Platform LSF
Version 9 Release 1.2

Using IBM Platform License Scheduler

SC27-5308-02

Platform LSF
Version 9 Release 1.2

Using IBM Platform License Scheduler

SC27-5308-02

Note
Before using this information and the product it supports, read the information in "Notices" on page 175.

First edition

This edition applies to version 9, release 1, modification 2 of IBM Platform License Scheduler (product number 5725G82) and to all subsequent releases and modifications until otherwise indicated in new editions.

Significant changes or additions to the text and illustrations are indicated by a vertical line (|) to the left of the change.

If you find an error in any Platform Computing documentation, or you have a suggestion for improving it, please let us know. Send your suggestions, comments and questions to the following email address:

pccdoc@ca.ibm.com

Be sure include the publication title and order number, and, if applicable, the specific location of the information about which you have comments (for example, a page number or a browser URL). When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1. Introduction

Overview

Applying policies to how licenses are shared can have significant benefits both in terms of productivity and cost savings.

Share licenses more easily

IBM Platform License Scheduler (License Scheduler) makes it easy to share licenses between project teams and departments within the same design center or around the globe. With tools to allocate and monitor licenses, license owners can share licenses not in use, while still ensuring immediate access to licenses when needed. With more effective sharing, all users perceive a larger pool of licenses.

Ensure appropriate license allocation

License Scheduler enables flexible, hierarchical sharing policies that reflect the needs of the business. During quiet periods, when licenses are not in contention, licenses can be allocated to anyone who needs them keeping utilization and throughput high. During busy periods, the supply of licenses can be allocated based on policy toward the most time or revenue critical projects.

Improve service levels and productivity

By ensuring access to a minimum share of licenses and enabling allocation to flex between clusters that are based on need, licenses are more readily available and jobs are less likely to pend in queues that are awaiting license resources. This translates into reduced wait times and better productivity, and contributes to a faster, more efficient design environment.

Reduce or avoid cost

By being able to allocate scarce licenses to the most critical projects, and by being able to analyze license usage in the context of cluster resources, users and projects, planners are better able to find and remove bottlenecks, making their existing licenses more productive. With better visibility to how licenses are being used, they can plan license requirements more effectively ultimately helping to contain costs and boost productivity.

License Scheduler controls the software license sharing in your organization. License Scheduler works with FlexNet™ products to control and monitor license usage.

Differences between License Scheduler editions

License Scheduler is available in two editions: Basic Edition and Standard Edition.

License Scheduler Basic Edition is included with LSF and is not intended to apply policies on how licenses are shared between clusters or projects. Rather, License Scheduler Basic Edition is intended to replace an external load information manager (**elim**) to collect external load indicies for licenses managed by FlexNet. To replace this **elim**, License Scheduler Basic Edition limits the license use of jobs

of a single cluster to prevent overuse of the licenses and tracks license use of individual jobs by matching license checkouts to these jobs. License Scheduler Basic Edition provides cluster mode features distributed to a single cluster with one service domain per license feature.

License Scheduler Standard Edition not only provides cluster mode features for a single cluster, but also provides full License Scheduler functionality, including support for all modes (cluster mode, project mode, and fast dispatch project mode), multiple clusters, features and feature groups, multiple service domains per license feature, and **taskman** jobs.

Important: A License Scheduler entitlement file (`ls.entitlement`) is now required to run License Scheduler Standard Edition. Copy the entitlement file (`ls.entitlement`) to the `$LSF_ENVDIR` directory before starting License Scheduler to run as Standard Edition.

To install and run License Scheduler Basic Edition, download and install the License Scheduler packages as described in "Install License Scheduler" on page 7, but follow any specific steps for installing and configuring License Scheduler Basic Edition instead of Standard Edition.

License Scheduler Standard Edition is assumed in all License Scheduler documentation unless it is explicitly stated as describing License Scheduler Basic Edition.

Glossary

blcollect

The License Scheduler daemon that queries FlexNet licensing software for license usage. **blcollect** collects information from **lmstat**.

You can spread the load of license collection by running the license information collection daemon on multiple UNIX hosts.

Also called the collector.

bld

The License Scheduler batch daemon.

cluster mode

License tokens are allocated to clusters by License Scheduler, and job scheduling within each cluster is managed by the local **mbatchd**. Cluster mode is only available for License Scheduler version 8.0 and later.

Each license feature can use either cluster mode or project mode, but not both.

lmgrd

The main FlexNet licensing daemon. Usually grouped into service domains inside License Scheduler.

project mode

License tokens are allocated to projects by License Scheduler, and job scheduling for license projects takes place across clusters that follow the license distribution policy that is configured for each project. Corresponds to License Scheduler version 7.0.5 and earlier.

Each license feature can use either cluster mode or project mode, but not both.

service domain

A group of one or more FlexNet license servers.

You configure the service domain with the license server names and port numbers that serve licenses to a network.

taskman job

A job that is run by theIBM Platform LSF (LSF) Task Manager (**taskman**) tool outside of LSF, but is scheduled by License Scheduler.

token

One license token represents one actual license, and is used by License Scheduler to track license use and determine which job to dispatch next.

License Scheduler manages license tokens instead of controlling the licenses directly. After they reserve license tokens, 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 FlexNet, so if a token is not available, the job is not dispatched.

Architecture

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

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

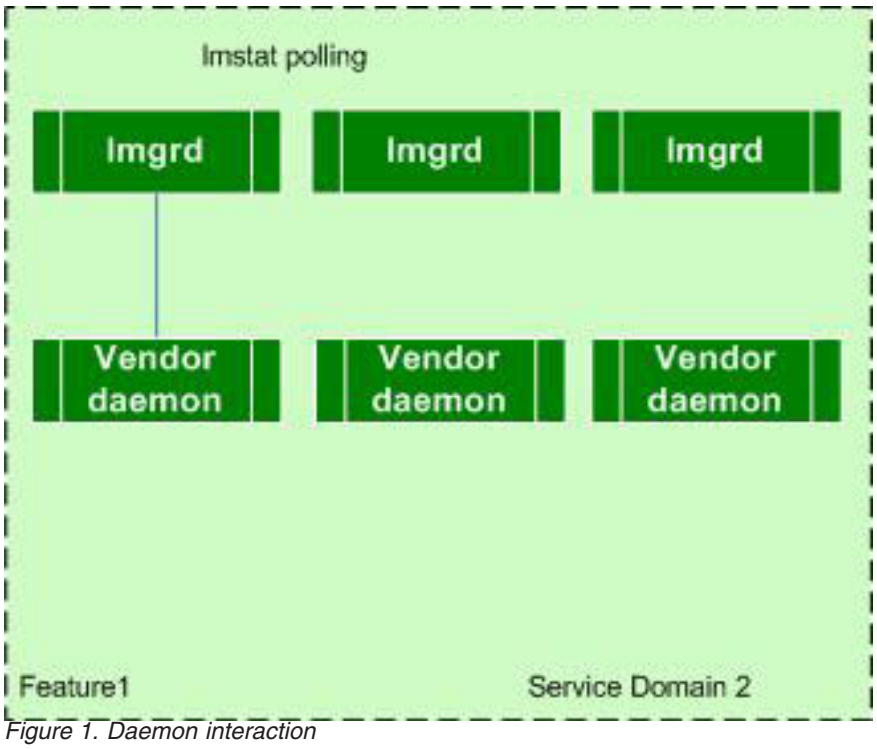


Figure 1. Daemon interaction

How scheduling policies work

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

The basic LSF scheduling comes first when starting a job. License Scheduler has no influence on job scheduling priority. Jobs are considered for dispatch according to the prioritization policies configured in each cluster.

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

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 License Scheduler.

When the mbatchd is offline

When a cluster is running, the **mbatchd** maintains a TCP connection to **bld**. When the cluster is disconnected (such as when the cluster goes down or is restarted), the **bld** removes all information about jobs in the cluster. License Scheduler considers licenses that are checked out by jobs in a disconnected cluster to be non-LSF use of licenses.

When **mbatchd** comes back online, the **bld** immediately receives updated information about the number of tokens that are currently distributed to the cluster.

When the bld is offline

If the **mbatchd** loses the connection with the **bld**, the **mbatchd** cannot get **bld**'s token distribution decisions to update its own.

However, because the **mbatchd** logs token status every minute in *$LSF_TOP*/work/data/*featureName*.*ServiceDomainName*.dat file, if the connection is lost, the **mbatchd** uses the last logged information to schedule jobs.

```
f3.LanServer1.dat
# f3 LanServer1 3 2
# p1 50 p2 50
  12/3         14:20:38        2    0    2    0         1    0    1    0
  12/3         14:21:39        2    0    2    0         1    0    1    0
  12/3         14:22:40        3    3    0    0         0    0    0    0
  12/3         14:23:41        3    3    0    0         0    0    0    0
  12/3         14:24:42        1    0    1    0         2    0    2    0
  12/3         14:25:43        1    0    1    0         2    0    2    0
  12/3         14:26:44        1    0    1    0         2    0    2    0
  12/3         14:27:55        1    0    1    0         2    0    2    0
```

f3 on LanServer1 has three tokens and two projects. Projects p1 and p2 share licenses 50:50.

At 14:27:55, the **bld** dispatched one token to p1, which has 0 in use, 1 free, 0 reserve. At the same time, the **bld** dispatched two tokens to p2, which has 0 in use, 2 free, and 0 reserve.

The **mbatchd** continues to schedule jobs that are based on the token distribution that is logged at 14:27:55 until the connection with the **bld** is re-established.

Chapter 2. Installing and starting License Scheduler

Install License Scheduler

1. Perform the pre-installation steps.
2. Choose an installation plan:
   - UNIX: License Scheduler manages licenses for jobs that run through LSF and through applications other than LSF.
   - Windows, in a mixed cluster:

     A Licenser Scheduler installation requires UNIX hosts to run the **bld**. Windows hosts in a mixed cluster can run License Scheduler commands.

     When you have License Scheduler UNIX hosts working with LSF, run License Scheduler on Windows hosts as well.

Before you install

IBM Platform LSF ("LSF") must be installed and running before you install License Scheduler.

Log on to any LSF host as root and use **lsid** to make sure that the cluster is running. If you see the message "Cannot open `lsf.conf` file", verify that the $LSF_ENVDIR environment variable is 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 License Scheduler setup script does

- Finds the appropriate `lsf.conf` for the running cluster.
- Copies the License Scheduler files to your LSF directories:
  - **$LSF_ENVDIR**:
    - `lsf.licensescheduler`
    - `ls.users`
  - **$LSF_SERVERDIR**:
    - **bld**
    - **blcollect**
    - **globauth**
    - **esub.ls_auth**
  - **$LSF_BINDIR**:
    - **blstat**
    - **blcstat**
    - **blusers**
    - **blinfo**
    - **bladmin**
    - **blstartup**
    - **blhosts**

    - **blkill**
    - **bltasks**
    - **blparams**
    - **taskman**
  - **$LSF_LIBDIR**:
    - **libglb.a**
    - **libglb.so**
    - liblic.so
  - **$LSF_MANDIR**: various man pages
- 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 License Scheduler administrators list in the `lsf.licensescheduler` file.
- Configures LSF to use License Scheduler.

Install License Scheduler with LSF (UNIX)

You must have write access to the LSF_TOP directories.

1. Log on as root to the installation file server host.
2. Download, uncompress, and extract the License Scheduler packages for the platforms you need.

   For example, for x86 64-bit systems that run Linux kernel 2.6.x and compiled with glibc 2.3.x:

   ```
   ftp> get lsf9.1.2_licsched_lnx26-libc23-x64.tar.Z
   ```

   Make sure that you download the 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 lsf9.1.2_licsched_lnx26-libc23-x64.tar.Z | tar xvf -
   ```
4. Change to the extracted distribution directory.

   For example:

   ```
   # cd lsf9.1.2_licsched_linux2.6-glibc2.3-x86_64
   ```
5. Edit `./setup.config` to specify the installation variables you want.

   Uncomment the options that you want in the template file, and replace the example values with your own settings.

   Tip: The sample values in the `setup.config` template file are examples only. They are not default installation values.
6. Run the **setup** script as root:

   ```
   # ./setup
   ```
7. 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.
8. 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.

9. 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 necessary.
10. If you are installing License Scheduler Standard Edition, copy the License Scheduler entitlement file (`ls.entitlement`) to the `$LSF_ENVDIR` directory.

    If you do not copy the entitlement file to `$LSF_ENVDIR` before starting License Scheduler, License Scheduler runs as Basic Edition.

If you are installing License Scheduler Basic Edition, configure License Scheduler Basic Edition and LSF as described in "Configure License Scheduler Basic Edition" on page 10.

Install License Scheduler on Windows

You can install License Scheduler on Windows hosts when your cluster includes both Windows and UNIX hosts.

The License Scheduler Windows Client package includes:

- README
- Commands:
  - **`blstat.exe`**
  - **`blcstat.exe`**
  - **`blinfo.exe`**
  - **`blusers.exe`**
  - **`bladmin.exe`**
  - **`blhosts.exe`**
  - **`blkill.exe`**
  - **`bltasks.exe`**
  - **`blparams.exe`**
  - **`taskman.exe`**
- `lsf.licensescheduler`: License Scheduler configuration file
- `lsf.conf`: LSF configuration file

Install License Scheduler with LSF (Windows)

You must already have LSF installed on all Windows hosts you intend to install License Scheduler on.

This installation option means that License Scheduler manages licenses for jobs that are submitted through LSF and through any other applications.

Install License Scheduler on Windows hosts only when your LSF cluster includes both UNIX and Windows hosts.

1. Download the License Scheduler Client for Windows package.
2. Copy all commands to **`$LSF_BINDIR`** (the `bin` subdirectory in your LSF installation directory) on your Windows hosts.
3. Copy `lsf.licensescheduler` to **`$LSF_ENVDIR`**.
4. Edit `lsf.licensescheduler` to suit your License Scheduler Master host configuration.

Troubleshoot

1. If you receive the following message, configure your Windows host name and IP address in the /etc/hosts file on the master host:

   ```
   Failed in an LSF library call: Failed in sending/receiving a message:
   error 0: The operation completed successfully.
   ```

2. To enable the **blhosts** command, make sure that your Windows host can resolve the master host IP address correctly.

Configure License Scheduler Basic Edition

Configure LSF and License Scheduler Basic Edition.

Configuring License Scheduler Basic Edition and LSF

Configure LSF to use License Scheduler Basic Edition as a replacement for an **elim** to collect external load indices where the external resources are licenses managed by FlexNet.

The following example assumes that LSF cluster named cluster1 uses an **elim** for a license resource named f1.

1. In the LSF environment, disable the existing **elim** for the license resource by removing the license feature configuration from the lsf.shared and lsf.cluster.*cluster_name* files.

   For example, remove the configuration for f1 from the lsf.shared and lsf.cluster.*cluster_name* files.

2. Configure the lsf.licenscheduler file with the appropriate hosts and the license feature.

   For example, configure the following sections in lsf.licenscheduler:

   ```
   Begin Parameters
   PORT=1700
   HOSTS=hostA
   ADMIN=lsadmin
   LM_STAT_INTERVAL=15
   LMSTAT_PATH=/usr/bin
   End Parameters

   Begin Clusters
   CLUSTERS
   cluster1
   End Clusters

   Begin ServiceDomain
   NAME=LanServer
   LIC_SERVERS=((19999@hostA))
   End ServiceDomain

   Begin Feature
   NAME=f1
   CLUSTER_MODE=Y
   CLUSTER_DISTRIBUTION=LanServer(cluster1)
   End Feature
   ```

3. Start License Scheduler and LSF.

   For more details, refer to “Start License Scheduler” on page 12.

From LSF, use **bsub** to submit a job without a duration requesting the f1 resource. For example,

```
bsub -R "rusage[f1=1]" myjob -f "f1 1" -c 19999@hostA -t 20000
```

Upgrading from License Scheduler Basic Edition to Standard Edition

If you use License Scheduler Basic Edition and wish to upgrade to License Scheduler Standard Edition, obtain the License Scheduler entitlement file, then upgrade License Scheduler as follows:

1. Copy the License Scheduler entitlement file (`ls.entitlement`) to the `LSF_ENVDIR` directory.
2. Restart License Scheduler.

   ```
   bladmin reconfig
   ```

3. Restart the **mbatchd** on the LSF master host.

   ```
   badmin mbdrestart
   ```

Supported parameters for License Scheduler Basic Edition

The following is a list of specific `lsf.licensescheduler` parameters that License Scheduler Basic Edition supports:

- Parameters section:
  - ADMIN
  - CLUSTER_MODE (License Scheduler Basic Edition only supports `CLUSTER_MODE=Y`)
  - HEARTBEAT_INTERVAL
  - HEARTBEAT_TIMEOUT
  - HOSTS
  - LIB_CONNTIMEOUT
  - LIB_RECVTIMEOUT
  - LM_STAT_INTERVAL
  - LM_STAT_TIMEOUT
  - LMSTAT_PATH
  - LOG_EVENT
  - LOG_INTERVAL
  - LS_DEBUG_BLC
  - LS_DEBUG_BLD
  - LS_DEBUG_CMD
  - LS_LOG_MASK
  - LS_MAX_STREAM_FILE_NUMBER
  - LS_MAX_STREAM_SIZE
  - LS_STREAM_SIZE
  - LS_STREAM_FILE
  - MBD_HEARTBEAT_INTERVAL
  - MBD_REFRESH_INTERVAL
  - STANDBY_CONNTIMEOUT
  - BLC_HEARTBEAT_FACTOR
- Clusters section:
  - CLUSTERS (one cluster only, License Scheduler Basic Edition ignores additional clusters)
- ServiceDomain section (one ServiceDomain section per license feature only, License Scheduler Basic Edition ignores additional ServiceDomain sections in the same license feature):

  - NAME
  - LIC_SERVERS
  - LM_STAT_INTERVAL
  - LM_STAT_TIMEOUT
  - LIC_COLLECTOR
- Feature section:
  - NAME
  - CLUSTER_MODE (Optional. This parameter may be specified in the Parameters section instead, but License Scheduler Basic Edition only supports `CLUSTER_MODE=Y`)
  - FLEX_NAME (Optional. License Scheduler Basic Edition does not support the specification of multiple FlexNet feature names to combine into a single alias)
  - CLUSTER_DISTRIBUTION (License Scheduler Basic Edition supports a single cluster with a single service domain only, and ignores any additional clusters or service domains).
  - LIC_COLLECTOR

Tip: A specific `lsf.licensescheduler` configuration template for License Scheduler Basic Edition is available and contains specifications for all supported parameters. This file is named `lsf.licensescheduler.basic` and is included in the License Scheduler installation package. License Scheduler uses the Standard Edition configuration file by default, but License Scheduler Basic Edition ignores unsupported Standard Edition parameters with a warning message. To ensure that License Scheduler Basic Edition uses only supported parameters and to prevent the logging of the warning messages, back up the `lsf.licensescheduler` configuration file, then move the `lsf.licensescheduler.basic` file to the `$LSF_ENVDIR` directory and rename it to `lsf.licensescheduler`.

Start 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 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**.
7. If you specified a **LIC_COLLECTOR** name in your service domains, start each license collector manually:

   `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.

   - *lic_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 a License Scheduler service domain, the master **bld** host starts a default **blcollect**.

LSF parameters in License Scheduler

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 compatibility with earlier versions only.

- **LSF_LIC_SCHED_PREEMPT_REQUEUE**: Requeues a job whose license is preempted by License Scheduler. The job is killed and requeued instead of suspended.
- **LSF_LIC_SCHED_PREEMPT_SLOT_RELEASE**: Releases memory and slot resources of a License Scheduler job that is suspended. These resources are only available to pending License Scheduler jobs that request at least one license that is the same as the suspended job.

  Job slots are released by default, but memory resources are also released if memory preemption is enabled (that is, `PREEMPTABLE_RESOURCES = mem` is set in `lsb.params`).
- **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.

LSF parameters used by License Scheduler

- **LSB_SHAREDIR**: Directory where the job history and accounting logs are kept for each cluster
- **LSF_LOG_MASK**: Logging level of error messages for LSF daemons
- **LSF_LOGDIR**: LSF system log file directory

About submitting jobs

When you submit an LSF job, you must reserve the license with the resource requirement usage section (**bsub -R "rusage..."** option).

Tip:

You cannot successfully reserve a license by running **bsub -R "select"**.

- Specify the license token name (same as specifying a shared resource).
- If you use project mode, specify a license project name with the **bsub -Lp** option.

  If you also have LSF_LIC_SCHED_STRICT_PROJECT_NAME=y in `lsf.conf` and without configuring a default project for the required feature, the job is rejected.

  Tip:

  Use the **blstat** command to view information about the `default` license project.
- Update resource requirements.

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

Examples:

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

This command submits a job named `myjob` to license project `Lp1` and requests one `AppB` license

```
bsub -R "rusage[AppC=1]" myjob
```

This command submits a job named `myjob` and requests one `AppC` license.

After configuration changes

If you make configuration changes to License Scheduler, you must reconfigure License Scheduler to apply the changes. If you make configuration changes to LSF, you must also reconfigure LSF.

1. Run `bld -C` to test for configuration errors.
2. Run `bladmin reconfig all`.
3. If you changed `lsf.conf` or other LSF configuration files, run `badmin mbdrestart` and `lsadmin reconfig`.

   Note:

After certain License Scheduler configuration changes, you must run **badmin mbdrestart** for the changes to take effect. The following configuration changes require you to run **badmin mbdrestart**:

- Project changes, additions, or deletions
- Feature changes, additions, or deletions, including mode changes
- Cluster locations changes

You must also run **lsadmin reconfig** for any changes to the LIM to take effect (for example, if you changed LSF_LIC_SCHED_HOSTS).

Add a cluster to License Scheduler

You must be a License Scheduler administrator.

You can add a new cluster to an existing License Scheduler implementation.

1. Download the License Scheduler package.

   Note: Acquire the same version of master **bld** binary files and other architectures that are used in existing member clusters.
2. Install the License Scheduler package on the new cluster.
3. Use an existing `lsf.licensescheduler` from *$LSF_ENVDIR* of another cluster with the same **bld** master.
4. Add new cluster name to the Clusters section of `lsf.licensescheduler`.
5. Add or modify license distribution policies that are defined in `lsf.licensescheduler`.
6. Maintain one central `lsf.licensescheduler` file and have all the clusters access it.

   Remember:

   The `lsf.licensescheduler` file in each cluster must be identical.

   You can accomplish this using either of the following methods:

   - Create a symbolic link from each cluster's $LSF_ENVDIR to the central `lsf.licensescheduler` file.
   - Use a CRON-based synchronization script to synchronize the changes that are made from the central `lsf.licensescheduler` file to the corresponding `lsf.licensescheduler` files in all the clusters.
7. Check that there is no firewall or network issue with communication from the PORT in the `lsf.licensescheduler` file
8. Run `bladmin reconfig` on all hosts where **bld** is running.
9. On the newly added cluster, run `lsadmin limrestart` and then `badmin mbdrestart`.

Configure 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 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 that are 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 that are 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 apply your changes.

Upgrade License Scheduler

You must have License Scheduler installed before you can upgrade. You must be a cluster administrator.

You can upgrade to a new version of License Scheduler without uninstalling and reinstalling.

1. Download the new version of the License Scheduler distribution tar files.
2. Deactivate all queues.

   Deactivating all queues pends any running jobs and prevents new jobs from being dispatched.

   ```
   badmin qinact all
   ```
3. If you have the IBM Platform Application Center installed, shut it down.

   ```
   pmcadmin stop
   ```
4. Back up your existing **LSF_CONFDIR**, **LSB_CONFDIR**, and **LSB_SHAREDIR** according to the procedures at your site.
5. Optional. To use the fast dispatch project mode in License Scheduler, upgrade LSF to version higher than 9.1.1. After completing the upgrade, restart LSF.
6. Use the setup script to upgrade License Scheduler.
   a. Source `cshrc.lsf` or `profile.lsf` in old LSF cluster.
   b. Navigate to the location of your tar files and extract.
   c. Run the setup script.
7. If you are installing License Scheduler Standard Edition, copy the License Scheduler entitlement file (`ls.entitlement`) to the $LSF_ENVDIR directory. If you do not copy the entitlement file to $LSF_ENVDIR before starting License Scheduler, License Scheduler runs as Basic Edition.
8. Start License Scheduler.
   a. Source `cshrc.lsf` or `profile.lsf`.
   b. Run `bladmin reconfig`.
   c. Run `ps -ef` to make sure the **bld** is running on the candidate hosts.
   d. Run `badmin mbdrestart`.
   e. Activate the queues.

      ```
      badmin qact all
      ```
9. If you have the IBM Platform Application Center installed, restart it.

   ```
   pmcadmin start
   ```

   Note:

IBM Platform Application Center version 8.0.1 and later displays License Scheduler workload for both project mode and cluster mode.

Firewalls

Configuration for LSF, License Scheduler, and **taskman** interoperability.

Set up firewall communication

The **mbatchd** and **bld** listening ports (inbound connections) must be open on either side of the firewall.

- **mbatchd**: Set by **LSB_MBD_PORT** in `lsf.conf`
- **bld**: Set by **PORT** in `lsf.licensescheduler`
- If a firewall is between the **mbatchd** and **bld** hosts, both listening ports must be open.
- If a firewall is between **bld** and **blcollect** hosts (for example, **blcollect** is configured to run locally on the license servers and **bld** is on the LSF master host), the **bld** listening port must be open.
- If a firewall is between **taskman** and **bld** (where jobs use **taskman** to interface with License Scheduler), the **bld** listening port must be open.

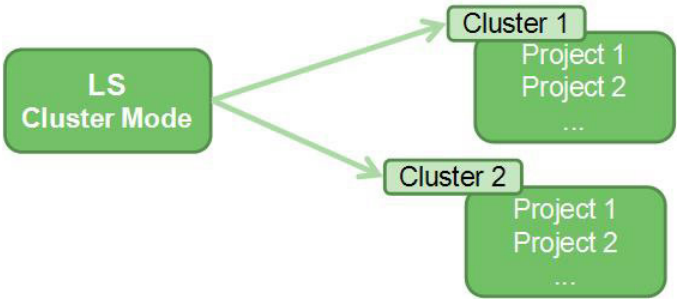
Chapter 3. License Scheduler concepts

License Scheduler modes

When you configure your installation of License Scheduler, you must choose which of project mode and cluster mode best suits your needs for each license you use. Both project mode and cluster mode can be configured in one installation, however, all different licenses that are required by a job must belong to the same mode.

cluster mode

Distributes license tokens to clusters, where LSF scheduling takes over.



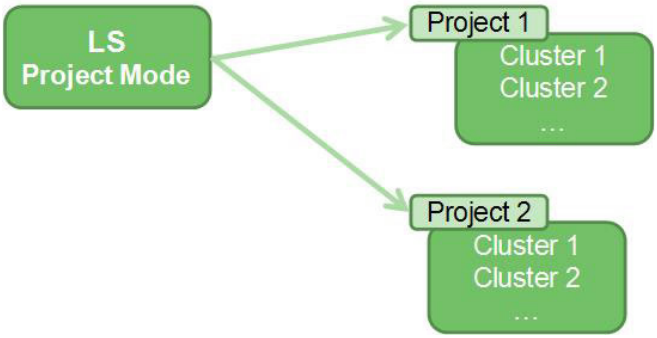
Cluster mode emphasizes high utilization of license tokens over other considerations such as ownership. License ownership and sharing can still be configured, but within each cluster instead of across multiple clusters. Preemption of jobs (and licenses) also occurs within each cluster instead of across clusters.

License tokens are reused by LSF when a job finishes, without waiting for confirmation from **lmstat** that license tokens are available and reported in the next **blcollect** cycle. This results in higher license utilization for short jobs.

Cluster mode was introduced in License Scheduler 8.0.

project mode

Distributes license token to projects configured across all clusters.



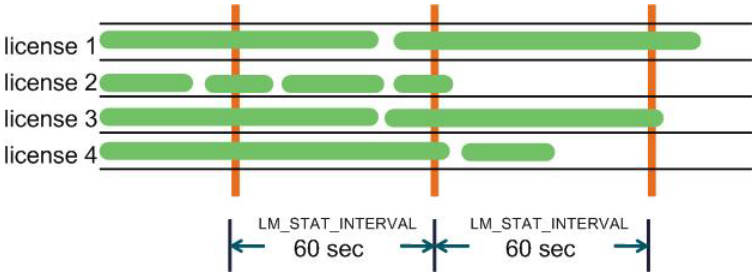
Project mode emphasizes ownership of license tokens by specific projects which span multiple clusters. When License Scheduler is running in project mode, License Scheduler checks demand from license owners across all LSF

clusters before allocating license tokens in project mode. The process of collecting and evaluating demand for all projects in all clusters slows down each scheduling cycle. License tokens are distributed in the next scheduling cycle after **lmstat** confirms license token availability.

Project mode was the only choice available before License Scheduler 8.0.

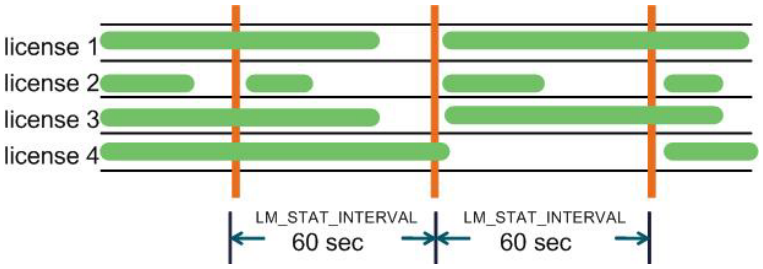
Difference between cluster mode and project mode

The following figure illustrates license utilization in cluster mode for short jobs with the corresponding **lmstat** reporting times:



In cluster mode, when one job finishes running, the next job gets its license immediately without having to wait for the next **lmstat** interval. For example, four jobs that require license 2 are able to run without waiting for **lmstat** to report token distribution.

The following figure illustrates license utilization in project mode for short jobs with the **lmstat** reporting times:



In project mode, each job must wait for **lmstat** to report token distribution before it can get a license and start running. In this example, three jobs that require license 2 are able to start within the **lmstat** intervals illustrated.

When to use cluster mode

Cluster mode is most appropriate for your needs if:

- Your primary goal is to maximize license use.
- Ownership of licenses is a secondary consideration.
- Many jobs are short relative to the **blcollect** cycle (60 seconds by default, set by **LM_STAT_INTERVAL**).

When to use project mode

Project mode is most appropriate for your needs if the following applies:

- Your primary goal is to ensure ownership of the group.
- Maximizing license use is a secondary consideration.
- Most jobs are long relative to the **blcollect** cycle (60 seconds by default, set by **LM_STAT_INTERVAL**).

Project groups

When you are configuring your installation of License Scheduler in project mode, you can choose to configure projects, or extend your project configuration further to form hierarchical 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 is applied 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.

Each feature has its own hierarchical group, but features can share hierarchy. The hierarchical scheduling is done per feature across service domains.

projects

Projects alone apply one distribution policy within one service domain. The same local distribution policy can be applied to more than one service domain, but is implemented locally.

groups of projects

Groups of projects apply one distribution policy within one service domain, but assign shares and ownership to groups of projects for greater flexibility. With group license ownership, projects trigger preemption either when the project is using fewer licenses than it owns or when the group to which the project belongs is using fewer licenses than the group owns.

project groups

Projects groups apply one distribution policy across multiple service domains following the configured hierarchical structure. You can also use project groups to apply hard limits to the number of licenses that are distributed to each project.

After configuration, the same project group hierarchy can be used for more than one feature.

When to use groups of projects

Grouping projects together in project mode is most appropriate for your needs if:

- Licenses are owned at multiple levels, for example by a department and also by projects within the department.
- License ownership is within one service domain. As for ungrouped projects, distribution policies are implemented locally for groups of projects.

When to use project groups

Extending your configuration to include project groups is most appropriate for your needs if:

- License ownership spans service domains.
- One distribution policy must be applied across several service domains.
- Project limits must be applied across clusters.

Note:

If required, use Platform LSF to configure license project limits within one LSF cluster.

Service domains in License Scheduler

A service domain is a group of one or more FlexNet license servers. License Scheduler manages the scheduling of the license tokens, but the license server actually supplies the licenses. You configure the service domain with the license server names and port numbers that serve licenses to a network.

- LAN: a service domain that serves licenses to a single cluster
- WAN: a sevice domain that serves licenses to multiple clusters

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 that is 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.

You must configure at least one service domain for License Scheduler. It groups FlexNet license server hosts that serve licenses to LSF jobs and is used when you define a policy for sharing software licenses among your projects.

If a FlexNet license server host is not part of a 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).

Service domain locality

You can use license feature locality 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 that requested the token. The LAN service domains that are used in cluster mode are configured with single-cluster locality.

Project mode

In project mode, a cluster can access the same license feature from multiple service domains.

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 for any features that cannot be shared across all the locations. This parameter avoids having to define different distribution and allocation policies for different service domains, and allows hierarchical project group configurations.

To use License Scheduler tokens in project mode, a job submission must specify the **-Lp** (license project) option. The project must be defined for the requested feature in `lsf.licensescheduler`.

Cluster mode

In cluster mode, each license feature in a cluster can access a single license feature from at most one WAN and one LAN service domain.

License Scheduler does not control application checkout behavior. If the same license is available from both the LAN and WAN service domains, License Scheduler expects jobs to try to obtain the license from the LAN first.

Parallel jobs

When LSF dispatches a parallel job, License Scheduler attempts to check out `user@host` keys in the parallel job constructed using the user name and all execution host names, and merges the corresponding checkout information on the service domain if found.

For example, in project mode, for feature `F1` with two projects (`P1` and `P2`) in service domain `sd1`, with ten tokens, a parallel job is dispatched to four execution hosts using the following command:

```
bsub -n 4 -Lp P1 -R "rusage[F1=4]" mycmd
```

The job on each execution host checks out one `F1` license from the `sd1` service domain. If the four execution hosts are `hostA`, `hostB`, `hostC`, and `hostD`, there are checkout keys for `user@hostA`, `user@hostB`, `user@hostC`, and `user@hostD`, and each entry contributes corresponds with one token checked out. These tokens all merge into data for the `P1` project in the `F1` feature. Therefore, running **blstat** displays the following information for the `F1` feature:

