND
a                      25.0 %  0   0       0      0     0
b                      25.0 %  0   0       0      0     0
c                      25.0 %  0   0       0      1     0
default                25.0 %  0   1       0      0     0
```

After the duration expires, myjob checks in the license, and the token for p1_2 is available again:

```
bltasks
TID    USER     STAT      HOST    PROJECT       FEATURES        CONNECT TIME
41     user1    RUN       hostA   default       p1_2            Oct 27 07:15:10 %
blstat -t p1_2
```

```
FEATURE: p1_2
SERVICE_DOMAIN: Lan1
TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 2    OTHERS: 0
PROJECT                 SHARE   OWN  INUSE RESERVE FREE   DEMAND
a                       25.0 %  0   0       0      0     0
b                       25.0 %  0   0       0      1     0
c                       25.0 %  0   0       0      1     0
default                 25.0 %  0   0       0      0     0
```

Class C applications

Class C applications require an initial feature license to start a job and additional feature or sub-feature licenses during job execution. The user who submits the job knows the main license feature needed to start the job, but might not know the additional feature names or the number of additional features required. At any time, the user application can either make a request to LSF without requesting verification from License Scheduler, or it can bypass LSF entirely by sending the license request directly to the FLEXnet license servers.

Managed Class C license requests

1. The user application makes a request to LSF without requesting verification from License Scheduler.
2. LSF gives authorization to the user application because the request did not specify the need for License Scheduler verification.
3. The user application sends a request to FLEXnet to check out a license.

Managing class C license checkout

To enforce license distribution policies for class C license features, configure ENABLE_DYNAMIC_RUSAGE=Y in the feature section of `lsf.licensescheduler`

Example: managed class C workload

License feature feat2 is configured as a managed class C feature:

```
Begin Feature
NAME = feat2
DISTRIBUTION = LanServer(proj1 1 default 1)
ENABLE_DYNAMIC_RUSAGE = y
End Feature
```

User user1 submits a job to run app1, which specifies license feature feat1:

```
bsub -R "rusage[feat1=1]" -Lp proj1 app1
```

The job runs and license feat1 is checked out:

```
blstat FEATURE: feat1 SERVICE_DOMAIN: LanServer TOTAL_INUSE: 1    TOTAL_RESERVE: 0    TOTAL_FREE:
4    OTHERS: 0
  PROJECT                SHARE   OWN  INUSE RESERVE FREE   DEMAND
 proj1                   50.0 %   0     1    0      2       0
 default                 50.0 %   0     0    0      3       0
FEATURE: feat2
SERVICE_DOMAIN: LanServer TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 10   OTHERS: 0
PROJECT                 SHARE   OWN  INUSE RESERVE FREE   DEMAND
 proj1                   50.0 %   0     0    0      5       0
 default                 50.0 %   0     0    0      5       0
```

```
blusers -l
FEATURE  SERVICE_DOMAIN  USER   HOST    NLICS   NTASKS OTHERS  DISPLAYS PIDS
feat1    LanServer       user1  hostA   1       1      0     (/dev/tty) (16326)
```

```
% blusers -J
JOBID   USER     HOST     PROJECT          CLUSTER        START_TIME
1896    user1    hostA    proj1            cluster1       Aug  9 10:01:25
RESOURCE        RUSAGE        SERVICE_DOMAIN
feat1           1             LanServer
```

Later, app1 checks out feature feat2. Since it was not specified at job submission, feat2 is a class C license checkout. But since it is configured with ENABLE_DYNAMIC_RUSAGE=Y, jobs that require feat2 are considered managed workload, and subject to the distribution policies of project proj1:

blstat

```
FEATURE: feat1
SERVICE_DOMAIN: LanServer
TOTAL_INUSE: 1    TOTAL_RESERVE: 0    TOTAL_FREE: 4    OTHERS: 0
PROJECT                 SHARE   OWN  INUSE RESERVE FREE   DEMAND
 proj1                    50.0 %   0     1    0      2       0
 default                  50.0 %   0     0    0      2       0
```

```
FEATURE: feat2
SERVICE_DOMAIN: LanServer
TOTAL_INUSE: 1    TOTAL_RESERVE: 0    TOTAL_FREE: 9    OTHERS: 0
PROJECT                 SHARE   OWN  INUSE RESERVE FREE   DEMAND
 proj1                    50.0 %   0     1    0      4       0
 default                  50.0 %   0     0    0      5       0
```

blusers -l

