Load Balancer Administration Guide

Contents

Chapter 1. Product overview

Load Balancer is a software solution for distributing incoming client requests across servers. It boosts the performance of servers by directing TCP/IP session requests to different servers within a group of servers; in this way, it balances the requests among all the servers. This load balancing is transparent to users and other applications. Load Balancer is useful for applications such as e-mail servers, World Wide Web servers, distributed parallel database queries, and other TCP/IP applications.

When used with Web servers, Load Balancer can help maximize the potential of your site by providing a powerful, flexible, and scalable solution to peak-demand problems. If visitors to your site can not get through at times of greatest demand, use Load Balancer to automatically find the optimal server to handle incoming requests, thus enhancing your customers' satisfaction and your profitability.

What are the advantages to using Load Balancer?

The number of users and networks connected to the global Internet is growing exponentially. This growth is causing scalability problems that can limit users' access to popular sites. Currently, network administrators are using numerous methods to try to maximize access. With some of these methods, you can choose a different server at random if an earlier choice is slow or not responding. This approach is cumbersome, annoying, and inefficient. Another method is standard round-robin, in which the domain name server selects servers in turn to handle requests. This approach is better, but still inefficient because it forwards traffic without any consideration of the server workload. In addition, even if a server fails, requests continue to be sent to it. The need for a more powerful solution has resulted in Load Balancer. It offers numerous benefits over earlier and competing solutions:

- **Scalability:** As the number of client requests increases, you can add servers dynamically, providing support for tens of millions of requests per day, on tens or even hundreds of servers.
- **Efficient use of equipment:** Load balancing ensures that each group of servers makes optimum use of its hardware by minimizing the hot-spots that frequently occur with a standard round-robin method.
- **Easy integration:** Load Balancer uses standard TCP/IP protocols. You can add it to your existing network without making any physical changes to the network. It is simple to install and configure.
- **Low overhead** Using a simple MAC level forwarding method, the Dispatcher component looks at the inbound client-to-server flows only. It does not need to see the outbound server-to-client flows. This significantly reduces its impact on the application compared with other approaches and can result in improved network performance.
- High availability: The Dispatcher component offers built-in high availability, utilizing a backup machine that remains ready at all times to take over load balancing if the primary server machine fails. When one of the servers fails, requests continue to be serviced by the other server. This process eliminates any server as a single point of failure and makes the site highly available.

- Client to server affinity: The affinity feature maps a client IP address to a back-end server, providing a higher level of efficiency by decreasing the memory and CPU utilization when compared to traditional connection forwarding.
- Load balancing a private network: You can set up Dispatcher and the TCP server machines using a private network. This configuration can reduce the contention on the public or external network that can affect performance.
- Learn how you can manage servers using Load Balancer.

  Load Balancer balances traffic among your servers through a unique combination of load balancing and management software. All client requests sent to the Dispatcher machine are directed to the ⌂best⌂ server according to weights that are set dynamically. You can use the default values for those weights or change the values during the configuration process.

  Dispatcher can also detect a failed server and forward traffic around it. Dispatcher supports HTTP, FTP, SSL, SMTP, NNTP, IMAP, POP3, Telnet, SIP, and any other TCP based application.

Load Balancer is the key to stable, efficient management of a large, scalable network of servers. You can link many individual servers into what seems to be a single, virtual server. Your site is presented as a single IP address to the world. Dispatcher functions independently of a domain name server; all requests are sent to the IP address of the Dispatcher machine.

Dispatcher brings distinct advantages in balancing traffic load to clustered servers, resulting in stable and efficient management of your site.

Related tasks

"Managing servers" on page 7
You can load balance traffic to existing server topologies without changing the physical configuration of the machines or how clients will connect to your site.

Related reference

"Types of cluster, port, and server configurations" on page 8
There are many ways that you can configure Load Balancer to support your site.

New in this release

Load Balancer for IBM WebSphere Application Server Version 7.0 contains a number of new features.

The most significant new features are:

- Load Balancer can detect changes in your network configuration without requiring you to restart the system.
- "Configuring the LDAP URI advisor" on page 57. The LDAP URI advisor allows you better gauge Lightweight Directory Access Protocol (LDAP) availability by processing a complete request to the LDAP server. The LDAP URI advisor opens a connection to the LDAP serve and sends a BIND request that is based on the advisorrequest field that you define on the server object. The advisor then waits for a response from the LDAP server and returns the elapsed time as a load.
- "Use encapsulation forwarding to forward traffic across network segments" on page 87. Use encapsulation forwarding when the back-end server is not located on the same network segment or if you are using virtualization technology and need to forward packets that are otherwise unable to be forwarded.
- There are now three options for the selection algorithm that Load Balancer uses to route traffic:

  - **connection (default):** specifies that the server selection is based on simple round-robin selection.
  - **affinity:** specifies that the server selection is based on client affinity.
  - **conn+affin:** specifies that server selection is based on an existing connection. For new connections, the server selection is based on affinity.

  Read dscontrol port for more information on this command and the available options.
- Quiesce a server on a daily schedule. You can now quiesce servers on a scheduled time to perform upgrades or general maintenance. Read more about this feature in "Quiesce servers for server maintenance windows" on page 88.
- "Support for ICMP forwarding and messaging" on page 94. Load Balancer now supports forwarding and processing ICMP messages to improve the robustness of connection protocols and permit Load Balancer to receive ICMP fragmentation messages.
- Crossport affinity allows you to expand the affinity feature across multiple ports so that client requests received on different ports can still be sent to the same server for subsequent requests. In order to use this feature, the ports must:
  - Share the same cluster address.
  - Share the same servers.
  - Use the affinity or conn+aff selection algorithm.

  Refer to the "dscontrol port" on page 154 command for more information.
- Forward UDP packets. Load Balancer includes an improved algorithm for handling connectionless UDP packets.
- Configure rules for servers or ports. You can configure rules to route connections for the following scenarios:
  - **active:** based on the number of active connections total for the port. This rule will work only if the manager is running.
  - **true:** specifies that this rule will always evaluate as true.

Functions that provide load balancing

The primary functions of Load Balancer interact with each other and your server configuration to balance network traffic in your environment.

Dispatcher consists of the following functions:

- **dsserver** handles requests from the command line to the executor, manager, and advisors.
- The **executor** supports port-based load balancing of TCP connections. It is able to forward connections to servers based on the type of request received (for example, HTTP, FTP, SSL, and so forth). The executor always runs when the Dispatcher component is being used for load balancing.
- The **manager** sets weights used by the executor based on:
  - Internal counters in the executor
  - Feedback from the servers provided by the advisors
  - Feedback from a system-monitoring program, such as Metric Server or WLM. Using the manager is optional. However, if the manager is not used, load balancing is performed using weighted round-robin scheduling based on the current server weights, and advisors are not available.
- The **advisors** query the servers and analyze results by protocol before calling the manager to set weights as appropriate. Currently there are advisors available for

the following protocols: HTTP, FTP, SSL, SMTP, NNTP, IMAP, POP3, SIP, and Telnet. Dispatcher also offers advisors that do not exchange protocol-specific information, such as the DB2 advisor that reports on the health of DB2 servers and the ping advisor that reports whether the server responds to a ping. For a complete list of advisors, see "List of advisors" on page 53. You also have the option of writing your own advisors (see "Creating a custom advisor" on page 61). Using the advisors is optional but recommended.

- To configure and manage the executor, advisors, and manager, use the command line (**dscontrol**) or the graphical user interface (**lbadmin**).

The three key functions of Dispatcher (executor, manager, and advisors) interact to balance and dispatch the incoming requests between servers. Along with load balancing requests, the executor monitors the number of new connections, active connections, and connections in a finished state. The executor also does garbage collection of completed or reset connections and supplies this information to the manager.

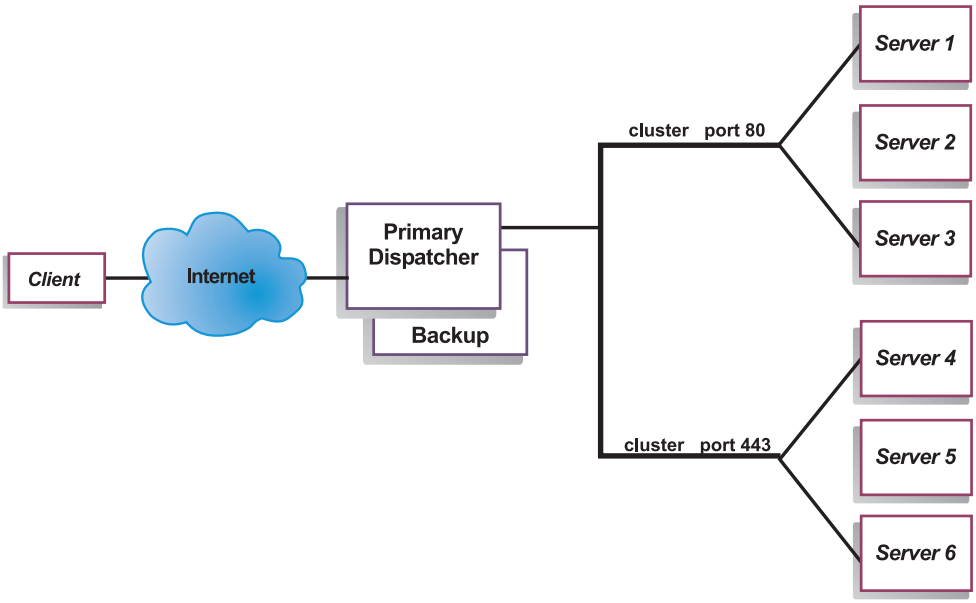
The manager collects information from the executor, the advisors, and a system-monitoring program, such as Metric Server. Based on the information the manager receives, it adjusts how the server machines are weighted on each port and gives the executor the new weighting for use in its balancing of new connections.

The advisors monitor each server on the assigned port to determine the server's response time and availability and then give this information to the manager. The advisors also monitor whether a server is up or down. Without the manager and the advisors, the executor does round-robin scheduling based on the current server weights.

High availability with Load Balancer

The Dispatcher component offers a built-in high availability feature, eliminating Dispatcher as a single point of failure from your network. This feature involves the use of a second Dispatcher machine that monitors the main, or primary, machine and stands by to take over the task of load balancing should the primary machine fail at any time.

Functioning in conjunction with content hosts, such as WebSphere Application Server, the Load Balancer Dispatcher component enables you to enhance your network's availability and scalability. Load Balancer is used by enterprise networks and is installed between the Internet and the enterprise's back-end servers.

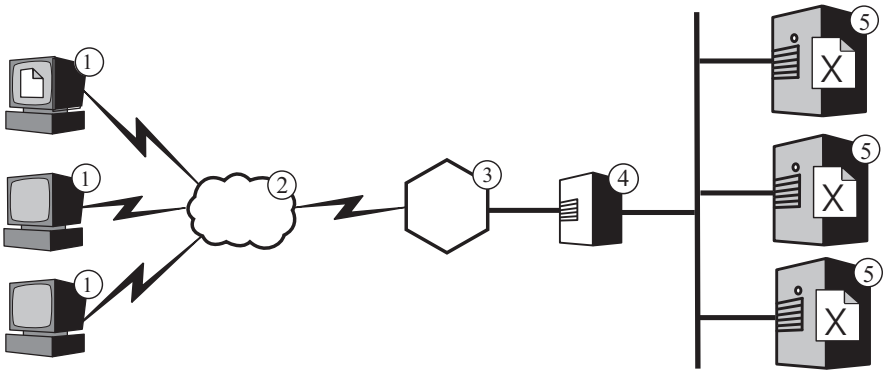


Load Balancer acts as the enterprise's single point-of-presence on the Internet, even if the enterprise uses multiple back-end servers because of high demand or a large amount of content. Availability is achieved through load balancing multiple content hosts and failover support..

Load balancing multiple content hosts

You can satisfy high demand by duplicating content on multiple hosts, but then you need a way to balance the load among them. Domain Name Service (DNS) can provide basic round-robin load balancing, but there are several situations in which it does not perform well.

A more sophisticated solution for load balancing multiple content hosts is to use the Dispatcher component as depicted below.



Legend: 1--Client 2--Internet 3--Router/Gateway 4--Dispatcher 5--Content host

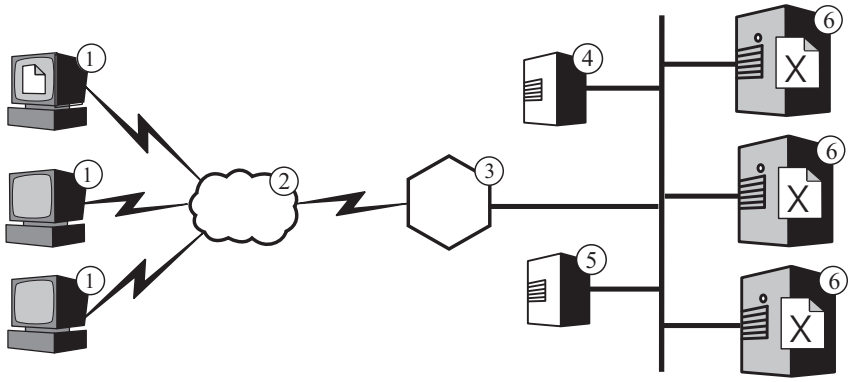
In this configuration, all of the content hosts (the machines marked 5) store the same content. They are defined to form a load-balanced cluster, and one of the network interfaces of the Load Balancer machine (4) is assigned a host name and IP address dedicated to the cluster. When an end user working on one of the machines marked 1 requests file X, the request crosses the Internet (2) and enters

the enterprise's internal network through its Internet gateway (3). The Dispatcher intercepts the request because its URL is mapped to the Dispatcher's host name and IP address. The Dispatcher determines which of the content hosts in the cluster is currently best able to service the request, and forwards the request to that host, which returns file X directly to the client (that is, file X does not pass through Load Balancer).

By default, the Dispatcher uses weighted round-robin load balancing, and it addresses many of DNS's inadequacies. Unlike DNS, it tracks whether a content host is unavailable or inaccessible and does not continue to direct clients to an unavailable content host. Further, it considers the current load on the content hosts by tracking new, active, and finished connections. You can further optimize load balancing by activating Load Balancer's optional advisor and manager components, which track a content host's status even more accurately and incorporate the additional information into the load-balancing decision process. The manager enables you to assign different weights to the different factors used in the decision process, further customizing load balancing for your site.

Failover support

Load Balancer acts as a single point-of-presence for your enterprise's content hosts. This is beneficial because you advertise the cluster host name and address in DNS, rather than the host name and address of each content host, which provides a level of protection against casual attacks and provides a unified feel for your enterprise's Web site. To further enhance Web site availability, configure another Load Balancer to act as a backup for the primary Load Balancer, as depicted in below. If one Load Balancer fails or becomes inaccessible due to a network failure, end users can still reach the content hosts.



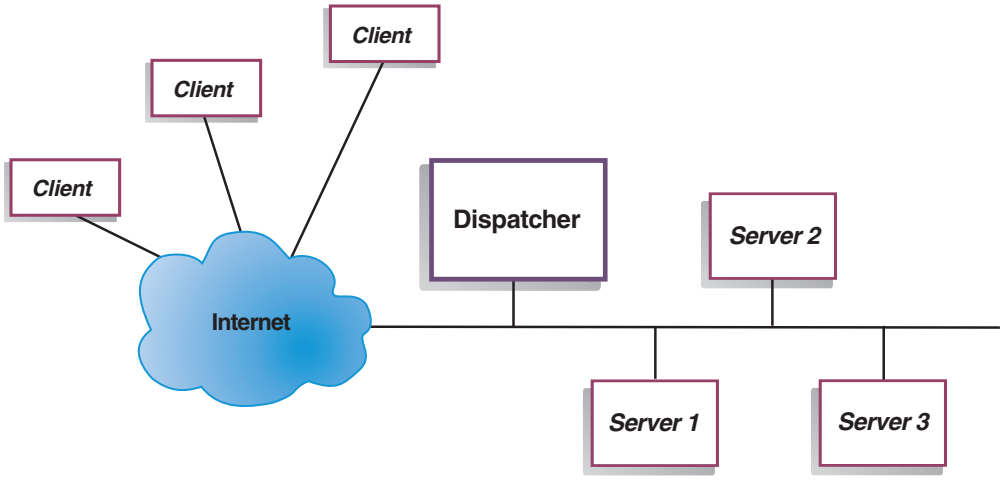
Legend: 1--Client 2--Internet 3--Router/Gateway 4--Primary Dispatcher 5--Backup Dispatcher 6--Content host

In the normal case, a browser running on one of the machines marked 1 directs its request for a file X to the cluster host name that is mapped to the primary Load Balancer (4). The Dispatcher routes the request to the content host (6) selected on the basis of the Dispatcher's load-balancing criteria. The content host sends file X directly to the browser, routing it through the enterprise's gateway (3) across the Internet (2) but bypassing Load Balancer. The backup Dispatcher (5) does not perform load balancing as long as the primary one is operational. The primary and backup Dispatchers track each other's status by periodically exchanging messages called heartbeats. If the backup Dispatcher detects that the primary has failed, it automatically takes over the responsibility for load balancing by intercepting requests directed to the primary's cluster host name and IP address.

Managing servers

You can load balance traffic to existing server topologies without changing the physical configuration of the machines or how clients will connect to your site.

The figure below shows a physical representation of the site using an Ethernet network configuration.

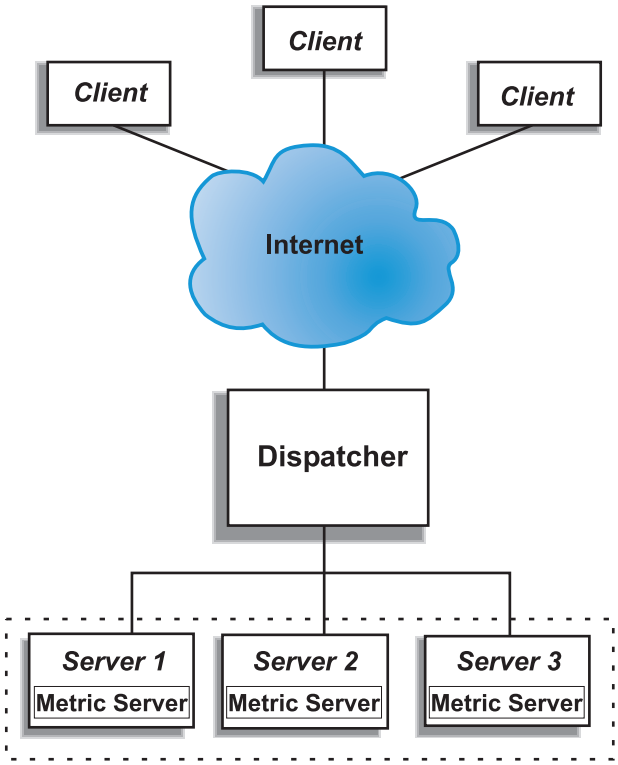


The Dispatcher machine can be installed without making any physical changes to the network. After a client request is directed to the optimal server by the Dispatcher, the response is then sent directly from server to client with no involvement by the Dispatcher.

Read “Types of cluster, port, and server configurations” on page 8 for examples of the different types of configurations you can use with Load Balancer.

Managing servers with Load Balancer and Metric Server

The figure below illustrates a site in which all servers are on a local network. The Dispatcher component is used to forward requests, and the Metric Server is used to provide system load information to the Dispatcher machine.



In this example, the Metric Server daemon is installed on each back-end server. You can use Metric Server with the Dispatcher component.

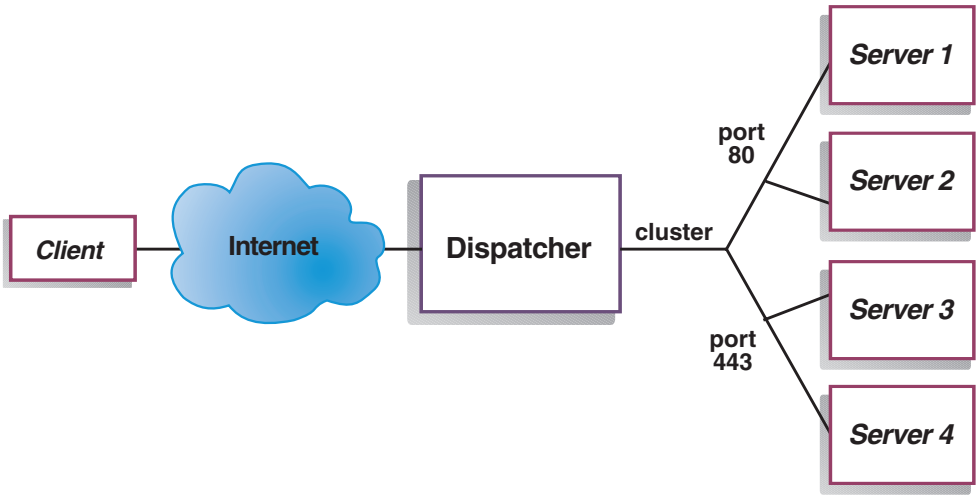
For more information on Metric Server refer to "Getting advice with Metric Server" on page 58

Types of cluster, port, and server configurations

There are many ways that you can configure Load Balancer to support your site.

1 cluster with 2 ports

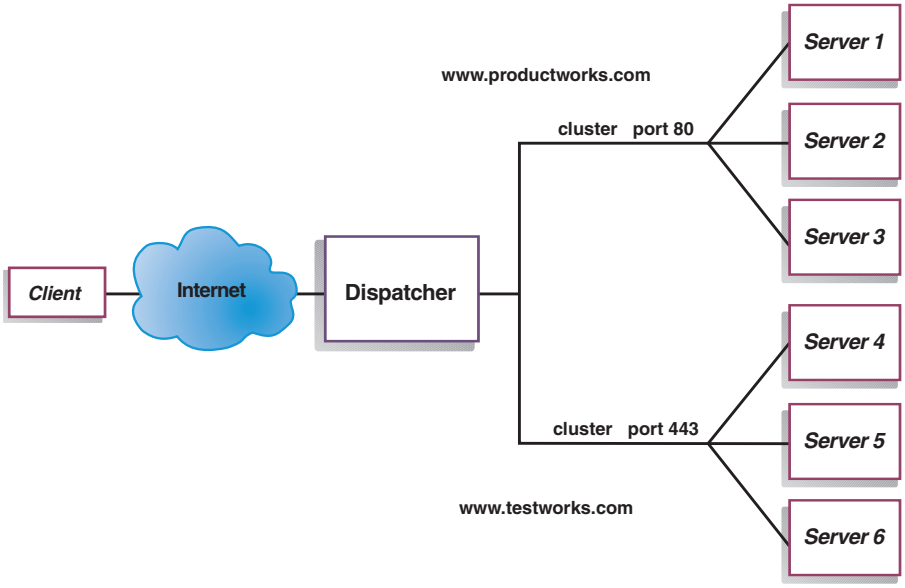
If you have only one host name for your site to which all of your customers will connect, you can define a single cluster of servers. For each of these servers, configure a port through which Load Balancer communicates.



In this example for the Dispatcher component, one cluster is defined at www.productworks.com. This cluster has two ports: port 80 for HTTP and port 443 for SSL. A client making a request to http://www.productworks.com (port 80) goes to a different server than a client requesting https://www.productworks.com (port 443).

2 clusters, each with 1 port

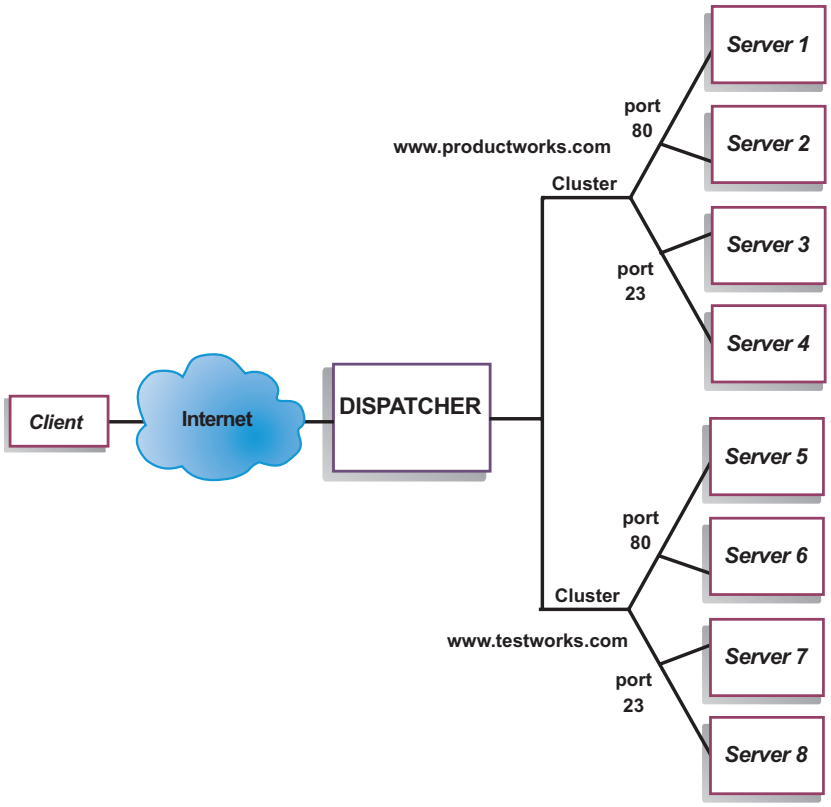
Another way of configuring Load Balancer might be appropriate if you have a very large site with many servers dedicated to each protocol supported. In this case, you might want to define a cluster for each protocol with a single port but with many servers.



In this example for the Dispatcher component, two clusters are defined: www.productworks.com for port 80 (HTTP) and www.testworks.com for port 443 (SSL). A third way of configuring Load Balancer might be necessary if your site does content hosting for several companies or departments, each one coming into your site with a different URL. In this case, you might want to define a cluster for each company or department and then define any ports to which you want to

receive connections at that URL, as shown in the configuration for 2 clusters, each with two ports.

2 clusters, each with 2 ports



In this example for the Dispatcher component, two clusters are defined with port 80 for HTTP and port 23 for Telnet for each of the sites at www.productworks.com and www.testworks.com.

Chapter 2. Installing Load Balancer

This section describes how to establish load balancing capability in new and existing environments. The information includes planning, preparing for, completing, and maintaining product installations.

"Installing Load Balancer"
This section provides an overview of how to install and customize the product on a variety of distributed operating systems.

"Uninstalling Load Balancer" on page 22
This section provides an overview of how to uninstall the product.

Installing Load Balancer

Install Load Balancer using system packaging tools or the command line for all operating systems.

For information on hardware and software requirements, including supported browsers, refer to the following Web page: http://www.ibm.com/support/docview.wss?rs=180&uid=swg27006921.

It is important to note that any previous Load Balancer must be uninstalled before installing Load Balancer for IPv4 and IPv6. Two Load Balancers cannot be installed on the same machine. If you have an earlier version installed, uninstall that copy before installing the current version. Refer to "Uninstalling Load Balancer" on page 22 for more information.

- For AIX operating systems, read "Installing Load Balancer on AIX systems."
- For HP-UX operating systems, read "Installing Load Balancer on HP-UX systems" on page 14.
- For Linux operating systems, read "Installing Load Balancer on Linux operating systems" on page 16.
- For Solaris operating systems, read "Installing Load Balancer on Solaris operating systems" on page 19.
- For Windows operating systems, read "Installing Load Balancer on Windows operating systems" on page 21.

Installing Load Balancer on AIX systems

This topic instructs you on Load Balancer installation using system packaging tools and requirements for AIX operating systems.

For information on hardware and software requirements, including supported browsers, refer to the following Web page: http://www.ibm.com/support/docview.wss?rs=180&uid=swg27006921.

You cannot have two installations of the Dispatcher component installed on the same system. If you have a previous version of the Edge components installed, uninstall the Dispatcher component before starting the installation process for Load Balancer for IPv4 and IPv6. Refer to "Uninstalling Load Balancer" on page 22 for more information.

The Java 2 SDK automatically installs with Load Balancer on all platforms. If you are migrating from a previous version of Load Balancer, or reinstalling the operating system, prior to installation you can save any of your previous configuration files or script files for Load Balancer.

- After installation, place your configuration files in the *install_root*/configurations/dispatcher directory.
- After installation, place your script files in the *install_root*/servers/bin directory in order to run them.

1. Log in as root, or ensure that you have root authority to install the software.
2. Insert the product media, or if you are installing from the Web, copy the installation images to a directory.
3. Install the installation image.

   The following is the list of packages:

Table 1. AIX Install Images

Package Name	**Install Image**
Base	ibmulb-base-7.0.0-0.noarch-rte
Dispatcher	ibmulb-disp-7.0.0-0.noarch.rte
License	ibmulb-lic-7.0.0-0.noarch-rte
Metric Server	ibmulb-ms-7.0.0-0.noarch.rte
Native	ibmulb-7.0.0-0.ppc64.rte Substitute ppc for ppc64 when it is appropriate for your system.
Messages	ibmulb-lang_*language*.7.0.0-0.noarch.rte where *language* can be: • cs_CZ • en_US • de_DE • es_ES • fr_FR • hu_HU • it_IT • ja_JP • ko_KR • pl_PL • pt_BR • ru_RU • zh_CN • zh_TW

You can choose to install some of the packages if you do not want the entire product installated. If you want to install Dispatcher, install the following packages:

- Base
- Dispatcher
- License
- Native

If you want to install Metric Server, install the following packages:

- Base
- Metric Server
- Native

Note: Use SMIT to install Load Balancer for AIX because SMIT will ensure that all messages are installed automatically.

a. **Using SMIT:**
   1) Select Install and Update Software
   2) Select Install and update from latest Available Software
   3) Enter the device or directory containing the install images
   4) Select Software Installation and Maintenance
   5) Enter on the *SOFTWARE to Install line, the appropriate information to specify options (or select List)
   6) Press **OK**.

   When the command completes, press **Done**, and then select **Exit Smit** from the Exit menu or press **F12**. If using SMITTY, press **F10** to exit the program.

b. **Using the Command Line:**
   1) If installing from a CD, you must enter the following commands to mount the CD:

      ```
      mkdir /cdrom
      mount -v cdrfs -p -r /dev/cd0 /cdrom
      ```

   2) Enter the following command to install the desired Load Balancer packages for AIX systems:

      ```
      installp -acXgd device install_image
      ```

      where *install_image* corresponds to an install image name from the table above, and *device* is:

      - `/cdrom` if you are installing from a CD.
      - `/dir` (the directory containing the install images) if you are installing from a file system.

      Ensure that the result column in the summary contains SUCCESS for each part of Load Balancer that you are installing. Do not continue until all of the parts you want to install are successfully applied.

      Note: To generate a list of file sets in any install image, including all available message catalogs, enter the following command:

      ```
      installp -ld device
      ```

      where *device* is:

      - `/cdrom` if you are installing from a CD.
      - `/dir` (the directory containing the install images) if you are installing from a file system.

c. Unmount the CD-ROM. Enter the following command:

   ```
   unmount /cdrom
   ```

4. Verify that the product is installed. Enter the following command:

   ```
   lslpp -h | grep ibmulb
   ```

If you successfully installed the full product, this command returns a list of all the packages.

The installation process does not add the command directories for Load Balancer into the PATH environment variable. To run Load Balancer commands from the system root, add the command directories to the PATH environment variable.

Note: If you had previous installation of the Dispatcher component installed, be aware that Load Balancer for IPv4 versions of the Dispatcher component used commands in the /usr/bin directory, which might be in the PATH variable. Load Balancer for IPv4 and IPv6 has commands in the *install_root*/bin directory, so be aware that the directory entries point to the appropriate directories for the dsserver and dscontrol command.

Related tasks

"Installing Load Balancer" on page 11
Install Load Balancer using system packaging tools or the command line for all operating systems.

"Uninstalling Load Balancer" on page 22

Installing Load Balancer on HP-UX systems

This topic instructs you on Load Balancer installation using system packaging tools and requirements for HP-UX operating systems.

For information on hardware and software requirements, including supported browsers, refer to the following Web page: http://www.ibm.com/support/docview.wss?rs=180&uid=swg27006921.

You cannot have two installations of the Dispatcher component installed on the same system. If you have a previous version of the Edge components installed, uninstall the Dispatcher component before starting the installation process for Load Balancer for IPv4 and IPv6. Refer to "Uninstalling Load Balancer" on page 22 for more information.

The Java 2 SDK automatically installs with Load Balancer on all platforms. If you are migrating from a previous version of Load Balancer, or reinstalling the operating system, prior to installation you can save any of your previous configuration files or script files for Load Balancer.

- After installation, place your configuration files in the *install_root*/configurations/dispatcher directory.
- After installation, place your script files in the *install_root*/servers/bin directory in order to run them.

1. Ensure that you have root level access, and login as the local superuser root. Enter the following command:

   ```
   su - root
   Password: password
   ```

2. Issue the install command:

   ```
   swinstall -s /source package_name
   ```

   where *source* is the absolute directory path for the location of the package, and *package_name* is the name of the package.

   The following is the list of packages:

Table 2. HP-UX Install Images

Package Name	Install Image
Base	ibmulb-base-7.0.0-0.noarch.depot

Table 2. HP-UX Install Images (continued)

Package Name	Install Image
Dispatcher	ibmulb-disp-7.0.0-0.noarch.depot
License	ibmulb-lic-7.0.0-0.noarch.depot
Metric Server	ibmulb-ms-7.0.0-0.noarch.depot
Native	ibmulb-native-7.0.0-0.parisc.depot Substitute ia64 for parisc when it is appropriate for your system.
Messages	ibmulb-lang_*language*.7.0.0-0.noarch.depot where *language* can be: • cs_CZ • en_US • de_DE • es_ES • fr_FR • hu_HU • it_IT • ja_JP • ko_KR • pl_PL • pt_BR • ru_RU • zh_CN • zh_TW

You can choose to install some of the packages if you do not want the entire product installated. If you want to install Dispatcher, install the following packages:

- Base
- Dispatcher
- License
- Native

If you want to install Metric Server, install the following packages:

- Base
- Metric Server
- Native

The following command installs just the base package for Load Balance, ibmulb.base, if you are installing the packages from the root of the CD:

```
swinstall -s /source ibmulb.base
```

To install all the packages for Load Balancer issue the following command, if you are installing the packages from the root of the CD:

```
swinstall -s /source ibmulb
```

3. Verify the installation of the Load Balancer packages. Issue the `swlist` command to list all the packages that you installed. For example:

```
swlist -l fileset ibmulb
```

The installation process does not add the command directories for Load Balancer into the PATH environment variable. To run Load Balancer commands from the system root, add the command directories to the PATH environment variable.

Note: If you had previous installation of the Dispatcher component installed, be aware that Load Balancer for IPv4 versions of the Dispatcher component used commands in the /usr/bin directory, which might be in the PATH variable. Load Balancer for IPv4 and IPv6 has commands in the *install_root*/bin directory, so be aware that the directory entries point to the appropriate directories for the dsserver and dscontrol command.

Related tasks

"Installing Load Balancer" on page 11
Install Load Balancer using system packaging tools or the command line for all operating systems.

"Uninstalling Load Balancer" on page 22

Installing Load Balancer on Linux operating systems

This topic instructs you on Load Balancer installation using system packaging tools and requirements for Linux operating systems.

For information on hardware and software requirements, including supported browsers, refer to the following Web page: http://www.ibm.com/support/docview.wss?rs=180&uid=swg27006921.

You cannot have two installations of the Dispatcher component installed on the same system. If you have a previous version of the Edge components installed, uninstall the Dispatcher component before starting the installation process for Load Balancer for IPv4 and IPv6. Refer to "Uninstalling Load Balancer" on page 22 for more information.

The Java 2 SDK automatically installs with Load Balancer on all platforms. If you are migrating from a previous version of Load Balancer, or reinstalling the operating system, prior to installation you can save any of your previous configuration files or script files for Load Balancer.

- After installation, place your configuration files in the *install_root*/configurations/dispatcher directory.
- After installation, place your script files in the *install_root*/servers/bin directory in order to run them.

Linux **Special Considerations for Linux systems**

- **Linux on zSeries systems require libstdc++.so.5:** There is a requirement that Linux on zSeries systems must have rpm package libstdc++.so.5 in order to install correctly, otherwise the install will fail.
- **Restriction when using qeth/OSA interface:** For Linux on zSeries systems, there is a restrictions when using a qeth/OSA interface. Forwarding out of a qeth/OSA interface natively is not supported. However, there is a workaround because Linux systems run in user space and can support Linux tunneling.
- **Use layer2 OSA with Load Balancer for IPv4 and IPv6 protocols, if available:** When you use Load Balancer for the IPv4 and IPv6 protocols on Linux for zSeries (s390x) operating system, use an OSA/qeth device and layer2 to possibly improve performance, reduce overhead, and simplify some configuration settings.

Non-ethernet, non-Address Resolution Protocol (ARP) interface types present specific challenges to Load Balancer because Load Balancer uses ARP and ICMP6 (Internet Control Message Protocol for IPv6) to advertise and move cluster addresses in high availability mode. The most effective way to deploy the Load Balancer Dispatcher component on the Linux for zSeries operating system is to deploy in an ethernet-like environment. Using OSA/qeth in layer2 mode provides this capability.

Special configuration steps are not required when you use Load Balancer for IPv4 and IPv6 with layer-2 OSA on the Linux for zSeries operating system.

- **Linux tunneling support:** Load Balancer for IPv4 and IPv6 installations can forward across tunnels such as IPIP and IPGRE. When using Linux on zSeries machines with a qeth/OSA interface, a Linux tunnel may be defined to traverse the qeth/OSA interface. Linux systems can forward between machines located on the same or other qeth/OSA devices, or anywhere else on the network.