```
FEATURE: F1
 SERVICE_DOMAIN: LanServer
 TOTAL_INUSE: 4    TOTAL_RESERVE: 0    TOTAL_FREE: 6    OTHERS: 0
  PROJECT                SHARE     OWN       INUSE     RESERVE   FREE      DEMAND
  P1                     50.0 %    0         4         0         1         0
  P2                     50.0 %    0         0         0         5         0
```

The four checkout keys from the four execution hosts are merged into the `P1` project.

If `MERGE_BY_SERVICE_DOMAIN=Y` is defined, License Scheduler also merges multiple `user@host` data for parallel jobs across different service domains.

For example, if you have the same setup as the previous example, but with an additional service domain `sd2` also with two projects (`P1` and `P2`) and ten tokens, and you have `MERGE_BY_SERVICE_DOMAIN=Y` defined, running **blstat** displays the following information for the `F1` feature:

```
blstat
FEATURE: F1
 SERVICE_DOMAIN: sd1
 TOTAL_INUSE: 2    TOTAL_RESERVE: 0    TOTAL_FREE: 8    OTHERS: 0
  PROJECT                SHARE     OWN       INUSE     RESERVE   FREE      DEMAND
  P1                     50.0 %    0         2         0         3         0
  P2                     50.0 %    0         0         0         5         0
 SERVICE_DOMAIN: sd2
 TOTAL_INUSE: 2    TOTAL_RESERVE: 0    TOTAL_FREE: 8    OTHERS: 0
  PROJECT                SHARE     OWN       INUSE     RESERVE   FREE      DEMAND
  P1                     50.0 %    0         2         0         3         0
  P2                     50.0 %    0         0         0         5         0
```

Two checkout keys are merged into the `P1` project in the `sd1` domain, while two checkout keys are merged into the `P1` project under the `sd2` domain.

If `CHECKOUT_FROM_FIRST_HOST_ONLY=Y` is defined, License Scheduler only considers `user@host` information for the first execution host of a parallel job when merging the license usage data. Setting in individual Feature sections overrides the global setting in the Parameters section.

If a feature has multiple Feature sections (using **LOCAL_TO**), each section must have the same setting for **CHECKOUT_FROM_FIRST_HOST_ONLY**.

Distribution policies

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

We refer to both licenses and license tokens because License Scheduler does not control licenses directly. Instead, it controls the dispatch of jobs that require licenses that are submitted through LSF or **taskman** by tracking license tokens.

Total license tokens

The total number of license tokens that are managed by License Scheduler for a single feature in one service domain depends on the following factors:

- The number of active license servers in the service domain
- The number of licenses that are checked out by applications that are not managed by LSF

License shares

License shares that are assigned in the distribution policy determine what portion of total licenses a project (in project mode) or cluster (in cluster mode) receives. Each project or cluster able to use a license feature must have a share of the license feature in the service domain.

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

$$\frac{\text{(shares assigned to project or cluster)}}{\text{(sum of all shares assigned)}} \times \text{(total number of licenses)}$$

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

When there are no jobs in the system, each project or cluster is assigned license tokens that are based on share assignments.

Cluster mode distribution policies

static

A portion of the total licenses is allocated to the cluster based on the configured share. The amount is static, and does not depend on the workload in the system.

dynamic

Shares of the total licenses are assigned to each cluster, along with a buffer size. The configured shares set the number of licenses each cluster receives initially, but this number is adjusted regularly based on demand from the cluster.

License distribution changes whenever a cluster requests an allocation update, by default every 15 seconds. In each update, the allocation can increase by as much as the buffer size. There is no restriction on decreasing cluster allocation.

When dynamic license distribution is used in cluster mode, minimum and maximum allocation values can be configured for each cluster. The minimum allocation is like the number of non-shared licenses for project mode, as this number of tokens is reserved for the exclusive use of the cluster.

If the minimum value configured exceeds the share assignment for the cluster, only the assigned share is reserved for the cluster.

Cluster shares take precedence over minimum allocations configured. If the minimum allocation exceeds the cluster's share of the total tokens, a cluster's allocation as given by **bld** may be less than the configured minimum allocation.

guarantees within a cluster

Guaranteed shares of licenses are assigned to projects within a cluster that use LSF guarantee-type SLAs. Optionally, sharing of guaranteed licenses not in use can be configured.

Guarantees are like ownership for cluster mode, and can be used with both static and dynamic distribution policies.

Note:

Guarantee-type SLAs are only available in LSF version 8.0 or newer.

When to use static license distribution

Configure shares for all license features in cluster mode. Static license distribution is the basic license distribution policy, and is built on by adding more configuration.

The basic static configuration can meet your needs if the demand for licenses across clusters is predictable and unchanging, or licenses are strictly owned by clusters, or you always have extra licenses.

When to use dynamic license distribution

Dynamic license allocation can meet your needs if the demand for licenses changes across clusters.

When to use LSF guarantee SLAs with License Scheduler

Configuring guarantee SLAs within LSF clusters can meet your needs if the licenses within a cluster are owned, and used either preferentially or exclusively by the license owners.

Project mode distribution policies

fairshare

Shares of the total licenses are assigned to each license project.

Unused licenses are shared wherever there is demand, however, when demand exceeds the number of licenses, share assignments are followed. Jobs are not preempted to redistribute licenses; instead licenses are redistributed when jobs finish running.

ownership and preemption

Shares of the total licenses are assigned to each license project. Owned shares of licenses are also assigned.

Unused licenses are shared wherever there is demand, however, when demand exceeds the number of licenses, the owned share is reclaimed using preemption.

Preemption occurs only while the specified number of owned licenses are not yet in use, and no free licenses are available. Once all owned licenses are used, License Scheduler waits for licenses to become free (instead of using preemption) and then distributes more tokens until the share is reached.

Jobs that are preempted by License Scheduler are automatically resumed once licenses become available.

By default, LSF releases 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 project, the applications must be able to release their licenses upon job suspension.

active ownership

Active ownership allows ownership to automatically adjust based on project activity. Ownership is expressed as a percent of the total ownership for active projects. The actual ownership for each project decreases as more projects become active. Set percentage ownership values to total more than 100% to benefit from active ownership.

non-shared licenses

Some licenses are designated as non-shared, and are reserved for exclusive use instead of being shared when not in use.

The number of non-shared licenses is contained by the number of owned licenses, but this number is not included in share calculations for the project. To designate some licenses as non-shared, add the non-shared number to both the owned and the non-shared values.

When to use fairshare with project mode

Configure fairshare for all license features in project mode. Fairshare is the basic license distribution policy, and is built on by adding additional configuration.

The basic fairshare configuration can meet your needs without configuring additional distribution policies if the licenses are assigned to specific license projects, but not strictly owned.

•

When to add ownership (and preemption)

Configure licenses as owned when:

- Licenses are owned by licenses projects, but can be loaned out when not in use.
- Maximizing license usage and license ownership are both important considerations. Loaned licenses must be returned to the owners as quickly as possible when needed (using preemption).
- Jobs borrowing licenses can be preempted.

When to add active ownership

Configure active ownership for owned licenses when:

- Ownership values are dynamic instead of being fixed values, and usually decrease as more projects actively seek licenses.

When to add non-shared licenses

Configure licenses as non-shared when:

- Licenses are owned.
- Licenses are used exclusively by the owners.
- Having licenses always available to the owners is more important than maximizing license use.

Project mode preemption

Preemption occurs only when there are no free licenses. During preemption, a project releases a borrowed license to the project that owns the license (and now has demand).

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

Preemption applies only to project mode, and depending on your configuration takes the following into consideration:

- runtime (a job that has the smallest run time gets preempted first, in general)
- fairshare settings
- ownership

- priority
- minimal job preemption

Depending on how your projects are set up (whether they are all at the same level or not), your preemption is either flat or hierarchical.

Basic preemption with projects configured

When preemption occurs, License Scheduler calculates token usage for each project. The calculation considers tokens in use, tokens that are required, and token ownership value.

Based on the token usage, License Scheduler determines the projects that require tokens, and the projects that have too many.

- Jobs belonging to projects that require tokens are scheduled first, ordered by project fairshare settings.
- Jobs belonging to projects with extra tokens are preempted first, if needed, ordered by project fairshare settings and the length of time each job is running.

With PRIORITY

If project **PRIORITY** is configured in the **Project** section, the sort order of projects is based on priority, where a higher priority project is preempted last.

With PREEMPT_ORDER

If **PREEMPT_ORDER** is set to **BY_OWNERSHIP** in the **Feature** section, the projects are sorted by ownership.

- Projects with the highest ownership are scheduled first.
- Projects with the smallest ownership are preempted first.

This setting overrides basic preemption and **PRIORITY**.

With ENABLE_MINJOB_PREEMPTION

If **ENABLE_MINJOB_PREEMPTION=Y**, the number of preempted jobs is minimized. Projects with extra tokens are sorted by **PRIORITY** (if configured) or fairshare. The jobs are then sorted by **RUSAGE**.

Jobs with higher **RUSAGE** are preempted first to minimize the number of jobs preempted.

This setting is used in addition to basic preemption or **PRIORITY**.

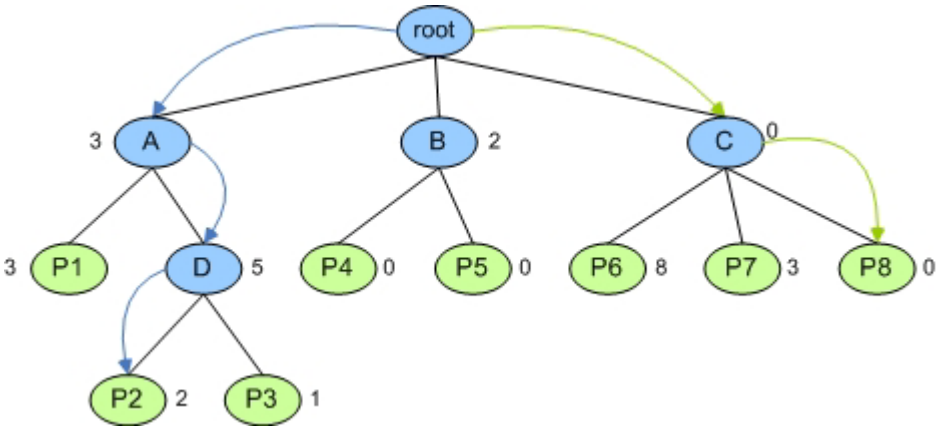
Hierarchical preemption with project groups configured

When project groups are configured, introducing a hierarchy into the project configuration, hierarchical preemption applies.

There are two methods of hierarchical preemption:

1. Top-down (default): Preemption occurs between cousins rather than siblings. The result is to balance preemption between the entire hierarchy of projects.
2. Bottom-up (if **LS_PREEMPT_PEER=Y**): Siblings can preempt each other. The result is to balance preemption within a family of projects first.

For example, your projects are set up as follows:



In top-down preemption, if P8 needs a token, it preempts from P1, P2, or P3 (who are more distant relations), not from P6 or P7 (siblings of P8).

In bottom-up preemption, P8 preempts instead from its siblings (P6 or P7).

Limits

Hierarchical preemption is also affected by any limits that are placed on the projects. If a limit is already reached (at any level of the hierarchy), License Scheduler considers the next possible node for preemption instead.

Preemption restrictions

A job cannot be preempted if:

- Preemption is restricted by a parameter such as: **MAX_JOB_PREEMPT**, **PREEMPT_RESERVE**, **LM_REMOVE_INTERVAL**, or **LS_WAIT_TO_PREEMPT**
- The preemptable job's server is not the current checking service domain.
- The job was submitted with a time duration and this time duration is expired.

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

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

LSF preemption with License Scheduler preemption

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.

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=Y** in `lsf.conf`, license job preemption works together with LSF slot-based preemption. Configured together **PREEMPTABLE_RESOURCES=mem** in `lsb.params` and

LSF_LIC_SCHED_PREEMPT_SLOT_RELEASE=Y in lsf.conf, license job preemption works together with LSF memory resource 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.

Five jobs are submitted and started with AppX, in proj2. Then, two 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 two jobs in proj2. The jobs are suspended, both their licenses and slots are released, and the two jobs in proj1 can run.

LSF JOB_CONTROLS configuration

If the LSF administrator defines JOB_CONTROLS in lsb.queues so that job controls (such as the signal SIGTSTP) take effect when License Scheduler preemption occurs, **LIC_SCHED_PREEMPT_STOP=Y** in lsf.conf must also be defined for License Scheduler preemption to work.

License usage with FlexNet

License Scheduler works differently with different types of applications, depending on how the application uses the license features and whether these license features are known at the start of the job.

Known license requirements

For many applications, all license features needed to run its jobs are known before the start of the job.

1. The job submission passes a license usage request to the 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.

Unknown license requirements

Some applications require an initial feature license to start a job and more feature or subfeature licenses during job execution. The user who submits the job knows the main license feature that is needed to start the job, but might not know the additional feature names or the number of more features required. This additional license feature not specified at job submission is considered unknown license use.

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.

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.

Project mode

Known license requirements

Project mode supports known license requirements that are specified in the rusage section of job submissions. By default, each license feature is reserved for the full length of the job.

Optionally, use the Feature section parameter **DYNAMIC=Y** to enable the use of duration in the rusage string, and release license features after a specified duration.

Unknown license requirements

Unknown license requirements not in the rusage string are counted as jobs not managed by LSF, and license distribution policies are not applied by default.

Optionally, license requirements not included in the rusage string can be tracked as part of the managed workload in project mode if there is at least one license feature that is specified in the job's rusage string. Set the parameter **ENABLE_DYNAMIC_RUSAGE=Y** in the Feature section to apply project distribution policies even when license rusage is not specified.

Cluster mode

Known license requirements

Cluster mode supports known license requirements that are specified in the rusage section of job submissions. Each license feature is reserved for the full length of the job.

In cluster mode, license requirements cannot be submitted with duration specified. If you have known license requirements for only a predetermined part of your job, you must choose between including them in the rusage and reserving for the entire job, or leaving them as unknown requirements.

Unknown license requirements

Unknown license requirements not in the rusage string are counted as part of the managed workload in cluster mode. License features not in the rusage string are not reserved for the job, however, distribution policies do apply. This behavior is equivalent to **ENABLE_DYNAMIC_RUSAGE=Y** in project mode.

Chapter 4. Configuring License Scheduler

Configure cluster mode

Use cluster mode to distribute licenses across LSF clusters, leaving the scheduler for each LSF cluster to schedule jobs, allocate licenses to projects within the cluster, and preempt jobs.

Configure parameters

1. Cluster mode can be set globally, or for individual license features. Set individually when using cluster mode for some features and project mode for some features.
   a. If you are using cluster mode for all license features, define `CLUSTER_MODE=Y` in the Parameters section of `lsf.licensescheduler`.
   b. If you are using cluster mode for some license features, define `CLUSTER_MODE=Y` for individual license features in the Feature section of `lsf.licensescheduler`.

      The Feature section setting of **CLUSTER_MODE** overrides the global Parameter section setting.
2. List the License Scheduler hosts.

   By default with an LSF installation, the **HOSTS** parameter is set to the **LSF_MASTER_LIST**.

   - List the hosts in order from most preferred to least preferred. The first host is the master license scheduler host.
   - Specify a fully qualified host name such as `hostX.mycompany.com` unless all your License Scheduler clients run in the same DNS domain.

   ```
   HOSTS=host1 host2
   ```
3. Specify the data collection frequency between License Scheduler and FlexNet.

   The default is 60 seconds.

   `LM_STAT_INTERVAL=`*seconds*
4. Specify the path to the FlexNet command **lmstat**.

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

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

Configure clusters

Configure the clusters that are permitted to use License Scheduler in the Clusters section of the `lsf.licensescheduler` file.

Configuring the clusters is only required if you are using more than one cluster.

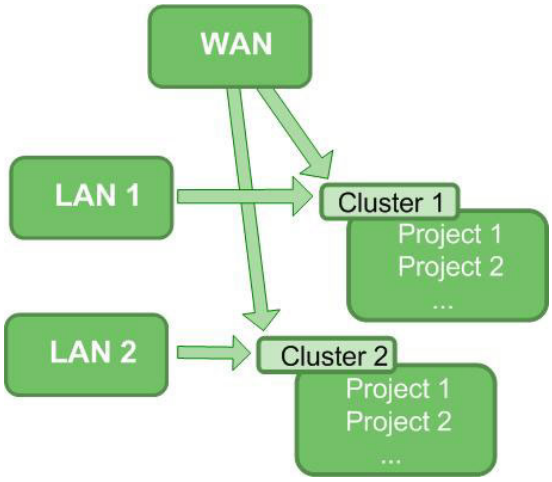
In the Clusters section, list all clusters that can use License Scheduler.
For example:

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

Cluster mode service domains

A service domain is a group of one or more FlexNet license servers. License Scheduler manages the scheduling of the license tokens, but the license server actually supplies the licenses.

In cluster mode, each cluster can access licenses from one WAN and one LAN service domain.



License Scheduler does not control application checkout behavior. If the same license is available from both the LAN and WAN service domains, License Scheduler expects jobs to try to obtain the license from the LAN first.

Configure ServiceDomain sections

You configure each service domain, with the license server names and port numbers that serve licenses to a network, in the ServiceDomain section of the `lsf.licensescheduler` file.

Whether the service domain is a WAN or LAN service domain is specified later in the Feature section.

1. Add a ServiceDomain section, and define **NAME** for each service domain.

   For example:

   ```
   Begin ServiceDomain
   NAME=DesignCenterA
   End ServiceDomain
   ```

2. Specify the FlexNet license server hosts for that domain, including the host name and FlexNet port number.

   For example:

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

   For multiple license servers:

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

   For redundant servers, the parentheses are used to group the three hosts that share the same `license.dat` file:

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

Note:

If FlexNet uses a port from the default range, you can specify the host name without the port number. See the FlexNet documentation for the values of the default port range.

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

Configure remote FlexNet license server hosts

The remote FlexNet license server hosts must have **lmutil** (or **lmstat**) in the **LMSTAT_PATH** directory before configuring these hosts with License Scheduler.

The license collector (**blcollect**) is a multi-threaded daemon that queries all FlexNet license servers under License Scheduler for license usage information. The license collector calls **lmutil** (or **lmstat**) to collect information from each license server. When there are both local and remote license servers (that is, license servers that are in a different subnet from the host running **blcollect**), the threads that collect information from the remote license servers are slower than the threads that collect information from local license servers.

If there are remote license servers, designate at least one remote license server within each domain as a remote agent host. The license collector connects to the remote agent host and calls **lmstat** on the remote agent host and gets license information from all license servers that the remote agent host serves. The remote agent host and the remote license servers should be in the same domain to improve access.

1. Select the connection method for the license collector to connect to remote hosts.

   License Scheduler supports the use of **ssh**, **rsh**, and **lsrun** to connect to remote hosts. If using **lsrun** as the connection method, the agent host must be a server host in the LSF cluster and RES must be started on this host. Otherwise, if using **ssh** or **rsh** as the connection method, the agent host does not have to be a server host in the LSF cluster.

   a. In the Parameters section, define the **REMOTE_LMSTAT_PROTOCOL** parameter and specify the connection command (and command options, if required) to connect to remote servers.

      ```
      REMOTE_LMSTAT_PROTOCOL=ssh [ssh_command_options] |
      rsh [rsh_command_options] | lsrun [lsrun_command_options]
      ```

      The default connection method is **ssh** with no command options. License Scheduler uses the specified command (and optional command options) to connect to the agent host. License Scheduler automatically appends the name of the agent host to the command, so there is no need to specify the host with the command.

      Note: License Scheduler does not validate the specified command, so you must ensure that you correctly specify the command. Any connection errors are noted in the **blcollect** log file.

   b. If the connection method is **ssh** or **rsh**, verify that this connection method is configured so the host running the license collector can connect to remote hosts without specifying a password.

2. Define remote license servers and remote agent hosts.

   In the ServiceDomain section, define the **REMOTE_LMSTAT_SERVERS** parameter:

   ```
   REMOTE_LMSTAT_SERVERS=host_name[(host_name ...)] [host_name[(host_name ...)] ...]
   ```

Specify a remote agent host, then any license servers that it serves in parentheses. The remote agent host and the license servers that it serves must be in the same subnet. If you specify a remote agent host by itself without any license servers (for example, REMOTE_LMSTAT_SERVERS=hostA), the remote agent host is considered to be a remote license server with itself as the remote agent host. That is, the license collector connects to the remote agent host and only gets license information on the remote agent host. You can specify multiple remote agent hosts to serve multiple subnets, or multiple remote agent hosts to serve specific license servers within the same subnet.

Any host that you specify here must be a license server defined in **LIC_SERVERS**. Any hosts defined in **REMOTE_LMSTAT_SERVERS** that are not also defined in **LIC_SERVERS** are ignored.

The following examples assume that the license collector (**blcollect**) is running on LShost1. That is, the following parameter is specified in the Parameters section:

```
Begin Parameters
...
HOSTS=LShost1
...
End Parameters
```

- One local license server (hostA) and one remote license server (hostB):

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

  – The license collector runs **lmutil** (or **lmstat**) directly on hostA to get license information on hostA.

  – Because hostB is defined without additional license servers, hostB is a remote agent host that only serves itself. The license collector connects to hostB (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmstat** to get license information on 1700@hostB.

- One local license server (hostA), one remote agent host (hostB) that serves one remote license server (hostC), and one remote agent host (hostD) that serves two remote license servers (hostE and hostF):

  ```
  LIC_SERVERS=((1700@hostA)(1700@hostB)(1700@hostC)(1700@hostD)(1700@hostE)(1700@hostF))
  REMOTE_LMSTAT_SERVERS=hostB(hostC) hostD(hostE hostF)
  ```

  – The license collector runs **lmutil** (or **lmstat**) directly to get license information from 1700@hostA, 1700@hostB, and 1700@hostD.

  – The license collector connects to hostB (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmstat** to get license information on 1700@hostC.

    hostB and hostC should be in the same subnet to improve access.

  – The license collector connects to hostD (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmutil** (or **lmstat**) to get license information on 1700@hostE and 1700@hostF.

    hostD, hostE, and hostF should be in the same subnet to improve access.

- One local license server (hostA), one remote license server (hostB), and one remote agent host (hostC) that serves two remote license servers (hostD and hostE):

  ```
  LIC_SERVERS=((1700@hostA)(1700@hostB)(1700@hostC)(1700@hostD)(1700@hostE))
  REMOTE_LMSTAT_SERVERS=hostB hostC(hostD hostE)
  ```

  – The license collector runs **lmutil** (or **lmstat**) directly to get license information on 1700@hostA and 1700@hostC.

- The license collector connects to hostB (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmstat** to get license information on 1700@hostB.
- The license collector connects to hostC (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmstat** to get license information on 1700@hostD and 1700@hostE.

  hostC, hostD, and hostE should be in the same subnet to improve access.

Configure LAN service domain

You configure LAN service domains in the Feature section of lsf.licensescheduler. Only a single cluster and service domain can be specified in each LAN Feature section. Licenses from the LAN service domain are statically allocated to the cluster.

In the Feature section, set

CLUSTER_DISTRIBUTION=*service_domain*(*cluster_name share*)

Use the service domain name that is defined in the ServiceDomain section. For example:

```
Begin Feature
NAME=verilog
CLUSTER_DISTRIBUTION=MyLanServer(tokyo_cluster 1)
End Feature
```

Configure WAN service domain

WAN configuration includes all clusters that are sharing the WAN service domain. As for a LAN service domain, you set this configuration in the **CLUSTER_DISTRIBUTION** parameter in the Feature section of the lsf.licensescheduler file.

For a WAN service domain, you can optionally configure dynamic license sharing based on past license use across all clusters that are served by the WAN service domain, and if required set minimum and maximum allocations for each cluster.

1. Set the WAN service domain name in the **CLUSTER_DISTRIBUTION** parameter.

   CLUSTER_DISTRIBUTION = *service_domain*(*cluster share*/*min*/*max*...)

   Use the service domain name that is defined in the ServiceDomain section.
2. Configure each cluster.

   All clusters with access to the WAN service domain licenses must be included.

   a. Set the cluster name.

   b. Set the share for each cluster.

      The share is a non-negative integer representing the share of licenses each cluster receives in a static license allocation, and the starting share in a dynamic license allocation.
3. Optionally, set **ALLOC_BUFFER** in the Feature section of the lsf.licensescheduler file. When set, this parameter enables a dynamic sharing policy.

   ALLOC_BUFFER = *buffer*

   or

   ALLOC_BUFFER = *cluster1 buffer1 cluster2 buffer2*...default *buffer*

   - When extra license tokens are available, each cluster's allocation increases to as much as **PEAK**+**BUFFER**.

     The value **BUFFER** is set by **ALLOC_BUFFER** in the Feature section, and the value PEAK is the peak value of dynamic license token use over a time interval that is set by **PEAK_INUSE_PERIOD** in the Parameters or Feature section.

- When allocated tokens are not being use in a cluster, the cluster's allocation goes down to **PEAK**+**BUFFER**.

  Since tokens are not being used in the cluster, the peak use value **PEAK** decreases, thus **PEAK**+**BUFFER** also decreases.

The allocation buffer sets both the rate at which the cluster allocation can grow, and the number of licenses that can go unused, depending on demand.

Allocation buffers help determine the maximum rate at which tokens can be transferred to a cluster as demand increases in the cluster. The maximum rate of transfer to a cluster is given by the allocation buffer that is divided by **MBD_REFRESH_INTERVAL**. Be careful not to set the allocation buffer too large so that licenses are not wasted because they are allocated to a cluster that cannot use them.

4. Optionally, when dynamic sharing is enabled (**ALLOC_BUFFER** is defined) you can set the minimum and maximum allocation for each cluster.

   The minimum allocation reserves license tokens for exclusive use by the cluster; the maximum allocation limits the total number of license tokens that are received by the cluster.

   Cluster shares take precedence over minimum allocations configured. If the minimum allocation exceeds the cluster's share of the total tokens, a cluster's allocation as given by **bld** may be less than the configured minimum allocation.

To allow a cluster to be able to use licenses only when another cluster does not need them, you can set the cluster distribution for the cluster to 0, and specify an allocation buffer for the number of tokens that the cluster can request.

For example:

```
Begin Feature
CLUSTER_DISTRIBUTION=Wan(CL1 0 CL2 1)
ALLOC_BUFFER=5
End Feature
```

When no jobs are running, the token allocation for CL1 is five. If CL2 does not require the tokens, CL1 can get more than five.

Examples

Static example (no allocation buffer set):

```
Begin Feature
NAME=verilog
CLUSTER_DISTRIBUTION=MyWanServer(tokyo_cl 1 newyork_cl 1 toronto_cl 2)
End Feature
```

In this example, licenses are statically allocated based solely on the number of shares that are assigned to each cluster. If the number of licenses is not evenly divisible by the number of shares, the additional licenses are distributed round-robin to clusters in the specified order in **CLUSTER_DISTRIBUTION**. Thus if there are 98 licenses in total, tokyo_cl receives 25, newyork_cl receives 25, and toronto_cl receives 48. Each cluster limits the total rusage of running jobs that are based on the allocated license tokens.

Dynamic example (allocation buffer set):

```
Begin Feature
NAME=verilog
CLUSTER_DISTRIBUTION=MyWanServer(tokyo_cl 1 newyork_cl 1 toronto_cl 2/10/50)
ALLOC_BUFFER=tokyo_cl 5 newyork_cl 1 toronto_cl 2
End Feature
```

In this example, licenses are initially distributed according to the assigned shares. Since allocation buffers are set, dynamic sharing that is based on past use is enabled. Based on the allocation buffers, toyko_cl receives license tokens the fastest when there is demand within the cluster. Minimum and maximum allocations of 10 and 50 are set for toronto_cl, which also has the largest share.

LAN and dynamic WAN example:

```
Begin Feature
NAME=verilog
CLUSTER_DISTRIBUTION=MyWan(c1 1/1/25 c2 1/1/30 c3 2/5/100) MyLan(c1 1)
ALLOC_BUFFER=c3 5 default 2
End Feature
```

In this example, the verilog license feature is available from both WAN and LAN service domain, however only cluster c1 receives the license feature from both servers. Licenses from the WAN service domain are initially distributed according to the assigned shares. Since allocation buffers are set, dynamic sharing that is based on past use is enabled. Based on the allocation buffers cluster c3 receives license tokens the fastest when there is demand within the cluster.

Configure license features

Each type of license requires a Feature section in the `lsf.licensescheduler` file.

1. Define the feature name that is used by FlexNet to identify the type of license by using the **NAME** parameter.

   Optionally, define an alias between License Scheduler and FlexNet feature names by using the **FLEX_NET** parameter to specify the FlexNet feature name and the **NAME** parameter to define the License Scheduler alias.

   You only need to specify **FLEX_NAME** if the License Scheduler token name is not identical to the FlexNet feature name, or for FlexNet feature names that either start with a number or contain a hyphen character (-), which are not supported in LSF.

   If the FlexNet feature name is AppZ201 and you intend to use this same name as the License Scheduler token name, define the **NAME** parameter as follows:

   ```
   Begin Feature
   NAME=AppZ201
   End Feature
   ```

   If the FlexNet feature name 201-AppZ, this is not supported in LSF because the feature name starts with a number and contains a hyphen. Therefore, define AppZ201 as an alias of the 201-AppZ FlexNet feature name as follows:

   ```
   Begin Feature
   NAME=AppZ201
   FLEX_NAME=201-AppZ
   End Feature
   ```

2. Optionally, combine multiple interchangeable FlexNet features into one License Scheduler alias by specifying multiple FlexNet feature names in **FLEX_NAME** as a space-delimited list.

   In this example, two FlexNet features named 201-AppZ and 202-AppZ are combined into an alias named AppZ201.

   ```
   Begin Feature
   NAME=AppZ201
   FLEX_NAME=201-AppZ 202-AppZ
   End Feature
   ```

AppZ201 is a combined feature that uses both 201-AppZ and 202-AppZ tokens. Submitting a job with AppZ201 in the rusage string (for example, bsub -Lp Lp1 -R "rusage[AppZ201=2]" myjob) means that the job checks out tokens for either 201-AppZ or 202-AppZ.

Configure taskman jobs in cluster mode

Optionally, to run taskman (interactive) jobs in cluster mode, include the dummy cluster interactive in your service domain configuration.

In the Feature section:

1. Include the dummy cluster interactive in the **CLUSTER_DISTRIBUTION** parameter.
2. Set a share for the dummy cluster interactive.
3. Optionally, set an allocation buffer for the dummy cluster interactive to enable dynamic allocation.

Examples

```
Begin Feature
NAME=licenseA
CLUSTER_DISTRIBUTION=MyLanServer(tokyo_cl 1 interactive 1)
End Feature
```

```
Begin Feature
NAME=licenseB
CLUSTER_DISTRIBUTION=MyWanServer(tokyo_cl 1 newyork_cl 1 interactive 2)
End Feature
```

Allocate licenses to non-LSF jobs

Applies to WAN service domains only.

Set **WORKLOAD_DISTRIBUTION** in the Feature section to allocate licenses for non-LSF use.

WORKLOAD_DISTRIBUTION=*service_domain_name*(LSF *lsf_distribution* NON_LSF *non_lsf_distribution*)

If **WORKLOAD_DISTRIBUTION** is set for a LAN service domain in cluster mode, the parameter is ignored.

For example, to set aside 20% of licenses for use outside of LSF:

```
Begin Feature
NAME=licenseB
CLUSTER_DISTRIBUTION=MyWanServer(tokyo_cl 1 newyork_cl 1)
WORKLOAD_DISTRIBUTION=MyWanServer(LSF 8 NON_LSF 2)
End Feature
```

Restart to implement configuration changes

1. Run bladmin reconfig to restart the bld.
2. If you deleted any Feature sections, restart mbatchd. In this case, a message is written to the log file, prompting the restart.

   If required, run badmin mbdrestart to restart each LSF cluster.

View license allocation

Run blstat -t *token_name* to view information for a specific license token (as configured in a Feature section).
blstat output differs for cluster mode and project mode.

Configure cluster mode with guarantees

Cluster mode distributes licenses across LSF clusters. To guarantee license resources to projects within a cluster and allow loaning of license resources when not in use, use LSF guarantee-type SLAs. Guarantees and loans in cluster mode are similar to non-shared licenses and ownership in project mode.

A guarantee provides jobs that belong to set consumers with specific resources (such as hosts). Jobs run with guaranteed resources when possible. When the guaranteed resources are used, jobs run outside the guarantee following whatever other scheduling features are configured. Guarantees are configured within a guaranteed resource pool.

Guarantee SLAs are configured in Platform LSF. For more information, see *Administering IBM Platform LSF* and *IBM Platform LSF Configuration Reference*.

Configure service classes

Service classes allow access to guaranteed resources. Configure a service class for each license project in the cluster.

Configure each ServiceClass section in the `lsb.serviceclasses` file. Begin with the line Begin ServiceClass and end with the line End ServiceClass. For each service class, you must specify:

1. NAME: the name of the service class.
2. GOALS = [GUARANTEE]
3. Optional parameters for the ServiceClass section are ACCESS_CONTROL, AUTO_ATTACH, and DESCRIPTION.

You can configure as many service class sections as you need.

Important:

The name that you use for your service class cannot be the same as an existing host partition or user group name.
For example:

```
Begin ServiceClass
NAME = sla1
GOALS = [GUARANTEE]
ACCESS_CONTROL=LIC_PROJECTS[ proj1 ]
DESCRIPTION = A guarantee SLA with access restricted to the license project proj1.
End ServiceClass
```

Automatically attach jobs to service classes

When the optional parameter AUTO_ATTACH is set, jobs are automatically attached to the service class.

When automatic attachment is not set, jobs can be submitted to the service class by running `bsub -sla` *`serviceclass_name`*.

If a job can access more than one SLA with automatic attachment set, it is attached to the first valid SLA in the order of the configuration file.

Set `AUTO_ATTACH=Y` in the ServiceClass section in the `lsb.serviceclasses` file.
For example,

