Using Platform MultiCluster

Platform LSF
Version 8.0
June 2011

Platform
Computing

Copyright

We’d like to hear from you

You can help us make this document better by telling us what you think of the content, organization, and usefulness of the information. If you find an error, or just want to make a suggestion for improving this document, please address your comments to doc@platform.com.

Your comments should pertain only to Platform documentation. For product support, contact support@platform.com.

Document redistribution and translation

Internal redistribution

Trademarks

Third-party license agreements

Contents

CHAPTER

1

Platform MultiCluster Overview

Benefits of Platform MultiCluster

Within an organization, sites may have separate, independently managed LSF clusters. Having multiple LSF clusters could solve problems related to:

- Ease of administration
- Different geographic locations
- Scalability

When you have more than one cluster, it is desirable to allow the clusters to cooperate to reap the following benefits of global load sharing:

- Access to a diverse collection of computing resources
- Enterprise grid computing becomes a reality
- Get better performance and computing capabilities
- Use idle machines to process jobs
- Use multiple machines to process a single parallel job
- Increase user productivity
- Add resources anywhere and make them available to the entire organization
- Plan computing resources globally based on total computing demand
- Increase computing power in an economical way

MultiCluster enables a large organization to form multiple cooperating clusters of computers so that load sharing happens not only within clusters, but also among them. MultiCluster enables:

- Load sharing across a large numbers of hosts
- Co-scheduling among clusters: Job forwarding scheduler considers remote cluster and queue availability and load before forwarding jobs.
- Resource ownership and autonomy to be enforced
- Non-shared user accounts and file systems to be supported
- Communication limitations among the clusters to be taken into consideration in job scheduling

Two Platform MultiCluster models

There are two different ways to share resources between clusters using MultiCluster. These models can be combined, for example, Cluster1 forwards jobs to Cluster2 using the job forwarding model, and Cluster2 borrows resources from Cluster3 using the resource leasing model.

Job forwarding model

In this model, the cluster that is starving for resources sends jobs over to the cluster that has resources to spare. To work together, two clusters must set up compatible send-jobs and receive-jobs queues.

With this model, scheduling of MultiCluster jobs is a process with two scheduling phases: the submission cluster selects a suitable remote receive-jobs queue, and forwards the job to it; then the execution cluster selects a suitable host and dispatches the job to it. This method automatically favors local hosts; a MultiCluster send-jobs queue always attempts to find a suitable local host before considering a receive-jobs queue in another cluster.

Resource leasing model

In this model, the cluster that is starving for resources takes resources away from the cluster that has resources to spare. To work together, the provider cluster must “export” resources to the consumer, and the consumer cluster must configure a queue to use those resources.

In this model, each cluster schedules work on a single system image, which includes both borrowed hosts and local hosts.

Choosing a model

Consider your own goals and priorities when choosing the best resource-sharing model for your site.

- The job forwarding model can make resources available to jobs from multiple clusters, this flexibility allows maximum throughput when each cluster’s resource usage fluctuates. The resource leasing model can allow one cluster exclusive control of a dedicated resource, this can be more efficient when there is a steady amount of work.
- The lease model is the most transparent to users and supports the same scheduling features as a single cluster.
- The job forwarding model has a single point of administration, while the lease model shares administration between provider and consumer clusters.

CHAPTER

2

Platform MultiCluster Setup

Setup overview

System requirements

The setup procedures will guide you through configuring your system to meet each requirement. However, you might find it helpful to understand the system requirements before you begin.

Requirements to install Platform MultiCluster

MultiCluster is a licensed product; you will have to obtain a license from Platform Computing in order to run MultiCluster.

You can use MultiCluster to link two or more LSF clusters. Then, the participating clusters can be configured to share resources.

MultiCluster files are automatically installed by LSF's regular Setup program (`lsfinstall`). Install LSF and make sure each cluster works properly as a standalone cluster before you proceed to configure MultiCluster.

Requirements for Platform MultiCluster communication between 2 clusters

- The local master host must be configured to communicate with the remote cluster:
  - The local cluster can only communicate with other clusters if they are specified in `lsf.shared`..
  - If the `RemoteClusters` section in `lsf.cluster.`*cluster_name* is defined, the local cluster has a list of recognized clusters, and is only aware of those clusters.
- The local master host must be able to contact the master host of the remote cluster:
  - The valid master host list for remote clusters is used to locate the current master host on that cluster and ensure that any remote host is a valid master host for its cluster. The valid master host list is defined in LSF_MASTER_LIST in `lsf.conf`.
  - Participating clusters must use the same port numbers for the LSF daemons RES, `mbatchd`, and `sbatchd`, and the LIM daemon.

Requirements for resource sharing between 2 clusters

- The local cluster must use the same resource definitions as the remote cluster:
  - Clusters should have common definitions of host types, host models, and resources. Each cluster finds this information in `lsf.shared`.
- A host cannot belong to more than one cluster.
- The local cluster and the remote cluster must have compatible configurations, with the resource owner sharing the resource and the resource consumer seeking to use the resource.

Requirements for jobs to run across clusters

- The user must have a valid user account in each cluster.
  - By default, LSF expects that the user accounts will have the same name in each cluster. If clusters do not share a file system and common user name space, you can configure account mapping.

- LSF must be able to transfer job files and data files between clusters.
- Dynamic IP addressing is not supported across clusters. LSF hosts require a fixed IP address to communicate with a host that belongs to another cluster.

Installation and configuration procedures

To install and configure MultiCluster, take the following steps:

1. Plan the cluster.
2. Establish communication between clusters.
3. Additional tasks that might be required to establish communication between clusters.
4. Test communication between clusters.
5. Establish resource sharing.
6. Optional tasks.

Plan the cluster (required)

1. Read the *Using Platform MultiCluster* Overview to learn about how MultiCluster can be useful to you.
2. Decide which clusters will participate. Read about setup to learn about the issues that could prevent clusters from working together.
3. Decide which resources you want to share.
4. Decide how you will share the resources among clusters. Read about the various configuration options available in the MultiCluster job forwarding model and the MultiCluster resource leasing model.
5. Read about setup to learn about configuration options common to both models.

Establish communication between clusters (required)

1. For each participating cluster, obtain and install a valid MultiCluster license.
2. For each participating cluster, add the MultiCluster product to the LSF cluster configuration file.
3. For resource sharing to work between clusters, the clusters should have common definitions of host types, host models, and resources. Configure this information in `lsf.shared`.
4. To establish communication, clusters must be aware of other clusters and know how to contact other clusters. Add each cluster name and its master host name and master host candidate names to the Cluster section of `lsf.shared`.

Additional tasks that might be required to establish communication between clusters

1. By default, LSF assumes a uniform user name space within a cluster and between clusters.
2. With MultiCluster, LSF daemons can use non-privileged ports. By default, LSF daemons in a MultiCluster environment use privileged port authentication.

Test communication between clusters (required)

1. Restart each cluster using the `lsadmin` and `badmin` commands:

   ```
   % lsadmin limrestart all
   % badmin mbdrestart
   ```

2. To verify that MultiCluster is enabled, run `lsclusters` and `bclusters`:

   ```
   % lsclusters
   ```

```
CLUSTER_NAME    STATUS    MASTER_HOST    ADMIN    HOSTS    SERVERS
cluster1         ok       hostA          admin1   1        1
cluster2         ok       hostD          admin2   3        3
% bclusters
[Remote Batch Information]
No local queue sending/receiving jobs from remote clusters
```

Establish resource sharing (required)

1. Run a simple test of resource sharing (optional).
2. Configure resource-sharing policies between clusters.

Optional tasks

1. By default, all the clusters in a MultiCluster environment are aware of all the other clusters. This makes it possible for clusters to share resources or information. You can restrict awareness of remote clusters at the cluster level.
2. With MultiCluster, LSF daemons can use non-privileged ports (by default, LSF daemons in a MultiCluster environment use privileged port authentication). You can also choose the method of daemon authentication.
3. When a local cluster requests load or host information from a remote cluster, the information is cached. If the local cluster is required to display the same information again, LSF displays the cached information, unless the cache has expired. The expiry period for cached information is configurable.
4. The default configuration of LSF is that clusters share information about the resources used by other clusters, and the information is updated every 5 minutes by the execution or provider cluster. You can disable the feature or modify how often MultiCluster resource usage is updated.
5. To learn about optional features related to each configuration model, read about the various configuration options available in the MultiCluster job forwarding model and the MultiCluster resource leasing model.

License Platform MultiCluster

You must license each cluster with a multicluster license if using MultiCluster. There is no partial licensing with MultiCluster.

1. Send the license server host IDs for each participating cluster to your LSF vendor, and Platform Computing will generate license keys for you.
2. Append the new FEATURE lines to your existing LSF `license.dat` files, so that each participating cluster is appropriately licensed. The feature line required to license MultiCluster is `lsf_multicluster`.
3. To make the change take effect, restart each LSF cluster and each license server.

Install Platform MultiCluster products

MultiCluster files are automatically installed by LSF's regular Setup program (`lsfinstall`). Install LSF and make sure each cluster works properly as a standalone cluster before you proceed to configure MultiCluster.

1. To make each cluster run MultiCluster, add LSF_MultiCluster to the products specified in the parameters section of `lsf.cluster.`*cluster_name*:

   ```
   Begin Parameters
   PRODUCTS=LSF_Base LSF_Manager LSF_MultiCluster
   ```

```
End Parameters
```

Set common ports

Participating clusters must use the same port numbers for the daemons LIM, RES, MBD, and SBD.

By default, all clusters have identical settings, as shown:

```
LSF_LIM_PORT=7869
LSF_RES_PORT=6878
LSB_MBD_PORT=6881
LSB_SBD_PORT=6882
```

LSF_LIM_PORT change

The default for LSF_LIM_PORT changed in LSF Version 7.0 to accommodate Platform EGO default port configuration. On EGO, default ports start with `lim` at 7869, and are numbered consecutively for the EGO `pem`, `vemkd`, and `egosc` daemons.

This is different from previous LSF releases where the default LSF_LIM_PORT was 6879. LSF `res`, `sbatchd`, and `mbatchd` continue to use the default pre-Version 7.0 ports 6878, 6881, and 6882.

Upgrade installation preserves existing port settings for `lim`, `res`, `sbatchd`, and `mbatchd`. EGO `pem`, `vemkd`, and `egosc` use default EGO ports starting at 7870, if they do not conflict with existing `lim`, `res`, `sbatchd`, and `mbatchd` ports.

Troubleshooting

To check your port numbers, check the `LSF_TOP/conf/lsf.conf` file in each cluster. (LSF_TOP is the LSF installation directory. On UNIX, this is defined in the `install.config` file). Make sure you have identical settings in each cluster for the following parameters:

- LSF_LIM_PORT
- LSF_RES_PORT
- LSB_MBD_PORT
- LSB_SBD_PORT

Set common resource definitions

For resource sharing to work between clusters, the clusters should have common definitions of host types, host models, and resources. Each cluster finds this information in `lsf.shared`, so the best way to configure MultiCluster is to make sure `lsf.shared` is identical for each cluster. If you do not have a shared file system, replicate `lsf.shared` across all clusters.

Define participating clusters and valid master hosts

To enable MultiCluster, define all participating clusters in the Cluster section of the `LSF_TOP/conf/lsf.shared` file.

1. For ClusterName, specify the name of each participating cluster. On UNIX, each cluster name is defined by LSF_CLUSTER_NAME in the `install.config` file.
2. For Servers, specify the master host and optionally candidate master hosts for the cluster. A cluster will not participate in MultiCluster resource sharing unless its current master host is listed here.

Example

```
Begin Cluster
ClusterName  Servers
Cluster1     (hostA hostB)
Cluster2     (hostD)
End Cluster
```

In this example, `hostA` should be the master host of `Cluster1` with `hostB` as the backup, and hostD should be the master host of `Cluster2`. If the master host fails in `Cluster1`, MultiCluster will still work because the backup master is also listed here. However, if the master host fails in `Cluster2`, MultiCluster will not recognize any other host as the master, so `Cluster2` will no longer participate in MultiCluster resource sharing.

Non-uniform name spaces

By default, LSF assumes a uniform user name space within a cluster and between clusters.

User account mapping

To support the execution of batch jobs across non-uniform user name spaces between clusters, LSF allows user account mapping.

File transfer

By default, LSF uses `lsrcp` for file transfer (`bsub -f` option),

Tip:

The lsrcp utility depends on a uniform user ID in different clusters.

Account mapping between clusters

By default, LSF assumes a uniform user name space within a cluster and between clusters. To support the execution of batch jobs across non-uniform user name spaces between clusters, LSF allows user account mapping.

For a job submitted by one user account in one cluster to run under a different user account on a host that belongs to a remote cluster, both the local and remote clusters must have the account mapping properly configured. System-level account mapping is configured by the LSF administrator, while user-level account mapping can be configured by LSF users.

System-level account mapping

You must be an LSF administrator to configure system level account mapping.

System-level account mapping is defined in the `UserMap` section of `lsb.users`. The submission cluster proposes a set of user mappings (defined using the keyword `export`) and the execution cluster accepts a set of user mappings (defined using the keyword `import`). For a user's job to run, the mapping must be both proposed and accepted.