1. Prepare to install. Log in as root.
2. Issue the install command from the same directory where the RPM files reside. Issue the following command to install each package:

   `rpm -i package.rpm`

   The following is the list of packages:

Table 3. Linux Install Images

Package Name	Install Image
Base	ibmulb-base-7.0.0-0.noarch.rpm
Dispatcher	ibmulb-disp-7.0.0-0.noarch.rpm
License	ibmulb-lic-7.0.0-0.noarch.rpm
Metric Server	ibmulb-ms-7.0.0-0.noarch.rpm
Native	ibmulb-native-7.0.0-0.i386.rpm where i386 can be • i386 • ppc • ppc64 • s390 • s390x • x86_64

Table 3. Linux Install Images (continued)

Package Name	Install Image
Messages	ibmulb-lang_*language*.7.0.0-0.noarch.rpm where *language* can be: • cs_CZ • en_US • de_DE • es_ES • fr_FR • hu_HU • it_IT • ja_JP • ko_KR • pl_PL • pt_BR • ru_RU • zh_CN • zh_TW

You can choose to install some of the packages if you do not want the entire product installated. If you want to install Dispatcher, install the following packages:

- Base
- Dispatcher
- License
- Native

If you want to install Metric Server, install the following packages:

- Base
- Metric Server
- Native

Red Hat Linux systems: Due to a known Red Hat Linux problem, you will also need to delete the _db* RPM files, or an error might occur.

3. Verify that the product is installed. Enter the following command:

   ```
   rpm -qa | grep ibmulb
   ```

   Installing the full product produces a listing like the following example:

   ```
   ibmulb-base-7.0.0-0.noarch.rpm
   ibmulb-disp-7.0.0-0.noarch.rpm
   ibmulb-lic-7.0.0-0.noarch.rpm
   ibmulb-ms-7.0.0-0.noarch.rpm
   ibmulb-native-7.0.0-0.i386.rpm
   ibmulb-lang_language.7.0.0-0.noarch.rpm
   ```

The installation process does not add the command directories for Load Balancer into the PATH environment variable. To run Load Balancer commands from the system root, add the command directories to the PATH environment variable.

Note: If you had previous installation of the Dispatcher component installed, be aware that Load Balancer for IPv4 versions of the Dispatcher component used commands in the /usr/bin directory, which might be in the PATH variable. Load

Balancer for IPv4 and IPv6 has commands in the *install_root*/bin directory, so be aware that the directory entries point to the appropriate directories for the dsserver and dscontrol command.

Related tasks

"Installing Load Balancer" on page 11
Install Load Balancer using system packaging tools or the command line for all operating systems.

"Uninstalling Load Balancer" on page 22

Installing Load Balancer on Solaris operating systems

This topic instructs you on Load Balancer installation using system packaging tools and requirements for Solaris operating systems.

For information on hardware and software requirements, including supported browsers, refer to the following Web page: http://www.ibm.com/support/docview.wss?rs=180&uid=swg27006921.

You cannot have two installations of the Dispatcher component installed on the same system. If you have a previous version of the Edge components installed, uninstall the Dispatcher component before starting the installation process for Load Balancer for IPv4 and IPv6. Refer to "Uninstalling Load Balancer" on page 22 for more information.

The Java 2 SDK automatically installs with Load Balancer on all platforms. If you are migrating from a previous version of Load Balancer, or reinstalling the operating system, prior to installation you can save any of your previous configuration files or script files for Load Balancer.

- After installation, place your configuration files in the *install_root*/configurations/dispatcher directory.
- After installation, place your script files in the *install_root*/servers/bin directory in order to run them.

1. Prepare to install.
   a. Log in as root user.
   b. Insert the CD-ROM that contains the Load Balancer software into the appropriate drive.
2. Display the list of packages to install and choose which ones you would like to install.
   a. At the command prompt, enter the command to display the list of packages. Enter the following:

      ```
      pkgadd -d   /path_name/xxx.pkg
      ```

      where *path_name* is the device name of the CD-ROM drive or the directory on the hard drive where the package is located. To use the CD-ROM, for example, enter the following: `pkgadd -d /cdrom/cdrom0/xxx.pkg`. The following is a list of packages:

Table 4. Solaris Install Images

Package Name	Install Image
Base	ibmulb-base-7.0.0-0.noarch.pkg
Dispatcher	ibmulb-disp-7.0.0-0.noarch.pkg
License	ibmulb-lic-7.0.0-0.noarch.pkg

Table 4. Solaris Install Images (continued)

Package Name	Install Image
Metric Server	ibmulb-ms-7.0.0-0.noarch.pkg
Native	ibmulb-native-7.0.0-0.sparc.pkg Substitute sparcv9 for sparc when it is appropriate for your system.
Messages	ibmulb-lang_*language*.7.0.0-0.noarch.pkg where *language* can be: • cs_CZ • en_US • de_DE • es_ES • fr_FR • hu_HU • it_IT • ja_JP • ko_KR • pl_PL • pt_BR • ru_RU • zh_CN • zh_TW

b. Optional: If you want to install some of the components, enter the names corresponding to the packages to be installed separated by a space or comma and press return.

   You can choose to install some of the packages if you do not want the entire product installated. If you want to install Dispatcher, install the following packages:

   - Base
   - Dispatcher
   - License
   - Native

   If you want to install Metric Server, install the following packages:

   - Base
   - Metric Server
   - Native

3. Verify that the product is installed. Issue the following command:

```
pkginfo | grep ibm
```

The installation process does not add the command directories for Load Balancer into the PATH environment variable. To run Load Balancer commands from the system root, add the command directories to the PATH environment variable.

Note: If you had previous installation of the Dispatcher component installed, be aware that Load Balancer for IPv4 versions of the Dispatcher component used commands in the /usr/bin directory, which might be in the PATH variable. Load

Balancer for IPv4 and IPv6 has commands in the *install_root*/bin directory, so be aware that the directory entries point to the appropriate directories for the dsserver and dscontrol command.

Related tasks

"Installing Load Balancer" on page 11
Install Load Balancer using system packaging tools or the command line for all operating systems.

"Uninstalling Load Balancer" on page 22

Installing Load Balancer on Windows operating systems

This topic instructs you on Load Balancer installation using system packaging tools and requirements for Windows operating systems.

For information on hardware and software requirements, including supported browsers, refer to the following Web page: http://www.ibm.com/support/docview.wss?rs=180&uid=swg27006921.

It is important to note that any previous Load Balancer must be uninstalled before installing Load Balancer. Two Load Balancers cannot be installed on the same machine. If you have an earlier version installed, uninstall that copy before installing the current version. Refer to "Uninstalling Load Balancer" on page 22 for more information.

The Java 2 SDK automatically installs with Load Balancer on all platforms. If you are migrating from a previous version of Load Balancer, or reinstalling the operating system, prior to installation you can save any of your previous configuration files or script files for Load Balancer.

- After installation, place your configuration files in the *install_root*/configurations/dispatcher directory.
- After installation, place your script files in the *install_root*/servers/bin directory in order to run them.

1. Insert the product CD-ROM, and the installation launchpad will appear. To manually start the installation program:
   a. Click **Start**.
   b. Select **Run**.
   c. Specify the CD-ROM disk drive, followed by **launchpad.exe**, for example, type:
      ```
      E:\launchpad.exe
      ```
2. Select the option to install Edge components for IPv4 and IPv6.
3. Follow the instructions of the setup program.
   a. Optional: If you want to change the drive or directory in which Load Balancer will be installed click **Browse**
   b. Optional: Choose to install **Typical** to install all of the components, or choose **Custom** to choose the packages based on your preferences and system requirements.
4. Reboot the system when you are prompted by the setup program.

Related tasks

"Installing Load Balancer" on page 11
Install Load Balancer using system packaging tools or the command line for all operating systems.

"Uninstalling Load Balancer" on page 22

Uninstalling Load Balancer

You might want to uninstall Load Balancer before upgrading to a newer version, or if you think the current installation is corrupted.

- First, ensure that you have stopped all the executors and the servers. Uninstall the product by issuing the following command: `installp -u package`.
  - To uninstall the entire product, enter the following command:

    ```
    installp -u ibmulb
    ```

    Use the previous name when it is applicable, for example **intnd**.
  - To uninstall specific file sets, list them instead of specifying the package name. Uninstall the packages in the reverse order in which they were installed (reverse the order of the table below):

Table 5. AIX Install Images

Package Name	Install Image
Base	ibmulb-base-7.0.0-0.noarch-rte
Dispatcher	ibmulb-disp-7.0.0-0.noarch.rte
License	ibmulb-lic-7.0.0-0.noarch-rte
Metric Server	ibmulb-ms-7.0.0-0.noarch.rte
Native	ibmulb-7.0.0-0.ppc64.rte Substitute ppc for ppc64 when it is appropriate for your system.
Messages	ibmulb-lang_*language*.7.0.0-0.noarch.rte where *language* can be: • cs_CZ • en_US • de_DE • es_ES • fr_FR • hu_HU • it_IT • ja_JP • ko_KR • pl_PL • pt_BR • ru_RU • zh_CN • zh_TW

- First, ensure that you have stopped all the executors and the servers. Use the swremove command to uninstall the packages.
  - To uninstall all the Load Balancer packages:

    ```
    swremove ibmulb
    ```
  - To uninstall an individual package, for example the Dispatcher component, enter:

    ```
    swremove ibmulb.disp
    ```

The package names are:

Table 6. HP-UX Install Images

Package Name	Install Image
Base	ibmulb-base-7.0.0-0.noarch.depot
Dispatcher	ibmulb-disp-7.0.0-0.noarch.depot
License	ibmulb-lic-7.0.0-0.noarch.depot
Metric Server	ibmulb-ms-7.0.0-0.noarch.depot
Native	ibmulb-native-7.0.0-0.parisc.depot Substitute ia64 for parisc when it is appropriate for your system.
Messages	ibmulb-lang_*language*.7.0.0-0.noarch.depot where *language* can be: • cs_CZ • en_US • de_DE • es_ES • fr_FR • hu_HU • it_IT • ja_JP • ko_KR • pl_PL • pt_BR • ru_RU • zh_CN • zh_TW

- Linux First, ensure that all the executors and all the servers are stopped. To uninstall the entire product, enter:

  ```
  rpm -e pkgname
  ```

  The package names are:

Table 7. Linux Install Images

Package Name	Install Image
Base	ibmulb-base-7.0.0-0.noarch.rpm
Dispatcher	ibmulb-disp-7.0.0-0.noarch.rpm
License	ibmulb-lic-7.0.0-0.noarch.rpm
Metric Server	ibmulb-ms-7.0.0-0.noarch.rpm

Table 7. Linux Install Images (continued)

Package Name	Install Image
Native	ibmulb-native-7.0.0-0.i386.rpm where i386 can be • i386 • ppc • ppc64 • s390 • s390x • x86_64
Messages	ibmulb-lang_*language*.7.0.0-0.noarch.rpm where *language* can be: • cs_CZ • en_US • de_DE • es_ES • fr_FR • hu_HU • it_IT • ja_JP • ko_KR • pl_PL • pt_BR • ru_RU • zh_CN • zh_TW

- First, ensure that you have stopped all the executors and the servers. Then, to uninstall Load Balancer enter the `pkgrm` command. Enter the following:

  ```
  pkgrm package
  ```

  The package names are:

Table 8. Solaris Install Images

Package Name	Install Image
Base	ibmulb-base-7.0.0-0.noarch.pkg
Dispatcher	ibmulb-disp-7.0.0-0.noarch.pkg
License	ibmulb-lic-7.0.0-0.noarch.pkg
Metric Server	ibmulb-ms-7.0.0-0.noarch.pkg
Native	ibmulb-native-7.0.0-0.sparc.pkg Substitute sparcv9 for sparc when it is appropriate for your system.

Table 8. Solaris Install Images (continued)

Package Name	Install Image
Messages	ibmulb-lang_*language*.7.0.0-0.noarch.pkg where *language* can be: • cs_CZ • en_US • de_DE • es_ES • fr_FR • hu_HU • it_IT • ja_JP • ko_KR • pl_PL • pt_BR • ru_RU • zh_CN • zh_TW

- To uninstall using the **Add/Remove Programs** option:
  1. Click **Start** > **Settings** > **Control Panel**.
  2. Double-click **Add/Remove Programs**.
  3. Select *IBM WebSphere Edge Components* (or previous name, for example, *IBM Edge Server*).
  4. Click **Change/Remove** button.

  Related tasks

  "Installing Load Balancer" on page 11
  Install Load Balancer using system packaging tools or the command line for all operating systems.

Updating Load Balancer

Use these instructions for obtaining and installing updates to installations of Load Balancer for IPv4 and IPv6.

If you are installing refreshes and fix packs for Load Balancer, the only prerequisite that is required for Edge Components Version 7.0 is the license file, which is nd70Full.LIC, because the refresh or fix pack does not provide the license. You can obtain the license by installing the Load Balancer license package.

You can obtain a fix pack in the following mediums:

- Product CDs for newly supported operating systems. If you are installing Load Balancer on operating systems that are newly supported in Version 7.0, install the product from the product CD or DVD.
- Downloadable fix packs for existing installations on operating systems that were previously supported. You can find the link to the refresh packs or fix packs for Edge Components in the Support & downloads web site. Find the corrective service release to which you are upgrading and follow the link to the download site. Follow the instructions on the site to download the Edge Components

refresh pack. You also can download the latest Edge components fix packs from the FTP server for the Edge Components.

- For information on hardware and software requirements, including supported browsers, refer to the following Web page: http://www.ibm.com/support/docview.wss?rs=180&uid=swg27006921.
- Install the update packages for AIX, HP-UX, Linux, or Solaris operating systems.
- Install the update packages for Windows operating systems.

Updating Load Balancer for AIX, HP-UX, Linux, or Solaris operating systems

Use these instructions for obtaining and installing updates to installations of Load Balancer for IPv4 and IPv6 on AIX, HP-UX, Linux, or Solaris operating systems.

Before installing the refresh or fix pack, stop and uninstall any existing versions of Load Balancer that are earlier than Version 7.0. Refer to "Uninstalling Load Balancer" on page 22 for more information.

1. Ensure that you have the license package installed from the CD. You don't have to have the entire product installed; you only need to have the license installed. The license package only comes on the CD, so you need the CD to install the license package.
2. Go to a command prompt with root authority.
3. Obtain the Load Balancer refresh or fix pack and place it in a temporary directory.
4. Uncompress and untar the build package. This results in a number of separate file sets.
5. Install the software using the system-specific commands in the following table. Use the following table for the commands to use for your operating system:

Operating system	Update commands

AIX	1. If a .toc file is not already present, generate a .toc file by issuing the command:
	`inutoc`
	2. Install the packages for Load Balancer. For example, to install the base package from the current directory, issue the following command:
	`installp -acXd . package_name`

HP-UX	`swinstall -s /source package_name`
	
	where *source* is the directory for the location of the package, and *package_name* is the name of the package.
	
	For example, to install the base package from the current directory, issue the following command:
	`swinstall -s /lb package_name`

Operating system	Update commands

Linux	`rpm -iv package_name`
	
	where *package_name* is the name of the package.
	
	For example, the following command installs all of the packages for Load Balancer when the packages reside in the current directory:
	`rpm -iv ibmulb*.rpm`
	
	Note: You can use the nodeps option to successfully install all of the packages in any order.

Solaris	`pkgadd -d source package_name`
	
	where *source* is the directory for the location of the package, and *package_name* is the name of the package.
	
	For example, to install the administration package from the current directory, issue the following command:
	`pkgadd -d . ibmulbadm`

6. Restore any configuration files and start scripts that you saved or modified during a previous uninstall.

After you install an update for Edge Components, the previous configuration for Edge Components is maintained. When new functions or enhancements are delivered with a refresh or fix pack, it might be necessary to add directives to the configuration files to enable the features.

Note: When you update the Load Balancer component, you must manually save and restore configuration files to maintain the previous configuration for Load Balancer. See "Installing Load Balancer" on page 11 for more information.

Rejecting an update

- On HP-UX, Linux, or Solaris operating systems, to remove a refresh or fix pack and return to a prepatched state, uninstall the product and reinstall the previous version.
- The mechanism that the AIX operating system provides for rejecting a patch requires that the patch be produced in refresh or fix pack format. The Edge Components refresh or fix pack is provided with product format packaging only, not refresh or fix pack format packaging. Therefore, you cannot use the AIX SMIT mechanisms for installing and removing patches. To reject a patch on an AIX system, you must uninstall the file sets and reinstall the previous version.

Updating Load Balancer for Windows operating systems

Use these instructions for obtaining and installing updates to installations of Load Balancer for IPv4 and IPv6 on Windows operating systems.

Before installing the refresh or fix pack, stop and uninstall any existing versions of Load Balancer that are earlier than Version 7.0. Refer to "Uninstalling Load Balancer" on page 22 for more information.

1. To prevent the currently installed Load Balancer from starting, edit any start scripts that you have created to temporarily suppress any commands that will start Load Balancer upon reboot.
2. Use the **Add or Remove Programs** option to uninstall the current Load Balancer, if it is present.
3. Download the Edge Components refresh or fix pack.
4. Run the installation program.
   - From a command prompt:
     - For a refresh pack, change to the /ulb directory, and enter the following:
       ```
       setup
       ```
     - For a fix pack, the Load balancer fix pack only contains Load balancer installation files and does not include the /ulb folder. Unzip the downloaded package to a folder and enter setup from that folder. For example, navigate to the unzipped files, and enter the following:
       ```
       setup
       ```
   - From the Start menu:
     a. Click **Run**.
     b. Click **Browse**.
        - For a refresh pack, change to the /ulb directory, and select setup.
        - For a fix pack, the Load balancer fix pack only contains Load balancer installation files and does not include the /ulb folder. Select the installation files for your operating system, and select setup.
     c. Click **Open**.
     d. Click **OK**.
     e.
5. Enter information as requested by the installation program.
6. Restart the system.

After you install an update for Edge Components, the previous configuration for Edge Components is maintained. When new functions or enhancements are delivered with a refresh or fix pack, it might be necessary to add directives to the configuration files to enable the features.

Note: When you update the Load Balancer component, you must manually save and restore configuration files to maintain the previous configuration for Load Balancer. See "Installing Load Balancer" on page 11 for more information.

Rejecting an update

1. Use the **Add or Remove Programs** option to uninstall the current Load Balancer.
2. Select **Remove** on the **Maintenance Options** window for the setup program.
3. Use the setup program for Edge Components to reinstall the previous version.

Directory conventions

References in product information to *install_root* and other directories infer specific default directory locations. This topic describes the conventions in use for IBM WebSphere Edge Components.

These file paths are default locations. You can install the product and other components in any directory where you have write access. You can create profiles in any valid directory where you have write access.

Linux

install_root - the root directory in which the product was installed

Load Balancer install paths include the following:

- Administration - /opt/ibm/edge/ulb/admin
- Load Balancer - /opt/ibm/edge/ulb/servers
- Metric Server - /opt/ibm/edge/ulb/ms
- Documentation - /opt/ibm/edge/ulb/docs/

install_root - the root directory in which the product was installed

Load Balancer install paths include the following:

- Administration - C:\Program Files\IBM\edge\ulb\admin
- Load Balancer - C:\Program Files\IBM\edge\ulb\servers
- Metric Server - C:\Program Files\IBM\edge\ulb\ms
- Documentation - C:\Program Files\IBM\edge\ulb\docs

Chapter 3. Configuring Load Balancer

This section focuses on configuring production environments and realistic test environments.

"Methods of configuration"
There are four methods you can use to configure Load Balancer: the command line, scripts, the graphical user interface (GUI), and the configuration wizard. The Information Center assumes use of the command line.

"Configuring the Load Balancer machine" on page 34
Configure Load Balancer for IPv4 and IPv6 on the machine that you will use to load balance server traffic.

"Configuring the server machines" on page 37
You need to configure each of the back-end server machines in your topology before Load Balancer will be able to properly work in your environment.

"Quick start configuration" on page 44
This quick start example shows how to configure three locally attached workstations to load-balance Web traffic between two Web servers.

"Logging with Load Balancer" on page 92
You can set up Dispatcher and the TCP server machines using a private network. This configuration can reduce the contention on the public or external network that can affect performance.

Methods of configuration

There are four methods you can use to configure Load Balancer: the command line, scripts, the graphical user interface (GUI), and the configuration wizard. The Information Center assumes use of the command line.

There are four basic methods of configuring the Dispatcher:

- "Command line"
- "Scripts" on page 32
- Graphical User Interface (GUI)
- "The configuration wizard" on page 33.

Command line

This is the most direct means of configuring the Dispatcher. The command parameter values must be entered in English characters. The only exceptions are host names (used in cluster, server, and high availability commands) and file names (used in file commands).

To start the Dispatcher from the command line:

1. Start dsserver:
   - Linux From the command prompt, issue the following:

     ```
     dsserver
     ```

     To stop the service, type:

```
dsserver stop
```

- Click **Start** > **Control Panel** > **Administrative Tools** > **Services**. Right-click **IBM Dispatcher (ULB)** and select **Start**. To stop the service, follow the same steps and select **Stop**.

2. Issue the Dispatcher control commands you want in order to set up your configuration. The command is `dscontrol`. For more information about commands, see "Commands" on page 137.

You can use a minimized version of the dscontrol command parameters by typing the unique letters of the parameters. For example, to get help on the file save command, you can type

```
dscontrol he f
```

instead of

```
dscontrol help file
```

To start up the command line interface issue `dscontrol` to receive an dscontrol command prompt. To end the command line interface issue the commands `exit` or `quit`.

Scripts

You can enter commands for configuring Dispatcher into a configuration script file and run them together. See Sample Load Balancer configuration files. To quickly run the content of a script file (for example, myscript), use either of the following commands:

- To update the current configuration, run the following executable commands from your script file:

  ```
  dscontrol file appendload myscript
  ```

- To completely replace the current configuration, run the following executable commands from your script file:

  ```
  dscontrol file newload myscript
  ```

To save the current configuration into a script file (for example, savescript), run the following command:

```
dscontrol file save savescript
```

This command will save the configuration script file in the *install_root*/servers/configurations/dispatcher directory.

GUI

To start the GUI, follow these steps:

1. Ensure **dsserver** is running:
   - Linux Run the following as root user:

     ```
     dsserver
     ```

   - **dsserver** runs as a service that starts automatically.
2. Start the Load Balancer GUI:
   - Linux Run the following:

     ```
     lbadmin
     ```

   - Click **Start** > **Programs** > **IBM WebSphere** > **Edge Components** > **IBM Load Balancer** > **Load Balancer**

To configure the Dispatcher component from the GUI, you must first select **Dispatcher** in the tree structure. You can start the executor and manager after you connect to a **Host**. You can also create clusters containing ports and servers, and start advisors for the manager.

The GUI can be used to do anything that you would do with the `dscontrol` command. For example, to define a cluster using the command line, you would enter `dscontrol cluster add` *`cluster`* command. To define a cluster from the GUI, right-click **Executor**, then in the popup menu, left-click **Add Cluster**. Enter the cluster address in the popup window, then click **OK**.

Pre-existing Dispatcher configuration files can be loaded using the **Load New Configuration** (for completely replacing the current configuration) and **Append to Current Configuration** (for updating the current configuration) options presented in the **Host** popup menu. You should save your Dispatcher configuration to a file periodically using the **Save Configuration File As** option also presented in the **Host** popup menu. The **File** menu located at the top of the GUI will allow you to save your current host connections to a file or restore connections in existing files across all Load Balancer components.

In order to run a command from the GUI:

1. Highlight the Host node from the GUI tree and select **Send command...** from the Host pop-up menu.
2. In the command entry field, type the command that you want to run, for example:

   ```
   executor report
   ```

   The results and history of the commands run in the current session and appear in the window provided.

You can access Help by clicking the question mark icon in the upper right corner of the Load Balancer window.

- **Help: Field level** -- describes each field, default values
- **Help: How do I** -- lists tasks that can be done from that screen
- **InfoCenter** -- provides centralized access to product information

The configuration wizard

The wizard guides you step by step through the process of creating a basic configuration for the Dispatcher component. You will be asked questions about your network. You will be guided through the setup of a cluster for Dispatcher to load balance traffic between a group of servers.

If you are using the configuration wizard, follow these steps:

1. Start the **dsserver** on Dispatcher:
   - Linux Run the following as root user:

     ```
     dsserver
     ```

   - **dsserver** runs as a service that starts automatically.
2. Start the wizard function of Dispatcher by issuing the following command:

   ```
   dswizard
   ```

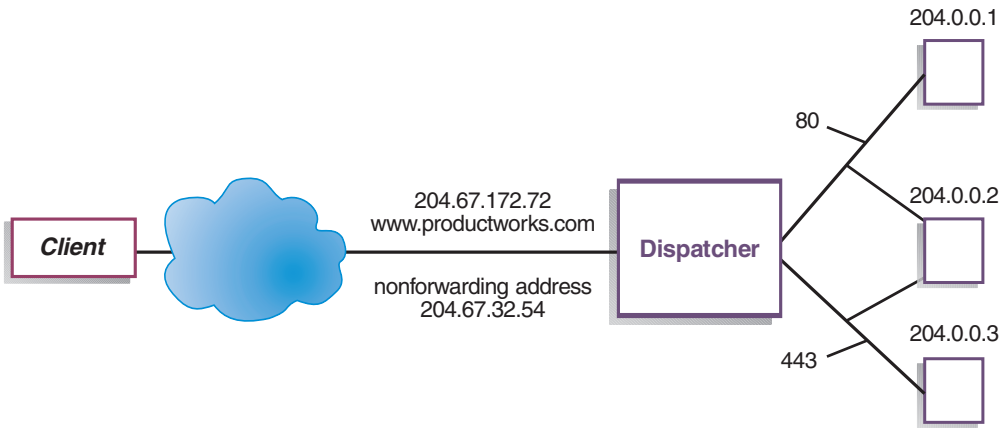
Configuring the Load Balancer machine

Configure Load Balancer for IPv4 and IPv6 on the machine that you will use to load balance server traffic.

Before setting up the Dispatcher machine, you must be the root user (for AIX, HP-UX, Linux, or Solaris systems) or the Administrator on Windows systems.

For the Dispatcher machine you will need at least two valid IP addresses.

The figure below shows an example of Load Balancer set up with a single cluster, two ports, and three servers.



You must configure the machine on which Load Balancer is installed before you can load balance traffic in your network environment.

Note: Be aware of the following back-end server restrictions:

Solaris back-end server

There is no support for load balancing IPv6 traffic to back-end Solaris 5.8 servers. On Solaris 5.8, there is an incompatibility with a MAC-forwarded IPv6 packet and the Solaris IPv6 stack. When the cluster is configured on a Solaris 5.8 back-end server using the ifconfig lo0 (loopback) command, the packet arrives at the Solaris 5.8 node, but is not accepted. However, you can use Load Balancer for IPv4 and IPv6 installations to load balance IPv4 traffic to back-end Solaris 5.8 servers.

z/OS back-end servers

There is no support for load-balancing IPv6 traffic to back-end z/OS servers. However, you can load balance IPv4 traffic to back-end z/OS servers using Load Balancer for IPv4 and IPv6 installations.

1. Linux Start the server function. To start the server function, type the following at a command prompt:

   ```
   dsserver
   ```

   Note: A default configuration file (default.cfg) gets automatically loaded when starting **dsserver**. If the user decides to save the Dispatcher configuration in default.cfg, then everything saved in this file is automatically loaded next time **dsserver** gets started.
2. Start the executor function.

a. Optional: If you are using IPv6 addresses, enable the processing of IPv6 packets.

   Linux Prior to starting the executor (dscontrol executor start), the following must be issued from the command line as root:

   - AIX
     ```
     autoconf6
     ```
     To enable uninterrupted processing of IPv6 packets, even after a system reboot, edit the etc/rc.tcpip file and uncomment the following line, and add the -A flag:
     ```
     start usr/bin/autoconf6 " " -A
     ```
   - Linux
     ```
     modprobe ipv6
     ```
   - Windows
     ```
     netsh interface ipv6 install
     ```

   These commands enable processing of IPv6 packets in the respective operating systems. Issue this command only once. Thereafter, you can start and stop the executor as often as you need. If you do not issue the command to enable processing of IPv6 packets on these systems, the executor will not start.

   Solaris Using the **ifconfig** command, IPv6 addresses must be plumbed and an interface configured in order for Dispatcher to inspect IPv6 packets. If you do not issue these commands, the executor will start, but no IPv6 packets can be viewed. Prior to starting the executor (dscontrol executor start), issue the following from the command line as root:

   - ```
     ifconfig device inet6 up
     ```
   - Change the device to your device name, and change the IPv6 IP address and prefix to your address and prefix values:
     ```
     ifconfig device inet6 plumb
     ifconfig device inet6 address/prefix up
     ```

b. To start the executor function, enter the dscontrol executor start command. You may also change various executor settings at this time.

3. Optional: Define the non-forwarding address if it is different from the host name. The non-forwarding address is used to connect to the machine for administrative purposes, such as using Telnet or SMTP to this machine.

   By default, this address is the hostname. To define the non-forwarding address, enter the following command, or edit the configuration file:
   ```
   dscontrol executor set nfa IP_address
   ```

   where *IP_address* is either the symbolic name or the IP address.

4. Define a cluster and set cluster options. Dispatcher will balance the requests sent to the cluster address to the servers configured on the ports for that cluster. The cluster is either the symbolic name, the dotted decimal address, or the special address 0.0.0.0 that defines a wildcard cluster. Wildcard clusters can be used to match multiple IP addresses for incoming packets to be load balanced.

   a. To define a cluster, issue the dscontrol cluster add command:
      ```
      dscontrol cluster add cluster
      ```

   b. To set cluster options, use the dscontrol cluster set command. Issue the following command:
      ```
      dscontrol cluster set options
      ```
      ```

c. Linux If you use a qeth/OSA implementation on Linux on zSeries, the following additional configuration steps are required to setup Load Balancer:

1) 1. Configure the cluster address using ip or ifconfig command:

```
ip -version addr add cluster_address/prefix_length dev device
```

For example:

```
ip -4 addr add 12.42.38.125/24 dev eth0
ip -6 addr add 3ffe:34::24:45/64 dev eth0
```

2) Add an iptables rule to drop incoming packets destined to the cluster address:

For IPv4 addresses:

```
iptables -t filter -A INPUT -d cluster_address -j DROP
```

For IPv6 addresses:

```
ip6tables -t filter -A INPUT -d cluster_address -j DROP
```

For example:

```
iptables -t filter -A INPUT -d 12.42.38.125 -j DROP
ip6tables -t filter -A INPUT -d 3ffe:34::24:45 -j DROP
```

3) To undo the above configuration, use the following commands:

```
ip -version addr del cluster_address/prefix_length dev device
iptables -t filter -D INPUT -d cluster_address -j DROP
ip6tables -t filter -D INPUT -d cluster_address -j DROP
```

5. Define ports and set port options with the dscontrol port add command. You must define and configure all servers for a port.

a. Define a port. Enter the following command:

```
dscontrol port add cluster@port
```

- *cluster* is either the symbolic name or the IP address
- *port* is the number of the port you are using for that protocol

b. Change various port settings. Read "dscontrol port" on page 154 for more information on this command and the available options.

Note: You can select a new option for the selection algorithm that Load Balancer uses to route traffic:

- **conn+affinity:** Specifies that server selection is based on an existing connection. For new connections, the server selection is based on affinity.

You can also edit the sample configuration file or use the GUI.

6. Define the load-balanced server machines. To define a load-balanced server machine, enter the following command:

```
dscontrol server add cluster@port@server
```

You can also edit the sample configuration file or use the GUI. *Cluster* is either the symbolic name or the IP address, and *port* is the number of the port you are using for that protocol. You must define more than one server to a port on a cluster in order to perform load balancing.

a. **Configure IPv6 link-local address:** With IPv6 addressing, each machine in the Load Balancer configuration must have an IPv6 link-local address. The link-local address is the address used for neighbor discovery traffic for IPv6, and without this address on the Load Balancer machine and on the back-end servers neighbor discovery does not occur, and the machines are not known to each other. Load Balancer for IPv6 cannot forward traffic

without a link-local IPv6 address configured on an interface of each machine in the Load Balancer configuration.

b. Optional: **Bind-specific servers:** If the Dispatcher component is load balancing to bind-specific servers, then the servers must be configured to bind to the cluster address. Because the Dispatcher forwards packets without changing the destination IP address, when the packets reach the server, the packets will still contain the cluster address as the destination. If a server has been configured to bind to an IP address other than the cluster address, then the server will be unable to accept requests destined for the cluster.

   To determine if the server is bind specific, issue the netstat -an command and look for the *server@port*. If the server is not bind specific, the result from this command will be 0.0.0.0@80. If the server is bind specific, you will see an address such as 192.168.15.103@80.

7. Optional: Start the manager function. The manager function improves load balancing. To start the manager, enter the dscontrol manager start command, edit the sample configuration file, or use the GUI. For example:

   ```
   dscontrol manager start
   ```

8. Optional: Start the advisor function. The advisors give the manager more information about the ability of the load-balanced server machines to respond to requests. An advisor is specific to a protocol. For example, to start the HTTP advisor, issue the following command:

   ```
   dscontrol advisor start http port
   ```

   For a list of advisors along with their default ports, see "List of advisors" on page 53.

   a. Set cluster proportions, as required. If you start advisors, you may modify the proportion of importance given to advisor information being included in the load balancing decisions. To set the cluster proportions, issue the dscontrol cluster set cluster proportions command. For more information, see "Tuning the proportion of importance given to status information" on page 102

9. Configure the server machines.

Configuring the server machines

You need to configure each of the back-end server machines in your topology before Load Balancer will be able to properly work in your environment.

Dispatcher will only load balance across servers that allow the loopback adapter to be configured with an additional IP address, for which the back-end server will never respond to ARP (address resolution protocol) requests.

1. Alias the loopback device.

   If you are using certain types of Linux operating systems, you may need to use an alternative method for aliasing the network card and loopback devices. Read "Configuring loopbacks with alternative methods" on page 42 for more information.

2. Check for an extra route. On some operating systems, a default route may have been created and needs to be removed. If problems are encountered with routing after aliasing, remove the alias and add it back using a different netmask.

   Note: Any extra routes should be ignored on Windows 2003.

- Check for an extra route on Windows operating systems with the following command:

  ```
  route print
  ```

  For example:

  a. After route print is entered, a table similar to the following example will be displayed. This example shows finding and removing an extra route to cluster 9.67.133.158 with a default netmask of 255.0.0.0.