```
Begin ServiceClass
NAME = sla1
GOALS = [GUARANTEE]
ACCESS_CONTROL=LIC_PROJECTS[ proj1 ]
AUTO_ATTACH=Y
DESCRIPTION = A guarantee SLA with access restricted to the license project proj1.
Jobs submitted to proj1 are attached to the SLA automatically and run on guaranteed
resources if possible.
End ServiceClass
```

Configure a resource pool of license tokens

Guaranteed resource pools provide a minimum resource guarantee to consumers, and can optionally loan out guaranteed resources not in use.

Guaranteed resource pools are defined in `lsb.resources` and used by consumers that are defined within ServiceClass sections in `lsb.serviceclasses`.

Configure a GuaranteedResourcePool section in `lsb.resources`. Begin with the line Begin GuaranteedResourcePool and end with the line End GuaranteedResourcePool. Specify the following parameters:

1. NAME: the name of the guaranteed resource pool.
2. TYPE: the guarantee type. For licenses, use the type resources and include the name of the license feature.
3. DISTRIBUTION: share assignments for all service classes using the resource pool. Can be percent or absolute numbers.
4. Optional parameters for GuaranteedResourcePool sections of resources are LOAN_POLICIES, and DESCRIPTION.

You can configure as many resource pools as you need. One resource pool can be used by several SLAs, and one SLA can access multiple resource pools.
For example:

```
Begin GuaranteedResourcePool
NAME = hspice_guarantees
TYPE = resource[hspice]
DISTRIBUTION = ([proj1_sc,50%][proj2_sc,50%])
DESCRIPTION = A resource pool of hspice licenses controlled by License Scheduler
and used by proj1_sc and proj2_sc.
End GuaranteedResourcePool
```

Configure loans

Loans from unused guarantees are recommended when you are using cluster mode. When loans are disabled, use a static license distribution policy.

When configured, unused license resources are loaned out based on the loan policy. The loan policy allows specific queues to access unused resources from guaranteed resource pools.

1. Configure a guaranteed resource pool in `lsb.resources` with the required **NAME**, **TYPE**, and **DISTRIBUTION** parameters.
2. Add a loan policy to the guaranteed resource pool.

   Use **LOAN_POLICIES= QUEUES[queue_name]** to specify which queues can access loaned resources. Use the keyword **all** to loan to jobs from any queue.

   For example, to allow loans to jobs from the queue my_queue:

```
Begin GuaranteedResourcePool
...
LOAN_POLICIES = QUEUES[my_queue]
...
End GuaranteedResourcePool
```

Configure loans to short jobs

Loans can be restricted based on job run time, or estimated run time.

Add the policy DURATION[minutes] to the guaranteed resource pool configuration in `lsb.resources`, where minutes is an integer.
Use DURATION to set a maximum job runtime limit (or estimated run time, whichever is shorter) for jobs to borrow resources. Omit **DURATION** completely to allow jobs with any run time to borrow from the guarantee.
For example, to allow loans to jobs from any queue with a run time of 10 minutes or less:

```
Begin GuaranteedResourcePool
...
LOAN_POLICIES = QUEUES[all] DURATION[10]
...
End GuaranteedResourcePool
```

Configure loans to stop when jobs are waiting for guaranteed resources

Loans can be restricted so that jobs have access to the loaned resources only when consumers with unused guaranteed resources do not have pending loads.

Restricting loans is useful when running jobs that require several licenses. With restricted loans enabled, loaning out single licenses does not delay jobs that are waiting for license resources to accumulate.

Add the policy CLOSE_ON_DEMAND to the guaranteed resource pool configuration in `lsb.resources`.
For example,

```
Begin GuaranteedResourcePool
...
LOAN_POLICIES = QUEUES[queue1] CLOSE_ON_DEMAND
...
End GuaranteedResourcePool
```

Configure a queue with access to all guaranteed resources

Queues with high priority (such as administrator test queues) can be configured with access to all guaranteed resources, regardless of SLA demand.

Configure a queue in `lsb.queues` with SLA_GUARANTEES_IGNORE = Y.

Note:

Using SLA_GUARANTEES_IGNORE=Y defeats the purpose of guaranteeing resources. sparingly for low traffic queues only.

Restart for changes to take effect

Cluster mode must be enabled, and LSF clusters must be restarted for LSF configuration changes to take effect.

1. In the Parameters section of `lsf.licensescheduler`, confirm cluster mode is enabled (`CLUSTER_MODE=Y`).

2. Run `badmin mbdrestart` to restart each LSF cluster.
3. Run `bladmin reconfig` to restart the bld.

View guaranteed resource pools

Guaranteed resource pool configuration includes the resource type, and distribution among consumers that are defined in the corresponding service classes.

Run **bresources -g -l -m** to see details of the guaranteed resource pool configuration, including a list of hosts currently in the resource pool.

Project mode with projects

You can configure license distribution when you are running license projects in project mode. Each distribution policy is applied locally, within service domains.

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.

Configure parameters

1. Project mode can be set globally, or for individual license features. Set individually when you are using project mode for some features and cluster mode for some features.
   a. If you are using project mode for all license features, define `CLUSTER_MODE=N` in the Parameters section of `lsf.licensescheduler`.
   b. If you are using project mode for some license features, define `CLUSTER_MODE=N` for individual license features in the Feature section of `lsf.licensescheduler`.

      The Feature section setting of **CLUSTER_MODE** overrides the global Parameter section setting.
2. List the License Scheduler hosts.

   By default with an LSF installation, the **HOSTS** parameter is set to the **LSF_MASTER_LIST**.
   - List the hosts in order from most preferred to least preferred. The first host is the master license scheduler host.
   - Specify a fully qualified host name such as hostX.mycompany.com unless all your License Scheduler clients run in the same DNS domain.

   ```
   HOSTS=host1 host2
   ```
3. Specify the data collection frequency between License Scheduler and FlexNet.

   The default is 30 seconds.

   ```
   LM_STAT_INTERVAL=seconds
   ```
4. Specify the path to the FlexNet command **lmstat**.

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

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

Configure clusters

Configure the clusters that are permitted to use License Scheduler in the Clusters section of the `lsf.licensescheduler` file.

This configuration is only required if you are using more than one cluster.

In the Clusters section, list all clusters that can use License Scheduler.
For example:

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

Configure projects

Each project that is defined in a Projects section of `lsf.licensescheduler` can have a distribution policy that is applied in the Feature section, where projects can be associated with license features.

Define the projects with or without priority.

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

The higher the number, the higher the priority. When two projects have the same priority number that is configured, the first listed project has a higher priority. Priority is taken into account when license preemption occurs, where lower priority projects are preempted first.
If not explicitly configured, the default project has the priority of 0. A default project is used when no license project is specified during job submission.

Add project description

Optionally, you can add a project description of up to 64 characters to your projects to help identify them.

In the `Project` section of `lsf.licensescheduler`, find the project and add a description in the DESCRIPTION column.
For example:

```
Begin Projects
PROJECTS PRIORITY DESCRIPTION
p1       10       "Engineering project 123"
p2       9        "QA build project 2C"
P3       8        ""
End Projects
```

When you are running **blinfo -Lp** or **blinfo -G**, any existing project descriptions display.

Project mode service domains

A service domain is a group of one or more FlexNet license servers. License Scheduler manages the scheduling of the license tokens, but the license server actually supplies the licenses. You must configure at least one service domain for License Scheduler.

In project mode, each cluster can access licenses from multiple WAN and LAN service domains. License Scheduler collects license availability and usage from

FlexNet license server hosts, and merges this information with license demand and usage information from LSF clusters to make distribution and preemption decisions.

Note:

Unless you require multiple service domains for some specific reason, configure both modes with at most one LAN and one WAN for each feature in a cluster. Because License Scheduler does not control license checkout, running with one cluster that is accessing multiple service domains is not optimal.

Configure service domains

You configure each service domain, with the license server names and port numbers that serve licenses to a network, in the ServiceDomain section of the `lsf.licensescheduler` file.

1. Add a ServiceDomain section, and define **NAME** for each service domain.

   For example:

   ```
   Begin ServiceDomain
   NAME=DesignCenterA
   End ServiceDomain
   ```

2. Specify the FlexNet license server hosts for that domain, including the host name and FlexNet port number.

   For example:

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

   For multiple license servers:

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

   For redundant servers, the parentheses are used to group the three hosts that share `license.dat` file:

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

   Note:

   If FlexNet uses a port from the default range, you can specify the host name without the port number. See the FlexNet documentation for the values of the default port range.

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

Configure remote FlexNet license server hosts

The remote FlexNet license server hosts must have **lmutil** (or **lmstat**) in the **LMSTAT_PATH** directory before configuring these hosts with License Scheduler.

The license collector (**blcollect**) is a multi-threaded daemon that queries all FlexNet license servers under License Scheduler for license usage information. The license collector calls **lmutil** (or **lmstat**) to collect information from each license server. When there are both local and remote license servers (that is, license servers that are in a different subnet from the host running **blcollect**), the threads that collect information from the remote license servers are slower than the threads that collect information from local license servers.

If there are remote license servers, designate at least one remote license server within each domain as a remote agent host. The license collector connects to the

remote agent host and calls **lmstat** on the remote agent host and gets license information from all license servers that the remote agent host serves. The remote agent host and the remote license servers should be in the same domain to improve access.

1. Select the connection method for the license collector to connect to remote hosts.

   License Scheduler supports the use of **ssh**, **rsh**, and **lsrun** to connect to remote hosts. If using **lsrun** as the connection method, the agent host must be a server host in the LSF cluster and RES must be started on this host. Otherwise, if using **ssh** or **rsh** as the connection method, the agent host does not have to be a server host in the LSF cluster.

   a. In the Parameters section, define the **REMOTE_LMSTAT_PROTOCOL** parameter and specify the connection command (and command options, if required) to connect to remote servers.

      `REMOTE_LMSTAT_PROTOCOL=ssh [`*ssh_command_options*`] | rsh [`*rsh_command_options*`] | lsrun [`*lsrun_command_options*`]`

      The default connection method is **ssh** with no command options. License Scheduler uses the specified command (and optional command options) to connect to the agent host. License Scheduler automatically appends the name of the agent host to the command, so there is no need to specify the host with the command.

      Note: License Scheduler does not validate the specified command, so you must ensure that you correctly specify the command. Any connection errors are noted in the **blcollect** log file.

   b. If the connection method is **ssh** or **rsh**, verify that this connection method is configured so the host running the license collector can connect to remote hosts without specifying a password.

2. Define remote license servers and remote agent hosts.

   In the ServiceDomain section, define the **REMOTE_LMSTAT_SERVERS** parameter:

   `REMOTE_LMSTAT_SERVERS=`*host_name*`[(`*host_name ...*`)] [`*host_name*`[(`*host_name ...*`)] ...]`

   Specify a remote agent host, then any license servers that it serves in parentheses. The remote agent host and the license servers that it serves must be in the same subnet. If you specify a remote agent host by itself without any license servers (for example, `REMOTE_LMSTAT_SERVERS=hostA`), the remote agent host is considered to be a remote license server with itself as the remote agent host. That is, the license collector connects to the remote agent host and only gets license information on the remote agent host. You can specify multiple remote agent hosts to serve multiple subnets, or multiple remote agent hosts to serve specific license servers within the same subnet.

   Any host that you specify here must be a license server defined in **LIC_SERVERS**. Any hosts defined in **REMOTE_LMSTAT_SERVERS** that are not also defined in **LIC_SERVERS** are ignored.

   The following examples assume that the license collector (**blcollect**) is running on LShost1. That is, the following parameter is specified in the Parameters section:

   ```
   Begin Parameters
   ...
   HOSTS=LShost1
   ...
   End Parameters
   ```

   - One local license server (`hostA`) and one remote license server (`hostB`):

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

  - The license collector runs **lmutil** (or **lmstat**) directly on hostA to get license information on hostA.
  - Because hostB is defined without additional license servers, hostB is a remote agent host that only serves itself. The license collector connects to hostB (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmstat** to get license information on 1700@hostB.
- One local license server (hostA), one remote agent host (hostB) that serves one remote license server (hostC), and one remote agent host (hostD) that serves two remote license servers (hostE and hostF):

```
LIC_SERVERS=((1700@hostA)(1700@hostB)(1700@hostC)(1700@hostD)(1700@hostE)(1700@hostF))
REMOTE_LMSTAT_SERVERS=hostB(hostC) hostD(hostE hostF)
```

  - The license collector runs **lmutil** (or **lmstat**) directly to get license information from 1700@hostA, 1700@hostB, and 1700@hostD.
  - The license collector connects to hostB (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmstat** to get license information on 1700@hostC.

    hostB and hostC should be in the same subnet to improve access.
  - The license collector connects to hostD (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmutil** (or **lmstat**) to get license information on 1700@hostE and 1700@hostF.

    hostD, hostE, and hostF should be in the same subnet to improve access.
- One local license server (hostA), one remote license server (hostB), and one remote agent host (hostC) that serves two remote license servers (hostD and hostE):

```
LIC_SERVERS=((1700@hostA)(1700@hostB)(1700@hostC)(1700@hostD)(1700@hostE))
REMOTE_LMSTAT_SERVERS=hostB hostC(hostD hostE)
```

  - The license collector runs **lmutil** (or **lmstat**) directly to get license information on 1700@hostA and 1700@hostC.
  - The license collector connects to hostB (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmstat** to get license information on 1700@hostB.
  - The license collector connects to hostC (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmstat** to get license information on 1700@hostD and 1700@hostE.

    hostC, hostD, and hostE should be in the same subnet to improve access.

Configure license features

Each type of license requires a Feature section in the lsf.licensescheduler file.

The Feature section includes the license distribution policy.

1. Define the feature name that is used by FlexNet to identify the type of license by using the **NAME** parameter.

   Optionally, define an alias between License Scheduler and FlexNet feature names by using the **FLEX_NAME** parameter to specify the FlexNet feature name and the **NAME** parameter to define the License Scheduler alias.

   You only need to specify **FLEX_NAME** if the License Scheduler token name is not identical to the FlexNet feature name, or for FlexNet feature names that either start with a number or contain a hyphen character (-), which are not supported in LSF.

If the FlexNet feature name is AppZ201 and you intend to use this same name as the License Scheduler token name, define the **NAME** parameter as follows:

```
Begin Feature
NAME=AppZ201
End Feature
```

If the FlexNet feature name 201-AppZ, this is not supported in LSF because the feature name starts with a number and contains a hyphen. Therefore, define AppZ201 as an alias of the 201-AppZ FlexNet feature name as follows:

```
Begin Feature
NAME=AppZ201
FLEX_NAME=201-AppZ
End Feature
```

2. Optionally, combine multiple interchangeable FlexNet features into one License Scheduler alias by specifying multiple FlexNet feature names in **FLEX_NAME** as a space-delimited list.

   In this example, two FlexNet features named 201-AppZ and 202-AppZ are combined into an alias named AppZ201.

   ```
   Begin Feature
   NAME=AppZ201
   FLEX_NAME=201-AppZ 202-AppZ
   End Feature
   ```

   `AppZ201` is a combined feature that uses both `201-AppZ` and `202-AppZ` tokens. Submitting a job with `AppZ201` in the rusage string (for example, `bsub -Lp Lp1 -R "rusage[AppZ201=2]" myjob`) means that the job checks out tokens for either `201-AppZ` or `202-AppZ`.

3. Define a distribution policy.

   A distribution policy defines the license fairshare policy in the format:

   ```
   DISTRIBUTION = ServiceDomain1 (project1 share_ratio project2 share_ratio ...)
   ServiceDomain2 (project3 share_ratio ...)
   ```

   For example, a basic configuration assigns shares:

   ```
   Begin Feature
   FLEX_NAME=201-AppZ
   NAME=AppZ201
   DISTRIBUTION = DesignCenterA (LpA 2 LpB 1 default 1)
   End Feature
   ```

   LpA has the right to twice as many licenses as LpB. Jobs that are submitted without a license project that is specified can run under the default project.

4. Optionally, add owned licenses to the distribution policy in the format:

   ```
   DISTRIBUTION = ServiceDomain1 (project1 share_ratio/number_owned
   project2 share_ratio/number_owned ...) ServiceDomain2
   (project3 share_ratio ...)
   ```

   If `LS_FEATURE_PERCENTAGE=Y` or `LS_ACTIVE_PERCENTAGE=Y` in `lsf.licensescheduler`, *number_owned* is expressed as a percentage of the total licenses.

   Example 1:

   ```
   DISTRIBUTION = LanServer(Lp1 1 Lp2 1/10)
   ```

   This example assumes that there are 10 licenses in total, all owned by Lp2.

   The two 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.

   Example 2:

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

Lp1 is set 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 preempts other license project jobs when licenses are needed immediately.

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 use licenses only when Lp2 does not need them.

Track partial and unspecified license use

When you want to manage licenses not included in job resource requirements or have applications that you know use licenses for only part of the length of each job, use these optional settings.

1. Optionally, specify **DYNAMIC=Y** to consider the license feature as a dynamic resource when it is only used for part of the job.

   Set **DYNAMIC=Y** for applications with known license use that do not use the license for the entire length of the job. Jobs are submitted with **duration** specified, then release the license when not in use.

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

   For example, a **taskman** job submission with **duration**:

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

2. Optionally, set **ENABLE_DYNAMIC_RUSAGE=Y** in the Feature section of lsf.licensescheduler to track license use of license features not specified at job submission.

   For example:

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

   Submit a job to run the application, specifying the license feature name:

   ```
   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   INUSE  EFFECTIVE_PROJECT
feat1           1             LanServer        1      proj1
```

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   INUSE  EFFECTIVE_PROJECT
feat1           1             LanServer        1      proj1
feat2           1             LanServer        1      proj1
```

Restart to implement configuration changes

1. Run `bladmin reconfig` to restart the **bld**.
2. If you deleted any Feature sections, restart **mbatchd**. In this case, a message is written to the log file, prompting the restart.

   If required, run `badmin mbdrestart` to restart each LSF cluster.

View projects and descriptions

Run **blinfo -Lp** to view projects and descriptions.
For example:

```
blinfo -Lp
PROJECT PRIORITY DESCRIPTION
p1      10       Engineering project 123
p2      9        QA build project 2C
P3      8
```

View license allocation

Run `blstat -t` *token_name* to view information for a specific license token (as configured in a Feature section).
blstat output differs for cluster mode and project mode.

Project mode optional settings

After you configure License Scheduler in project mode with projects or project groups, you can include some additional configuration that is not required, but can be useful.

Active ownership

With ownership defined, projects with demand for licenses are able to reclaim licenses up to the assigned ownership share for the project. With active ownership enabled, ownership is expressed as a percent of the total ownership for active projects, and the actual ownership for each project decreases as more projects become active. Active ownership allows ownership to automatically adjust based on project activity.

Active ownership can be used with projects, groups of projects, and project groups. Set percentage ownership values to total more than 100% to benefit from active ownership.

Configure active ownership

When active ownership is enabled, ownership settings for inactive projects are disregarded during license token distribution.

1. Set `LS_ACTIVE_PERCENTAGE=Y` in the Feature section.

   All ownership values for inactive projects are set to zero, and if the total ownership percent exceeds 100%, the total ownership is adjusted.

   `LS_FEATURE_PERCENTAGE=Y` is automatically set, and owned and non-shared values are expressed in percent. If used with project groups, **`OWNERSHIP`**, **`LIMITS`** and **`NON_SHARED`** are expressed in percent.
2. Set the percentage of owned licenses in the **`DISTRIBUTION`** parameter (Feature section) for a total percentage that exceeds 100%.

   For example:

   ```
   ...
   DISTRIBUTION=wanserver (Lp1 2/50 Lp2 1/30 Lp3 2/30 Lp4 3/30)
   LS_ACTIVE_PERCENTAGE=Y
   ...
   ```

   In this example, all four license projects are configured with a share and an owned value. Lp1 has the greatest number of owned licenses, and can use preemption to reclaim the most licenses.

   If only Lp1 is active, Lp1 owns 50% of licenses. Total active ownership is 50%, so no adjustment is made.

   If Lp1 and Lp2 are active, Lp1 owns 50% and Lp2 owns 30%. Total active ownership is 80%, so no adjustment is made.

   If Lp1, Lp2, and Lp3 are active, Lp1 owns 50%, Lp2 owns 30%, and Lp3 owns 30%. Total active ownership is 110%, so ownership is scaled to result in Lp1 owning 46%, Lp2 owning 27%, and Lp3 owning 27% (Exact numbers are rounded).

   If all projects are active, the total active ownership is 140%. Ownership is scaled to result in Lp1 owning 37%, Lp2 owning 21%, Lp3 owning 21%, and Lp4 owning 21% (Exact numbers are rounded).

Default projects

Jobs requiring a license feature but not submitted to a license project for that feature are submitted to the default project. For jobs to run, a share of license tokens must be assigned to the default project.

If you do not want the default project to get shares of license tokens, you do not have to define a default project in the distribution policy for a feature, however jobs in the default project become pending by default.

To avoid having jobs that are submitted without a project pend, either assign shares to the default project, or disable default projects so jobs are rejected.

Configure default project shares

Jobs cannot run in the default project unless shares are assigned.

Define a **default** project in the Feature section **DISTRIBUTION** parameter.
Any job that is submitted without a project name that is specified by `-Lp` can now use tokens from the **default** project.

Disable default projects

License token jobs that are submitted without a project that is specified are accepted and assigned to the default project, unless your configuration specifies that such jobs be rejected.

Optionally, set `LSF_LIC_SCHED_STRICT_PROJECT_NAME=y` in `lsf.conf`.
Jobs that are submitted without a project that is specified are rejected, and the default license project is not used.

Groups of projects

If you configure groups of projects, you can set shares and ownership for each group and distribute license features to groups of projects. Configure a license project to belong only to one group. Preemption first occurs between groups of projects, and then occurs between projects.

Preemption with groups of projects

The following tables show changes in preemption behavior that is based on ownership that is configured for groups of projects, with a total of 20 licenses. With groups of projects that are configured, GroupA is able to preempt to reclaim 10 owned licenses. Since Lp2 is not using all five owned licenses, Lp1 can use more than the share it owns.

Project license ownership only

License project	Licenses owned	Licenses used
Lp1	5	6
Lp2	5	0
Lp3	5	7
Lp4	5	7

Groups of projects with license ownership

Group	License projects	Project licenses owned	Licenses that are used after preemptions
GroupA	Lp1	5	9
	Lp2	5	1
GroupB	Lp3	5	6
	Lp4	5	4

Configure group license ownership

In `lsf.licensescheduler`, set the **GROUP** parameter in the Feature section.

1. Set up groups and members.

For 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.

Configure interactive (taskman) jobs

By default, interactive (**taskman**) jobs do not receive a share of the license token allocation, while all clusters receive equal shares.

You can allocate a share of all license features to interactive jobs in the Parameters section.

To globally enable a share of the licenses for interactive tasks, you must set the `ENABLE_INTERACTIVE` in `lsf.licensescheduler`.
In `lsf.licensescheduler`, edit the `Parameters` section:

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

When the change in configuration takes effect, interactive tasks are allocated the same share (by default) as each cluster.

Configure cluster and interactive allocations

By default in project mode, each cluster receives one allocation share from a license feature, and interactive tasks receive no shares.

You can modify the allocation of license shares across clusters and to interactive tasks in individual Feature sections.

In the Features section of `lsf.licensescheduler`, set the **ALLOCATION** parameter.
`ALLOCATION=`*project_name* (*cluster_name* [*number_shares*] ...)

Allocation examples

For example, this **ALLOCATION** setting matches the default when **ALLOCATION** is undefined and interactive jobs are enabled with ENABLE_INTERACTIVE=Y. An equal share is allocated to each cluster and to interactive jobs.

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

In this example, licenses are shared equally between cluster1 and interactive tasks, with cluster2 receiving nothing:

```
Begin Parameters
...
ENABLE_INTERACTIVE = y
...
End Parameters
Begin Feature
NAME = AppY
DISTRIBUTION = LanServer (Lp1 1)
ALLOCATION = Lp1(cluster1 2 cluster2 0 interactive 2)
End Feature
```

In the following example, even though the global allocation to interactive jobs is disabled (ENABLE_INTERACTIVE = N), **ALLOCATION** defined in the Feature section can assign a share to interactive jobs for this license feature.

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

Given a total of 12 licenses, 4 are allocated to cluster2 and 8 are allocated to interactive tasks.

Configure feature groups

Feature groups that are configured in one FeatureGroup section allow you to view the information for multiple features, which are grouped together.

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

- Each FeatureGroup section must have a unique name.
- The feature names in **FEATURE_LIST** must already be defined in Feature sections.
- **FEATURE_LIST** cannot be empty or contain duplicate feature names.
- Features can be in more than one FeatureGroup section.

For example:

```
Begin FeatureGroup
NAME = Corporate
FEATURE_LIST = ASTRO VCS_Runtime_Net Hsim Hspice
End FeatureGroup
Begin FeatureGroup
NAME = Offsite
FEATURE_LIST = Encounter NCSim  NCVerilog
End FeatureGroup
```

Restart to implement configuration changes

Changes that are made in `lsf.licensescheduler` require restarting the **bld**.

Changes that are made in `lsf.conf` require restating the LSF clusters.

1. Run `badmin mbdrestart` to restart each LSF cluster.
2. Run `lsadmin limrestart` or `bladmin reconfig` to restart the **bld**.

View license feature group information

When **FEATURE_LIST** is configured for a group of license features in `lsf.licensescheduler`, you can view detailed information about the groups.

Run **blinfo -g** or **blstat -g**.
For example, if the feature group called **myFeatureGroup1** has the members **feature2** and **feature3**:
`blstat -g "myFeatureGroup1"`
Information displays for **feature2** and **feature3** in descending alphabetical order.
Run **blstat -g** alone or with options **-Lp**, **-t**, **-D** ,**-G**', **-s**.
Run **blinfo '-g'** alone or with options **-a, -t**, **-C**, and **-A**.

License feature locality

Use license feature locality 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 that is requesting the token.

How locality works

Setting locality means that license resources requested from different clusters are mapped to different tokens in License Scheduler

Features with different locality are treated as different tokens by License Scheduler. You must configure separate feature sections for same feature with different localities.

Note:

You must make sure that your features are configured so that the applications always first try to check out licenses locally.

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

Configure locality

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

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 specification 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 as different resources.

1. In `lsf.licensescheduler`'s Feature section, configure LOCAL_TO.

   For example: LOCAL_TO=Site1(clusterA clusterB) configures the feature for more than one cluster, where the cluster names are already defined in the Clusters section of `lsf.licensescheduler`.

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

   License Scheduler now treats license features that are served to different locations as different token names, and distributes the tokens to projects according to the distribution and allocation policies for the feature.

2. (Optional) View locality settings.

   a. Run `blinfo -A`.

      The feature allocation by cluster locality displays.

```
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%]
```

   b. Run `blinfo -C`.

      The cluster locality information for the features displays.

```
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
```

   c. Run `blusers`.

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

d. Run `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
```

e. Run `bhosts -s`.

Different resource information displays depending on the cluster locality of the features.

From clusterA:

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

From clusterB in siteB:

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

Example configuration: two sites and four service domains:
Some of your service domains may have geographical restrictions when the domains are serving licenses. In this example, two clusters in one location can run **hspice** jobs. and four service domains are defined for the **hspice** 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 seven **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

```
Begin Feature
NAME = hspice
DISTRIBUTION = SD1 (Lp1 1 Lp2 1)
LOCAL_TO = clusterA
End Feature
Begin Feature
NAME = hspice
DISTRIBUTION = SD2 (Lp1 1 Lp2 1)
LOCAL_TO = clusterB
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 = clusterA
End Feature
Begin Feature
NAME = hspice
GROUP_DISTRIBUTION = group1
SERVICE_DOMAINS = SD2
LOCAL_TO = clusterB
End Feature
Begin Feature
NAME = hspice
GROUP_DISTRIBUTION = group1
SERVICE_DOMAINS = SD3 SD4
End Feature
```

Submit jobs that use locality

LOCAL_TO is configured in `lsf.licensescheduler`.

Job submission is simplified when locality is configured.

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
```

How locality works with other settings

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. You can define different License Scheduler tokens for the same FlexNet feature. The service domain names (in either the **DISTRIBUTION** line or the **SERVICE_DOMAINS** for group configurations) of the same FlexNet feature in different feature sections must be exclusive. They cannot overlap.
2. 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.
- Use same location name for different FlexNet features with the same pattern of locality, but License Scheduler does not check whether the same location name of a different feature contains the same list of clusters.

3. Features must either have a different **NAME** or have **LOCAL_TO** defined. The service domains for each License Scheduler token of same FlexNet feature must be exclusive.

How locality 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 that are 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 that are 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** is equal shares for all the clusters defined in **LOCAL_TO** parameter. This share 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 1 clusterB 0)
No	clusterA	Lp1(clusterA 1) Lp2(clusterA 1)

About interactive taskman jobs

The License Scheduler command **taskman** is a job starter for **taskman** jobs to use License Scheduler without **bsub**. **taskman** checks out a license token and manages interactive UNIX applications.

You can use the logical AND operator (**:**) to combine rusage strings and the logical OR operator (**||**) to separate rusage string siblings. For example:

```
taskman -Lp P1 -R "rusage[f1=1:f2=1||f1=5:f3=1||f4=1]" myjob
```

If you specify multiple rusage string siblings, License Scheduler checks each of the rusage string siblings from left to right. If at least one of the rusage string sibling

requirements are met, the task can start. If none of the rusage string sibling requirements are met, License Scheduler sends the DEMAND of all the unsatisfied rusage string siblings.

If a particular unsatisfied resource is specified in multiple rusage string siblings, only the highest value for DEMAND is sent. For example:

```
taskman -Lp P1 -R "rusage[f1=1:f2=2||f1=3:f2=1]" myjob
```

The f1 resource requirement is 1 for the first rusage string sibling and 3 for the second rusage string sibling. If the f1 resource is not satisfied, the demand of f1 is 3, not 3+1. This task will not start until at least one of the requirements of the rusage string siblings is met.

If **LOCAL_TO** is specified for a feature, **taskman** jobs must specify feature names with locality information similar to submission with **bsub**. You must 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
```

Project mode with project groups

Project groups use a ProjectGroup section to build a hierarchical project structure, which you can use to set limits on projects that span multiple clusters.

Depending on your license usage, you can configure different project groups for different license features, or reuse the same hierarchical structure.

Each license feature in project mode can either use projects or project groups. Changing from projects to project groups involves adding a ProjectGroup section and changing the license token distribution that is configured in the Feature section. Other configuration remains the same.

Configuring project groups

ProjectGroup sections use configured projects (each with a Projects section in the lsf.licensescheduler file) to form a hierarchical structure for each feature.

Note:

The Feature section **GROUP** parameter is used to group projects together, simplifying configuration, and is not the same as a ProjectGroup section.

1. Add a ProjectGroup section to the lsf.licensescheduler file:

   ```
   Begin ProjectGroup
   GROUP     SHARES    OWNERSHIP   LIMITS     NON_SHARED
   End Projectgroup
   ```

   If LS_FEATURE_PERCENTAGE=Y or LS_ACTIVE_PERCENTAGE=Y in lsf.licensescheduler, values for **OWNERSHIP**, **LIMITS**, and **NON_SHARED** are expressed as a percentage of the total licenses, not as an absolute number.
2. For each branch in the hierarchy, add a line to the ProjectGroup section.
   a. Under the heading **GROUP**, indicate the project that branches, and direct descendants in the hierarchy (*group*(*member ...*)).
   b. Under the heading **SHARES**, set the integer share for each member project.

c. Under the heading **OWNERSHIP**, set the integer ownership for each bottom-level group member (leaf node), with a dash (-) representing no ownership. The OWNERSHIP value must be greater than or equal to the NON_SHARED value.

d. Under the heading **LIMITS** set the integer license limit for each member project, with a dash (-) representing unlimited. The LIMITS value must be greater than or equal to the OWNERSHIP value.

e. Under the heading **NON_SHARED**, set the integer number of non-shared licenses each bottom-level group member (leaf node) uses exclusively, with '-' representing none.

f. Optionally, under the heading **DESCRIPTION**, add a description up to 64 characters long, using a backslash (\) to extend to multiple lines.

For example, the branch g4 splits into three members:

```
GROUP                SHARES    OWNERSHIP   LIMITS     NON_SHARED
(g4 (p4 p5 p6))      (1 1 1)   (1 1 1)     ()         (- 3 -)
```

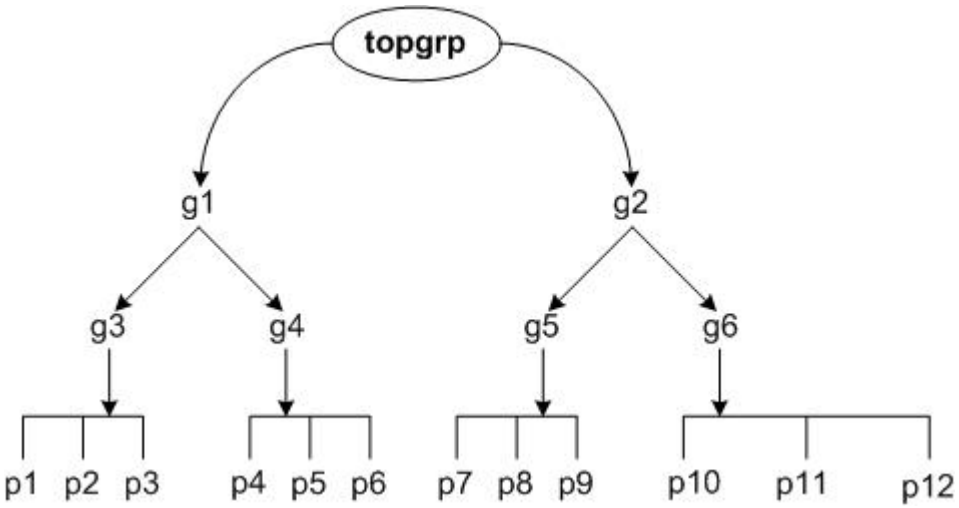
3. In the Feature section, set parameter **GROUP_DISTRIBUTION** to the top level of the ProjectGroup section hierarchy.

   The **DISTRIBUTION** parameter that is used for projects is no longer used.

4. In the Feature section, list service domains in the **SERVICE_DOMAINS** parameter.

   Unlike for projects, service domains are not included in the distribution for project groups.

Project group examples



This hierarchy is implemented by the project group configuration:

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

License feature configuration that uses this project group:

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

Use the **LIMITS** column to limit token use, so tokens are sometimes not distributed even if they are available. By default, License Scheduler distributes all available tokens if possible. For example, if total of six 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 allocates only five tokens according to the distribution. License Scheduler gives three tokens to group `A` and three tokens to group `B`, but project `c` and project `d` are limited to one token each, so one token is not allocated within group `A`. As more demand comes in for project `e` and project `f`, the tokens that are not allocated are distributed to group `B`.

Configuring preemption priority within project groups

The optional **PRIORITY** parameter in the ProjectGroup section, if defined, is used for preemption instead of basing preemption on the accumulated `inuse` for each project.

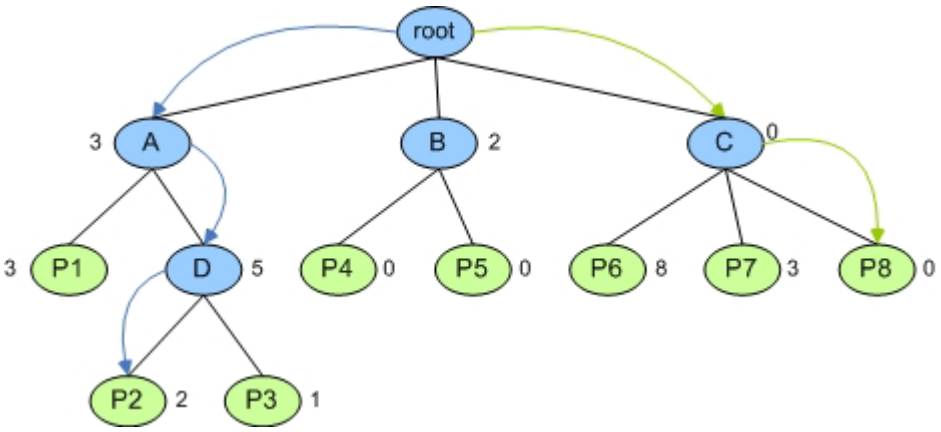
Under the heading **PRIORITY**, set the integer priority for each group member, with '0' being the lowest priority.
PRIORITY can be set for all members in the project group hierarchy.

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 -)
(D (P2 P3))      (1 1)    ()         ()         ()      (2 1)
End ProjectGroup
```

By default, priority is evaluated from top to bottom. The priority of any specific 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.