Example

`lsb.users` on `cluster1`:

```
Begin UserMap
LOCAL    REMOTE                          DIRECTION
user1    user2@cluster2                  export
user3    (user4@cluster2 user6@cluster2)    export
End UserMap
```

`lsb.users` on `cluster2`:

```
Begin UserMap
LOCAL            REMOTE                   DIRECTION
user2            user1@cluster1           import
(user6 user8)    user3@cluster1           import
End UserMap
```

`Cluster1` configures `user1` to run jobs as `user2` in `cluster2`, and `user3` to run jobs as `user4` or `user6` in `cluster2`.

`Cluster2` configures `user1` from `cluster1` to run jobs as `user2`, and `user3` from `cluster1` to run jobs as `user6` or `user8`.

Only mappings configured in both clusters work. The common account mappings are for `user1` to run jobs as `user2`, and for `user3` to run jobs as `user6`. Therefore, these mappings work, but the mappings of `user3` to users 4 and 8 are only half-done and so do not work.

User-level account mapping

1. To set up your own account mapping, set up a `.lsfhosts` file in your home directory with Owner Read-Write permissions only.

   Do not give other users and groups permissions on this file.

   Tip:

   Account mapping can specify cluster names in place of host names.

Example #1

You have two accounts: user1 on `cluster1`, and `user2` on `cluster2`. To run jobs in either cluster, configure `.lsfhosts` as shown.

On each host in `cluster1`:

```
% cat ~user1/.lsfhosts
cluster2 user2
```

On each host in `cluster2`:

```
% cat ~user2/.lsfhosts
cluster1 user1
```

Example #2

You have the account `user1` on `cluster1`, and want to run jobs on `cluster2` under the `lsfguest` account. Configure `.lsfhosts` as shown.

On each host in `cluster1`:

```
% cat ~user1/.lsfhosts
cluster2 lsfguest send
```

On each host in `cluster2`:

```
% cat ~lsfguest/.lsfhosts
cluster1 user1 recv
```

Example #3

You have a uniform account name (`user2`) on all hosts in `cluster2`, and a uniform account name (`user1`) on all hosts in `cluster1` except `hostX`. On `hostX`, you have the account name `user99`.

To use both clusters transparently, configure `.lsfhosts` in your home directories on different hosts as shown.

On `hostX` in `cluster1`:

```
% cat ~user99/.lsfhosts
cluster1    user1
hostX       user99
cluster2    user2
```

On every other host in `cluster1`:

```
% cat ~user1/.lsfhosts
cluster2    user2
```

```
hostX       user99
```

On each host in `cluster2`:

```
% cat ~user2/.lsfhosts
cluster1    user1
hostX       user99
```

Restricted awareness of remote clusters

By default, all the clusters in a MultiCluster environment are aware of all the other clusters. This makes it possible for clusters to share resources or information when you configure MultiCluster links between them.

You can restrict awareness of remote clusters at the cluster level, by listing which of the other clusters in the MultiCluster environment are allowed to interact with the local cluster. In this case, the local cluster cannot display information about unrecognized clusters and does not participate in MultiCluster resource sharing with unrecognized clusters.

How it works

By default, the local cluster can obtain information about all other clusters specified in `lsf.shared`. The default behavior of RES is to accept requests from all the clusters in `lsf.shared`.

If the RemoteClusters section in `lsf.cluster.`*cluster_name* is defined, the local cluster has a list of recognized clusters, and is only aware of those clusters. The local cluster is not aware of the other clusters in the MultiCluster environment:

- The cluster does not forward jobs to unrecognized clusters, even if a local queue is configured to do so.
- The cluster does not borrow resources from unrecognized clusters, even if the remote cluster has exported the resources.
- The cluster does not export resources to unrecognized clusters, even if the local resource export section is configured to do to.
- The cluster does not receive jobs from unrecognized clusters, even if a local queue is configured to do so.
- The cluster cannot view information about unrecognized clusters.

However, remote clusters might still be aware of this cluster:

- Unrecognized clusters can view information about this cluster.
- Unrecognized clusters can send MultiCluster jobs to this cluster (they will be rejected, even if a local queue is configured to accept them).
- Unrecognized clusters can export resources to this cluster (this cluster will not use the resources, even if a local queue is configured to import them).

Example

This example illustrates how the RemoteClusters list works.

The MultiCluster environment consists of 4 clusters with a common `lsf.shared`:

```
CLUSTERS
cluster1
cluster2
cluster3
cluster4
```

In addition, cluster2 is configured with a RemoteClusters list in `lsf.cluster.`*cluster_name*:

```
Begin RemoteClusters
CLUSTERNAME
cluster3
cluster4
End RemoteClusters
```

Because of the RemoteClusters list, local applications in cluster2 are aware of cluster3 and cluster4, but not cluster1. For example, if you view information or configure queues using the keyword all, LSF will behave as if you specified the list of recognized clusters instead of all clusters in `lsf.shared`.

Add or modify RemoteClusters list

You must have cluster administrator privileges in the local cluster to perform this task.

1. Open `lsf.cluster.`*cluster_name* of the local cluster.
2. If it does not already exist, create the RemoteClusters section as shown:

   ```
   Begin RemoteClusters
   CLUSTERNAME
   ...
   End RemoteClusters
   ```
3. Edit the `RemoteClusters` section. Under the heading CLUSTERNAME, specify the names of the remote clusters that you want the local cluster recognize.

   These clusters must also be listed in `lsf.shared`, so the RemoteClusters list is always a subset of the clusters list in `lsf.shared`.

Security of daemon communication

LSF daemons in a MultiCluster environment use privileged port authentication by default. LSF `mbatchd` and `lim` daemons can be configured to communicate over non-privileged ports. If disabling the privileged port authentication makes you concerned about the security of daemon authentication, you can use an `eauth` program to enable any method of authentication for secure communication between clusters.

Configuring an `eauth` or setting LSF_MC_NON_PRIVILEGED_PORTS to **N** disables privileged port authentication.

Note:

Windows does not use privileged ports for authentication.

Requirements

- All clusters must be configured to use non-privileged ports for LSF daemon communication.
- If you use a firewall, it must accept incoming communication from non-privileged source ports if the destination ports are the LIM port configured LSF_LIM_PORT in `lsf.conf` and `mbatchd` port configured in LSB_MBD_PORT in `lsf.conf`.
- If you use a firewall, it must allow outgoing communication from non-privileged source ports to non-privileged destination ports.

1. To make LSF daemons use non-privileged ports, edit `lsf.conf` in every cluster as shown:

   LSF_MC_NON_PRIVILEGED_PORTS=Y
2. To make the changes take effect, restart the master LIM and MBD in every cluster, and the LIM on all master host candidates. For example, if a cluster’s master host is hostA and master host candidate is hostB, run the following commands in that cluster:

   ```
   lsadmin limrestart hostA
   lsadmin limrestart hostB
   badmin mbdrestart
   ```

Authentication between clusters

Because this is configured for individual clusters, not globally, different cluster pairs can use different systems of authentication. You use a different `eauth` program for each different authentication mechanism.

- For extra security, you can use any method of external authentication between any two clusters in the MultiCluster grid.
- If no common external authentication method has been configured, two clusters communicate with the default security, which is privileged port authentication.

eauth executables

Contact Platform Professional Services for more information about the `eauth` programs that Platform distributes to allow LSF to work with different security mechanisms. If you already have an `eauth` that works with LSF for daemon authentication within the cluster, use a copy of it.

If different clusters use different methods of authentication, set up multiple `eauth` programs.

1. Copy the corresponding `eauth` program to LSF_SERVERDIR.
2. Name the `eauth` program eauth.*method_name*.

   If you happen to use the same `eauth` program for daemon authentication within the cluster, you should have two copies, one named `eauth` (used by LSF) and one named eauth.*method_name* (used by MultiCluster).

Platform MultiCluster configuration

1. Edit the `lsf.cluster.`*cluster_name* RemoteClusters section.

   If the cluster does not already include a RemoteClusters list, you must add it now. To maintain the existing compatibility, specify all remote clusters in the list, even if the preferred method of authentication is the default method.
2. If necessary, add the AUTH column to the RemoteClusters section.
3. For each remote cluster, specify the preferred authentication method. Set AUTH to *method_name* (using the same method name that identifies the corresponding `eauth` program). For default behavior, specify a dash (-).
4. To make the changes take effect in a working cluster, run the following commands:

   ```
   lsadmin limrestart master_host
   lsadmin limrestart master_candidate_host
   badmin mbdrestart
   ```

   Repeat the steps for each cluster that will use external authentication, making sure that the configurations of paired-up clusters match.

Configuration example

In this example, Cluster1 and Cluster2 use Kerberos authentication with each other, but not with Cluster3. It does not matter how Cluster3 is configured, because without a common authentication method between clusters no communication occurs.

RECV_FROM set to Y indicates the local cluster accepts parallel jobs that originate in a remote cluster.

EQUIV set to Y changes the default behavior of LSF commands and utilities and causes them to automatically return load (lsload(1)), host (lshosts(1)), or placement (lsplace(1)) information about the remote cluster as well as the local cluster, even when you don't specify a cluster name.

Cluster1

`lsf.cluster.cluster1`:

```
Begin RemoteClusters
CLUSTERNAME  EQUIV   CACHE_INTERVAL   RECV_FROM   AUTH
cluster2       Y           60             Y        KRB
cluster3       N           30             N         -
End RemoteClusters
```

LSF_SERVERDIR in Cluster1 includes an `eauth` executable named `eauth.KRB`.

Cluster2

`lsf.cluster.cluster2`:

```
Begin RemoteClusters
CLUSTERNAME  EQUIV   CACHE_INTERVAL   RECV_FROM   AUTH
cluster1       Y           60             Y        KRB
cluster3       N           30             N         -
End RemoteClusters
```

LSF_SERVERDIR in Cluster2 includes an `eauth` executable named `eauth.KRB`.

Resource usage updates for Platform MultiCluster jobs

Upon installation, the default configuration of LSF is that clusters share information about the resources used by other clusters, and the information is updated every 5 minutes by the execution or provider cluster. You can disable the feature or modify how often MultiCluster resource usage is updated. Depending on load, updating the information very frequently can affect the performance of LSF.

Configure resource usage updates for Platform MultiCluster jobs

To change the timing of resource usage updating between clusters, set MC_RUSAGE_UPDATE_INTERVAL in `lsb.params` in the execution or provider cluster. Specify how often to update the information to the submission or consumer cluster, in seconds.

To disable LSF resource usage updating between clusters, specify zero:

```
MC_RUSAGE_UPDATE_INTERVAL=0
```

Platform MultiCluster information cache

When a local cluster LIM requests load or host information from a remote cluster using commands such as `lsload` or `lshosts`, the information is cached. If the local cluster is required to display the same information again, LSF displays the cached information, unless the cache has expired.

The expiry period for cached LIM information is configurable, so you can view more up-to-date information if you don't mind connecting to the remote cluster more often.

It is more efficient to get information from a local cluster than from a remote cluster. Caching remote cluster information locally minimizes excessive communication between clusters.

Cache thresholds

The cache threshold is the maximum time that remote cluster information can remain in the local cache.

There are two cache thresholds, one for load information, and one for host information. The threshold for host information is always double the threshold for load information.

By default, cached load information expires after 60 seconds and cached host information expires after 120 seconds.

How it works

When a local cluster requests load or host information from a remote cluster, the information is cached by the local master LIM.

When the local cluster is required to display the same information again, LSF evaluates the age of the information in the cache.

- If the information has been stored in the local cluster for longer than the specified time, LSF contacts the remote cluster again, updates the cache, and displays current information.
- If the age of the cached information is less than the threshold time, LSF displays the cached information.

Configure cache threshold

Set CACHE_INTERVAL in the `RemoteClusters` section of `lsf.cluster.`*cluster_name*, and specify the number of seconds to cache load information.

CHAPTER

3

Platform MultiCluster Job Forwarding Model

Job forwarding model overview

In this model, the cluster that is starving for resources sends jobs over to the cluster that has resources to spare. Job status, pending reason, and resource usage are returned to the submission cluster. When the job is done, the exit code returns to the submission cluster.

Tracking

bhosts

By default, `bhosts` shows information about hosts and resources that are available to the local cluster and information about jobs that are scheduled by the local cluster.

bjobs

The `bjobs` command shows all jobs associated with hosts in the cluster, including MultiCluster jobs. Jobs from remote clusters can be identified by the FROM_HOST column, which shows the remote cluster name and the submission or consumer cluster job ID in the format *host_name@remote_cluster_name*:*remote_job_ID*.

If the MultiCluster job is running under the job forwarding model, the QUEUE column shows a local queue, but if the MultiCluster job is running under the resource leasing model, the name of the remote queue is shown in the format *queue_name@remote_cluster_name*.

Use `-w` or `-l` to prevent the MultiCluster information from being truncated.

bclusters

Displays remote resource provider and consumer information, resource flow information, and connection status between the local and remote cluster.

Use -app to view available application profiles in remote clusters.

Information related to the job forwarding model is displayed under the heading `Job Forwarding Information`.