  ```
  Active Routes:

  Network Address    Netmask            Gateway Address       Interface          Metric
  0.0.0.0            0.0.0.0            9.67.128.1            9.67.133.67         1
  9.0.0.0            255.0.0.0          9.67.133.158          9.67.133.158        1
  9.67.128.0         255.255.248.0      9.67.133.67           9.67.133.67         1
  9.67.133.67        255.255.255.255    127.0.0.1             127.0.0.1           1
  9.67.133.158       255.255.255.255    127.0.0.1             127.0.0.1           1
  9.255.255.255      255.255.255.255    9.67.133.67           9.67.133.67         1
  127.0.0.0          255.0.0.0          127.0.0.1             127.0.0.1           1
  224.0.0.0          224.0.0.0          9.67.133.158          9.67.133.158        1
  224.0.0.0          224.0.0.0          9.67.133.67           9.67.133.67         1
  255.255.255.255    255.255.255.255    9.67.133.67           9.67.133.67         1
  ```

  b. Find your cluster address under the “Gateway Address” column. If you have an extra route, the cluster address will appear twice. In the example given, the cluster address (9.67.133.158) appears in row 2 and row 8.

  c. Find the network address in each row in which the cluster address appears. You need one of these routes and will need to delete the other route, which is extraneous. The extra route to be deleted is the one whose network address begins with the first digit of the cluster address, followed by three zeroes. In the example shown, the extra route is the one in row two, which has a network address of 9.0.0.0:

  ```
  Network Address    Netmask            Gateway Address       Interface          Metric
  9.0.0.0            255.0.0.0          9.67.133.158          9.67.133.158        1
  ```

- Linux Check for an extra route on all Linux and UNIX systems with the following command:

  ```
  netstat -nr
  ```

3. Delete any extra route. You must delete the extra route. Use the command for your operating system shown below:

   -
   ```
   route delete -net network_address cluster_address
   ```
   -
   ```
   route delete cluster_address cluster_address
   ```
   -
   ```
   route delete network_address cluster_address
   ```

   Note: You must delete the extra route every time you reboot the server.

   To delete the extra route as shown in the ⌂Active Routes⌂ example above, enter:

   ```
   route delete 9.0.0.0 9.67.133.158
   ```

   On Windows 2003, it is not possible to delete routes. Any extra routes should be ignored on Windows 2003. If problems are encountered with routing after aliasing, remove the alias and add it back using a different netmask.

4. Verify server is properly configured. To verify if a back-end server is properly configured, perform the following steps from a different machine on the same subnet when the Load Balancer is not running and cluster is unconfigured:

   a. Issue the arp command. For example:

```
arp -d cluster
```

b. Issue the ping command. For example:

```
ping cluster
```

There should be no response. If there is a response to the ping, ensure that you did not ifconfig the cluster address to the interface. Ensure that no machine has a published arp entry to the cluster address.

c. Ping the back-end server, then immediately issue the arp -a command For example:

```
arp -a
```

In the output from the command, you should see the MAC address of your server. Issue the command:

```
arp -s cluster server_MAC_address
```

d. Ping the cluster. You should get a response. Issue a http, telnet, or other request that is addressed to the cluster that you expect your back-end server to handle. Ensure that it works properly.

e. Issue the arp -d command. For example:

```
arp -d cluster
```

f. Ping the cluster. There should be no response. If there is a response, issue an arp cluster instruction to get the MAC address of the machine that is not configured correctly, and repeat steps 1 through 6.

Aliasing the network interface card or loopback device

To alias the loopback device on servers that are being load-balanced, you must use the operating system's adapter configure commands. For the load-balanced server machines to work, you must set, or preferably alias, the loopback device, which is often called lo0, to the cluster address. By setting or aliasing the loopback device to the cluster address, the load balanced server machines will accept a packet that was addressed to the cluster address.

If you have an operating system that supports network interface aliasing (such as AIX, HP-UX, Linux, Solaris, or Windows systems), you should alias the loopback device to the cluster address. The benefit of using an operating system that supports aliases is that you have the ability to configure the load-balanced server machines to serve multiple cluster addresses.

Note: Linux See "Configuring loopbacks with alternative methods" on page 42. If you have a server with an operating system that does not support aliases you must set the loopback device to the cluster address.

- Use the following commands to alias the network interface and the loopback device (*interface_name*) for AIX systems :
  - **For IPv4 addresses:**
    - AIX 4.3 or earlier:

      ```
      ifconfig lo0 alias cluster_address netmask netmask
      ```

      Note: Use the netmask of the primary adapter
    - AIX 5.x:

      ```
      ifconfig lo0 alias cluster_address netmask 255.255.255.255
      ```
  - **For IPv6 addresses:**

    ```
    ifconfig interface_name inet6 cluster_address/prefix_length alias
    ```

For example, to alias the loopback device on servers that are being load-balanced:

```
ifconfig lo0 inet6 2002:4a::541:56/128 alias
```

- Use these commands to alias the network interface and the loopback device (*interface_name*) for HP-UX systems:
  - **For IPv4 addresses:**

    ```
    ifconfig lo0:1 cluster_address up
    ```

  - **For IPv6 addresses:**

    ```
    ifconfig interface_name:alias inet6 cluster_address up prefix prefix_length
    ```

    For example, to alias the loopback device on servers that are being load-balanced:

    ```
    ifconfig lo0:1 inet6 3ffe:34::24:45 up prefix 128
    ```

  Note: When using bind-specific server applications that bind to a list of IP addresses that do not contain the server's IP, use the `arp publish` command instead of `ifconfig` to dynamically set an IP address on the Load Balancer machine. For example:

  ```
  arp -s <cluster> <Load Balancer's MAC address> pub
  ```

- Linux Use these commands to alias the network interface and the loopback device (*interface_name*) for Linux systems:
  - **For IPv6 or IPv4 addresses:**

    ```
    ip -version addr add cluster_address/prefix_length dev lo
    ```

    For example, to alias the loopback device on servers that are being load-balanced:

    ```
    ip -6 addr add 3ffe:34::24:45/128 dev lo
    ip -4 addr add 12.42.38.125/32 dev lo
    ```

    Note: You can also use the ifconfig command. See below to alias the loopback device using the ifconfig command. Once you issue one of the configuration commands on your machine, it is important to consistently use the same configuration command (ip or ifconfig), or unexpected results can occur.

  - **Using the ifconfig command:**

    ```
    ifconfig lo:1 cluster_address netmask 255.255.255.255 up
    ```

- Use these commands to alias the network interface and the loopback device (*interface_name* for an OS2 system).

  ```
  ifconfig lo cluster_address
  ```

- Use these commands to alias the network interface and the loopback device (*interface_name*) for OS390 systems.
  - In the IP parameter member (file), an Administrator will need to create an entry in the Home address list. For example

    ```
    HOME
    ;Address              Link
    192.168.252.11        tr0
    192.168.100.100       1tr1
    192.168.252.12        loopback
    ```

  - Several addresses can be defined for the loopback.
  - The loopback address of 127.0.0.1 is configured by default.
- The following commands can be used to alias the network interface and the loopback device (*interface_name*) for Solaris systems.
  - **For IPv4 addresses:**
    - For Solaris 7:

```
ifconfig lo0:1 cluster_address 127.0.0.1 up
```

- For Solaris 8, 9, and 10:

```
ifconfig lo0:1 plumb cluster_address netmask netmask up
```

– **For IPv6 addresses:**

```
ifconfig interface_name inet6 addif cluster_address/prefix_length up
```

For example, to alias the loopback device on servers that are being load-balanced:

```
ifconfig lo0 inet6 addif 3ffe:34::24:45/128 up
```

Note: When using bind-specific server applications that bind to a list of IP addresses that do not contain the server's IP, use the `arp publish` command instead of `ifconfig` to dynamically set an IP address on the Load Balancer machine. For example:

```
arp -s <cluster> <Load Balancer's MAC address> pub
```

- Use these commands to alias the network interface and the loopback device for Windows operating systems.

1. Use the `ipconfig /all` command to determine the interface name for the loopback device. This command locates the connection with a description of the Microsoft Loopback Adapter. The following example is the output from the `ipconfig /all` command, where the Microsoft Loopback Adapter is Ethernet adapter Local Area Connection 2, so the connection is Local Area Connection 2:

```
Windows IP Configuration

 Host Name . . . . . . . . . . . . : ndserv10
 Primary Dns Suffix . . . . . . . : rtp.raleigh.ibm.com
 Node Type . . . . . . . . . . . . : Unknown
 IP Routing Enabled. . . . . . . . : No
 WINS Proxy Enabled. . . . . . . . : No
 DNS Suffix Search List. . . . . . : rtp.raleigh.ibm.com

Ethernet adapter Local Area Connection 2:

 Connection-specific DNS Suffix . :
 Description . . . . . . . . . . . : Microsoft Loopback Adapter
 Physical Address. . . . . . . . . : 02-00-4C-4F-4F-50
 DHCP Enabled. . . . . . . . . . . : No
 IP Address. . . . . . . . . . . . : 9.42.92.158
 Subnet Mask . . . . . . . . . . . : 255.255.252.0
 IP Address. . . . . . . . . . . . : 9.42.92.159
 Subnet Mask . . . . . . . . . . . : 255.255.252.0
 IP Address. . . . . . . . . . . . : 2002:92a:8f7a:162:9:42:92:160
 IP Address. . . . . . . . . . . . : 2002:92a:8f7a:162:9:42:92:159
 IP Address. . . . . . . . . . . . : fe80::4cff:fe4f:4f50%4
 Default Gateway . . . . . . . . . :
 DNS Servers . . . . . . . . . . . : 127.0.0.1
                                     fec0:0:0:ffff::1%1
                                     fec0:0:0:ffff::2%1
                                     fec0:0:0:ffff::3%1
```

2. Add the cluster address to the loopback using the netsh command. For example:

```
netsh interface ipv6 add address "Local Area Connection 2"
   2002:92a:8f7a:162:9:42:92:161
```

Note: If you are using a high-availability configuration, and the machine is running as the primary machine, do not alias to the loopback device because this scenario prevents traffic to the cluster address from being routed by the Load Balancer machine.

3. Issue the following ipconfig /all command again, and you should see the address added on the loopback adapter. For example, issue the following command:

   ```
   ipconfig /all
   ```

   You should see output that is similar to the following:

   ```
   Ethernet adapter Local Area Connection 2:

    Connection-specific DNS Suffix . :
    Description . . . . . . . . . . . : Microsoft Loopback Adapter
    Physical Address. . . . . . . . . : 02-00-4C-4F-4F-50
    DHCP Enabled. . . . . . . . . . . : No
    IP Address. . . . . . . . . . . . : 9.42.92.158
    Subnet Mask . . . . . . . . . . . : 255.255.252.0
    IP Address. . . . . . . . . . . . : 9.42.92.159
    Subnet Mask . . . . . . . . . . . : 255.255.252.0
    IP Address. . . . . . . . . . . . : 2002:92a:8f7a:162:9:42:92:161
    IP Address. . . . . . . . . . . . : 2002:92a:8f7a:162:9:42:92:160
    IP Address. . . . . . . . . . . . : 2002:92a:8f7a:162:9:42:92:159
    IP Address. . . . . . . . . . . . : fe80::4cff:fe4f:4f50%4
    Default Gateway . . . . . . . . . :
    DNS Servers . . . . . . . . . . . : 127.0.0.1
                                        fec0:0:0:ffff::1%1
                                        fec0:0:0:ffff::2%1
                                        fec0:0:0:ffff::3%1
   ```

4. Enable forwarding for all the interfaces in the machine using the `netsh interface ipv6 show interface` command. Any interfaces listed with a name of Local Area Connection must have IP forwarding enabled. For example:

   ```
   netsh interface ipv6>show interface
   Querying active state...

   Idx  Met  MTU   State         Name
   ---  ---- ----- ------------  -----
   6    2    1280  Disconnected  Teredo Tunneling Pseudo-Interface
   5    0    1500  Connected     Local Area Connection
   4    0    1500  Connected     Local Area Connection 2
   3    1    1280  Connected     6to4 Pseudo-Interface
   2    1    1280  Connected     Automatic Tunneling Pseudo-Interface
   1    0    1500  Connected     Loopback Pseudo-Interface

   netsh interface ipv6>set interface "Local Area Connection"
      forwarding=enabled
   Ok.

   netsh interface ipv6>set interface "Local Area Connection 2"
      forwarding=enabled
   Ok.
   ```

5. Verify that the forward packets for each Local Area Connection is set to ″Yes.″ Use the following commands:

   ```
   netsh interface ipv6>show interface "Local Area Connection"

   netsh interface ipv6>show interface "Local Area Connection 2"
   ```

Configuring loopbacks with alternative methods

Some versions of Linux systems issue ARP responses for any IP address configured on the machine on any interface present on the machine. It may also choose an ARP source IP address for ARP who-has queries based on all IP addresses present on the machine, regardless of the interfaces on which those addresses are configured. This causes all cluster traffic to be directed to a single server in an indeterminate manner.

With Dispatcher's forwarding method, a mechanism must be employed to ensure that cluster-addressed traffic can be accepted by the stacks of the back-end servers.

In most cases, you must alias the cluster address on the loopback; therefore, back-end servers must have the cluster aliased on the loopback. To ensure that Linux systems do not advertise addresses on the loopback, you can use any of these four solutions to make Linux systems compatible.

- Use a kernel that does not advertise the addresses. This is the preferred option, as it does not incur a per-packet overhead and it does not require per-kernel reconfiguration.
  - United Linux 1 / SLES8 with SP2(x86) or SP3 (all other architectures) and higher contains the Julian ARP hidden patch. Ensure that it is always in effect before aliasing the cluster address with the command:

    ```
    # sysctl -w net.ipv4.conf.all.hidden=1 net.ipv4.conf.lo.hidden=1
    ```

    Clusters can then be aliased in the normal way, such as:

    ```
    # ifconfig lo:1 $CLUSTER_ADDRESS netmask 255.255.255.255 up
    ```

  - Use the arp_ignore sysctl available in 2.4.25 and 2.6.5 and higher, but note that distributions sometimes backport features. Ensure that it is enabled before aliasing the cluster addresses with the commands:

    ```
    # sysctl -w net.ipv4.conf.all.arp_ignore=3
    net.ipv4.conf.all.arp_announce=2
    ```

    Clusters must then be aliased with the following command:

    ```
    # ip addr add $CLUSTER_ADDRESS/32 scope host dev lo
    ```

    Note: When using sysctl, ensure that these settings survive reboot by adding the settings to the *install_root*/etc/sysctl.conf file.
- Use IP tables to redirect all incoming cluster traffic to the localhost. If you use this method, do not configure the loopback adapter with an alias. Instead, use the command:

  ```
  # iptables -t nat -A PREROUTING -d $CLUSTER_ADDRESS -j REDIRECT
  ```

  This command causes Linux systems to do destination NAT on each packet, converting the cluster address to the interface address. This method has about a 6.4% connections-per-second throughput penalty. This method works on any supported stock distribution; no kernel module or kernel patch+build+install is needed.
- Apply the noarp module version 1.2.0 or higher. The kernel source must be available and properly configured, and development tools (gcc, gnu make, and so forth) must be available. You must build and install the module every time the kernel is upgraded. It is available at http://www.masarlabs.com/noarp/. Because the kernel code itself is not modified, it is much less intrusive than solution 4, and it is much less prone to error. It also must be configured before any cluster address is aliased on the loopback. For example:

  ```
  # modprobe noarp # noarpctl add $CLUSTER_ADDRESS nic-primary-addr
  ```

  where *nic-primary-addr* is an address in the same subnet as the cluster address. Clusters can then be aliased in the normal way, such as:

  ```
  # ifconfig lo:1 cluster address netmask 255.255.255.255 up
  ```
- Obtain the Julian patch from the following Web site: http://www.ssi.bg/~ja/#hidden. Follow your distribution instructions for patching and compiling a kernel suitable for use with that distribution. After you build, install, and run your kernel with the Julian hidden patch, following the instructions under the first solution listed for enabling the patch.

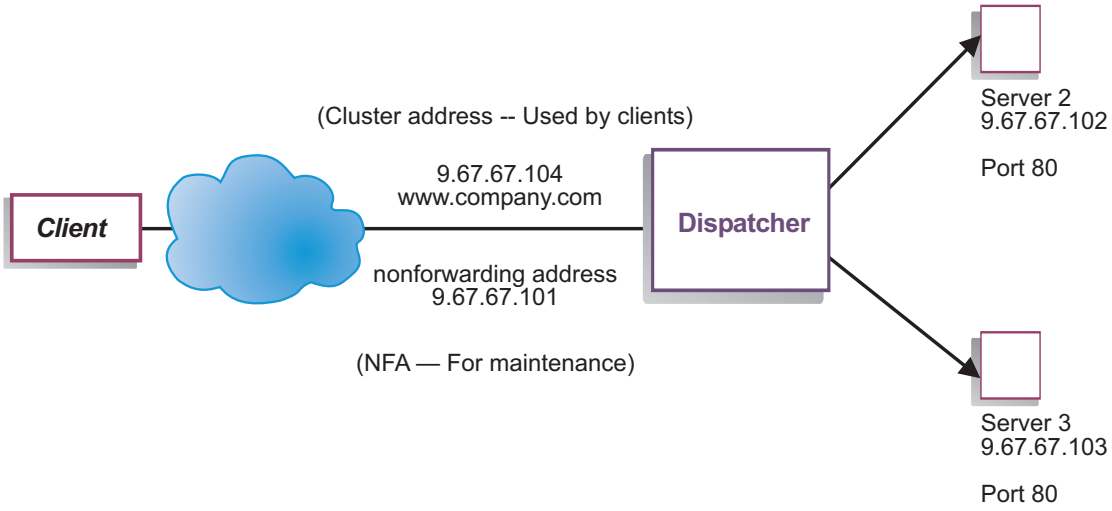
Note: Distribution support implications might exist for running a custom kernel.

Quick start configuration

This quick start example shows how to configure three locally attached workstations to load-balance Web traffic between two Web servers.

For the quick start example, you need three workstations and four IP addresses. One workstation is the Dispatcher machine; the other two workstations are the Web servers. Each Web server requires one IP address. The Dispatcher workstation requires two addresses: the non-forwarding address (NFA), and the cluster address (the address which is load balanced) that you provide to clients to access your Web site.

Note: The NFA is the address that is returned by the hostname command. This address is used for administrative purposes.



For more information on the different ways Load Balancer can be setup, read "Types of cluster, port, and server configurations" on page 8 to help you design your topology.

Use this configuration method for a quick way to establish a connection between servers and the dispatcher machine. This method does not include configuring advisors or tuning performance. For a full configuration, read "Configuring the Load Balancer machine" on page 34 and "Configuring the server machines" on page 37.

1. Prepare your servers.
   a. For this locally attached configuration example, set up your workstations on the same LAN segment. Ensure that network traffic between the three machines does not have to pass through any routers or bridges.
   b. Configure the network adapters of the three workstations. For this example, we will assume you have the following network configuration, and each of the workstations contains only one standard Ethernet network interface card:

Table 9. Sample network configuration

Workstation	Name	IP Address
1	server1.Intersplashx.com	2002:92a:8f7a:162:9:42:92:160
2	server2.Intersplashx.com	2002:92a:8f7a:162:9:42:92:161
3	server3.Intersplashx.com	9.47.47.103
Netmask = 255.255.255.0		

c. 3. Ensure that all the servers can communicate with each other.
   1) Ensure that server1.Intersplashx.com can ping both server2.Intersplashx.com and server3.Intersplashx.com.
   2) Ensure that server2.Intersplashx.com and server3.Intersplashx.com can ping server1.Intersplashx.com.

d. Ensure that content is identical on the two Web servers (Server 2 and Server 3). This can be done by replicating data on both workstations, by using a shared file system such as NFS, AFS®, or DFS™, or by any other means appropriate for your site.

e. Ensure that Web servers on server2.Intersplashx.com and server3.Intersplashx.com are operational. Use a Web browser to request pages directly from http://server2.Intersplashx.com and http://server3.Intersplashx.com.

f. Obtain another valid IP address for this LAN segment. This is the address you will provide to clients who wish to access your site. For this example we will use:

```
Name= www.Intersplashx.com
IP=9.47.47.104
```

g. Configure the two Web server workstations to accept traffic for www.Intersplashx.com. Add an alias for www.Intersplashx.com to the loopback interface on server2.Intersplashx.com and server3.Intersplashx.com:

   -
     ```
     ifconfig lo0 alias www.Intersplashx.com netmask 255.255.255.255
     ```
   -
     ```
     ifconfig lo0:1 plumb www.Intersplashx.com netmask 255.255.255.0 up
     ```
   - For other operating systems see your operating system's instructions in "Aliasing the network interface card or loopback device" on page 39.

h. Delete any extra route that may have been created as a result of aliasing the loopback interface. See Step 2 in Configuring the server machines.

i.

2. Configure Load Balancer using the command line, the GUI, or the configuration wizard.

   - **Configuring with the command line:**

     a. Start the dsserver on Dispatcher:

        – Linux Run the following command as root user:

          ```
          dsserver
          ```

        – dsserver runs as a service that starts automatically

     b. Start the executor function of Dispatcher. Enter the command

        ```
        dscontrol executor start
        ```

     c. Add the cluster address to the Dispatcher configuration:

        ```
        dscontrol cluster add www.Intersplashx.com
        ```

d. Add the HTTP protocol port to the Dispatcher configuration:

```
dscontrol port add www.Intersplashx.com@80
```

e. Add each of the Web servers to the Dispatcher configuration:

```
dscontrol server add www.Intersplashx.com@80@server2.Intersplashx.com
dscontrol server add www.Intersplashx.com@80@server3.Intersplashx.com
```

f. Start the manager function of Dispatcher:

```
dscontrol manager start
```

Dispatcher will now do load balancing based on server performance.

g. Start the advisor function of Dispatcher:

```
dscontrol advisor start http 80
```

Dispatcher will now make sure that client requests are not sent to a failed Web server.

- **Configuring with the configuration wizard:**
  a. Start the dsserver on Dispatcher:
     - Linux Run the following command as root user: `dsserver`
     - dsserver runs as a service that starts automatically
  b. Start the wizard function of Dispatcher:

     ```
     dswizard
     ```

     The wizard guides you step-by-step through the process of creating a basic configuration for the Dispatcher component. It asks questions about your network and guides you through the setup of a cluster for Dispatcher to load balance the traffic for a group of servers. The configuration wizard contains the following panels:
     - Introduction to the wizard
     - What is going to happen
     - Preparing for the setup
     - Defining a cluster
     - Adding a port
     - Adding a server
     - Starting an advisor
     - Server machine setup
- **Configuring with the GUI:**
  - Linux At a command prompt, enter the following:

    ```
    lbadmin
    ```
  - Click **Start** > **Programs** > **IBM WebSphere** > **Edge Components** > **IBM Load Balancer** > **Load Balancer**.

3. Test your configuration.
   a. From a Web browser, go to location http://www.Intersplashx.com. If a page is displayed, the configuration is working.
   b. Reload the page in the Web browser.
   c. Look at the results of the following command:

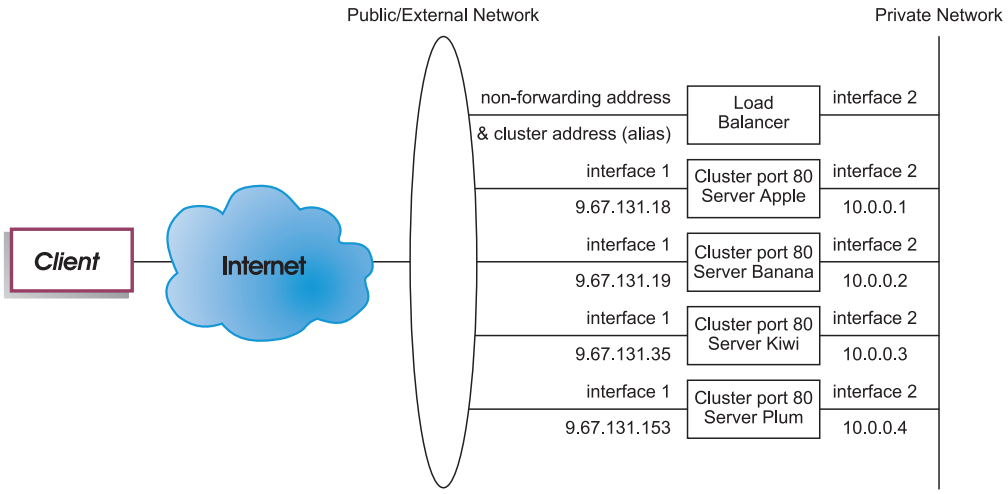
      ```
      dscontrol server report www.Intersplashx.com@80@
      ```

      The total connections column of the two servers should add up to "2."

Load balancing a private network

You can set up Dispatcher and the TCP server machines using a private network. This configuration can reduce the contention on the public or external network that can affect performance.

To create a private network, each machine must have at least two LAN cards, with one of the cards connected to the private network. You must also configure the second LAN card on a different subnet.



When you configure a private network, the Load Balancer machine will then send the client requests to the TCP server machines through this network.

For AIX systems, this configuration can also take advantage of the fast speeds of the SP™ High Performance Switch if you are running Dispatcher and the TCP server machines on nodes in an SP Frame.

The servers added using the `dscontrol server add` command must be added using the private network addresses. For example, a sample command could be coded as:

```
dscontrol server add cluster_address@80@10.0.0.1
```

not as

```
dscontrol server add cluster_address@80@9.67.131.18
```

Chapter 4. Administering Load Balancer

This section focuses on administering production environments and realistic test environments.

improve the robustness of connection protocols and permit Load Balancer to receive ICMP fragmentation messages.

"Configure rules to manage traffic to busy or unavailable servers" on page 95
Use rules-based load balancing to fine tune when and why packets are sent to which servers. Load Balancer reviews any rules you add from first priority to last priority, stopping on the first rule that it finds to be true, then load balancing the traffic between any servers associated with the rule. It already balances the load based on the destination and port, but using rules expands your ability to distribute connections.

"Sample scripts to generate alerts and record server failure" on page 96
Load Balancer provides user exits that trigger scripts that you can customize. You can create the scripts to perform automated actions, such as alerting an Administrator when servers are marked down by the manager or simply record the event of the failure.

Enabling advisors to manage load balancing

Advisors are software agents that work within Load Balancer to provide information about the load on a given server. A different advisor exists for each standard protocol (HTTP, SSL, and others). Periodically, the Load Balancer base code performs an advisor cycle, during which it individually evaluates the status of all servers in its configuration.

Advisors are agents within Load Balancer. Their purpose is to assess the health and loading of server machines. They do this with a proactive client-like exchange with the servers. Advisors can be considered as lightweight clients of the application servers.

By writing your own advisors for the Load Balancer, you can customize how your server machines' load is determined.

For more information on how advisors work, read "Advisors" on page 52.

When using IPv6 protocols: If you are using IPv6 protocol on your machine and want to use advisors, you must modify the protocol file. To enable IPv6, insert the following line in the protocol file:

```
ipv6-icmp 58 IPv6-ICMP # IPv6 interface control message protocol
```

The protocol file is in the following directory:

- Linux
  /etc/protocols
- C:\windows\system32\drivers\etc\

The product provides several protocol-specific advisors for the most popular protocols. However, it does not make sense to use all of the provided advisors with Load Balancer. Load Balancer also supports the concept of a "custom advisor" that allows users to write their own advisors.

Limitation on using advisors with bind-specific server applications:

- In order to use advisors on bind specific servers, start two instances of the server: One instance to bind on the *cluster@port* and the other instance to bind on the *server@port*. To determine if the server is bind specific, issue the `netstat`

`-an` command and look for the *server@port*. If the server is not bind specific, the result from this command will be 0.0.0.0:80. If the server is bind specific, you will see an address such as 192.168.15.103:80.

- If using `arp publish` instead of the `ifconfig alias` command, Load Balancer will support the use of advisors when load-balancing servers with bind-specific server applications when they are binding to the cluster IP address.

You can start an advisor for a particular port across all clusters (group advisor). Or, you can choose to run different advisors on the same port, but on different clusters (cluster specific advisor).

Note: If Load Balancer is running on a computer with multiple network adapter cards, you cannot force the source IP address of the packet to a specific address when you want the advisor traffic to flow over a particular adapter.

1. Start the advisor of your choice. For a list of possible advisors, refer to the list of advisors, or create a custom advisor.
   - **Cluster specific advisor:** To start an advisor on port 80 for clusterA, for example, specify both the cluster and port:

     ```
     dscontrol advisor start ADV_name clusterA@80
     ```

     This command will start an advisor on port 80 for clusterA. This advisor will advise on all servers attached to port 80 for clusterA.
   - **Group advisor:** To start an advisor on port 80 for all other clusters, simply specify the port:

     ```
     dscontrol advisor start ADV_name 80
     ```

     This command will start the advisor on port 80 for all clusters and sites that do not currently have a cluster or site specific advisor. Your advisor will advise on all servers attached to port 80.

   a. Optional: If you are starting the HTTP or HTTPS advisor, you might want to define a unique client URL string to allow the advisor to monitor individual services in the server. For more information on this option, refer to "Getting service-specific advice with the advisor request or response option" on page 56.

   b. Optional: If you are using the self advisor in two-tiered WAN configuration, read rprf_selfadv2tier.dita for more information on how the self advisor garners information.
2. Optional: Set the advisor interval. The advisor interval sets how often an advisor asks for status from the servers on the port it is monitoring and then reports the results to the manager. If the advisor interval is too low, it can mean poor performance as a result of the advisor constantly interrupting the servers. If the advisor interval is too high, it can mean that the manager's decisions about weighting will not be based on accurate, up-to-date information.

   Note: The advisor defaults should work efficiently for the great majority of possible scenarios. Be careful when entering values other than the defaults. For example, to set the interval to 3 seconds for the HTTP advisor for port 80, enter the following command:

   ```
   dscontrol advisor interval http 80 3
   ```

   It does not make sense to specify an advisor interval that is smaller than the manager interval. The default advisor interval is 7 seconds.
3. Optional: Set the advisor report timeout. To make sure that out-of-date information is not used by the manager in its load-balancing decisions, the manager will not use information from the advisor whose time stamp is older than the time set in the advisor report timeout. The advisor report timeout

should be larger than the advisor polling interval. If the timeout is smaller, the manager will ignore reports that logically should be used. By default, advisor reports do not timeout — the default value is unlimited.

For example, to set the advisor report timeout to 30 seconds for the HTTP advisor for port 80, enter the following command:

```
dscontrol advisor timeout http 80 30
```

For more information on setting the advisor report timeout, see "dscontrol advisor" on page 138.

4. Optional: Set the advisors connect and receive timeout values. For Load Balancer, you can set the advisor's timeout values at which it detects a particular port on the server (a service) is failed. The failed-server timeout values (connecttimeout and receivetimeout) determine how long an advisor waits before reporting that either a connect or receive has failed.

   To obtain the fastest failed-server detection, set the advisor connect and receive timeouts to the smallest value (one second), and set the advisor and manager interval time to the smallest value (one second).

   Note: If your environment experiences a moderate to high volume of traffic such that server response time increases, be careful not to set the connecttimeout and receivetimeout values too small, or the advisor may prematurely mark a busy server as failed.

   For example, to set the connecttimeout and receivetimeout to 9 seconds for the HTTP advisor on port 80, type the following command:

   ```
   dscontrol advisor connecttimeout http 80 9

   dscontrol advisor receivetimeout http 80 9
   ```

   The default for connect and receive timeout is 3 times the value specified for the advisor interval time.

5. Optional: Set the advisor retry value. Advisors have the ability to retry a connection before marking a server down. The advisor will not mark a server down until the server query has failed the number of retries plus 1. The retry value should be no larger than 3.

   The following command sets a retry value of 2 for the LDAP advisor on port 389:

   ```
   dscontrol advisor retry ldap 389 2
   ```

Advisors

Advisors periodically open a TCP connection with each server and send a request message to the server. The content of the message is specific to the protocol running on the server. For example, the HTTP advisor sends an HTTP "HEAD" request to the server.

Advisors then listen for a response from the server. After getting the response, the advisor makes an assessment of the server. To calculate this "load" value, most advisors measure the time for the server to respond, and then use this value (in milliseconds) as the load.

Advisors then report the load value to the manager function, where it appears in the manager report in the "Port" column. The manager then calculates aggregate weight values from all its sources, per its proportions, and sets these weight values

into the executor function. The Executor will then use these weights for load balancing new incoming client connections.

If the advisor determines that a server is alive and functioning properly, it will report a positive, non-zero load number to the Manager. If the advisor determines that a server is not active, it will return a special load value of negative one (-1). The Manager and the Executor will not forward any further connections to that server until that server has come back up.

Note: Before sending the initial request message, the advisor will ping the server. This is intended to provide quick status to determine if the machine is online. After the server responds to the ping, no more pings are sent. To disable the pings, add -DLB_ADV_NB_PING to the Load Balancer start script file.

List of advisors

Advisors are agents within Load Balancer. Their purpose is to assess the health and loading of server machines. This list of advisors are already provided with Load Balancer, but you can also write a custom advisor to suit specific needs.

Table 10. List of advisors

Advisor Name	Description
connect	The connect advisor does not exchange any protocol-specific data with the server. It simply measures the time it takes to open and close a TCP connection with the server. This advisor is useful for server applications which use TCP, but with a higher-level protocol for which an IBM-supplied or custom advisor is not available.
Custom advisors	Dispatcher provides the ability for a customer to write a custom (customizable) advisor. This enables support for proprietary protocols (on top of TCP) for which IBM has not developed a specific advisor. For more information, see “Creating a custom advisor” on page 61.
db2	The DB2 advisor works in conjunction with the DB2 servers. Dispatcher has the built in capability of checking the health of DB2 servers without the need for customers to write their own custom advisors. The DB2 advisor communicates with the DB2 connection port only, not the Java connection port.
dns	The DNS advisor opens a connection, sends a pointer query for DNS, waits for a response, closes the connection and returns the elapsed time as a load.
ftp	The FTP advisor opens a connection, sends a SYST request, waits for a response, closes the connection, and returns the elapsed time as a load.

Table 10. List of advisors (continued)

Advisor Name	Description
http	The HTTP advisor opens a connection, sends a HEAD request by default, waits for a response connection, and returns the elapsed time as a load. See “Getting service-specific advice with the advisor request or response option” on page 56for more information on how to change the type of request sent by the HTTP advisor.
https	The HTTPS advisor is a ⌂heavyweight⌂ advisor for SSL connections. It performs a full SSL socket connection with the server. The HTTPS advisor opens an SSL connection, sends an HTTPS request, waits for a response, closes the connnection, and returns the elapsed time as a load. (See also the SSL advisor, which is a ⌂lightweight⌂ advisor for SSL connections.) **Note:** The HTTPS advisor has no dependency upon server key or certificate content, but they must not be expired.
imap	The IMAP advisor opens a connection, waits for an initial message from the server, sends a quit command, closes the connection, and returns the elapsed time as a load.
ldap	The LDAP advisor opens a connection, sends an anonymous BIND request, waits for a response, closes the connection, and returns the elapsed time as a load.
ldapuri	**Note:** The LDAP URI advisor allows you better gauge LDAP availability by processing a complete request to the LDAP server. The advisor: 1. Opens a connection. 2. Sends a BIND request, which is based on the advisorrequest field that you define on the server object. 3. Waits for a response. 4. Closes the connection. 5. Returns the elapsed time as a load. Read “Configuring the LDAP URI advisor” on page 57 for more information on configuring this advisor.
nntp	The NNTP advisor opens a connection, waits for an initial message from the server, sends a quit command, closes the connection, and returns the elapsed time as a load.

Table 10. List of advisors (continued)

Advisor Name	Description
ping	The ping advisor does not open a TCP connection with the servers, but instead reports whether the server responds to a ping. While the ping advisor may be used on any port, it is also designed for configurations using the wildcard port, over which multiple protocol traffic may be flowing. It is also useful for configurations using non-TCP protocols with their servers.
pop3	The POP3 advisor opens a connection, waits for an initial message from the server, sends a quit command, closes the connection, and returns the elapsed time as a load.
reach	The reach advisor pings its target machines. This advisor is also designed for the Dispatcher's high availability components to determine reachability of its reach targets. Its results flow to high availability component and do not appear in the manager report. Unlike the other advisors, the reach advisor starts automatically by the manager function of the Dispatcher component.
sip	The SIP advisor opens a connection, sends an OPTIONS request, waits for a response, closes the connection, and returns the elapsed time as a load. The SIP advisor that is supported runs on TCP only and requires an application to be installed on a server that responds to an OPTIONS request.
smtp	The SMTP advisor opens a connection, waits for an initial message from the server, sends a quit, closes the connection, and returns the elapsed time as a load.
ssl	The SSL advisor is a ⌂lightweight⌂ advisor for SSL connections. It does not establish a full SSL socket connection with the server. The SSL advisor opens a connection, sends an SSL CLIENT_HELLO request, waits for a response, closes the connection, and returns the elapsed time as a load. (See also the HTTPS advisor, which is a ⌂heavyweight⌂ advisor for SSL connections.) **Note:** The SSL advisor has no dependency upon key management or certificates.
ssl2http	The ssl2http advisor starts and advises on the servers listed under port 443, but the advisor will open a socket to the "mapport⌂ for HTTP requests.

Table 10. List of advisors (continued)

Advisor Name	Description
self	The self advisor collects load status information on back-end servers. You can use the self advisor when using Dispatcher in a two–tiered configuration, where the Dispatcher furnishes information from the self advisor to the top-tiered Load Balancer. The self advisor specifically measures the connections per second rate on back-end servers of the Dispatcher at the executor level. See rprf_selfadv2tier.dita for more information.
telnet	The Telnet advisor opens a connection, waits for an initial message from the server, closes the connection, and returns the elapsed time as a load.
was	The WAS (WebSphere Application Server) advisor works in conjunction with the WebSphere Application servers. Customizable sample files for this advisor are provided in the installation directory. For more information, see "Example: Implementing the WAS advisor" on page 79.
wlm	The WLM (Workload Manager) advisor is designed to work in conjunction with servers on OS/390 mainframes running the MVS™ Workload Manager (WLM) component. For more information, see "The Workload Management Advisor" on page 61.

Getting service-specific advice with the advisor request or response option

After you have started an HTTP or HTTPS advisor, you can define a unique client HTTP URL string, specific for the service that you want to query on the server. This allows the advisor to assess the health of the individual services within a server.

For each defined logical server under the HTTP port you can specify a unique client HTTP URL string, specific for the service that you want to query on the server. The HTTP or HTTPS advisor uses the **advisorrequest** string to query the health of the servers. The default value is "HEAD / HTTP/1.0."

The **advisorresponse** string is the response that the advisor scans for in the HTTP response. The advisor uses the **advisorresponse** string to compare to the real response that is received from the server. The default value is null.

- Issue the `server set` command with the advisorrequest and advisorresponse parameters.
  - When issuing the command from the **dscontrol>>** shell prompt, you must place quotes around the string if a blank is contained within the string. For example:

```
server set cluster@port@server advisorrequest "head / http/1.0"

server set cluster@port@server advisorresponse "HTTP 200 OK"
```

– When issuing the dscontrol command from the operating system prompt, you must precede the text with ⌂\⌂ and follow the text with \"". For example:

```
dscontrol server set cluster@port@server advisorrequest "\"head / http/1.0\""

dscontrol server set cluster@port@server advisorresponse "\"HTTP 200 OK\""
```

When you create the request that the HTTP or HTTPS advisor sends to back-end servers to see if they are functioning, you type the start of the HTTP request and Load Balancer completes the end of the request with the following:

```
\r\nAccept:
*/*\r\nUser-Agent:IBM_Network_Dispatcher_HTTP_Advisor\r\n\r\n
```

- Optional: If you want to add other HTTP header fields before Load Balancer appends this string to the end of the request, you can do so by including your own \r\n string in the request. The following is an example of what you might type to add the HTTP host header field to your request:

```
GET /pub/WWW/TheProject.html HTTP/1.0 \r\nHost: www.w3.org
```

Note: After starting an HTTP or HTTPS advisor for a specified HTTP port number, the advisor request and response value is enabled for servers under that HTTP port. See "dscontrol server" on page 158 for more information.

Configuring the LDAP URI advisor

The LDAP URI advisor allows you better gauge Lightweight Directory Access Protocol (LDAP) availability by processing a complete request to the LDAP server. The LDAP URI advisor opens a connection to the LDAP serve and sends a BIND request that is based on the advisorrequest field that you define on the server object. The advisor then waits for a response from the LDAP server and returns the elapsed time as a load.

In situations in which you cannot perform an anonymous bind request to an LDAP server, you can use the LDAP URI advisor to bind with an LDAP server that requires a user name and password. The LDAP URI advisor might provide a more precise measurement of workload, since the LDAP server will be required to process a complete request rather perform only an anonymous bind.

1. Set the advisorrequest field for the server that will use the LDAP URI advisor.
   a. Set the advisorrequest field on the server object with the dscontrol server command. Use the following guidelines for setting the advisorrequest field:
      - Set the advisorrequest field to an LDAP:// URL request that is compliant with the RFC2255 - The LDAP URL Format.
      - Use the bindname extension to perform a bind request that is not anonymous.
      - Load Balancer extends the LDAP URL base with the bindpass extension, allowing you to supply the password for the LDAP server on the URL line. This password must be provided as an optional extension to preserve the portability of the URL.

      For example:

```
dscontrol server set cluster@server@port advisorrequest "ldap://ldap1.mycompany.com:389/ou=
```

Note: Be aware of the ?!bindpass=MYPASS extension that is used above. Replace MYPASS with the password that is used to authenticate the LDAP request.

b. Optional: Set the advisorresponse field on the server object. If you set this field, you must set the value to a substring that is expected to be present in the response from the LDAP server.

2. Start the LDAP URI advisor. To start the LDAP URI advisor, use the dscontrol advisor command:

```
dscontrol advisor start ldapuri cluster@port
```

Note: Verify that you are using the LDAP URI advisor, and not the LDAP advisor. The LDAP advisor only supports anonymous bind requests to LDAP servers.

Getting advice with Metric Server

Metric Server provides server load information to the Load Balancer in the form of system-specific metrics, reporting on the health of the servers.

The Metric Server agent must be installed and running on all servers that are being load balanced.

If you are using IPv6 protocol on your machine and want to use Metric Server, you must check to see if protocol 58 is defined to be ICMPv6 in the **protocol** file.

When using IPv6 protocols: If you are using IPv6 protocol on your machine and want to use advisors, you must modify the protocol file. To enable IPv6, insert the following line in the protocol file:

```
ipv6-icmp 58 IPv6-ICMP # IPv6 interface control message protocol
```

The protocol file is in the following directory:

- Linux
  /etc/protocols
- C:\windows\system32\drivers\etc\

Metric Server Restriction: Like the Metric Server agent, the WLM agent reports on server systems as a whole, rather than on individual protocol-specific server daemons. Metric Server and WLM place their results into the system column of the manager report. As a consequence, running both the WLM advisor and Metric Server at the same time is not supported.

The Load Balancer manager queries the Metric Server agent residing on each of the servers, assigning weights to the load balancing process using the metrics gathered from the agents. The results are also placed into the manager report.

Note: When two or more metrics are gathered and normalized for each server into a single system load value, rounding errors may occur.

1. Configure Metric Server on the Load Balancer machine.
   a. Start **dsserver**. Start the executor, and add clusters, ports and servers to your configuration.
   b. Start the manager. Issue the command:

```
dscontrol manager start manager.log port
```

where *port* is the RMI port chosen for all the Metric Server agents to run on. The default RMI port that is set in the metricserver.cmd file is 10004.

c. Add the system metric script to the cluster. Issue the command:

```
dscontrol metric add cluster@systemMetric
```

systemMetric is the name of the script (residing on the back-end server) which should run on each of the servers in the configuration under the specified cluster. Two scripts are provided for the customer - cpuload and memload - or you can create custom system metric scripts.

The script contains a command which should return a numeric value in the range of 0-100 or a value of -1 if the server is down. This numeric value should represent a load measurement, not an availability value.

Note: If the name of your System Metric script has an extension other than ⌂.exe⌂, you must specify the full name of the file (for example, ⌂mysystemscript.bat⌂). This is due to a Java limitation.

d. Add to the configuration only servers that contain a Metric Server agent running on the port specified in the metricserver.cmd file. The port should match the port value specified in the manager start command.

Note: To ensure security:

- On the Load Balancer machine, create a key file (using the lbkeys create command).
- On the back-end server machine, copy the resulting key file, for the component you are using, to the *install_root*/admin/keys directory. Verify that the key file's permissions enable the file to be readable by the root.

2. Configure Metric Server on the server machines.

a. Install the Metric Server package from the Load Balancer installation files.

b. Check the metric server script in the *install_root*/ms/bin directory to verify that the desired RMI port is being used. The default RMI port is 10004.

Note: The RMI port value specified must be the same value as the RMI port value that was specified in the manager start command in Step 1b.

c. Optional: You can write their own customized metric script files which define the command that the Metric Server will issue on the server machines. Ensure that any custom scripts are executable and located in the *install_root*/ms/script directory. Custom scripts must return a numeric load value in the range of 0-100.

Note: A custom metric script must be a valid program or script with a ⌂.bat⌂ or ⌂.cmd⌂ extension.

Linux Specifically, for Linux and other UNIX-based systems, scripts must begin with the shell declaration, otherwise they may not properly run.

The following two scripts are provided for the customer in the *install_root*/ms/script directory:

- **cpuload:** returns the percentage of cpu in use ranging from 0-100
- **memload:** returns the percentage of memory in use ranging from 0-100.

d. Start the metric server agent. On a command line of each server machine where Metric Server resides, type

```
metricserver start
```

Click **Start** > **Control Panel** > **Administrative Tools** > **Services**. Right-click **IBM Metric Server (ULB)** and select **Start**.

e. Optional: Stop the metric server agent.

Linux To stop the Metric Server agent, issue this command on every server machine where Metric Server resides:

```
metricserver stop
```

Click **Start** > **Control Panel** > **Administrative Tools** > **Services**. Right-click **IBM Metric Server (ULB)** and select **Stop**.

3. Optional: Change the log level in the Metric Server startup script. You can specify a log level range of 0 through 5, similar to the log level range in Load Balancer logs. This will generate an agent log in the *install_root*/ms/logs directory.
4. Optional: To have Metric Server run on an address other than the local host, you need to edit the metricserver file on the load balanced server machine.

   Note: When gathering metrics across different domains, you must explicitly set the java.rmi.server.hostname in the server script (dsserver, etc) to the fully qualified domain name (FQDN) of the machine that is requesting the metrics. This is necessary because InetAddress.getLocalHost.getHostName() might not return the FQDN.

   a. After the occurrence of ⌂java⌂ in the metricserver file, insert the following:

   ```
   -Djava.rmi.server.hostname=OTHER_ADDRESS
   ```

   b. Before the ⌂if⌂ statements in the metricserver file, add the following line:

   ```
   hostname OTHER_ADDRESS
   ```

   c. You will also need to alias the OTHER_ADDRESS on the Microsoft stack of the Metric Server machine. For example:

   ```
   call netsh interface ip add address "Local Area Connection"
     addr=9.37.51.28 mask=255.255.240.0
   ```

5. Optional: Configure Metric Server for IPv4 only or IPv6 only. In a Load Balancer configuration that supports both IPv4 and IPv6 clusters, servers that run the Metric Server function can be configured as an IPv4 server only or as an IPv6 server only, but not both. To force Metric Server to use a particular IP protocol, specify the Java property java.rmi.server.hostname in the metricserver script.

   Note: The host name specified in the Java property must be the physical IP address of the Metric Server.

   - Linux For Metric Server to communicate over the IPv6 address 2002:92a:8f7a:162:9:42:92:67, specify the Java property after $LB_CLASSPATH in the metricserver startup script, in the *install_root*/bin directory, as follows:

     ```
     install_root/java/jre/bin/java ..... $ULB_CLASSPATH
     -Djava.rmi.server.hostname=2002:92a:8f7a:162:9:42:92:67
     com.ibm.internet.nd.sma.SMA_Agent $LB_RMIPORT $LOG_LEVEL $LOG_SIZE $LOG_DIRECTORY $KEYS_DIRECTO
     $SCRIPT_DIRECTORY &
     ```

   - For Metric Server to communicate over the IPv6 address 2002:92a:8f7a:162:9:42:92:67, you must edit the metricserver.cmd file, in the *install_root*/bin directory, as follows:

     ```
     start
     /min /wait %IBMULBPATH%\java\jre\bin\java
     -Djava.rmi.server.hostname=2002:92a:8f7a:162:9:42:92:67
     -Djava.net.preferIPv4Stack=false
     -Djava.net.preferIPv6Stack=true -Xrs -cp
     %LB_CLASSPATH% com.ibm.internet.nd.sma.SMA_Agent
     %RMI_PORT% %LOG_LEVEL% %LOG_SIZE% %LOG_DIRECTORY% %KEYS_DIRECTORY%
     %SCRIPT_DIRECTORY%
     goto done
     ```

```
:stop
%IBMLBPATH%\java\jre\bin\java
-Djava.rmi.server.hostname=2002:92a:8f7a:162:9:42:92:67
-Djava.net.preferIPv4Stack=false
-Djava.net.preferIPv6Stack=true -cp %LB_CLASSPATH% com.ibm.internet.nd.sma.SMA_AgentStop %RM
:done
```

The Workload Management Advisor

WLM is code that runs on MVS mainframes. It can be queried to ask about the load on the MVS machine. When MVS Workload Management has been configured on your OS/390 system, Dispatcher can accept capacity information from WLM and use it in the load balancing process.

Using the WLM advisor, Dispatcher will periodically open connections through the WLM port on each server in the Dispatcher host table and accept the capacity integers returned. Because these integers represent the amount of capacity that is still available and Dispatcher expects values representing the loads on each machine, the capacity integers are inverted by the advisor and normalized into load values (that is, a large capacity integer but a small load value both represent a healthier server). The resulting loads are placed into the System column of the manager report.

There are several important differences between the WLM advisor and other Dispatcher advisors:

- Other advisors open connections to the servers using the same port on which flows normal client traffic. The WLM advisor opens connections to the servers using a port different from normal traffic. The WLM agent on each server machine must be configured to listen on the same port on which the Dispatcher WLM Advisor is started. The default WLM port is 10007.
- Other advisors only assess those servers defined in the Dispatcher cluster@port@server configuration for which the server's port matches the advisor's port. The WLM advisor advises upon every server in the Dispatcher configuration (regardless of the cluster@port). Therefore you must not define any non-WLM servers when using the WLM advisor.
- Other advisors place their load information into the manager report under its "Port" column. The WLM advisor places its load information into the manager report under its system column.
- It is possible to use both protocol-specific advisors along with the WLM advisor. The protocol-specific advisors will poll the servers on their normal traffic ports, and the WLM advisor will poll the system load using the WLM port.

Metric Server Restriction: Like the Metric Server agent, the WLM agent reports on server systems as a whole, rather than on individual protocol-specific server daemons. Metric Server and WLM place their results into the system column of the manager report. As a consequence, running both the WLM advisor and Metric Server at the same time is not supported.

Creating a custom advisor

A custom advisor is a small piece of Java code, provided as a class file, that is called by the Load Balancer base code to determine the load on a server. The base code provides all necessary administrative services, including starting and stopping an instance of the custom advisor, providing status and reports, recording history information in a log file, and reporting advisor results to the manager component.

Custom advisors are called after native, or standard, advisors have been searched. If the Load Balancer does not find a specified advisor among the list of standard advisors, it consults the list of custom advisors. When the Load Balancer base code calls a custom advisor, the following steps happen:

1. The Load Balancer base code opens a connection with the server machine.
2. If the socket opens, the base code calls the specified advisor's GetLoad function.
3. The advisor's GetLoad function performs the steps that the user has defined for evaluating the server's status, including waiting for a response from the server. The function terminates execution when the response is received.
4. The Load Balancer base code closes the socket with the server and reports the load information to the manager. Depending on whether the custom advisor operates in normal mode or in replace mode, the base code sometimes does additional calculations after the GetLoad function terminates.

Custom advisors can be designed to interact with the Load Balancer in either normal mode or replace mode. The choice for the mode of operation is specified in the custom advisor file as a parameter in the constructor method. (Each advisor operates in only one of these modes, based on its design.)

- **Normal mode:** the custom advisor exchanges data with the server, and the base advisor code times the exchange and calculates the load value. The base code then reports this load value to the manager. The custom advisor returns the value zero to indicate success, or negative one to indicate an error.

  To specify normal mode, set the replace flag in the constructor to false.
- **Replace mode:** the base code does not perform any timing measurements. The custom advisor code performs whatever operations are specified, based on its unique requirements, and then returns an actual load number. The base code accepts the load number and reports it, unaltered, to the manager. For best results, normalize your load numbers between 10 and 1000, with 10 representing a fast server and 1000 representing a slow server.

  To specify replace mode, set the replace flag in the constructor to true.

Like all advisors, a custom advisor extends the functionality of the advisor base class, which is called ADV_Base. The advisor base performs most of the advisor's functions, such as reporting loads back to the manager for use in the manager's weight algorithm. The advisor base also performs socket connect and close operations and provides send and receive methods for use by the advisor. The advisor is used only for sending and receiving data on the specified port for the server that is being investigated. The TCP methods provided within the advisor base are timed to calculate load. A flag within the constructor of the advisor base overwrites the existing load with the new load returned from the advisor, if desired.

Note: Based on a value set in the constructor, the advisor base supplies the load to the weight algorithm at specified intervals. If the advisor has not completed processing and cannot return a valid load, the advisor base uses the previously reported load.

1. Name your advisor. Custom advisor file names must follow the form ADV_*name*.java, where *name* is the name that you choose for your advisor.

   Note:

   - You must use the ADV_ prefix for the advisor name.

- You must name the custom advisor using lower-case alphabetic characters to eliminate case sensitivity when an operator types commands on a command line.
- The custom advisor class must be located within the *install_root*/lib/CustomAdvisors subdirectory.
- According to Java conventions, the name of the class defined within the file must match the name of the file.

2. Write your custom advisor. Read "Custom advisor methods and function calls" on page 64 for a list of methods and function calls to use in your advisor. Be aware that custom advisors need to have all the required routines. Advisors must have the following base class methods:
   - A constructor routine. The constructor calls the base class constructor.
   - An ADV_AdvisorInitialize method. This method provides a way to perform additional steps after the base class completes its initialization.
   - A getLoad routine. The base advisor class performs the socket opening; the getLoad function only needs to issue the appropriate send and receive requests to complete the advising cycle.
3. Compile the advisor.
   - You must write custom advisors in the Java language and compile them with a Java compiler that is at the same level as the Load Balancer code. To check the version of Java on your system, run the following command from the `install_root/java/bin` directory:

     ```
     java -fullversion
     ```

     If the current directory is not part of your path, you will need to specify that Java should be run from the current directory to ensure you are getting the correct version information. In this case, run the following command from the `install_root/java/bin` directory:

     ```
     ./java -fullversion
     ```
   - The following files are referenced during compilation:
     – The custom advisor file.
     – The base classes file, ibmnd.jar, which is found in the *install_root*`/servers/lib` directory.
   - Your classpath environment variable must point to both the custom advisor file and the base classes file during the compilation. A compile command might have the following format, if your advisor is in the current directory:

     install_path`/java/bin/javac -classpath` *install_root*`/servers/lib/ibmlb.jar` *ADV_name*`.java`
   - The output of the compilation is a class file, for example, *ADV_name*.class. Before starting the advisor, copy the class file to the *install_root*`/servers/lib/CustomAdvisors/` directory.

   Note: You can compile custom advisors on one operating system and run on another operating system. For example, you can compile your advisor on a Windows system, copy the resulting class file, in binary format, to a Linux machine, and run the custom advisor there. For AIX, HP-UX, Linux, and Solaris operating systems, the syntax is similar.
4. Run your custom advisor. Custom advisors are called after native, or standard, advisors are searched. If Load Balancer does not find a specified advisor among the list of standard advisors, it consults the list of custom advisors.
   a. If you have not already done so, copy the advisor's class file to the CustomAdvisors subdirectory on the Load Balancer machine. For example,

for a custom advisor named myping, the file path is *install_root*/servers/lib/CustomAdvisors/ADV_myping.class.

b. Configure the Load Balancer, start its manager function, and issue the command to start your custom advisor. The custom advisor is specified by its name, excluding the ADV_ prefix and the file extension:

```
dscontrol advisor start name.ext port
```

The port number specified in the command is the port on which the advisor will open a connection with the target server.

Custom advisor methods and function calls

Use the following advisor methods and function calls in your custom advisors.

Be aware that custom advisors need to have all the required routines. Advisors must have the following base class methods:

- A constructor routine. The constructor calls the base class constructor.
- An ADV_AdvisorInitialize method. This method provides a way to perform additional steps after the base class completes its initialization.
- A getLoad routine. The base advisor class performs the socket opening; the getLoad function only needs to issue the appropriate send and receive requests to complete the advising cycle.

Constructor (provided by advisor base)

```
public <advisor_name> {
  String sName;
  String sVersion;
  int iDefaultPort;
  int iInterval;
  String sDefaultLogFileName;
  boolean replace
)
```

sName
The name of the custom advisor

sVersion
The version of the custom advisor.

iDefaultPort
The port number on which to contact the server if no port number is specified in the call.

iInterval
The interval at which the advisor will query the servers.

sDefaultLogFileName
This parameter is required but not used. The only acceptable value is a null string, ""

replace
Whether or not this advisor functions in replace mode. Possible values are the following:

- *true* – Replace the load calculated by the advisor base code with the value reported by the custom advisor.
- *false* – Add the load value reported by the custom advisor to the load value calculated by the advisor base code.

ADV_AdvisorInitialize() method

```
void ADV_AdvisorInitialize()
```

This method is provided to perform any initialization that might be required for the custom advisor. This method is called after the advisor base module starts. In many cases, including the standard advisors, this method is not used and its code consists of a return statement only. This method can be used to call the "suppressBaseOpeningSocket()" on page 68 method, which is valid only from within this method.

In many cases, including the standard advisors, this method is not used and its code consists of a return statement only. You can use this method to call the suppressBaseOpeningSocket method, which is valid only from within the ADV_AdvisorInitialize method.

ADVLOG() method

The ADVLOG function allows a custom advisor to write a text message to the advisor base log file. The format follows:

```
void ADVLOG (int logLevel, String message)
```

This command has the following parameters:

logLevel
The status level at which the message is written to the log file. The advisor log file is organized in stages; the most urgent messages are given status level 0 and less urgent messages receive higher numbers. The most verbose type of message is given status level 5. These levels are used to control the types of messages that the user receives in real time (The dscontrol command is used to set verbosity). Catastrophic errors should always be logged at level 0.

message
The message to write to the log file. The value for this parameter is a standard Java string.

getAdvisorName function

The getAdvisorName function returns a Java string with the suffix portion of your custom advisor name. For example, for an advisor named ADV_cdload.java, this function returns the value cdload.

This function does not take parameters.

Note: It is not possible for this value to change during one instantiation of an advisor.

caller.getCurrentServerId()

The getCurrentServerId function returns a Java string which is a unique representation for the current server. Typically, this value changes each time you call your custom advisor, because the advisor base code queries all server machines in series.

This function takes no parameters.

caller.getCurrentClusterId()

The getCurrentClusterId function call returns a Java string which is a unique representation for the current cluster. Typically, this value changes each time you call your custom advisor, because the advisor base queries all clusters in series.

This function takes no parameters.

caller.getSocket()

The getSocket function call returns a Java socket which represents the socket opened to the current server for communication.

This function takes no parameters.

caller.getLatestLoad()

The getLatestLoad function allows a custom advisor to obtain the latest load value for a given server object. The load values are maintained in internal tables by the advisor base code and the manager daemon. This function call is useful if you want to make the behavior of one protocol or port dependent on the behavior of another. For example, you might use this function call in a custom advisor that disabled a particular application server if the Telnet server on that same machine was disabled.

The syntax is:

```
int caller.getLatestLoad (String clusterId, int port, String serverId)
```

The three arguments together define one server object.

This command has the following parameters:

clusterId
The cluster identifier of the server object for which to obtain the current load value. This argument must be a Java string.

port
The port number of the server object for which to obtain the current load value.

serverId
The server identifier of the server object for which to obtain the current load value. This argument must be a Java string. The return value is an integer.

- A positive return value represents the actual load value assigned for the object that was queried.
- The value -1 indicates that the server asked about is down.
- The value -2 indicates that the status of the server asked about is unknown.

caller.receive()

The receive function gets information from the socket connection. The syntax is:

```
caller.receive(StringBuffer *response)
```

This command has the following parameters:

response
This is a string buffer into which the retrieved data is placed. Additionally, the function returns an integer value with the following significance:

- 0 indicates data was sent successfully.
- A negative number indicates an error.

caller.send()

The send function uses the established socket connection to send a packet of data to the server, using the specified port. The syntax is as follows:

```
caller.send(String command)
```

This command has the following parameters:

command
This is a string containing the data to send to the server. The function returns an integer value with the following significance:

- 0 indicates data was sent successfully.
- A negative number indicates an error.

getLoad()

```
int getLoad( int iConnectTime; ADV_Thread *caller )
```

This function has the following parameters:

iConnectTime
The length of time, in milliseconds, that it took the connection to complete. This load measurement is performed by the advisor base code and passed to the custom advisor code, which can use or ignore the measurement when returning the load value. If the connection fails, this value is set to -1.

caller
The instance of the advisor base class where advisor base methods are provided.Function calls available to custom advisors: The methods, or functions, described in the following sections can be called from custom advisors. These methods are supported by the advisor base code. Some of these function calls can be made directly, for example, function_name(), but others require the prefix caller. Caller represents the base advisor instance that supports the custom advisor that is being executed.

getAdviseOnPort()

The getAdviseOnPort function returns the port number on which the calling custom advisor is running.

The return value is a Java integer (int), and the function does not take parameters.

Note: It is not possible for this value to change during one instantiation of an advisor.

getAdvisorName()

The getAdvisorName function returns a Java string with the suffix portion of your custom advisor's name. For example, for an advisor named ADV_cdload.java, this function returns the value cdload. This function takes no parameters. Note that it is not possible for this value to change during one instantiation of an advisor.

getInterval()

The getInterval function returns the advisor interval, that is, the number of seconds between advisor cycles. This value is equal to the default value set in the custom advisor's constructor, unless the value has been modified at run time by using the dscontrol command. The return value is a Java integer (int).

The function takes no parameters.

suppressBaseOpeningSocket()

The suppressBaseOpeningSocket function call allows a custom advisor to specify whether the base advisor code opens a TCP socket to the server on the custom advisor's behalf. If your advisor does not use direct communication with the server to determine its status, it might not be necessary to open this socket. This function call can be issued only once, and it must be issued from the "ADV_AdvisorInitialize() method" on page 65 routine.

The function takes no parameters.

Related tasks

"Creating a custom advisor" on page 61
A custom advisor is a small piece of Java code, provided as a class file, that is called by the Load Balancer base code to determine the load on a server. The base code provides all necessary administrative services, including starting and stopping an instance of the custom advisor, providing status and reports, recording history information in a log file, and reporting advisor results to the manager component.

Related reference

"Example: Sample advisor"
This is a sample advisor file called ADV_sample.

Example: Sample advisor

This is a sample advisor file called ADV_sample.

```
/ * *
 * ADV_sample: The Load Balancer HTTP advisor
 *
 *
 * This class defines a sample custom advisor for Load Balancer. Like all
 * advisors, this custom advisor extends the function of the advisor base,
 * called ADV_Base. It is the advisor base that actually performs most of
 * the advisor's functions, such as reporting loads back to the Load Balancer
 * for use in the Load Balancer's weight algorithm. The advisor base also
 * performs socket connect and close operations and provides send and receive
 * methods for use by the advisor. The advisor itself is used only for
 * sending and receiving data to and from the port on the server being
 * advised. The TCP methods within the advisor base are timed to calculate
 * the load. A flag within the constructor in the ADV_base overwrites the
 * existing load with the new load returned from the advisor if desired.
 *
 * Note: Based on a value set in the constructor, the advisor base supplies
 * the load to the weight algorithm at specified intervals. If the actual
 * advisor has not completed so that it can return a valid load, the advisor
 * base uses the previous load.
 *
 * NAMING
 *
 * The naming convention is as follows:
 *
 *  - The file must be located in the following Load Balancer directory:
```

```
*
*     ulb/servers/lib/CustomAdvisors/ (ulb\servers\lib\CustomAdvisors on Windows)
*
*  - The Advisor name must be preceded with "ADV_". The advisor can be
*    started with only the name, however; for instance, the "ADV_sample"
*    advisor can be started with "sample".
*
*  - The advisor name must be in lowercase.
*
* With these rules in mind, therefore, this sample is referred to as:
*
*     <base directory="">/lib/CustomAdvisors/ADV_sample.class
*
*
* Advisors, as with the rest of Load Balancer, must be compiled with the
* prerequisite version of Java. To ensure access to Load Balancer classes, make
* sure that the ibmlb.jar file (located in the lib subdirectory of the base
* directory) is included in the system's CLASSPATH.
*
* Methods provided by ADV_Base:
*
* - ADV_Base (Constructor):
*
*   - Parms
*     - String sName = Name of the advisor
*     - String sVersion = Version of the advisor
*     - int iDefaultPort = Default port number to advise on
*     - int iInterval = Interval on which to advise on the servers
*     - String sDefaultName = Unused. Must be passed in as "".
*     - boolean replace = True - replace the load value being calculated
*                                by the advisor base
*                         False - add to the load value being calculated
*                                 by the advisor base
*   - Return
*     - Constructors do not have return values.
*
* Because the advisor base is thread based, it has several other methods
* available for use by an advisor. These methods can be referenced using
* the CALLER parameter passed in getLoad().
*
* These methods are as follows:
*
* - send - Send a packet of information on the established socket connection
*          to the server on the specified port.
* - Parms
*   - String sDataString - The data to be sent in the form of a string
* - Return
*   - int RC - Whether the data was sucessfully sent or not: zero indicates
*              data was sent; a negative integer indicates an error.
*
* - receive - Receive information from the socket connection.
*   - Parms
*     - StringBuffer sbDataBuffer - The data received during the receive call
*   - Return
*     - int RC - Whether the data was successfully received or not; zero
*                indicates data was sent; a negative integer indicates
*                an error.
*
* If the function provided by the advisor base is not sufficient,
* you can create the appropriate function within the advisor and
* the methods provided by the advisor base will then be ignored.
*
* An important question regarding the load returned is whether to apply
* it to the load being generated within the advisor base,
* or to replace it; there are valid instances of both situations.
*
* This sample is essentially the Load Balancer HTTP advisor. It functions
```

```
 * very simply: a send request--an http head request--is issued. Once a
 * response is received, the getLoad method terminates, flagging the advisor
 * base to stop timing the request. The method is then complete. The
 * information returned is not parsed; the load is based on the time
 * required to perform the send and receive operations.
 */

package CustomAdvisors;
import com.ibm.internet.nd.advisors.*;
  public class ADV_sample extends ADV_Base implements ADV_MethodInterface
  {
    String COPYRIGHT =
              "(C) Copyright IBM Corporation 1997, All Rights Reserved.\n";
    static final String ADV_NAME = "Sample";
    static final int ADV_DEF_ADV_ON_PORT = 80;
    static final int ADV_DEF_INTERVAL = 7;

    // Note: Most server protocols require a carriage return ("\r") and line
    // feed ("\n") at the end of messages. If so, include them in
    // your string here.

    static final String ADV_SEND_REQUEST =
       "HEAD / HTTP/1.0\r\nAccept:  */ *\r\nUser-Agent: " +
       "IBM_Load_Balancer_HTTP_Advisor\r\n\r\n";

    /**
     * Constructor.
     *
     * Parms: None; but the constructor for ADV_Base has several parameters
     *        that must be passed to it.
     *
     */
     public ADV_sample()
     {
       super( ADV_NAME,
          "2.0.0.0-03.27.98",
               ADV_DEF_ADV_ON_PORT,
               ADV_DEF_INTERVAL,
               "", // not used false);
       super.setAdvisor( this );
     }


     /**
      * ADV_AdvisorInitialize
      *
      * Any Advisor-specific initialization that must take place after the
      * advisor base is started. This method is called only once and is
      * typically not used.
      */
      public void ADV_AdvisorInitialize()
      {
        return;
      }

      /**
       * getLoad()
       *
       * This method is called by the advisor base to complete the advisor's
       * operation, based on details specific to the protocol. In this sample
       * advisor, only a single send and receive are necessary; if more complex
       * logic is necessary, multiple sends and receives can be issued. For
       * example, a response might be received and parsed. Based on the
       * information learned thereby, another send and receive could be issued.
       *
       * Parameters:
       *
```

```
     * - iConnectTime - The current load as it refers to the length of time it
     *                  took to complete the connection to the server through
     *                  the specified port.
     *
     * - caller - A reference to the advisor base class where the Load
     *            Balancer-supplied methods are to perform simple TCP requests,
     *            mainly send and receive.
     *
     * Results:
     *
     * - The load - A value, expressed in milliseconds, that can either be added
     *   to the existing load, or that can replace the existing load, as
     *   determined by the constructor's "replace" flag.
     *
     *   The larger the load, the longer it took the server to respond;
     *   therefore, the lower the weight will become within the Load Balancer.
     *
     *   If the value is negative, an error is assumed. An error from an
     *   advisor indicates that the server the advisor is trying to reach is not
     *   accessible and has been identified as being down. Load Balancer will
     *   not attempt to load balance to a server that is down. Load Balancer will
     *   resume load balancing to the server when a positive value is received.
     *
     */
    public int getLoad(int iConnectTime, ADV_Thread caller)
    {
      int iRc;
      int iLoad = ADV_HOST_INACCESSIBLE; // -1

      // Send tcp request iRc = caller.send(ADV_SEND_REQUEST);
      if (iRc >= 0)
      {
          // Perform a receive
          StringBuffer sbReceiveData = new StringBuffer("");
          iRc = caller.receive(sbReceiveData);

       /**
        * In the normal advisor mode ("replace" flag is false), the load
        * returned is either 0 or 1 indicating the server is up or down.
        * If the receive is successful, a load of zero is returned
        * indicating that the load built within the base advisor is to be used.
        *
        * Otherwise ("replace" flag is true), return the desired load value.
        */

        if (iRc >= 0)
        {
           iLoad = 0;
        }
      }
      return iLoad;
    }
} // End - ADV_sample
```

Example: Implementing standard advisors:

The following example demonstrates how to use a standard custom advisor.

This sample source code is similar to the standard Load Balancer HTTP advisor. It functions as follows:

1. A send request, a ″HEAD/HTTP″ command, is issued.
2. A response is received. The information is not parsed, but the response causes the getLoad method to terminate.
3. The getLoad method returns 0 to indicate success or -1 to indicate a failure.

This advisor operates in normal mode, so the load measurement is based on the elapsed time in milliseconds required to perform the socket open, send, receive, and close operations.

```
package CustomAdvisors;
import com.ibm.internet.lb.advisors.*;
public class ADV_sample extends ADV_Base implements ADV_MethodInterface {
  static final String ADV_NAME ="Sample";
  static final int ADV_DEF_ADV_ON_PORT = 80;
  static final int ADV_DEF_INTERVAL = 7;
  static final String ADV_SEND_REQUEST =
    "HEAD / HTTP/1.0\r\nAccept: */*\r\nUser-Agent: " +
    "IBM_Load_Balancer_HTTP_Advisor\r\n\r\n";

  //--------
  // Constructor
    public ADV_sample() {
      super(ADV_NAME, "3.0.0.0-03.31.00",
            ADV_DEF_ADV_ON_PORT, ADV_DEF_INTERVAL, "",
            false);
      super.setAdvisor( this );
   }

  //--------
  // ADV_AdvisorInitialize
    public void ADV_AdvisorInitialize() {
      return;                               // usually an empty routine
    }

  //--------
  // getLoad

    public int getLoad(int iConnectTime, ADV_Thread caller) {
      int iRc;
      int iLoad = ADV_HOST_INACCESSIBLE;          // initialize to inaccessible

      iRc = caller.send(ADV_SEND_REQUEST);        // send the HTTP request to
                                                  // the server
      if (0 <= iRc) {                             // if the send is successful
        StringBuffer sbReceiveData = new StringBuffer("");      // allocate a buffer
                                                                // for the response
      iRc = caller.receive(sbReceiveData);        // receive the result

      // parse the result here if you need to

      if (0 <= iRc) {                             // if the receive is successful
        iLoad = 0;                                // return 0 for success
      }                                           // (advisor's load value is ignored by
    }                                             // base in normal mode)
    return iLoad;
  }
}
```

Example: Implementing a side stream advisor:

The following example demonstrates how a side stream advisor can be implemented. This sample illustrates suppressing the standard socket opened by the advisor base. Instead, this advisor opens a side stream Java socket to query a server. This procedure can be useful for servers that use a different port from normal client traffic to listen for an advisor query.

In this example, a server is listening on port 11999 and when queried returns a load value with a hexadecimal int ″4″. This sample runs in replace mode, that is, the last parameter of the advisor constructor is set to true and the advisor base code uses the returned load value rather than the elapsed time.

Note the call to supressBaseOpeningSocket() in the initialization routine. Suppressing the base socket when no data will be sent is not required. For example, you might want to open the socket to ensure that the advisor can contact the server. Examine the needs of your application carefully before making this choice.