Viewing hierarchical configuration

Use **blinfo -G** to view the hierarchical configuration:
For the previous example:

```
blinfo -G
```

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

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: 4    OTHERS: 0
SHARE_INFO_FOR:  /topgrp
GROUP/PROJECT           SHARE   OWN  INUSE RESERVE FREE   DEMAND
g2                      100.0 % 0    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: 4     OTHERS: 0
SHARE_INFO_FOR:  /topgrp
```

```
GROUP/PROJECT          SHARE   OWN  INUSE RESERVE FREE   DEMAND
g2                     100.0 % 0    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
```

Configure fast dispatch project mode

Use fast dispatch project mode to increase license utilization for project licenses. Fast dispatch project mode has the scheduling performance of cluster mode with the functionality of project mode, and is most appropriate for your needs if:

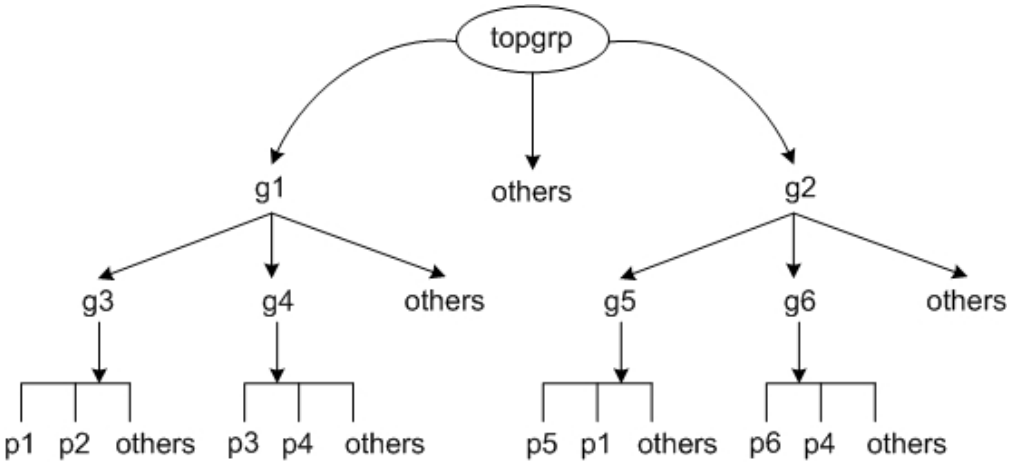
- Your primary goals are to maximize license use and ensure ownership of groups
- Most jobs are short relative to the **blcollect** cycle (60 seconds by default, set by **LM_STAT_INTERVAL**).

In fast dispatch project mode, License Scheduler does not have to run the FlexNet command **lmstat** to verify that a license is free before each job dispatch. As soon as a job finishes, the cluster can reuse its licenses for another job of the same project, which keeps gaps between jobs small. However, because License Scheduler does not run **lmstat** to verify that the license is free, there is an increased chance of a license checkout failure for jobs if the license is already in use by a job in another project.

Hierarchical project group paths

By default, hierarchical project groups in fast dispatch project mode are the same as hierarchical project groups in project mode. Fast dispatch project mode also supports the use of hierarchical project group paths, which helps License Scheduler dispatch more jobs in fast dispatch project mode. To use hierarchical project group paths, you need LSF, Version 9.1.1, or later.

The following hierarchical group structure illustrates hierarchical project group paths:



Enabling hierarchical project group paths enables the following:

- Features can use hierarchical project groups with project and project group names that are not unique, as long as the projects or project groups do not have the same parent. That is, you can define projects and project groups in more than one hierarchical project group.

For example, the p4 project can be defined for project groups g4 and g6, each with its specific resource allocation within the project groups.

Note: Do not define a project group as a child of itself, because this results in a loop. For example, if project group g3 is a child of project group g1, do not define project g1 as a child of g3, as this will result in a loop of g1 and g3 being child project groups of one another.

- When specifying -Lp *license_project*, you can use paths to describe the project hierarchy without specifying the root group.

  For example, if you have topgrp as your root group, which has a child project group named g1 with a child project group named g3, which has a project named p1, you can use -Lp /g1/g3/p1 to specify this project.
- Hierarchical project groups have a default project named others with a default share value of 0. Any projects that do not match the defined projects in a project group are assigned into the others project. If the others project has a share value of 0, this project can still use licenses if the defined projects with shares are not using the licenses. Therefore, by default, the others project has the lowest priority within a project group.

  For example, if you have topgrp as your root group, which has a child project group named g1 with a child project group named g3, which has a project named p1, if you specify -Lp /g1/g3/project3 (which does not match a project), the effective license project is /g1/g3/others project. Similarly, specifying -Lp /g1/g3/gA/gB/gC/project3 results in an effective license project of /g1/g3/others because there are no subsequent child project groups under /g1/g3.

  If there is already a project named others, the preexisting others project specification overrides the default project.

Defining hierarchical project groups for fast dispatch project mode is the same as for project mode, allowing for project and project group names that are not unique.

For example, to define the previously-illustrated hierarchical project group in lsf.licensescheduler:

```
Begin ProjectGroup
GROUP              SHARES  OWNERSHIP LIMITS NON_SHARED
(topgrp (g1 g2)    (1 1)   ()        ()     ()
(g1 (g3 g4 others) (1 1 1) ()        ()     ()
(g2 (g5 g6)        (1 2)   ()        ()     ()
(g3 (p1 p2 others) (2 2 1) ()        ()     ()
(g4 (p3 p4)        (2 1)   ()        ()     ()
(g5 (p5 p1)        (1 3)   ()        ()     ()
(g6 (p6 p4)        (1 1)   ()        ()     ()
End ProjectGroup

Begin Feature
NAME=f1
GROUP_DISTRIBUTION=topgrp
SERVICE_DOMAINS=LanServer
End Feature
```

The others projects are explicitly defined for g1 and g3 (with a specific share), while the other project groups use the default others projects with 0 share.

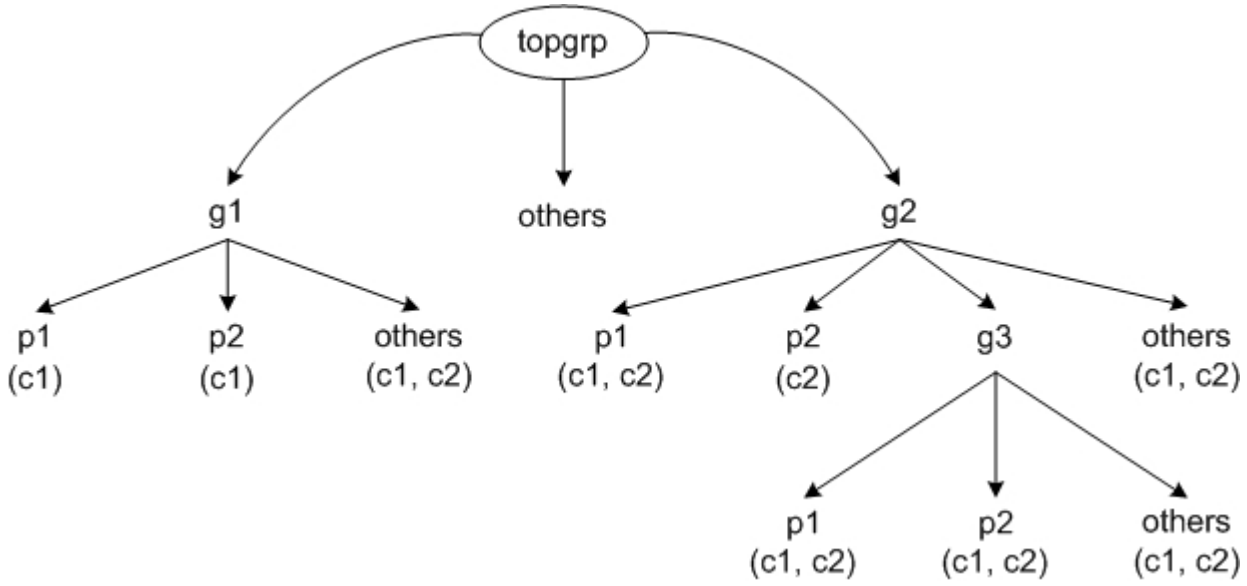
The p1 project is defined for both g3 and g5, with a larger share if specified through the g5 project group. The p4 project is defined for both g4 and g6, with a larger share if specified through the g6 project group.

You can also specify different project groups with different root groups. Different features can use different root groups (as defined by the GROUP_DISTRIBUTION parameter), each with its own project group hierarchy and share policies.

When a job requests multiple features in fast dispatch project mode, License Scheduler generates an effective license project for each feature. This means that it is possible for one job to have multiple effective license projects if the features use different project group hierarchies. License Scheduler and LSF will calculate the effective license project for the feature based on its related project group hierarchy. The effective project is the path of the project resulting from the `-Lp` specification.

When specifying a project name without a hierarchical project group path in fast dispatch project mode with hierarchical group paths enabled, License Scheduler uses the shortest path to the left that ends with the name of the project, as long as the cluster that submitted the job is authorized to use the selected project. If License Scheduler cannot find such a project in the hierarchy, License Scheduler uses the `/others` project.

For example, the following hierarchical group structure illustrates which clusters (c1 and c2) are authorized to use each project:



If you specify `-Lp p2` from the `c2` cluster (by submitting `bsub -Lp p2 -R "rusage[f1=1]" myjob`) without specifying a hierarchical group path, `c2` is authorized to use `/g2/p2` and `/g2/g3/p2`. The shortest path to the left that leads to `p2` is `/g2/p2`, so the job is associated with the `/g2/p2` hierarchical project.

Configure parameters

Before configuring fast dispatch project mode, ensure that you enabled and configured project mode using projects or project groups. However, you can only specify one service domain per feature in fast dispatch project mode.

1. Fast dispatch project mode can be set globally, or for individual license features. Set individually when using fast dispatch project mode for some features and cluster mode or project mode for other features.
   a. If you are using fast dispatch project mode for all license features, define `FAST_DISPATCH=Y` in the Parameters section of `lsf.licensescheduler`.

b. If you are using fast dispatch project mode for some license features, define `FAST_DISPATCH=Y` for individual license features in the Feature section of `lsf.licensescheduler`.

   The Feature section setting of **FAST_DISPATCH** overrides the global Parameter section setting.

2. Set the limit to which License Scheduler considers the demand by each project in each cluster when allocating licenses.

   The default is 5.

   `DEMAND_LIMIT=`*integer*

   Define DEMAND_LIMIT in the Parameters section of `lsf.licensescheduler` to set the limit for all license features, or define DEMAND_LIMIT in the Feature section for individual license features. Setting in the Feature section overrides the global setting in the Parameters section.

   Periodically, each cluster sends a demand for each project. This is calculated in a cluster for a project by summing up the rusage of all jobs of the project pending due to lack of licenses. Whether to count a job's rusage in the demand depends on the job's pending reason. In general, the demand reported by a cluster only represents a potential demand from the project. It does not take into account other resources that are required to start a job. For example, a demand for 100 licenses is reported for a project. However, if License Scheduler allocates 100 licenses to the project, the project does not necessarily use all 100 licenses due to slot available, limits, or other scheduling constraints.

   mbatchd in each cluster sends a demand for licenses from each project. In fast dispatch project mode, **DEMAND_LIMIT** limits the amount of demand from each project in each cluster that is considered when scheduling.

3. To enable hierarchical project group paths, define `PROJECT_GROUP_PATH=Y` in the Parameters section of `lsf.licensescheduler`.

   Note: To use **PROJECT_GROUP_PATH**, you need LSF, Version 9.1.1, or later.

Restart to implement configuration changes

1. Run `bladmin reconfig` to restart the bld.
2. If you deleted any Feature sections, restart mbatchd. In this case, a message is written to the log file, prompting the restart.

   If required, run `badmin mbdrestart` to restart each LSF cluster.

View license allocation

Run `blstat -c` *token_name* to view information for a specific license token (as configured in a Feature section).
blstat -c output differs for fast dispatch project mode, project mode, and cluster mode.

Configure lmremove preemption

Enable and configure **lmremove** as a preemption action.

Preemption is enabled by configuring license ownership for a project. When a project has ownership of licenses that are occupied by another project, these licenses can be preempted by the project that has ownership when it needs to use the licenses.

```
Begin Feature
NAME = lic1
FAST_DISPATCH= Y
DISTRIBUTION = serviceDomain1(projectA 1/10 projectB 1 )
End Feature
```

The default preemption action is to send a TSTP signal to the job. Some applications will respond well to this action, and will free up their licenses and suspend their processes. If your applications respond well to the TSTP signal, leave this default as the preemption action.

For applications that do not respond well to the TSTP signal, an alternative preemption action for projects in fast dispatch project mode is to suspend the job's processes, then use **lmremove** to remove licenses from the application. **lmremove** causes **lmgrd** and vendor daemons to close the TCP connection with the application. Once the application is resumed, it will try to reacquire the licenses. In general, **lmremove** will fail to remove licenses from an application for a period of time after the licenses are checked out. This period depends on the application itself.

When License Scheduler calls **lmremove**, it may remove licenses from running jobs when the running jobs share the same user and host as a suspended job. License Scheduler will continue to reserve the licenses (from the rusage) for the running job. Therefore, when the running job tries to reacquire its licenses, there will be licenses available for it.

License Scheduler calls **lmremove** in the same directory as it calls the **lmstat** command (as defined in the **LMSTAT_PATH** parameter).

1. Enable lmremove as a preemption action by specifying the **LMREMOVE_SUSP_JOBS** parameter in the Parameters or Feature section of lsf.licensescheduler.

   LMREMOVE_SUSP_JOBS = *seconds*

   Set this parameter for a license feature in its corresponding Feature section as long as it is using the fast dispatch project mode. If you set this parameter in the Parameters section, this applies to all license features using the fast dispatch project mode. When this parameter is configured for a license feature, License Scheduler will periodically use **lmremove** to try to remove the license feature from each recently-suspended job.

   For a given application, set **LMREMOVE_SUSP_JOBS** to a value greater than the period following a license checkout that **lmremove** will fail for that application. In this way, you can be sure that when a job is suspended, its licenses will be released. The length of this period depends on the application.

   License Scheduler will continue to try removing the license feature for the specified number of seconds after the job is first suspended.

   For example, if you define LMREMOVE_SUSP_JOBS = 10, when a job is suspended due to preemption,License Scheduler will continue to try removing the license feature for up to ten seconds after the job is first suspended.

2. Enable License Scheduler to preempt a job immediately after a license checkout by defining LM_REMOVE_INTERVAL = 0 in the Parameters section of lsf.licensescheduler.

   LM_REMOVE_INTERVAL = 0

   Defining this parameter to a larger value prevents License Scheduler from preempting a job for a period of time after License Scheduler first detects a license checkout by the job (the default value is 180 seconds). Defining LM_REMOVE_INTERVAL = 0 ensures that License Scheduler can preempt a job

immediately after checkout. After the job is suspended, License Scheduler calls **lmremove** to release licenses from the job.

3. To limit the amount of time between subsequent forks of child processes to run **lmremove**, define the **LMREMOVE_SUSP_JOBS_INTERVAL** parameter in the Parameters or section of lsf.licensescheduler.

   LMREMOVE_SUSP_JOBS_INTERVAL = *seconds*

   By default, License Scheduler forks a child process to run **lmremove** every time it receives an update from a license collector daemon (**blcollect**). Defining this parameter controls the minimum amount of time between subsequent forks.

Automatic time-based configuration

Variable time-based configuration is used in both project mode and cluster mode to automatically change configuration that is set 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 that are 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 License Scheduler every 10 minutes based on **bld** start time. When an expression evaluates true, License Scheduler dynamically changes the configuration that is based on the associated configuration statements and restarts **bld**

The #if, #else, #endif keywords are not interpreted as comments by License Scheduler, but as if-else constructs.

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.

Specify time values

Specify at least the hour.
Day and minutes are optional.

Specify time windows

Specify two time values that are 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*

For example:

- 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
  ```

Specify time expressions

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

Define a time expression.

A time expression is made up of the time keyword followed by one or more space-separated time windows that are enclosed in parentheses. Use the `&&`, `||`, and `!` logical operators to combine time expressions.

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

For 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)
```

Create if-else constructs

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

- Define an if-else expression.

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

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

- Define an elif expression.

  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.

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

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

Restart to implement configuration changes

All time-based configuration is within the lsf.licensescheduler file, so restarting the **bld** applies all changes.

1. Run bladmin ckconfig to check configuration.
2. Run lsadmin limrestart or bladmin restart to restart the **bld**.

Verify configuration

Verify time-based configuration by viewing License Scheduler information.

1. Run blinfo.
2. Run blstat.

Examples

Project configuration in project mode

```
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
```

Project group configuration in project mode

```
#
# 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
```

Cluster distribution configuration in cluster mode

```
Begin Feature
NAME = f1
#if time(5:16:30-1:8:30 20:00-8:30)
CLUSTER_DISTRIBUTION=Wan(C11 1 C12 1)
#elif time(3:8:30-3:18:30)
CLUSTER_DISTRIBUTION= Wan(C11 2 C12 1/2/100) Lan(C12 1)
#else
CLUSTER_DISTRIBUTION= Wan(C11 10 C12 1/1/10) Lan(C11 1)
#endif
End Feature
```

Failover

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.

failover host

A master candidate host that runs the License Scheduler daemon (**bld**), and can take over license management if the master License Scheduler host fails or loses its connection to the network (in either a LAN or WAN environment).

failover provisioning

The configuration of a list of failover hosts in the event of a host failure or network breakdown. License Scheduler can be configured for failover provisioning in both LANs and WANs.

Failover provisioning for LANs

Configuring failover ensures enhanced performance and reliable license distribution.

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

Define the list of License Scheduler hosts in `LSF_CONFDIR/lsf.conf` and `lsf.licensescheduler` for your LAN (Designer Center A in this example).

1. `lsf.conf`: Specify a space-separated list of hosts for the **LSF_LIC_SCHED_HOSTS** parameter:

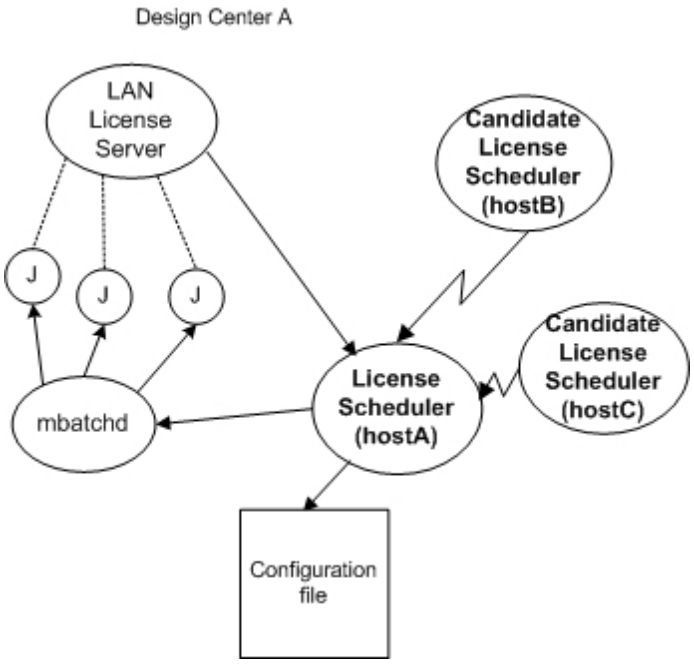
   ```
   LSF_LIC_SCHED_HOSTS="hostA.designcenter_a.com hostB.designcenter_a.com
   hostC.designcenter_a.com"
   ```

   Tip: List the hosts in order of preference for running License Scheduler, from most preferred to least preferred.
2. `lsf.licensescheduler`: Specify a space-separated list of hosts for the HOSTS parameter:

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

   List the hosts in the same order as `lsf.conf`.

The LIM starts the **bld** (License Scheduler daemon) on each host in the **LSF_LIC_SCHED_HOSTS** list.
Every host in defined in **LSF_LIC_SCHED_HOSTS** is a failover candidate and runs the **bld** daemon.



- `hostA.designcenter_a.com` is the License Scheduler host, and the remaining hosts are candidate hosts that are running the **bld** daemon, ready to take over the management of the licenses if there is a network failure
- Each host contains the list of candidate hosts in memory
- Each candidate License Scheduler host communicates with the License Scheduler host, License Scheduler (`hostA`)
- If the License Scheduler host fails, each candidate host checks to see if a more eligible host is running the License Scheduler daemon. If not, it becomes the failover host and inherits the communication links that existed between the original 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 need only one host to run License Scheduler, but you can configure your site for a failover mechanism with multiple candidate hosts to take over the scheduling in a failure.

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

Configure and start License Scheduler in a WAN

In a WAN configuration:

1. As the root user, install License Scheduler on each cluster in the WAN configuration and select one cluster to be the main 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`:
   a. 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 License Scheduler.

      hostname_n is the least preferred host for running License Scheduler.

   b. In the Clusters section, specify the names of the clusters in the WAN.

      For example:

      ```
      Begin Clusters
      CLUSTERS
      design_SJ
      design_BOS
      End Clusters
      ```

4. 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 License Scheduler daemon (**bld**).

   The first host that is listed in the HOSTS list is the default master License Scheduler host for the WAN.

   The order of the host names in LSF_LIC_SCHED_HOSTS is ignored.

5. In the other clusters in the WAN:
   a. Configure the LSF_LIC_SCHED_HOSTS parameter in `lsf.conf` with a local list of candidate hosts.
   b. 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.

6. In the cluster that contains the WAN license server and the other clusters in the WAN, run the following commands:
   a. Run **bld -C** to test for configuration errors.
   b. Run **bladmin reconfig** to configure License Scheduler.
   c. Run `lsadmin reconfig` to reconfigure LIM.
   d. Use `ps -ef` to make sure that **bld** is running on the candidate hosts.
   e. 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:

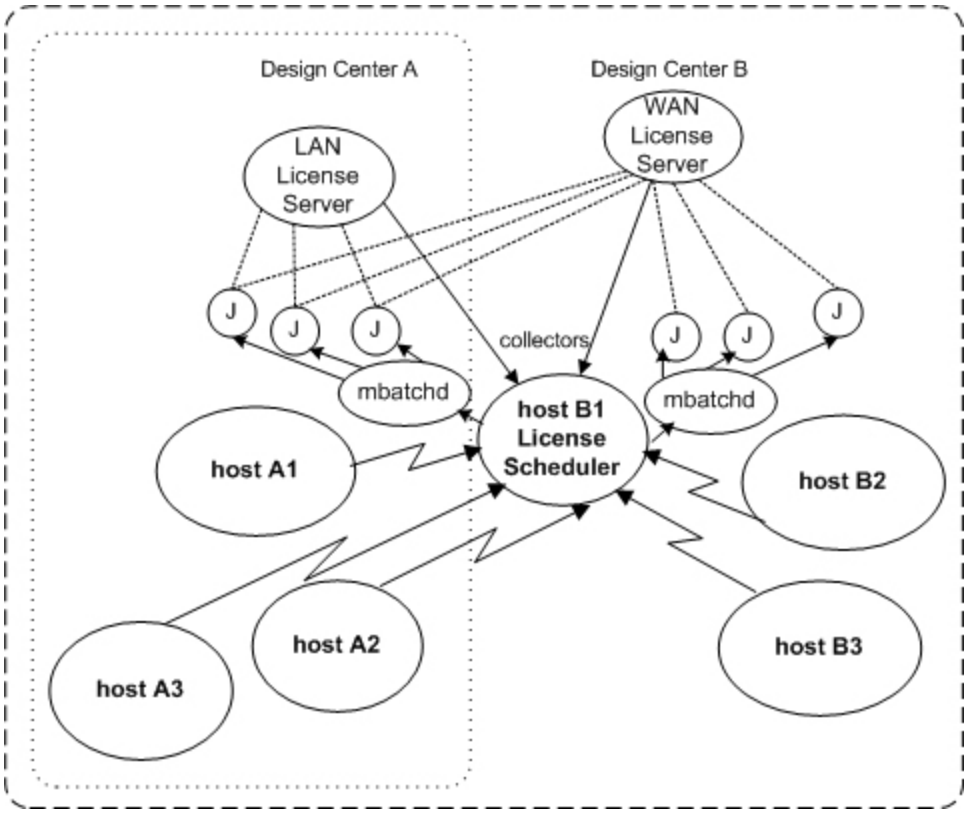
- `lsf.licensescheduler` in Design Center A

  ```
  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 License Scheduler daemon (**bld**) on each host that is 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 License Scheduler candidate in Design Center A and is running the **bld** daemon, but only one host becomes the 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:

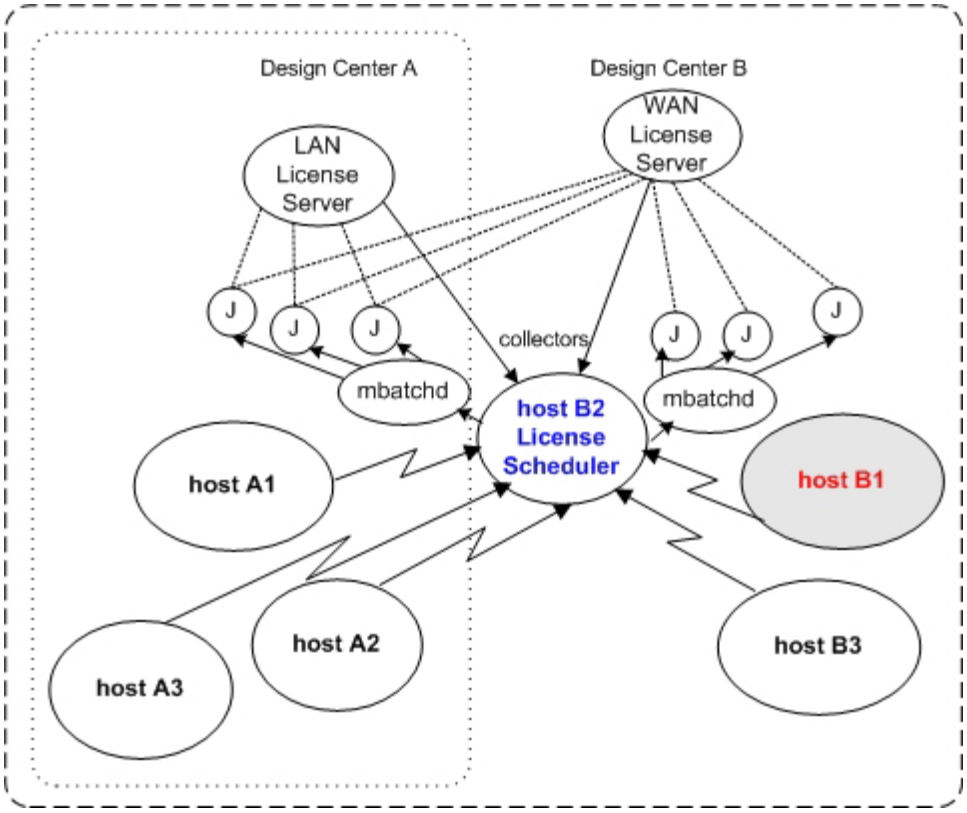
- Both design centers list `hostB1.designcenter_b.com` at the top of their **HOSTS** lists.
- `hostB1.designcenter_b.com` is the License Scheduler host for Design Center A and for Design Center B.
- The rest of the hosts in both design centers remain on standby as candidate License Scheduler hosts.
-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

Service provisioning at the host and network levels

In the following example configuration, there are two potential points of failure: host and network.

Host failure

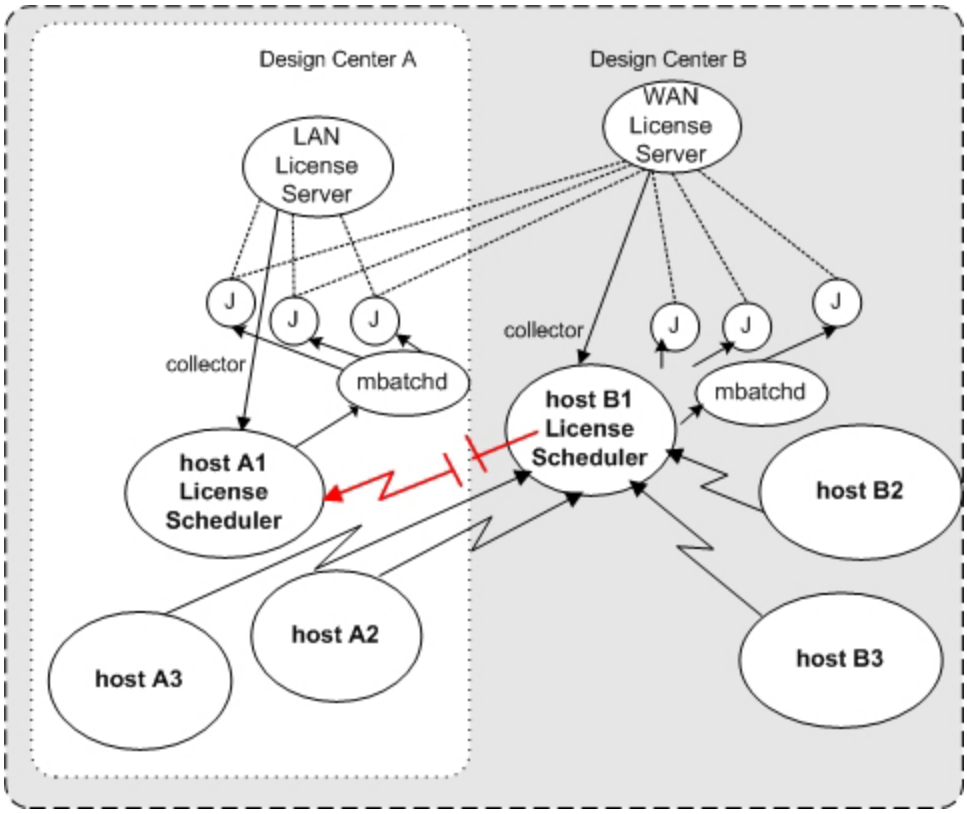
If `hostB1.designcenter_b.com` fails, and **`bld`** stops running, a candidate License Scheduler host must take over the license management. The next host on the HOSTS list in both Design Center A and Design Center B is `hostB2designcenter_b.com`. License Scheduler fails over to this host if it is up 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 if 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.



Set up fod

The **fod** daemon manages failover for the **blcollect** daemons. **fod** can restart any failed **blcollect** processes if the local host (and thus the local **fod**) is down. The failover host **fod** starts new **blcollect** daemons until the primary host comes back online and the primary **fod** contacts the secondary **fod**.

The fod files are in the License Scheduler package, but must be copied, configured, and started manually.

1. Install the failover daemon (**fod**) files on each host.
   a. Create a directory to hold the **fod** files, with subdirectories `bin`, `conf`, `etc`, and `man`.

      For example: `/usr/local/fod`
   b. Copy all user command files and the `fod.shell` file to `.../bin`.
   c. Copy the `fod.conf` file to `.../conf`.
   d. Copy the `fod` file to `.../etc`.
   e. Copy the `fodapps.1`, `fodhosts.1` and `fodid.1` files to `.../man/man1`.
   f. Copy the `fod.conf.5` file to `.../man/man5`.
   g. Copy the `fodadmin.8` file to `.../man/man8`.
2. Edit the `fod.shell` file, and set the **FOD_ROOT** parameter to the name of your new directory.