```
FEATURE  SERVICE_DOMAIN  USER   HOST    NLICS  NTASKS  OTHERS  DISPLAYS   PIDS
feat1     LanServer       user1  hostA  1      1       0    (/dev/tty)  (16326)
feat2     LanServer       user1  hostA  1      1       0    (/dev/tty)  (16344)
```

blusers -J

```
JOBID   USER      HOST      PROJECT             CLUSTER        START_TIME
1896    user1     hostA     proj1               cluser1        Aug  9 10:01:25
RESOURCE        RUSAGE        SERVICE_DOMAIN
feat1           1             LanServer
feat2           1 (class-C)   LanServer
```

Using license feature locality

License feature locality allows you to limit features from different service domains to a specific cluster, so that License Scheduler does not grant tokens to jobs from license that legally cannot be used on the cluster requesting the token.

Example configuration: 2 sites and 4 service domains

Some of your service domains may have geographical restrictions when serving licenses. In this example, two clusters in one location can run hspice jobs. and 4 service domains are defined for the hpsice feature:

- SD1 is a local license file for clusterA with 25 hspice licenses
- SD2 is a local license file for clusterB with 65 hspice licenses
- SD3 is a WANable license with 15 hspice licenses
- SD4 is a globally WANable license with 7 hspice licenses

The geographical license checkout restrictions are:

- Jobs in clusterA can check out licenses from SD1 SD3 and SD4 but not SD2
- Jobs in clusterB can check out licenses from SD2 SD3 and SD4 but not SD1

Configuring license feature locality (LOCAL_TO)

Use LOCAL_TO in the feature configuration in `lsf.licensescheduler` to configure token locality for the license feature. You must configure different feature sections for same feature based on their locality.

LOCAL_TO allows you to limit features from different service domains to specific clusters, so License Scheduler only grants tokens of a feature to jobs from clusters that are entitled to them.

By default, if LOCAL_TO is not defined, the feature is available to all clients and is not restricted by geographical location. When LOCAL_TO is configured, for a feature, License Scheduler treats license features served to different locations as different token names, and distributes the tokens to projects according the distribution and allocation policies for the feature.

For example, if your license servers restrict the serving of license tokens to specific geographical locations, use LOCAL_TO to specify the locality of a license token if any feature cannot be shared across all the locations. This avoids having to define different distribution and allocation policies for different service domains, and allows hierarchical group configurations.

License Scheduler manages features with different localities are different resources. Use `blinfo`, `blusers`, and `blstat` to see the different resource information for the features depending on their cluster locality.

License features with different localities must be defined in different feature sections. The same Service Domain can appear only once in the configuration for a given license feature.

A configuration like LOCAL_TO=Site1(clusterA clusterB) configures the feature for more than one cluster.

A configuration like LOCAL_TO=clusterA configures locality for only one cluster. This is the same as LOCAL_TO=clusterA(clusterA).

Cluster names must be the names of clusters defined in the Clusters section of `lsf.licensescheduler`.

Examples:

```
Begin Feature
NAME = hspice
DISTRIBUTION = SD1 (Lp1 1 Lp2 1)
LOCAL_TO = siteUS(clusterA clusterB)
End Feature

Begin Feature
NAME = hspice
DISTRIBUTION = SD2 (Lp1 1 Lp2 1)
LOCAL_TO = clusterA
End Feature

Begin Feature
NAME = hspice
DISTRIBUTION = SD3 (Lp1 1 Lp2 1) SD4 (Lp1 1 Lp2 1)
End Feature
```

Or use the hierarchical group configuration (GROUP_DISTRIBUTION):

```
Begin Feature
NAME = hspice
GROUP_DISTRIBUTION = group1
SERVICE_DOMAINS = SD1
LOCAL_TO = siteUS(clusterA clusterB)
End Feature

Begin Feature
NAME = hspice
GROUP_DISTRIBUTION = group1
SERVICE_DOMAINS = SD2
LOCAL_TO = clusterA
End Feature

Begin Feature
NAME = hspice
GROUP_DISTRIBUTION = group1
SERVICE_DOMAINS = SD3 SD4
End Feature
```

How locality works

When LOCAL_TO is specified in the feature definition in `lsf.licensescheduler`, license resources requested from different clusters are mapped to different tokens in License Scheduler

You must make sure that your features are configured so that the applications always first tries to checkout licenses locally.

Features with different locality are treated a different tokens by License Scheduler. You must configure separate feature sections for same feature with different localities. For example, feature hspice, because of locality, comprises three different tokens, hspice@clusterA, hspice@clusterB, and hspice (without locality).

How job license demand is passed to License Scheduler

When License Scheduler receives license requests from LSF, it knows where the request is from, and it will interpret the request into demands for tokens usable by that cluster. For example, if clusterA sends a request to `bld` asking for 1 hspice licence, License Scheduler marks the demand for both hspice@clusterA and hspice. When the job gets either token to run, the demand will be cleaned up for both tokens.

Submitting jobs to use license feature locality

When LOCAL_TO is specified for a feature, job submission is simplified. To request a particular license, specify the resource usage string with the same resource name you see in `bhosts -s`. No OR rusage string is needed. For example:

bsub -Lp Lp1 -R "rusage[hspice=1]" myjob

Viewing feature locality information

When LOCAL_TO is configured for a feature in `lsf.licensescheduler`, `blinfo` shows general cluster locality information and distribution for the features.

blinfo