```
package CustomAdvisors;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Date;
import com.ibm.internet.lb.advisors.*;
import com.ibm.internet.lb.common.*;
import com.ibm.internet.lb.server.SRV_ConfigServer;

public class ADV_sidea extends ADV_Base implements ADV_MethodInterface {
  static final String ADV_NAME = "sidea";
  static final int ADV_DEF_ADV_ON_PORT = 12345;
  static final int ADV_DEF_INTERVAL = 7;

  // create an array of bytes with the load request message
  static final byte[] abHealth = {(byte)0x00, (byte)0x00, (byte)0x00,
                                  (byte)0x04};

  public ADV_sidea() {
    super(ADV_NAME, "3.0.0.0-03.31.00", ADV_DEF_ADV_ON_PORT,
          ADV_DEF_INTERVAL, "",
          true);                     // replace mode parameter is true
    super.setAdvisor( this );
  }

//--------
// ADV_AdvisorInitialize
  public void ADV_AdvisorInitialize()
  {
    suppressBaseOpeningSocket();        // tell base code not to open the
                                        // standard socket
    return;
  }

//--------
// getLoad
  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;    // -1
    int iControlPort = 11999;             // port on which to communicate
                                          // with the server
    String sServer = caller.getCurrentServerId(); // address of server to query
    try {
      socket soServer = new Socket(sServer, iControlPort);   // open socket to
                                                             // server
      DataInputStream disServer = new DataInputStream(
                                                  soServer.getInputStream());
      DataOutputStream dosServer = new DataOutputStream(
                                                  soServer.getOutputStream());
      int iRecvTimeout = 10000;           // set timeout (in milliseconds)
                                          // for receiving data
      soServer.setSoTimeout(iRecvTimeout);
      dosServer.writeInt(4);              // send a message to the server
      dosServer.flush();
      iLoad = disServer.readByte();       // receive the response from the server

    } catch (exception e) {
      system.out.println("Caught exception " + e);
```

```
    }
    return iLoad;                          // return the load reported from the server
  }
}
```

Example: Implementing a two-port advisor:

The following example shows how to implement a two-port advisor. This custom advisor sample demonstrates the capability to detect failure for one port of a server based upon both its own status and on the status of a different server daemon that is running on another port on the same server machine.

For example, if the HTTP daemon on port 80 stops responding, you might also want to stop routing traffic to the SSL daemon on port 443.

This advisor is more aggressive than standard advisors, because it considers any server that does not send a response to have stopped functioning, and marks it as down. Standard advisors consider unresponsive servers to be very slow. This advisor marks a server as down for both the HTTP port and the SSL port based on a lack of response from either port.

To use this custom advisor, the administrator starts two instances of the advisor: one on the HTTP port, and one on the SSL port. The advisor instantiates two static global hash tables, one for HTTP and one for SSL. Each advisor tries to communicate with its server daemon and stores the results of this event in its hash table. The value that each advisor returns to the base advisor class depends on both the ability to communicate with its own server daemon and the ability of the partner advisor to communicate with its daemon.

The following custom methods are used.

- ADV_nte() is a simple container object to hold information about a server. These objects are stored in the hash table as table elements. Each object has a time stamp that is used to determine whether the element is current.
- putNte() and getNte() are synchronized methods that ensure that the two advisor instances access the hash table in a controlled fashion.
- getLoadHTTP is a method that queries the responsiveness of an HTTP server. It is a low-level routine and does not gather or use information about SSL.
- getLoadSSL() is a method that queries the responsiveness of an SSL server. It is a low-level routine and does not gather or use information about HTTP.
- getLoad() is the entry point routine for this custom advisor. It can handle both protocols and can store and fetch information from the hash table. This is the routine that links the two ports.

The following error conditions are detected:

- Unresponsive server machine - The base advisor classes periodically send a ping signal to the server address. If the address is not reachable, the base advisor classes marks the server down. Neither of the two instances of the custom advisor is called, and both servers on that machine are marked down.
- One daemon on a server machine becomes unresponsive, but the other is working - When the base code attempts to open a socket with the server, the connection is refused, and the base advisor for this protocol marks the server as down. The custom advisor code for that protocol is not called. Although the custom advisor for the other protocol continues communicating with its server, it

learns from the hash table that the other custom advisor cannot communicate with its server daemon. Therefore, the second protocol's advisor also marks its server as down.

- One daemon does not send a response, but the other daemon does - The custom advisor for the unresponsive protocol detects the failure to communicate, marks the server as down, and stores the data in the hash table. The custom advisor for the other port learns that information from the hash table and marks its server as down.

This sample is written to link ports 80 for HTTP and 443 for SSL, but it can be tailored to any combination of ports:

```
package CustomAdvisors;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Date;
import com.ibm.internet.lb.advisors.*;
import com.ibm.internet.lb.common.*;
import com.ibm.internet.lb.manager.*;
import com.ibm.internet.lb.server.SRV_ConfigServer;

//--------
// Define the table element for the hash tables used in this custom advisor

class ADV_nte implements Cloneable {
  private String sCluster;
  private int iPort;
  private String sServer;
  private int iLoad;
  private Date dTimestamp;

//--------
// constructor

  public ADV_nte(String sClusterIn, int iPortIn, String sServerIn,
                 int iLoadIn) {
    sCluster = sClusterIn;
    iPort = iPortIn;
    sServer = sServerIn;
    iLoad = iLoadIn;
    dTimestamp = new Date();
  }

//--------
// check whether this element is current or expired
  public boolean isCurrent(ADV_twop oThis) {
    boolean bCurrent;
    int iLifetimeMs = 3 * 1000 * oThis.getInterval();     // set lifetime as
                                                          // 3 advisor cycles
    Date dNow = new Date();
    Date dExpires = new Date(dTimestamp.getTime() + iLifetimeMs);

    if (dNow.after(dExpires)) {
      bCurrent = false;
    } else {
      bCurrent = true;
    } return bCurrent;
  }

//--------
// value accessor(s)

 public int getLoadValue() { return iLoad; }
```

```
//--------
// clone (avoids corruption between threads)

 public synchronized Object Clone() {
   try {
     return super.clone();
   } catch (cloneNotSupportedException e) {
     return null;
    }
  }

}

//--------
// define the custom advisor

public class ADV_twop extends ADV_Base
   implements ADV_MethodInterface, ADV_AdvisorVersionInterface {
   static final int ADV_TWOP_PORT_HTTP = 80;
   static final int ADV_TWOP_PORT_SSL = 443;

   //--------
   // define tables to hold port-specific history information

   static HashTable htTwopHTTP = new Hashtable();
   static HashTable htTwopSSL = new Hashtable();
   static final String ADV_TWOP_NAME = "twop";
   static final int ADV_TWOP_DEF_ADV_ON_PORT = 80;
   static final int ADV_TWOP_DEF_INTERVAL = 7;
   static final String ADV_HTTP_REQUEST_STRING =
      "HEAD / HTTP/1.0\r\nAccept: */*\r\nUser-Agent: " +
      "IBM_LB_Custom_Advisor\r\n\r\n";

   //--------
   // create byte array with SSL client hello message

   public static final byte[] abClientHello = {
     (byte)0x80, (byte)0x1c,
     (byte)0x01,                                  // client hello
     (byte)0x03, (byte)0x00,                      // SSL version
     (byte)0x00, (byte)0x03,                      // cipher spec len (bytes)
     (byte)0x00, (byte)0x00,                      // session ID len (bytes)
     (byte)0x00, (byte)0x10,                      // challenge data len (bytes)
     (byte)0x00, (byte)0x00, (byte)0x03,          // cipher spec
     (byte)0x1A, (byte)0xFC, (byte)0xE5, (byte)Ox20,    // challenge data
     (byte)0xFD, (byte)0x3A, (byte)0x3C, (byte)0x18,
     (byte)0xAB, (byte)0x67, (byte)0xB0, (byte)0x52,
     (byte)0xB1, (byte)0x1D, (byte)0x55, (byte)0x44, (byte)0x0D, (byte)0x0A };

  //--------
  // constructor

  public ADV_twop() {
    super(ADV_TWOP_NAME, VERSION, ADV_TWOP_DEF_ADV_ON_PORT,
          ADV_TWOP_DEF_INTERVAL, "",
          false);                          // false = load balancer times the response
    setAdvisor ( this );
  }

  //--------
  // ADV_AdvisorInitialize

  public void ADV_AdvisorInitialize() {
    return; }

  //--------
  // synchronized PUT and GET access routines for the hash tables
```

```
  synchronized ADV_nte getNte(Hashtable ht, String sName, String sHashKey) {
    ADV_nte nte = (ADV_nte)(ht.get(sHashKey));
    if (null != nte) {
      nte = (ADV_nte)nte.clone();
    }
    return nte;
  }
 synchronized void putNte(Hashtable ht, String sName, String sHashKey,
                          ADV_nte nte) { ht.put(sHashKey,nte); return;
}


  //--------
  // getLoadHTTP - determine HTTP load based on server response

  int getLoadHTTP(int iConnectTime, ADV_Thread caller) {
    int iLoad = ADV_HOST_INACCESSIBLE;
    int iRc = caller.send(ADV_HTTP_REQUEST_STRING);   // send request message
                                                      // to server
    if (0 <= iRc) {                                   // did the request return a failure?
      StringBuffer sbReceiveData = new StringBuffer("") // allocate a buffer
                                                        // for the response
      iRc = caller.receive(sbReceiveData);             // get response from server

      if (0 <= iRc) {                               // did the receive return a failure?
        if (0 < sbReceiveData.length()) {           // is data there?
          iLoad = SUCCESS;                          // ignore retrieved data and
                                                    // return success code
      }
    }
  }
  return iLoad;
}


//--------
// getLoadSSL() - determine SSL load based on server response

int getLoadSSL(int iConnectTime, ASV_Thread caller) {
  int iLoad = ADV_HOST_INACCESSIBLE;
  int iRc;

  CMNByteArrayWrapper cbawClientHello = new CMNByteArrayWrapper(
                                                    abClientHello);
  Socket socket = caller.getSocket();

  try {
      socket.getOutputStream().write(abClientHello); // Perform a receive.
      socket.getInputStream().read();                // If receive is successful,
                                                     // return load of 0. We are not
                                                     // concerned with data's contents,
                                                     // and the load is calculated by
                                                     // the ADV_Thread thread.
      iLoad = 0;
  } catch (IOException e) {           // Upon error, iLoad will default to it.
  }
  return iLoad;
}


//--------
// getLoad - merge results from the HTTP and SSL methods

public int getLoad(int iConnectTime, ADV_Thread caller) {
  int iLoadHTTP;
  int iLoadSSL;
```

```
 int iLoad;
 int iRc;

 String sCluster = caller.getCurrentClusterId();  // current cluster address
 int iPort = getAdviseOnPort();
 String sServer = caller.getCurrentServerId();
 String sHashKey = sCluster = ":" + sServer;      // hash table key

 if (ADV_TWOP_PORT_HTTP == iPort) {                // handle an HTTP server
   iLoadHTTP = getLoadHTTP(iConnectTime, caller);  // get the load for HTTP

   ADV_nte nteHTTP = newADV_nte(sCluster, iPort, sServer, iLoadHTTP);
   putNte(htTwopHTTP, "HTTP", sHashKey, nteHTTP);     // save HTTP load
                                                      // information
   ADV_nte nteSSL = getNte(htTwopSSL, "SSL", sHashKey);  // get SSL
                                                         // information
     if (null != nteSSL) {
       if (true == nteSSL.isCurrent(this)) {              // check the time stamp
         if (ADV_HOST_INACCESSIBLE != nteSSL.getLoadValue()) {    // is SSL
                                                                  // working?
           iLoad = iLoadHTTP;
         } else {                       // SSL is not working, so mark the HTTP server down
           iLoad= ADV_HOST_INACCESSIBLE;
         }
       } else {                 // SSL information is expired, so mark the
                                // HTTP server down
         iLoad = ADV_HOST_INACCESSIBLE;
       }
     } else {                   // no load information about SSL, report
                                // getLoadHTTP() results
       iLoad = iLoadHTTP;
     }
  }
  else if (ADV_TWOP_PORT_SSL == iPort) {             // handle an SSL server
    iLoadSSL = getLoadSSL(iConnectTime, caller);     // get load for SSL

    ADV_nte nteSSL = new ADV_nte(sCluster, iPort, sServer, iLoadSSL);
    putNte(htTwopSSL, "SSL", sHashKey, nteSSL);      // save SSL load info.

    ADV_nte nteHTTP = getNte(htTwopHTTP, "SSL", sHashKey);   // get HTTP
                                                             // information
    if (null != nteHTTP) {
      if (true == nteHTTP.isCurrent(this)) {                 // check the timestamp
        if (ADV_HOST_INACCESSIBLE != nteHTTP.getLoadValue()) {   // is HTTP
                                                                 // working?
          iLoad = iLoadSSL;
        } else {             // HTTP server is not working, so mark SSL down
          iLoad = ADV_HOST_INACCESSIBLE;
        }
      } else {     // expired information from HTTP, so mark SSL down
        iLoad = ADV_HOST_INACCESSIBLE;
      }
    } else {                // no load information about HTTP, report
                            // getLoadSSL() results
      iLoad = iLoadSSL;
    }
  }

 //--------
 // error handler

   else {
     iLoad = ADV_HOST_INACCESSIBLE;
   }
   return iLoad;
 }
}
```

Example: Implementing the WAS advisor:

The following examples show how custom advisors can be implemented.

A sample custom advisor for WebSphere Application Server is included in the *install_root*/servers/samples/CustomAdvisors/ directory. The full code is not duplicated in this document. Ensure that the following will be implemented:

- ADV_was.java is the advisor source code file that is compiled and run on the Load Balancer machine.
- LBAdvisor.java.servlet is the servlet source code that must be renamed to LBAdvisor.java, compiled, and run on the WebSphere Application Server machine.

The complete advisor is only slightly more complex than the sample. It adds a specialized parsing routine that is more compact than the StringTokenizer example shown in the topic "Example: Using data returned from advisors" on page 80.

The more complex part of the sample code is in the Java servlet. Among other methods, the servlet contains two methods required by the servlet specification: init() and service(), and one method, run(), that is required by the Java.lang.thread class.

- init() is called once by the servlet engine at initialization time. This method creates a thread named _checker that runs independently of calls from the advisor and sleeps for a period of time before resuming its processing loop.
- service() is called by the servlet engine each time the servlet is invoked. In this case, the method is called by the advisor. The service() method sends a stream of ASCII characters to an output stream.
- run() contains the core of the code execution. It is called by the start() method that is called from within the init() method.

The relevant fragments of the servlet code appear below:

```
...
  public void init(ServletConfig config) throws ServletException {
    super.init(config);
    ...
    _checker = new Thread(this);
    _checker.start();
  }

  public void run() {
    setStatus(GOOD);

    while (true) {
      if (!getKeepRunning())
        return;
      setStatus(figureLoad());
      setLastUpdate(new java.util.Date());

    try {
      _checker.sleep(_interval * 1000);
    } catch (Exception ignore) { ; }
  }
}

public void service(HttpServletRequest req, HttpServletResponse res)
                    throws ServletException, IOException {
  ServletOutputStream out = null;
  try {
    out = res.getOutputStream();
```

```
  } catch (Exception e) { ... }
  ...
  res.setContentType("text/x-application-LBAdvisor");
  out.println(getStatusString());
  out.println(getLastUpdate().toString());
  out.flush(); return;
}
...
```

Example: Using data returned from advisors:

Whether you use a standard call to an existing part of the application server or add a new piece of code to be the server-side counterpart of your custom advisor, you possibly want to examine the load values returned and change server behavior.

The Java StringTokenizer class, and its associated methods, make this investigation easy to do. The content of a typical HTTP command might be

```
GET /index.html HTTP/1.0 90
```

A typical response to this command might be the following:

```
HTTP/1.1 200 OK
Date: Mon, 20 November 2000 14:09:57 GMT
Server: Apache/1.3.12 (Linux and UNIX)
Content-Location: index.html.en
Vary: negotiate
TCN: choice
Last-Modified: Fri, 20 Oct 2000 15:58:35 GMT
ETag: "14f3e5-1a8-39f06bab;39f06a02"
Accept-Ranges: bytes
Content-Length: 424
Connection: close
Content-Type: text/html
Content-Language: en

<!DOCTYPE HTML PUBLIC "-//w3c//DTD HTML 3.2 Final//EN">
<HTML><HEAD><TITLE>Test Page</TITLE></HEAD>
<BODY><H1>Apache server</H1>
<HR>
<P><P>This Web server is running Apache 1.3.12.
</P>
<P><IMG SRC="apache_pb.gif" ALT="">
</P></P>
</HR>
</BODY></HTML>
```

The items of interest are contained in the first line, specifically the HTTP return code. The HTTP specification classifies return codes that can be summarized as follows:

- 2xx return codes are successes
- 3xx return codes are redirections
- 4xx return codes are client errors
- 5xx return codes are server errors

If you know precisely what codes the server can possibly return, your code might not need to be as detailed as this example. However, keep in mind that limiting the return codes you detect might limit the future flexibility of your program.

The following example is a stand-alone Java program that contains a minimal HTTP client. The example invokes a simple, general-purpose parser for examining HTTP responses.

```
import java.io.*;
import java.util.*;
import java.net.*;

public class ParseTest {
   static final int iPort = 80;
   static final String sServer = "www.ibm.com";
   static final String sQuery = "GET /index.html HTTP/1.0\r\n\r\n";
   static final String sHTTP10 = "HTTP/1.0";
   static final String sHTTP11 = "HTTP/1.1";

   public static void main(String[] Arg) {
     String sHTTPVersion = null;
     String sHTTPReturnCode = null;
     String sResponse = null; int iRc = 0;
     BufferedReader brIn = null;
     PrintWriter psOut = null;
     Socket soServer= null;
     StringBuffer sbText = new
     StringBuffer(40);

   try {
     soServer = new Socket(sServer, iPort);
     brIn = new BufferedReader(new InputStreamReader(
                                  soServer.getInputStream()));
     psOut = new PrintWriter(soServer.getOutputStream());
     psOut.println(sQuery);
     psOut.flush();
     sResponse = brIn.readLine();
     try {
       soServer.close();
     } catch (Exception sc) {;}
   } catch (Exception swr) {;}

   StringTokenizer st = new StringTokenizer(sResponse, " ");
   if (true == st.hasMoreTokens()) {
     sHTTPVersion = st.nextToken();
     if (sHTTPVersion.equals(sHTTP110) || sHTTPVersion.equals(sHTTP11)) {
       System.out.println("HTTP Version: " + sHTTPVersion);
     } else {
       System.out.println("Invalid HTTP Version: " + sHTTPVersion);
     }
   } else {
     System.out.println("Nothing was returned");
     return;
   }

   if (true == st.hasMoreTokens()) {
     sHTTPReturnCode = st.nextToken();
     try {
       iRc = Integer.parseInt(sHTTPReturnCode);
     } catch (NumberFormatException ne) {;}

     switch (iRc) {
     case(200):
       System.out.println("HTTP Response code: OK, " + iRc);
       break;
     case(400): case(401): case(402): case(403): case(404):
       System.out.println("HTTP Response code: Client Error, " + iRc);
       break;
     case(500): case(501): case(502): case(503):
       System.out.println("HTTP Response code: Server Error, " + iRc);
       break;
```

```
      default:
        System.out.println("HTTP Response code: Unknown, " + iRc);
        break;
      }
    }

    if (true == st.hasMoreTokens()) {
      while (true == st.hasMoreTokens()) {
        sbText.append(st.nextToken());
        sbText.append(" ");
        }
      System.out.println("HTTP Response phrase: " + sbText.toString());
    }
   }
}
```

Configuring high availability

The high availability feature involves the use of a second Dispatcher machine. The first Dispatcher machine performs load balancing for all the client traffic as it does in a single Dispatcher configuration. The second Dispatcher machine monitors the "health" of the first, and takes over the task of load balancing if it detects that the first Dispatcher machine has failed.

When you configure high availability, each of the two machines is assigned a specific role, either primary or backup. The primary machine sends connection data to the backup machine on an ongoing basis. While the primary is active (load balancing), the backup is in a standby state, continually updated and ready to take over, if necessary.

The communication sessions between the two machines are referred to as heartbeats. The heartbeats allow each machine to monitor the health of the other. If the backup machine detects that the active machine has failed, it will take over and begin load balancing. At that point the statuses of the two machines are reversed: the backup machine becomes active and the primary becomes standby.

Note: In the high availability configuration, both primary and backup machines must be on the same subnet with identical configuration.

For the complete syntax see "dscontrol highavailability" on page 146. For a more complete discussion of many of the tasks below, see "Configuring the Load Balancer machine" on page 34.

Tips for configuring high availability:

1. To configure a single Dispatcher machine to route packets without a backup, do not issue any of the high availability commands at startup.
2. To convert two Dispatcher machines configured for high availability to one machine running alone, stop the executor on one of the machines, then delete the high availability features (the heartbeats, reach, and backup) on the other.
3. Linux **Linux for s390:** In both of the two cases above, you must alias the network interface card with cluster addresses, as required.
4. When running two Dispatcher machines in a high availability configuration, unexpected results can occur if you set any of the parameters for the executor, cluster, port, or server (for example, port stickytime) to different values on the two machines.

1. Linux If you are running Linux for s390 operating systems, create alias script files on each of the two Dispatcher machines. See "Scripts to run with high availability" on page 86 for more information.
2. Start the server on both Dispatcher server machines.
3. Start the executor on both machines.
4. Ensure that the non-forwarding address (NFA) of each Dispatcher machine is configured, and is a valid IP address for the subnet of the Dispatcher machines.
5. Add the heartbeat information on both machines:

   ```
   dscontrol highavailability heartbeat add source_address destination_address
   ```

   Source_address and *destination_address* are the IP addresses (either DNS names or IP addresses) of the Dispatcher machines. The values will be reversed on each machine. For example:

   ```
   Primary - highavailability heartbeat add 9.67.111.3 9.67.186.8
   Backup - highavailability heartbeat add 9.67.186.8 9.67.111.3
   ```

   At least one heartbeat pair must have the NFAs of the pair as the source and destination address. If possible, at least one of the heartbeat pairs should be across a separate subnet than the regular cluster traffic. Keeping the heartbeat traffic distinct will help prevent false takeovers during very heavy network loads and also improve complete recovery times after a failover.

   a. Optional: Set the number of seconds that the executor uses to timeout high availability heartbeats. The default is 2 seconds. For example:

      ```
      dscontrol executor set hatimeout 3
      ```

6. On both machines, configure the list of IP addresses that the Dispatcher must be able to reach in order to ensure full service, using the reach add command. Reach targets are recommended but not required. See "Detecting server failures with heartbeats and reach targets" on page 85 for more information. For example:

   ```
   dscontrol highavailability reach add 9.67.125.18
   ```

7. Add the backup information to each machine:

   a. For the primary machine:

      ```
      dscontrol highavailability backup add primary [auto | manual] port
      ```

   b. For the backup machine:

      ```
      dscontrol highavailability backup add backup [auto | manual] port
      ```

      Note: Select an unused port on your machines as the port. The port number entered will be used as a key to ensure the correct host is receiving the packet.

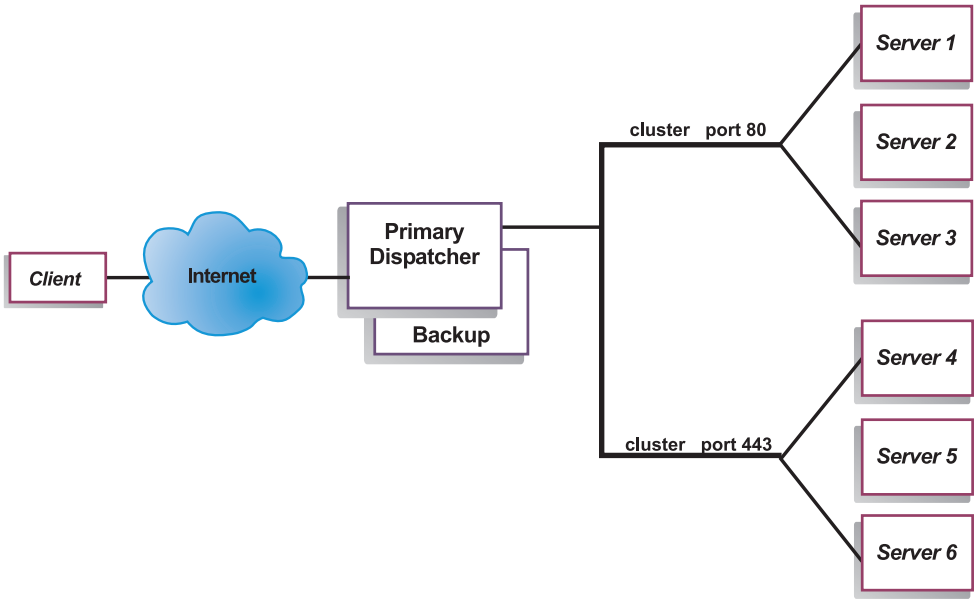
8. Check the high availability status on each machine:

   ```
   dscontrol highavailability status
   ```

   The machines should each have the correct role (backup or primary) and states. The primary should be active; the backup should be in standby mode. The recovery strategies must be the same.

9. Set up the cluster, port, and server information on both machines.
10. Start the manager and advisors on both machines.

How high availability works



To improve Dispatcher availability, the Dispatcher high availability functions as follows:

1. Two Dispatchers with connectivity to the same clients, and the same cluster of servers, as well as connectivity between the Dispatchers. Both Dispatchers must run on the same type of operating system and platform.
2. A "heartbeat" mechanism between the two Dispatchers detects a Dispatcher failure. At least one heartbeat pair must have the NFAs of the pair as the source and destination address. If possible, at least one of the heartbeat pairs should be across a separate subnet than the regular cluster traffic. Keeping the heartbeat traffic distinct will help prevent false takeovers during very heavy network loads and also improve complete recovery times after a failover.
3. A list of reach targets, addresses that both Dispatcher machines must be able to contact in order to load balance traffic normally. For more information, see "Detecting server failures with heartbeats and reach targets" on page 85.
4. Synchronization of the Dispatcher information
5. Logic to elect the active Dispatcher which is in charge of a given cluster of servers, and the standby Dispatcher which continuously gets synchronized for that cluster of servers.
6. A mechanism to perform IP takeover, when the logic or an operator decides to switch active and standby.

Planning for high availability

When configuring for high availability, consider that the Load Balancer machine is supported with the following limitations or special considerations:

- If you are using IPv6 protocol on your machine and want to use high availability, you must check to see if protocol 58 is defined to be ICMPv6 in the protocol file.
- In the high availability configuration, both primary and backup machines must be on the same subnet with identical configuration.

- The heartbeat pairs (which is the mechanism between the primary and standby Dispatchers to detect Dispatcher failure) must be both IPv4 format or both IPv6 format.
- In a high availability or a stand-alone environment, you must not alias the cluster address against the network adaptor.
- The HighAvailChange script can be moved from the *install_root*/servers/samples directory to the *install_root*/servers/bin directory to log high availability state changes for the Dispatcher machine, but this script does not need to be changed.

Detecting server failures with heartbeats and reach targets

Configure heartbeats and reach targets to detect server failures and control when failovers can occur.

Besides the basic criteria of failure detection (the loss of connectivity between active and standby Dispatchers, detected through the heartbeat messages), there is another failure detection mechanism named reachability criteria. When you configure the Dispatcher you can provide a list of hosts that each of the Dispatchers should be able to reach in order to work correctly. The two high availability partners continually communicate with each other through heartbeats, and they update one another on how many reach targets either one of them can ping. If the standby pings more reach targets than the active, a failover occurs.

Heartbeats: Heartbeats are sent by the active Dispatcher and are expected to be received by the standby Dispatcher every half second. If the standby Dispatcher fails to receive a heartbeat within 2 seconds, a failover begins. All heartbeats must break for a takeover from the standby Dispatcher to occur. In other words, when two heartbeat pairs are configured, both heartbeats must break. To stabilize a high availability environment and to avoid failover, add more than one heartbeat pair.

Reach target considerations: For reach targets, you should choose at least one host for each subnet your Dispatcher machine uses. The hosts could be routers, IP servers or other types of hosts. Host reachability is obtained by the reach advisor, which pings the host. Failover takes place either if the heartbeat messages cannot go through, or if the reachability criteria are met better by the standby Dispatcher than by the primary Dispatcher. To make the decision based on all available information, the active Dispatcher regularly sends the standby Dispatcher its reachability capabilities. The standby Dispatcher then compares those capabilities with its own and decides whether to switch.

Note: When you configure the reach target, the reach advisor must also be started. The reach advisor starts automatically when you start the manager function. For more information on the reach advisor, see "List of advisors" on page 53.

- Use the dscontrol highavailability command to add or delete a reach target to a server:

  ```
  dscontrol highavailability reach add|delete address mask
  ```

- Use the dscontrol highavailability command with the heartbeat option to add a heartbeat:

  ```
  dscontrol highavailability heartbeat add srcaddress dstaddress
  ```

  To delete a heartbeat, use the following:

  ```
  dscontrol highavailability heartbeat delete address
  ```

High Availability recovery strategy for failed servers

The recovery strategy dictates how Load Balancer behaves when one Dispatcher machine fails and there is another configured as a backup.

Two Dispatcher machines are configured: the primary machine, and a second machine called the backup. At startup, the primary machine sends all the connection data to the backup machine until that machine is synchronized. The primary machine becomes active, that is, it begins load balancing. The backup machine, meanwhile, monitors the status of the primary machine, and is said to be in standby state.

If the backup Load Balancer machine detects that the primary machine has failed, it performs a takeover load balancing functions and becomes the active machine. After the primary machine has once again become operational, the machines respond according to how the recovery strategy has been configured by the user.

There are two kinds of strategy:

- **Automatic** - The primary machine resumes routing packets as soon as it becomes operational again.
- **Manual** - intervention is required to return the primary machine to active state and reset the backup machine to standby. The manual recovery strategy allows you to force the routing of packets to a particular machine, using the takeover command. Manual recovery is useful when maintenance is being performed on the other machine

Note: The strategy parameter must be the same for both machines.

Scripts to run with high availability

Before using a script, remember that for Dispatcher to route packets, each cluster address must be aliased to a network interface device.

- In a stand-alone Dispatcher configuration, each cluster address must be aliased to a network interface card (for example, en0, tr0).
- In a high availability configuration:
  - On the active machine, each cluster address must be aliased to a network interface card (for example, en0, tr0)
  - On the standby machine, each cluster address must be aliased to a loopback device (for example, lo0).
  - To customize the scripts for certain situations, read Customizing the scripts for high availability.
- In any machine in which the executor has been stopped, all aliases should be removed to prevent conflicts with another machine that may be started. For information on aliasing the network interface card, see “Aliasing the network interface card or loopback device” on page 39.

The following sample script can be used:

- **HighAvailChange**

  The HighAvailChange script runs whenever the high availability state changes within the Dispatcher. You can create this script to use state change information, for instance, to alert an Administrator or simply record the event.

On Linux for S/390: Dispatcher issues a gratuitous ARP to move IP addresses from one Dispatcher to another. This mechanism is therefore tied to the underlying

network type. When running Linux for S/390, Dispatcher can natively do high availability takeovers (complete with IP address moves) only on those interfaces which can issue a gratuitous ARP and configure the address on the local interface. This mechanism will not work properly on point-to-point interfaces such as IUCV and CTC and will not work properly in certain configurations of qeth/QDIO.

Related tasks

"Configuring high availability" on page 82
The high availability feature involves the use of a second Dispatcher machine. The first Dispatcher machine performs load balancing for all the client traffic as it does in a single Dispatcher configuration. The second Dispatcher machine monitors the "health" of the first, and takes over the task of load balancing if it detects that the first Dispatcher machine has failed.

Customizing the scripts for high availability
Customize your scripts to tune the performance of the high availability function for Load Balancer.

"Detecting server failures with heartbeats and reach targets" on page 85
Configure heartbeats and reach targets to detect server failures and control when failovers can occur.

Related reference

"High Availability recovery strategy for failed servers" on page 86
The recovery strategy dictates how Load Balancer behaves when one Dispatcher machine fails and there is another configured as a backup.

Use encapsulation forwarding to forward traffic across network segments

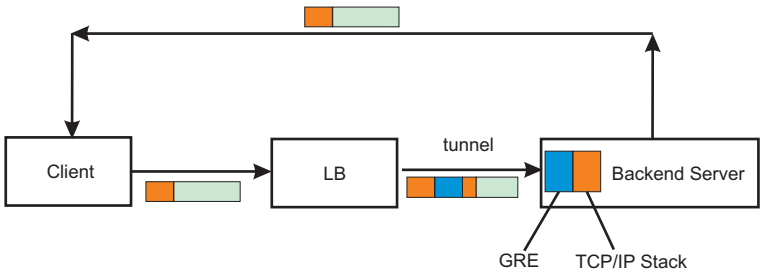
Use encapsulation forwarding when the back-end server is not located on the same network segment or if you are using virtualization technology and need to forward packets that are otherwise unable to be forwarded.

In a typical configuration, Load Balancer receives a packet, P, and forwards it as packet P', where only the time-to-live (TTL) has been decremented. When you enable encapsulation, Load Balancer receives a packet P, and forwards it as E(P'), where the encapsulated packet E contains P'. The outer packet E has a unique IP header, which permits Load Balancer to forward packets across routers and across some types of virtualization technology that you could not otherwise forward packets across.

Encapsulation forwarding:

- Is implemented like MAC forwarding:
  - Packets from server to client do not go through the load balancer
  - Alias the loopback device to cluster address on back-end server
- Requires that you configure an IPIP or GRE tunnel only on the back-end server.
- Does not require you to add routes while configuring the tunnel.

Load Balancer will act as the tunnel on the other end.



Additionally, this functionality allows you to forward packets to Solaris zones or AIX workload partitions that are on the same host, since Load Balancer can use the existing stack configuration instead of bypassing it entirely.

1. On the Load Balancer machine, add a server with encapsulation enabled. When this server is selected to forward the packet, it is encapsulated. Use the dscontrol server command:

   ```
   dscontrol server set encap_source_IP encapforward [yes/no] encaptype [ipip/gre] encapcond [auto/a
   ```

   For example, you can type the following at the prompt:

   ```
   dscontrol server set 1.2.3.4@80@1.2.3.5 encapforward yes encaptype ipip encapcond always
   ```

2. Configure the IPIP or GRE tunnel on the back-end server for network traffic. For example, you can type the following:

   -

     ```
     ifconfig gre0 tunnel 9.184.119.242  9.184.118.200 # The IP address of the server and Load Balar
     ifconfig gre0  inet 9.184.114.25      # Some IP address on this subnet
     ### loopback...
     ifconfig lo0 alias 9.184.114.24 netmask 255.255.255.255
     ```

   - Linux To set up a GRE tunnel, use the following:

     ```
     sysctl -w net.ipv4.conf.all.arp_ignore=3 net.ipv4.conf.all.arp_announce=2

     # for gre
     modprobe ipgre
     ip link set gre0 up
     ip addr add <clusterip> scope host dev gre0
     ```

     To set up an IPIP tunnel, use the following:

     ```
     sysctl -w net.ipv4.conf.all.arp_ignore=3 net.ipv4.conf.all.arp_announce=2
     modprobe ipip
     ip link set tunl0 up
     ip addr add <clusterip> scope host dev tunl0
     ```

   -

     ```
     /sbin/ifconfig ip.tun0 plumb 9.184.114.25 netmask 255.255.255.255 up  #  Some free IP address c
     /sbin/ifconfig ip.tun0 9.184.114.25  9.184.114.222 up   #  Some free IP address on this subnet
     /sbin/ifconfig ip.tun0 up /sbin/ifconfig ip.tun0 tsrc 9.184.112.183 tdst 9.184.118.203  # The
     ### loopback...
     ifconfig lo0:1 plumb 9.184.114.24 netmask 255.0.0.0 up
     ```

   - Tunneling is not supported on Windows operating systems.

Quiesce servers for server maintenance windows

To remove a server from the Load Balancer configuration for any reason (updates, upgrades, service, etc.), you can use the dscontrol manager quiesce command.

The quiesce subcommand allows existing connections to complete without being severed and disallows any new connections to the server.

Note: You can quiesce servers on a scheduled time to perform upgrades or general maintenance. The daily option specifies to quiesce the server at a time that you specify.

- Quiesce a server immediately. Use the following command:

  ```
  dscontrol manager quiesce server
  ```

  The following is an example of using the option to quiesce server 9.40.25.67:

  ```
  dscontrol manager quiesce 9.40.25.67
  ```

- Quiesce a server on a daily schedule. Use the following command:

  ```
  dscontrol manager quiesce server daily start_hour end_hour
  ```

  Note:

  – *start hour* and *end hour* are values from 0 to 23. For example, (0 0) indicates to quiesce the server from 12:00 AM to 12:59 AM. (12 13) indicates to quiesce the server from 12:00 PM to 1:59 PM, which is a 2 hour period.

  – Specify (-1 -1) to disable the daily quiesce for a particular server.

  The following is an example of using the daily option to quiesce server 9.40.25.67 from 2:00 AM to 5:59 AM:

  ```
  dscontrol manager quiesce 9.40.25.67 daily 2 5
  ```

Optimize connections with client-to-server affinity

The Load Balancer affinity feature maps a client IP address to a back-end server. Affinity is established once a packet's destination IP address matches the cluster, the destination port matches the Load Balancer port, and the source IP address matches.

When affinity is established, subsequent packets are sent to the same back-end server. When affinity is broken, due to a server down or a server removal, all affinity and thus connections to that server are broken. Also, there is no "connection" information reported in the command line or GUI clients. Only the number of active affinity records are used.

This approach has the advantages of providing a hard affinity and of being more efficient for Load Balancer. The affinity method that is used decreases memory and CPU utilization as compared to connection forwarding.

Note: Because the removal of an affinity record also breaks connections, when you migrate from Load Balancer for IPv4 to Load Balancer for IPv4 and IPv6, the maximum staletimeout value should be used as the new staletimeout for Load Balancer.

- **Behavior when affinity is enabled:**

  With the affinity feature enabled, if a subsequent request is received from the same client, the request is directed to the same server.

  Over time, the client will finish sending transactions, and the affinity record will go away. Each affinity record lives for the "staletimeout" in seconds. When subsequent connections are received within the staletimeout, the affinity record is still valid and the request will go to the same server. If a subsequent connection is not received within staletimeout, the record is purged; a connection that is received after that time will have a new server selected for it.

  The server down command (dscontrol server down) is used to bring a server offline. The server is not taken down until after the staletimeout value expires.

- **Behavior when affinity is disabled:**

  With the affinity feature disabled, whenever a new TCP connection is received from a client, Load Balancer picks the right server at that moment in time and forwards the packets to it. If a subsequent connection comes in from the same client, Load Balancer treats it as an unrelated new connection, and again picks the right server at that moment in time.
- Optional: Enable affinity by adding a port and setting the selection algorithm and stickytime at the port level to some number of seconds using the dscontrol port command.

  Note: There are now three selections you can choose from for selection algorithm:
  - **affinity:** specifies that the server selection is based on client affinity.
  - **connection:** specifies that the server selection is based on weighted round-robin selection (default).
  - **conn+affin:** specifies that server selection is based on the relationship to an existing connection. For new connections, the server selection is based on affinity.

  For example, use the following command to set the selection algorithm to affinity and stickytime to 60 seconds:

  ```
  dscontrol port add cluster@port selectionalgorithm affinity stickytime 60
  ```

  Note: When you enable affinity, it cannot be disabled unless you completely remove the port and add it again without affinity configured.
- Optional: Enable cross-port affinity. Cross port affinity is the affinity feature that has been expanded to cover multiple ports. For example, if a client request is first received on one port and the next request is received on another port, cross port affinity allows Load Balancer to send the client request to the same server. To use this feature, the ports must:
  - Share the same cluster address.
  - Share the same servers.
  - Use the same selection algorithm, which must be affinity or conn+affin.
  - Have the same stickytime value, which is not zero. After cross port affinity has been established, you have the flexibility to modify the stickytime value for the port. However, it is recommended that you change the stickytime values for all shared ports to the same value, otherwise results might occur that are not expected.

  More than one port can link to the same crossport. When subsequent connections arrive from the same client on the same port or a shared port, the same server will be accessed.

  1. Configure the selection algorithm to affinity or conn+affin with the dscontrol port command. For example, use the following command:

     ```
     dscontrol port add cluster@port selectionalgorithm conn+affin
     ```
  2. Configure the stickytime value with the dscontrol port command. For example, use the following command to set the stickytime to 60 seconds:

     ```
     dscontrol port set cluster@port stickytime 60
     ```
  3. Configure the crossport value with the dscontrol port add command. The following is an example of configuring multiple ports with a cross port affinity to port 10:

```
dscontrol port add cluster@20 selectionalgorithm conn+affin stickytime 60 crossport 10
dscontrol port add cluster@30 selectionalgorithm conn+affin stickytime 60 crossport 10
dscontrol port add cluster@40 selectionalgorithm conn+affin stickytime 60 crossport 10
```

Note: You can only specify a value for crossport with the dscontrol port add command and cannot be modified afterwards. You cannot use the dscontrol port set command to configure the crossport value.

See dscontrol port for detailed information on command syntax for the crossport option.

Restricting incoming traffic with ipchains and iptables

Built into the Linux kernel is a firewall facility called ipchains. When Load Balancer and ipchains run concurrently, Load Balancer sees packets first, followed by ipchains. This allows the use of ipchains to harden a Linux Load Balancer machine, which could be, for example, a Load Balancer machine that is used to load balance firewalls.

In general, an appropriate ipchains strategy for the Load Balancer machines is to disallow all traffic, except that which is to or from the back-end servers, the partner high availability Load Balancer, any reach targets, or any configuration hosts.

Linux It is not recommended to activate iptables when running Load Balancer on Linux kernel version 2.4.10.x. Activation on this Linux kernel version can result in performance degradation over time.

- To activate iptables or ipchains, configure them to be completely restricted, so no inbound or outbound traffic permitted. The packet-forwarding portion of Load Balancer continues to function normally.

  Some additional traffic must be permitted for all of Load Balancer to function properly. Some examples of this communication are:

  – Advisors communicate between the Load Balancer machine and the back-end servers.
  – Load Balancer pings back-end servers, reach targets, and high availability partner Load Balancer machines.
  – User interfaces (graphical user interface, command line, and wizards) use RMI.
  – Back-end servers must respond to pings from the Load Balancer machine.
- To deactivate iptables:

  1. List the modules which are using ip_tables and ip_conntrack. Issue the following command:

     ```
     lsmod
     ```

  2. Remove them by issuing the following commands:

     ```
     rmmod ip_tables
     rmmod ip_conntrack
     ```

     When you reboot the machine these modules will be added again, so you need to repeat these steps each time you reboot.

Logging with Load Balancer

Load Balancer posts entries to a server log, a manager log, a metric monitor log (logging communications with Metric Server agents), and a log for each advisor you use.

You can set the logging level to define the expansiveness of the messages written to the log. At level 0, errors are logged and Load Balancer also logs headers and records of events that happen only once (for example, a message about an advisor starting to be written to the manager log). Level 1 includes ongoing information, and so on, with level 5 including every message produced to aid in debugging a problem if necessary. The default for the manager, advisor, server, or subagent logs is 1.

You can also set the maximum size of a log. When you set a maximum size for the log file, the file will wrap; when the file reaches the specified size, the subsequent entries are written at the top of the file, overwriting the previous log entries.

Note: You cannot set the log size to a value that is smaller than the current one. The higher you set the log level, the more carefully you should choose the log size. At level 0, it is probably safe to leave the log size to the default of 1MB; however, when logging at level 3 and above, you should limit the size without making it too small to be useful.

Log entries are time stamped so you can tell the order in which they were written.

- Display the current settings for the server log, Use the dscontrol logstatus command:

  ```
  dscontrol logstatus
  ```

- Configure the logging level or maximum log size for a server log. Use the dscontrol set command:

  ```
  dscontrol set loglevel level logsize size
  ```

  where:

  – *level* is 0-5.
  – *size* is unlimited or a file size in bytes.

- Configure the logging level or maximum log size for a manager log. Use the dscontrol manager command:

  ```
  dscontrol manager loglevel level

  dscontrol manager logsize size
  ```

  where:

  – *level* is 0-5.
  – *size* is unlimited or a file size in bytes.

- Configure the logging level or maximum log size for the metric monitor log that logs communication with Metric Server agents. Use the dscontrol manager metric set command:

  ```
  dscontrol manager metric set loglevel level

  dscontrol manager metric set logsize size
  ```

- Configure the logging level or maximum log size for an advisor log. Use the dscontrol advisor command:

  ```
  dscontrol advisor loglevel name cluster@port level

  dscontrol advisor logsize name cluster@port size
  ```

where:

– *cluster@port* the cluster is the address in IP address format or symbolic name. The port is the number of the port that the advisor is monitoring. The cluster value is optional on the advisor commands, but the port value is required. If the cluster value is not specified, then the advisor will start running on the port for all clusters. If you specify a cluster, then the advisor will start running on the port, but only for the cluster you have specified. See "Enabling advisors to manage load balancing" on page 50 for more information.

Related tasks

"Enabling advisors to manage load balancing" on page 50
Advisors are software agents that work within Load Balancer to provide information about the load on a given server. A different advisor exists for each standard protocol (HTTP, SSL, and others). Periodically, the Load Balancer base code performs an advisor cycle, during which it individually evaluates the status of all servers in its configuration.

Related reference

"dscontrol logstatus" on page 149
Use this command to display the log settings for a server.

"dscontrol metric" on page 153
You can configure system metrics with the `dscontrol metric` command.

"dscontrol manager" on page 149
You can control the manager function with the `dscontrol manager` command.

Logging server statistics with binary logging

The binary logging feature allows server information to be stored in binary files. These files can then be processed to analyze the server information that has been gathered over time.

The following information is stored in the binary log for each server defined in the configuration.

- cluster address
- port number
- serverID
- server address
- server weight
- server total connections
- server active connections
- server port load
- server system load

Some of this information is retrieved from the executor as part of the manager cycle. Therefore the manager must be running in order for the information to be logged to the binary logs.

A sample Java program and command file have been provided in the *install_root*/servers/samples/BinaryLog directory. This sample shows how to retrieve all the information from the log files and print it to the screen. It can be customized to do any type of analysis you want with the data. An example using the supplied script and program to get a report of the Load Balancer's server information from 8:00 AM to 5:00 PM on May 1, 2001:

```
dslogreport 2001/05/01 8:00 2001/05/01 17:00
```

Use dscontrol binlog command set to configure binary logging

- Start binary logging:

  ```
  dscontrol binlog start
  ```

  The start option starts logging server information to binary logs in the logs directory. One log is created at the start of every hour with the date and time as the name of the file.
- Stop binary logging:

  ```
  dscontrol binlog stop
  ```

  The stop option stops logging server information to the binary logs. The log service is stopped by default.
- Set the interval value to control how often information is written to the logs.