   For example: `FOD_ROOT=/usr/local/fod`
3. Set the environment variables.
   a. Set the **PATH** environment variable to include the `/bin` directory.
   b. Set the **FOD_ENVDIR** environment variable to `$FOD_ROOT/conf`.

c. Set the **MANPATH** environment variable to include the /man directory.

4. In fod.conf, set the required parameters.
   - **FOD_ADMIN**: The License Scheduler administrator
   - **FOD_PORT**: The TCP listening port and UDP port for the failover daemon
   - **FOD_WORK_DIR**: The working directory
   - **FOD_LOG_DIR**: The log directory

   For example:

   ```
   FOD_CLUSTERNAME = fod
   FOD_ADMIN = lsadmin
   FOD_PORT = 9583
   FOD_WORK_DIR = /usr/local/fod/work
   FOD_LOG_DIR = /usr/local/fod/work
   ```

5. In the **Hosts** section of fod.conf, specify the hosts where the failover daemons run.

   If your hosts run in different DNS domains, you must use a fully qualified domain name when you specify the host name. The first host in the **Hosts** section is the first host on which the failover daemon runs (the master failover daemon host).

   For example:

   ```
   Begin Hosts
   HOSTNAME
   fodhost1.domain_name
   fodhost2
   End Host
   ```

6. Modify the **Applications** section of fod.conf.

```
Begin Applications
NAME         PATH                                    PARAMS     FATAL_EXIT_VALUE
blcollect   /pcc/apps/lsf6/6.0/sparc-sol7-64/etc   (-2 -m "sasun3 augustus claudius" -p 9581 -c lan -i 20
-D /sparc-sol7-64/etc)   (-)
End Applications
```

7. Start **fod** on each host.
   a. Log on as the Platform License Scheduler administrator.
   b. Source the LSF environment.
      - For **csh** or **tsh** run source LSF_TOP/conf/cshrc.lsf
      - For **sh**, **ksh**, or **bash**, run . LSF_TOP/conf/profile.lsf
   c. Launch the failover daemons by running the fod.shell file.

      Check the progress of a successful launch by running **ps -ef**.

      View the **fod** log under **$LSF_LOGDIR**.

      Check configuration from $FOD_ROOT/etc by running **fod -C**.

User authentication

When a user claims a job belongs to 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, License Scheduler allows the license request.
- If the user does not belong to the project or the project does not match any projects in the configuration, License Scheduler rejects the request.
- If a default project is configured in the License Scheduler user authentication configuration file `ls.users`, License Scheduler changes the project to default and allows the license request.
- If the project is default, no authentication is needed and License Scheduler allows the request.

Enable user authentication

1. To enable user authentication for LSF jobs, configure LSF to use authentication esub (**esub.ls_auth**).

   Define `LSB_ESUB_METHOD=lsauth` in `lsf.conf`.
2. To enable user authentication for **taskman** jobs, define `AUTH=Y` in `lsf.licensescheduler`.
3. Configure users and their associated projects in the `LSF_CONFDIR/ls.users` file.

   The file defines one project per line using the following format:

   project_name`:::[`*user_name*`][,user_name2 ...]`

   For example,

   ```
   Project1:::user1,user2
   default:::
   ```

   Note: Ensure that projects in `ls.users`, including the default project, conform to the `lsf.licensescheduler` configuration.

Chapter 5. Viewing information and troubleshooting

About viewing available licenses

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

The licenses in use are checked out from FlexNet by your projects. Free licenses and licenses that are reserved by a project are not yet 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** increases and the TOTAL_FREE count decreases. The number of licenses for each project changes whenever LSF redistributes license tokens among competing projects.

View license server and license feature information passed to jobs

You can display the license servers that are used by each service domain that is allocated to the license features.

Run `blstat -S`.

```
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
```

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.

View license usage

Run **blstat -s** to display license usage.

```
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
```

If there are any distribution policy violations, **blstat** marks these violations with an asterisk (*) at the beginning of the line.

View workload distribution information

Run **blinfo -a** to display WORKLOAD_DISTRIBUTION information.

```
blinfo -a
FEATURE      MODE       SERVICE_DOMAIN  TOTAL  DISTRIBUTION
g1           Project    LS              10     [p1, 50.0%] [p2, 50.0%]
                                WORKLOAD_DISTRIBUTION
                                [LSF 66.7%, NON_LSF 33.3%]
```

Sort license feature information

You can sort license feature information alphabetically, by total licenses, or by available licenses.

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

- Sort alphabetically:

  ```
  blstat -o alpha
  ```

- Sort by total licenses:

  ```
  blstat -o total
  ```

  The feature with the largest number of total licenses displays first.

- Sort by available licenses:

  ```
  blstat -o avail
  ```

  The feature with the largest number of available licenses displays first.

  You can also run **blstat -o** with options **-Lp**, **-t**, **-D**, **-G**, **-s**, **-S**.

  Note:

  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 that are configured to supply licenses to the feature. Licenses that are borrowed by non-LSF workload are subtracted from this sum.

  – Options**-s**, **-S**: Total licenses means all the licenses (supplied by the license vendor daemon) from all the service domains that are configured to supply licenses to that feature.

Limitations with viewing multiple jobs running on an execution host

If there are multiple jobs submitted by a user that run on the same execution host, **blstat** might not display the correct license usage information. This is because **lmstat** only provides the user and host information of each license checkout, but does not provide additional information for License Scheduler to match the license checkout to a specific LSF job.

License Scheduler attempts to match the license checkout to each LSF job based on the user, execution host, and rusage string. If the multiple jobs running on the same execution host are submitted by the same user and request the same license, the information that **lmstat** provides is insufficient for License Scheduler to provide an exact match for each LSF job. License Scheduler estimates the job, but this may be incorrect.

For example,

- There are multiple service domains providing the same feature and a user submits multiple jobs that run on the same execution host.

  Although License Scheduler dispatches the tokens correctly, **blstat** might not show the correct token usage (such as TOTAL_INUSE, TOTAL_RESERVE, or TOTAL_FREE). Incorrect tokens are counted in OTHERS.
- There is one license server with multiple projects and a user submits multiple jobs with some jobs reserving tokens for a time. The jobs that reserve tokens are running on the same host as other License Scheduler jobs.

  Although License Scheduler dispatches the tokens correctly, **blstat** may show reversed token usage, so that some INUSE tokens are counted in RESERVED, and some RESERVED tokens are counted in INUSE.

About error logs

Error logs maintain important information about License Scheduler operations.

Tip: Log files grow over time. Occasionally clear or back up these files (manually or using automatic scripts).

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

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

The error log file names for the LSF License Scheduler system daemons are:

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

About blcollect log messages

Messages that are 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 that gathered successfully or not.
- Detailed information: The number of tokens, the name of tokens, the license server name for license tokens that are collected by **blcollect**.

Manage log files

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

1. Set **LS_LOG_MASK** in `lsf.licensescheduler` to the wanted logging level.

   Note:

   If LS_LOG_MASK is not defined, the value of LSF_LOG_MASK in `lsf.conf` is used. If LS_LOG_MASK or LSF_LOG_MASK are not defined, the default is LOG_WARNING.

   Log levels (highest to lowest):

   - LOG_WARNING: Default. Essential error messages only.

- LOG_DEBUG: Fewest number of debug messages, useful for debugging a problem.
- LOG_DEBUG1: More debug messages than LOG_DEBUG.
- LOG_DEBUG2: Most frequently used debug level.
- LOG_DEBUG3: All debug messages. Use sparingly.

Messages that are logged at the specified level and higher are recorded, while lower-level messages are discarded.

2. Clean up or back up log files periodically.

Temporarily change the log level

You must submit the commands from the host on which the daemon is running (only applicable to the **bld**).

You can temporarily change the class or message log level for the **bld** and **blcollect** daemons without changing `lsf.licensescheduler`.

The message log level that you set is 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 that is specified by **LSF_LOGDIR**.

1. Set the log level for the **bld**.

   `bladmin blddebug [-l` *debug_level*`] [-c` *class_name*`]`

   For example:

   ```
   bladmin blddebug -l 1 -c "LC_TRACE LC_FLEX"
   ```

   Logs messages for **bld** running on the local host and sets the log message level to LOG_DEBUG1. The log class is LC_TRACE LC_FLEX.

2. Set the log level for **blcollect**.

   `bladmin blcdebug [-l` *debug_level*`]` *collector_name* `... | all`

   For example:

   ```
   bladmin blcdebug -l 3 all
   ```

   The log mask of all collectors is changed to LOG_DEBUG3.

3. Return the debug settings to their configured values (set with **LS_LOG_MASK** in `lsf.licensescheduler`).

   ```
   bladmin blddebug -o
   bladmin blcdebug -o
   ```

For a detailed description of these commands and their options, see the *IBM Platform LSF Command Reference*.

Troubleshooting

Techniques

- Run **blstat** to check the current license usage information.
- Run **blusers** to check the current job and license usage. This information is the set intersection of License Scheduler Jobs and FlexNet information.
- Run **blinfo** command to check the current License Scheduler configuration.
- Run **BLD -C** to check that the configuration is correct. This action, with **LOG_DEBUG**, writes detailed configuration settings to the debug log.
- Turn on debugging by setting **LSF_LOG_MASK=LOG_DEBUG** and reconfiguring the daemon with **bladmin reconfig all**.

- Set the log class for **mbatchd** debug (**LSB_DEBUG_MBD**) in `lsf.conf`: **LC_LICSCHED**.
- Use **LSB_TIME_SCH=timelevel** (similar to **LSB_TIME_MBD**) in `lsf.conf` to enable the logging of timing information.
- Run **bhosts -s** to check that the resources are being reported correctly to LSF.

File locations

- **BLD** logs are in the standard `$LSF_LOGDIR`.
- **BLCOLLECT** logs are in `/tmp` or `$LSF_LOGDIR` on the hosts the daemon is running.
- Core files from **BLD**, **BLCOLLECT**, **mbatchd**, **lim**, and **mbsched** are in `/tmp` on the daemon local hosts.

Check that lmstat is supported by blcollect

1. Create shell script to output (for example, echo) target **lmstat** output.
2. Point **LMSTAT_PATH** in `lsf.licensescheduler` to the shell script.
3. If **LIC_COLLECTOR** is not set, restart the **bld** to restart **blcollect**. If **LIC_COLLECTOR** is set, kill **blcollect** and restart **blcollect** manually.
4. Observe the **blcollect** log to view if there are any errors to determine whether **blcollect** is able to parse **lmstat** output properly.

Chapter 6. Reference

lsf.licensescheduler

The `lsf.licensescheduler` file contains License Scheduler configuration information. All sections except ProjectGroup are required. In cluster mode, the Project section is also not required.

Changing lsf.licensescheduler configuration

After making any changes to `lsf.licensescheduler`, run the following commands:

- **`bladmin reconfig`** to reconfigure **`bld`**
- If you made the following changes to this file, you may need to restart mbatchd:
  - Deleted any feature.
  - Deleted projects in the DISTRIBUTION parameter of the Feature section.

  In these cases a message is written to the log file prompting the restart.

  If you have added, changed, or deleted any Feature or Projects sections, you may need to restart **`mbatchd`**. In this case a message is written to the log file prompting the restart.

  If required, run `badmin mbdrestart` to restart each LSF cluster.

Parameters section

Description

Required. Defines License Scheduler configuration parameters.

Parameters section structure

The Parameters section begins and ends with the lines `Begin Parameters` and `End Parameters`. Each subsequent line describes one configuration parameter. Mandatory parameters are as follows:

```
Begin Parameters
ADMIN=lsadmin
HOSTS=hostA hostB hostC
LMSTAT_PATH=/etc/flexlm/bin
LM_STAT_INTERVAL=30
PORT=9581
End Parameters
```

Parameters

- ADMIN
- AUTH
- BLC_HEARTBEAT_FACTOR
- CHECKOUT_FROM_FIRST_HOST_ONLY
- CLUSTER_MODE
- DEMAND_LIMIT
- DISTRIBUTION_POLICY_VIOLATION_ACTION
- ENABLE_INTERACTIVE
- FAST_DISPATCH

- HEARTBEAT_INTERVAL
- HEARTBEAT_TIMEOUT
- HIST_HOURS
- HOSTS
- INUSE_FROM_RUSAGE
- LIB_CONNTIMEOUT
- LIB_RECVTIMEOUT
- LM_REMOVE_INTERVAL
- LM_STAT_INTERVAL
- LM_STAT_TIMEOUT
- LMREMOVE_SUSP_JOBS
- LMREMOVE_SUSP_JOBS_INTERVAL
- LMSTAT_PATH
- LOG_EVENT
- LOG_INTERVAL
- LS_DEBUG_BLC
- LS_DEBUG_BLD
- LS_ENABLE_MAX_PREEMPT
- LS_LOG_MASK
- LS_MAX_STREAM_FILE_NUMBER
- LS_MAX_STREAM_SIZE
- LS_MAX_TASKMAN_PREEMPT
- LS_MAX_TASKMAN_SESSIONS
- LS_STREAM_FILE
- LS_PREEMPT_PEER
- MBD_HEARTBEAT_INTERVAL
- MBD_REFRESH_INTERVAL
- MERGE_BY_SERVICE_DOMAIN
- PEAK_INUSE_PERIOD
- PORT
- PREEMPT_ACTION
- PROJECT_GROUP_PATH
- REMOTE_LMSTAT_PROTOCOL
- STANDBY_CONNTIMEOUT

ADMIN

Syntax

`ADMIN=`*user_name* ...

Description

Defines the License Scheduler administrator using a valid UNIX user account. You can specify multiple accounts.

Used for both project mode and cluster mode.

AUTH
Syntax

```
AUTH=Y
```

Description

Enables License Scheduler user authentication for projects for **taskman** jobs.

Used for both project mode and cluster mode.

BLC_HEARTBEAT_FACTOR
Syntax

```
BLC_HEARTBEAT_FACTOR=integer
```

Description

Enables **bld** to detect **blcollect** failure. Defines 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.

Used for both project mode and cluster mode.

Default

3

CHECKOUT_FROM_FIRST_HOST_ONLY
Syntax

```
CHECKOUT_FROM_FIRST_HOST_ONLY=Y
```

Description

If enabled, License Scheduler to only consider `user@host` information for the first execution host for a parallel job when merging the license usage data. Setting in individual Feature sections overrides the global setting in the Parameters section.

If disabled, License Scheduler attempts to check out `user@host` keys in the parallel job constructed using the user name and all execution host names, and merges the corresponding checkout information on the service domain if found. In addition, if `MERGE_BY_SERVICE_DOMAIN=Y` is defined, License Scheduler merges multiple `user@host` data for parallel jobs across different service domains.

Default

Undefined (N).License Scheduler attempts to check out `user@host` keys in the parallel job constructed using the user name and all execution host names, and merges the corresponding checkout information on the service domain if found.

CLUSTER_MODE
Syntax

```
CLUSTER_MODE=Y
```

Description

Enables cluster mode (instead of project mode) in License Scheduler. Setting in individual Feature sections overrides the global setting in the Parameters section.

Cluster mode emphasizes high utilization of license tokens above other considerations such as ownership. License ownership and sharing can still be configured, but within each cluster instead of across multiple clusters. Preemption of jobs (and licenses) also occurs within each cluster instead of across clusters.

Cluster mode was introduced in License Scheduler 8.0. Before cluster mode was introduced, project mode was the only choice available.

Default

Not defined (N). License Scheduler runs in project mode.

DEMAND_LIMIT

Syntax

`DEMAND_LIMIT=`*integer*

Description

Sets a limit to which License Scheduler considers the demand by each project in each cluster when allocating licenses. Setting in the Feature section overrides the global setting in the Parameters section.

Used for fast dispatch project mode only.

When enabled, the demand limit helps prevent License Scheduler from allocating more licenses to a project than can actually be used, which reduces license waste by limiting the demand that License Scheduler considers. This is useful in cases when other resource limits are reached, License Scheduler allocates more tokens than Platform LSF can actually use because jobs are still pending due to lack of other resources.

When disabled (that is, `DEMAND_LIMIT=0` is set), License Scheduler takes into account all the demand reported by each cluster when scheduling.

`DEMAND_LIMIT` does not affect the `DEMAND` that **`blstat`** displays. Instead, **`blstat`** displays the entire demand sent for a project from all clusters. For example, one cluster reports a demand of 15 for a project. Another cluster reports a demand of 20 for the same project. When License Scheduler allocates licenses, it takes into account a demand of five from each cluster for the project and the `DEMAND` that **`blstat`** displays is 35.

Periodically, each cluster sends a demand for each project. This is calculated in a cluster for a project by summing up the rusage of all jobs of the project pending due to lack of licenses. Whether to count a job's rusage in the demand depends on the job's pending reason. In general, the demand reported by a cluster only represents a potential demand from the project. It does not take into account other resources that are required to start a job. For example, a demand for 100 licenses is reported for a project. However, if License Scheduler allocates 100 licenses to the project, the project does not necessarily use all 100 licenses due to slot available, limits, or other scheduling constraints.

In project mode and fast dispatch project mode, **mbatchd** in each cluster sends a demand for licenses from each project. In project mode, License Scheduler assumes that each project can actually use the demand that is sent to it. In fast dispatch project mode, **DEMAND_LIMIT** limits the amount of demand from each project in each cluster that is considered when scheduling.

Default

5

DISTRIBUTION_POLICY_VIOLATION_ACTION

Syntax

DISTRIBUTION_POLICY_VIOLATION_ACTION=(PERIOD *reporting_period* CMD *reporting_command*)

reporting_period

Specify the keyword PERIOD with a positive integer representing the interval (a multiple of LM_STAT_INTERVAL periods) at which License Scheduler checks for distribution policy violations.

reporting_command

Specify the keyword CMD with the directory path and command that License Scheduler runs when reporting a violation.

Description

Optional. Defines how License Scheduler handles distribution policy violations. Distribution policy violations are caused by non-LSF workloads; License Scheduler explicitly follows its distribution policies.

License Scheduler reports a distribution policy violation when the total number of licenses given to the LSF workload, both free and in use, is less than the LSF workload distribution specified in WORKLOAD_DISTRIBUTION. If License Scheduler finds a distribution policy violation, it creates or overwrites the LSF_LOGDIR/bld.violation.*service_domain_name*.log file and runs the user command specified by the CMD keyword.

Used for project mode only.

Example

The LicenseServer1 service domain has a total of 80 licenses, and its workload distribution and enforcement is configured as follows:

```
Begin Parameter
...
DISTRIBUTION_POLICY_VIOLATION_ACTION=(PERIOD 5 CMD /bin/mycmd)
...
End Parameter
```

```
Begin Feature
NAME=ApplicationX
DISTRIBUTION=LicenseServer1(Lp1 1 Lp2 2)
WORKLOAD_DISTRIBUTION=LicenseServer1(LSF 8 NON_LSF 2)
End Feature
```

According to this configuration, 80% of the available licenses, or 64 licenses, are available to the LSF workload. License Scheduler checks the service domain for a violation every five scheduling cycles, and runs the **/bin/mycmd** command if it finds a violation.

If the current LSF workload license usage is 50 and the number of free licenses is 10, the total number of licenses assigned to the LSF workload is 60. This is a violation of the workload distribution policy because this is less than the specified LSF workload distribution of 64 licenses.

ENABLE_INTERACTIVE

Syntax

```
ENABLE_INTERACTIVE=Y
```

Description

Optional. Globally enables one share of the licenses for interactive tasks.

Tip:

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

Used for project mode only.

FAST_DISPATCH

Syntax

```
FAST_DISPATCH=Y
```

Description

Enables fast dispatch project mode for the license feature, which increases license utilization for project licenses. Setting in the Feature section overrides the global setting in the Parameters section.

Used for project mode only.

When enabled, License Scheduler does not have to run the FlexNet command **lmstat** to verify that a license is free before each job dispatch. As soon as a job finishes, the cluster can reuse its licenses for another job of the same project, which keeps gaps between jobs small. However, because License Scheduler does not run **lmstat** to verify that the license is free, there is an increased chance of a license checkout failure for jobs if the license is already in use by a job in another project.

The fast dispatch project mode supports the following parameters in the Feature section:

- ALLOCATION
- DEMAND_LIMIT
- DISTRIBUTION
- FLEX_NAME
- GROUP_DISTRIBUTION
- LS_FEATURE_PERCENTAGE

- NAME
- NON_SHARED_DISTRIBUTION
- SERVICE_DOMAINS
- WORKLOAD_DISTRIBUTION

The fast dispatch project mode also supports the MBD_HEARTBEAT_INTERVAL parameter in the Parameters section.

Other parameters are not supported, including those that project mode supports, such as the following parameters:

- ACCINUSE_INCLUDES_OWNERSHIP
- DYNAMIC
- GROUP
- LOCAL_TO
- LS_ACTIVE_PERCENTAGE

Default

Not defined (N). License Scheduler runs in project mode without fast dispatch.

HEARTBEAT_INTERVAL

Syntax

```
HEARTBEAT_INTERVAL=seconds
```

Description

The time interval between bld heartbeats indicating the bld is still running.

Default

60 seconds

HEARTBEAT_TIMEOUT

Syntax

```
HEARTBEAT_TIMEOUT=seconds
```

Description

The time a slave **bld** waits to hear from the master **bld** before assuming it has died.

Default

120 seconds

HIST_HOURS

Syntax

```
HIST_HOURS=hours
```

Description

Determines the rate of decay the accumulated use value used in fairshare and preemption decisions. When HIST_HOURS=0, accumulated use is not decayed.

Accumulated use is displayed by the **blstat** command under the heading ACUM_USE.

Used for project mode only.

Default

5 hours. Accumulated use decays to 1/10 of the original value over 5 hours.

HOSTS
Syntax

HOSTS=*host_name.domain_name* ...

Description

Defines License Scheduler hosts, including License Scheduler candidate hosts.

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.

Used for both project mode and cluster mode.

INUSE_FROM_RUSAGE
Syntax

INUSE_FROM_RUSAGE=Y|N

Description

When not defined or set to N, the **INUSE** value uses rusage from **bsub** job submissions merged with license checkout data reported by **blcollect** (as reported by **blstat**).

When INUSE_FROM_RUSAGE=Y, the **INUSE** value uses the rusage from **bsub** job submissions instead of waiting for the blcollect update. This can result in faster reallocation of tokens when using dynamic allocation (when **ALLOC_BUFFER** is set).

When for individual license features, the Feature section setting overrides the global Parameters section setting.

Used for cluster mode only.

Default

N

LIB_CONNTIMEOUT
Syntax

LIB_CONNTIMEOUT=*seconds*

Description

Specifies a timeout value in seconds for communication between License Scheduler and LSF APIs. `LIB_CONNTIMEOUT=0` indicates no timeout.

Used for both project mode and cluster mode.

Default

5 seconds

LIB_RECVTIMEOUT
Syntax

`LIB_RECVTIMEOUT=`*seconds*

Description

Specifies a timeout value in seconds for communication between License Scheduler and LSF.

Used for both project mode and cluster mode.

Default

5 seconds

LM_REMOVE_INTERVAL
Syntax

`LM_REMOVE_INTERVAL=`*seconds*

Description

Specifies the minimum time a job must have a license checked out before **lmremove** can remove the license (using preemption). **lmremove** causes **lmgrd** and vendor daemons to close the TCP connection with the application, then retries the license checkout.

License Scheduler only considers preempting a job after this interval has elapsed. **LM_REMOVE_INTERVAL** overrides the **LS_WAIT_TO_PREEMPT** value if **LM_REMOVE_INTERVAL** is larger.

When using **lmremove** as part of the preemption action (**LMREMOVE_SUSP_JOBS**), define `LM_REMOVE_INTERVAL=0` to ensure that License Scheduler can preempt a job immediately after checkout. After suspending the job, License Scheduler then uses **lmremove** to release licenses from the job.

Used for both project mode and cluster mode.

Default

180 seconds

LM_STAT_INTERVAL
Syntax

LM_STAT_INTERVAL=*seconds*

Description

Defines a time interval between calls that License Scheduler makes to collect license usage information from FlexNet license management.

Default

60 seconds

LM_STAT_TIMEOUT
Syntax

LM_STAT_TIMEOUT=*seconds*

Description

Sets the timeout value passed to the **lmstat** (or **lmutil lmstat**) command. The Parameters section setting is overwritten by the ServiceDomain setting, which is overwritten by the command line setting (**blcollect -t timeout**).

Used for both project mode and cluster mode.

Default

180 seconds

LMREMOVE_SUSP_JOBS
Syntax

LMREMOVE_SUSP_JOBS=*seconds*

Description

Enables License Scheduler to use **lmremove** to remove license features from each recently-suspended job. After enabling this parameter, the preemption action is to suspend the job's processes and use **lmremove** to remove licences from the application. **lmremove** causes **lmgrd** and vendor daemons to close the TCP connection with the application.

License Scheduler continues to try removing the license feature for the specified number of seconds after the job is first suspended. When setting this parameter for an application, specify a value greater than the period following a license checkout that **lmremove** will fail for the application. This ensures that when a job suspends, its licenses are released. This period depends on the application.

When using **lmremove** as part of the preemption action, define LM_REMOVE_INTERVAL=0 to ensure that License Scheduler can preempt a job immediately after checkout. After suspending the job, License Scheduler then uses **lmremove** to release licenses from the job.

This parameter applies to all features in fast dispatch project mode.

Used for fast dispatch project mode only.

Default

Undefined. The default preemption action is to send a TSTP signal to the job.

LMREMOVE_SUSP_JOBS_INTERVAL
Syntax

LMREMOVE_SUSP_JOBS_INTERVAL=*seconds*

Description

Specifies the minimum length of time between subsequent child processes that License Scheduler forks to run **lmremove** every time it receives an update form a license collector daemon (**blcollect**).

Use this parameter when using **lmremove** as part of the preemption action (**LMREMOVE_SUSP_JOBS**).

Used for fast dispatch project mode only.

Default

0

LMSTAT_PATH
Syntax

LMSTAT_PATH=*path*

Description

Defines the full path to the location of the FlexNet command **lmutil** (or **lmstat**).

Used for project mode, fast dispatch project mode, and cluster mode.

LOG_EVENT
Syntax

LOG_EVENT=Y

Description

Enables logging of License Scheduler events in the bld.stream file.

Default

Not defined. Information is not logged.

LOG_INTERVAL
Syntax

LOG_INTERVAL=*seconds*

Description

The interval between token allocation data logs in the data directory

Default

60 seconds

LS_DEBUG_BLC
Syntax

LS_DEBUG_BLC=*log_class*

Description

Sets the debugging log class for the License Scheduler blcollect daemon.

Used for both project mode and cluster mode.

Specifies the log class filtering to be applied to blcollect. Only messages belonging to the specified log class are recorded.

LS_DEBUG_BLC sets the log class and is used in combination with **LS_LOG_MASK**, which sets the log level. For example:

```
LS_LOG_MASK=LOG_DEBUG LS_DEBUG_BLC="LC_TRACE"
```

To specify multiple log classes, use a space-separated list enclosed in quotation marks. For example:

```
LS_DEBUG_BLC="LC_TRACE"
```

You need to restart the **blcollect** daemons after setting **LS_DEBUG_BLC** for your changes to take effect.

Valid values

Valid log classes are:

- LC_AUTH and LC2_AUTH: Log authentication messages
- LC_COMM and LC2_COMM: Log communication messages
- LC_FLEX - Log everything related to FLEX_STAT or FLEX_EXEC Flexera APIs
- LC_PERFM and LC2_PERFM: Log performance messages
- LC_PREEMPT - Log license preemption policy messages
- LC_RESREQ and LC2_RESREQ: Log resource requirement messages
- LC_SYS and LC2_SYS: Log system call messages
- LC_TRACE and LC2_TRACE: Log significant program walk steps
- LC_XDR and LC2_XDR: Log everything transferred by XDR

Default

Not defined.

LS_DEBUG_BLD
Syntax

LS_DEBUG_BLD=*log_class*

Description

Sets the debugging log class for the License Scheduler bld daemon.

Used for both project mode and cluster mode.

Specifies the log class filtering to be applied to bld. Messages belonging to the specified log class are recorded. Not all debug message are controlled by log class.

LS_DEBUG_BLD sets the log class and is used in combination with MASK, which sets the log level. For example:

```
LS_LOG_MASK=LOG_DEBUG LS_DEBUG_BLD="LC_TRACE"
```

To specify multiple log classes, use a space-separated list enclosed in quotation marks. For example:

```
LS_DEBUG_BLD="LC_TRACE"
```

You need to restart the **bld** daemon after setting LS_DEBUG_BLD for your changes to take effect.

If you use the command **bladmin blddebug** to temporarily change this parameter without changing `lsf.licensescheduler`, you do not need to restart the daemons.

Valid values

Valid log classes are:

- LC_AUTH and LC2_AUTH: Log authentication messages
- LC_COMM and LC2_COMM: Log communication messages
- LC_FLEX - Log everything related to FLEX_STAT or FLEX_EXEC Flexera APIs
- LC_MEMORY - Log memory use messages
- LC_PREEMPT - Log license preemption policy messages
- LC_RESREQ and LC2_RESREQ: Log resource requirement messages
- LC_TRACE and LC2_TRACE: Log significant program walk steps
- LC_XDR and LC2_XDR: Log everything transferred by XDR

Valid values

Valid log classes are the same as for LS_DEBUG_CMD.

Default

Not defined.

LS_ENABLE_MAX_PREEMPT

Syntax

```
LS_ENABLE_MAX_PREEMPT=Y
```

Description

Enables maximum preemption time checking for LSF and **taskman** jobs.

When **LS_ENABLE_MAX_PREEMPT** is disabled, preemption times for **taskman** job are not checked regardless of the value of parameters **LS_MAX_TASKMAN_PREEMPT** in `lsf.licensescheduler` and **MAX_JOB_PREEMPT** in `lsb.queues`, `lsb.applications`, or `lsb.params`.

Used for project mode only.

Default

N

LS_LOG_MASK

Syntax

`LS_LOG_MASK=`*message_log_level*

Description

Specifies the logging level of error messages for License Scheduler daemons. If **LS_LOG_MASK** is not defined in `lsf.licensescheduler`, 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**.

Used for both project mode and cluster mode.

For example:

```
LS_LOG_MASK=LOG_DEBUG
```

The log levels in order from highest to lowest are:

- LOG_ERR
- LOG_WARNING
- LOG_INFO
- 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 only useful for debugging.

Although message log level implements similar functionality to UNIX `syslog`, there is no dependency on UNIX `syslog`. It works even if messages are being logged to files instead of `syslog`.

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. 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.

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.

Default

LOG_WARNING

LS_MAX_STREAM_FILE_NUMBER

Syntax

LS_MAX_STREAM_FILE_NUMBER=*integer*

Description

Sets the number of saved bld.stream.timestamp log files. When LS_MAX_STREAM_FILE_NUMBER=2, for example, the two most recent files are kept along with the current bld.stream file.

Used for both project mode and cluster mode.

Default

0 (old bld.stream file is not saved)

LS_MAX_STREAM_SIZE

Syntax

LS_MAX_STREAM_SIZE=*integer*

Description

Defines the maximum size of the bld.stream file in MB. once this size is reached an **EVENT_END_OF_STREAM** is logged, a new bld.stream file is created, and the old bld.stream file is renamed bld.stream.*timestamp*.

Used for both project mode and cluster mode.

Default

1024

LS_MAX_TASKMAN_PREEMPT

Syntax

LS_MAX_TASKMAN_PREEMPT=*integer*

Description

Defines the maximum number of times **taskman** jobs can be preempted.

Maximum preemption time checking for all jobs is enabled by **LS_ENABLE_MAX_PREEMPT**.

Used for project mode only.

Default

unlimited

LS_MAX_TASKMAN_SESSIONS

Syntax

LS_MAX_TASKMAN_SESSIONS=*integer*

Description

Defines the maximum number of **taskman** jobs that run simultaneously. This prevents system-wide performance issues that occur if there are a large number of **taskman** jobs running in License Scheduler.

The number taskman sessions must be a positive integer.

The actual maximum number of taskman jobs is affected by the operating system file descriptor limit. Make sure the operating system file descriptor limit and the maximum concurrent connections are large enough to support all **taskman** tasks, License Scheduler (**bl***) commands, and connections between License Scheduler and LSF.

Used for both project mode and cluster mode.

LS_STREAM_FILE

Syntax

LS_STREAM_FILE=*path*

Used for both project mode and cluster mode.

Description

Defines the full path and filename of the bld event log file, `bld.stream` by default.

Note:

In License Scheduler 8.0 the `bld.events` log file was replaced by the `bld.stream` log file.

Default

LSF_TOP/work/db/bld.stream

LS_PREEMPT_PEER

Syntax

LS_PREEMPT_PEER=Y

Description

Enables bottom-up license token preemption in hierarchical project group configuration. License Scheduler attempts to preempt tokens from the closest projects in the hierarchy first. This balances token ownership from the bottom up.

Used for project mode only.

Default

Not defined. Token preemption in hierarchical project groups is top down.

MBD_HEARTBEAT_INTERVAL
Syntax

```
MBD_HEARTBEAT_INTERVAL=seconds
```

Description

Sets the length of time the cluster license allocation remains unchanged after a cluster has disconnected from **bld**. After **MBD_HEARTBEAT_INTERVAL** has passed, the allocation is set to zero and licenses are redistributed to other clusters.

Used for cluster mode and fast dispatch project mode only.

Default

900 seconds

MBD_REFRESH_INTERVAL
Syntax

```
MBD_REFRESH_INTERVAL=seconds
```

Description

MBD_REFRESH_INTERVAL: Cluster mode and project mode. This parameter allows the administrator to independently control the minimum interval between load updates from **bld**, and the minimum interval between load updates from LIM. The parameter controls the frequency of scheduling interactive (taskman) jobs. The parameter is read by **mbatchd** on startup. When MBD_REFRESH_INTERVAL is set or changed, you must restart **bld**, and restart **mbatchd** in each cluster.