- LOCAL_QUEUE: Name of a local MultiCluster send-jobs or receive-jobs queue.
- JOB_FLOW: Indicates direction of job flow.
    - send

      The local queue is a MultiCluster send-jobs queue (SNDJOBS_TO is defined in the local queue).
    - recv

      The local queue is a MultiCluster receive-jobs queue (RCVJOBS_FROM is defined in the local queue).
- REMOTE: For send-jobs queues, shows the name of the receive-jobs queue in a remote cluster.

  For receive-jobs queues, always “-”.
- CLUSTER: For send-jobs queues, shows the name of the remote cluster containing the receive-jobs queue.

For receive-jobs queues, shows the name of the remote cluster that can send jobs to the local queue.

- STATUS: Indicates the connection status between the local queue and remote queue.
    - ok

      The two clusters can exchange information and the system is properly configured.
    - disc

      Communication between the two clusters has not been established. This could occur because there are no jobs waiting to be dispatched, or because the remote master cannot be located.
    - reject

      The remote queue rejects jobs from the send-jobs queue. The local queue and remote queue are connected and the clusters communicate, but the queue-level configuration is not correct. For example, the send-jobs queue in the submission cluster points to a receive-jobs queue that does not exist in the remote cluster.

      If the job is rejected, it returns to the submission cluster.

For example, consider the following application profile configurations:

- On the submission cluster (Cluster1) in the `lsb.applications` file:

```
Begin Application
NAME         = fluent
DESCRIPTION  = FLUENT Version 6.2
CPULIMIT     = 180/bp860-10      # 3 hours of host hostA
FILELIMIT    = 20000
DATALIMIT    = 20000          # jobs data segment limit
CORELIMIT    = 20000
PROCLIMIT    = 5              # job processor limit
PRE_EXEC     = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
POST_EXEC    = /usr/local/lsf/misc/testq_post |grep -v "Hi"
REQUEUE_EXIT_VALUES = 55 34 78
End Application
Begin Application
NAME         = catia
DESCRIPTION  = CATIA V5
CPULIMIT     = 24:0/bp860-10     # 24 hours of host hostA
FILELIMIT    = 20000
DATALIMIT    = 20000           # jobs data segment limit
CORELIMIT    = 20000
PROCLIMIT    = 5              # job processor limit
PRE_EXEC     = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
POST_EXEC    = /usr/local/lsf/misc/testq_post |grep -v "Hi"
REQUEUE_EXIT_VALUES = 55 34 78
End Application
Begin Application
NAME         = djob
DESCRIPTION  = distributed jobs
FILELIMIT    = 20000
DATALIMIT    = 2000000          # jobs data segment limit
RTASK_GONE_ACTION="KILLJOB_TASKEXIT IGNORE_TASKCRASH"
DJOB_ENV_SCRIPT   = /lsf/djobs/proj_1/djob_env
DJOB_RU_INTERVAL  = 300
DJOB_HB_INTERVAL  = 30
DJOB_COMMFAIL_ACTION="KILL_TASKS"
End Application
```

- On the execution cluster (Cluster2) in the `lsb.applications` file:

```
Begin Application
NAME         = dyna
DESCRIPTION  = ANSYS LS-DYNA
CPULIMIT     = 8:0/amd64dcore   # 8 hours of host model SunIPC
```

```
FILELIMIT    = 20000
DATALIMIT    = 20000          # jobs data segment limit
CORELIMIT    = 20000
PROCLIMIT    = 5              # job processor limit
PRE_EXEC     = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
POST_EXEC    = /usr/local/lsf/misc/testq_post |grep -v "Hi"
REQUEUE_EXIT_VALUES = 55 255 78
End Application
Begin Application
NAME         = default
DESCRIPTION  = global defaults
CORELIMIT    = 0              # No core files
STACKLIMIT   = 200000         # Give large default
RERUNNABLE   = Y              #
RES_REQ      = order[mem:ut]  # change the default ordering method
End Application
```

Verify that MultiCluster is enabled:

```
lsclusters
CLUSTER_NAME   STATUS   MASTER_HOST               ADMIN    HOSTS  SERVERS
cluster1       ok       master_c1                admin     1      1
cluster2       ok       master_c2                admin     2      2
```

View available applications on remote clusters from the submission cluster (Cluster1):

```
bclusters -app
REMOTE_CLUSTER  APP_NAME        DESCRIPTION
cluster2        dyna            ANSYS LS-DYNA
cluster2        default         global defaults
```

View available applications on remote clusters from the execution cluster (Cluster2):

```
bclusters -app
REMOTE_CLUSTER  APP_NAME        DESCRIPTION
cluster1        catia           CATIA V5
cluster1        fluent          FLUENT Version 6.2
cluster1        djob            distributed jobs
```

Job scheduling under the job forwarding model

With this model, scheduling of MultiCluster jobs is a process with two scheduling phases:

- the submission cluster selects a suitable remote receive-jobs queue, and forwards the job to it
- the execution cluster selects a suitable host and dispatches the job to it.

  If a suitable host is not found immediately, the job remains pending in the execution cluster, and is evaluated again the next scheduling cycle.

  This method automatically favors local hosts; a MultiCluster send-jobs queue always attempts to find a suitable local host before considering a receive-jobs queue in another cluster.

Phase I, local scheduling phase (all jobs)

1. The send-jobs queue receives the job submission request from a user.
2. The send-jobs queue parameters affect whether or not the job is accepted. For example, a job that requires 100 MB memory will be rejected if queue-level parameters specify a memory limit of only 50 MB.
3. If the job is accepted, it becomes pending in the send-jobs queue with a job ID assigned by the submission cluster.
4. During the next scheduling cycle, the send-jobs queue attempts to place the job on a host in the submission cluster. If a suitable host is found, the job is dispatched locally.
5. If the job cannot be placed locally (local hosts may not satisfy its resource requirements, or all the local hosts could be busy), the send-jobs queue attempts to forward the job to another cluster.

Phase II, job forwarding phase (Platform MultiCluster submission queues only)

1. The send-jobs queue has a list of remote receive-jobs queues that it can forward jobs to. If a job cannot be placed locally, the send-jobs queue evaluates each receive-jobs queue. All queues that will accept more MultiCluster jobs are candidates. To find out how many additional MultiCluster jobs a queue can accept, subtract the number of MultiCluster jobs already pending in the queue from the queue's pending MultiCluster job threshold (IMPT_JOBBKLG).
2. If information available to the submission cluster indicates that the first queue is suitable, LSF forwards the job to that queue.
    a) By default, only queue capacity is considered and the first queue evaluated is the one that has room to accept the most new MultiCluster jobs.
    b) When **MC_PLUGIN_REMOTE_RESOURCE=Y** is set, boolean resource requirements and available remote resources are considered.

Tip:

When MC_PLUGIN_REMOTE_RESOURCE is defined, only the following resource requirements (boolean only) are supported: -R "type==*type_name*", -R "same[type]", and -R "defined (resource_name)"

   c) When MC_PLUGIN_SCHEDULE_ENHANCE is defined, remote resources are considered as for **MC_PLUGIN_REMOTE_RESOURCE=Y**, and the scheduler is enhanced to consider remote queue preemptable jobs, queue priority, and queue workload, based on the settings selected.

3. If the first queue is not suitable, LSF considers the next queue.
4. If LSF cannot forward the job to any of the receive-jobs queues, the job remains pending in the send-jobs cluster and is evaluated again during the next scheduling cycle.

Phase III, remote scheduling phase (Platform MultiCluster jobs only)

1. The receive-jobs queue receives the MultiCluster job submission.
2. The receive-jobs queue parameters affect whether or not the job is accepted. For example, a job that requires 100 MB memory will be rejected if queue-level parameters specify a memory limit of only 50 MB.
3. If the job is rejected, it returns to the submission cluster.
4. If the job is accepted, it becomes pending in the receive-jobs queue with a new job ID assigned by the execution cluster.
5. During the next scheduling cycle, the receive-jobs queue attempts to place the job on a host in the execution cluster. If a suitable host is found, the job is dispatched. If a suitable host is not found, the job remains pending in the receive-jobs cluster, and is evaluated again the next scheduling cycle.
6. If the job is dispatched to the execution host but cannot start after the time interval MAX_RSHED_TIME (`lsb.params`), it returns to the submission cluster to be rescheduled. However, if the job repeatedly returns to the submission cluster because it could not be started in a remote cluster, after LSB_MC_INITFAIL_RETRY tries to start the job (`lsf.conf`), LSF suspends the job (PSUSP) in the submission cluster.

Queue scheduling parameters under job forwarding model

Forcing consistent scheduling behavior

If the queue policies of the send-jobs queue are the same as the queue policies of the receive-jobs queue, the user should see identical behavior, whether the job is scheduled locally or remotely.

Queue policies differ

The job-level (user-specified) requirements and queue-level parameters (set by the administrator) are used to schedule and run the job.

If a job runs in the submission cluster, the send-jobs queue parameters apply. If a job becomes a MultiCluster job and runs in another cluster, the receive-jobs queue parameters apply.

Since the receive-jobs queue policies replace the send-jobs queue polices, LSF users might notice that identical jobs are subject to different scheduling policies, depending on whether or not the job becomes a MultiCluster job.

Send-jobs queue parameters that affect MultiCluster jobs

- If the job requirements conflict with the send-jobs queue parameters, the job is rejected by the send-jobs queue.
- In general, queue-level parameters at the submission side don’t affect the scheduling of MultiCluster jobs once the jobs have been forwarded to the execution queue.

Receive-jobs queue parameters that affect MultiCluster jobs

In general, queue-level policies set on the execution side are the only parameters that affect MultiCluster jobs:

- If the job requirements conflict with the receive-jobs queue parameters, the job is rejected by the receive-jobs queue and returns to the submission cluster.
- Runtime queue level parameters (terminate when, job starter, load threshold, exclusive, etc): the receive-jobs queue settings are enforced, the send-jobs queue settings are ignored.
- Resource requirements: the receive-jobs queue settings are enforced, the send-jobs queue settings are ignored.
- Resource limits: the execution cluster settings are enforced, the submission cluster settings are ignored.
- Job slot limits (hjob limit, ujob limit, qjob limit): the execution cluster settings are enforced, the submission cluster settings are ignored.

Advance reservations across clusters

Users can create and use advance reservation for the MultiCluster job forwarding model. To enable this feature, you must upgrade all clusters to LSF Version 8 or later.

Advance reservation

The user from the submission cluster negotiates an advance reservation with the administrator of the execution cluster. The administrator creates the reservation in the execution cluster.

The reservation information is visible from the submission cluster. To submit a job and use the reserved resources, users specify the reservation at the time of job submission.

A job that specifies a reservation can only start on the reserved resources during the time of the reservation, even if other resources are available. Therefore, this type of job does not follow the normal scheduling process. Instead, the job is immediately forwarded to the execution cluster and is held in PEND until it can start. These jobs are not affected by the remote timeout limit (MAX_RSCHED_TIME in `lsb.queues`) since the system cannot automatically reschedule the job to any other cluster.

Missed reservations

If the execution cluster cannot accept the job because the reservation is expired or deleted, the job will be in the submission cluster in the PSUSP state.

The pending reason is:

```
Specified reservation has expired or has been deleted.
```

The job should be modified or killed by the owner.

If the execution cluster accepts the job and then the reservation expires or is deleted while job is pending, the job will be in the execution cluster detached from the reservation and scheduled as a normal job.

Broken connections

If cluster connectivity is interrupted, all remote reservations are forgotten.

During this time, submission clusters will not be able to see remote reservations; jobs submitted with remote reservation and not yet forwarded will PEND; and new jobs will not be able to use the reservation. Reservation information will not be available until cluster connectivity is re-established and the clusters have a chance to synchronize on reservation. At that time (given that reservation is still available), jobs will be forwarded, new jobs can be submitted with specified reservation, and users will be able to see the remote reservation.

Modify a reservation

After an advance reservation is made, you can use `brsvmod` to modify the reservation.

Advance reservations only can be modified with `brsvmod` in the local cluster. A modified remote reservation is visible from the submission cluster. The jobs attached to the remote reservation are treated as the local jobs when the advance reservation is modified in the remote cluster.

Delete a reservation

After an advance reservation is made, you can use `brsvdel` to delete the reservation from the execution cluster.

```
brsvdel reservation_ID
```

If you try to delete the reservation from the submission cluster, you will see an error.

Submit jobs to a reservation in a remote cluster

Submit the job and specify the remote advance reservation as shown:

```
bsub -U reservation_name@cluster_name
```

In this example, we assume the default queue is configured to forward jobs to the remote cluster.

Extend a reservation

`bmod -t` allows the job to keep running after the reservation expires.

The command `bmod` does not apply to pending jobs or jobs that are already forwarded to the remote cluster. However it can be used on the execution cluster. For that, it behaves as if it is a local job.

Special considerations under job forwarding model

Chunk jobs

Job chunking is done after a suitable host is found for the job. MultiCluster jobs can be chunked, but they are forwarded to the remote execution cluster one at a time, and chunked in the execution cluster. Therefore, the CHUNK_JOB_SIZE parameter in the submission queue is ignored by MultiCluster jobs that are forwarded to a remote cluster for execution.

If MultiCluster jobs are chunked, and one job in the chunk starts to run, both clusters display the WAIT status for the remaining jobs. However, the execution cluster sees these jobs in the PEND state, while the submission cluster sees these jobs in the RUN state. This affects scheduling calculations for fairshare and limits on both clusters.