```
FEATURE          SERVICE_DOMAIN  TOTAL  DISTRIBUTION
hspice           SD3             15     [Lp1, 50.0%] [Lp2, 50.0%]
hspice           SD4             7      [Lp1, 50.0%] [Lp2, 50.0%]
hspice@clusterA  SD1             25     [Lp1, 50.0%] [Lp2, 50.0%]
hspice@siteB     SD2             65     [Lp1, 50.0%] [Lp2, 50.0%]
```

When LOCAL_TO is configured for a feature in `lsf.licensescheduler`, `blinfo -A` shows the feature allocation by cluster locality.

blinfo -A

```
FEATURE          PROJECT   ALLOCATION
hspice           Lp1       [clusterA, 25.0%] [clusterB, 25.0%]
                           [clusterC, 25.0%] [interactive, 25.0%])
                 Lp2       [clusterA, 50.0%] [clusterB, 50.0%])
hspice@clusterA  Lp1       [clusterA, 100.0%])
                 Lp2       [clusterA, 100.0%])
hspice@siteB     Lp1       [clusterB, 80.0%] [clusterC, 20%])
                 Lp2       [clusterB, 80.0%] [clusterC, 20%])
hspice@clusterC  Lp1       [clusterC, 60.0%] [interactive, 40.0%)
                 Lp2       [clusterC, 60.0%] [interactive, 40.0%)
                 Lp3       [clusterC, 60.0%] [interactive, 40.0%)
vcs              Lp1       [clusterA, 33.0%] [clusterB, 33.0%]
                           [interactive, 33.0%])
                 Lp2       [clusterA, 50.0%] [clusterB, 50.0%])
vcs@clusterA     Lp1       [clusterA, 100.0%])
                 Lp2       [clusterA, 100.0%])
vcs@siteB        Lp1       [clusterB, 80.0%] [clusterC, 20%])
                 Lp2       [clusterB, 80.0%] [clusterC, 20%])
vcs@clusterC     Lp1       [clusterC, 60.0%] [interactive, 40.0%)
                 Lp2       [clusterC, 60.0%] [interactive, 40.0%)
                 Lp3       [clusterC, 60.0%] [interactive, 40.0%)
```

When LOCAL_TO is configured for a feature, `blinfo -C` shows the cluster locality information for the features.

blinfo -C

```
NAME: hspice     FLEX_NAME: hspice
 CLUSTER_NAME    FEATURE          SERVICE_DOMAINS
 clusterA        hspice           SD3 SD4
                 hspice@clusterA  SD1
 clusterB        hspice           SD3 SD4
                 hspice@siteB     SD3
 clusterC        hspice           SD3 SD4
                 hspice@siteB     SD3
                 hspice@clusterC  SD5

NAME: vcs        FLEX_NAME: VCS_Runtime
 CLUSTER_NAME    FEATURE          SERVICE_DOMAINS
 clusterA        vcs              SD3 SD4
                 vcs@clusterA     SD1
 clusterB        vcs              SD3 SD4
                 vcs@siteB        SD3
 clusterC        vcs              SD3 SD4
                 vcs@siteB        SD3
                 vcs@clusterC     SD5
```

blusers

When LOCAL_TO is configured for a feature in `lsf.licensescheduler`, `blusers` shows the cluster locality information for the features.

blusers

```
FEATURE          SERVICE_DOMAIN  USER     HOST       NLICS    NTASKS
hspice@clusterA  SD1             user1    host1      1        1
hspice@siteB     SD2             user2    host2      1        1
```

blstat

When LOCAL_TO is configured for a feature in lsf.licensescheduler, `blstat` shows the cluster locality information for the features.

With the group distribution configuration:

blstat

```
FEATURE: hspice
 SERVICE_DOMAIN: SD3 SD4
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 22   OTHERS: 0
```

PROJECT SHARE OWN INUSE RESERVE FREE DEMAND

Lp1 50.0 % 0 0 0 11 0

Lp2 50.0 % 0 0 0 11 0