Used for both project mode and cluster mode.

Default

15 seconds

MERGE_BY_SERVICE_DOMAIN
Syntax

```
MERGE_BY_SERVICE_DOMAIN=Y | N
```

Description

If enabled, correlates job license checkout with the **lmstat** output across all service domains first before reserving licenses.

In project mode (but not fast dispatch project mode), this parameter supports the case where the application's checkout license number is less than or equal to the job's rusage. If the checked out licenses are greater than the job's rusage, the ENABLE_DYNAMIC_RUSAGE parameter is still required.

Default

N (Does not correlate job license checkout with the **lmstat** output across all service domains before reserving licenses)

PEAK_INUSE_PERIOD

Syntax

PEAK_INUSE_PERIOD=*seconds*

Description

Defines the interval over which a peak **INUSE** value is determined for dynamic license allocation in cluster mode for all license features over all service domains.

Used for cluster mode only.

When defined in both the Parameters section and the Feature section, the Feature section definition is used for that license feature.

Default

300 seconds

PORT

Syntax

PORT=*integer*

Description

Defines the TCP listening port used by License Scheduler hosts, including candidate License Scheduler hosts. Specify any non-privileged port number.

Used for both project mode and cluster mode.

PREEMPT_ACTION

Syntax

PREEMPT_ACTION=*action*

Description

Specifies the action used for taskman job preemption.

By default, if **PREEMPT_ACTION** is not configured, bld sends a TSTP signal to preempt taskman jobs.

You can specify a script using this parameter. For example, PREEMPT_ACTION = /home/user1/preempt.s issues preempt.s when preempting a taskman job.

Used for project mode only.

Default

Not defined. A TSTP signal is used to preempt taskman jobs.

PROJECT_GROUP_PATH

Syntax

```
PROJECT_GROUP_PATH=Y
```

Description

Enables hierarchical project group paths for fast dispatch project mode, which enables the following:

- Features can use hierarchical project groups with project and project group names that are not unique, as long as the projects or project groups do not have the same parent. That is, you can define projects and project groups in more than one hierarchical project group.
- When specifying `-Lp` *license_project*, you can use paths to describe the project hierarchy without specifying the root group.

  For example, if you have `root` as your root group, which has a child project group named `groupA` with a project named `proj1`, you can use `-Lp /groupA/proj1` to specify this project.
- Hierarchical project groups have a default project named `others` with a default share value of 0. Any projects that do not match the defined projects in a project group are assigned into the `others` project.

  If there is already a project named `others`, the preexisting `others` project specification overrides the default project.

If **LSF_LIC_SCHED_STRICT_PROJECT_NAME** (in `lsf.conf`) and **PROJECT_GROUP_PATH** are both defined, **PROJECT_GROUP_PATH** takes precedence and overrides the **LSF_LIC_SCHED_STRICT_PROJECT_NAME** behavior for fast dispatch project mode.

Note: To use **PROJECT_GROUP_PATH**, you need LSF, Version 9.1.1, or later.

Used for fast dispatch project mode only.

Default

Not defined (N).

REMOTE_LMSTAT_PROTOCOL

Syntax

`REMOTE_LMSTAT_PROTOCOL=ssh` [*ssh_command_options*] | `rsh` [*rsh_command_options*] | `lsrun` [*lsrun_command_options*]

Description

Specifies the method that License Scheduler uses to connect to the remote agent host if there are remote license servers that need a remote agent host to collect license information.

If there are remote license servers that need a remote agent host to collect license information, License Scheduler uses the specified command (and optional command options) to connect to the agent host. License Scheduler automatically appends the name of the remote agent host to the command, so there is no need to specify the host with the command.

Note: License Scheduler does not validate the specified command, so you must ensure that you correctly specify the command. The **blcollect** log file notes that the command failed, but not any details on the connection error. To determine specific connection errors, manually specify the command to connect to the remote server before specifying it in **REMOTE_LMSTAT_PROTOCOL**.

If using **lsrun** as the connection method, the remote agent host must be a server host in the LSF cluster and RES must be started on this host. If using **ssh** or **rsh** as the connection method, the agent host does not have to be a server host in the LSF cluster.

REMOTE_LMSTAT_PROTOCOL works with **REMOTE_LMSTAT_SERVERS**, which defines the remote license servers and remote agent hosts. If you do not define **REMOTE_LMSTAT_SERVERS**, **REMOTE_LMSTAT_PROTOCOL** is not used.

Used for both project mode and cluster mode.

Default

ssh

STANDBY_CONNTIMEOUT
Syntax

STANDBY_CONNTIMEOUT=*seconds*

Description

Sets the connection timeout the standby bld waits when trying to contact each host before assuming the host is unavailable.

Used for both project mode and cluster mode.

Default

5 seconds

Clusters section
Description

Required. Lists the clusters that can use License Scheduler.

When configuring clusters for a WAN, the Clusters section of the master cluster must define its slave clusters.

The Clusters section is the same for both project mode and cluster mode.

Clusters section structure

The Clusters section begins and ends with the lines `Begin Clusters` and `End Clusters`. The second line is the column heading, `CLUSTERS`. Subsequent lines list participating clusters, one name per line:

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

CLUSTERS

Defines the name of each participating LSF cluster. Specify using one name per line.

ServiceDomain section

Description

Required. Defines License Scheduler service domains as groups of physical license server hosts that serve a specific network.

The ServiceDomain section is the same for both project mode and cluster mode.

ServiceDomain section structure

Define a section for each License Scheduler service domain.

This example shows the structure of the section:

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

Parameters

- LIC_SERVERS
- LIC_COLLECTOR
- LM_STAT_INTERVAL
- LM_STAT_TIMEOUT
- NAME
- REMOTE_LMSTAT_SERVERS

LIC_SERVERS

Syntax

LIC_SERVERS=([(*host_name* | *port_number*@*host_name* | (*port_number*@*host_name* *port_number*@*host_name port_number*@*host_name*))] ...)

Description

Defines the FlexNet license server hosts that make up the License Scheduler service domain. For each FlexNet license server host, specify the number of the port that FlexNet uses, then the at symbol (@), then the name of the host. If FlexNet uses the default port on a host, you can specify the host name without the port number. Put one set of parentheses around the list, and one more set of parentheses around each host, unless you have redundant servers (three hosts sharing one license file). If you have redundant servers, the parentheses enclose all three hosts.

Used for both project mode and cluster mode.

Examples

- One FlexNet license server host:
  `LIC_SERVERS=((1700@hostA))`
- Multiple FlexNet license server hosts with unique `license.dat` files:
  `LIC_SERVERS=((1700@hostA)(1700@hostB)(1700@hostC))`
- Redundant FlexNet license server hosts sharing the same `license.dat` file:
  `LIC_SERVERS=((1700@hostD 1700@hostE 1700@hostF))`

LIC_COLLECTOR

Syntax

`LIC_COLLECTOR=`*license_collector_name*

Description

Optional. Defines a name for the license collector daemon (**blcollect**) to use in each service domain. **blcollect** collects license usage information from FlexNet and passes it to the License Scheduler daemon (**bld**). It improves performance by allowing you to distribute license information queries on multiple hosts.

You can only specify one collector per service domain, but you can specify one collector to serve multiple service domains. Each time you run **blcollect**, you must specify the name of the collector for the service domain. You can use any name you want.

Used for both project mode and cluster mode.

Default

Undefined. The License Scheduler daemon uses one license collector daemon for the entire cluster.

LM_STAT_INTERVAL

Syntax

`LM_STAT_INTERVAL=`*seconds*

Description

Defines a time interval between calls that License Scheduler makes to collect license usage information from FlexNet license management.

The value specified for a service domain overrides the global value defined in the Parameters section. Each service domain definition can specify a different value for this parameter.

Used for both project mode and cluster mode.

Default

License Scheduler applies the global value defined in the Parameters section.

LM_STAT_TIMEOUT

Syntax

LM_STAT_TIMEOUT=*seconds*

Description

Sets the timeout value passed to the **lmstat** (or **lmutil lmstat**) command. The Parameters section setting is overwritten by the ServiceDomain setting, which is overwritten by the command line setting (**blcollect -t timeout**).

Used for both project mode and cluster mode.

Default

180 seconds

NAME

Defines the name of the service domain.

Used for both project mode and cluster mode.

REMOTE_LMSTAT_SERVERS

Syntax

REMOTE_LMSTAT_SERVERS=*host_name*[(*host_name ...*)] [*host_name*[(*host_name ...*)] ...]

Description

Defines the remote license servers and, optionally, the remote agent hosts that serve these remote license servers.

A remote license server is a license server that does not run on the same domain as the license collector. A remote agent host serves remote license servers within the same domain, allowing the license collector to get license information on the remote license servers with a single remote connection.

Defining remote agent hosts are useful when there are both local and remote license servers because it is slower for the license collector to connect to multiple remote license servers to get license information than it is to connect to local license servers. The license collector connects to the remote agent host (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and calls **lmutil** (or **lmstat**) to collect license information from the license servers that the agent hosts serve. This allows the license collector to connect to one remote agent host to get license information from all the remote license servers on the same domain as the remote agent host. These license servers should be in the same subnet as the agent host to improve access.

Remote license servers must also be license servers defined in **LIC_SERVERS**. Any remote license servers defined in **REMOTE_LMSTAT_SERVERS** that are not also defined in **LIC_SERVERS** are ignored. Remote agent hosts that serve other license servers do not need to be defined in **LIC_SERVERS**. Remote agent hosts that are not defined in **LIC_SERVERS** function only as remote agents and not as license servers.

If you specify a remote agent host without additional servers (that is, the remote agent host does not serve any license servers), the remote agent host is considered

to be a remote license server with itself as the remote agent host. That is, the license collector connects to the remote agent host and only gets license information on the remote agent host. Because these hosts are remote license servers, these remote agent hosts must also be defined as license servers in **LIC_SERVERS**, or they will be ignored.

Used for both project mode and cluster mode.

Examples

- One local license server (hostA) and one remote license server (hostB):

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

  – The license collector runs **lmutil** (or **lmstat**) directly on hostA to get license information on hostA.

  – Because hostB is defined without additional license servers, hostB is a remote agent host that only serves itself. The license collector connects to $_{hostB}$ (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmstat** to get license information on 1700@hostB.

- One local license server (hostA), one remote agent host (hostB) that serves one remote license server (hostC), and one remote agent host (hostD) that serves two remote license servers (hostE and hostF):

  ```
  LIC_SERVERS=((1700@hostA)(1700@hostB)(1700@hostC)(1700@hostD)(1700@hostE)(1700@hostF))
  REMOTE_LMSTAT_SERVERS=hostB(hostC) hostD(hostE hostF)
  ```

  – The license collector runs **lmutil** (or **lmstat**) directly to get license information from 1700@hostA, 1700@hostB, and 1700@hostD.

  – The license collector connects to hostB (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmstat** to get license information on 1700@hostC.

    hostB and hostC should be in the same subnet to improve access.

  – The license collector connects to hostD (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmutil** (or **lmstat**) to get license information on 1700@hostE and 1700@hostF.

    hostD, hostE, and hostF should be in the same subnet to improve access.

- One local license server (hostA), one remote license server (hostB), and one remote agent host (hostC) that serves two remote license servers (hostD and hostE):

  ```
  LIC_SERVERS=((1700@hostA)(1700@hostB)(1700@hostC)(1700@hostD)(1700@hostE))
  REMOTE_LMSTAT_SERVERS=hostB hostC(hostD hostE)
  ```

  – The license collector runs **lmutil** (or **lmstat**) directly to get license information on 1700@hostA and 1700@hostC.

  – The license collector connects to hostB (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmstat** to get license information on 1700@hostB.

  – The license collector connects to hostC (using the command specified by the **REMOTE_LMSTAT_PROTOCOL** parameter) and runs **lmstat** to get license information on 1700@hostD and 1700@hostE.

    hostC, hostD, and hostE should be in the same subnet to improve access.

Feature section

Description

Required. Defines license distribution policies.

Feature section structure

Define a section for each feature managed by License Scheduler.

```
Begin Feature
NAME=vcs
FLEX_NAME=vcs
...
Distribution policy
Parameters
...
End Feature
```

Parameters

- ACCINUSE_INCLUDES_OWNERSHIP
- ALLOC_BUFFER
- ALLOCATION
- CLUSTER_DISTRIBUTION
- CLUSTER_MODE
- DEMAND_LIMIT
- DISTRIBUTION
- DYNAMIC
- ENABLE_DYNAMIC_RUSAGE
- ENABLE_MINJOB_PREEMPTION
- CHECKOUT_FROM_FIRST_HOST_ONLY
- FAST_DISPATCH
- FLEX_NAME
- GROUP
- GROUP_DISTRIBUTION
- INUSE_FROM_RUSAGE
- LM_REMOVE_INTERVAL
- LMREMOVE_SUSP_JOBS
- LOCAL_TO
- LS_ACTIVE_PERCENTAGE
- LS_FEATURE_PERCENTAGE
- LS_WAIT_TO_PREEMPT
- NAME
- NON_SHARED_DISTRIBUTION
- PEAK_INUSE_PERIOD
- PREEMPT_ORDER
- PREEMPT_RESERVE
- RETENTION_FACTOR
- SERVICE_DOMAINS
- WORKLOAD_DISTRIBUTION

ACCINUSE_INCLUDES_OWNERSHIP

Syntax

```
ACCINUSE_INCLUDES_OWNERSHIP=Y
```

Description

When not defined, accumulated use is incremented each scheduling cycle by (tokens in use) + (tokens reserved) if this exceeds the number of tokens owned.

When defined, accumulated use is incremented each scheduling cycle by (tokens in use) + (tokens reserved) regardless of the number of tokens owned.

This is useful for projects that have a very high ownership set when considered against the total number of tokens available for LSF workload. Projects can be starved for tokens when the ownership is set too high and this parameter is not set.

Accumulated use is displayed by the **blstat** command under the heading ACUM_USE.

Used for project mode only. Cluster mode and fast dispatch project mode do not track accumulated use.

Default

N, not enabled.

ALLOC_BUFFER

Syntax

ALLOC_BUFFER = *buffer* | *cluster_name buffer* ... default *buffer*

Description

Enables dynamic distribution of licenses across clusters in cluster mode.

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

Used for cluster mode only.

ALLOC_BUFFER=*buffer* sets one buffer size for all clusters, while ALLOC_BUFFER=*cluster_name buffer* ... sets a different buffer size for each cluster.

The buffer size is used during dynamic redistribution of licenses. Increases in allocation are determined by the PEAK value, and increased by DEMAND for license tokens to a maximum increase of BUFFER, the buffer size configured by **ALLOC_BUFFER**. The license allocation can increase in steps as large as the buffer size, but no larger.

Allocation buffers help determine the maximum rate at which tokens can be transferred to a cluster as demand increases in the cluster. The maximum rate of transfer to a cluster is given by the allocation buffer divided by **MBD_REFRESH_INTERVAL**. Be careful not to set the allocation buffer too large so that licenses are not wasted because they are be allocated to a cluster that cannot use them.

Decreases in license allocation can be larger than the buffer size, but the allocation must remain at **PEAK**+**BUFFER** licenses. The license allocation includes up to the buffer size of extra licenses, in case demand increases.

Increasing the buffer size allows the license allocation to grow faster, but also increases the number of licenses that may go unused at any given time. The buffer value must be tuned for each license feature and cluster to balance these two objectives.

Detailed license distribution information is shown in the **blstat** output.

Use the keyword default to apply a buffer size to all remaining clusters. For example:

```
Begin Feature
NAME = f1
CLUSTER_DISTRIBUTION = WanServers(banff 1 berlin 1 boston 1)
ALLOC_BUFFER = banff 10 default 5
End Feature
```

In this example, dynamic distribution is enabled. The cluster banff has a buffer size of 10, and all remaining clusters have a buffer size of 5.

To allow a cluster to be able to use licenses only when another cluster does not need them, you can set the cluster distribution for the cluster to 0, and specify an allocation buffer for the number of tokens that the cluster can request.

For example:

```
Begin Feature
CLUSTER_DISTRIBUTION=Wan(CL1 0 CL2 1)
ALLOC_BUFFER=5
End Feature
```

When no jobs are running, the token allocation for CL1 is 5. CL1 can get more than 5 tokens if CL2 does not require them.

Default

Not defined. Static distribution of licenses is used in cluster mode.

ALLOCATION

Syntax

ALLOCATION=*project_name* (*cluster_name* [*number_shares*] ...) ...

cluster_name

Specify LSF cluster names or interactive tasks that licenses are to be allocated to.

project_name

Specify a License Scheduler project (described in the Projects section or as default) that is allowed to use the licenses.

number_shares

Specify a positive integer representing the number of shares assigned to the cluster.

The number of shares assigned to a cluster is only meaningful when you compare it to the number assigned to other clusters. The total number of shares is the sum of the shares assigned to each cluster.

Description

Defines the allocation of license features across clusters and interactive tasks.

Used for project mode and fast dispatch project mode only.

ALLOCATION ignores the global setting of the ENABLE_INTERACTIVE parameter because ALLOCATION is configured for the license feature.

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

Tip: To manage interactive tasks in License Scheduler projects, use the LSF Task Manager, `taskman`. The Task Manager utility is supported by License Scheduler.

Default

If ENABLE_INTERACTIVE is not set, each cluster receives equal share, and interactive tasks receive no shares.

Examples:
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=ApplicationX
DISTRIBUTION=LicenseServer1 (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=ApplicationX
DISTRIBUTION=LicenseServer1 (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=ApplicationY
DISTRIBUTION=LicenseServer1 (Lp2 1)
ALLOCATION=Lp2(cluster1 1 cluster2 0 interactive 1)
End Feature
```

The ENABLE_INTERACTIVE setting is ignored. Licenses are shared equally between `cluster1` and interactive tasks. Six licenses of `ApplicationY` 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=ApplicationZ
DISTRIBUTION=LicenseServer1 (Lp1 1)
ALLOCATION=Lp1(cluster1 0 cluster2 1 interactive 2)
End Feature
```

The ENABLE_INTERACTIVE setting is ignored. Four licenses of `ApplicationZ` are allocated to `cluster2`. Eight licenses are allocated to interactive tasks.

CHECKOUT_FROM_FIRST_HOST_ONLY

Syntax

```
CHECKOUT_FROM_FIRST_HOST_ONLY=Y
```

Description

If enabled, License Scheduler only considers `user@host` information for the first execution host of a parallel job when merging the license usage data. Setting in individual Feature sections overrides the global setting in the Parameters section.

If a feature has multiple Feature sections (using **LOCAL_TO**), each section must have the same setting for **CHECKOUT_FROM_FIRST_HOST_ONLY**.

If disabled, License Scheduler attempts to check out `user@host` keys in the parallel job constructed using the user name and all execution host names, and merges the corresponding checkout information on the service domain if found. If

MERGE_BY_SERVICE_DOMAIN=Y is defined, License Scheduler also merges multiple user@host data for parallel jobs across different service domains.

Default

Undefined (N).License Scheduler attempts to check out user@host keys in the parallel job constructed using the user name and all execution host names, and merges the corresponding checkout information on the service domain if found.

CLUSTER_DISTRIBUTION

Syntax

CLUSTER_DISTRIBUTION=*service_domain*(*cluster shares*/*min*/*max* ...)...

service_domain

Specify a License Scheduler WAN service domain (described in the ServiceDomain section) that distributes licenses to multiple clusters, and the share for each cluster.

Specify a License Scheduler LAN service domain for a single cluster.

cluster

Specify each LSF cluster that accesses licenses from this service domain.

shares

For each cluster specified for a WAN service domain, specify a positive integer representing the number of shares assigned to the cluster. (Not required for a LAN service domain.)

The number of shares assigned to a cluster is only meaningful when you compare it to the number assigned to other clusters, or to the total number assigned by the service domain. The total number of shares is the sum of the shares assigned to each cluster.

min

Optionally, specify a positive integer representing the minimum number of license tokens allocated to the cluster when dynamic allocation is enabled for a WAN service domain (when **ALLOC_BUFFER** is defined for the feature).

The minimum allocation is allocated exclusively to the cluster, and is similar to the non-shared allocation in project mode.

Cluster shares take precedence over minimum allocations configured. If the minimum allocation exceeds the cluster's share of the total tokens, a cluster's allocation as given by **bld** may be less than the configured minimum allocation.

max

Optionally, specify a positive integer representing the maximum number of license tokens allocated to the cluster when dynamic allocation is enabled for a WAN service domain (when **ALLOC_BUFFER** is definedfor the feature).

Description

CLUSTER_DISTRIBUTION must be defined when using cluster mode.

Defines the cross-cluster distribution policies for the license. The name of each service domain is followed by its distribution policy, in parentheses. The distribution policy determines how the licenses available in each service domain are distributed among the clients.

The distribution policy is a space-separated list with each cluster name followed by its share assignment. The share assignment determines what fraction of available licenses is assigned to each cluster, in the event of competition between clusters.

Examples

```
CLUSTER_DISTRIBUTION=wanserver(Cl1 1 Cl2 1 Cl3 1 Cl4 1)
CLUSTER_DISTRIBUTION = SD(C1 1 C2 1) SD1(C3 1 C4 1) SD2(C1 1) SD3(C2 1)
```

In these examples, wanserver, SD, and SD1 are WAN service domains, while SD2 and SD3 are LAN service domains serving a single cluster.

CLUSTER_MODE
Syntax

```
CLUSTER_MODE=Y
```

Description

Enables cluster mode (instead of project mode) for the license feature. Setting in the Feature section overrides the global setting in the Parameters section.

Cluster mode emphasizes high utilization of license tokens above other considerations such as ownership. License ownership and sharing can still be configured, but within each cluster instead of across multiple clusters. Preemption of jobs (and licenses) also occurs within each cluster instead of across clusters.

Cluster mode was introduced in License Scheduler 8.0. Before cluster mode was introduced, project mode was the only choice available.

Default

Undefined (N). License Scheduler runs in project mode.

DEMAND_LIMIT
Syntax

DEMAND_LIMIT=*integer*

Description

Sets a limit to which License Scheduler considers the demand by each project in each cluster when allocating licenses. Setting in the Feature section overrides the global setting in the Parameters section.

Used for fast dispatch project mode only.

When enabled, the demand limit helps prevent License Scheduler from allocating more licenses to a project than can actually be used, which reduces license waste by limiting the demand that License Scheduler considers. This is useful in cases

when other resource limits are reached, License Scheduler allocates more tokens than Platform LSF can actually use because jobs are still pending due to lack of other resources.

When disabled (that is, DEMAND_LIMIT=0 is set), License Scheduler takes into account all the demand reported by each cluster when scheduling.

DEMAND_LIMIT does not affect the DEMAND that **blstat** displays. Instead, **blstat** displays the entire demand sent for a project from all clusters. For example, one cluster reports a demand of 15 for a project. Another cluster reports a demand of 20 for the same project. When License Scheduler allocates licenses, it takes into account a demand of five from each cluster for the project and the DEMAND that **blstat** displays is 35.

Periodically, each cluster sends a demand for each project. This is calculated in a cluster for a project by summing up the rusage of all jobs of the project pending due to lack of licenses. Whether to count a job's rusage in the demand depends on the job's pending reason. In general, the demand reported by a cluster only represents a potential demand from the project. It does not take into account other resources that are required to start a job. For example, a demand for 100 licenses is reported for a project. However, if License Scheduler allocates 100 licenses to the project, the project does not necessarily use all 100 licenses due to slot available, limits, or other scheduling constraints.

In project mode and fast dispatch project mode, **mbatchd** in each cluster sends a demand for licenses from each project. In project mode, License Scheduler assumes that each project can actually use the demand that is sent to it. In fast dispatch project mode, **DEMAND_LIMIT** limits the amount of demand from each project in each cluster that is considered when scheduling.

Default

5

DISTRIBUTION

Syntax

DISTRIBUTION=[*service_domain_name*([*project_name number_shares*[/ *number_licenses_owned*]] ... [default])] ...

service_domain_name

Specify a License Scheduler service domain (described in the ServiceDomain section) that distributes the licenses.

project_name

Specify a License Scheduler project (described in the Projects section) that is allowed to use the licenses.

number_shares

Specify a positive integer representing the number of shares assigned to the project.

The number of shares assigned to a project is only meaningful when you compare it to the number assigned to other projects, or to the total number assigned by the service domain. The total number of shares is the sum of the shares assigned to each project.

number_licenses_owned

Optional. Specify a slash (/) and a positive integer representing the number of licenses that the project owns. When configured, preemption is enabled and owned licenses are reclaimed using preemption when there is unmet demand.

default

A reserved keyword that represents the default project if the job submission does not specify a project (**bsub -Lp**), or the specified project is not configured in the Projects section of `lsf.licensescheduler`. Jobs that belong to projects do not get a share of the tokens when the project is not explicitly defined in DISTRIBUTION.

Description

Used for project mode and fast dispatch project mode only.

One of **DISTRIBUTION** or **GROUP_DISTRIBUTION** must be defined when using project mode. **GROUP_DISTRIBUTION** and **DISTRIBUTION** are mutually exclusive. If defined in the same feature, the License Scheduler daemon returns an error and ignores this feature.

Defines the distribution policies for the license. The name of each service domain is followed by its distribution policy, in parentheses. The distribution policy determines how the licenses available in each service domain are distributed among the clients.

When in fast dispatch project mode, you can only specify one service domain.

The distribution policy is a space-separated list with each project name followed by its share assignment. The share assignment determines what fraction of available licenses is assigned to each project, in the event of competition between projects. Optionally, the share assignment is followed by a slash and the number of licenses owned by that project. License ownership enables a preemption policy (In the event of competition between projects, projects that own licenses preempt jobs. Licenses are returned to the owner immediately).

Examples

```
DISTRIBUTION=wanserver (Lp1 1 Lp2 1 Lp3 1 Lp4 1)
```

In this example, the service domain named `wanserver` shares licenses equally among four projects. If all projects are competing for a total of eight licenses, each project is entitled to two licenses at all times. If all projects are competing for only two licenses in total, each project is entitled to a license half the time.

```
DISTRIBUTION=lanserver1 (Lp1 1 Lp2 2/6)
```

In this example, the service domain named `lanserver1` allows `Lp1` to use one third of the available licenses and `Lp2` can use two thirds of the licenses. However, `Lp2` is always entitled to six licenses, and can preempt another project to get the licenses immediately if they are needed. If the projects are competing for a total of 12 licenses, `Lp2` is entitled to eight licenses (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.

DYNAMIC
Syntax

```
DYNAMIC=Y
```

Description

If you specify DYNAMIC=Y, 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.

Used for project mode only. Cluster mode and fast dispatch project mode do not support rusage duration.

ENABLE_DYNAMIC_RUSAGE
Syntax

```
ENABLE_DYNAMIC_RUSAGE=Y
```

Description

Enforces license distribution policies for class-C license features.

When set, ENABLE_DYNAMIC_RUSAGE enables all class-C license checkouts to be considered managed checkout, instead of unmanaged (or OTHERS).

Used for project mode only. Cluster mode and fast dispatch project mode do not support this parameter.

ENABLE_MINJOB_PREEMPTION
Syntax

```
ENABLE_MINJOB_PREEMPTION=Y
```

Description

Minimizes the overall number of preempted jobs by enabling job list optimization. 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.

Used for project mode only

Default

Undefined: License Scheduler does not optimize the job list when selecting jobs to preempt.

FAST_DISPATCH
Syntax

```
FAST_DISPATCH=Y
```

Description

Enables fast dispatch project mode for the license feature, which increases license utilization for project licenses. Setting in the Feature section overrides the global setting in the Parameters section.

Used for project mode only.

When enabled, License Scheduler does not have to run the FlexNet command **lmstat** to verify that a license is free before each job dispatch. As soon as a job finishes, the cluster can reuse its licenses for another job of the same project, which keeps gaps between jobs small. However, because License Scheduler does not run **lmstat** to verify that the license is free, there is an increased chance of a license checkout failure for jobs if the license is already in use by a job in another project.

The fast dispatch project mode supports the following parameters in the Feature section:

- ALLOCATION
- DEMAND_LIMIT
- DISTRIBUTION
- FLEX_NAME
- GROUP_DISTRIBUTION
- LS_FEATURE_PERCENTAGE
- NAME
- NON_SHARED_DISTRIBUTION
- SERVICE_DOMAINS
- WORKLOAD_DISTRIBUTION

The fast dispatch project mode also supports the MBD_HEARTBEAT_INTERVAL parameter in the Parameters section.

Other parameters are not supported, including those that project mode supports, such as the following parameters:

- ACCINUSE_INCLUDES_OWNERSHIP
- DYNAMIC
- GROUP
- LOCAL_TO
- LS_ACTIVE_PERCENTAGE

Default

Undefined (N). License Scheduler runs in project mode without fast dispatch.

FLEX_NAME

Syntax

`FLEX_NAME=`*`feature_name1`* [*`feature_name2`* ...]

Description

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. For feature names that start with a number or contain a dash (-), you must set both **NAME** and **FLEX_NAME**, where **FLEX_NAME** is the actual FlexNet Licensing feature name, and NAME is an arbitrary license token name you choose.

Specify a space-delimited list of feature names in **FLEX_NAME** to combine multiple FlexNet features into one feature name specified under the **NAME** parameter. This allows you to use the same feature name for multiple FlexNet features (that are interchangeable for applications). LSF recognizes the alias of the combined feature (specified in **NAME**) as a feature name instead of the individual FlexNet feature names specified in **FLEX_NAME**. When submitting a job to LSF, users specify the combined feature name in the **bsub** `rusage` string, which allows the job to use any token from any of the features specified in **FLEX_NAME**.

Example

To specify `AppZ201` as an alias for the FlexNet feature named `201-AppZ`:

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

To combine two FlexNet features (`201-AppZ` and `202-AppZ`) into a feature named `AppZ201`:

```
Begin Feature
FLEX_NAME=201-AppZ 202-AppZ
NAME=AppZ201
DISTRIBUTION=LanServer1(Lp1 1 Lp2 1)
End Feature
```

`AppZ201` is a combined feature that uses both `201-AppZ` and `202-AppZ` tokens. Submitting a job with `AppZ201` in the rusage string (for example, `bsub -Lp Lp1 -R "rusage[AppZ201=2]" myjob`) means that the job checks out tokens for either `201-AppZ` or `202-AppZ`.

GROUP

Syntax

`GROUP=[`*group_name*`(`*project_name*`... )] ...`

group_name

Specify a name for a group of projects. This is different from a ProjectGroup section; groups of projects are not hierarchical.

project_name

Specify a License Scheduler project (described in the Projects section) that is allowed to use the licenses. The project must appear in the DISTRIBUTION and only belong to one group.

Description

Optional. Defines groups of projects and specifies the name of each group. The groups defined here are used for group preemption. The number of licenses owned by the group is the total number of licenses owned by member projects.

Used for project mode only. Cluster mode and fast dispatch project mode do not support this parameter.

This parameter is ignored if GROUP_DISTRIBUTION is also defined.

Example

For example, without the GROUP configuration shown, proj1 owns 4 license tokens and can reclaim them using preemption. After adding the GROUP configuration, proj1 and proj2 together own 8 license tokens. If proj2 is idle, proj1 is able to reclaim all 8 license tokens using preemption.

```
Begin Feature
NAME = AppY
DISTRIBUTION = LanServer1(proj1 1/4 proj2 1/4 proj3 2)
GROUP = GroupA(proj1 proj2)
End Feature
```

GROUP_DISTRIBUTION

Syntax

GROUP_DISTRIBUTION=*top_level_hierarchy_name*

top_level_hierarchy_name

Specify the name of the top level hierarchical group.

Description

Defines the name of the hierarchical group containing the distribution policy attached to this feature, where the hierarchical distribution policy is defined in a ProjectGroup section.

One of **DISTRIBUTION** or **GROUP_DISTRIBUTION** must be defined when using project mode. **GROUP_DISTRIBUTION** and **DISTRIBUTION** are mutually exclusive. If defined in the same feature, the License Scheduler daemon returns an error and ignores this feature.

If **GROUP** is also defined, it is ignored in favor of **GROUP_DISTRIBUTION**.

Example

The following example shows the **GROUP_DISTRIBUTION** parameter hierarchical scheduling for the top-level hierarchical group named groups. The **SERVICE_DOMAINS** parameter defines a list of service domains that provide tokens for the group.

```
Begin Feature
NAME = myjob2
GROUP_DISTRIBUTION = groups
SERVICE_DOMAINS = LanServer wanServer
End Feature
```

INUSE_FROM_RUSAGE

Syntax

```
INUSE_FROM_RUSAGE=Y|N
```

Description

When not defined or set to N, the **INUSE** value uses rusage from **bsub** job submissions merged with license checkout data reported by **blcollect** (as reported by **blstat**).

When INUSE_FROM_RUSAGE=Y, the **INUSE** value uses the rusage from **bsub** job submissions instead of waiting for the blcollect update. This can result in faster reallocation of tokens when using dynamic allocation (when **ALLOC_BUFFER** is set).

When for individual license features, the Feature section setting overrides the global Parameters section setting.

Used for cluster mode only.

Default

N

LM_REMOVE_INTERVAL

Syntax

```
LM_REMOVE_INTERVAL=seconds
```

Description

Specifies the minimum time a job must have a license checked out before **lmremove** can remove the license. **lmremove** causes **lmgrd** and vendor daemons to close the TCP connection with the application. They then retry the license checkout.

When using **lmremove** as part of the preemption action (**LMREMOVE_SUSP_JOBS**), define LM_REMOVE_INTERVAL=0 to ensure that License Scheduler can preempt a job immediately after checkout. After suspending the job, License Scheduler then uses **lmremove** to release licenses from the job.

Used for both project mode and cluster mode.

The value specified for a feature overrides the global value defined in the Parameters section. Each feature definition can specify a different value for this parameter.

Default

Undefined: License Scheduler applies the global value.

