CHAPTER 7

Clustering for Load Balancing and Failover

This chapter describes the benefits of clustering *Luna Server*s and explains how to configure *Luna Server*s into clusters.

The main topics in this chapter are:

Server, cluster, and client relationship

Luna allows you to group multiple *Luna Servers* on the same local area network (LAN) into a logical *cluster.*

Data flow

Figure 1 shows an example of a network that connects a single data source to multiple *Luna Servers*. Luna clusters accommodate a fanout pattern of many clients connected through fewer Web servers to even fewer Luna clusters, all fetching from and updating the same data source. Although Figure 1 shows only one cluster, client applications can access data through multiple Luna clusters.

Note... Applications that take advantage of Luna security cannot run on multiple clusters.

Each Web server hosts a Luna client servlet using an HTTP protocol. The configuration in Figure 1 also includes one Java client that is running an application using a traditional RMI protocol.

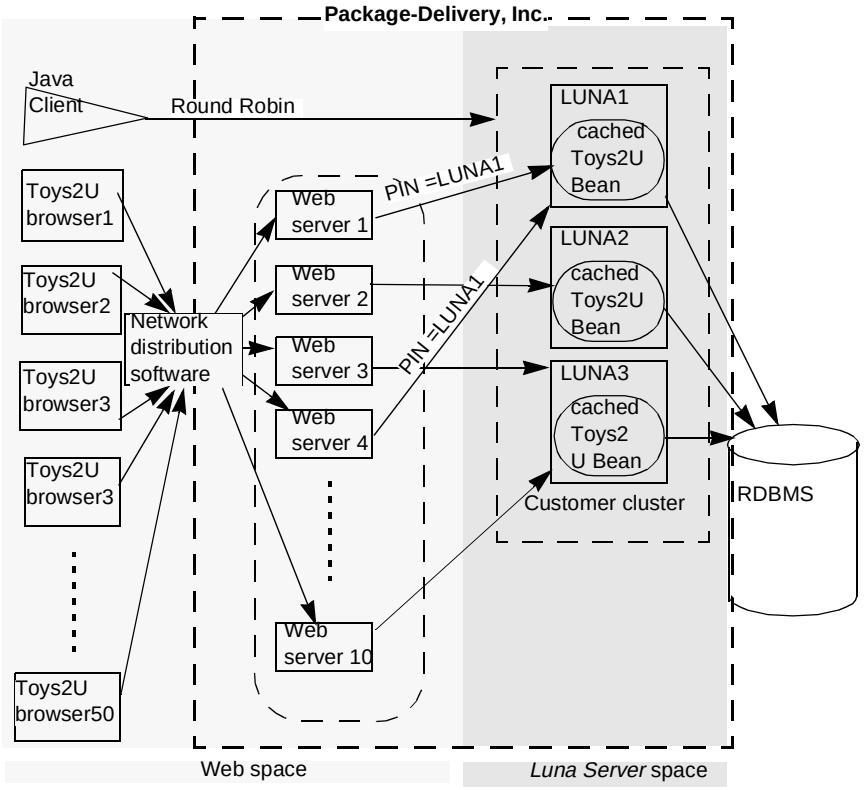

Figure 1. Sample network that includes a Luna cluster

Common characteristics

To start servers in a cluster, the administrator provides the following identical configuration properties:

- `LUNA_CLUSTER_NAME` specifies the name of the cluster that combines the individual *Luna Server*s
- `OSAGENT_PORT` specifies the port number through which clients locate server-application objects

- `com_luna_server_provider_url` specifies the location of a configuration directory on a shared LAN disk
- Contracts from a common SQL agreement data source

The administrator or developer assigns the following characteristics at the cluster level, rather than for individual servers:

- Cache-pool properties
- Data sources and data-source drivers
- Deployed JAR files and descriptors

  JAR files and descriptors are stored in the shared configuration directory that the `com_luna_server_provider_url` specifies.

The term *deployed-content symmetry* refers to this consistency in cache properties, data-source access, JAR files, and descriptors. Figure 2 shows how the *Luna Administration Center* displays the items that are cluster-level.