  ```
  dscontrol binlog interval seconds
  ```

  The manager will send server information to the log server every manager interval. The information is written to the logs only if the specified log interval seconds have elapsed since the last record was written to the log. By default, the log interval is set to 60 seconds. There is some interaction between the settings of the manger interval and the log interval. Since the log server is provided with information no faster than manager interval seconds setting the log interval less than the manager interval effectively sets it to the same as the manager interval.

  This logging technique allows you to capture server information at any granularity. You can capture all changes to server information that are seen by the manager for calculating server weights. However, this amount of information is probably not required to analyze server usage and trends. Logging server information every 60 seconds gives you snapshots of server information over time. Setting the log interval very low can generate huge amounts of data.
- Set the retention option to control how long log files are kept.

  ```
  dscontrol binlog retention hours
  ```

  Log files older than the retention *hours* specified are deleted by the log server. This will only occur if the log server is being called by the manager, so stopping the manager will cause old log files not to be deleted.
- View the current status for binary logging:

  ```
  dscontrol binlog status
  ```

  The status option returns the current settings of the log service. These settings are whether the service is started, what the interval is, and what the retention hours are.

Support for ICMP forwarding and messaging

Load Balancer now supports forwarding and processing ICMP messages to improve the robustness of connection protocols and permit Load Balancer to receive ICMP fragmentation messages.

Load Balancer will forward an ICMP message based on the following guidelines:

- For ICMP packets that contain headers with IP and TCP/UDP fragments, Load Balancer will forward to packets to the correct back-end server.
- For ICMP packets that do not contain TCP/UDP fragments, Load Balancer will forward the packets in a round robin manner.

- For ICMP messages that are for an IPGRE or an IPIP message that Load Balancer generated, Load Balancer will limit the outbound size appropriately for future packets.
- If Load Balancer forwards an IP packet, but the time to live (TTL) for the packet becomes zero when the TTL is decremented, Load Balancer will send an ″ICMP Time Exceeded″ message to the client.
- When Load Balancer cannot forward an packet, it will generate an ICMP message and send the appropriate message back to the client:
  - The outbound interface MTU is too small, or you need to use encapsulation.
  - Load Balancer cleaned up a connection record. For example, the cluster and port designation match, but the server is not present.

Configure rules to manage traffic to busy or unavailable servers

Use rules-based load balancing to fine tune when and why packets are sent to which servers. Load Balancer reviews any rules you add from first priority to last priority, stopping on the first rule that it finds to be true, then load balancing the traffic between any servers associated with the rule. It already balances the load based on the destination and port, but using rules expands your ability to distribute connections.

In most cases when configuring rules, you should configure a default always true rule in order to catch any request that is passed by other higher priority rules. This default can be a ″Sorry, the site is currently down, try again later″ response when all other servers fail for the client request.

All rules have a name, type, priority, and might have a begin range and end range, along with a set of servers. Rules are evaluated in priority order. A rule with a priority of 1 (lower number) is evaluated before a rule with a priority of 2 (higher number). The first rule that is satisfied will be used. When a rule has been satisfied, no further rules are evaluated. For a rule to be satisfied, it must meet two conditions:

1. The predicate of the rule must be true. That is, the value it is evaluating must be between the begin and end ranges, or the content must match the regular expression that is specified in the rule's pattern. For rules of type ″true,″ the predicate is always satisfied, regardless of the begin and end ranges. If a rule has no servers that are associated with it, the rule only needs to meet this first condition to be satisfied. In this case, Load Balancer will drop the connection request.
2. If there are servers associated with the rule, at least one server must have a weight greater than 0 to forward packets so Load Balancer will have a server to which connections can be forwarded.

If a connection request does not satisfy any rules, Load Balancer will select a server from the full set of servers available on the port.

- Configure a rule that is based on the total active connections. You may want to use rules based on active connections total on a port if your servers get overloaded and start throwing packets away. Certain Web servers will continue to accept connections even though they do not have enough threads to respond to the request. As a result, the client requests time out and the customer coming to your Web site is not served. You can use rules based on active connections to balance capacity within a pool of servers. For example, you know from experience that your servers will stop serving after they have accepted 250 connections.

Note: The manager must be running for the rules to work.
Create a rule using the dscontrol rule command. You would then add to the rule your current servers plus some additional servers, which will otherwise be used for other processing. For example:

```
dscontrol rule add 130.40.52.153:80:pool2 type active beginrange 250 endrange 500
```

- Create a rule that always evaluates as true. Such a rule will always be selected, unless all the servers associated with it are down. Therefore, this rule should ordinarily be at a lower priority than other rules. You can even have multiple "always true" rules, each with a set of servers that are associated with it. Load Balancer will choose a rule based on the first rule that is true and has an available server.

  For example, assume you have six servers. You want two of them to handle your traffic under all circumstances, unless they are both down. If the first two servers are down, you want a second set of servers to handle the traffic. If all four of these servers are down, then you will use the final two servers to handle the traffic. You could set up three "always true" rules, then the first set of servers will always be chosen as long as at least one is up. If both servers are down, one from the second set is chosen, and so forth.

  As another example, you might want an "always true" rule to ensure that if incoming clients do not match any of the rules you have set, they will not be served. Then you would not add any servers to the rule, causing the clients packets to be dropped with no response. You can define more than one "always true" rule, and thereafter adjust which one gets run by changing their priority levels. Create a rule using the dscontrol rule command:

  ```
  dscontrol rule add 130.40.52.153:80:jamais type true priority 100
  ```

  You do not need to set a beginrange or endrange values when you create an always true rule.
- Add one or more servers to a rule set. You can use the dscontrol rule useserver command to add one or more servers to a rule set that is already defined. For example:

  ```
  dscontrol rule useserver 130.40.52.153:80:jamais server1

  dscontrol rule useserver 130.40.52.153:80:jamais server1+server2+server3
  ```

Related reference

"dscontrol rule" on page 156
Control the executor function with the `dscontrol rule` command.

Related information

Tuning

Sample scripts to generate alerts and record server failure

Load Balancer provides user exits that trigger scripts that you can customize. You can create the scripts to perform automated actions, such as alerting an Administrator when servers are marked down by the manager or simply record the event of the failure.

Sample scripts, which you can customize, are in the *install_root*/servers/samples directory. In order to run the files, you must move them to the *install_root*/servers/bin directory and remove the ⌂sample⌂ file extension. The following sample scripts are provided:

- **serverDown** — a server is marked down by the manager.
- **serverUp** — a server is marked back up by the manager.

- **managerAlert** — all servers are marked down for a particular port.
- **managerClear** — at least one server is now up, after all were marked down for a particular port.

If all servers on a cluster are marked down (either by the user or by the advisors), the managerAlert (if configured) starts, and Load Balancer attempts to route traffic to the servers using a round-robin technique. The serverDown script does not start when the last server in the cluster is detected as offline. By design, Load Balancer attempts to continue to route the traffic in case a server comes back online and responds to the request. If Load Balancer instead dropped all traffic, the client would receive no response. When Load Balancer detects that the first server of a cluster is back online, the managerClear script (if configured) starts, but the serverUp script (if configured) is not run until an additional server is brought back online.

Here are some considerations for using the serverUp and serverDown scripts:

- If you define the manager cycle to be less than 25% of the advisor time, false reports of servers up or down can result. By default, the manager runs every 2 seconds, but the advisor runs every 7 seconds. Therefore, the manager expects new advisor information within 4 cycles. However, removing this restriction (that is, defining the manager cycle to be greater than 25% of the advisor time) significantly decreases performance because multiple advisors can advise on a single server.
- When a server goes down, the serverDown script starts. However, if you issue a serverUp command, it is assumed that the server is up until the manager obtains new information from the advisor cycle. If the server is still down, the serverDown script runs again.

Chapter 5. Tuning Load Balancer

How well a Web site performs while receiving heavy user traffic is an essential factor in the overall success of an organization. This topic highlights a few main ways you can improve performance through a combination of product features and application development considerations.

The manager report

The manager function of Load Balancer calculates a weight for each server. These weights are used to determine how many connections a server should receive as compared with the other servers in the same cluster and port configuration. Understanding the manager report is critical to understanding how the network traffic is distributed.

The manager report contains a list of each cluster, port, and server that is defined to that cluster:port combination. Each server shows two weights, now and new, and four columns that are used to calculate the weight:

- Active connections (ACTV)
- New connections (NEWC)
- Port Load (PORT)
- System load (SYS)

Each of the four columns is assigned a percentage that is used to calculate the weight for the server. The percentages are set with the cluster set proportion command. By default, only the active connections and new connections are considered when calculating the weight of the server. When an advisor is started, the proportion for the port load is set to 1% so that the port load is used in the weight calculation. Similarly, when a metric is added the proportion for the system load is set to 1%. The manager function returns the following values for each server:

Active connections (ACTV)
Active connections are TCP connections that are closed at the start of the manager cycle.

New connections (NEWC)
New connections represent the increase in total connections from the start of the manager cycle to the start of the last manager cycle.

Port Load (PORT)
The port load is the value that is obtained from an advisor that is defined on this cluster:port combination. If an advisor is not started, the port load is always zero. When an advisor is defined, the port load typically represents the number of milliseconds for the advisor to receive a response from the server.

When the port load is shown as -1, the advisor did not receive a successful response to its query. Increase the log level and log size for the advisor to investigate why the server did not respond. If the server never responds to the connection request, complete the following steps:

1. Ensure that you can successfully ping the server from the Load Balancer machine.
2. Verify that the server application is started and listening on the port that is defined. The server should be listening on the wildcard address (0.0.0.0), or both the cluster IP address and the real server IP address to successfully respond to the advisor requests.

If the server responds to the connection, then it might be responding to the query in a manner that is different from what Load Balancer is expecting to see. Check the advisorresponse string that is defined to ensure it matches what the server has transmitted. This scenario applies to both http and https advisors.

System load (SYS)

The system load represents the value that is returned from the metric server. If metrics have not been added for this cluster:port combination, the system load is zero (0). When metrics are defined, the system load is a value in between -1 and 100, which represents the status of the server. 100 is very busy and zero (0) is idle.

If the system load shows -1, Load Balancer cannot communicate with the metric server on the back-end machine. Ensure Load Balancer keys are properly distributed to the server, that the server can be pinged from the Load Balancer, and that metric server is started on the machine. If the problem persists, complete the following steps:

1. Edit the script for the metric server on the back-end machine and increase both the log level and log size.
2. Restart the metric server.
3. Increase the the log size and the log level for the metric monitor at the Load Balancer.
4. Examine the log files on both the Load Balancer machine and the back-end machine to determine why communication is failing.

The number of active connections and new connections are determined based upon the number of connections that the executor has forwarded within the last cycle of the last manager function. The manager cycle is two seconds, by default.

Configuring server weights

Under normal circumstances, Load Balancer uses all of the values that have proportions that are not zero to calculate the new weight. For example, if the proporations are 40 40 20 0, the active connections and new connections are 40% of the weight calculation each and the port load is 20%.

As an example, assume the manager function returns the following values:

```
             ACTV      NEWC       PORT      SYS
Server1       50        200        25        0
Server2       25        100        50        0
```

The initial weight calculations will be:

- Server1 = .40(50) + .40(200) + .2(25) = 20 + 80 + 5 = 105
- Server2 = .40(25) + .40(100) + .2(50) = 10 + 40 + 10 = 60

The initial weights are scaled to be proportional to the weightbound for the cluster:port. By default, the weightbound is 10. Thus, in the previous example, the final weights, which are rounded to the nearest whole number, are:

- Server1 = (105/165) * 10 = 6
- Server2 = (60/165) * 10 = 4

The calculated weight is shown as the NEW weight in the manager report. The weight is only pushed to the executor function if it exceeds the sensitivity level that is configured for the cluster:port combination. The NOW weight represents the weight that is obtained from the executor at the start of this manager cycle.

If the port load or the system load is -1, and the respective proportion for the port or system column is greater than 0, the calculated weight is zero (0). Zero (0) indicates that the server is not active and new requests are not sent to the server.

If you quiesce a server, you will see that the weight is also shown as zero (0), but the port load is positive if the server is still online. If a quiesced server goes offline, the port load is -1.

If a user issues a ″server down″ function on a server to prevent Load Balancer from sending requests to that server, the weight is -1 regardless of the value for the port load and system load.

Optimizing the manager interval

To optimize overall performance, the manager is restricted in how often it can interact with the executor. You can make changes to this interval by entering the dscontrol manager interval and dscontrol manager refresh commands.

The manager interval specifies how often the manager will update the server weights that the executor uses in routing connections. If the manager interval is too low, it can mean poor performance as a result of the manager constantly interrupting the executor. If the manager interval is too high, it can mean that the executor's request routing will not be based on accurate, up-to-date information.

The manager refresh cycle specifies how often the manager will ask the executor for status information. The refresh cycle is based on the interval time.

1. Set the manager interval time, in seconds. For example, to set the manager interval to 1 second, enter the following command:

   ```
   dscontrol manager interval 1
   ```

2. Set the manager refresh time, in seconds. For example, to set the manager refresh cycle to 3, enter the following command:

   ```
   dscontrol manager refresh 3
   ```

   This will cause the manager to wait for 3 intervals before asking the executor for status.

Tuning the proportion of importance given to status information

The manager uses ratios to determine the importance of status information coming from advisors and Load Balancer. You can change the default ratios that the manager uses to weight this information.

The manager can use some or all of the following external factors in its weighting decisions:

- *Active connections:* The number of active connections on each load balanced server machine (as tracked by the executor).
- *New connections:* The number of new connections on each load balanced server machine (as tracked by the executor).
- *Port-specific:* The input from advisors listening on the port.
- *System metric:* The input from the system monitoring tools, such as Metric Server or WLM.

Along with the current weight for each server and some other information required for its calculations, the manager gets the first two values (active and new connections) from the executor. These values are based on information that is generated and stored internally in the executor.

You can change the relative proportion of importance of the four values on a per cluster basis. Think of the proportions as percentages; the sum of the relative proportions must equal 100%. The default ratio is 50/50/0/0, which ignores the advisor and system information. In your environment, you may need to try different proportions to find the combination that gives the best performance.

Note:

- When adding an advisor (other than WLM), if the port proportion is zero, then the manager increases this value to 1. Because the sum of the relative proportions must total 100, the highest value is then decreased by 1.
- When adding the WLM advisor, if the system metric proportion is zero, then the manager increases this value to 1. Because the sum of the relative proportions must total 100, the highest value is then decreased by 1.

The number of active connections is dependent upon the number of clients as well as the length of time necessary to use the services that are being provided by the load balanced server machines. If the client connections are quick (such as small Web pages served using HTTP GET), then the number of active connections are fairly low. If the client connections are slower (such as a database query), then the number of active connections are higher.

You should avoid setting active and new connections proportions values too low. You will disable load balancing and smoothing unless you have these first two values set to at least 20 each.

To set the proportion of importance that is given to the different factors, use the "dscontrol cluster" on page 142 command. For example:

```
dscontrol cluster set cluster proportions value
```

See the topic on the "dscontrol cluster" on page 142 command for more information.

Managing traffic with server weights

Weights are applied to all servers on a port. For any particular port, the requests are distributed between servers based on their weights relative to each other. For example, if one server is set to a weight of 10, and the other to 5, the server set to 10 should get twice as many requests as the server set to 5.

Weights are set by the manager function based upon internal counters in the executor, feedback from the advisors, and feedback from a system-monitoring program, such as Metric Server. If you want to set weights manually while running the manager, specify the fixedweight option on the dscontrol server command. For a description of the fixedweight option, see "dscontrol manager" on page 149.

The maximum weight boundary affects how much difference there can be between the number of requests each server will get. If you set the maximum weightbound to 1, then all the servers can have a weight of 1, 0 if quiesced, or -1 if marked down. As you increase this number, the difference in how servers can be weighted is increased. At a maximum weightbound of 2, one server could get twice as many requests as another. At a maximum weightbound of 10, one server could get 10 times as many requests as another. The default maximum weightbound is 20.

If an advisor finds that a server has gone down, it tells the manager, which sets the weight for the server to zero. As a result, the executor will not send any additional

connections to that server as long as that weight remains zero. If there were any active connections to that server before the weight changed, they will be left to complete normally.

If all the servers are down, the manager sets the weights to half the weightbound.

- To specify the maximum weight boundary that any server can have, use the following command:

  ```
  dscontrol port set port weightbound weight
  ```

- Configure fixed weights for servers.
  1. Turn on the fixedweight option. For more information, see "dscontrol server" on page 158.

     ```
     dscontrol server set cluster@port@server fixedweight yes
     ```

     The server weight value remains fixed while the manager is running until you issue another dscontrol server command with fixedweight set to no.
  2. After fixedweight is set to yes, use the dscontrol server set weight command to set the weight to the value you desire. For example:

     ```
     dscontrol server set cluster@port@server weight value
     ```

Related reference

"dscontrol manager" on page 149
You can control the manager function with the `dscontrol manager` command.

Related information

Tuning

Optimizing the sensitivity threshold

To work at top speed, updates to the weights for the servers are only made if the weights have changed significantly. Constantly updating the weights when there is little or no change in the server status could create unnecessary overhead.

When the percentage weight change for the total weight for all servers on a port is greater than the sensitivity threshold, the manager updates the weights used by the executor to distribute connections. Consider, for example, that the total weight changes from 100 to 105. The change is 5%. With the default sensitivity threshold of 5, the manager will not update the weights used by the executor, because the percentage change is not above the threshold. If, however, the total weight changes from 100 to 106, the manager will update the weights.

Note: In most cases, you will not need to change this value.

To set the manager's sensitivity threshold to a value other than the default (for example, 6), enter the following command:

```
dscontrol manager sensitivity 6
```

Optimizing the smoothing index

The smoothing index limits the amount that a server's weight can change, effectively smoothing the change in the distribution of requests.

The manager calculates the server weights dynamically. As a result, an updated weight can be very different from the previous one. Under most circumstances, this will not be a problem. Occasionally, however, it may cause an oscillating effect in the way the requests are load balanced. For example, one server can end up receiving most of the requests due to a high weight. The manager will see that the

server has a high number of active connections and that the server is responding slowly. It will then shift the weight over to the free servers and the same effect will occur there too, creating an inefficient use of resources.

To alleviate this problem, the manager uses a smoothing index. A higher smoothing index will cause the server weights to change less drastically. A lower index will cause the server weights to change more drastically. The default value for the smoothing index is 1.5. At 1.5, the server weights can be rather dynamic. An index of 4 or 5 will cause the weights to be more stable.

Note: In most cases, you will not need to change this value.

Set the smoothing index, in seconds. For example, to set the smoothing index to 4 enter the following command:

```
dscontrol manager smoothing 4
```

Controlling connection records with the staletimeout value

Connections are considered stale when there has been no activity on that connection for the number of seconds specified in stale timeout. When the number of seconds has been exceeded with no activity, Load Balancer will remove that connection record from its tables, and subsequent traffic for that connection is discarded. The staletimeout command controls the way Load Balancer handles idle connections and the associated connection records.

Use the staletimeout command to control the period during which Load Balancer should keep connections in the ″Established″ state and accept traffic when no active traffic has been seen in the Dispatcher tables.

A client sends a FIN packet after it has sent all its packets so that the server will know that the transaction is finished. When Dispatcher receives the FIN packet, it marks the transaction from active state to FIN state. When a transaction is marked FIN, the memory that is reserved for the connection can be cleared.

To change the staletimeout value, use the dscontrol executor set command. Type the following at a command prompt:

```
dscontrol executor set staletimeout time
```

where the value for *time* is in seconds.

Note: Some services might have staletimeout values of their own.

Note: For example, LDAP (Lightweight Directory Access Protocol) has a configuration parameter called idletimeout. When idletimeout seconds have been exceeded, an idle client connection will be forcibly closed. Idletimeout may also be set to 0, which means that the connection will never be forcibly closed.

Connectivity problems can occur when Load Balancer's stale timeout value is smaller than the service's timeout value. In the case of LDAP, the Load Balancer staletimeout value defaults to 300 seconds. If there is no activity on the connection for 300 seconds, Load Balancer will remove the connection record from its tables. If the idletimeout value is larger than 300 seconds (or set to 0), the client may still believe that it has a connection to the server. When the client sends packets, the packets will be discarded by Load Balancer. This causes LDAP to hang when a request is made to the server.

To avoid this problem, set the LDAP idletimeout to a nonzero value that is the same or smaller than the Load Balancer staletimeout value.

Chapter 6. Troubleshooting Load Balancer

Use the information that is provided to help you solve problems that can occur in Load Balancer.

Click a link in the table to go to a full description and possible solution for the problem that you are experiencing.

Table 11. Troubleshooting table for Load Balancer

Symptom	Possible Cause	Go to...
Dispatcher not running correctly	Conflicting port numbers	"Problem: Load Balancer will not run" on page 111
Connections from client machines not being served or connections timing out	• Wrong routing configuration • Server does not have loopback device aliased to the cluster address • Extra route not deleted • Port not defined for each cluster	"Problem: Load Balancer requests are not being balanced" on page 111
Server not serving requests (Windows® platform)	An extra route has been created in the routing table	"Problem: Extra routes (Windows 2000)" on page 111
Dispatcher, Microsoft® IIS, and SSL are not working or will not continue	Unable to send encrypted data across protocols	"Problem: Dispatcher, Microsoft IIS, and SSL do not work (Windows platform)" on page 112
The dscontrol or lbadmin command fails with 'Server not responding' or 'unable to access RMI server' message	1. Commands fail due to socksified stack. Or commands fail due to not starting dsserver 2. RMI ports are not set correctly 3. Host file has incorrect local host	"Problem: dscontrol or lbadmin command fails" on page 112
Advisors not working correctly	Advisors are not running	"Problem: Advisors not working correctly" on page 112
"Cannot Find the File..." error message, when running Netscape as default browser to view online help (Windows platform)	Incorrect setting for HTML file association	"Problem: "Cannot find the file..." error message when trying to view online Help (Windows platform)" on page 113
Graphical user interface does not start correctly	Insufficient paging space	"Problem: Graphical user interface (GUI) does not start correctly" on page 113
Graphical user interface does not display correctly.	Resolution is incorrect.	"Problem: Graphical user interface (GUI) does not display correctly" on page 113

Table 11. Troubleshooting table for Load Balancer (continued)

Symptom	Possible Cause	Go to...
Help panels sometimes disappear behind other windows	Java™ limitation	"Problem: On Windows platform, help windows sometimes disappear behind other open windows" on page 113
GUI hangs (or unexpected behavior) when trying to load a large configuration file.	Java does not have access to enough memory to handle such a large change to the GUI	"Problem: GUI hangs (or unexpected behavior) when trying to load a large configuration file" on page 113
Korean Load Balancer interface displays overlapping or undesirable fonts on AIX® and Linux® systems	Default fonts must be changed	"Problem: Korean Load Balancer interface displays overlapping or undesirable fonts on AIX and Linux systems" on page 114
Unexpected GUI behavior when using Windows platform paired with Matrox AGP video card	Problem occurs when using Matrox AGP video cards while running the Load Balancer GUI	"Problem: On Windows platform, unexpected GUI behavior when using Matrox AGP video cards" on page 115
Slow response time when running commands on the Dispatcher machine	Slow response time can be due to machine overloading from a high volume of client traffic	"Problem: Slow response time running commands on Dispatcher machine" on page 115
SSL or HTTPS advisor not registering server loads	Problem occurs because the SSL server application not configured with the cluster IP address	"Problem: SSL or HTTPS advisor not registering server loads" on page 115
Socket pooling is enabled and the Web server is binding to 0.0.0.0	Configure the Microsoft IIS server to be bind specific	"Problem: Socket pooling is enabled and the Web server is binding to 0.0.0.0" on page 115
On Windows platform, corrupted Latin-1 national characters appear in command prompt	Change font properties of command prompt window	"Problem: On Windows systems, corrupted Latin-1 national characters appear in command prompt window" on page 116
On Windows platform, advisors and reach targets mark all servers down	Task offloading is not disabled or may need to enable ICMP.	"Problem: On Windows systems, advisors and reach targets mark all servers down" on page 116
On Windows platform, advisors not working in a high availability setup after a network outage	When the system detects a network outage, it clears its Address Resolution Protocol (ARP) cache	"Problem: On Windows systems, after network outage, advisors not working in a high availability setup" on page 117
On Linux systems, "IP address add" command and multiple cluster loopback aliases are incompatible	When aliasing more than one address on the loopback device, should use **ifconfig** command, not **ip address add**	"Problem: On Linux systems, do not use "IP address add" command when aliasing multiple clusters on the loopback device" on page 117

Table 11. Troubleshooting table for Load Balancer (continued)

Symptom	Possible Cause	Go to...
On Solaris systems, Load Balancer processes end when you exit the terminal session window from which they started	Use the **nohup** command to prevent the processes that you started from receiving a hangup signal when you exit the terminal session.	"Problem: On Solaris systems, Load Balancer processes end when you exit the terminal window from which they started" on page 117
Slow down occurs when loading Load Balancer configurations	The delay might be due to Domain Name System (DNS) calls that are made to resolve and verify the server address.	"Problem: Delay occurs while loading a Load Balancer configuration" on page 118
On Windows systems, the following error message appears: There is an IP address conflict with another system on the network	If high availability is configured, cluster addresses may be configured on both machines for a brief period which causes this error message to appear.	"Problem: On Windows systems, an IP address conflict error message appears" on page 118
On Windows systems, ″Server not responding″ error occurs when issuing a dscontrol or lbadmin command	When more than one IP address exists on a Windows system and the host file does not specify the address to associate with the hostname.	"Problem: On Windows systems, ″Server not responding″ error occurs when issuing dscontrol or lbadmin" on page 118
Dispatcher MAC forwarding configuration limitations with zSeries and S/390 platforms	On Linux, there are limitations when using zSeries or S/390 servers that have Open System Adapter (OSA) cards. Possible workarounds are provided.	"Problem: On Linux, Dispatcher configuration limitations when using zSeries or S/390 servers that have Open System Adapter (OSA) cards" on page 119
On Linux systems, iptables can interfere with the routing of packets	Linux iptables can interfere with load balancing of traffic and must be disabled on the Load Balancer machine.	"Problem: Linux iptables can interfere with the routing of packets" on page 120
On Solaris systems, when you try to configure an IPv6 server on the Dispatcher machine, the message ″unable to add server″ appears	This can be caused by the way the Solaris operating system handles the ping request for an IPv6 address.	"Problem: Unable to add an IPv6 server to the Load Balancer configuration on Solaris systems" on page 121
A Java fileset warning message appears when installing service fixes or installing natively, using system packaging tools	The product installation consists of several packages which are not required to be installed on the same machine, so each of these packages installs a Java fileset. When installed on the same machine a warning messages stating that the Java fileset is also owned by another fileset.	"Problem: Java warning message appears when installing service fixes" on page 121

Table 11. Troubleshooting table for Load Balancer (continued)

Symptom	Possible Cause	Go to...
Upgrading the Java fileset provided with the Load Balancer installations	If a problem is found with the Java file set, you should report the problem to IBM Service so that you can receive an upgrade for the Java file set that was provided with the Load Balancer installation.	"Upgrading the Java file set provided with the Load Balancer installation" on page 121
Client requests fail when forwarding to HP-UX back-end servers	After setting up Load Balancer for IPv6 on the HP-UX operating system, client requests to the cluster address fail. This error is a result of the interaction between the neighbor discovery function for the operating system and the Load Balancer.	"Problem: Client requests fail when using IPv6 MAC forwarding with HP-UX back-end servers" on page 121
Load Balancer for IPv4 and IPv6 conflicts with IP security (IPsec)	If you are using the Load Balancer for IPv4 and IPv6 with IP security (IPsec) enabled, output packets might be incorrect and dispatcher configuration information might display incorrectly in the command line interface and administrative console for WebSphere Application Server. Load Balancer reports that it is forwarding connections, but clients do not receive responses.	"Problem: On AIX systems, Load Balancer conflicts with IP security (IPsec)" on page 122
Install program will not run on the 32-bit Linux operating system for zSeries	Installing Load Balancer using ./install on the 32-bit Linux operating system for zSeries produces a "JVM Not Found" message.	"Problem: Installing WebSphere Edge Server using ./install on the 32-bit Linux operating system for zSeries produces a "JVM Not Found" message" on page 122
The uninstall process does not complete successfully on Linux operating systems	The uninstall process for WebSphere Edge Server hangs on Linux operating systems.	"Problem: The uninstall process for WebSphere Edge Server hangs on Linux operating systems" on page 122

Table 11. Troubleshooting table for Load Balancer (continued)

Symptom	Possible Cause	Go to...
The serverUp script might run when you issue commands for Load Balancer that affect the status of servers	You might experience problems if you run a command that affects the status of a server, such as the **dscontrol server up** and **dscontrol server down** commands, after a manager cycle has already retrieved the weights of the servers. If you run these commands, it might overwrite the values that are saved during the manager cycle and cause the serverUp script to run unexpectedly.	"Problem: The serverUp script might run when you issue commands for Load Balancer that affect the status of servers" on page 123

Problem: Load Balancer will not run

This problem can occur when another application is using one of the ports used by the Load Balancer. For more information, go to "Configuring the Load Balancer machine" on page 34.

Problem: Load Balancer requests are not being balanced

This problem has symptoms such as connections from client machines not being served or connections timing out. Check the following to diagnose this problem:

1. Have you configured the nonforwarding address, clusters, ports, and servers for routing? Check the configuration file.
2. Does the loopback device on each server have the alias set to the cluster address?

   Linux Use `netstat -ni` to check.
3. Is the extra route deleted?

   Linux Use `netstat -nr` to check.
4. Use the **dscontrol cluster status** command to check the information for each cluster you have defined. Make sure you have a port defined for each cluster.
5. Use the **dscontrol server report :** command to make sure that your servers are neither down nor set to a weight of zero.

Problem: Extra routes (Windows 2000)

After setting up server machines, you may find that you have inadvertently created one or more extra routes. If not removed, these extra routes will prevent the Load Balancer from operating.

Problem: Dispatcher, Microsoft IIS, and SSL do not work (Windows platform)

When using Dispatcher, Microsoft IIS, and SSL, if they do not work together, there may be a problem with enabling SSL security. For more information about generating a key pair, acquiring a certificate, installing a certificate with a key pair, and configuring a directory to require SSL, see the *Microsoft Information and Peer Web Services* documentation.

Problem: dscontrol or lbadmin command fails

1. The dscontrol command returns: **Error: Server not responding**. Or, the lbadmin command returns: **Error: unable to access RMI server**. These errors can result when your machine has a socksified stack. To correct this problem, edit the socks.cnf file to contain the following lines:

   ```
   EXCLUDE-MODULE java
   EXCLUDE-MODULE javaw
   ```

2. The administration consoles for Load Balancer interfaces (command line, graphical user interface, and wizards) communicate with dsserver using remote method invocation (RMI). The default communication uses three ports; each port is set in the dsserver start script:
   - 10099 to receive commands from dscontrol
   - 10004 to send metric queries to Metric Server
   - 10199 for the RMI server port

   This can cause problems when one of the administration consoles runs on the same machine as a firewall or through a firewall. For example, when Load Balancer runs on the same machine as a firewall, and you issue dscontrol commands, you might see errors such as **Error: Server not responding**.

   To avoid this problem, edit the dsserver script file to set the port used by RMI for the firewall (or other application). Change the line: LB_RMISERVERPORT=10199 to LB_RMISERVERPORT=*yourPort*. Where *yourPort* is a different port.

   When complete, restart dsserver and open traffic for ports 10099, 10004, 10199, and 10100, or for the chosen port for the host address from which the administration console will be run.
3. These errors can also occur if you have not already started **dsserver**.
4. If there are multiple adapters on the machine, you must designate which adapter that dsserver is to use by adding the following in the dsserver script:`java.rmi.server.hostname=<host_name or IPaddress>`

   For example: java -Djava.rmi.server.hostname=″10.1.1.1″

Problem: Advisors not working correctly

An ICMP ping is issued to the servers before the advisor request. If a firewall exists between Load Balancer and the servers, ensure that pings are supported across the firewall. If this setup poses a security risk to your network, modify the `java` statement in `dsserver` to turn off all pings to the servers by adding the `java` property:

```
LB_ADV_NO_PING="true"
java  -DLB_ADV_NO_PING="true"
```

Problem: "Cannot find the file..." error message when trying to view online Help (Windows platform)

For Windows platforms, when using Netscape as your default browser, the following error message may result: "Cannot find the file '<filename>.html' (or one of its components). Make sure the path and filename are correct and that all required libraries are available."

The problem is due to an incorrect setting for HTML file association. The solution is the following:

1. Click **My Computer**, click **Tools**, select **Folder Options**, and click **File Types** tab
2. Select "Netscape Hypertext Document"
3. Click **Advanced** button, select **open**, click **Edit** button
4. Enter *NSShell* in the **Application:** field (not the Application Used to Perform Action: field), and click **OK**

Problem: Graphical user interface (GUI) does not start correctly

The graphical user interface (GUI), which is lbadmin, requires a sufficient amount of paging space to function correctly. If insufficient paging space is available, the GUI might not start up completely. If this occurs, check your paging space and increase it if necessary.

Problem: Graphical user interface (GUI) does not display correctly

If you experience a problem with the appearance of the Load Balancer GUI, check the setting for the operating system's desktop resolution. The GUI is best viewed at a resolution of 1024x768 pixels.

Problem: On Windows platform, help windows sometimes disappear behind other open windows

On Windows platform, when you first open help windows, they sometimes disappear into the background behind existing windows. If this occurs, click on the window to bring it forward again.

Problem: GUI hangs (or unexpected behavior) when trying to load a large configuration file

When using lbadmin or Web administration (lbwebaccess) to load a large configuration file (roughly 200 or more **add** commands), the GUI may hang or display unexpected behavior, such as responding to screen changes at an extremely slow rate of speed.

This occurs because Java does not have access to enough memory to handle such a large configuration.

There is an option on the runtime environment that can be specified to increase the memory allocation pool available to Java.

The option is -Xmx*n* where *n* is the maximum size, in bytes, for the memory allocation pool. n must be a multiple of 1024 and must be greater than 2MB. The

value *n* may be followed by k or K to indicate kilobytes, or m or M to indicate megabytes. For example, `-Xmx128M` and `-Xmx81920k` are both valid. The default value is 64M. Solaris 8 has a maximum value of 4000M.

For example, to add this option, edit the lbadmin script file, modifying ″javaw″ to ″javaw -Xmx*n*″ as follows. For AIX systems, modify ″java″ to ″java -Xmx*n*″.

- **AIX systems**

```
java -Xmx256m -cp $LB_CLASSPATH $LB_INSTALL_PATH $LB_CLIENT_KEYS
com.ibm.internet.nd.framework.FWK_Main 1>/dev/null 2>&1 &
```

- **HP-UX systems**

```
java -Xmx256m -cp $LB_CLASSPATH $LB_INSTALL_PATH $LB_CLIENT_KEYS
com.ibm.internet.nd.framework.FWK_Main 1>/dev/null 2>&1 &
```

- **Linux systems**

```
javaw -Xmx256m -cp $LB_CLASSPATH $LB_INSTALL_PATH $LB_CLIENT_KEYS
com.ibm.internet.nd.framework.FWK_Main 1>/dev/null 2>&1 &
```

- **Solaris systems**

```
java -Xmx256m -cp $LB_CLASSPATH $LB_INSTALL_PATH $LB_CLIENT_KEYS
com.ibm.internet.nd.framework.FWK_Main 1>/dev/null 2>&1 &
```

- **Windows systems**

```
START javaw -Xmx256m -cp %LB_CLASSPATH% %LB_INSTALL_PATH%
 %LB_CLIENT_KEYS% com.ibm.internet.nd.framework.FWK_Main
```

There is no recommended value for *n* , but it should be greater than the default option. A good place to start would be with twice the default value.

Problem: Korean Load Balancer interface displays overlapping or undesirable fonts on AIX and Linux systems

To correct overlapping or undesirable fonts in the Korean Load Balancer interface:

- **On AIX systems**
  1. Stop all Java processes on the AIX system.
  2. Open the font.properties.ko file in an editor. This file is located in *home*/jre/lib where *home* is the Java home.
  3. Search for this string:

     ```
     -Monotype-TimesNewRomanWT-medium-r-normal
     --*-%d-75-75-*-*-ksc5601.1987-0
     ```

  4. Replace all instances of the string with:

     ```
     -Monotype-SansMonoWT-medium-r-normal
     --*-%d-75-75-*-*-ksc5601.1987-0
     ```

  5. Save the file.
- **On Linux systems**
  1. Stop all Java processes on the system.
  2. Open the font.properties.ko file in an editor. This file is located in *home*/jre/lib where *home* is the Java home.
  3. Search for this string (with no spaces):

     ```
     -monotype-
     timesnewromanwt-medium-r-normal--*-%d-75-75-p-*-microsoft-symbol
     ```

  4. Replace all instances of the string with:

     ```
     -monotype-sansmonowt-medium-r-normal--*-%d-75-75-p-*-microsoft-symbol
     ```

  5. Save the file.

Problem: On Windows platform, unexpected GUI behavior when using Matrox AGP video cards

On Windows platform when using a Matrox AGP card, unexpected behavior can occur in the Load Balancer GUI. When clicking the mouse, a block of space slightly larger than the mouse pointer can become corrupted causing possible highlighting reversal or images to shift out of place on the screen. Older Matrox cards have not shown this behavior. There is no known fix when using Matrox AGP cards.

Problem: Slow response time running commands on Dispatcher machine

If you are running the Dispatcher component for load balancing, it is possible to overload the computer with client traffic. The Load Balancer kernel module has the highest priority, and if it is constantly handling client packets, the rest of the system may become unresponsive. Running commands in user space may take a very long time to complete, or may never complete.

If this happens, you should begin to restructure your setup to avoid overloading the Load Balancer machine with traffic. Alternatives include spreading the load across several Load Balancer machines, or replacing the machine with a stronger and faster computer.

When trying to decide if the slow response time on the machine is due to high client traffic, consider whether this occurs during client peak traffic times. Misconfigured systems that cause routing loops can also cause the same symptoms. But before changing the Load Balancer setup, determine whether the symptoms may be due to high client load.

Problem: SSL or HTTPS advisor not registering server loads

Load Balancer will send packets to the servers using the cluster address that is aliased on the loopback. Some server applications (such as SSL) require that configuration information, such as certificates, are based on the IP address. The IP address must be the cluster address which is configured on the loopback in order to match the contents of the incoming packets. If the IP address of the cluster is not used when configuring the server application, then the client request will not get properly forwarded to the server.

Problem: Socket pooling is enabled and the Web server is binding to 0.0.0.0

When running Microsoft IIS server, version 5.0 on Windows back-end servers, you must configure the Microsoft IIS server to be bind specific. Otherwise, socket pooling is enabled as the default, and the Web server binds to 0.0.0.0 and listens for all traffic, rather than binding to the virtual IP addresses configured as multiple identities for the site. If an application on the local host goes down while socket pooling is enabled, AIX or Windows ND server advisors detect this; however, if an application on a virtual host goes down while the local host stays up, the advisors do not detect the failure and Microsoft IIS continues to respond to all traffic, including that of the downed application.

To determine whether socket pooling is enabled and the Web server is binding to 0.0.0.0, issue the following command:

```
netstat -an
```

The instructions for how to configure the Microsoft IIS server to be bind-specific (disable socket pooling), are located on the Microsoft Product Support Services Web site. You can also go to one of these URLs for this information:

IIS5: Hardware Load Balance Does Not Detect a Stopped Web Site (Q300509)
http://support.microsoft.com/default.aspx?scid=kb;en-us;Q300509

How to Disable Socket Pooling (Q238131)
http://support.microsoft.com/default.aspx?scid=kb;en-us;Q238131

Problem: On Windows systems, corrupted Latin-1 national characters appear in command prompt window

In a command prompt window on the Windows operating system, some national characters of the Latin-1 family might appear corrupted. For example, the letter ″a″ with a tilde may display as a pi symbol. To fix this, you must change the font properties of the command prompt window. To change the font, do the following:

1. Click the icon in the upper left corner of the command prompt window
2. Select Properties, then click the Font tab
3. The default font is Raster fonts; change this to Lucida Console and click OK

Problem: On Windows systems, advisors and reach targets mark all servers down

When configuring your adapter on a Load Balancer machine, you must ensure that the following two settings are correct for the advisor to work:

- Disable Task Offloading.
  - To disable Task offloading: Go to Start > Settings > Control Panel > Network and Dial-up Connections, then select the adapter.
  - In the pop-up window, click Properties.
  - Click Configure, then select the Advanced tab.
  - In the property pane, select the Task Offload property, then select disable in the value field.
- Enable Protocol 1 (ICMP) for IP protocols if you are enabling TCP/IP filtering. If ICMP is not enabled, the ping test to the back-end server will not succeed. To check whether ICMP is enabled:
  - Go to Start > Settings > Control Panel > Network and Dial-up Connections, then select the adapter.
  - In the pop-up window, click Properties.
  - From the components pane, select Internet Protocol (TCP/IP), then click Properties.
  - Click Advanced, then select the Options tab.
  - Select TCP/IP filtering in the options pane, then click Properties.
  - If you have selected **Enable TCP/IP Filtering** and **permit only** for IP protocols, you must add IP Protocol 1. This must be added in addition to the existing TCP and UDP ports that you enabled.

Problem: On Windows systems, after network outage, advisors not working in a high availability setup

By default, when the Windows operating system detects a network outage, it clears its address resolution protocol (ARP) cache, including all static entries. After the network is available, the ARP cache is repopulated by ARP requests sent on the network.

With a high availability configuration, both servers take over primary operations when a loss of network connectivity affects one or both. When the ARP request is sent to repopulate the ARP cache, both servers respond, which causes the ARP cache to mark the entry as not valid. Therefore, the advisors are not able to create a socket to the backup servers.

Preventing the Windows operating system from clearing the ARP cache when there is a loss of connectivity solves this problem. Microsoft has published an article that explains how to accomplish this task. This article is on the Microsoft Web site, located in the Microsoft Knowledge Base, article number 239924: http://support.microsoft.com/default.aspx?scid=kb;en-us;239924.

The following is a summary of the steps, described in the Microsoft article, to prevent the system from clearing the ARP cache:

1. Use the Registry editor (regedit or regedit32) to open the registry.
2. View the following key in the registry:
   `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters`
3. Add the following registry value: Value Name: DisableDHCPMediaSense Value Type: REG_DWORD.
4. After the key is added, edit the value and set it to 1.
5. Reboot the machine for the change to take effect.

Note: This affects the ARP cache regardless of the DHCP setting.

Problem: On Linux systems, do not use ″IP address add″ command when aliasing multiple clusters on the loopback device

Certain considerations must be taken when using Linux kernel 2.4.x servers. If the server has a cluster address configured on the loopback device using the **ip address add** command, only one cluster address can be aliased.

When aliasing multiple clusters to the loopback device use the **ifconfig** command, for example:

```
ifconfig lo:num clusterAddress netmask 255.255.255.255 up
```

Additionally, there are incompatibilities between the ifconfig method of configuring interfaces and the ip method of configuring interfaces. Best practice suggests that a site choose one method and use that method exclusively.

Problem: On Solaris systems, Load Balancer processes end when you exit the terminal window from which they started

On Solaris systems, after starting Load Balancer scripts (such as dsserver or lbadmin) from a terminal window, if you exit from that window, the Load Balancer process also exits.

To resolve this problem, start the Load Balancer scripts with the **nohup** command. For example: **nohup dsserver**. This command prevents the processes started from the terminal session from receiving a hangup signal from the terminal when it exits, allowing the processes to continue even after the terminal session has ended. Use the **nohup** command in front of any Load Balancer scripts that you want to continue to process beyond the end of a terminal session.

Problem: Delay occurs while loading a Load Balancer configuration

Loading a Load Balancer configuration might take a long time due to Domain Name System (DNS) calls that are made to resolve and verify the server address.

If the DNS of the Load Balancer machine is configured incorrectly, or if DNS in general takes a long time, this will cause a slow down in loading the configuration due to the Java processes that are sending DNS requests on the network.

A workaround for this is to add your server addresses and hostnames to your local `/etc/hosts` file.

Problem: On Windows systems, an IP address conflict error message appears

If high availability is configured, the cluster addresses may be configured on both machines for a brief period and cause the following error message to occur: There is an IP address conflict with another system on the network. In this case, you can safely ignore the message. It is possible for a cluster address to be briefly configured on both high availability machines at the same time, especially during startup of either machine, or when a takeover has been initiated.

Problem: On Windows systems, ″Server not responding″ error occurs when issuing dscontrol or lbadmin

When more than one IP address is on a Windows system and the **hosts** file does not specify the address to associate with the host name, the operating system chooses the smallest address to associate with the host name.

To resolve this problem, update the `c:\Windows\system32\drivers\etc\`**`hosts`** file with your machine host name and the IP address that you want to associate with the host name.

If you are using dscontrol, you can specify the connection address using the following command:

```
dscontrol host::<ip_address or host_name> <command>
```

IMPORTANT: The IP address cannot be a cluster address.

Problem: On Linux, Dispatcher configuration limitations when using zSeries or S/390 servers that have Open System Adapter (OSA) cards

In general, servers in the Load Balancer configuration must all be on the same network segment regardless of the platform. Active network devices such as router, bridges, and firewalls interfere with Load Balancer. This is because Load Balancer functions as a specialized router, modifying only the link-layer headers to its next and final hop. Any network topology in which the next hop is not the final hop is not valid for Load Balancer.

Note: Tunnels, such as channel-to-channel (CTC) or inter-user communication vehicle (IUCV), are often supported. However, Load Balancer must forward across the tunnel directly to the final destination, it cannot be a network-to-network tunnel.

There is a limitation for zSeries and S/390 servers that share the OSA card, because this adapter operates differently than most network cards. The OSA card has its own virtual link layer implementation, which has nothing to do with ethernet, that is presented to the Linux and z/OS hosts behind it. Effectively, each OSA card looks just like ethernet-to-ethernet hosts (and not to the OSA hosts), and hosts that use it will respond to it as if it is ethernet.

The OSA card also performs some functions that relate to the IP layer directly. Responding to ARP (address resolution protocol) requests is one example of a function that it performs. Another is that shared OSA routes IP packets based on destination IP address, instead of on ethernet address as a layer 2 switch. Effectively, the OSA card is a bridged network segment unto itself.

Load Balancer that runs on an S/390 Linux or zSeries Linux host can forward to hosts on the same OSA or to hosts on the ethernet. All the hosts on the same shared OSA are effectively on the same segment.

Load Balancer can *forward out* of a shared OSA because of the nature of the OSA bridge. The bridge knows the OSA port that owns the cluster IP. The bridge knows the MAC address of hosts directly connected to the ethernet segment. Therefore, Load Balancer can MAC-forward across one OSA bridge.

However, Load Balancer cannot forward into a shared OSA. This includes the Load Balancer on an S/390 Linux when the back-end server is on a different OSA card than the Load Balancer. The OSA for the back-end server advertises the OSA MAC address for the server IP, but when a packet arrives with the ethernet destination address of the server's OSA and the IP of the cluster, the server's OSA card does not know which of its hosts, if any, should receive that packet. The same principles that permit OSA-to-ethernet MAC-forwarding to work out of one shared OSA do not hold when trying to forward into a shared OSA.

Workaround:

In Load Balancer configurations that use zSeries or S/390 servers that have OSA cards, there are two approaches you can take to work around the problem that has been described.

1. Using platform features

   If the servers in the Load Balancer configuration are on the same zSeries or S/390 platform type, you can define point-to-point (CTC or IUCV) connections between Load Balancer and each server. Set up the endpoints with private IP

addresses. The point-to-point connection is used for Load Balancer-to-server traffic only. Then add the servers with the IP address of the server endpoint of the tunnel. With this configuration, the cluster traffic comes through the Load Balancer OSA card and is forwarded across the point-to-point connection where the server responds through its own default route. The response uses the server's OSA card to leave, which might or might not be the same card.

2. Using Load Balancer's encapsulation feature.

   If the servers in the Load Balancer configuration are not on the same zSeries or S/390 platform type, or if it is not possible to define a point-to-point connection between Load Balancer and each server, it is recommended that you use Load Balancer's encapsulation feature, which is a protocol that permits Load Balancer to forward across routers.

   When using encapsulation, the client->cluster IP packet is received by Load Balancer, encapsulated, and sent to the server. At the server, the original client->cluster IP packet is excapsulated, and the server responds directly to the client. The advantage with using GRE is that Load Balancer sees only the client-to-server traffic, not the server-to-client traffic. The disadvantage is that it lowers the maximum segment size (MSS) of the TCP connection due to encapsulation overhead.

   Refer to the topic "Use encapsulation forwarding to forward traffic across network segments" on page 87 for more information on how to configure Load Balancer to forward with encapsulation.

Problem: Linux iptables can interfere with the routing of packets

Linux iptables can interfere with load balancing of traffic and must be disabled on the Dispatcher machine.

Issue the following command to determine if iptables are loaded:

```
lsmod | grep ip_tables
```

The output from the preceding command might be similar to this:

```
ip_tables         22400   3
iptable_mangle,iptable_nat,iptable_filter
```

Issue the following command for each iptable listed in the output to display the rules for the tables:

```
iptables -t <short_name> -L
```

For example:

```
iptables -t mangle -L
iptables -t nat    -L
iptables -t filter -L
```

If iptable_nat is loaded, it must be unloaded. Because iptable_nat has a dependency on iptable_conntrack, iptable_conntrack also must be removed. Issue the following command to unload these two iptables:

```
rmmod iptable_nat iptable_conntrack
```

Problem: Unable to add an IPv6 server to the Load Balancer configuration on Solaris systems

On Solaris systems, when you try to configure an IPv6 server on a installation, the message `unable to add server` appears. This can be caused by the way the Solaris operating system handles the ping request for an IPv6 address.

On Solaris systems, when adding a server to the configuration, Load Balancer tries to ping the server to obtain the MAC address of the server. The Solaris machine might choose a configured cluster address as the source address of the ping request, instead of using the NFA address of the machine. If the cluster address is configured on the server loopback, the ping response is not received at the Load Balancer machine; therefore, it does not add the server to the configuration.

The solution is to configure another IPv6 address on the Load Balancer machine either before or after configuring the IPv6 cluster address. This address must be an address that is not aliased on the loopback of the back-end server on which you are trying to add to the Load Balancer configuration. Then add the server to the Load Balancer configuration.

Problem: Java warning message appears when installing service fixes

Load Balancer provides a Java file set along with the product installation. The product installation consists of several packages that are not required to be installed on the same machine. Examples of this are the Metric Server package, the administration package, and the base package. All of these code packages require a Java file set to operate but each of the three packages could be installed on separate machines. As such, each of these packages installs a Java file set. When installed on the same machine, the Java file set will be owned by each of these file sets. When you install the second and third Java file set, you will receive a warning messages stating that the Java file set is also owned by another file set.

When installing code using the native installation methods (for example, installp on AIX), you should ignore the warning messages that the Java file set is owned by another fileset.

Upgrading the Java file set provided with the Load Balancer installation

During the Load Balancer installation process, a Java file set also gets installed. Load Balancer will be the only application that uses the Java version which installs with the product. You should not upgrade this version of the Java file set on your own. If there are problem which requires an upgrade for the Java file set, you should report the problem to IBM Service so the Java file set which is shipped within Load Balancer will be upgraded with an official fix level.

Problem: Client requests fail when using IPv6 MAC forwarding with HP-UX back-end servers

After setting up Load Balancer for IPv6 on the HP-UX operating system, client requests to the cluster address fail. This error is a result of the interaction between the neighbor discovery function for the operating system and the Load Balancer.

When a back-end server is added to the configuration, Load Balancer tries to ping the new server for the MAC address. The HP-UX server might choose a configured cluster address as the source address of the ping request, instead of using the nonforwarding address (NFA) of the machine. In this case, a new entry is created for the cluster address in the routing table of the back-end HP-UX server in addition to the local loopback entry. The new entry has a higher routing priority than the local loopback. Thus, the client requests that reach the back-end server are then redirected back to the Load Balancer server, which does not respond.

To work around this problem, after Load Balancer is set up completely, configure the loopback alias on the back-end server as a final step. If the cluster address is aliased on the loopback when the Load Balancer configuration is loaded, remove the cluster loopback alias on the back-end server completely and then re-alias it. To bring down the loopback alias, use the `ifconfig lo0:1 inet6` command from the terminal window. Re-alias the loopback alias.

Problem: On AIX systems, Load Balancer conflicts with IP security (IPsec)

If you are using Load Balancer with IP security (IPsec) enabled, output packets might be incorrect and dispatcher configuration information might display incorrectly in the command line interface and administrative console for WebSphere Application Server. Load Balancer reports that it is forwarding connections, but clients do not receive responses.

If you are using Load Balancer function and IP security on the same host, there might be communication problems between Load Balancer and the application server. The Load Balancer component is not fully compatible with IPsec features and it transmits data from both sides of the security layer. Load Balancer receives packets below IPsec and, as a result, receives encrypted packets that it does not decrypt. When sending data, Load Balancer transmits them above IPsec, so it sends unencrypted packets to the application server that are encrypted on the other end by IPsec. The application server, therefore, receives encrypted data that cannot be used.

Problem: Installing WebSphere Edge Server using ./install on the 32-bit Linux operating system for zSeries produces a ″JVM Not Found″ message

This problem is caused by a limitation of the Edge Installer on zSeries 32-bit Linux operating systems.

You can work around this problem by performing a manual installation for 32-bit zSeries Linux operating system. See “Installing Load Balancer on Linux operating systems” on page 16 for more information.

Problem: The uninstall process for WebSphere Edge Server hangs on Linux operating systems

This problem is the result of a limitation in the Edge Installer on Linux operating systems.

To uninstall WebSphere Edge Server on a Linux operating system, you need to manually uninstall the product. To uninstall the entire product, enter the command:

```
rpm -e 'rpm -qa | grep ibmlb'
```

To uninstall an individual package, enter the command `rpm -e <package name>`. The package names can be found in "Installing Load Balancer on Linux operating systems" on page 16. Remember to uninstall the packages in the reverse order of which they were installed.

Problem: The serverUp script might run when you issue commands for Load Balancer that affect the status of servers

Weights are set by the manager during a manager cycle. At the start of the manager cycle, the manager retrieves the current weights from the executor function. The manager uses these values as the last known weight to determine if the status of a server has changed.

If you issue a server down command, for example, dscontrol server down, the executor function saves the current weight of the server and associates a new weight to the server with a value of -1. When you issue a server up command, for example, dscontrol server up, a call is made to the executor function to revert the weight of the server to the saved value. The system sets a flag to indicate that the server is no longer marked down by the user.

If the server up command occurs after the manager has retrieved the weights, the executor function overwrites the weight that is used to determine if the server state has changed. This process does not cause any side effects unless the server is also quiesced.

A quiesced server has a weight of 0, which is the same value as a server that is detected down by the advisor. If you run a server up command on a quiesced server, the executor function saves a value of 0 for the weight that determines if the state of the server has changed. When the server is unquiesced, the serverUp script might run because of this saved value.

The chances of experiencing this problem increase with larger configurations because the manager cycle takes longer to run. Also, there is a higher probability that the manager cycle will be in progress when the server up command is issued.

Chapter 7. Reference

Reference information is organized to help you locate particular facts quickly.

"Advanced configuration"
Settings are properties that you can configure using configuration files, or by other means.

"Commands" on page 137
Look up a command by its name to find detailed syntax and usage of the command.

"Examples" on page 161
Examples of code snippets, command syntax, and configuration values that are relevant to performing tasks with Load Balancer.

Advanced configuration

Settings are properties that you can configure using the configuration files, or by other means..

To open the information center table of contents to the location of this reference information, click the **Show in Table of Contents** button ( ) on your information center border.

Directory conventions

References in product information to *install_root* and other directories infer specific default directory locations. This topic describes the conventions in use for IBM WebSphere Edge Components.

These file paths are default locations. You can install the product and other components in any directory where you have write access. You can create profiles in any valid directory where you have write access.

Linux

install_root - the root directory in which the product was installed
Load Balancer install paths include the following:

- Administration - /opt/ibm/edge/ulb/admin
- Load Balancer - /opt/ibm/edge/ulb/servers
- Metric Server - /opt/ibm/edge/ulb/ms
- Documentation - /opt/ibm/edge/ulb/docs/

install_root - the root directory in which the product was installed
Load Balancer install paths include the following:

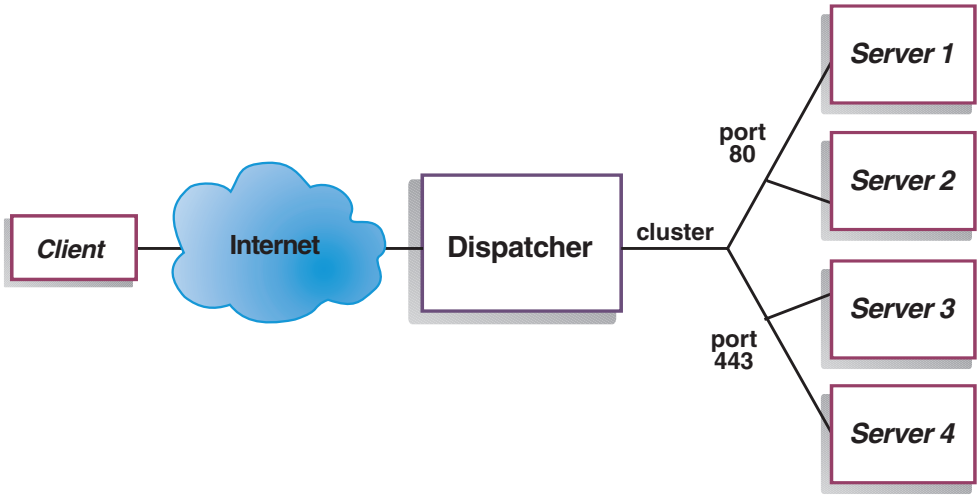
- Administration - C:\Program Files\IBM\edge\ulb\admin
- Load Balancer - C:\Program Files\IBM\edge\ulb\servers
- Metric Server - C:\Program Files\IBM\edge\ulb\ms
- Documentation - C:\Program Files\IBM\edge\ulb\docs

Types of cluster, port, and server configurations

There are many ways that you can configure Load Balancer to support your site.

1 cluster with 2 ports

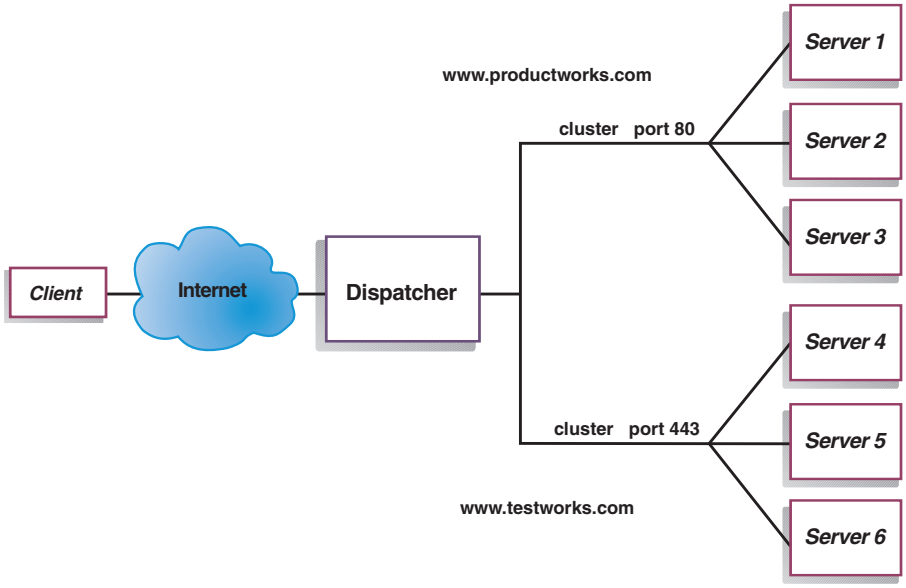
If you have only one host name for your site to which all of your customers will connect, you can define a single cluster of servers. For each of these servers, configure a port through which Load Balancer communicates.



In this example for the Dispatcher component, one cluster is defined at www.productworks.com. This cluster has two ports: port 80 for HTTP and port 443 for SSL. A client making a request to http://www.productworks.com (port 80) goes to a different server than a client requesting https://www.productworks.com (port 443).

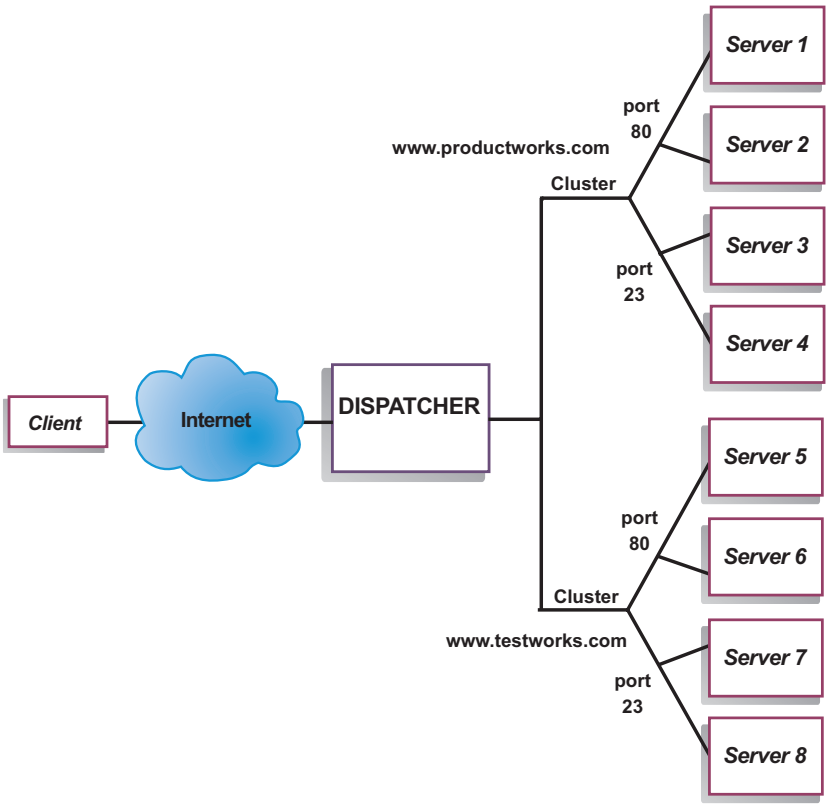
2 clusters, each with 1 port

Another way of configuring Load Balancer might be appropriate if you have a very large site with many servers dedicated to each protocol supported. In this case, you might want to define a cluster for each protocol with a single port but with many servers.



In this example for the Dispatcher component, two clusters are defined: www.productworks.com for port 80 (HTTP) and www.testworks.com for port 443 (SSL). A third way of configuring Load Balancer might be necessary if your site does content hosting for several companies or departments, each one coming into your site with a different URL. In this case, you might want to define a cluster for each company or department and then define any ports to which you want to receive connections at that URL, as shown in the configuration for 2 clusters, each with two ports.

2 clusters, each with 2 ports



In this example for the Dispatcher component, two clusters are defined with port 80 for HTTP and port 23 for Telnet for each of the sites at www.productworks.com and www.testworks.com.

Custom advisor methods and function calls

Use the following advisor methods and function calls in your custom advisors.

Be aware that custom advisors need to have all the required routines. Advisors must have the following base class methods:

- A constructor routine. The constructor calls the base class constructor.
- An ADV_AdvisorInitialize method. This method provides a way to perform additional steps after the base class completes its initialization.
- A getLoad routine. The base advisor class performs the socket opening; the getLoad function only needs to issue the appropriate send and receive requests to complete the advising cycle.

Constructor (provided by advisor base)

```
public <advisor_name> {
  String sName;
  String sVersion;
  int iDefaultPort;
  int iInterval;
  String sDefaultLogFileName;
  boolean replace
)
```

sName
The name of the custom advisor

sVersion
The version of the custom advisor.

iDefaultPort
The port number on which to contact the server if no port number is specified in the call.

iInterval
The interval at which the advisor will query the servers.

sDefaultLogFileName
This parameter is required but not used. The only acceptable value is a null string, ""

replace
Whether or not this advisor functions in replace mode. Possible values are the following:

- *true* – Replace the load calculated by the advisor base code with the value reported by the custom advisor.
- *false* – Add the load value reported by the custom advisor to the load value calculated by the advisor base code.

ADV_AdvisorInitialize() method

```
void ADV_AdvisorInitialize()
```

This method is provided to perform any initialization that might be required for the custom advisor. This method is called after the advisor base module starts. In many cases, including the standard advisors, this method is not used and its code

consists of a return statement only. This method can be used to call the "suppressBaseOpeningSocket()" on page 68 method, which is valid only from within this method.

In many cases, including the standard advisors, this method is not used and its code consists of a return statement only. You can use this method to call the suppressBaseOpeningSocket method, which is valid only from within the ADV_AdvisorInitialize method.

ADVLOG() method

The ADVLOG function allows a custom advisor to write a text message to the advisor base log file. The format follows:

```
void ADVLOG (int logLevel, String message)
```

This command has the following parameters:

logLevel
The status level at which the message is written to the log file. The advisor log file is organized in stages; the most urgent messages are given status level 0 and less urgent messages receive higher numbers. The most verbose type of message is given status level 5. These levels are used to control the types of messages that the user receives in real time (The dscontrol command is used to set verbosity). Catastrophic errors should always be logged at level 0.

message
The message to write to the log file. The value for this parameter is a standard Java string.

getAdvisorName function

The getAdvisorName function returns a Java string with the suffix portion of your custom advisor name. For example, for an advisor named ADV_cdload.java, this function returns the value cdload.

This function does not take parameters.

Note: It is not possible for this value to change during one instantiation of an advisor.

caller.getCurrentServerId()

The getCurrentServerId function returns a Java string which is a unique representation for the current server. Typically, this value changes each time you call your custom advisor, because the advisor base code queries all server machines in series.

This function takes no parameters.

caller.getCurrentClusterId()

The getCurrentClusterId function call returns a Java string which is a unique representation for the current cluster. Typically, this value changes each time you call your custom advisor, because the advisor base queries all clusters in series.

This function takes no parameters.

caller.getSocket()

The getSocket function call returns a Java socket which represents the socket opened to the current server for communication.

This function takes no parameters.

caller.getLatestLoad()

The getLatestLoad function allows a custom advisor to obtain the latest load value for a given server object. The load values are maintained in internal tables by the advisor base code and the manager daemon. This function call is useful if you want to make the behavior of one protocol or port dependent on the behavior of another. For example, you might use this function call in a custom advisor that disabled a particular application server if the Telnet server on that same machine was disabled.

The syntax is:

```
int caller.getLatestLoad (String clusterId, int port, String serverId)
```

The three arguments together define one server object.

This command has the following parameters:

clusterId
The cluster identifier of the server object for which to obtain the current load value. This argument must be a Java string.

port
The port number of the server object for which to obtain the current load value.

serverId
The server identifier of the server object for which to obtain the current load value. This argument must be a Java string. The return value is an integer.

- A positive return value represents the actual load value assigned for the object that was queried.
- The value -1 indicates that the server asked about is down.
- The value -2 indicates that the status of the server asked about is unknown.

caller.receive()

The receive function gets information from the socket connection. The syntax is:

```
caller.receive(StringBuffer *response)
```

This command has the following parameters:

response
This is a string buffer into which the retrieved data is placed. Additionally, the function returns an integer value with the following significance:

- 0 indicates data was sent successfully.
- A negative number indicates an error.

caller.send()

The send function uses the established socket connection to send a packet of data to the server, using the specified port. The syntax is as follows:

```
caller.send(String command)
```

This command has the following parameters:

command
This is a string containing the data to send to the server. The function returns an integer value with the following significance:

- 0 indicates data was sent successfully.
- A negative number indicates an error.

getLoad()