Fairshare

If fairshare scheduling is enabled, resource usage information is a factor used in the calculation of dynamic user priority. MultiCluster jobs count towards a user's fairshare priority in the execution cluster, and do not affect fairshare calculations in the submission cluster.

There is no requirement that both clusters use fairshare or have the same fairshare policies. However, if you submit a job and specify a local user group for fairshare purposes (`bsub -G`), your job cannot run remotely unless you also belong to a user group of the same name in the execution cluster.

For more information on fairshare, see *Administering Platform LSF*.

Parallel jobs

A parallel job can be forwarded to another cluster, but the job cannot start unless the execution cluster has enough hosts and resources to run the entire job. A parallel job cannot span clusters.

Resizable jobs

Resizable jobs across MultiCluster clusters is not supported. This implies the following behavior:

- For the forwarding model, once job is forwarded to remote cluster, job is not autoresizable.
- You cannot run `bresize` commands to shrink allocations from submission clusters in either lease model or job forwarding model

Only `bresize release` is supported in the job forwarding model from the execution cluster:

- The submission cluster does log all events related to bresize release in submission cluster lsb.events file
- The submission cluster logs JOB_RESIZE events into lsb.acct file after the allocation is changed.
- Users should be able to view allocation changes from submission cluster through bjobs, bhist and bacct, busers, bqueues etc.

Job requeue

If job requeue is enabled, LSF requeues jobs that finish with exit codes that indicate job failure.

For more information on job requeue, see *Administering Platform LSF*.

User-specified job requeue

`brequeue` in the submission cluster causes the job to be requeued in the send-jobs queue.

`brequeue` in the execution cluster causes the job to be requeued in the receive-jobs queue.

Automatic job requeue

1. If job requeue (REQUEUE_EXIT_VALUES in `lsb.queues`) is enabled in the receive-jobs queue, and the job's exit code matches, the execution cluster requeues the job (it does not return to the submission cluster). Exclusive job requeue works properly.
2. If the execution cluster does not requeue the job, the job returns to the send-jobs cluster, and gets a second chance to be requeued. If job requeue is enabled in the send-jobs queue, and the job's exit code matches, the submission cluster requeues the job.
3. Exclusive job requeue values configured in the send-jobs queue always cause the job to be requeued, but for MultiCluster jobs the exclusive feature does not work; these jobs could be dispatched to the same remote execution host as before.

Automatic retry limits

The pre-execution command retry limit (MAX_PREEXEC_RETRY, LOCAL_MAX_PREEXEC_RETRY, and REMOTE_MAX_PREEXEC_RETRY), job requeue limit (MAX_JOB_REQUEUE), and job preemption retry limit (MAX_JOB_PREEMPT) configured in `lsb.params`, `lsb.queues`, and `lsb.applications` on the execution cluster are applied.

If the forwarded job requeue limit exceeds the limit on the execution cluster, the job exits and returns to the submission cluster and remains pending for rescheduling.

Job rerun

If job rerun is enabled, LSF automatically restarts running jobs that are interrupted due to failure of the execution host.

If queue-level job rerun (RERUNNABLE in `lsb.queues`) is enabled in both send-jobs and receive-jobs queues, only the receive-jobs queue reruns the job.

For more information on job rerun, see *Administering Platform LSF*.

1. If job rerun is enabled in the receive-jobs queue, the execution cluster reruns the job. While the job is pending in the execution cluster, the job status is returned to the submission cluster.
2. If the receive-jobs queue does not enable job rerun, the job returns to the submission cluster and gets a second chance to be rerun. If job rerun is enabled at the user level, or is enabled in the send-jobs queue, the submission cluster reruns the job.

Job migration

- As long as a MultiCluster job is rerunnable (`bsub -r` or `RERUNNABLE=yes` in the send-jobs queue) and is not checkpointable, you can migrate it to another host, but you cannot specify which host. Migrated jobs return to the submission cluster to be dispatched with a new job ID.

  For more information on job migration, see Administering Platform LSF.

User-specified job migration

1. To migrate a job manually, run `bmig` in either the submission or execution cluster, using the appropriate job ID.

   You cannot use `bmig -m` to specify a host.

   Tip:

   Operating in the execution cluster is more efficient than sending the `bmig` command through the submission cluster.

Automatic job migration

1. To enable automatic job migration, set the migration threshold (MIG in `lsb.queues`) in the receive-jobs queue.
2. You can also set a migration threshold at the host level on the execution host (MIG in `lsb.hosts`).

   The lowest migration threshold applies to the job.

   Tip:

   Automatic job migration configured in the send-jobs queue does not affect MultiCluster jobs.

Migration of checkpointable jobs

1. Checkpointable MultiCluster jobs cannot be migrated to another host.

   The migration action stops and checkpoints the job, then schedules the job on the same host again.

Checkpoint a Platform MultiCluster job

Checkpointing of a MultiCluster job is only supported when the send-jobs queue is configured to forward jobs to a single remote receive-jobs queue, without ever using local hosts:

- The checkpointable MultiCluster jobs resume on the same host.

  For more information on checkpointing, see Administering Platform LSF.

Configuration

Checkpointing MultiCluster jobs

1. To enable checkpointing of MultiCluster jobs, define a checkpoint directory:
   a) in both the send-jobs and receive-jobs queues (CHKPNT in `lsb.queues`)
   b) or in an application profile (CHKPNT_DIR, CHKPNT_PERIOD, CHKPNT_INITPERIOD, CHKPNT_METHOD in `lsb.applications`) of both submission cluster and execution cluster.

   LSF uses the directory specified in the execution cluster and ignores the directory specified in the submission cluster.

   LSF writes the checkpoint file in a subdirectory named with the submission cluster name and submission cluster job ID. This allows LSF to checkpoint multiple jobs to the same checkpoint directory. For example, the submission cluster is `ClusterA`, the submission job ID is 789, and the send-jobs queue enables checkpointing. The job is forwarded to `clusterB`, the execution job ID is

123, and the receive-jobs queue specifies a checkpoint directory called XYZ_dir. LSF will save the checkpoint file in:

```
XYZ_dir/clusterA/789/
```

Tip:

You cannot use `bsub -k` to make a MultiCluster job checkpointable.

Checkpoint a job

1. To checkpoint and stop a MultiCluster job, run `bmig` in the execution cluster and specify the local job ID.

Tip:

You cannot run `bmig` from the submission cluster. You cannot use `bmig -m` to specify a host.

Force a checkpointed job

1. Use `brun` to force any pending job to be dispatched immediately to a specific host, regardless of user limits and fairshare priorities. This is the only way to resume a checkpointed job on a different host. By default, these jobs attempt to restart from the last checkpoint.

Tip:

Use `brun -b` if you want to make checkpointable jobs start over from the beginning (for example, this might be necessary if the new host does not have access to the old checkpoint directory).

Example

In this example, users in a remote cluster submit work to a data center using a send-jobs queue that is configured to forward jobs to only one receive-jobs queue. You are the administrator of the data center and you need to shut down a host for maintenance. The host is busy running checkpointable MultiCluster jobs.

1. Before you perform maintenance on a host in the execution cluster, take these steps:
   a) Run `badmin hclose` to close the host and prevent additional jobs from starting on the host.
   b) Run `bmig` and specify the execution cluster job IDs of the checkpointable MultiCluster jobs running on the host. For example, if jobs from a remote cluster use job IDs 123 and 456 in the local cluster, type the following command to checkpoint and stop the jobs:

      ```
      bmig 123 456
      ```

      You cannot use `bmig -m` to specify a host.
   c) Allow the checkpoint process to complete. The jobs are requeued to the submission cluster. From there, they will be forwarded to the same receive-jobs queue again, and scheduled on the same host. However, if the host is closed, they will not start.
   d) Shut down LSF daemons on the host.
2. After you perform maintenance on a host, take these steps:
   a) Start LSF daemons on the host.
   b) Use `badmin hopen` to open the host. The MultiCluster jobs resume automatically.

Absolute priority scheduling

When absolute priority scheduling (APS) is enabled in the submission queue:

- The APS value at the submission cluster:
    - The APS value will affect the job forwarding order for new incoming jobs, but not for jobs that have already been forwarded (that is, the job is still pending at the execution cluster)
    - The APS value does not affect the job order at the remote cluster. Job order is determined by the local policies at the remote cluster.
    - `bmod -aps` does not apply to the send-jobs queue
    - `bjobs -aps` shows the job order and APS value at the local cluster
- The APS value at the execution cluster:
    - The APS value at receiving queue will affect remote job execution at the execution cluster
    - The APS value at the execution cluster will not be sent back to the submission cluster

Strict resource requirement select string syntax

When LSF_STRICT_RESREQ=y is configured in `lsf.conf`, resource requirements are checked before jobs are forwarded to the remote cluster. If the selection string is valid, the job is forwarded.

When strict resource requirement checking configuration does not match between the submission and remote clusters, jobs may be rejected by the remote cluster.

Compute unit requirement strings

When a job is submitted with compute unit resource requirements, any requirements apply only to the execution cluster. Only the syntax of the resource requirement string is checked on the submission side, and if the `cu`[] string is valid, the job is forwarded.

When compute unit requirements cannot be satisfied in the remote cluster (such as a non-existent compute unit type) jobs may be rejected by the remote cluster. Hosts running LSF 7 Update 4 or earlier cannot satisfy compute unit resource requirements.

Enable Platform MultiCluster queues

By default, clusters do not share resources, even if MultiCluster has been installed. To enable job forwarding, enable MultiCluster queues in both the submission and execution clusters.

How it works

Send-jobs queue

A send-jobs queue can forward jobs to a specified remote queue. By default, LSF attempts to run jobs in the local cluster first. LSF only attempts to place a job remotely if it cannot place the job locally.

Receive-jobs queue

A receive-jobs queue accepts jobs from queues in a specified remote cluster. Although send-jobs queues only forward jobs to specific queues in the remote cluster, receive-jobs queues can accept work from any and all queues in the remote cluster.

Multiple queue pairs

- You can configure multiple send-jobs and receive-jobs queues in one cluster.
- A queue can forward jobs to as many queues in as many clusters as you want, and can also receive jobs from as many other clusters as you want.
- A receive-jobs queue can also borrow resources using the resource leasing method, but a send-jobs queue using the job forwarding method cannot also share resources using the resource leasing method.

Enable Platform MultiCluster queues

To set up a pair of MultiCluster queues, do the following:

1. In the submission cluster, configure a send-jobs queue that forwards work to the execution queue.
2. In the execution cluster, configure a receive-jobs queue that accepts work from the cluster that contains the send-jobs queue.

Send-jobs queues

1. To configure a send-jobs queue, define SNDJOBS_TO in the `lsb.queues` queue definition. Specify a space-separated list of queue names in the format *queue_name@cluster_name*.

   If the send-jobs queue has not got SNDJOBS_TO configured, it cannot forward MultiCluster jobs. The job remains pending in the submission cluster and is evaluated again during the next scheduling cycle.
2. Make sure the `lsb.queues` HOSTS parameter specifies only local hosts (or the special keyword none). If HOSTS specifies any remote hosts, SNDJOBS_TO is ignored, and the queue behaves as a receive-jobs queue under the resource leasing method.

Receive-jobs queues

1. To configure a receive-jobs queue, define RCVJOBS_FROM in the `lsb.queues` queue definition. Specify a space-separated list of cluster names.

Use the keyword `allclusters` to specify any remote cluster.

Example

```
Begin Queue
QUEUE_NAME=queue1
SNDJOBS_TO=queue2@cluster2 queue3@cluster3
RCVJOBS_FROM=cluster2 cluster3
PRIORITY=30
NICE=20
End Queue
```

This queue is both a send-jobs and receive-jobs queue, and links with multiple remote clusters. If `queue1` cannot place a job in the local cluster, it can forward the job to `queue2` in `cluster2`, or to `queue3` in `cluster3`. If any queues in clusters 2 or 3 are configured to send MultiCluster jobs to `queue1`, `queue1` accepts them.

Remote-only queues

By default, LSF tries to place jobs in the local cluster. If your local cluster is occupied, it may take a long time before your jobs can run locally. You might want to force some jobs to run on a remote cluster instead of the local cluster. Submit these jobs to a remote-only queue. A remote-only queue forwards all jobs to a remote cluster without attempting to schedule the job locally.

Configure a remote-only queue

To make a queue that only runs jobs in remote clusters, take the following steps:

1. Edit the `lsb.queues` queue definition for the send-jobs queue.
    a) Define SNDJOBS_TO. This specifies that the queue can forward jobs to specified remote execution queues.
    b) Set HOSTS to none. This specifies that the queue uses no local hosts.
2. Edit the `lsb.queues` queue definition for each receive-jobs queue.
    a) Define RCVJOBS_FROM. This specifies that the receive-jobs queue accepts jobs from the specified submission cluster.

Example

In `cluster1`:

```
Begin Queue
QUEUE_NAME = queue1
HOSTS = none
SNDJOBS_TO = queue2@cluster2
MAX_RSCHED_TIME = infinit
DESCRIPTION = A remote-only queue that sends jobs to cluster2.
End Queue
```

In `cluster2`:

```
Begin Queue
QUEUE_NAME = queue2
RCVJOBS_FROM = cluster1
DESCRIPTION = A queue that receives jobs from cluster1.
End Queue
```

`Queue1` in `cluster1` forwards all jobs to `queue2` in `cluster2`.