LMREMOVE_SUSP_JOBS

Syntax

```
LMREMOVE_SUSP_JOBS=seconds
```

Description

Enables License Scheduler to use **lmremove** to remove license features from each recently-suspended job. After enabling this parameter, the preemption action is to suspend the job's processes and use **lmremove** to remove licences from the application. **lmremove** causes **lmgrd** and vendor daemons to close the TCP connection with the application.

License Scheduler continues to try removing the license feature for the specified number of seconds after the job is first suspended. When setting this parameter for an application, specify a value greater than the period following a license checkout that **lmremove** will fail for the application. This ensures that when a job suspends, its licenses are released. This period depends on the application.

When using **lmremove** as part of the preemption action, define LM_REMOVE_INTERVAL=0 to ensure that License Scheduler can preempt a job immediately after checkout. After suspending the job, License Scheduler then uses **lmremove** to release licenses from the job.

Used for fast dispatch project mode only.

The value specified for a feature overrides the global value defined in the Parameters section. Each feature definition can specify a different value for this parameter.

Default

Undefined. The default preemption action is to send a TSTP signal to the job.

LOCAL_TO

Syntax

LOCAL_TO=*cluster_name* | *location_name*(*cluster_name* [*cluster_name* ...])

Description

Used for project mode only. Cluster mode and fast dispatch project mode do not support this parameter.

Configures token locality for the license feature. You must configure different feature sections for same feature based on their locality. 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.

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.

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 as different resources. Use **blinfo** 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 when using project mode.

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 = clusterA
End Feature
```

```
Begin Feature
NAME = hspice
GROUP_DISTRIBUTION = group1
SERVICE_DOMAINS = SD2
LOCAL_TO = clusterB
End Feature
```

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

Default

Not defined. The feature is available to all clusters and taskman jobs, and is not restricted by cluster.

LS_ACTIVE_PERCENTAGE

Syntax

```
LS_ACTIVE_PERCENTAGE=Y | N
```

Description

Configures license ownership in percentages instead of absolute numbers and adjusts ownership for inactive projects. Sets LS_FEATURE_PERCENTAGE=Y automatically.

Settings LS_ACTIVE_PERCENTAGE=Y dynamically adjusts ownership based on project activity, setting ownership to zero for inactive projects and restoring the configured ownership setting when projects become active. If the total ownership for the license feature is greater than 100%, each ownership value is scaled appropriately for a total ownership of 100%.

Used for project mode only. Cluster mode and fast dispatch project mode do not support this parameter.

Default

N (Ownership values are not changed based on project activity.)

LS_FEATURE_PERCENTAGE

Syntax

```
LS_FEATURE_PERCENTAGE=Y | N
```

Description

Configures license ownership in percentages instead of absolute numbers. When not combined with hierarchical projects, affects the owned values in DISTRIBUTION and the NON_SHARED_DISTRIBUTION values only.

When using hierarchical projects, percentage is applied to OWNERSHIP, LIMITS, and NON_SHARED values.

Used for project mode and fast dispatch project mode only.

For example:

```
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 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

With LS_FEATURE_PERCENTAGE=Y in feature section and using 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%.

Default

N (Ownership is not configured with percentages, but with absolute numbers.)

LS_WAIT_TO_PREEMPT

Syntax

LS_WAIT_TO_PREEMPT=*seconds*

Description

Defines the number of seconds that jobs must wait (time since it was dispatched) before it can be preempted. Applies to LSF and taskman jobs.

Used for project mode only.

When **LM_REMOVE_INTERVAL** is also defined, the **LM_REMOVE_INTERVAL** value overrides the **LS_WAIT_TO_PREEMPT** value.

Default

0. The job can be preempted even if it was just dispatched.

NAME

Required. Defines the token name—the name used by License Scheduler and LSF to identify the license feature.

Normally, license token names should be the same as the FlexNet Licensing 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 Licensing feature name.

NON_SHARED_DISTRIBUTION

Syntax

NON_SHARED_DISTRIBUTION=*service_domain_name* ([*project_name number_non_shared_licenses*] ...) ...

service_domain_name

Specify a License Scheduler service domain (described in the ServiceDomain section) that distributes the licenses.

project_name

Specify a License Scheduler project (described in the section) that is allowed to use the licenses.

number_non_shared_licenses

Specify a positive integer representing the number of non-shared licenses that the project owns.

Description

Optional. Defines non-shared licenses. Non-shared licenses are privately owned, and are not shared with other license projects. They are available only to one project.

Used for project mode and fast dispatch project mode only.

Use **blinfo -a** to display **NON_SHARED_DISTRIBUTION** information.

For projects defined with **NON_SHARED_DISTRIBUTION**, you must assign the project **OWNERSHIP** an equal or greater number of tokens than the number of non-shared licenses. If the number of owned licenses is less than the number of non-shared licenses, **OWNERSHIP** is set to the number of non-shared licenses.

Examples

- If the number of tokens normally given to a project (to satisfy the **DISTRIBUTION** share ratio) is larger than its **NON_SHARED_DISTRIBUTION** value, the **DISTRIBUTION** share ratio takes effect first.

  ```
  Begin Feature
  NAME=f1 # total 15 on LanServer
  FLEX_NAME=VCS-RUNTIME
  DISTRIBUTION=LanServer(Lp1 4/10 Lp2 1)
  NON_SHARED_DISTRIBUTION=LanServer(Lp1 10)
  End Feature
  ```

  In this example, 10 non-shared licenses are defined for the Lp1 project on LanServer. The **DISTRIBUTION** share ratio for Lp1:Lp2 is 4:1. If there are 15 licenses, Lp1 will normally get 12 licenses, which is larger than its **NON_SHARED_DISTRIBUTION** value of 10. Therefore, the **DISTRIBUTION** share ratio takes effect, so Lp1 gets 12 licenses and Lp2 gets 3 licenses for the 4:1 share ratio.
- If the number of tokens normally given to a project (to satisfy the **DISTRIBUTION** share ratio) is smaller than its **NON_SHARED_DISTRIBUTION** value, the project will first get the number of tokens equal to **NON_SHARED_DISTRIBUTION**, then the **DISTRIBUTION** share ratio for the other projects takes effect for the remaining licenses.
  - For one project with non-shared licenses and one project with no non-shared licenses: , the project with no non-shared licenses is given all the remaining licenses since it would normally be given more according to the **DISTRIBUTION** share ratio:

```
Begin Feature
NAME=f1 # total 15 on LanServer
FLEX_NAME=VCS-RUNTIME
DISTRIBUTION=LanServer(Lp1 1/10 Lp2 4)
NON_SHARED_DISTRIBUTION=LanServer(Lp1 10)
End Feature
```

In this example, 10 non-shared licenses are defined for the Lp1 project on LanServer. The **DISTRIBUTION** share ratio for Lp1:Lp2 is 1:4. If there are 15 licenses, Lp1 will normally get three licenses, which is smaller than its **NON_SHARED_DISTRIBUTION** value of 10. Therefore, Lp1 gets the first 10 licenses, and Lp2 gets the remaining five licenses (since it would normally get more according to the share ratio).

- For one project with non-shared licenses and two or more projects with no non-shared licenses, the two projects with no non-shared licenses are assigned the remaining licenses according to the **DISTRIBUTION** share ratio with each other, ignoring the share ratio for the project with non-shared licenses.

```
Begin Feature
NAME=f1 # total 15 on LanServer
FLEX_NAME=VCS-RUNTIME
DISTRIBUTION=LanServer(Lp1 1/10 Lp2 4 Lp3 2)
NON_SHARED_DISTRIBUTION=LanServer(Lp1 10)
End Feature
```

In this example, 10 non-shared licenses are defined for the Lp1 project on LanServer. The **DISTRIBUTION** share ratio for Lp1:Lp2:Lp3 is 1:4:2. If there are 15 licenses, Lp1 will normally get two licenses, which is smaller than its **NON_SHARED_DISTRIBUTION** value of 10. Therefore, Lp1 gets the first 10 licenses. The remaining licenses are given to Lp2 and Lp3 to a ratio of 4:2, so Lp2 gets three licenses and Lp3 gets two licenses.

- For two projects with non-shared licenses and one with no non-shared licenses, the one project with no non-shared licenses is given the remaining licenses after the two projects are given their non-shared licenses:

```
Begin Feature
NAME=f1 # total 15 on LanServer
FLEX_NAME=VCS-RUNTIME
DISTRIBUTION=LanServer(Lp1 1/10 Lp2 4 Lp3 2/5)
NON_SHARED_DISTRIBUTION=LanServer(Lp1 10 Lp3 5)
End Feature
```

In this example, 10 non-shared licenses are defined for the Lp1 project and five non-shared license are defined for the Lp3 project on LanServer. The **DISTRIBUTION** share ratio for Lp1:Lp2:Lp3 is 1:4:2. If there are 15 licenses, Lp1 will normally get two licenses and Lp3 will normally get four licenses, which are both smaller than their corresponding **NON_SHARED_DISTRIBUTION** values. Therefore, Lp1 gets 10 licenses and Lp3 gets five licenses. Lp2 gets no licenses even though it normally has the largest share because Lp1 and Lp3 have non-shared licenses.

PEAK_INUSE_PERIOD

Syntax

PEAK_INUSE_PERIOD=*seconds* | *cluster seconds* ...

Description

Defines the interval over which a peak **INUSE** value is determined for dynamic license allocation in cluster mode for this license features and service domain.

Use keyword default to set for all clusters not specified, and the keyword interactive (in place of cluster name) to set for **taskman** jobs. For example:

```
PEAK_INUSE_PERIOD = cluster1 1000 cluster2 700 default 300
```

Used for cluster mode only.

When defined in both the Parameters section and the Feature section, the Feature section definition is used for that license feature.

Default

300 seconds

PREEMPT_ORDER

Syntax

```
PREEMPT_ORDER=BY_OWNERSHIP
```

Description

Optional. Sets the preemption order based on configured **OWNERSHIP**.

Used for project mode only.

Default

Not defined.

PREEMPT_RESERVE

Syntax

```
PREEMPT_RESERVE=Y | N
```

Description

Optional. If PREEMPT_RESERVE=Y, enables License Scheduler to preempt either licenses that are reserved or already in use by other projects. The number of jobs must be greater than the number of licenses owned.

If PREEMPT_RESERVE=N, License Scheduler does not preempt reserved licenses.

Used for project mode only.

Default

Y. Reserved licenses are preemptable.

RETENTION_FACTOR

Syntax

RETENTION_FACTOR=*integer%*

Description

Ensures that when tokens are reclaimed from an overfed cluster, the overfed cluster still gets to dispatch additional jobs, but at a reduced rate. Specify the retention factor as a percentage of tokens to be retained by the overfed cluster.

For example:

```
Begin Feature
NAME = f1
CLUSTER_MODE = Y
CLUSTER_DISTRIBUTION = LanServer(LAN1 1 LAN2 1)
ALLOC_BUFFER = 20
RETENTION_FACTOR = 25%
End Feature
```

With RETENTION_FACTOR set, as jobs finish in the overfed cluster and free up tokens, at least 25% of the tokens can be reused by the cluster to dispatch additional jobs. Tokens not held by the cluster are redistributed to other clusters. In general, a higher value means that the process of reclaiming tokens from an overfed cluster takes longer, and an overfed cluster gets to dispatch more jobs while tokens are being reclaimed from it.

Used for cluster mode only.

Default

Not defined

SERVICE_DOMAINS

Syntax

SERVICE_DOMAINS=*service_domain_name* ...

service_domain_name

Specify the name of the service domain.

Description

Required if GROUP_DISTRIBUTION is defined. Specifies the service domains that provide tokens for this feature.

Only a single service domain can be specified when using cluster mode or fast dispatch project mode.

WORKLOAD_DISTRIBUTION

Syntax

WORKLOAD_DISTRIBUTION=[*service_domain_name*(LSF *lsf_distribution* NON_LSF *non_lsf_distribution*)] ...

service_domain_name

Specify a License Scheduler service domain (described in the ServiceDomain section) that distributes the licenses.

lsf_distribution

Specify the share of licenses dedicated to LSF workloads. The share of licenses dedicated to LSF workloads is a ratio of *lsf_distribution*:*non_lsf_distribution*.

non_lsf_distribution

Specify the share of licenses dedicated to non-LSF workloads. The share of licenses dedicated to non-LSF workloads is a ratio of *non_lsf_distribution*:*lsf_distribution*.

Description

Optional. Defines the distribution given to each LSF and non-LSF workload within the specified service domain.

When running in cluster mode, **WORKLOAD_DISTRIBUTION** can only be specified for WAN service domains; if defined for a LAN feature, it is ignored.

Use **blinfo -a** to display **WORKLOAD_DISTRIBUTION** configuration.

Example

```
Begin Feature
NAME=ApplicationX
DISTRIBUTION=LicenseServer1(Lp1 1 Lp2 2)
WORKLOAD_DISTRIBUTION=LicenseServer1(LSF 8 NON_LSF 2)
End Feature
```

On the LicenseServer1 domain, the available licenses are dedicated in a ratio of 8:2 for LSF and non-LSF workloads. This means that 80% of the available licenses are dedicated to the LSF workload, and 20% of the available licenses are dedicated to the non-LSF workload.

If LicenseServer1 has a total of 80 licenses, this configuration indicates that 64 licenses are dedicated to the LSF workload, and 16 licenses are dedicated to the non-LSF workload.

FeatureGroup section

Description

Optional. Collects license features into groups. Put FeatureGroup sections after Feature sections in lsf.licensescheduler.

The FeatureGroup section is supported in both project mode and cluster mode.

FeatureGroup section structure

The FeatureGroup section begins and ends with the lines Begin FeatureGroup and End FeatureGroup. Feature group definition consists of a unique name and a list of features contained in the feature group.

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
```

Parameters

- NAME
- FEATURE_LIST

NAME

Required. Defines the name of the feature group. The name must be unique.

FEATURE_LIST

Required. Lists the license features contained in the feature group.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.

ProjectGroup section

Description

Optional. Defines the hierarchical relationships of projects.

Used for project mode only. When running in cluster mode, any ProjectGroup sections are ignored.

The hierarchical groups can have multiple levels of grouping. You can configure a tree-like scheduling policy, with the leaves being the license projects that jobs can belong to. Each project group in the tree has a set of values, including shares, limits, ownership and non-shared, or exclusive, licenses.

Use **blstat -G** to view the hierarchical dynamic license information.

Use **blinfo -G** to view the hierarchical configuration.

ProjectGroup section structure

Define a section for each hierarchical group managed by License Scheduler.

The keywords GROUP, SHARES, OWNERSHIP, LIMIT, and NON_SHARED are required. The keywords PRIORITY and DESCRIPTION are optional. Empty brackets are allowed only for OWNERSHIP, LIMIT, and PRIORITY. SHARES must be specified.

```
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
```

If desired, ProjectGroup sections can be completely independent, without any overlapping projects.

```
Begin ProjectGroup
GROUP               SHARES  OWNERSHIP LIMITS  NON_SHARED(digital_sim (sim sim_reg)) (40 60)  (100 0)
End ProjectGroup
Begin ProjectGroup
GROUP               SHARES  OWNERSHIP LIMITS  NON_SHARED
(analog_sim (app1 multitoken app1_reg)) (50 10 40)  (65 25 0) (- 50 -) ()
End ProjectGroup
```

Parameters

- DESCRIPTION
- GROUP
- LIMITS
- NON_SHARED
- OWNERSHIP
- PRIORITY
- SHARES

DESCRIPTION

Optional. Description of the project group.

The text can include any characters, including white space. The text can be extended to multiple lines by ending the preceding line with a backslash (\). The maximum length for the text is 64 characters. When the DESCRIPTION column is not empty it should contain one entry for each project group member.

For example:

```
GROUP       SHARES OWNERSHIP   LIMITS NON_SHARED  DESCRIPTION
(R (A B))   (1 1)  ()          ()     (10 10)     ()
(A (p1 p2)) (1 1)  (40 60)     ()     ()          ("p1 desc." "")
(B (p3 p4)) (1 1)  ()          ()     ()          ("p3 desc." "p4 desc.")
```

Use **blinfo -G** to view hierarchical project group descriptions.

GROUP

Defines the project names in the hierarchical grouping and its relationships. Each entry specifies the name of the hierarchical group and its members.

For better readability, you should specify the projects in the order from the root to the leaves as in the example.

Specify the entry as follows:

(*group* (*member* ...))

LIMITS

Defines the maximum number of licenses that can be used at any one time by the hierarchical group member projects. Specify the maximum number of licenses for each member, separated by spaces, in the same order as listed in the GROUP column.

A dash (-) is equivalent to INFINIT_INT, which means there is no maximum limit and the project group can use as many licenses as possible.

You can leave the parentheses empty () if desired.

NON_SHARED

Defines the number of licenses that the hierarchical group member projects use exclusively. Specify the number of licenses for each group or project, separated by spaces, in the same order as listed in the GROUP column.

A dash (-) is equivalent to a zero, which means there are no licenses that the hierarchical group member projects use exclusively.

For hierarchical project froups in fast dispatch project mode, License Scheduler ignores the NON_SHARED value configured for project groups, and only uses the NON_SHARED value for the child projects. The project group's NON_SHARED value is the sum of the NON_SHARED values of its child projects.

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)
(G1 (AP2 AP1))  (1 1)     ()          ()          (1 1)
```

Valid values

Any positive integer up to the LIMITS value defined for the specified hierarchical group.

If defined as greater than LIMITS, NON_SHARED is set to LIMITS.

OWNERSHIP

Defines the level of ownership of the hierarchical group member projects. Specify the ownership for each member, separated by spaces, in the same order as listed in the GROUP column.

You can only define OWNERSHIP for hierarchical group member projects, not hierarchical groups. Do not define OWNERSHIP for the top level (root) project group. Ownership of a given internal node is the sum of the ownership of all child nodes it directly governs.

A dash (-) is equivalent to a zero, which means there are no owners of the projects. You can leave the parentheses empty () if desired.

Valid values

A positive integer between the NON_SHARED and LIMITS values defined for the specified hierarchical group.

- If defined as less than NON_SHARED, OWNERSHIP is set to NON_SHARED.
- If defined as greater than LIMITS, OWNERSHIP is set to LIMITS.

PRIORITY

Optional. Defines the priority assigned to the hierarchical group member projects. Specify the priority for each member, separated by spaces, in the same order as listed in the GROUP column.

“0” is the lowest priority, and a higher number specifies a higher priority. This column overrides the default behavior. Instead of preempting based on the

accumulated inuse usage of each project, the projects are preempted according to the specified priority from lowest to highest.

By default, priorities are evaluated top down in the project group hierarchy. The priority of a given node is first decided by the priority of the parent groups. When two nodes have the same priority, priority is determined by the accumulated inuse usage of each project at the time the priorities are evaluated. Specify LS_PREEMPT_PEER=Y in the Parameters section to enable bottom-up license token preemption in hierarchical project group configuration.

A dash (-) is equivalent to a zero, which means there is no priority for the project. You can leave the parentheses empty () if desired.

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

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.

SHARES

Required. Defines the shares assigned to the hierarchical group member projects. Specify the share for each member, separated by spaces, in the same order as listed in the GROUP column.

Projects section

Description

Required for project mode only. Ignored in cluster mode. Lists the License Scheduler projects.

Projects section structure

The Projects section begins and ends with the lines `Begin Projects` and `End Projects`. The second line consists of the required column heading `PROJECTS` and the optional column heading `PRIORITY`. Subsequent lines list participating projects, one name per line.

Examples

The following example lists the projects without defining the priority:

```
Begin Projects
PROJECTS
Lp1
Lp2
Lp3
Lp4
...
End Projects
```

The following example lists the projects and defines the priority of each project:

```
Begin Projects
PROJECTS        PRIORITY
Lp1             3
Lp2             4
Lp3             2
```

```
Lp4             1
default         0
...
End Projects
```

Parameters

- DESCRIPTION
- PRIORITY
- PROJECTS

DESCRIPTION

Optional. Description of the project.

The text can include any characters, including white space. The text can be extended to multiple lines by ending the preceding line with a backslash (\). The maximum length for the text is 64 characters.

Use **blinfo -Lp** to view the project description.

PRIORITY

Optional. Defines the priority for each project where "0" is the lowest priority, and the higher number specifies a higher priority. This column overrides the default behavior. Instead of preempting in order the projects are listed under PROJECTS based on the accumulated inuse usage of each project, the projects are preempted according to the specified priority from lowest to highest.

Used for project mode only.

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

Use **blinfo -Lp** to view project priority information.

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.

PROJECTS

Defines the name of each participating project. Specify using one name per line.

Automatic time-based configuration

Variable configuration is used to automatically change License Scheduler license token distribution policy configuration based on time windows. You define automatic configuration changes in `lsf.licensescheduler` by using if-else constructs and time expressions in the Feature section. After you change the file, check the configuration with the **bladmin ckconfig** command, and restart License Scheduler in the cluster with the **bladmin reconfig** command.

Used for both project mode and cluster mode.

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

When LSF determines a feature has been added, removed, or changed, mbatchd no longer restarts automatically. Instead a message indicates that a change has been detected, prompting the user to restart manually with **badmin mbdrestart**.

This affects automatic time-based configuration in the Feature section of `lsf.licensescheduler`. When **mbatchd** detects a change in the Feature configuration, you must restart **mbatchd** for the change to take effect.

Example

```
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
```

bladmin

Administrative tool for License Scheduler.

Synopsis

bladmin *subcommand*

bladmin [**-h** | **-V**]

Description

bladmin provides a set of subcommands to control License Scheduler.

You must be root or a License Scheduler administrator to use this command.

Subcommand synopsis

ckconfig

reconfig [*host_name* ... | **all**]

shutdown [*host_name* ... | **all**]

blddebug [**-c** *class_name* ...] [**-l** *debug_level*] [**-f** *logfile_name*] [**-o**]

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

-h

-V

Usage

ckconfig

Checks License Scheduler configuration in `LSF_ENVDIR/lsf.licensescheduler` and `lsf.conf`.

By default, **bladmin ckconfig** displays only the result of the configuration file check. If warning errors are found, **bladmin** prompts you to enter "y" to display detailed messages.

reconfig [*host_name* **... | all]**

Reconfigures License Scheduler.

shutdown [*host_name* **... | all]**

Shuts down License Scheduler.

blddebug [-c *class_name* **...] [-l** *debug_level***] [-f** *logfile_name***] [-o]**

Sets the message log level for **bld** to include additional information in log files. You must be root or the LSF administrator to use this command.

If the **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*

-c *class_name* **...**

Specifies software classes for which debug messages are to be logged.

Format of *class_name* is the name of a class, or a list of class names separated by spaces and enclosed in quotation marks. Classes are also listed in lsf.h.

Valid log classes:

- LC_AUTH: Log authentication messages
- LC_COMM: Log communication messages
- LC_FLEX: Log everything related to FLEX_STAT or FLEX_EXEC Flexera APIs
- LC_LICENCE: Log license management messages
- LC_PREEMPT: Log preemption policy messages
- LC_RESREQ: Log resource requirement messages
- LC_TRACE: Log significant program walk steps
- LC_XDR: Log everything transferred by XDR

Default: 0 (no additional classes are logged)

-l *debug_level*

Specifies level of detail in debug messages. The higher the number, the more detail that is logged. Higher levels include all lower logging levels. For example, LOG_DEBUG3 includes LOG_DEBUG2 LOG_DEBUG1, and LOG_DEBUG levels.

Possible values:

0 LOG_DEBUG level in parameter LS_LOG_MASK in lsf.conf.

1 LOG_DEBUG1 level for extended logging.

2 LOG_DEBUG2 level for extended logging.

3 LOG_DEBUG3 level for extended logging.

Default: 0 (LOG_DEBUG level in parameter LS_LOG_MASK)

-f *logfile_name*

Specifies the name of the file where debugging messages are logged. The file name can be a full path. If a file name without a path is specified, the file is saved in the LSF system log directory.

The name of the file has the following format:

logfile_name.daemon_name.log.*host_name*

On UNIX, if the specified path is not valid, the log file is created in the /tmp directory.

On Windows, if the specified path is not valid, no log file is created.

Default: current LSF system log file in the LSF system log file directory.

-o

Turns off temporary debug settings and resets 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 LSB_DEBUG_BLD. The log file is also reset back to the default log file.

blcdebug [-l *debug_level***] [-f** *logfile_name***] [-o]** *collector_name* **| all**

Sets the message log level for **blcollect** to include additional information in log files. You must be root or the LSF administrator to use this command.

If the **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

-l *debug_level*

Specifies level of detail in debug messages. The higher the number, the more detail that is logged. Higher levels include all lower logging levels. For example, LOG_DEBUG3 includes LOG_DEBUG2 LOG_DEBUG1, and LOG_DEBUG levels.

Possible values:

0 LOG_DEBUG level in parameter LS_LOG_MASK in lsf.conf.

1 LOG_DEBUG1 level for extended logging.

2 LOG_DEBUG2 level for extended logging.

3 LOG_DEBUG3 level for extended logging.

Default: 0 (LOG_DEBUG level in parameter LS_LOG_MASK)

-f *logfile_name*

Specifies the name of the file where debugging messages are logged. The file name can be a full path. If a file name without a path is specified, the file is saved in the LSF system log directory.

The name of the file has the following format:

logfile_name.daemon_name.log.*host_name*

On UNIX, if the specified path is not valid, the log file is created in the /tmp directory.

On Windows, if the specified path is not valid, no log file is created.

Default: current LSF system log file in the LSF system log file directory.

-o

Turns off temporary debug settings and resets 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 LSB_DEBUG_BLD. The log file is also reset back to the default log file.

If a collector name is not specified, default value is to restore the original log mask and log file directory for the default collector.

collector_name **...** | **all**

Specifies the collector names separated by blanks. all means all the collectors.

-h

Prints command usage to `stderr` and exits.

-V

Prints release version to `stderr` and exits.

See also

blhosts, `lsf.licensescheduler`, `lsf.conf`

---

blcollect

license information collection daemon that collects license usage information

Synopsis

blcollect -c *collector_name* **-m** *host_name [...]* **-p** *license_scheduler_port* [**-i** *lmstat_interval* | **-D** *lmstat_path*] [**-t** *timeout*]

blcollect [**-h** | **-V**]

Description

Periodically collects license usage information from Flexera FlexNet. It queries FlexNet for license usage information from the FlexNet **lmstat** command, and passes the information to the License Scheduler daemon (bld). The **blcollect** daemon improves performance by allowing you to distribute license information queries on multiple hosts.

By default, license information is collected from FlexNet on one host. Use **blcollect** to distribute the license collection on multiple hosts.

For each service domain configuration in `lsf.licensescheduler`, specify one name for **blcollect** to use. You can only specify one collector per service domain, but you can specify one collector to serve multiple service domains. You can choose any collector name you want, but must use that exact name when you run **blcollect**.

Options

-c

Required. Specify the collector name you set in `lsf.licensescheduler`. You must use the collector name (`LIC_COLLECTOR`) you define in the `ServiceDomain` section of the configuration file.

-m

Required. Specifies a space-separated list of hosts to which license information is sent. The hosts do not need to be running License Scheduler or a FlexNet. Use fully qualified host names.

-p

Required. You must specify the License Scheduler listening port, which is set in `lsf.licensescheduler` and has a default value of 9581.

-i *lmstat_interval*

Optional. The frequency in seconds of the calls that License Scheduler makes to **lmstat** to collect license usage information from FlexNet.

The default interval is 60 seconds.

-D *lmstat_path*

Optional. Location of the FlexNet command **lmstat**.

-t *timeout*

Optional. Timout value passed to the FlexNet command **lmstat**, overwriting the value defined by **LM_STAT_TIMEOUT** in the Parameters or ServiceDomain section of the `lsf.licensescheduler` file.

-h

Prints command usage to `stderr` and exits.

-V

Prints release version to `stderr` and exits.

See also

lsf.licensescheduler

blcstat

displays dynamic **blcollect** update information for License Scheduler.

Synopsis

blcstat [**-l**] [*collector_name* ...]

blcstat [**-h** | **-V**]

Description

Displays the time each license collector daemon (**bcollect**) last sent an update to **bld**, along with the current status of each **blcollect**.

Options

-l

Long format. Displays detailed information for each **blcollect** in a multiline format.

collector_name

Displays information only for the specified **blcollect** daemons.

-h

Prints command usage to stderr and exits.

-V

Prints the release version to stderr and exits.

Output

COLLECTOR_NAME

The name of the license collector daemon as defined by LIC_COLLECTOR=*license_collector_name* in the ServiceDomain sections of the lsf.licensescheduler file. By default, the name is _default_.

STATUS

The current status of the collector.

- ok: The collector is working and all license servers can be reached.
- -ok: The collector is working, however, not all licenses servers can be reached
- unavail: The collector cannot be reached.

LAST_UPD_TIME

The time the last update was received by **bld** for this collector.

-l Output

The -l option displays a long format listing with the following additional fields:

HOST_NAME

The name of the host running this collector.

LICENSE_SERVER

The license server configured in the ServiceDomain section lsf.licensescheduler for this collector.

Multiple lines indicate multiple license servers.

Multiple entries in one line separated by '|' indicate configured redundant license servers (sharing the same license file).

License server state is one of:

- reachable: The license server is running and providing information to **lmstat**.
- unreachable: The license server is not running, or some other problem has blocked the flow of information to **lmstat**.
- unknown: **blcollect** is down.

FEATURES

The names of features running on license servers for this collector.

LMSTAT_INTERVAL

The interval between updates from this collector as set by the **LM_STAT_INTERVAL** parameter in the Parameters or ServiceDomain section of the `lsf.licensescheduler` file, or by blcollect at collector startup.

See also

blcollect

blhosts

displays the names of all the hosts running the License Scheduler daemon (`bld`)

Synopsis

blhosts [**-h** | **-V**]

Description

Displays a list of hosts running the License Scheduler daemon. This includes the License Scheduler master host and all the candidate License Scheduler hosts running bld.

Options

-h

Prints command usage to `stderr` and exits.

-V

Prints release version to `stderr` and exits.

Output

Prints out the names of all the hosts running the License Scheduler daemon (bld).

For example, the following sample output shows the License Scheduler master host and two candidate License Scheduler hosts running bld:

```
bld is running on:
master: host1.domain1.com
slave: host2.domain1 host3.domain1
```

See also

blinfo, **blstat**, **bladmin**

blinfo

Displays static License Scheduler configuration information

Synopsis

blinfo -Lp | **-p** | **-D** | **-G** | **-P**

blinfo [**-a** [**-t** *token_name* | **"***token_name* ...**"**]] [**-o alpha** | **total**] [**-g "***feature_group* ...**"**]

blinfo -A [**-t** *token_name* | "*token_name* ..."] [**-o alpha** | **total**] [**-g** "*feature_group* ..."]

blinfo -C [**-t** *token_name* | "*token_name* ..."] [**-o alpha** | **total**] [**-g** "*feature_group* ..."]

blinfo [**-t** *token_name* | "*token_name* ..."] [**-o alpha** | **total**] [**-g** "*feature_group* ..."]

blinfo [**-h** | **-V**]

Description

Displays different license configuration information, depending on the option selected.

By default, displays information about the distribution of licenses managed by License Scheduler.

Options (cluster mode and project mode)

-a

Shows all information, including information about non-shared licenses (**NON_SHARED_DISTRIBUTION**) and workload distribution (**WORKLOAD_DISTRIBUTION**).

You can optionally provide license token names.

blinfo -a does not display **NON_SHARED** information for hierarchical project group scheduling policies. Use **blinfo -G** to see hierarchical group configuration.

-C

Shows the cluster locality information for the features.

You can optionally provide license token names.

-D

Lists the License Scheduler service domains and the corresponding FlexNet license server hosts.

-g *feature_group* **...**

When **FEATURE_GROUP** is configured for a group of license features in `lsf.licensescheduler`, shows only information about the features configured in the **FEATURE_LIST** of specified feature groups. You can specify more than one feature group at one time.

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

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

-o alpha | total

Sorts license feature information by total tokens.

- alpha: Features are listed in descending alphabetical order.
- total: Features are sorted by the descending order of 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 included in this amount.

-p

Displays values of `lsf.licensescheduler` configuration parameters and `lsf.conf` parameters related to License Scheduler. This is useful for troubleshooting.

-t *token_name* | **"***token_name* **..."**

Only shows information about specified license tokens. Use spaces to separate multiple names, and enclose them in quotation marks.

-P

When `LS_FEATURE_PERCENTAGE=Y` or `LS_ACTIVE_PERCENTAGE=Y`, lists the license ownership (if applicable) in percentage.

-h

Prints command usage to `stderr` and exits.

-V

Prints the License Scheduler release version to `stderr` and exits.

Options (project mode only)

-A

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

You can optionally provide license token names.

-G

Lists the hierarchical configuration information.

If **PRIORITY** is defined in the `ProjectGroup` Section of `lsf.licensescheduler`, this option also shows the priorities of each project.

-Lp

Lists the active projects managed by License Scheduler.

`-Lp` only displays projects associated with configured features.

If **PRIORITY** is defined in the `Projects` Section of `lsf.licensescheduler`, this option also lists the priorities of each project.

Default output

Displays the following fields:

FEATURE

The license name. This becomes the license token name.

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

MODE

The mode of the license:

Cluster
: Cluster mode

Project
: Project mode

SERVICE_DOMAIN

The name of the service domain that provided the license.

TOTAL

The total number of licenses managed by FlexNet. This number comes from FlexNet.

DISTRIBUTION

The distribution of the licenses among license projects in the format [*project_name*, *percentage*[/*number_licenses_owned*]]. This determines how many licenses a project is entitled to use when there is competition for licenses. The percentage is calculated from the share specified in the configuration file.

All output (-a)

As default output, plus all other feature-level parameters defined for each feature.

Cluster locality output (-C)

NAME

The license feature token name.

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

FLEX_NAME

The actual FlexNet feature name—the name used by FlexNet to identify the type of license. May be different from the License Scheduler token name if a different **FLEX_NAME** is specified in `lsf.licensescheduler`.

CLUSTER_NAME

The name of the cluster the feature is assigned to.

FEATURE

The license feature name. This becomes the license token name.

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

SERVICE_DOMAIN

The service domain name.