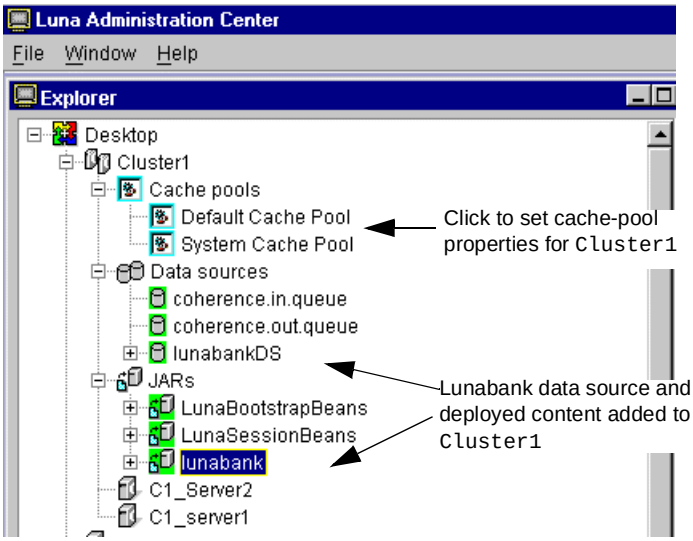

Figure 2. Cluster tree in *Administration Center Explorer*

For more information about using the *Luna Administration Center*, refer to the online help. For additional information about configuring cache, data sources, and deploying JARs, refer to the online help in the *Luna Development Center*.

Clustering Benefits

Clustering *Luna Server*s allows you to provide *failover* and *load balancing* efficiencies in the Luna system.

Failover

Every server in the cluster mirrors the same cached entity Bean state. If one *Luna Server* in the cluster experiences a problem or is put in quiescent state, an active transaction on that server can continue, uninterrupted, on another *Luna Server* in the cluster. The cluster reroutes the client that initiated the transaction

without the client knowing that it is connected to a different physical server. The cluster heals itself transparently in real time in a process called *failover.*

Cache coherency

To maintain *across-server cache coherency*, an application that commits a transaction sends a post-transacton notice to all other servers in its cluster to *invalidate*, or purge from cache, those Beans that the commit altered. The invalidation removes the entity Bean image from the cache on every other server in the cluster. Any active application in the cluster reloads the entity Bean from the database the next time it is accessed.

Isolation level

For best results, set the `Isolation level` property to Serializable. Avoid using Transaction Read Uncommitted because it forces the application to reload data if any other application in the cluster concludes a transaction.

If you use Serializable Isolation, your transaction rolls back if another transaction alters a Bean that your transaction accesses and commits while your transaction is still open. Similarly, if your application alters an entity Bean, that Bean is automatically promoted to Serializable. If another application commits a different image of that Bean anywhere in the cluster, your application rolls back.

Consider the following simple example:

1. Transaction1 starts on Server1 and reads into Server1 cache an instance of the customer Bean with the primary key accountNumber 101.

2. Transaction2 reads into Server2 cache the same customer Bean, accountNumber 101.

3. Transaction1 sets a new balance for customer 101, commits, and then notifies Server2 to purge customer 101 from cache.

4. Transaction2 attempts to get the balance for customer 101 but finds that there is no longer a committed instance in server2 cache. Transaction2 must fetch a new committed instance of customer Bean 101 into cache.

In the following example, both transactions attempt to alter the same entity Bean.

1. Transaction1 starts on Server1 and reads into Server1 cache an instance of the customer Bean with account number 101.

2. Transaction2 reads into Server2 cache the same customer Bean, account number 101.

3. Transaction1 changes customer balance for customer key 101.

4. Transaction2 changes customer name for customer key 101, which places a transaction lock on the instance in Server2 cache.

5. Transaction1 commits its instance from Server1 cache, with the altered balance, then invalidates customer Bean 101.

6. Server2 does not purge customer 101 because the transaction lock causes the instance to stay in cache.

7. Transaction2 attempts to commit and is forced to roll back.

In summary, invalidation does not purge locked instances in cache of any clustered server, but does force a roll back of any transaction that includes an invalidated instance. An instance becomes locked due to Serializable isolation or an update to the cached values.

Transaction demarcation

Failover does not pertain to the following circumstances:

- Method invocation on a stateful session Bean
- Client-demarcated transactions

To ensure failover, use stateless session Bean methods or entity Bean methods on the server to represent business logic. The transactional attribute value of a Bean or Bean method determines whether it begins its own transaction. For more information about transactional attributes, refer to the *Luna Server Development Guide*.