Disable timeout in remote-only queues

A remote-only send-jobs queue that sends to only one receive-jobs queue.

1. Set MAX_RSCHED_TIME=infinit to maintain FCFS job order of MultiCluster jobs in the execution queue.

    Otherwise, jobs that time out are rescheduled to the same execution queue, but they lose priority and position because they are treated as a new job submission.

    In general, the timeout is helpful because it allows LSF to automatically shift a pending MultiCluster job to a better queue.

Force a job to run in a remote cluster

Jobs can be forced to run in a remote cluster at submission.

1. Use `bsub -q` and specify a remote-only MultiCluster queue.

   This is not compatible with `bsub -m`; when your job is forwarded to a remote queue, you cannot specify the execution host by name.

   Example:

   `queue1` is a remote-only MultiCluster queue.

   ```
   % bsub -q queue1 myjob
   Job <101> is submitted to queue <queue1>.
   ```

   This job will be dispatched to a remote cluster.

Remote cluster equivalency

By default, if no cluster name is specified, LSF utilities such as `lsload` return information about the local cluster.

If you configure a remote cluster to be equivalent to the local cluster, LSF displays information about the remote cluster as well. For example, `lsload` with no options lists hosts in the local cluster and hosts in the equivalent remote clusters.

The following commands automatically display information about hosts in a remote cluster if equivalency is configured:

- `lshosts`
- `lsload`
- `lsplace`
- `lsrun`

Performance limitation

Expect performance in a cluster to decrease as the number of equivalent clusters increases, because you must wait while LSF retrieves information from each remote cluster in turn. Defining all clusters in a large MultiCluster system as equivalent can cause a performance bottleneck as the master LIM polls all clusters synchronously.

Transparency for users

To make resources in remote clusters as transparent as possible to the user, configure a remote cluster to be equivalent to the local cluster. The users see information about the local and equivalent clusters without having to supply a cluster name to the command.

Hosts in equivalent clusters are all identified by the keyword remoteHost instead of the actual host name. For example, `bjobs -p -l` will show `remoteHost@cluster_name` instead of *host_name*`@cluster_name`.

Simplify Platform MultiCluster administration

If you have many clusters configured to use MultiCluster, create one cluster for administrative purposes, and configure every other cluster to be equivalent to it. This allows you to view the status of all clusters at once, and makes administration of LSF easier.

Configuration

To specify equivalent clusters, set EQUIV in the `RemoteClusters` section of `lsf.cluster.`*cluster_name* to Y for the equivalent clusters.

Remote Resources

You can allow the submission forward policy to consider remote resource availability before forwarding jobs. This allows jobs to be forwarded more successfully, but may result in the submission cluster only running local jobs.

Configure remote resource availability

Submission cluster scheduler considers whether remote resources exist, and only forwards jobs to a queue with free slots or space in the MultiCluster pending job threshold (IMPT_JOBBKLG).

1. To enable a submission forward policy and consider remote resource availability, define **MC_PLUGIN_REMOTE_RESOURCE=y** in `lsf.conf`.

   When MC_PLUGIN_REMOTE_RESOURCE is defined, the following resource requirements are supported: -R "type==*type_name*", and -R "same[type]"

   Note:

   Both remote resource availability and remote queue workload are considered by the scheduler when the parameter MC_PLUGIN_SCHEDULE_ENHANCE is defined.

Remote queue workload job-forwarding scheduler

Enhanced scheduler decisions can be customized to consider characteristics of remote queues before forwarding a job. Remote queue attributes such as queue priority, number of preemptable jobs, and queue workload are sent to the submission scheduler. The decisions made by the scheduler, based on this information, depend on the setting of MC_PLUGIN_SCHEDULE_ENHANCE in `lsb.params`.

Queue workload and configuration is considered in conjunction with remote resource availability (**MC_PLUGIN_REMOTE_RESOURCE=Y** is automatically set in `lsf.conf`).

Tip:

Defining MC_PLUGIN_SCHEDULE_ENHANCE as a valid value, the submission scheduler supports the same remote resources as MC_PLUGIN_REMOTE_RESOURCE: -R "type==*type_name*", and -R "same[type]"

Remote queue counter collection

The submission cluster receives up-to-date information about each queue in remote clusters. This information is considered during job forwarding decisions.

Queue information is collected by the submission cluster when MC_PLUGIN_SCHEDULE_ENHANCE (on the submission cluster) is set to a valid value. Information is sent by each execution cluster when MC_PLUGIN_UPDATE_INTERVAL (on the execution cluster) is defined, and the submission cluster is collecting queue information.

Some jobs may be forwarded between counter update intervals. The submission scheduler increments locally stored counter information as jobs are forwarded, and reconciles incoming counter updates to account for all jobs.

The following counter information is collected for each queue:

- queue ID
- queue priority
- total slots: The total number of slots (on all hosts) jobs are dispatched to from this queue. This includes slots on hosts with the status `ok`, and with the status `closed` due to running jobs.
- available slots: The free slots, or slots (out of the total slots) which do not currently have a job running.
- running slots: The number of slots currently running jobs from the queue.
- pending slots: The number of slots required by jobs pending on the queue.
- preemptable available slots: The number of slots the queue can access through preemption.
- preemptable slots
- [preemptable queue counters](1...n)
    - preemptable queue ID
    - preemptable queue priority
    - preemptable available slots

Note:

After a MultiCluster connection is established, counters take the time set in MC_PLUGIN_UPDATE_INTERVAL to update. Scheduling decisions made before this first interval has passed do not accurately account for remote queue workload.

The parameter MC_PLUGIN_SCHEDULE_ENHANCE was introduced in LSF Version 7 Update 6. All clusters within a MultiCluster configuration must be running a version of LSF containing this parameter to enable the enhanced scheduler.

Remote queue selection

The information considered by the job-forwarding scheduler when accounting for workload and remote resources depends on the setting of MC_PLUGIN_SCHEDULE_ENHANCE in `lsb.params`. Valid settings for this parameter are:

- `RESOURCE_ONLY`

  Jobs are forwarded to the remote queue with the requested resources and the largest (available slots)-(pending slots).
- `COUNT_PREEMPTABLE`

  Jobs are forwarded as with RESOURCE_ONLY, but if no appropriate queues have free slots, the best queue is selected based on the largest (preemptable available slots)-(pending slots).
- `COUNT_PREEMPTABLE` with `HIGH_QUEUE_PRIORITY`

  Jobs are forwarded as with `COUNT_PREEMPTABLE`, but jobs are forwarded to the highest priority remote queue.
- `COUNT_PREEMPTABLE` with `PREEMPTABLE_QUEUE_PRIORITY`

  Jobs are forwarded as with `COUNT_PREEMPTABLE`, but queue selection is based on which queues can preempt lowest priority queue jobs.
- `COUNT_PREEMPTABLE` with `PENDING_WHEN_NOSLOTS`

  Jobs are forwarded as with `COUNT_PREEMPTABLE`, but if no queues have free slots even after preemption, submitted jobs pend.
- `COUNT_PREEMPTABLE` with `HIGH_QUEUE_PRIORITY` and `PREEMPTABLE_QUEUE_PRIORITY`

  If no appropriate queues have free slots, the best queue is selected based on:

  - queues that can preempt lowest priority queue jobs
  - the number of preemptable jobs
  - the pending job workload
- `COUNT_PREEMPTABLE` with `HIGH_QUEUE_PRIORITY` and `PENDING_WHEN_NOSLOTS`

  If no appropriate queues have free slots, queues with free slots after jobs are preempted are considered.

  If no queues have free slots even after preemption, submitted jobs pend.
- `COUNT_PREEMPTABLE` with `PREEMPTABLE_QUEUE_PRIORITY` and `PENDING_WHEN_NOSLOTS`

  If no appropriate queues have free slots, queues are considered based on:

  - the most free slots after preempting lowest priority queue jobs and preemptable jobs

  If no queues have free slots even after preemption, submitted jobs pend.
- `COUNT_PREEMPTABLE` with `HIGH_QUEUE_PRIORITY` and `PREEMPTABLE_QUEUE_PRIORITY` and `PENDING_WHEN_NOSLOTS`

  If no appropriate queues have free slots, queues are considered based on:

- the most free slots after preempting lowest priority queue jobs and preemptable jobs

If no queues have free slots even after preemption, submitted jobs pend.

The figure shown illustrates the scheduler decision-making process for valid settings of MC_PLUGIN_SCHEDULE_ENHANCE.

Note:

When the scheduler looks for maximum values, such as for (available slots)-(pending slots), these values can be negative so long as they are within the pending job limit for a receive-jobs queue set by IMPT_JOBBKLG in `lsb.queues`.

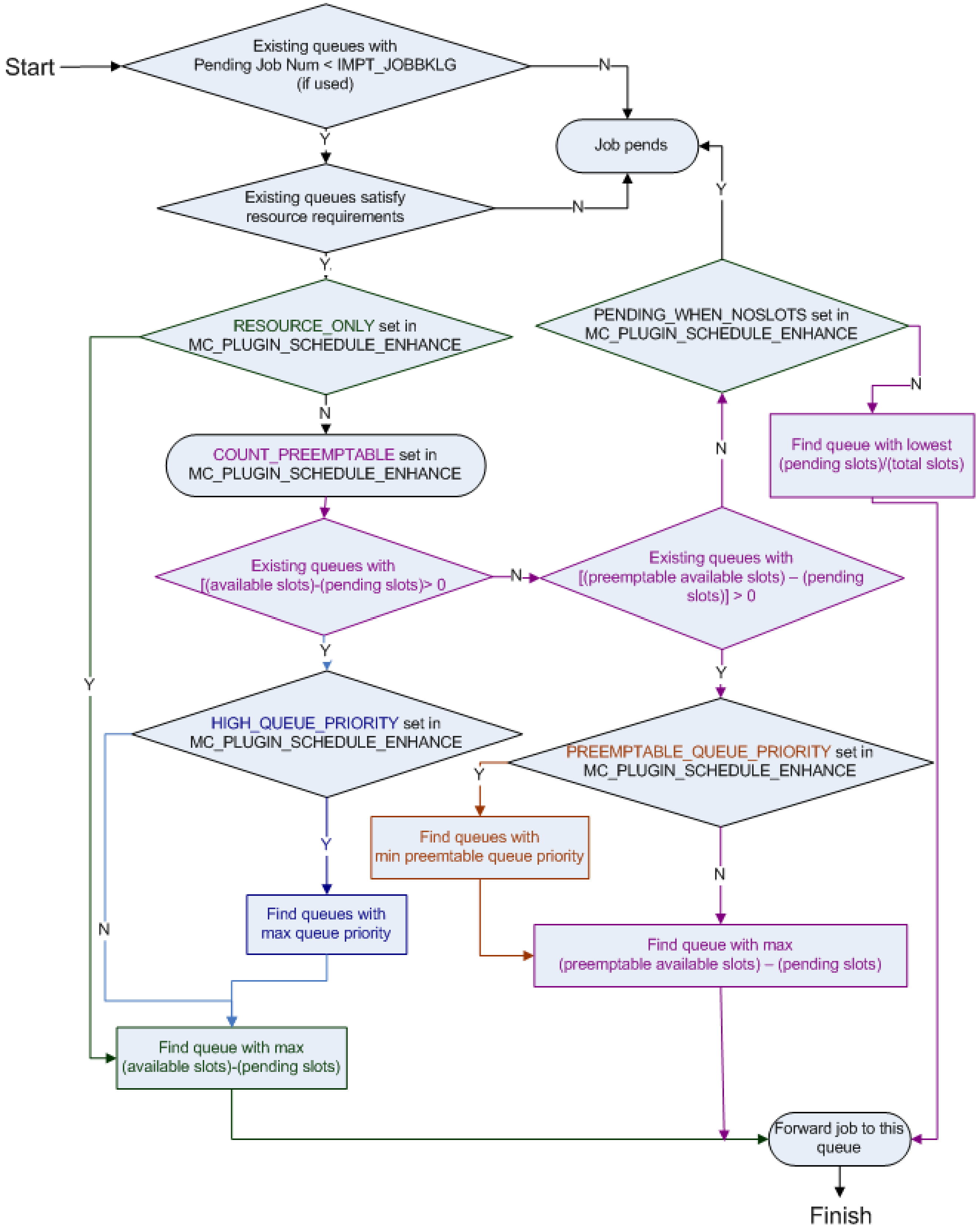
Start
Existing queues with
Pending Job Num < IMPT_JOBBKLG
(if used)
N
Y
Job pends
Existing queues satisfy
resource requirements
N
Y
Y
RESOURCE_ONLY set in
MC_PLUGIN_SCHEDULE_ENHANCE
PENDING_WHEN_NOSLOTS set in
MC_PLUGIN_SCHEDULE_ENHANCE
N
N
COUNT_PREEMPTABLE set in
MC_PLUGIN_SCHEDULE_ENHANCE
N
Find queue with lowest
(pending slots)/(total slots)
Existing queues with
[(available slots)-(pending slots)> 0
N
Existing queues with
[(preemptable available slots) – (pending
slots)] > 0
Y
Y
Y
HIGH_QUEUE_PRIORITY set in
MC_PLUGIN_SCHEDULE_ENHANCE
PREEMPTABLE_QUEUE_PRIORITY set in
MC_PLUGIN_SCHEDULE_ENHANCE
Y
Find queues with
min preemtable queue priority
Y
N
Find queues with
max queue priority
N
Find queue with max
(preemptable available slots) – (pending slots)
Find queue with max
(available slots)-(pending slots)
Forward job to this
queue
Finish

Figure 1: Scheduler decisions with MC_PLUGIN_SCHEDULE_ENHANCE set in `lsb.params`.