```
FEATURE: hspice@clusterA
 SERVICE_DOMAIN: SD1
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 25   OTHERS: 0
```

PROJECT SHARE OWN INUSE RESERVE FREE DEMAND

Lp1 50.0 % 0 0 0 12 0

Lp2 50.0 % 0 0 0 13 0

```
FEATURE: hspice@siteB
 SERVICE_DOMAIN: SD2
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 65   OTHERS: 0
```

PROJECT SHARE OWN INUSE RESERVE FREE DEMAND

Lp1 50.0 % 0 0 0 32 0

Lp2 50.0 % 0 0 0 33 0

bhosts -s

When LOCAL_TO is configured for a feature in `lsf.licensescheduler`, `bhosts -s` shows different resource information depending on the cluster locality of the features.

From clusterA:

bhosts -s

```
RESOURCE                TOTAL       RESERVED        LOCATION
hspice                  36.0        0.0             host1
```

From clusterB in siteB:

bhosts -s

```
RESOURCE                TOTAL       RESERVED        LOCATION
hspice                  76.0        0.0             host2
```

How LOCAL_TO works with the other feature section parameters

The following table shows various combinations of LOCAL_TO and other feature section parameters:

	NAME	FLEX_NAME
1	AppX	-
2	AppZ201	201-AppZ
3	AppB_v1	AppB

1. The feature name is same as the real FLEXnet name. Without LOCAL_TO, the feature name AppX cannot be duplicated in feature section. Only one feature section can contain the NAME=AppX.
2. LSF does not support names that start with a number, or names containing a dash or hyphen character (-), even though they are valid FLEXnet feature names. For these feature names, you must set both NAME and FLEX_NAME in the Features section of `lsf.licensescheduler`. FLEX_NAME is the actual FLEXnet feature name, and NAME

is an arbitrary license token name you choose. Without LOCAL_TO, NAME and FLEX_NAME cannot be in more than one feature section.

3. You can define different License Scheduler tokens for the same FLEXnet feature. The the service domain names (in either the DISTRIBUTIONS line or the SERVICE_DOMAINS for group configurations) of the same FLEXnet feature in different feature sections must be exclusive. They cannot overlap.
4. When LOCAL_TO is configured for a feature, you can define different License Scheduler tokens for the same FLEXnet feature with different localities. The constraints are:
   - For the same FLEXnet feature, service domains must be exclusive.
   - The location name of LOCAL_TO defines the locality of that feature, so the name must be unique for all tokens with same FLEXnet feature.
   - You should use same location name for different FLEXnet features with the same pattern of locality, but License Scheduler will not check whether the same location name of a different feature contains the same list of clusters.

1. Features must either have a different NAME or have LOCAL_TO defined. The service domains for each LS token of same FLEXnet feature must be exclusive.

How LOCAL_TO works with ALLOCATION and ENABLE_INTERACTIVE

The LOCAL_TO parameter simplifies the ALLOCATION configuration. Most of the time you are only interested in who can participate to share a particular token. LOCAL_TO gives the equal share for all the clusters defined in LOCAL_TO and applies to all the projects. Use ALLOCATION to fine tune the shares for individual projects between different clusters:

- Except for the keyword interactive, all the cluster names defined in ALLOCATION must also be defined in the LOCAL_TO parameter.
- The global parameter ENABLE_INTERACTIVE and ALLOCATION with interactive share defined works same as before. If ALLOCATION is configured, it ignores the global setting of the ENABLE_INTERACTIVE parameter.
- If ALLOCATION is not defined, but LOCAL_TO is defined, the default value for ALLOCATION will be equal shares for all the clusters defined in LOCAL_TO parameter. This applies to all license projects defined in DISTRIBUTION or GROUP_DISTRIBUTION.
- If both ALLOCATION and LOCAL_TO are defined, ALLOCATION parameter can be used to fine tune the shares between the clusters for different projects.

The following table shows example configurations with two clusters and 12 hspice licenses distributed as follows:

```
DISTRIBUTION = LanServer (Lp1 1 Lp2 1)
```

ENABLE_INTERACTIVE	LOCAL_TO	ALLOCATION
No	SiteA(clusterA interactive)	—
No	clusterA	Lp1(clusterA 0 clusterB 1)
No	clusterA	Lp1(clusterA 1)\ Lp2(clusterA 1)

About interactive taskman jobs

The License Scheduler command `taskman` is a job starter for interactive jobs to use License Scheduler without `bsub`. `taskman` checks out a license token and manages interactive UNIX applications. If LOCAL_TO is specified for a feature,

`taskman` jobs need to specify feature names with locality information similar to submission with `bsub`. You need to know which token can be used from the location where task is going to run. For example:

```
taskman -Lp P1 -R "rusage[hspice@siteB=1]" myjob
taskman -Lp P1 -R "rusage[hspice=1]" myjob
taskman -Lp P1 -R "rusage[hspice@clusterA=1]" myjob
```

User authentication

When a user claims a job belongs a project, License Scheduler checks if this user belongs to this project, since projects assign fairshare priority, and preemption is based on ownership. When users submit jobs to license projects they do not belong to, the request is refused, or the job gets put in a "default" bucket with a low number of shares or no shares at all.

Administrators can control who can run what project. By default, such authentication is not enabled for compatibility with the previous versions of License Scheduler.

When enabled, user authentication has the following behavior:

- If the user belongs to the project, allow the license request
- If the user does not belong to the project or the project does not match any projects in the configuration, reject the request
- If a default project is configured in the LS user authentication configuration file `ls.users`, change the project to default and allow the license request
- If the project equals default, no authentication is needed, allow the request

Enable user authentication

- For LSF jobs, configure LSF to use the authentication esub (`esub.ls_auth`). In `lsf.conf`, add the line:

```
LSB_ESUB_METHOD=ls_auth
```

- For taskman jobs, define AUTH=Y in `lsf.licensescheduler`.
- Configure users and their associated projects in the `LSF_CONFDIR/ls.users` file.

Users must make sure projects configured in `ls.users`, including the default project, conform to `lsf.licensescheduler` configuration.

Sample ls.users file

```
# The format is same as /etc/group
#
# client_name1:::user1,user2,user3
#
# Each line represents client, the client name is followed by
# a colon.

Project1:::user1,user2
Project2:::user1,user2,user3
default:::
```

Time syntax and configuration

Using automatic time-based configuration

Variable time-based configuration is used to automatically change LSF License Scheduler configuration in `lsf.licensescheduler` based on time windows. For example, if you have design centers in remote locations, one use of time-based configuration is to switch ownership of license tokens based on local time of day.

You define automatic configuration changes in `lsf.licensescheduler` by using if-else constructs and time expressions. After you change the files, reconfigure the cluster with the `bladmin reconfig` command.

The expressions are evaluated by LSF License Scheduler every 10 minutes based on `bld` start time. When an expression evaluates true, License Scheduler dynamically changes the configuration based on the associated configuration statements and restarts `bld`

In the following examples, the #if, #else, #endif keywords are not interpreted as comments by License Scheduler, but as if-else constructs.

Specifying time values

To specify a time value, a specific point in time, specify at least the hour. Day and minutes are optional.

Time value syntax

```
time = hour | hour:minute | day:hour:minute
```

hour

integer from 0 to 23, representing the hour of the day.

minute

integer from 0 to 59, representing the minute of the hour.

If you do not specify the minute, License Scheduler assumes the first minute of the hour (:00).

day

integer from 0 to 7, representing the day of the week, where 0 represents every day, 1 represents Monday, and 7 represents Sunday.

If you do not specify the day, License Scheduler assumes every day. If you do specify the day, you must also specify the minute.

Specifying time windows

To specify a time window, specify two time values separated by a hyphen (-), with no space in between.

```
time_window = time1-time2
```

time1 is the start of the window and *time2* is the end of the window. Both time values must use the same syntax. Use one of the following ways to specify a time window:

- *hour*-*hour*
- *hour*:*minute*-*hour*:*minute*
- *day*:*hour*:*minute*-*day*:*hour*:*minute*

Examples of time windows

Daily window

To specify a daily window omit the day field from the time window. Use either the hour-hour or hour:minute-hour:minute format. For example, to specify a daily 8:30 a.m. to 6:30 p.m window:

```
8:30-18:30
```

Overnight window

To specify an overnight window make *time1* greater than *time2*. For example, to specify 6:30 p.m. to 8:30 a.m. the following day:

```
18:30-8:30
```

Weekend window

To specify a weekend window use the day field. For example, to specify Friday at 6:30 p.m to Monday at 8:30 a.m.:

```
5:18:30-1:8:30
```

Specifying time expressions

Time expressions use time windows to specify when to change configurations.

Time expression syntax

A time expression is made up of the time keyword followed by one or more space-separated time windows enclosed in parenthesis. Time expressions can be combined using the &&, ||, and ! logical operators.

The syntax for a time expression is:

```
expression = time(time_window[ time_window ...])
             | expression && expression
             | expression || expression
             | !expression
```

Example

Both of the following expressions specify weekends (Friday evening at 6:30 p.m. until Monday morning at 8:30 a.m.) and nights (8:00 p.m. to 8:30 a.m. daily).