Load balancing

When multiple requests reach the cluster simultaneously, requests are distributed among the *Luna Servers* in that cluster. Because each *Luna Server* in a cluster services only a portion of client requests, clustering reduces the load on the cache and processor resources of each server. The ability to distribute work among servers and to prevent bottlenecks is one type of *load balancing*.

Normally, a round robin scheme distributes clients among the *Luna Servers* in a cluster. You can change the distribution scheme using affinity.

Affinity

Affinity is the ability to specify a process or a thread looks for Luna objects on a specific *Luna Server* or cluster. *Client affinity* allows you to force, or *pin*, a client application's object lookup to an *affinitied* cluster or server.

The configuration in Figure 1 on page 111 shows Web browsers application with client affinity specified. *Redirector* hardware routes all HTTP transmissions from a particular browser to a particular Web server.

Affinity takes advantage of the redirector load-balancing efficiencies. Instead of relying on a round-robin assignment to the next available *Luna Server*, the browser that connects to an affinitied Web server continues to access the same *Luna Server* unless a failover condition reroutes the Web server. If a client always connects to the same *Luna Server*, each request from that client to the server uses the data that the previous request left in cache. Further, stateful session beans remain active from one request to another.

Specifying Affinity

You can specify affinity in any of the following ways:

- Within the client code, a developer can set affinity at a process level via a system property, or per thread using the `com.luna.Runtime` class.
- For a Web application, the developer can set affinity as a Web server engine initialization parameter.
- To set up affinity as part of the JRun configuration, refer to *"Luna-specific JRun installation and setup"* in Chapter 4, "*Installing Luna Client Support*."

For more information about writing client applications, refer to the *Luna Client Development Guide*.

Configuration requirements

To setup a clustering environment, you perform the following separate actions:

- Set up the network
- Configure each server
- Deploy JAR files to the cluster

Network configuration

A network administrator must set up a shared area of LAN disk storage to hold a copy of the JAR files for the cluster. This shared area is the *configuration directory* for the cluster. Figure 3 shows an example using the Windows NT remote drive mapping facility.

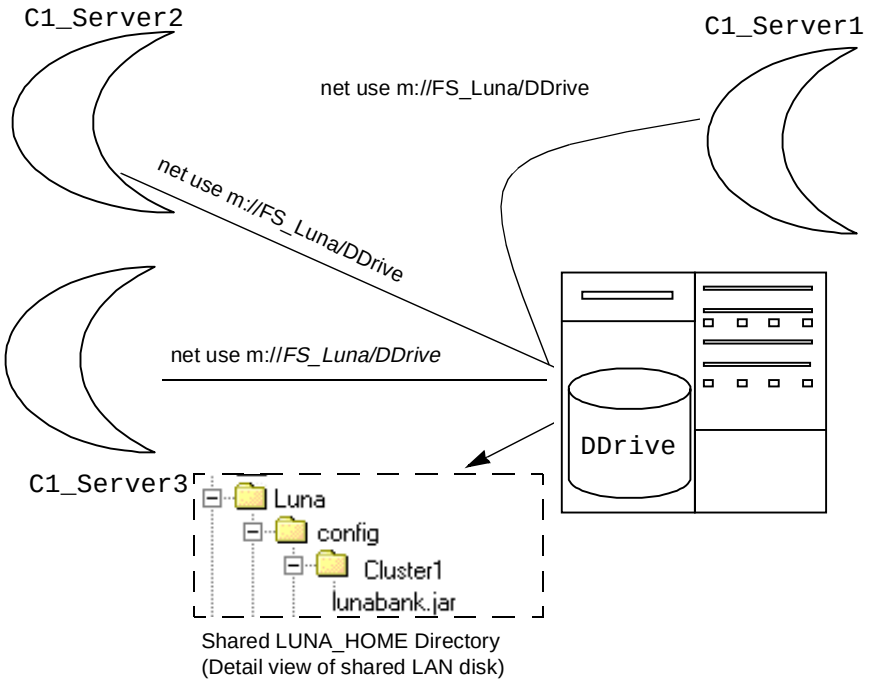

Figure 3. Shared LAN deployment repository

To set up the shared directory, a network administrator performs the following tasks:

1. Choose a machine to host the configuration directory.

   In the example for Figure 3, the administrator chose the host machine named `FS_Luna`.