Limitations

Advance reservation

When an advance reservation is active on a remote cluster, slots within the advance reservation are excluded from the number of available slots. Inactive advance reservations do not affect the number of available slots since the slots may still be available for backfill jobs.

Same boolean resource within hostgroups

Hosts in a hostgroup configured without the required same boolean resources can cause ineffectual job-forwarding decisions from the scheduler.

For example, a job may be forwarded to a queue accessing a hostgroup with many slots available, only some of which have the boolean resource required. If there are not enough slots to run the job it will return to the submission cluster, which may continue forwarding the same job back to the same queue.

Same host type within hostgroups

A remote queue hostgroup satisfies host type requirements when any one of the hosts available is the host type requested by a job. As for boolean resources, the submission cluster assumes all slots within a hostgroup are of the same host type. Other hostgroup configurations can result in unexpected job-forwarding decisions.

Configure remote resource and preemptable job scheduling

Submission cluster scheduler considers whether remote resources exist, and only forwards jobs to a queue with free slots or space in the MultiCluster pending job threshold (IMPT_JOBBKLG).

If no appropriate queues with free slots or space for new pending jobs are found, the best queue is selected based on the number of preemptable jobs and the pending job workload.

1. In the submission cluster define **MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE** in `lsb.params`.
2. In the execution cluster set MC_PLUGIN_UPDATE_INTERVAL in `lsb.params` to a non-zero value.
3. To make the changes take effect in both the submission and execution clusters run the following command:

   ```
   badmin reconfig
   ```

Configure remote resource and free slot scheduling

Submission cluster scheduler considers whether remote resources exist, and only forwards jobs to a queue with free slots or space in the MultiCluster pending job threshold (IMPT_JOBBKLG).If no appropriate queues with free slots or space for new pending jobs are found, the best queue is selected based on which queues can preempt lower priority jobs.

If no queues have free slots even after preemption, jobs pend on the submission cluster.

1. In the submission cluster define **MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE PENDING_WHEN_NOSLOTS** in `lsb.params`.

2. In the execution cluster set MC_PLUGIN_UPDATE_INTERVAL in `lsb.params` to a non-zero value.
3. To make the changes take effect in both the submission and execution clusters run the following command:

```
badmin reconfig
```

Configure remote resource, preemptable job, and queue priority free slot scheduling

All scheduler options are configured.

Submission cluster scheduler considers whether remote resources exist, and only forwards jobs to a queue with free slots or space in the MultiCluster pending job threshold (IMPT_JOBBKLG).

If no appropriate queues with free slots or space for new pending jobs are found, the best queue is selected based on the number of free slots after preempting low priority jobs and preemptable jobs.

If no queues have free slots even after preemption, jobs pend on the submission cluster.

1. In the submission cluster define **MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE HIGH_QUEUE_PRIORITY PREEMPABLE_QUEUE_PRIORITY PENDING_WHEN_NOSLOTS** in `lsb.params`.
2. In the execution cluster set MC_PLUGIN_UPDATE_INTERVAL in `lsb.params` to a non-zero value.
3. To make the changes take effect in both the submission and execution clusters run the following command:

```
badmin reconfig
```

Examples

MultiCluster job forwarding is enabled from a send-queue on Cluster1 to the receive-queues HighPriority@Cluster2 and HighPriority@Cluster3. Both clusters have lower priority queues from running local jobs, and the high priority queues can preempt jobs from the lower priority queues. The scheduler on Cluster1 has the following information about the remote clusters:

Example 1: **MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE**:

Cluster2 (100 total slots)

- queue=HighPriority, priority=60, running slots=20, pending slots=20
- queue=LowPriority, priority=20, running slots=50, pending slots=0

Cluster3 (100 total slots)

- queue=HighPriority, priority=70, running slots=30, pending slots=5
- queue=LowPriority, priority=20, running slots=60, pending slots=0

Cluster2 has a total of 70 running slots out of 100 total slots, with 20 pending slots. The number of (available slots) -(pending slots) for Cluster2 is 10. Cluster3 has a total of 90 running slots out of 100 total slots, with 5 pending slots. The number of (available slots) -(pending slots) for Cluster3 is 5. Thus a job forwarded from Cluster1 is sent to HighPriority@Cluster2.

Example 2: **MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE PREEMPTABLE_QUEUE_PRIORITY**:

Cluster2 (100 total slots)

- queue=HighPriority, priority=50, running slots=20, pending slots=20
- queue=LowPriority, priority=30, running slots=80, pending slots=0

Cluster3 (100 total slots)

- queue=HighPriority, priority=50, running slots=30, pending slots=15
- queue=LowPriority, priority=20, running slots=70, pending slots=0

In both Cluster1 and Cluster2, running jobs occupy all 100 slots. LowPriority@Cluster2 has a queue priority of 30, while LowPriority@Cluster3 has a queue priority of 20. Thus a job forwarded from Cluster1 is sent to HighPriority@Cluster3, where slots can be preempted from the lowerest priority queue.

Example 3: **MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE HIGH_QUEUE_PRIORITY PREEMPTABLE_QUEUE_PRIORITY**:

Cluster2 (100 total slots)

- queue=HighPriority, priority=60, running slots=20, pending slots=20
- queue=LowPriority, priority=20, running slots=50, pending slots=0

Cluster3 (100 total slots)

- queue=HighPriority, priority=70, running slots=30, pending slots=5
- queue=LowPriority, priority=20, running slots=60, pending slots=0

Cluster2 has a total of 70 running slots out of 100 total slots, with 20 pending slots. The number of (available slots) -(pending slots) for Cluster2 is 10. Cluster3 has a total of 90 running slots out of 100 total slots, with 5 pending slots. The number of (available slots) -(pending slots) for Cluster3 is 5.

Although (available slots)-(pending slots) is higher for Cluster2, Cluster3 contains a higher priority queue. Thus a job forwarded from Cluster1 is sent to HighPriority@Cluster3.

Example 4: **MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE HIGH_QUEUE_PRIORITY PREEMPTABLE_QUEUE_PRIORITY**:

Cluster2 (100 total slots)

- queue=HighPriority, priority=60, running slots=20, pending slots=20
- queue=LowPriority, priority=20, running slots=80, pending slots=0

Cluster3 (100 total slots)

- queue=HighPriority, priority=60, running slots=30, pending slots=5
- queue=LowPriority, priority=20, running slots=70, pending slots=0

In both Cluster1 and Cluster2, running jobs occupy all 100 slots. In this case (preemptable available slots)-(pending slots) is considered. For HighPriority@Cluster2 this number is (80-20)=60; for HighPriority@Cluster3 this number is (70-5)=65. Both queues have the same priority, thus a job forwarded from Cluster1 is sent to HighPriority@Cluster3.

Pre-exec retry threshold

When a job has a pre-execution command, LSF runs the job's pre-execution command first. By default, LSF retries the pre-execution command five times.

With a threshold configured, LSF returns the entire job to the submission cluster if the pre-execution command fails to run after a certain number of attempts. The submission cluster can then reschedule the job.

Configure pre-exec retries

To limit the number of times the local cluster attempts to run the pre-execution command, set LOCAL_MAX_PREEXEC_RETRY in `lsb.params` and specify the maximum number of attempts. Configure MAX_PREEXEC_RETRY or REMOTE_MAX_PREEXEC_RETRY to limit pre-execution retry attempts on the remote cluster.

The pre-execution command retry limit configured in `lsb.params`, `lsb.queues`, and `lsb.applications` on the execution cluster is applied.

Retry threshold and suspend notification

If a job is forwarded to a remote cluster and then fails to start, it returns to the submission queue and LSF retries the job. After a certain number of failed retry attempts, LSF suspends the job (PSUSP). The job remains in that state until the job owner or administrator takes action to resume, modify, or remove the job.

By default, LSF tries to start a job up to 6 times (the threshold is 5 retry attempts). The retry threshold is configurable.

You can also configure LSF to send email to the job owner when the job is suspended. This allows the job owner to investigate the problem promptly. By default, LSF does not alert users when a job has reached its retry threshold.

Configure retries

Set LSB_MC_INITFAIL_RETRY in `lsf.conf` and specify the maximum number of retry attempts. For example, to attempt to start a job no more than 3 times in total, specify 2 retry attempts:

```
LSB_MC_INITFAIL_RETRY = 2
```

Configure mail notification

To make LSF email the user when a job is suspended after reaching the retry threshold, set LSB_MC_INITFAIL_MAIL in `lsf.conf` to y:

```
LSB_MC_INITFAIL_MAIL = y
```

By default, LSF does not notify the user.

Pending Platform MultiCluster job limit

The pending MultiCluster job limit determines the maximum number of MultiCluster jobs that can be pending in the queue. The queue rejects jobs from remote clusters when this limit is reached. It does not matter how many MultiCluster jobs are running in the queue, or how many local jobs are running or pending.

By default, the limit is 50 pending MultiCluster jobs.

Configure a pending Platform MultiCluster job limit

Edit IMPT_JOBBKLG in `lsb.queues`, and specify the maximum number of MultiCluster jobs from remote clusters that can be pending in the queue. This prevents jobs from being over-committed to an execution cluster with limited resources.

If you specify the keyword `infinit`, the queue will accept an infinite number of jobs.

Considerations

When you set the limit, consider the following:

- Make sure there are enough pending jobs in the queue for LSF to dispatch, in order to make full use of the execution servers. If you use advance reservation, set the limit higher to allow for the pending jobs that are waiting to use a reservation.
- Make sure the queue does not fill up with so many MultiCluster jobs that LSF cannot dispatch them all in the near future.

Therefore, estimate your expected job flow and set the limit 50% or 100% higher than the estimate.

Example

Assume that locally submitted jobs do not occupy all the available resources, so you estimate that each processor can schedule and execute 2 MultiCluster jobs per scheduling session. To make full use of the job slots, and make sure the queue never runs out of jobs to dispatch, set the limit at 3 or 4 jobs per processor: if this queue has 20 processors, set the limit to allow 60 or 80 MultiCluster jobs pending. You expect to run about 40 of them immediately, and the remainder only wait for one scheduling cycle.

Update pending reason for Platform MultiCluster jobs

By default, the pending reasons for MultiCluster jobs are updated every 5 minutes by the execution cluster, but the maximum amount of data transferred between clusters is 512 KB. If LSF cannot update the pending reasons for all jobs at once, it will update the additional jobs during the next cycles.

You can disable the feature or modify how often the pending reasons are updated and how much data can be transferred at one time. Depending on load, updating the information very frequently or sending an unlimited amount of information can affect the performance of LSF.

Configure the pending reason updating interval

Change the timing of pending reason updating between clusters.

1. Set MC_PENDING_REASON_UPDATE_INTERVAL in `lsb.params` in the execution cluster.
2. Specify how often to update the information in the submission cluster, in seconds.

   To disable pending reason updating between clusters, specify zero:

   ```
   MC_PENDING_REASON_UPDATE_INTERVAL=0
   ```

Configure the pending reason update package size

Change the package size of each pending reason update.

1. Set MC_PENDING_REASON_PKG_SIZE in `lsb.params` in the execution cluster.
2. Specify the maximum package size, in KB.

   To disable the limit and allow any amount of data in one package, specify zero:

   ```
   MC_PENDING_REASON_PKG_SIZE=0
   ```

   This parameter has no effect if pending reason updating is disabled (MC_PENDING_REASON_UPDATE_INTERVAL=0).

Remote timeout limit

Remote timeout limit

The remote timeout limit is set in the submission cluster and determines how long a MultiCluster job stays pending in the execution cluster. After the allowed time, the job returns to the submission cluster to be rescheduled.

The remote timeout limit in seconds is:

```
MAX_RSCHED_TIME(lsb.queues) * MBD_SLEEP_TIME(lsb.params)
```

In a default installation, MBD_SLEEP_TIME is 20 seconds and the multiplying factor for MultiCluster is 20, so the timeout limit is normally 400 seconds.

Problem with remote-only queues

By default, LSF queues dispatch jobs in FCFS order. However, there is one case in which the default behavior can be a problem. This is when a send-jobs queue sends to only one remote queue, and never uses local hosts.

In this case, jobs that time out in the receive-jobs cluster can only be re-dispatched to the same receive-jobs queue. When this happens, the receive-jobs queue takes the re-dispatched job as a new submission, gives it a new job ID, and gives it lowest priority in FCFS ordering. In this way, the highest-priority MultiCluster job times out and then becomes the lowest-priority job. Also, since local jobs don’t time out, the MultiCluster jobs get a lower priority than local jobs that have been pending for less time.