```
time(5:18:30-1:8:30 20:00-8:30)
```

```
time(5:18:30-1:8:30) || time(20:00-8:30)
```

Creating if-else constructs

The if-else construct can express single decisions and multi-way decisions by including elif statements in the construct.

If-else

The syntax for constructing if-else expressions is:

```
#if time(expression) statement #else statement #endif
```

The #endif part is mandatory and the #else part is optional.

elif

The #elif expressions are evaluated in order. If any expression is true, the associated statement is used, and this terminates the whole chain.

The #else part handles the default case where no other conditions are satisfied.

When you use #elif, the #else and #endif parts are required.

```
#if time(expression)
statement
#elif time(expression)
statement
#elif time(expression)
statement
#else
statement
#endif
```

Verify configuration

Use the following LSF commands to verify configuration:

- bladmin ckconfig
- blinfo
- blstat

Examples

Flat project configuration

```
Begin Feature
NAME = f1
#if time(5:16:30-1:8:30 20:00-8:30)
DISTRIBUTION=Lan(P1 2/5  P2 1)
#elif time(3:8:30-3:18:30)
DISTRIBUTION=Lan(P3 1)
#else
DISTRIBUTION=Lan(P1 1 P2 2/5)
#endif
End Feature
```

Hierarchical project configuration

```
#
# ProjectGroup section
#
Begin ProjectGroup
GROUP              SHARES   OWNERSHIP  LIMITS     NON_SHARED
(group1 (A B))     (1 1)    (5 -)         ()         ()
End ProjectGroup

Begin ProjectGroup
GROUP              SHARES   OWNERSHIP  LIMITS     NON_SHARED
(group2 (A B))     (1 1)    (- 5)         ()         ()
End ProjectGroup

#
# Feature section
#
Begin Feature
NAME = f1
#if time(5:16:30-1:8:30 20:00-8:30)
GROUP_DISTRIBUTION=group1
#elif time(3:8:30-3:18:30)
GROUP_DISTRIBUTION=group2
#else
GROUP_DISTRIBUTION=group2
#endif
SERVICE_DOMAINS=Lan1 Lan2
End Feature
```

Managing error logs

Error logs maintain important information about LSF License Scheduler operations. When you see any abnormal behavior in License Scheduler, you should first check the appropriate error logs to find out the cause of the problem.

Log files grow over time. These files should occasionally be cleared, either by hand or using automatic scripts.

Daemon error logs

Log files are reopened each time a message is logged, so if you rename or remove a daemon log file, the daemons will automatically create a new log file.

The License Scheduler daemons log messages when they detect problems or unusual situations. The daemons can be configured to put these messages into files. The error log file names for the LSF License Scheduler system daemons are:

- `bld.log.`*host_name*
- `blcollect.log.`*host_name*

License Scheduler daemons log error messages in different levels so that you can choose to log all messages, or only log messages that are deemed critical.

Controlling error message logging level

License Scheduler logs error messages in different levels so that you can choose to log all messages, or only log messages that are deemed critical.

Set LS_LOG_MASK in `lsf.licensescheduler` to control message logging for License Scheduler daemons. LS_LOG_MASK Specifies the logging level of error messages for LSF License Scheduler daemons.

The level specified by LS_LOG_MASK determines which messages are recorded and which are discarded. All messages logged at the specified level or higher are recorded, while lower level messages are discarded.

If LS_LOG_MASK is not defined, the value of LSF_LOG_MASK in `lsf.conf` is used. If neither LS_LOG_MASK nor LSF_LOG_MASK is defined, the default is LOG_WARNING.

The log levels in order from highest to lowest are:

- LOG_WARNING
- LOG_DEBUG
- LOG_DEBUG1
- LOG_DEBUG2
- LOG_DEBUG3

The most important License Scheduler log messages are at the LOG_WARNING level. Messages at the LOG_DEBUG level are useful for debugging.

For debugging purposes, the level LOG_DEBUG contains the fewest number of debugging messages and is used for basic debugging. The level LOG_DEBUG3 records all debugging messages, and can cause log files to grow very large; it is not often used. Most debugging is done at the level LOG_DEBUG2.

Setting bld daemon message log to debug level

The message log level for LSF daemons is set in `lsf.licensescheduler` with the parameter LS_LOG_MASK. To include debugging messages, set LS_LOG_MASK to one of:

- LOG_DEBUG
- LOG_DEBUG1
- LOG_DEBUG2
- LOG_DEBUG3

By default, LS_LOG_MASK=LOG_WARNING, and debugging messages are not displayed.

The debugging log classes for License Scheduler daemons is set in `lsf.licensescheduler` with the parameter LS_DEBUG_BLD.

The location of log files is specified with the parameter LSF_LOGDIR in `lsf.conf`.

You can use the and `bladmin` command to temporarily change the class, log file, or message log level for the `bld` daemon without changing `lsf.licensescheduler`.

How the message log level takes effect

The message log level you set will only be in effect from the time you set it until you turn it off or the daemon stops running, whichever is sooner. If the daemon is restarted, its message log level is reset back to the value of LS_LOG_MASK and the log file is stored in the directory specified by LSF_LOGDIR.

Debug command for daemons

The following command sets temporary message log level options for `bld`:

```
bladmin blbddebug [-c class_name] [-l debug_level ] [-f logfile_name] [-o]
```

If `bladmin blddebug` is used without any options, the following default values are used:

- *class_name*=0 (no additional classes are logged)
- *debug_level*=0 (LOG_DEBUG level in parameter LS_LOG_MASK)
- *logfile_name*=current LSF system log file in the LSF system log file directory, in the format *daemon_name*`.log.`*host_name*

For a detailed description of `bladmin blddebug`, see the Platform LSF Command Reference.

Debug command for blcollect

The following command sets temporary message log level options for `blcollect`:

```
bladmin blcdebug [-l debug_level ] [-f logfile_name] [-o] collector_name ... | all
```

If `bladmin blcdebug` is used without any options, the following default values are used:

- *debug_level*=0 (LOG_DEBUG level in parameter LS_LOG_MASK)
- *logfile_name*=current LSF system log file in the LSF system log file directory, in the format *daemon_name*`.log.`*host_name*
- *collector_name*=default

For a detailed description of `bladmin blcdebug`, see the Platform LSF Command Reference.

Examples

bladmin blddebug -o

Turn off temporary debug settings for `bld` on the local host (host from which the command was submitted) and reset them to the daemon starting state. The message log level is reset back to the value of LS_LOG_MASK and classes are reset to the value of LS_DEBUG_BLD. The log file is reset to the LSF system log file in the directory specified by LSF_LOGDIR in the format `bld.log.`*host_name*.

bladmin blddebug -l 1 -c "LC_TRACE LC_FLEX"

Log messages for `bld` running on the local host and set the log message level to LOG_DEBUG1. This command must be submitted from the host on which `bld` is running. The log class is LC_TRACE LC_FLEX.

bladmin blddebug -f hostB/myfolder/myfile

Log messages for `bld` to the folder `myfolder` on the server hostB, with the file name `myfile.bld.log.hostA`. The debug level is the default value, LOG_DEBUG level in parameter LS_LOG_MASK.

bladmin blcdebug -l 2

The log mask of the default collector will be changed to LOG_DEBUG2..

bladmin blcdebug -l 3 all

The log mask of all collectors is changed to LOG_DEBUG3.

blcollect log messages

Messages logged by `blcollect` include the following information:

- Time—The message log time.
- `blcollect` name—The service domain name, which is the license server host name , accessed by blcollect as defined in `lsf.licensescheduler`
- Status report for feature collection—`blcollect` information gathered successfully or not
- Detailed information—the number of tokens, the name of tokens, the license server name for license tokens collected by `blcollect`.

License maximization

The built-in functionality of License Scheduler helps ensure that your licenses are always being used efficiently. For example, if the `sbatchd` encounters any problems, the job acquires the state UNKNOWN. However, License Scheduler ensures that any in use licenses continue to be allocated, but charges them to the OTHERS category until the `sbatchd` recovers and the job state is known again.

Add project description

You can add a project description of up to 64 characters to your projects or project groups to help identify them.

1. In the `Project` section of `lsf.licensescheduler`, find the project you want to add a description for..
2. Add descriptions in the appropriate locations.
3. Save and close the file.
4. Run `badmin reconfig`.

When running `blinfo -Lp` or `blinfo -G`, any existing project descriptions display.

CHAPTER

6

Frequently Asked Questions

Submitting jobs using JSDL

Why does the license not become available after I suspend the job that is using it?

1. You submit a job to a license project that requires a specific license.
2. You submit a second higher priority job that requires this license. No licenses are free.
3. You suspend the first job to release the license it is using.
4. The second job starts but fails because it cannot obtain the required license.

Solution: Suspend the job with bkill -s TSTP. This sends a SIGTSTP signal to the job, suspends the job, and releases the license that the job was using.

Why does lsf not recognize the names of features I configured in lsf.licensescheduler?

1. You installed LSF License Scheduler.
2. You configured new license features in the License Scheduler configuration file, lsf.licensescheduler.
3. You reconfigured the License Scheduler daemon (bld) with bladmin reconfig all.
4. You submit a job that requires the new feature you configured. For example:

   ```
   bsub -R "rusage[feature_name=1]" ...
   ```
5. LSF does not recognize the feature name and you receive the following error:

   Bad resource requirement syntax. Job not submitted.

Solution: Run badmin reconfig to reconfigure mbatchd if you have added new features to lsf.licensescheduler. You must reconfigure mbatchd after you install License Scheduler and configure your license features. You must also reconfigure mbatchd each time you add a new license feature. LSF treats license tokens as LSF resources, and mbatchd must be reconfigured to recognize the resources if they change.

Note:

If you increase the number of tokens for a license feature, you do not need to reconfigure mbatchd. You only reconfigure mbatchd if you add a new license feature.

Why does blhosts -s display license tokens after you shut down the LSF License Scheduler daemon (bld)?

The License Scheduler daemon (bld) keeps a local backup database of all the license tokens in `LSB_SHAREDIR/`*`cluster_name/log_dir/`*`lsb.tokens`. This backup file provides redundancy for License Scheduler when it runs in a WAN configuration.

mbatchd can still read this file after you shut down bld.

If you do not want mbatchd to recognize the license token names, you can remove the backup file and reconfigure mbatchd by running badmin reconfig.

Why does my job submission fail when the license feature name includes numbers?

Normally, license token names should be the same as the FLEXnet feature names, as they represent the same license. However, LSF does not support names that start with a number, or names containing a dash or hyphen character (-), which may be used in the FLEXnet feature name.

Solution: Set both NAME and FLEX_NAME in the Features section of `lsf.licensescheduler`. FLEX_NAME is the actual FLEXnet feature name, and NAME is an arbitrary license token name you choose that does not start with a number or contain a dash.

Example:

```
Begin Feature
FLEX_NAME=201-AppZ
NAME=AppZ201
DISTRIBUTION=LanServer1(Lp1 1 Lp2 1)
End Feature
```

I see the following error in the bld.log: server_name file: globInit(): cannot initialize the listening TCP - 2 channel Address already in use.

1. You edit the `LSF_CONFDIR/lsf.conf` file to include a list of hosts for the LSF_LIC_SCHED_HOSTS parameter.
2. You run `lsadmin reconfig` to reconfigure the LIM.
3. You use `ps -efl | grep bld` to make sure that bld is running on the candidate hosts, but find that bld is not running on the *server_name* host.
4. You view the `bld.log.`*server_name* file and see the following error message:

   globInit(): cannot initialize the listening TCP -2 channel Address already in use

Explanation: Normally, the LSF LIM daemon starts the License Scheduler daemon (bld) automatically on startup. If you already started bld manually, the LSF LIM daemon still tries to start bld, but the port used by bld is already open.

Solution: Run `bladmin shutdown` to shut down License Scheduler. The LSF LIM starts bld automatically. Run `badmin reconfig` to reconfigure mbatchd.

Why does the job pending reason show the wrong license feature?

1. You have two License Scheduler resources: feat2 and feat3, both with value 10.
2. Submit a job with rusage[feat2=11:feat3=1]"
3. `bjobs` pending reason shows (feat3) not satisfied when it should be (feat2) not satisfied.

Reason: When scheduling a job that requests license resources, `mbatchd` sends a request to `bld` to reserve the requested licenses for the job. This will take some time. During this time, the requested license resources are not available and `mbschd` sets the pending reason.

Solution: Wait a few moments for `mbatchd` to get the license resource and passed it to `mbschd`, and run `bjobs` again to see the correct pending reason.

I see the following error when running LSF License Scheduler commands: Network I/O error with the License Scheduler server

1. You are running LSF License Scheduler on a Windows client.
2. You run an LSF License Scheduler command and see the following error message:

Network I/O error with the License Scheduler Server.

3. You see the following message in the mbatchd log file:

   callglb(): cc -1Failed in an LSF library call: Failed in sending/receiving a message: error 0: The operation completed successfully

Reason: The master host does not recognise your Windows client when you try to issue LSF License Scheduler commands to `bld` because you did not specify your Windows client host name and IP address in the `/etc/hosts` file on the master host.

Solution: Add your Windows client host name and IP address to the `/etc/hosts` file on the master host.

Index

F

G

H

I

J

L

M

N

O