2. Set up a shared volume on the host.

   To specify that other machines on the LAN can share the D drive in Figure 3, the administrator issues the Windows NT network command:

   ```
   net share D: DDrive
   ```

3. Provide a means for each server to access the shared volume.

   For the example in Figure 3, each server issues the Windows NT network command

   ```
   net use m://LS_Luna/DDrive
   ```

   Other network systems techniques for sharing disk storage include the following examples:

   ```
   LDAP:127.0.0.1:400/Luna/config/Cluster1
   nfs://export/Luna/config/Cluster1
   ```

For more information about sharing disk resources on a LAN, refer to your operating system and network administration documentation. For more information about specifying a configuration directory, refer to the next section, *"Server configuration."*

Server configuration

To set up the Luna-specific runtime environment for a cluster, set the following values for each clustered server:

- A unique `LUNA_SERVER_NAME`
- Identical `LUNA_CLUSTER_NAME`, `OSAGENT_PORT`, and `com_luna_server_provider_url` (configuration directory).

Specify the same OSAGENT_PORT number for the Smart Agent and Location Service as the OSAGENT_PORT assigned to the clustered servers.

Note... A cluster needs only one Smart Agent and Location Service running on the LAN subnet for that port number because they service the only port that the servers on the cluster need. Luna recommends that the administrator run two Smart Agents and Location Services per subnet on different machines to ensure continued service in a failover situation.

Specify the OSAGENT_PORT value for the *Luna Administration Center* and *Development Center* that matches the OSAGENT_PORT value of the clustered servers.The *Administration Center* displays any cluster and *Luna Server* that is configured or started with the common OSAGENT_PORT number. The OSAGENT_PORT number also provides access to the clustered servers from the *Development Center*.

For example, to set up the cluster shown in Figure 3 on page 118, use the following common runtime environment values for C1_Server1, C1_Server2, C1_Server3.

Note... Typically you set the OSAGENT_PORT as an environment variable, so that it pertains to the Smart Agent, Location Service, and *Administration Center* without further setup.

Configuration property	Value
LUNA_CLUSTER_NAME	Cluster1
OSAGENT_PORT	14987
com_luna_server_provider_url	m:\Luna\config\Cluster1

For information about using environment variables and configuration files to set up a cluster runtime environment, refer to the Chapter 2, "*Configuring Non-Default Installations*."

Deployment

You deploy applications for the following reasons:

- To start a new cluster
- To add a new application to an active cluster

This section describes how to deploy an application to a cluster. A developer must first build and save the JAR file or files for the application before you, the administrator, deploy the application to servers in the cluster.

Deploying an application entails two steps:

- Specify a JAR file to deploy. The JAR file is copied to the shared cluster configuration directory.
- Activate the application. Once activated, each server that you start in the cluster loads the JAR contents from the shared cluster configuration directory into memory.

To deploy and activate an application, start the **Deployment Wizard** from either the *Luna Development Center* or *Administration Center*. All but of the one active servers in a cluster are quiesced when you start the **Deployment Wizard** and they are returned to the active state when deployment to the remaining active server succeeds.

> **Note...**If deployment does not succeed, you must shutdown and restart the quiesced servers.

You can use either the *Administration Center* or the *Development Center* to deploy and activate an application. This section illustrates how to deploy and activate from the *Administration Center*. For detailed instructions on how to start the Luna components that the following steps describe, refer to Chapter 3, "*Running Luna Server.*" Start the components in the following order:

1. Start the Smart Agent and Location Services helper applications for the port that the cluster uses.

2. Start at least one server in the cluster.

3. Start the *Administration Center*.

4. Deploy the JAR file to the running server using the *Deployment Wizard* in the *Luna Administration Center*. Figure 4 shows how to start the wizard from a popup menu.