To make sure that jobs are always dispatched in the original order, you can disable remote timeout for the send-jobs queue.

Disable timeout

To disable remote timeout, edit MAX_RSCHED_TIME in `lsb.queues` in the submission cluster, and specify the keyword INFINIT. This increases the remote timeout limit to infinity.

Even if the limit is set to infinity, jobs time out if a remote execution cluster gets reconfigured. However, all the pending jobs time out at once, so when the queue attempts to send them again, the original priority is maintained.

CHAPTER

4

Platform MultiCluster Resource Leasing Model

Lease model overview

Two clusters agree that one cluster will borrow resources from the other, taking control of the resources. Both clusters must change their configuration to make this possible, and the arrangement, called a “lease”, does not expire, although it might change due to changes in the cluster configuration.

With this model, scheduling of jobs is always done by a single cluster. When a queue is configured to run jobs on borrowed hosts, LSF schedules jobs as if the borrowed hosts actually belonged to the cluster.

1. Setup:
   - A resource provider cluster “exports” hosts, and specifies the clusters that will use the resources on these hosts.
   - A resource consumer cluster configures a queue with a host list that includes the borrowed hosts.
2. To establish a lease:
   a) Configure two clusters properly (the provider cluster must export the resources, and the consumer cluster must have a queue that requests remote resources).
   b) Start up the clusters.
   c) In the consumer cluster, submit jobs to the queue that requests remote resource

   At this point, a lease is established that gives the consumer cluster control of the remote resources.

   - If the provider did not export the resources requested by the consumer, there is no lease. The provider continues to use its own resources as usual, and the consumer cannot use any resources from the provider.
   - If the consumer did not request the resources exported to it, there is no lease. However, when entire hosts are exported the provider cannot use resources that it has exported, so neither cluster can use the resources; they will be wasted.
3. Changes to the lease:
   - The lease does not expire. To modify or cancel the lease, you should change the export policy in the provider cluster.
   - If you export a group of workstations allowing LSF to automatically select the hosts for you, these hosts do not change until the lease is modified. However, if the original lease could not include the requested number of hosts, LSF can automatically update the lease to add hosts that become available later on.
   - If the configuration changes and some resources are no longer exported, jobs from the consumer cluster that have already started to run using those resources will be killed and requeued automatically.

   If LSF selects the hosts to export, and the new export policy allows some of the same hosts to be exported again, then LSF tries to re-export the hosts that already have jobs from the consumer cluster running on them (in this case, the jobs continue running without interruption). If LSF has to kill some jobs from the consumer cluster to remove some hosts from the lease, it selects the hosts according to job run time, so it kills the most recently started jobs.

Using the lease model

Submit jobs

LSF will automatically schedule jobs on the available resources, so jobs submitted to a queue that uses borrowed hosts can automatically use the borrowed resources.

bsub

To submit a job and request a particular host borrowed from another cluster, use the format *host_name@cluster_name* to specify the host. For example, to run a job on `hostA` in `cluster4`:

```
bsub -q myqueue -m hostA@cluster4 myjob
```

This will not work when you first start up the MultiCluster grid; the remote host names are not recognized until the lease has been established.

bmod

The `bmod` syntax also allows you to specify borrowed hosts in the same format *host_name@cluster_name*.

Administration

badmin

The administrator of the consumer cluster can open and close borrowed hosts using `badmin`. Use the format *host_name@cluster_name* to specify the borrowed host. This action only affects scheduling on the job slots that belong to that consumer cluster. For example, if slots on a host are shared among multiple consumers, one consumer can close the host, but the others will not be affected or be aware of any change.

You must be the administrator of the provider cluster to shut down or start up a host. This action will affect the consumer cluster as well.

Host groups or host partitions

When you define a host group in `lsb.hosts`, or a host partition, you can use the keyword allremote to indicate all borrowed hosts available to the cluster. You cannot define a host group that includes borrowed hosts specified by host name or cluster name.

Compute units

Compute units defined in `lsb.hosts` can use wild cards to include the names of borrowed hosts available to the cluster. You cannot define a host group that includes borrowed hosts specified by host name or cluster name directly.

Hosts running LSF 7 Update 4 or earlier cannot satisfy compute unit resource requirements, and thus cannot be included in compute units.

Automatic retry limits

The pre-execution command retry limit (MAX_PREEXEC_RETRY and REMOTE_MAX_PREEXEC_RETRY), job requeue limit (MAX_JOB_REQUEUE), and job preemption retry limit (MAX_JOB_PREEMPT) configured in `lsb.params`, `lsb.queues`, and `lsb.applications` apply to jobs running on remote leased hosts as if they are running on local hosts

Tracking

bhosts

By default, `bhosts` only shows information about hosts and resources that are available to the local cluster and information about jobs that are scheduled by the local cluster. Therefore, borrowed resources are included in the summary, but exported resources are not normally included (the exception is reclaimed resources, which are shown during the times that they are available to the local cluster).

For borrowed resources, the host name is displayed in the format *host_name@cluster_name*. The number of job slots shown is the number available to the consumer cluster, the JL/U and host status shown is determined by the consumer cluster, and the status shown is relative to the consumer cluster. For example, the consumer might see `closed` or `closed_Full` status, while the provider sees `ok` status.

- Cluster1 has borrowed one job slot on hostA. It shows the borrowed host is closed because that job slot is in use by a running job.

```
bhosts
HOST_NAME       STATUS  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV
hostA@cluster2  closed    -     1    1      1     0      0     0
```

- Cluster2 has kept 3 job slots on hostA for its own use. It shows the host is open, because all the available slots are free.

```
bhosts
HOST_NAME       STATUS  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV
hostA            ok      -     3    0      0     0      0     0
```

bhosts -e

This option displays information about the exported resources. The provider cluster does not display JL/U or host status; this status information is determined by the consumer cluster and does not affect the provider.

bhosts -e -s

This option displays information about exported shared resources.

bjobs

The `bjobs` command shows all jobs associated with hosts in the cluster, including MultiCluster jobs. Jobs from remote clusters can be identified by the FROM_HOST column, which shows the remote cluster name and the submission or consumer cluster job ID in the format *host_name@remote_cluster_name:remote_job_ID*.

If the MultiCluster job is running under the job forwarding model, the QUEUE column shows a local queue, but if the MultiCluster job is running under the resource leasing model, the name of the remote queue is shown in the format *queue_name@remote_cluster_name*.

Use `-w` or `-l` to prevent the MultiCluster information from being truncated.

bclusters

For the resource leasing model, `bclusters` shows information about each lease.

- Status
  - `ok` means that the resources are leased and the resources that belong to the provider are being used by the consumer.
  - `conn` indicates that a connection has been established but the lease has not yet started; probably because the consumer has not yet attempted to use the shared resources. The `conn` status remains until jobs are submitted, at which point the status changes to `ok`. If this status persists in a production environment, it could mean that the consumer cluster is not properly configured.
  - `disc` indicates that there is no connection between the two clusters.
- Resource flow
  - For resources exported to another cluster, the resource flow direction is “EXPORT”, and the remote cluster specified is the consumer of the resources.
  - For resources borrowed from another cluster, the resource flow direction is IMPORT, and the remote cluster specified is the resource provider.

Special considerations under resource leasing model

Resizable jobs

Resizable jobs across MultiCluster clusters is not supported. This implies the following behavior:

- For the lease model, the initial allocation for the job may contain lease hosts. But once the job allocation includes a leased host, LSF does not generate a pending allocation request. LSF does not allocate any leased hosts to pending allocation requests.
- You cannot run `bresize` commands to shrink allocations from submission clusters in either the lease model or job forwarding model

Checkpointing

Checkpointing is not supported if a job runs on a leased host.

Resource export

lsb.resources file

The `lsb.resources` file contains MultiCluster configuration information for the lease model, including the export policies which describe the hosts and resources that are exported, and the clusters that can use them.

You must reconfigure the cluster to make the configuration take effect.

Resources that can be exported

Job slots

To export resources, you must always export job slots on hosts, so that the consumer cluster can start jobs on the borrowed hosts.

Additional host-based resources

By default, all the jobs on a host compete for its resources. To help share resources fairly when a host's job slots are divided among multiple clusters, you can export quantities of memory and swap space, also for the use of the consumer cluster.

Shared resources

By default, shared resources such as software licenses are not exported. You can create a separate policy to export these resources.

Who can use exported resources

The export policy defines the consumers of exported resources. By default, resources that are exported can be used by the provider; this applies to job slots on a host and also to resources like memory.

With resource reclaim, exported job slots can be reclaimed by the provider if the consumer is not using them to run jobs. In this way, the provider can share in the use of the exported job slots.

Create an export policy

An export policy defined in `lsb.resources` is enclosed by the lines:

```
Begin HostExport
...
End HostExport
```

In each policy, you must specify which hosts to export, how many job slots, and distribution of resources. Optionally, you can specify quantities of memory and swap space.

Tip:

To export hosts of HostExport Type==DLINUX, specifying swap space is mandatory.

Configure as many different export policies as you need.

Each export policy corresponds to a separate lease agreement.

Export policy examples

This simple export policy exports a single job slot on a single host to a single consumer cluster:

```
Begin HostExport
PER_HOST=HostA
SLOTS=1
DISTRIBUTION=([Cluster5, 1])
End HostExport
```

This simple policy exports all the resources on a single Linux host to a single consumer cluster:

```
Begin HostExport
RES_SELECT=type==LINUX
NHOSTS=1
DISTRIBUTION=([Cluster5, 1])
End HostExport
```

Export hosts

To export resources such as job slots or other resources, you must specify which hosts the resources are located on. There are two ways to specify which hosts you want to export: you can list host names, or you can specify resource requirements and let LSF find hosts that match those resource requirements. The method you use to specify the exported hosts determines the method that LSF uses to share the hosts among competing consumer clusters.

Export a large number of hosts

If you have a group of similar hosts, you can share a portion of these hosts with other clusters. To choose this method, let LSF automatically select the hosts to export. The group of hosts can be shared among multiple consumer clusters, but each host is leased to only one consumer cluster, and all the job slots on the host are exported to the consumer.

Share a large computer

You can share a powerful multiprocessor host among multiple clusters. To choose this method, export one or more hosts by name and specify the number of job slots to export. The exported job slots on each host are divided among multiple consumer clusters.

Distribute exported resources

An export policy exports specific resources. The distribution statement in `lsb.resources` partitions these resources, assigning a certain amount exclusively to each consumer cluster. Clusters that are not named in the distribution list do not get to use any of the resources exported by the policy.

The simplest distribution policy assigns all of the exported resources to a single consumer cluster:

```
DISTRIBUTION=([Cluster5, 1])
```

Distribution list syntax

The syntax for the distribution list is a series of share assignments. Enclose each share assignment in square brackets, as shown, and use a space to separate multiple share assignments. Enclose the full list in parentheses:

```
DISTRIBUTION=([share_assignment]...)
```

Share assignment syntax

The share assignment determines what fraction of the total resources is assigned to each cluster.

The syntax of each share assignment is the cluster name, a comma, and the number of shares.

[*cluster_name*, *number_shares*]

- *cluster_name*

  Specify the name of a cluster allowed to use the exported resources.
- *number_shares*

  Specify a positive integer representing the number of shares of exported resources 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, or to the total number. The total number of shares is just the sum of all the shares assigned in each share assignment.

Examples

- In this example, resources are leased to 3 clusters in an even 1:1:1 ratio. Each cluster gets 1/3 of the resources.

  ```
  DISTRIBUTION=([C1, 1] [C2, 1] [C3, 1])
  ```
- In this example, resources are leased to 3 clusters in an uneven ratio. There are 5 shares assigned in total, so C1 gets 2/5 of the resources, C2 gets the same, and C3 gets 1/5 of the resources.

  ```
  DISTRIBUTION=([C1, 2] [C2, 2] [C3, 1])
  ```

Export workstations

These steps describe the way to share part of a large farm of identical hosts. This is most useful for reallocating resources among different departments, to meet a temporary need for more processing power.

1. Create the new policy.
2. Specify the hosts that are affected by the policy. Each host is entirely exported; the provider cluster does not save any job slots on the exported hosts for its own use.
3. Specify the distribution policy. This determines which clusters share in the use of the exported job slots.
4. Optional. Share additional resources (any combination of memory, swap space, or shared resources).

Allow Platform LSF to select the hosts you want to export

1. Specify both RES_SELECT and NHOSTS in `lsb.resources`.
2. For RES_SELECT, specify the selection criteria using the same syntax as the “select” part of the resource requirement string (normally used in the LSF `bsub` command).

   For details about resource selection syntax, see Administering Platform LSF. For this parameter, if you do not specify the required host type, the default is “`type==any`”.
3. For NHOSTS, specify a maximum number of hosts to export.

   ```
   Begin HostExport
   RES_SELECT=type==LINUX
   NHOSTS=4
   ```

   In this example, we want to export 4 Linux hosts. If the cluster has 5 Linux hosts available, 4 are exported, and the last one is not exported. If the cluster has only 3 Linux hosts available at this time, then only 3 hosts are exported, but LSF can update the lease automatically if another host becomes available to export later on.
4. Use `lshosts` to view the host types that are available in your cluster.

Distribution policy for automatically selected hosts

For syntax of the distribution policy, see “Distributing exported resources”.