Service Domain Output (-D)

SERVICE_DOMAIN

The service domain name.

LIC_SERVERS

Names of FlexNet license server hosts that belong the to service domain. Each host name is enclosed in parentheses, as shown:

(*port_number@host_name*)

Redundant hosts (that share the same FlexNet license file) are grouped together as shown:

(*port_number@host_name port_number@host_name port_number@host_name*)

Parameters Output (-p)

Displays values set in the Parameters section of `lsf.licensescheduler`.

Displays the following parameter values from `lsf.conf`:

LS_LOG_MASK or LOG_MASK

Specifies the logging level of error messages for License Scheduler daemons. If LS_LOG_MASK is not defined in `lsf.licensescheduler`, 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.

For example:

```
LS_LOG_MASK=LOG_DEBUG
```

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 only useful for debugging.

LSF_LIC_SCHED_HOSTS

List of hosts that are candidate License Scheduler hosts. Defined in `lsf.conf`.

LSF_LIC_SCHED_PREEMPT_REQUEUE

Specifies whether to requeue or suspend a job whose license is preempted by License Scheduler. Defined in `lsf.conf`.

LSF_LIC_SCHED_PREEMPT_SLOT_RELEASE

Specifies whether to release the resources of a job that is suspended when its license is preempted by License Scheduler. Defined in `lsf.conf`.

LSF_LIC_SCHED_PREEMPT_STOP

Specifies whether to use job controls to stop a job that is preempted. Defined in `lsf.conf`.

Allocation output (-A, project mode)

FEATURE

The license name. This becomes the license token name.

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

PROJECT

The License Scheduler project name.

ALLOCATION

The percentage of shares assigned to each cluster for a feature and a project.

Hierarchical Output (-G, project mode)

The following fields describe the values of their corresponding configuration fields in the `ProjectGroup` Section of `lsf.licensescheduler`.

GROUP

The project names in the hierarchical grouping and its relationships. Each entry specifies the name of the hierarchical group and its members. The entry is enclosed in parentheses as shown:

(*group* (*member* ...))

SHARES

The shares assigned to the hierarchical group member projects.

OWNERSHIP

The number of licenses that each project owns.

LIMITS

The maximum number of licenses that the hierarchical group member project can use at any one time.

NON_SHARED

The number of licenses that the hierarchical group member projects use exclusively.

PRIORITY

The priority of the project if it is different from the default behavior. A larger number indicates a higher priority.

DESCRIPTION

The description of the project group.

Project Output (-Lp, project mode)

List of active License Scheduler projects.

`-Lp` only displays projects associated with configured features.

PROJECT

The project name.

PRIORITY

The priority of the project if it is different from the default behavior. A larger number indicates a higher priority.

DESCRIPTION

The description of the project.

Examples

blinfo -a (project mode) displays both **NON_SHARED_DISTRIBUTION** and **WORKLOAD_DISTRIBUTION** information when they are defined:

```
blinfo -a
FEATURE      SERVICE_DOMAIN  TOTAL  DISTRIBUTION
g1           LS              3      [p1, 50.0%] [p2, 50.0% / 2]
                                    NON_SHARED_DISTRIBUTION
```

```
                                        [p2, 2]
                                        WORKLOAD_DISTRIBUTION
                                        [LSF 66.7%, NON_LSF 33.3%]
```

Files

Reads `lsf.licensescheduler`

See also

blstat, **blusers**, `lsf.licensescheduler`, `lsf.conf`

blkill

terminates an interactive (taskman) License Scheduler task

Synopsis

blkill [**-t** *seconds*] *task_ID*

blkill [**-h** | **-V**]

Description

Terminates a running or waiting interactive task in License Scheduler.

Users can kill their own tasks. You must be a License Scheduler administrator to terminate another user's task.

By default, **blkill** notifies the user and waits 60 seconds before killing the task.

Options

task_ID

Task ID of the task you want to kill.

-t *seconds*

Specify how many seconds to delay before killing the task. A value of 0 means to kill the task immediately (do not give the user any time to save work).

-h

Prints command usage to `stderr` and exits.

-V

Prints License Scheduler release version to `stderr` and exits.

blparams

displays information about configurable License Scheduler parameters defined in the files `lsf.licensescheduler` and `lsf.conf`

Synopsis

blparams [**-h** | **-V**]

Description

Displays values set in the Parameters section of `lsf.licensescheduler`.

Displays the following parameter values from `lsf.conf`:

LS_LOG_MASK or LOG_MASK

Specifies the logging level of error messages for License Scheduler daemons. If LS_LOG_MASK is not defined in `lsf.licensescheduler`, 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.

For example:

```
LS_LOG_MASK=LOG_DEBUG
```

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 only useful for debugging.

LSF_LIC_SCHED_HOSTS

List of hosts that are candidate License Scheduler hosts. Defined in `lsf.conf`.

LSF_LIC_SCHED_PREEMPT_REQUEUE

Specifies whether to requeue or suspend a job whose license is preempted by License Scheduler. Defined in `lsf.conf`.

LSF_LIC_SCHED_PREEMPT_SLOT_RELEASE

Specifies whether to release the slot of a job that is suspended when its license is preempted by License Scheduler. Defined in `lsf.conf`.

LSF_LIC_SCHED_PREEMPT_STOP

Specifies whether to use job controls to stop a job that is preempted. Defined in `lsf.conf`.

Options

-h

Prints command usage to `stderr` and exits.

-V

Prints LSF release version to `stderr` and exits.

See also

`lsf.licensescheduler`, `lsf.conf`

blstat

displays dynamic license information

Synopsis

blstat [**-s**] [**-S**] [**-D** *service_domain_name* | "*service_domain_name* ..."] [**-P**][**-t** *token_name* | "*token_name* ..."] [**-o alpha** | **total** | **avail**] [**-g** "*feature_group* ..."] [**-slots**]

blstat [**-a**] [**-c** *token_name*] [**-G**] [**-Lp** *ls_project_name* | "*ls_project_name* ..."]

blstat [**-h** | **-V**]

Description

Displays license usage statistics for License Scheduler.

By default, shows information about all licenses and all clusters.

Options (cluster mode and project mode)

-S

Displays information on the license servers associated with license features.

-s

Displays license usage of the LSF and non-LSF workloads. 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.

-D *service_domain_name* | "*service_domain_name* ..."

Only shows information about specified service domains. Use spaces to separate multiple names, and enclose them in quotation marks.

-g *feature_group* ...

When **FEATURE_GROUP** is configured for a group of license features in `lsf.licensescheduler`, shows information about features configured in the **FEATURE_LIST** of specified feature groups. You can specify more than one feature group.

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

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

-slots

Displays how many slots are using currently by License Scheduler jobs (`Current job slots in use`) and the peak number of slots in use (`Peak job slots used`).

-o alpha | total | avail

Sorts license feature information alphabetically, by total licenses, or by available licenses.

- alpha: Features are listed in descending alphabetical order.
- total: Features are sorted by the descending order of 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 not included in this amount.

- avail: Features are sorted by descending order of licenses available, including free tokens.

-P

Displays percentage values for INUSE and RESERVE. The percentage value represents the number of tokens this project has used and reserved compared to total number of licenses.

-t *token_name* | **"***token_name* **..."**

Only shows information about specified license tokens. Use spaces to separate multiple names, and enclose them in quotation marks.

-h

Prints command usage to `stderr` and exits.

-V

Prints the release version to `stderr` and exits.

Options (project mode only)

-a

Displays each project group's accumulated value of licenses. The license token dispatching order is based on the sort order, which is based on the scaled accumulate value of each project. The lower the value, the sooner the license token is dispatched to that project.

-c *token_name*

Displays cross cluster information for tokens.

In project mode, the information is sorted by the value of **SCALED_ACUM**. The first cluster listed receives tokens first.

Information displayed includes token usage, reserved tokens, free tokens, demand for tokens, accumulated value of tokens, and scaled accumulate value of tokens in each cluster.

For fast dispatch project mode, also displays the actual and ideal number of tokens allocated to the cluster:

- TARGET: The ideal amount of licenses allocated to the cluster
- OVER: The number of licenses checked out by RUN jobs in the cluster under the license projects in excess of the rusage
- FREE: The number of license allocated to the cluster but not used.
- DEMAND: The number of tokens required by the cluster under the license project

-G

Displays dynamic hierarchical license information.

blstat -G also works with the `-t` option to only display hierarchical information for the specified feature names.

-Lp *ls_project_name* | **"***ls_project_name* **..."**

Shows project description for specified projects (non-hierarchical). Use spaces to separate multiple names and enclose them in quotation marks.

If project group paths are enabled (`PROJECT_GROUP_PATH=Y` in `lsf.licensescheduler`), **blstat -Lp** displays the license projects associated with

the specified project for all features. **blstat -Lp -t** displays the associated license projects for the specified feature. If the parameter is disabled, only the specified project is displayed.

Output

Information is organized first by license feature, then by service domain. For each combination of license and service domain, License Scheduler displays a line of summary information followed by rows of license project or cluster information.

In each group of statistics, numbers and percentages refer only to licenses of the specified license feature that can be checked out from FlexNet license server hosts in the specified service domain.

Cluster mode summary output

FEATURE

The license name. (This appears only once for each feature.)

SERVICE_DOMAIN

The name of the service domain that provided the license.

TOTAL_TOKENS

The number of licenses from this service domain reserved for License Scheduler jobs.

TOTAL_ALLOC

The number of licenses from this service domain allocated to clusters by License Scheduler.

In most cases **TOTAL_ALLOC** is equal to **TOTAL_USE**, however, when there are licenses counted under **OTHERS** or when tokens are reclaimed, **TOTAL_ALLOC** may be less than **TOTAL_TOKENS**.

TOTAL_USE

The number of licenses in use by License Scheduler projects, determined by totalling all **INUSE**, **RESERVE**, and **OVER** values.

OTHERS

The number of licenses checked out by applications outside of License Scheduler.

Cluster output (cluster mode)

For each cluster that is configured to use the license, `blstat` displays the following information.

CLUSTER

The cluster name.

SHARE

The percentage of licenses assigned to the license project by the License Scheduler administrator. This determines how many licenses the project is entitled to when there is competition for licenses. This information is static, and for a LAN service domain is always 100%.

The percentage is calculated to one decimal place using the share assignment in `lsf.licensescheduler`.

ALLOC

The number of licenses currently allocated to the cluster by the bld.

TARGET

The ideal amount of licenses allocated to the cluster. Normally, this amount is the same as the ALLOC field, but the values may temporarily be different. For example, when reclaiming a license, where one cluster is using more than its allocation, which prevents another cluster from getting its ideal amount.

INUSE

The number of licenses checked out by jobs in the cluster.

RESERVE

The number of licenses reserved in the service domain for jobs running in the cluster. This is determined as the difference between the job rusage and the number of checked out licenses attributed to the job by License Scheduler.

If the same license is available from both LAN and WAN service domains in cluster mode, License Scheduler expects jobs to try to obtain the license from the LAN first. It is the responsibility of the administrator to ensure that applications bahave in this manner, using the FlexNet environment variable **LM_LICENSE_FILE**.

OVER

The amount of license checkouts exceeding rusage, summed over all jobs.

PEAK

The maximum of **INUSE**+**RESERVE**+**OVER** observed over the past 5 minutes (by default), The observation period is set by **PEAK_INUSE_PERIOD** in either the **Parameters** or **Feature** section.

PEAK is used in scheduling to estimate the cluster's capacity to use licenses in this service domain.

BUFFER

The optional allocation buffer configured in the Feature section **ALLOC_BUFFER** parameter for WAN service domains. When defined, dynamic license token allocation is enabled.

FREE

The number of licenses the cluster has free. (The license tokens have been allocated to the license project by License Scheduler, but the licenses are not reserved and have not yet been checked out from the FlexNet license manager.)

DEMAND

Numeric value indicating the number of tokens required by each cluster.

Project mode summary output

FEATURE

The license name. (This appears only once for each feature.)

SERVICE_DOMAIN

The name of the service domain that provided the license.

TOTAL_INUSE

The number of licenses in use by License Scheduler projects. (Licenses in use have been checked out from the FlexNet license manager.)

TOTAL_RESERVE

The number of licenses reserved for License Scheduler projects. (Licenses that are reserved and have not been checked out from the FlexNet license manager.)

TOTAL_FREE

The number of free licenses that are available to License Scheduler projects. (Licenses that are not reserved or in use.)

OTHERS

The number of licenses checked out by users who are not submitting their jobs to License Scheduler projects.

By default, in project mode these licenses are not being managed by License Scheduler policies.

To enforce license distribution policies for these license features, configure `ENABLE_DYNAMIC_RUSAGE=Y` in the Feature section for those features in `lsf.licensescheduler`. (Project mode only.)

Workload output (both modes)

LSF_USE

The total number of licenses in use by License Scheduler projects in the LSF workload.

LSF_DESERVE

The total number of licenses assigned to License Scheduler projects in the LSF workload.

LSF_FREE

The total number of free licenses available to License Scheduler projects in the LSF workload.

NON_LSF_USE

The total number of licenses in use by projects in the non-LSF workload.

NON_LSF_DESERVE

The total number of licenses assigned to projects in the non-LSF workload.

NON_LSF_FREE

The total number of free licenses available to projects in the non-LSF workload.

Project output (project mode)

For each project that is configured to use the license, `blstat` displays the following information.

PROJECT

The License Scheduler project name.

SHARE

The percentage of licenses assigned to the license project by the License Scheduler administrator. This determines how many licenses the project is entitled to when there is competition for licenses. This information is static.

The percentage is calculated to one decimal place using the share assignment in `lsf.licensescheduler`.

LIMITS

The maximum number of licenses that the hierarchical project group member project can use at any one time.

OWN

Numeric value indicating the number of tokens owned by each project.

INUSE

The number of licenses in use by the license project. (Licenses in use have been checked out from the FlexNet license manager.)

RESERVE

The number of licenses reserved for the license project. (The corresponding job has started to run, but has not yet checked out its license from the FlexNet license manager.)

FREE

The number of licenses the license project has free. (The license tokens have been allocated to the license project by License Scheduler, but the licenses are not reserved and have not yet been checked out from the FlexNet license manager.)

DEMAND

Numeric value indicating the number of tokens required by each project.

NON_SHARED

The number of non-shared licenses belonging to the license project. (The license tokens allocated to non-shared distribution are scheduled before the tokens allocated to shared distribution.)

DESCRIPTION

Description of the project.

ACUM_USE

The number of tokens accumulated by each consumer at runtime. It is the number of licenses assigned to a given consumer for a specific feature.

SCALED_ACUM

The number of tokens accumulated by each consumer at runtime divided by the SHARE value. License Scheduler uses this value to schedule the tokens for each project.

Cross cluster token output (project mode)

For each project that is configured to use the license, `blstat -c` displays the following information.

PROJECT

The License Scheduler project name.

CLUSTER
The name of a cluster using the project.

INUSE

The number of licenses in use by the license project. (Licenses in use have been checked out from the FlexNet license manager.)

RESERVE

The number of licenses reserved for the license project. (The corresponding job has started to run, but has not yet checked out its license from the FlexNet license manager.)

FREE

The number of licenses the license project has free. (The license tokens have been allocated to the license project by License Scheduler, but the licenses are not reserved and have not yet been checked out from the FlexNet license manager.)

NEED
The total number of tokens required by pending jobs (**rusage**).

ACUM_USE

The number of tokens accumulated by each consumer at runtime. It is the number of licenses assigned to a given consumer for a specific feature.

SCALED_ACUM

The number of tokens accumulated by each consumer at runtime divided by the SHARE value. License Scheduler uses this value to schedule the tokens for each project.

Cross cluster token output (fast dispatch project mode)

For each project in fast dispatch project mode that is configured to use the license, `blstat -c` displays the following information.

PROJECT

The License Scheduler project name.

CLUSTER
The name of a cluster using the project.

ALLOC

The actual number of licenses currently allocated to the cluster. It is possible that the sum of licenses in the INUSE, RESERVE, and OVER fields is larger than ALLOC. In this case, the number of tokens that the cluster occupies will eventually decrease towards the ALLOC value after the job finishes.

The percentage is calculated to one decimal place using the share assignment in `lsf.licensescheduler`.

TARGET

The ideal amount of licenses allocated to the cluster. Normally, this amount is the same as the ALLOC field, but the values may temporarily be different. For example, when reclaiming a license, where one cluster is using more than its allocation, which prevents another cluster from getting its ideal amount.

INUSE

The number of licenses in use by the cluster under the license project (Licenses in use have been checked out from the FlexNet license manager).

RESERVE

The number of licenses reserved by jobs in the cluster under the license project (The corresponding job has started to run, but has not yet checked out its license from the FlexNet license manager). The INUSE and RESERVE fields add up to the rusage of RUN jobs in the cluster.

OVER

The number of licenses checked out by RUN jobs in the cluster under the license project in excess of the rusage.

FREE

The number of licenses that the cluster under the license project has free (The license tokens have been allocated to the license project by License Scheduler, but the licenses are not reserved and have not yet been checked out from the FlexNet license manager).

DEMAND

Numeric value reported from the cluster indicating the number of tokens required by the cluster under the license project.

Project group output (project mode)

SHARE_INFO_FOR

The root member and name of the hierarchical project group. The project information displayed after this title shows the information specific to this particular project group. If this root member is itself a member of another project group, the relationship is displayed as follows:

/root_name/member_name/...

PROJECT/GROUP

The members of the hierarchical group, listed by group or project name.

-slots output

Displays the following:

- `Current job slots in use`: The total number of slots currently being used by License Scheduler jobs, including **taskman** jobs.
- `Peak job slots used`: The peak number of slots in use since the last time License Scheduler was restarted.

Viewing license feature locality

In project mode, when **LOCAL_TO** is configured for a feature in `lsf.licensescheduler`, **blstat** shows the cluster locality information for the license features.

Sample output

For example, for a cluster mode feature:

```
blstat -t f1000
FEATURE: f1000          q
 SERVICE_DOMAIN: Lan12
```

```
 TOTAL_TOKENS: 1000 TOTAL_ALLOC: 967  TOTAL_USE: 655  OTHERS: 25
  CLUSTER     SHARE   ALLOC TARGET INUSE RESERVE OVER  PEAK  BUFFER FREE  DEMAND
  clusterA     66.7 %  647  15     0     655     0     658   100    0     7452
  clusterB     33.3 %  320  15     0     0       0     0     -      320   0
 SERVICE_DOMAIN: Lan99
 TOTAL_TOKENS: 2000 TOTAL_ALLOC: 2000 TOTAL_USE: 0    OTHERS: 0
  CLUSTER     SHARE   ALLOC TARGET INUSE RESERVE OVER  PEAK  BUFFER FREE  DEMAND
  clusterA     25.0 %  500  15     0     0       0     0     100    500   0
  clusterB     25.0 %  500  15     0     0       0     0     100    500   0
  clusterC     25.0 %  500  15     0     0       0     0     -      500   0
  clusterD     25.0 %  500  15     0     0       0     0     -      500   0
```

For example, for a project mode feature with a group distribution configuration **blstat** shows the locality of the hspice feature configured for various sites:

```
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 %   3     1    0        0     11
  Lp2                    50.0 %   1     3    0        0     11
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 %   4     0    0      12       3
  Lp2                    50.0 %   5     0    0      13       1
FEATURE: hspice@siteB
 TOTAL_INUSE: 0    TOTAL_RESERVE: 0    TOTAL_FREE: 65   OTHERS: 0
  PROJECT                SHARE   OWN  INUSE RESERVE FREE   DEMAND
  Lp1                    50.0 %   4     0    0      32       2
  Lp2                    50.0 %   5     0    0      33       6
```

For example, for a project mode feature, **blstat -c** displays the following:

```
blstat -c f50
FEATURE: f50
PROJECT         CLUSTER        INUSE   RESERVE   FREE   NEED   ACUM_USE   SCALED_ACUM   AVAIL
myProj2         interactive    0       0         9      0      0.0        0.0           9
                clusterA       0       0         8      0      0.0        0.0           0
                clusterB       0       0         8      0      0.0        0.0           0
default         interactive    0       0         9      0      0.0        0.0           9
                clusterA       0       0         8      0      0.0        0.0           0
                clusterB       0       0         8      0      0.0        0.0           0
```

For example, for a fast dispatch project mode feature, **blstat -c** displays the following:

```
blstat -c f100
FEATURE: f100
PROJECT     CLUSTER         ALLOC TARGET INUSE  RESERVE OVER  FREE  DEMAND
myProj1     interactive     4     4      0      0       0     4     0
            clusterA        3     3      0      0       0     3     0
            clusterB        3     3      0      0       0     3     0
myProj2     interactive     30    30     0      0       0     30    0
            clusterA        30    30     0      0       0     30    0
            clusterB        30    30     0      0       0     30    0
```

See also

blhosts, **blinfo**

bltasks

Displays License Scheduler interactive task information

Synopsis

bltasks [**-l**] [*task_ID*]

bltasks [**-l**] [**-p** | **-r** | **-w**] [**-Lp** "*ls_project_name*..."] [**-m** "*host_name*..."] [**-t** "*terminal_name*..."] [**-u** "*user_name*..."]

bltasks [| **-h** | **-V**]

Description

Displays current information about interactive tasks managed by License Scheduler (submitted using **taskman**).

By default, displays information about all tasks.

Options

task_ID

Only displays information about the specified task.

-l

Long format. Displays detailed information for each task in a multiline format.

-p

Only displays information about tasks with PREEMPTED status.

Cannot be used with -r or -w.

-r

Only displays information about tasks with RUN status.

Cannot be used with -p or -w.

-w

Only displays information about tasks with WAIT status.

Cannot be used with -p or -r.

-Lp "*ls_project_name***..."**

Only displays information about tasks associated with the specified projects.

If project group paths are enabled (PROJECT_GROUP_PATH=Y in lsf.licensescheduler) and a task has multiple effective license projects, only displays the first task associated with the specified effective license project.

-m "*host_name***..."**

Only displays information about tasks submitted from the specified hosts.

-t "*terminal_name***..."**

Only displays information about tasks submitted from the specified terminals.

-u "*user_name***..."**

Only displays information about tasks submitted by the specified users.

-h

Prints command usage to stderr and exits.

-V

Prints License Scheduler release version to `stderr` and exits.

Default Output

Displays the short format with the following information:

TID

Task ID that License Scheduler assigned to the task.

USER

The user who submitted the task.

STAT

The current status of the task.

- RUN: Task is running.
- WAIT: Task has not yet started.
- PREEMPT: Task has been preempted and currently has no license token.

HOST

The name of host from which the task was submitted.

PROJECT

The name of the project to which the task belongs.

FEATURES

Name of the License Scheduler token.

CONNECT TIME

The submission time of the task.

EFFECTIVE_PROJECT

The actual project that the job used. If group project paths are enabled (PROJECT_GROUP_PATH=Y in the Parameters section of `lsf.licensescheduler`), License Scheduler attempts to calculate a proper project according to the configuration if the license project does not exist or is not authorized for the features. Otherwise, the submission license project is the effective license project.

Output for -l Option

Displays detailed information for each task in multi-line format. If the task is in WAIT status, **bltasks** displays "The application manager is waiting for a token to start" and the resource requirement. Otherwise, the current resource usage of task is displayed as follows:

TERMINAL

The terminal the task is using.

PGID

UNIX process group ID.

CPU

The total accumulated CPU time of all processes in a task, in seconds.

MEM

Total resident memory usage of all processes in a task, in KB.

SWAP

Total virtual memory usage of all processes in a task, in KB.

Keyboard idle since

Time at which the task became idle.

RES_REQ

The resource requirement of the task.

Command line

The command the License Scheduler task manager is executing.

blusers

Displays license usage information for License Scheduler

Synopsis

blusers [**-J** [**-u** *user_name*]] [**-t** *token_name*...] [**-l**]

blusers -P -j *job_ID* **-u** *user_name* **-m** *host_name* [**-c** *cluster_name*]

blusers [**-h** | **-V**]

Description

By default, displays summarized information about usage of licenses.

Options

-J

Displays detailed license resource request information about each job.

In cluster mode, **blusers -J** displays tokens for CLASS-C features, which are tokens that are checked out to features that a job did not explicitly request. These features have an INUSE value, but no RUSAGE value.

-u *user_name*

Displays detailed license resource request information about each job belonging to the single user specified.

-t

Displays detailed license resource request information about each job using the token names specified.

-l

Long format. Displays additional license usage information.

-P -j *job_ID* **-u** *user_name* **-m** *host_name*
-P -c *cluster_name* **-j** *job_ID* **-u** *user_name* **-m** *host_name*

This string of options is designed to be used in a customized preemption script. To identify a job, specify the LSF job ID, the user name, the name of the host where the job is running, and the cluster name.

(If the job is an interactive task submitted using `taskman`, do not specify *-c cluster_name*.)
You see the display terminal used by the job, the licenses it has checked out, and the license servers that provided the licenses. There is one line of output for each license feature from each FlexNet license server, in the format:
port_number@host_name token_name user_name host_name display

-h

Prints command usage to `stderr` and exits.

-V

Prints License Scheduler release version to `stderr` and exits.

Default Output

FEATURE

The license name. This becomes the license token name.

SERVICE_DOMAIN

The name of the service domain that provided the license.

USER

The name of the user who submitted the jobs.

HOST

The name of the host where jobs have started.

NLICS

The number of licenses checked out from FlexNet.

NTASKS

The number of running tasks using these licenses.

-J Output

Displays the following summary information for each job:

JOBID

The job ID assigned by LSF.

USER

The name of the user who submitted the job.

HOST

The name of the host where the job has been started.

PROJECT

The name of the license project that the job is associated with.

CLUSTER

The name of the LSF cluster that the job is associated with. Displays "-" for an interactive job.

START_TIME

The job start time.

Displays the following information for each license in use by the job:

RESOURCE

The name of the license requested by the job.

RUSAGE

The number of licenses requested by the job.

SERVICE_DOMAIN

The name of the service domain that provided the license.

The keyword UNKNOWN means the job requested a license from License Scheduler but has not checked out the license from FlexNet.

INUSE

The number of checked out licenses. Displays '-' when **SERVICE_DOMAIN** is UNKNOWN.

EFFECTIVE_PROJECT

The actual project that the job used. If group project paths are enabled (PROJECT_GROUP_PATH=Y in the Parameters section of `lsf.licensescheduler`), License Scheduler attempts to calculate a proper project according to the configuration if the license project does not exist or is not authorized for the feature. Otherwise, the submission license project is the effective license project.

Long Output (-l)

Displays the default output and the following additional information for each job:

OTHERS

License usage for non-managed or non-LSF workload.

DISPLAYS

Terminal display associated with the license feature.

Viewing license feature locality

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

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

Examples

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

```
blusers -J
JOBID   USER      HOST      PROJECT             CLUSTER        START_TIME
553     user1     hostA     project3            cluster1       Oct  5 15:47:14
RESOURCE        RUSAGE        SERVICE_DOMAIN   INUSE  EFFECTIVE_PROJECT
feature1        1             SD1              1      /group2/project3
feature2        1             SD1              1      /group2/others
feature3        -             SD1              1      /group2/project3
```

See also

blhosts, **blinfo**, **blstat**

---

fod.conf

The `fod.conf` file contains FOD configuration information. All sections are required.

The command **fodinfo** displays configuration information from this file.

Parameters section

Defines FOD configuration.

Structure

The first and last lines are:

```
Begin Parameters


End Parameters
```

Each subsequent line describes one configuration parameter. All parameters are required.

FOD_ADMIN

Syntax

FOD_ADMIN=*user_name*

Description

The FOD administrator. Specify a valid UNIX user account.

FOD_CLUSTERNAME

Syntax

FOD_CLUSTERNAME=*cluster_name*

Description

The FOD cluster name.

FOD_LOG_DIR

Syntax

FOD_LOG_DIR=*dir*

Description

Location of the FOD log files.

FOD_PORT

Syntax

FOD_PORT=*integer*

Description

UDP port used by FOD. Specify any port number from 512 to 65536.

FOD_WORK_DIR

Syntax

FOD_LOG_DIR=*dir*

Description

Location of the FOD working files.

Hosts section

Lists the FOD master host candidates.

Structure

The Hosts section begins and ends with the lines Begin Hosts and End Hosts. The second line is column heading, HOSTNAME. Subsequent lines list candidate master hosts, one name per line:

```
Begin Hosts
HOSTNAME
host_name1
host_name2
End Hosts
```

HOSTNAME

Specify a fully qualified host name such as hostX.mycompany.com. The first host listed is the master..

The domain name may be omitted if all the hosts are in the same DNS domain.

Applications section

The application controlled by FOD. Specify only one application.

Structure

```
Begin Applications
NAME                    Path              PARAMS        FATAL_EXIT_VALUE
application_name       dir             parameters    (integer...)
End Applications
```

NAME

The name of the application managed by FOD.

PATH

The path to the location of the application.

PARAMS

The application parameters. Specify a dash (-) to indicate that the application has no parameters.

FATAL_EXIT_VALUE

Optional. Exit values for which FOD does not automatically restart the application. Specify a space-separated list of one or more exit values, within parentheses.

fodadmin

Starts applications under FOD or shuts down FOD.

Synopsis

fodadmin shutdown [*host_name... | all*] **fodadmin** [*-h | -V*]

You must be License Scheduler administrator to use this command.

This command starts applications under FOD or shuts down FOD.

By default, shuts down FOD on the local host.

Options

shutdown [*host_name***... | all]**

Shuts down FOD on the specified hosts. This may shut down applications on the hosts that are managed by FOD. If you shut down the master host, FOD starts up on another host, if possible. Specify `all` to shut down FOD for the cluster.

-h

Prints command usage to `stderr` and exits.

-V

Prints FOD release version to `stderr` and exits.

fodapps

Displays status of applications managed by FOD.

Synopsis

fodapps [*-l | | -h | -V*]

Description

Lists all applications managed by FOD and displays information about them.

By default, displays status, PID, and host for each application.

Options

-l

Long format. Also displays path and parameters for each application.

-h

Prints command usage to `stderr` and exits.

-V

Prints FOD release version to `stderr` and exits.

Default output

NAME

Name of the application managed by FOD.

STATUS

The status of the application:

running

The application has started and is running properly.

initial

FOD has not yet attempted to start the application. This state is only seen at startup time.

exit

The application failed to start properly. FOD automatically restarts the application.

PID

The application process ID.

HOST

The name of the FOD master host. All applications managed by FOD run on the FOD master host.

-l output

PATH

The full path of the application.

PARAMETERS

The application parameters.

fodhosts

Displays the status of FOD hosts.

Synopsis

fodhosts [*-h* | *-V*]

Description

Lists all FOD hosts and displays status.

The first host listed with `ok` status is the master host..

Options

-h

Prints command usage to `stderr` and exits.

-V

Prints FOD release version to `stderr` and exits.

Output

HOST_NAME

Name of FOD host.

STATUS

Status of FOD host.

ok

FOD is running properly on the host.

unavail

Unavailable. The host may be down or FOD may not be started on the host.

fodid

Displays FOD master host and version information.

Synopsis

fodid [*-h* | *-V*]

Description

Displays name of current master host and current version of FOD. Confirms that FOD is started and running.

Options

-h

Prints command usage to `stderr` and exits.

-V

Prints FOD release version to `stderr` and exits.

taskman

checks out a license token and manages interactive UNIX applications

Synopsis

taskman -R "rusage[*token*=*number*[**:duration**=*minutes* | *hours* h] [:*token*=*number*[**:duration**=*minutes* | *hours* h][| | *token*=*number*[**:duration**=*minutes* | *hours* h] [:*token*=*number*[**:duration**=*minutes* | *hours* h]] ...] [**-Lp** *project*] [**-N** *n_retries*] [**-v**] *command*

taskman [**-h** | **-V**]

Description

Runs the interactive UNIX application on behalf of the user. When it starts, the task manager connects to License Scheduler to request the application license tokens. When all the requested licenses are available, the task manager starts the application. While the application is running, the task manager monitors resource

usage, CPU, and memory, and reports the usage to License Scheduler. When the application terminates, the task manager exits.

By default, a license is reserved for the duration of the task, so the application can check out the license at any time. Use the duration keyword if you want unused licenses to be reallocated if the application fails to check out the license before the reservation expires.

Options

command

Required. The command to start the job that requires the license.

-v

Verbose mode. Displays detailed messages about the status of configuration files.

-N *n_retries*

Specifies the maximum number of retry attempts taskman takes to connect to the daemon. If this option is not specified, taskman retries indefinitely.

-Lp *project*

Optional. Specifies the interactive license project that is requesting tokens. The client must be known to License Scheduler.

License project limits do not apply to taskman jobs even with `-Lp` specified.

-R rusage[token=*number* **[:***duration=minutes* **|** *hours* **h] [:***token=number* **[:***duration=minutes* **|** *hours* **h]][||** *token=number* **[:***duration=minutes* **|** *hours* **h] [:***token=number* **[:***duration=minutes* **|** *hours* **h]]] ...]**

Required. Specifies the type and number of license tokens to request from License Scheduler. Optionally, specifies a time limit for the license reservation, expressed as an integer (the keyword h following the number indicates hours instead of minutes). You may specify multiple license types, with different duration values. Separate each requirement with a colon (`:`) as a logical AND operator, and a double-pipe (`||`) as a logical OR operator. Enclose the entire list in one set of square brackets.

Note: If you specify alternative or compound resource requirements, **taskman** only accepts the first resource requirement string and ignores the other resource requirement strings.

For example,

Alternative resource requirement

```
taskman -R "{rusage[f2=2]}||{rusage[f2=1]}" myjob
```

Compound resource requirement

```
taskman -R "{rusage[f2=2]}+{rusage[f2=1]}" myjob
```

In both cases, **taskman** only accepts the `rusage[f2=2]` string.

-h

Prints command usage to `stderr` and exits.

-V

Prints the License Scheduler release version to `stderr` and exits.

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web

sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Intellectual Property Law
Mail Station P300
2455 South Road,
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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

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

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application

programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

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


Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

LSF®, Platform, and Platform Computing are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software

Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, See IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Printed in USA

SC27-5308-02