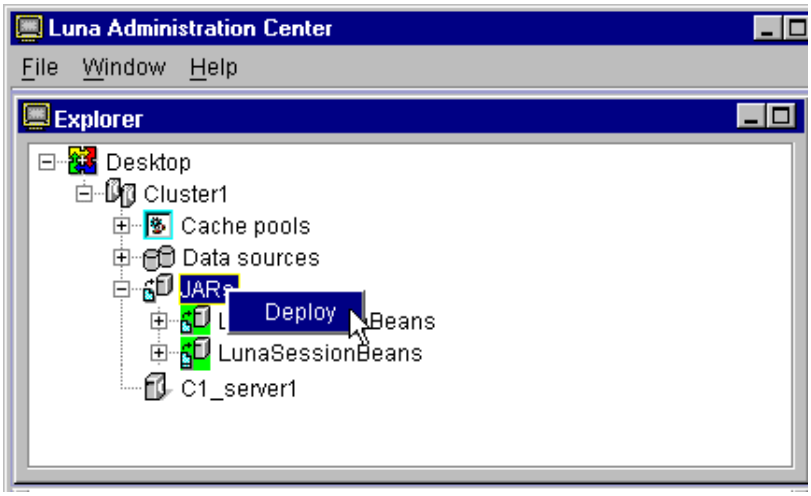

Figure 4. Starting the Deployment Wizard

   Deployment copies the application JAR files to the shared LAN directory (Figure 3 on page 118).

5. Activate the deployed JAR files.

   Activating the deployed application makes it available to the cluster's servers and clients. To activate the deployed applications, click the **Yes** button when the wizard displays a message that asks if you want to activate. (Figure 5.)

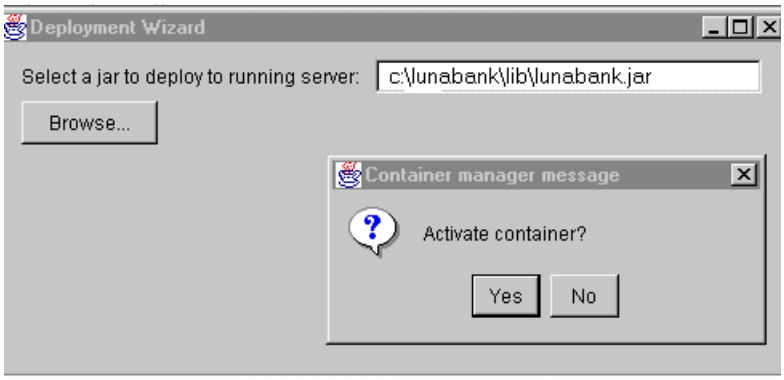

Figure 5. Activating deployed content

6. Start the other servers in the cluster, if they are not already running.

When you start other *Luna Servers* in the cluster, they automatically use the content that you copy to the shared configuration directory.

Note... Refresh the cluster information in the *Administration Center Explorer* window to display any additional servers that you start.

Warning!!! If you coldboot any server in the cluster, every active server in the cluster loses the deployed content. If you must coldboot a member of the cluster, first shut down every server in the cluster.

Important reminders

Use this section as a check list for client developers, server developers, Luna administrators, and Network administrators. The various participants must take the actions listed in this section for clustering to work.

Administrative considerations

The following list includes tasks that a *Luna Server* administrator performs from the *Luna Administration Center* or a network administrator performs using the network-administration tools.

- Use consistent spelling to set the value of `LUNA_CLUSTER_NAME` for every *Luna Server* in the cluster.

Note... Cluster names are case-sensitive.

- Set the same port for every server in the cluster and for every client that uses the cluster.
- Explicitly assign a unique `LUNA_SERVER_NAME` value to every server in the cluster.
- If the cluster is accessed by a group of Web Servers with a redirector module, set affinity for (or *pin*) each Web Server to an individual server to optimize cache persistence across Web sessions. See *"Installing JRun 2.3.3"* on page 87 for details.
- Create a shared disk directory on the LAN to hold the deployed content repository.
- Ensure that contracts are mapped to SQL, not JNDI. See *"Setting Up Luna Components"* on page 95 for more information.
- Deploy any new or changed application JAR files to the cluster, not to individual servers within the cluster.
- Deploy an application that takes advantage of Luna security to only one cluster.

Application developer tasks

The following points pertain to both client and server applications:

- Always do a JNDI lookup to locate Home and Remote interfaces.

  JNDI collects Home factories from every active *Luna Server* in the cluster on a periodic bases to refresh its cached cluster information.

- Never use the Read Uncommitted isolation level.

Client considerations

A client might interact with a clustered server application through any of the following types of connection:

- Java or Enterprise Java Beans (EJBs), using CORBA or RMI
- Web Servers and Java Server Pages (JSP) servlets using HTTP

The *Luna Installation and Configuration Guide* describes how to install Luna client components in these various environments. For instructions in writing client applications for any of these environments, refer to the *Luna Client Development Guide*.