When you export hosts by specifying the resource selection statement, multiple hosts are divided among multiple consumer clusters, but each host is entirely exported to a single consumer cluster. All the job slots on a host are exported to the consumer cluster, along with all its other host-based resources including swap space and memory.

Example

```
Begin HostExport
RES_SELECT=type==LINUX
NHOSTS=2
DISTRIBUTION=([C1, 1] [C2, 1])
End HostExport
```

In this example, 2 hosts that match the resource requirements are selected, suppose they are HostA and HostB, and each has 2 job slots. All job slots on each host are exported. Resources are shared evenly among 2 clusters, each cluster gets 1/2 of the resources.

Since the hosts are automatically selected, the hosts are distributed to only one consumer cluster, so the first host, HostA, goes to Cluster1, and the second host, HostB, goes to Cluster2. Assume each host has 2 job slots for use by the consumer cluster. Cluster1 gets 2 job slots on HostA, and Cluster2 gets 2 job slots on HostB.

In this example there is an even distribution policy, but it is still possible for one consumer cluster to get more resources than the other, if the exported hosts are not all identical.

Export special hosts

These steps describe the way to share a large multiprocessor host among multiple clusters. This is most useful for allowing separate departments to share the cost and use of a very powerful host. It might also be used to allow multiple clusters occasional access to a host that has some unique feature.

1. Create the new policy.
2. Specify the hosts that are affected by the policy.
3. Specify how many job slots you want to export from each host. Optionally, reduce the number of job slots available to the local cluster by the same amount.
4. Specify the distribution policy. This determines which clusters share in the use of the exported job slots.
5. Optional. Share additional resources (any combination of memory, swap space, or shared resources).

Name the hosts you want to export

- Specify the name of a host in the PER_HOST parameter in `lsb.resources`:

```
Begin HostExport
PER_HOST=HostA
```

  If you specify multiple hosts, this policy will apply to all the hosts you specify:

```
Begin HostExport
PER_HOST=HostA HostB HostC
```

Control job slots

- Use the SLOTS parameter to specify the number of job slots to export from each host.

  By default, the provider can still run the usual number of jobs at all times. The additional jobs that the consumer clusters are allowed to start might overload the host. If you are concerned with keeping the host’s performance consistent, reduce the job slot configuration in the local cluster to compensate for the number of slots exported to remote clusters.

For example, this policy exports 4 job slots on each host:

```
Begin HostExport
PER_HOST=HostA HostB
SLOTS=4
```

- Default configuration of `lsb.hosts` in the provider cluster:

```
HOST_NAME   MXJ
HostA        6
HostB        8
```

- How you can update `lsb.hosts` to compensate for the exported job slots:

```
HOST_NAME   MXJ
HostA        2
HostB        4
```

Distribution policy for named hosts

For syntax of the distribution policy, see “Distributing exported resources”.

When you export hosts by specifying host names, the job slots on each host are divided among multiple consumer clusters, so each cluster gets a part of each host.

Example

```
Begin HostExport
PER_HOST=HostA HostB
SLOTS=2
DISTRIBUTION=([C1, 1] [C2, 1])
End HostExport
```

In this example, 2 job slots are exported from HostA and HostB. Resources are shared evenly among 2 clusters, so each cluster is entitled to 1/2 of the resources.

Because the hosts are specified by name, the distribution policy is applied at the job slot level. The first job slot on HostA goes to Cluster1, and the second job slot on HostA goes to Cluster2. Similarly, one job slot on HostB goes to Cluster1, and the other job slot on HostB goes to Cluster2. Each consumer cluster can start 2 jobs, one on HostA, and one on HostB.

The provider cluster can always use the number of job slots that are configured in the provider cluster (no matter how many slots are exported). You might want to adjust the configuration of the provider cluster after exporting hosts and reduce the number of job slots (MXJ in `lsb.hosts`); otherwise, you might notice a difference in performance because of the extra jobs that can be started by the consumer clusters.

Export other resources

Once you have exported a host, you can export memory and swap space in addition to job slots.

By default, the consumer cluster borrows a job slot but is not guaranteed that there will be free memory or swap space, because all jobs on the host compete for the host's resources. If these resources are exported, each consumer cluster schedules work as if only the exported amount is available (the exported amount acts a limit for the consumer cluster), and the provider cluster can no longer use the amount that has been exported.

- The distribution policies that apply to job slots also apply to other resources.
- If the provider cluster doesn't have the amount that is specified in the export policy, it will export as much as it has.

Tip:

To export hosts of HostExport Type==DLINUX, exporting swap space is mandatory. If you do not specify swap space, the hosts of this host type are filtered because the resource is seen as unavailable

Export memory

To export memory, set MEM in `lsb.resources` host export policy, and specify the number of MB per host:

- exporting 100 MB on each host:

```
RES_SELECT=type==LINUX
NHOSTS=3
MEM=100
```

Export swap space

To export swap space, set SWP in `lsb.resources` host export policy, and specify the number of MB per host:

- exporting 100 MB on each host:

```
PER_HOST=HostA HostB HostC
SWP=100
```

Export shared resources

In addition to job slots and some other built-in resources, it is possible to export numeric shared resources (for example, representing software application licenses). The resource definitions in `lsf.shared` must be the same in both clusters.

Export policies for shared resources are defined in `lsb.resources`, after export policies for hosts. The configuration is different—shared resources are not exported per host.

When you export a shared resource to a consumer cluster, you must already have a host export policy that exports hosts to the same consumer cluster, and the shared resource must be available on one or more of those exported hosts. Otherwise, the export policy does not have any effect.

Configure shared resource export

In `lsb.resources`, configure a resource export policy for each resource as shown:

```
Begin SharedResourceExport
NAME         = AppX
NINSTANCES   = 10
DISTRIBUTION = ([C1, 30] [C2, 70])
End SharedResourceExport
```

In each policy, you specify one shared numeric resource (here, a license for ApplicationX), the maximum number of these you want to export, and distribution, using the same syntax as a host export policy.

If some quantity of the resource is available, but not the full amount you configured, LSF exports as many instances of the resource as are available to the exported hosts.

Shared lease

Optional.

You can export resources from a cluster and enable shared lease, which allows the provider cluster to share in the use of the exported resources. This type of lease dynamically balances the job slots according to the load in each cluster.

Only job slots will be shared. If you export memory, swap space, and shared resources, they become available to the consumer cluster exclusively.

About shared lease

By default, exported resources are for the exclusive use of the consumer, they cannot be used by the provider. If they are not being used by the consumer, they are wasted.

There is a way to lease job slots to a cluster part-time. With shared lease, both provider and consumer clusters can have the opportunity to take any idle job slots. The benefit of the shared lease is that the provider cluster has a chance to share in the use of its exported resources, so the average resource usage is increased.

Shared lease is not compatible with advance reservation.

If you enable shared leasing, each host can only be exported to a single consumer cluster. Therefore, when shared leasing is enabled, you can export a group of workstations to multiple consumers using RES_SELECT syntax, but you cannot share a powerful multiprocessor host among multiple consumer clusters using PER_HOST syntax unless the distribution policy specifies just one cluster.

How it works

By default, a lease is exclusive, which means a fixed amount of exported resources is always dedicated exclusively to a consumer cluster. However, if you configure leases to be shared, the job slots exported by each export policy can also become available to the provider cluster.

Reclaimable resources are job slots that are exported with shared leasing enabled. The reclaim process is managed separately for each lease, so the set of job slots exported by one resource export policy to one consumer cluster is managed as a group.

When the provider cluster is started, the job slots are allocated to the provider cluster, except for one that is reserved for the consumer cluster, to allow a lease to be made. Therefore, all but one slot is initially available to the provider cluster, and one slot could be available to the consumer. The lease is made when the consumer schedules a job to run on the single job slot that is initially available to it.

To make job slots available to a different cluster, LSF automatically modifies the lease contract. The lease will go through a temporary “inactive” phase each time. When a lease is updated, the slots controlled by the corresponding export policy are distributed as follows: the slots that are being used to run jobs remain under the control of the cluster that is using them, but the slots that are idle are all made available to just one cluster.

To determine which cluster will reclaim the idle slots each time, LSF considers the number of idle job slots in each cluster:

```
idle_slots_provider = available_slots_provider - used_slots_provider
idle_slots_consumer = available_slots_consumer - used_slots_consumer
```

The action depends on the relative quantity of idle slots in each cluster.

- If the consumer has more idle slots:

  ```
  idle_slots_consumer > idle_slots_provider
  ```

  then the provider reclaims idle slots from the consumer, and all the idle slots go to the provider cluster.
- If the provider has more idle slots:

  ```
  idle_slots_provider > idle_slots_consumer
  ```

  then the reverse happens, and all the idle slots go to the consumer cluster.
- However, if each cluster has an equal number of idle slots:

  ```
  idle_slots_consumer = idle_slots_provider
  ```

  then the lease does not get updated.

LSF evaluates the status at regular intervals, specified by MC_RECLAIM_DELAY in `lsb.params`.

The calculations are performed separately for each set of reclaimable resources, so if a provider cluster has multiple resource export policies, some leases could be reconfigured in favor of the provider while others get reconfigured in favor of the consumer.

Enable shared leasing

- Set `TYPE=shared` in the resource export policy (`lsb.resources` HostExport section).

  Remember that each resource export policy using PER_HOST syntax must specify just one cluster in the distribution policy, if the lease is shared.

  ```
  Begin HostExport
  PER_HOST=HostA
  SLOTS=4
  TYPE=shared
  DISTRIBUTION=([C1, 1])
  End HostExport
  ```

  In this example, `HostA` is exported with shared leasing enabled, so the lease can be reconfigured at regular intervals, allowing LSF to give any idle job slots to the cluster that needs them the most.

Configure reclaim interval

Optionally set the reclaim interval.

- Set MC_RECLAIM_DELAY in `lsb.params` and specify how often to reconfigure a shared lease, in minutes. The interval is the same for every lease in the cluster.

  The default interval is 10 minutes.

Borrow resources

Default queues

When you add new hosts to a single LSF cluster, you might need to update your queues to start sending work to the new hosts. This is often not necessary, because queues with the default configuration can use all hosts in the local cluster.

However, when a MultiCluster provider cluster exports resources to a consumer cluster, the default queue configuration does not allow the consumer cluster to use those resources. You must update your queue configuration to start using the borrowed resources.

Queues that use borrowed hosts

By default, LSF queues only use hosts that belong to the submission cluster. Queues can use borrowed resources when they are configured to use borrowed hosts (and the provider cluster’s export policy must be compatible).

Queues for parallel jobs

If your clusters do not have a shared file system, then parallel jobs that require a common file space could fail if they span multiple clusters. One way to prevent this is to submit these jobs to a queue that uses hosts all from one cluster (for example, configure the queue to use local hosts or borrowed hosts, but not both).

Configure a queue to use borrowed resources

To configure a queue to use borrowed resources, edit `lsb.queues` HOSTS parameter and specify the hosts you want to borrow from one or more other clusters.

- The keyword `all` does not include borrowed hosts, only hosts that belong to the consumer cluster.
- The keyword `allremote` specifies the group of borrowed hosts belonging to all provider clusters.
- The keyword `others` does not include borrowed hosts, only hosts that belong to the consumer cluster.
- The keyword `none` is not compatible with the resource leasing model.
- You can specify a borrowed host in the format *host_name@cluster_name*. Make sure you configure this correctly, LSF does not validate names of borrowed hosts when you reconfigure the cluster.
- You can specify a host group that includes borrowed resources.
- You can specify all the hosts borrowed from another cluster in the format `all@`*cluster_name*.

all and allremote

- Queues configured with the keyword `all` can use all available resources that belong to the consumer cluster. You can specify additional clusters or hosts to use selected borrowed resources also.

```
HOSTS = all all@cluster2 hostB@cluster4
```

- Queues configured with the keyword `allremote` can use all available borrowed resources, from all other clusters. You can also specify additional host names to use selected resources that belong to the consumer cluster.

```
HOSTS = hostB hostC allremote
```

- Queues configured with both keywords can use all available resources whether the hosts are borrowed or belong to the consumer cluster.

```
HOSTS = all allremote
```

Preference

You can specify preference levels for borrowed resources, as well as for local resources. If your clusters do not have a common file system, the extra overhead of file transfer between clusters can affect performance, if a job involves large files. In this case, you should give preference to local hosts.

```
HOSTS = all+1 allremote
```

Parallel jobs and the lease model

To run parallel jobs (specifying multiple processors with `bsub -n`) across clusters, you must configure the RemoteClusters list in each cluster. By default, this list is not configured. For more information on running parallel jobs, see Administering Platform LSF.

1. If you do not already have a RemoteClusters list, create the RemoteClusters list and include the names of all remote clusters (the same list as `lsf.shared`). This enables proper communication among all clusters, and enables cross-cluster parallel jobs for all clusters.
2. If you have a RemoteClusters list, and you do not want to run parallel jobs on resources from all provider clusters, configure the RECV_FROM column in `lsf.cluster.`*cluster_name*.

   Specify “N” to exclude a remote cluster (LSF will not start parallel jobs on resources that belong to the remote cluster).Specify “Y” to enable resource-sharing for parallel jobs. This is the default.

Index