```
int getLoad( int iConnectTime; ADV_Thread *caller )
```

This function has the following parameters:

iConnectTime
The length of time, in milliseconds, that it took the connection to complete. This load measurement is performed by the advisor base code and passed to the custom advisor code, which can use or ignore the measurement when returning the load value. If the connection fails, this value is set to -1.

caller
The instance of the advisor base class where advisor base methods are provided.Function calls available to custom advisors: The methods, or functions, described in the following sections can be called from custom advisors. These methods are supported by the advisor base code. Some of these function calls can be made directly, for example, function_name(), but others require the prefix caller. Caller represents the base advisor instance that supports the custom advisor that is being executed.

getAdviseOnPort()

The getAdviseOnPort function returns the port number on which the calling custom advisor is running.

The return value is a Java integer (int), and the function does not take parameters.

Note: It is not possible for this value to change during one instantiation of an advisor.

getAdvisorName()

The getAdvisorName function returns a Java string with the suffix portion of your custom advisor's name. For example, for an advisor named ADV_cdload.java, this function returns the value cdload. This function takes no parameters. Note that it is not possible for this value to change during one instantiation of an advisor.

getInterval()

The getInterval function returns the advisor interval, that is, the number of seconds between advisor cycles. This value is equal to the default value set in the custom advisor's constructor, unless the value has been modified at run time by using the dscontrol command. The return value is a Java integer (int).

The function takes no parameters.

suppressBaseOpeningSocket()

The suppressBaseOpeningSocket function call allows a custom advisor to specify whether the base advisor code opens a TCP socket to the server on the custom advisor's behalf. If your advisor does not use direct communication with the server to determine its status, it might not be necessary to open this socket. This function call can be issued only once, and it must be issued from the "ADV_AdvisorInitialize() method" on page 65 routine.

The function takes no parameters.

Related tasks

"Creating a custom advisor" on page 61
A custom advisor is a small piece of Java code, provided as a class file, that is called by the Load Balancer base code to determine the load on a server. The base code provides all necessary administrative services, including starting and stopping an instance of the custom advisor, providing status and reports, recording history information in a log file, and reporting advisor results to the manager component.

Related reference

"Example: Sample advisor" on page 68
This is a sample advisor file called ADV_sample.

List of advisors

Advisors are agents within Load Balancer. Their purpose is to assess the health and loading of server machines. This list of advisors are already provided with Load Balancer, but you can also write a custom advisor to suit specific needs.

Table 12. List of advisors

Advisor Name	Description
connect	The connect advisor does not exchange any protocol-specific data with the server. It simply measures the time it takes to open and close a TCP connection with the server. This advisor is useful for server applications which use TCP, but with a higher-level protocol for which an IBM-supplied or custom advisor is not available.
Custom advisors	Dispatcher provides the ability for a customer to write a custom (customizable) advisor. This enables support for proprietary protocols (on top of TCP) for which IBM has not developed a specific advisor. For more information, see "Creating a custom advisor" on page 61.
db2	The DB2 advisor works in conjunction with the DB2 servers. Dispatcher has the built in capability of checking the health of DB2 servers without the need for customers to write their own custom advisors. The DB2 advisor communicates with the DB2 connection port only, not the Java connection port.

Table 12. List of advisors (continued)

Advisor Name	Description
dns	The DNS advisor opens a connection, sends a pointer query for DNS, waits for a response, closes the connection and returns the elapsed time as a load.
ftp	The FTP advisor opens a connection, sends a SYST request, waits for a response, closes the connection, and returns the elapsed time as a load.
http	The HTTP advisor opens a connection, sends a HEAD request by default, waits for a response connection, and returns the elapsed time as a load. See “Getting service-specific advice with the advisor request or response option” on page 56for more information on how to change the type of request sent by the HTTP advisor.
https	The HTTPS advisor is a ⌂heavyweight⌂ advisor for SSL connections. It performs a full SSL socket connection with the server. The HTTPS advisor opens an SSL connection, sends an HTTPS request, waits for a response, closes the connnection, and returns the elapsed time as a load. (See also the SSL advisor, which is a ⌂lightweight⌂ advisor for SSL connections.) **Note:** The HTTPS advisor has no dependency upon server key or certificate content, but they must not be expired.
imap	The IMAP advisor opens a connection, waits for an initial message from the server, sends a quit command, closes the connection, and returns the elapsed time as a load.
ldap	The LDAP advisor opens a connection, sends an anonymous BIND request, waits for a response, closes the connection, and returns the elapsed time as a load.
ldapuri	**Note:** The LDAP URI advisor allows you better gauge LDAP availability by processing a complete request to the LDAP server. The advisor: 1. Opens a connection. 2. Sends a BIND request, which is based on the advisorrequest field that you define on the server object. 3. Waits for a response. 4. Closes the connection. 5. Returns the elapsed time as a load. Read “Configuring the LDAP URI advisor” on page 57 for more information on configuring this advisor.

Table 12. List of advisors (continued)

Advisor Name	Description
nntp	The NNTP advisor opens a connection, waits for an initial message from the server, sends a quit command, closes the connection, and returns the elapsed time as a load.
ping	The ping advisor does not open a TCP connection with the servers, but instead reports whether the server responds to a ping. While the ping advisor may be used on any port, it is also designed for configurations using the wildcard port, over which multiple protocol traffic may be flowing. It is also useful for configurations using non-TCP protocols with their servers.
pop3	The POP3 advisor opens a connection, waits for an initial message from the server, sends a quit command, closes the connection, and returns the elapsed time as a load.
reach	The reach advisor pings its target machines. This advisor is also designed for the Dispatcher's high availability components to determine reachability of its reach targets. Its results flow to high availability component and do not appear in the manager report. Unlike the other advisors, the reach advisor starts automatically by the manager function of the Dispatcher component.
sip	The SIP advisor opens a connection, sends an OPTIONS request, waits for a response, closes the connection, and returns the elapsed time as a load. The SIP advisor that is supported runs on TCP only and requires an application to be installed on a server that responds to an OPTIONS request.
smtp	The SMTP advisor opens a connection, waits for an initial message from the server, sends a quit, closes the connection, and returns the elapsed time as a load.
ssl	The SSL advisor is a ⌂lightweight⌂ advisor for SSL connections. It does not establish a full SSL socket connection with the server. The SSL advisor opens a connection, sends an SSL CLIENT_HELLO request, waits for a response, closes the connection, and returns the elapsed time as a load. (See also the HTTPS advisor, which is a ⌂heavyweight⌂ advisor for SSL connections.) **Note:** The SSL advisor has no dependency upon key management or certificates.
ssl2http	The ssl2http advisor starts and advises on the servers listed under port 443, but the advisor will open a socket to the "mapport⌂ for HTTP requests.

Table 12. List of advisors (continued)

Advisor Name	Description
self	The self advisor collects load status information on back-end servers. You can use the self advisor when using Dispatcher in a two–tiered configuration, where the Dispatcher furnishes information from the self advisor to the top-tiered Load Balancer. The self advisor specifically measures the connections per second rate on back-end servers of the Dispatcher at the executor level. See rprf_selfadv2tier.dita for more information.
telnet	The Telnet advisor opens a connection, waits for an initial message from the server, closes the connection, and returns the elapsed time as a load.
was	The WAS (WebSphere Application Server) advisor works in conjunction with the WebSphere Application servers. Customizable sample files for this advisor are provided in the installation directory. For more information, see "Example: Implementing the WAS advisor" on page 79.
wlm	The WLM (Workload Manager) advisor is designed to work in conjunction with servers on OS/390 mainframes running the MVS™ Workload Manager (WLM) component. For more information, see "The Workload Management Advisor" on page 61.

Sample scripts to generate alerts and record server failure

Load Balancer provides user exits that trigger scripts that you can customize. You can create the scripts to perform automated actions, such as alerting an Administrator when servers are marked down by the manager or simply record the event of the failure.

Sample scripts, which you can customize, are in the *install_root*/servers/samples directory. In order to run the files, you must move them to the *install_root*/servers/bin directory and remove the ⌂sample⌂ file extension. The following sample scripts are provided:

- **serverDown** — a server is marked down by the manager.
- **serverUp** — a server is marked back up by the manager.
- **managerAlert** — all servers are marked down for a particular port.
- **managerClear** — at least one server is now up, after all were marked down for a particular port.

If all servers on a cluster are marked down (either by the user or by the advisors), the managerAlert (if configured) starts, and Load Balancer attempts to route traffic to the servers using a round-robin technique. The serverDown script does not start when the last server in the cluster is detected as offline. By design, Load Balancer attempts to continue to route the traffic in case a server comes back online and responds to the request. If Load Balancer instead dropped all traffic, the client

would receive no response. When Load Balancer detects that the first server of a cluster is back online, the managerClear script (if configured) starts, but the serverUp script (if configured) is not run until an additional server is brought back online.

Here are some considerations for using the serverUp and serverDown scripts:

- If you define the manager cycle to be less than 25% of the advisor time, false reports of servers up or down can result. By default, the manager runs every 2 seconds, but the advisor runs every 7 seconds. Therefore, the manager expects new advisor information within 4 cycles. However, removing this restriction (that is, defining the manager cycle to be greater than 25% of the advisor time) significantly decreases performance because multiple advisors can advise on a single server.
- When a server goes down, the serverDown script starts. However, if you issue a serverUp command, it is assumed that the server is up until the manager obtains new information from the advisor cycle. If the server is still down, the serverDown script runs again.

High Availability recovery strategy for failed servers

The recovery strategy dictates how Load Balancer behaves when one Dispatcher machine fails and there is another configured as a backup.

Two Dispatcher machines are configured: the primary machine, and a second machine called the backup. At startup, the primary machine sends all the connection data to the backup machine until that machine is synchronized. The primary machine becomes active, that is, it begins load balancing. The backup machine, meanwhile, monitors the status of the primary machine, and is said to be in standby state.

If the backup Load Balancer machine detects that the primary machine has failed, it performs a takeover load balancing functions and becomes the active machine. After the primary machine has once again become operational, the machines respond according to how the recovery strategy has been configured by the user.

There are two kinds of strategy:

- **Automatic** - The primary machine resumes routing packets as soon as it becomes operational again.
- **Manual** - intervention is required to return the primary machine to active state and reset the backup machine to standby. The manual recovery strategy allows you to force the routing of packets to a particular machine, using the takeover command. Manual recovery is useful when maintenance is being performed on the other machine

Note: The strategy parameter must be the same for both machines.

Scripts to run with high availability

Before using a script, remember that for Dispatcher to route packets, each cluster address must be aliased to a network interface device.

- In a stand-alone Dispatcher configuration, each cluster address must be aliased to a network interface card (for example, en0, tr0).
- In a high availability configuration:

  - On the active machine, each cluster address must be aliased to a network interface card (for example, en0, tr0)
  - On the standby machine, each cluster address must be aliased to a loopback device (for example, lo0).
  - To customize the scripts for certain situations, read Customizing the scripts for high availability.
- In any machine in which the executor has been stopped, all aliases should be removed to prevent conflicts with another machine that may be started. For information on aliasing the network interface card, see "Aliasing the network interface card or loopback device" on page 39.

The following sample script can be used:

- **HighAvailChange**

  The HighAvailChange script runs whenever the high availability state changes within the Dispatcher. You can create this script to use state change information, for instance, to alert an Administrator or simply record the event.

On Linux for S/390: Dispatcher issues a gratuitous ARP to move IP addresses from one Dispatcher to another. This mechanism is therefore tied to the underlying network type. When running Linux for S/390, Dispatcher can natively do high availability takeovers (complete with IP address moves) only on those interfaces which can issue a gratuitous ARP and configure the address on the local interface. This mechanism will not work properly on point-to-point interfaces such as IUCV and CTC and will not work properly in certain configurations of qeth/QDIO.

Related tasks

"Configuring high availability" on page 82
The high availability feature involves the use of a second Dispatcher machine. The first Dispatcher machine performs load balancing for all the client traffic as it does in a single Dispatcher configuration. The second Dispatcher machine monitors the "health" of the first, and takes over the task of load balancing if it detects that the first Dispatcher machine has failed.

Customizing the scripts for high availability
Customize your scripts to tune the performance of the high availability function for Load Balancer.

"Detecting server failures with heartbeats and reach targets" on page 85
Configure heartbeats and reach targets to detect server failures and control when failovers can occur.

Related reference

"High Availability recovery strategy for failed servers" on page 86
The recovery strategy dictates how Load Balancer behaves when one Dispatcher machine fails and there is another configured as a backup.

Commands

Look up a command by its name to find detailed syntax and usage of the command.

To open the information center table of contents to the location of this reference information, click the **Show in Table of Contents** button ( ) on your information center border.

Note: You need to follow different syntax conventions depending upon the location from which you issue the command. In most cases, you can use one of the following options:

- Issue the command from within the dscontrol shell. For example,

  ```
  enter dscontrol
  ```

  Press Enter, and you will enter the command shell.
- Issue the command through a configuration file.
- Enter the command from the command prompt. For example, enter:

  ```
  dscontrol rule add
  ```

The following syntax is applicable for any rule add command if you are using the dscontrol shell:

- If you are entering the command from the dscontrol shell or a configuration file, place quotation marks around the special characters, as shown in the reference topics. The following example is for a configuration file:

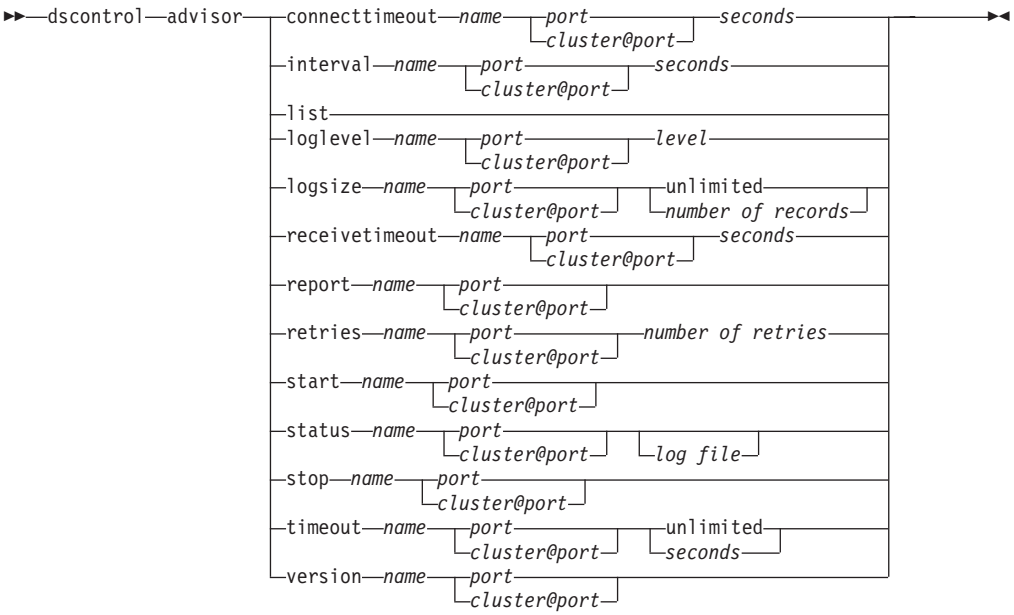
  ```
  dscontrol rule add 10.1.203.4@80@true type true
  ```
- If you are entering the same the same command from a Windows prompt, surround the entire command with quotation marks. For example:

  ```
  dscontrol "rule add 10.1.203.4@80@true type true"
  ```

dscontrol advisor

Use this command to control various features of the advisor function.

Syntax



Parameters

connecttimeout

Set how long an advisor waits before reporting that a connect to a server for a particular port on a server (a service) fails. For more information, see "Enabling advisors to manage load balancing" on page 50.

- *name*

The name of the advisor. Possible values include **connect**, **db2**, **dns**, **ftp**, **http**, **https**, **cachingproxy**, **imap**, **ldap**, **ldapuri**, **nntp**, **ping**, **pop3**, **self**, **sip**, **smtp**, **ssl**, **ssl2http**, **telnet**, and **wlm**.

See the topic "List of advisors" on page 53 for more information on the advisors that Load Balancer provides.

Names of customized advisors are of the format ADV_*xxxx*, where *xxxx* is the name of the class that implements the custom advisor. See "Creating a custom advisor" on page 61 for more information.

- *port*

  The number of the port that the advisor is monitoring.

- *cluster@port*

  The cluster value is optional on the advisor commands, but the port value is required. If the cluster value is not specified, then the advisor will start running on the port for all clusters. If you specify a cluster, then the advisor will start running on the port, but only for the cluster you have specified. See the topic "Enabling advisors to manage load balancing" on page 50 for more information on starting and stopping advisors.

  The cluster is the address in IP address format or symbolic name. The port is the number of the port that the advisor is monitoring.

- *seconds*

  A positive integer representing the timeout in seconds at which the advisor waits before reporting that a connect to a server fails. The default is 3 times the value specified for the advisor interval.

interval

Set how often the advisor will query the servers for information.

- *seconds*

  A positive integer representing the number of seconds between requests to the servers about their current status. The default is 7.

list

Show list of advisors that are currently providing information to the manager.

loglevel

Set the logging level for an advisor log.

- *level*

  The number of the level (0 to 5). The default is 1. The higher the number, the more information that is written to the advisor log. The following are the possible values: 0 is None, 1 is Minimal, 2 is Basic, 3 is Moderate, 4 is Advanced, 5 is Verbose.

logsize

Set the maximum size of an advisor log. When you set a maximum size for the log file, the file will wrap; when the file reaches the specified size, the subsequent entries are written from the top of the file, overwriting the previous log entries. Log size cannot be set smaller than the current size of the log. Log entries are time–stamped so you can tell the order in which they were written. The higher you set the log level, the more carefully you should choose the log size, because you can quickly run out of space when logging at the higher levels.

- *number of records*

  The maximum size in bytes for the advisor log file. You can specify either a positive number greater than zero, or the word unlimited. The log file may

not reach the exact maximum size before overwriting because the log entries themselves vary in size. The default value is 1 MB.

receivetimeout
Set how long an advisor waits before reporting that a receive from a particular port on a server (a service) fails. For more information, see "Enabling advisors to manage load balancing" on page 50.

- *seconds*

  A positive integer representing the timeout in seconds at which the advisor waits before reporting that a receive from a server fails. The default is 3 times the value specified for the advisor interval.

report
Display a report on the state of the advisor.

retry
Retry sets the number of retries that an advisor can make before marking a server down.

- *number of retries*

  An integer greater than or equal to zero. This value should be no larger than 3. If retries keyword is not configured, the number of retries defaults to zero.

start
Start the advisor. There are advisors for each protocol. The default ports are as follows:

Table 13. Default ports for advisors

Advisor Name	Protocol	Port
connect	ICMP	12345
db2	private	50000
dns	DNS	53
ftp	FTP	21 **Note:** The FTP advisor should advise only on the FTP control port (21). Do not start an FTP advisor on the FTP data port (20).
http	HTTP	80
https	SSL	443
imap	IMAP	143
ldap	LDAP	389
ldapuri	LDAP	389
nntp	NNTP	119
ping	PING	0
pop3	POP3	110
self	private	12345
sip	SIP	5060
smtp	SMTP	25
ssl	SSL	443
ssl2http	SSL	443
telnet	Telnet	23

Table 13. Default ports for advisors (continued)

Advisor Name	Protocol	Port
WLM	private	10007

- *log file*

  File name to which the management data is logged. Each record in the log is time–stamped.

  The default file is advisorname_port.log, for example, http_80.log. To change the directory where the log files are kept, see "Logging with Load Balancer" on page 92. The default log files for cluster (or site) specific advisors are created with the cluster address, for example, http_127.40.50.1_80.log.

status
Display the current status of all the values in an advisor that can be set globally and their defaults.

stop
Stop the advisor.

timeout
Set the number of seconds for which the manager will consider information from the advisor as valid. If the manager finds that the advisor information is older than this timeout period, the manager will not use that information in determining weights for the servers on the port the advisor is monitoring. An exception to this timeout is when the advisor has informed the manager that a specific server is down. The manager will use that information about the server even after the advisor information has timed out.

- *seconds*

  A positive number representing the number of seconds, or the word unlimited. The default value is unlimited.

version
Display the current version of the advisor.

Samples

- To start the http advisor on port 80 for cluster 127.40.50.1:

  ```
  dscontrol advisor start http 127.40.50.1@80
  ```

- To start the http advisor on port 88 for all clusters:

  ```
  dscontrol advisor start http 88
  ```

- To stop the http advisor at port 80 for cluster 127.40.50.1:

  ```
  dscontrol advisor stop http 127.40.50.1@80
  ```

- To set the time (30 seconds) an HTTP advisor for port 80 waits before reporting that a connect to a server fails:

  ```
  dscontrol advisor connecttimeout http 80 30
  ```

- To set the time (20 seconds) an HTTP advisor for port 80 on cluster 127.40.50.1 waits before reporting that a connect to a server fails:

  ```
  dscontrol advisor connecttimeout http 127.40.50.1@80 20
  ```

- To set the interval for the FTP advisor (for port 21) to 6 seconds:

  ```
  dscontrol advisor interval ftp 21 6
  ```

- To display the list of advisors currently providing information to the manager:

  ```
  dscontrol advisor list
  ```

- To change the log level of the advisor log to 0 for better performance:

  ```
  dscontrol advisor loglevel http 80 0
  ```

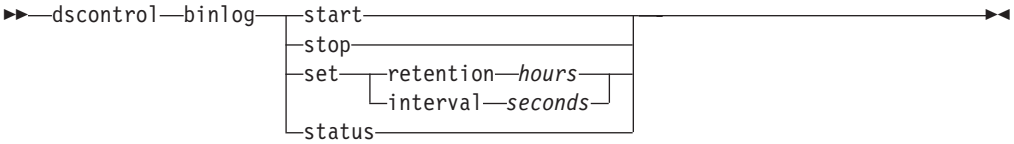
- To change the ftp advisor log size for port 21 to 5000 bytes:
  ```
  dscontrol advisor logsize ftp 21 5000
  ```
- To set the time (60 seconds) an HTTP advisor (for port 80) waits before reporting that a receive from a server fails:
  ```
  dscontrol advisor receivetimeout http 80 60
  ```
- To display a report on the state of the ftp advisor (for port 21):
  ```
  dscontrol advisor report ftp 21
  ```
- To display the current status of values associated with the http advisor for port 80:
  ```
  dscontrol advisor status http 80
  ```
- To set the timeout value for the ftp advisor information on port 21 to 5 seconds:
  ```
  dscontrol advisor timeout ftp 21 5
  ```
- To display the current version number of the ssl advisor for port 443:
  ```
  dscontrol advisor version ssl 443
  ```

  This command produces output similar to the following:
  ```
  Version: 04.00.00.00 - 07/12/2001-10:09:56-EDT
  ```

dscontrol binlog

You can control the settings and operation of the binary log file with the `dscontrol binlog` command.

Syntax



Parameters

start
Starts the binary log.

stop
Stops the binary log.

set

- **retention** *hours*

  The number of hours that binary log files are kept. The default value for retention is 24.
- **interval** *seconds*

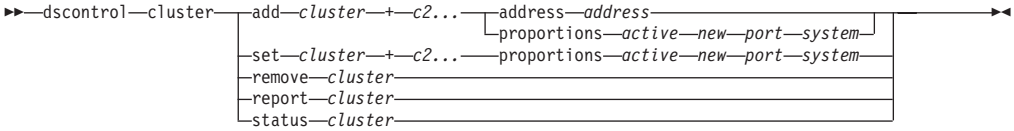
  The number of seconds between log entries. The default value for interval is 60.

status
Shows the retention and intervals of the binary log.

dscontrol cluster

Configure clusters and cluster properties with the dscontrol cluster command.

Syntax



Parameters

add

- *cluster*

  The cluster name or address to which clients connect. The cluster value is either a symbolic name or in IP address format.

  With the exception of the dscontrol cluster add command, you can use the colon (:) symbol to act as a wild card. For example, the following command,

  ```
  dscontrol cluster set : weightbound 80
  ```

  will result in setting a weightbound of 80 to all clusters.

  Note: Additional clusters are separated by a plus sign (+).

- **address** *address*

  The unique IP address of the TCP machine as either a host name or in IP address format. If the cluster value is unresolvable, you must provide the IP address of the physical machine.

- **proportions**

  At the cluster level, set the proportion of importance for active connections (active), new connections (new), information from any advisors (port), and information from a system monitoring program such as Metric Server (system) that are used by the manager to set server weights. Each of these values, described below, is expressed as a percentage of the total and they therefore always total 100. For more information see "Tuning the proportion of importance given to status information" on page 102.

  – *active*

    A number from 0–100 representing the proportion of weight to be given to the active connections. The default is 50.

  – *new*

    A number from 0–100 representing the proportion of weight to be given to new connections. The default is 50.

  – *port*

    A number from 0–100 representing the proportion of weight to be given to the information from the advisors. The default is 0.

    Note: When an advisor is started and if the port proportion is 0, Load Balancer automatically sets this value to 1 in order for the manager to use the advisor information as input for calculating server weight.

  – *system*

    A number from 0–100 representing the proportion of weight to be given to the information from the system metrics, such as from Metric Server. The default is 0.

set

Set the properties of the cluster.

remove
Remove this cluster.

report
Show the internal fields of the cluster.

status
Show current status of a specific cluster.

Samples

- To add cluster address 130.40.52.153:

```
dscontrol cluster add 130.40.52.153
```

- To remove cluster address 130.40.52.153:

```
dscontrol cluster remove 130.40.52.153
```

- To set the relative importance placed on input (active, new, port, system) received by the manager for servers residing on cluster 9.6.54.12:

```
dscontrol cluster set 9.6.54.12 proportions 60 35 5 0
```

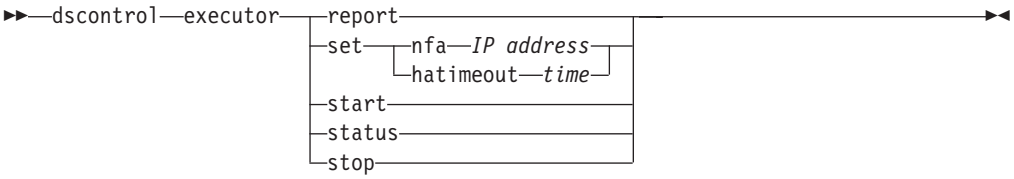
- To show the status for cluster address 9.67.131.167:

```
dscontrol cluster status 9.67.131.167
```

dscontrol executor

Control the executor function with the dscontrol executor command.

Syntax



Parameters

report
Display a statistics snapshot report.

set
Set the fields of the executor.

- **nfa** *IP address*

  Set the non-forwarding address. Any packet sent to this address will not be forwarded by the Dispatcher machine.

  The Internet Protocol address as either a symbolic name or in dotted decimal format.

- **hatimeout** *seconds*

  The number of seconds that the executor uses to timeout high availability heartbeats. The default value is 2.

start
Start the executor.

status
Display the current status of the values in the executor that can be set and their defaults.

stop
Stop the executor.

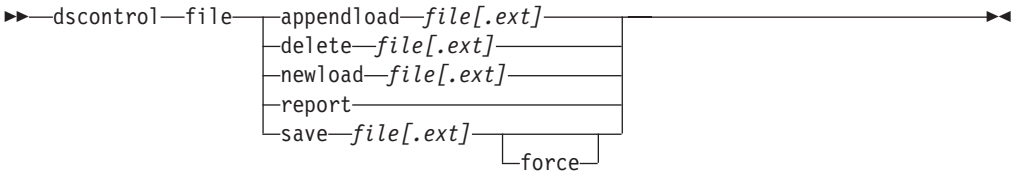
Samples

- To display the internal counters for Dispatcher:
  ```
  dscontrol executor status
  ```
- To set the non-forwarding address to 130.40.52.167:
  ```
  dscontrol executor set nfa 130.40.52.167
  ```
- To start the executor:
  ```
  dscontrol executor start
  ```
- To stop the executor:
  ```
  dscontrol executor stop
  ```

dscontrol file

Manage your configuration files with the dscontrol file command.

Syntax



Parameters

appendload
To update the current configuration, the appendload command runs the executable commands from your script file.

- *file[.ext]*

  A configuration file consisting of dscontrol commands. The file extension (.ext) can be anything you like and can be omitted.

delete
Delete the file.

newload
Loads and runs a new configuration file into the Load Balancer. The new configuration file replaces the current configuration.

report
Report on the available file or files.

save
Save the current configuration for Load Balancer to the file.

Note: Files are saved into and loaded from the *install_root*/servers/configurations/dispatcher directory.

- **force**

  To save your file to an existing file of the same name, use force to delete the existing file before saving the new file. If you do not use the force option, the existing file is not overwritten.

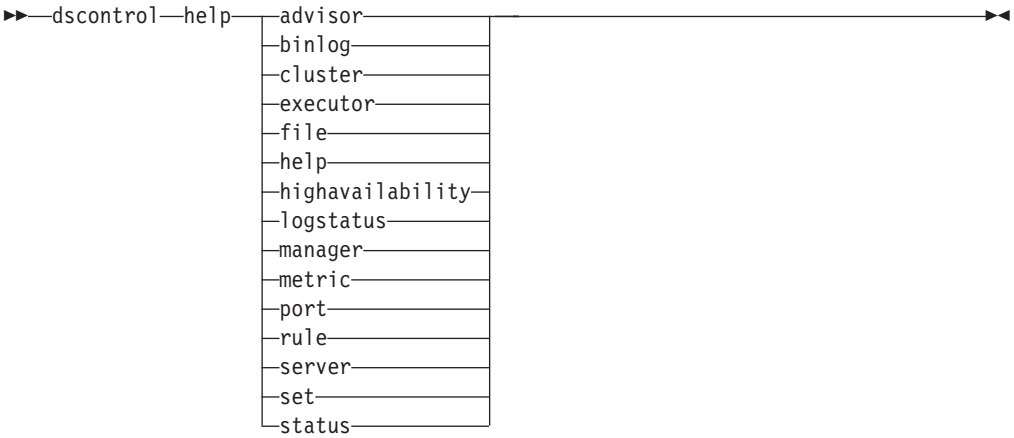
Samples

- To delete a file:
  ```
  dscontrol file delete file3
  ```
- To load a new configuration file to replace the current configuration:
  ```
  dscontrol file newload file1.sv
  ```
- To append a configuration file to the current configuration and load:
  ```
  dscontrol file appendload file2.sv
  ```
- To view a report of your files (files that you saved earlier):
  ```
  dscontrol file report
  ```
- To save your configuration into a file named file3:
  ```
  dscontrol file save file3
  ```

dscontrol help

Display or print help for any `dscontrol` command with the `dscontrol help` command.

Syntax



Sample

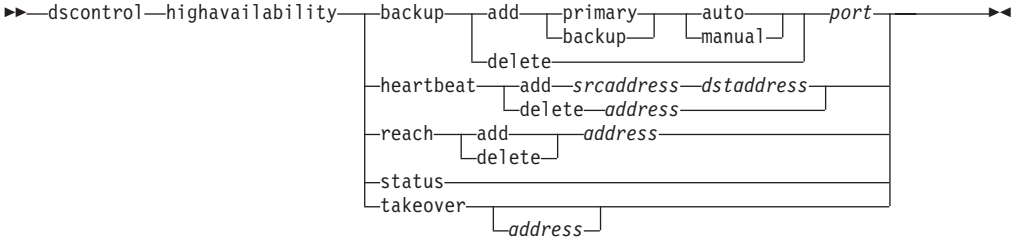
To get help on the dscontrol command:

```
dscontrol help
```

dscontrol highavailability

You can control high availability functions with the `dscontrol highavailability` command.

Syntax



Parameters

backup
Specify information for either the primary or backup machine.

- **add**

  Defines and runs the high availability functions for this machine.

  – **primary**

  Identifies the Dispatcher machine that has a primary role.

  – **backup**

  Identifies the Dispatcher machine that has a backup role.

  – **auto**

  Specifies an automatic recovery strategy, in which the primary machine will resume routing packets as soon as it comes back into service.

  – **manual**

  Specifies a manual recovery strategy, in which the primary machine does not resume routing packets until the administrator issues a takeover command.

- **delete**

  Removes this machine from high availability, so that it will no longer be used as a backup or primary machine.

- *port*

  An unused TCP port on both machines, to be used by Dispatcher for its heartbeat messages. The port must be the same for both the primary and backup machines.

heartbeat
Defines a communication session between the primary and backup Dispatcher machines.

- **add**

  Tell the source Dispatcher the address of its partner (destination address).

  – *source_address*

  Source address. The address (IP or symbolic) of this Dispatcher machine.

  – *destination_address*

  Destination address. The address (IP or symbolic) of the other Dispatcher machine.

  The *source_address* and *destination_address* must be the NFAs of the machines for at least one heartbeat pair.

- **delete** *address*

Removes the address pair from the heartbeat information. You can specify either the destination or source address of the heartbeat pair. The address (IP or symbolic) of either the destination or the source.

reach

Add or delete target address for the primary and backup Dispatchers, the reach advisor sends out pings from both the backup and the primary Dispatchers to determine how reachable their targets are.

- **add** *address*

  Adds a target address for the reach advisor. *address* is the IP address, format or symbolic, of the target node.

- **delete** *address*

  Removes a target address from the reach advisor. *address* is the IP address, format or symbolic, of the target node.

Note: When configuring the reach target, you must also start the reach advisor. The reach advisor starts automatically when you use the dscontrol manager reach command.

status

Return a report on high availability. Machines are identified as having one of three status conditions or states:

- **Active:** A given machine (either a primary, backup, or both) is routing packets.
- **Standby:** A given machine (either a primary, backup, or both) is not routing packets; it is monitoring the state of an active Dispatcher.
- **Idle:** A given machine is routing packets, and is not trying to establish contact with its partner Dispatcher.

takeover

Simple high availability configuration (role of the Dispatcher machines are either primary or backup).

Takeover instructs a standby Dispatcher to become active and to begin routing packets. This will force the currently active Dispatcher to become standby. The takeover command must be issued on the standby machine and works only when the strategy is manual. The substate must be synchronized.

- *address*

  The takeover address value is optional. It should only be used when the role of the machine is both primary and backup (mutual high availability configuration). The address specified is the NFA of the Dispatcher machine which normally routes this cluster's traffic. When there is a takeover of both clusters, specify the Dispatcher's own NFA address.

Note:

- The roles of the machines (primary and backup) do not change. Only their relative status (active or standby) changes.
- There are three possible takeover scripts, which are goActive, goStandby, and goInOp. See "Scripts to run with high availability" on page 86 for more information on these scripts.

Samples

- To check the high availability status of a machine:

```
dscontrol highavailability status
```

- To add the backup information to the primary machine using the automatic recovery strategy and port 80:

  ```
  dscontrol highavailability backup add primary auto 80
  ```

- To add an address that the Dispatcher must be able to reach:

  ```
  dscontrol highavailability reach add 9.67.125.18
  ```

- To add heartbeat information for the primary and backup machines

  ```
  Primary - highavailability heartbeat add 9.67.111.3 9.67.186.8

  Backup - highavailability heartbeat add 9.67.186.8 9.67.111.3
  ```

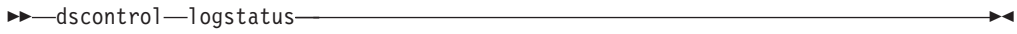
- To tell the standby Dispatcher to become active, forcing the active machine to become standby:

  ```
  dscontrol highavailability takeover
  ```

dscontrol logstatus

Use this command to display the log settings for a server.

Syntax



Parameters

There are no parameters for this command.

Sample

To display the log status:

```
dscontrol logstatus
```

dscontrol manager

You can control the manager function with the `dscontrol manager` command.

Syntax

Parameters

interval

Set how often the manager will update the weights of the servers to the executor, updating the criteria that the executor uses to route client requests.

- *seconds*

  A positive number representing in seconds how often the manager will update weights to the executor. The default is 2.

loglevel

Set the logging level for the manager log.

- *level*

  The number of the level (0 to 5). The higher the number, the more information that is written to the manager log. The default is 1. The following are the possible values: 0 is None, 1 is Minimal, 2 is Basic, 3 is Moderate, 4 is Advanced, 5 is Verbose.

logsize

Set the maximum size of the manager log. When you set a maximum size for the log file, the file will wrap; when the file reaches the specified size, the subsequent entries are written from the top of the file, overwriting the previous log entries. Log size cannot be set smaller than the current size of the log. Log entries are time stamped so you can tell the order in which they were written. The higher you set the log level, the more carefully you should choose the log size, because you can quickly run out of space when logging at the higher levels.

- *bytes*

  The maximum size in bytes for the manager log file. You can specify either a positive number greater than zero, or the word unlimited. The log file may not reach the exact maximum size before overwriting because the log entries themselves vary in size. The default value is 1 MB.

metric set
Sets the **loglevel** and **logsize** for the metric monitor log. The loglevel is the metric monitor logging level (0 - None,1 - Minimal,2 - Basic,3 - Moderate, 4 - Advanced, or 5 - Verbose). The default log level is 1. The log size is the maximum number of bytes to be logged in the metric monitor log file. You can specify either a positive number greater than zero, or unlimited. The default logsize is 1 MB.

quiesce
Specify no more connections to be sent to a server except subsequent new connections from the client to the quiesced server if the connection is designated as sticky and stickytime has not expired. The manager sets the weight for that server to 0 in every port to which it is defined. Use this command if you want to do some quick maintenance on a server and then unquiesce it. If you delete a quiesced server from the configuration and then add it back, it will not retain its status prior to being quiesced. For more information, see "Quiesce servers for server maintenance windows" on page 88.

- *server*

  The IP address of the server as either a symbolic name or in dotted decimal format.

- **daily** *start_hour end_hour*

  Note: This setting specifies to quiesce the server at a time of day, *start_hour*, and unquiesce the server at a later point, *end_hour*. The values for *start_hour* and *end_hour* are can range from 0 to 23. For example, (0 0) indicates to quiesce the server from 12:00 AM to 12:59 AM. (12 13) indicates to quiesce the server from 12:00 PM to 1:59 PM, which is a 2 hour period. Specify (-1 -1) to disable the daily quiesce for a particular server.

reach set
Sets the interval, loglevel, and logsize for the reach advisor.

refresh
Set the number of intervals before querying the executor for a refresh of information about new and active connections.

- *refresh cycle*

  A positive number representing the number of intervals. The default is 2.

report
Display a statistics snapshot report.

- *cluster*

  The address of the cluster you want displayed in the report. The address can be either a symbolic name or in IP address format. The default is a manager report display for all the clusters.

  Note: Additional clusters are separated by a plus sign (+).

restart
Restart all servers (that are not down) to normalized weights (1/2 of maximum weight).

- *message*

  A message that you want written to the manager log file.

sensitivity
Set minimum sensitivity to which weights update. This setting defines when the manager should change its weighting for the server based on external information.

- *weight*

  A number from 1 to 100 to be used as the weight percentage. The default of 5 creates a minimum sensitivity of 5%.

smoothing
Set an index that smooths the variations in weight when load balancing. A higher smoothing index will cause server weights to change less drastically as network conditions change. A lower index will cause server weights to change more drastically.

- *index*

  A positive floating point number. The default is 1.5.

start
Start the manager.

- *log file*

  File name to which the manager data is logged. Each record in the log is time stamped. The default file is installed in the logs directory. See "Examples" on page 161. To change the directory where the log files are kept, see "Logging with Load Balancer" on page 92.

- *metric_port*

  Port that Metric Server will use to report system loads. If you specify a metric port, you must specify a log file name. The default metric port is 10004.

status
Display the current status of all the values in the manager that can be set globally and their defaults.

stop
Stop the manager.

unquiesce
Specify that the manager can begin to give a weight higher than 0 to a server that was previously quiesced, in every port to which it is defined.

- *server*

  The IP address of the server as either a symbolic name or in dotted decimal format.

version
Display the current version of the manager.

Samples

- To set the updating interval for the manager to every 5 seconds:

  ```
  dscontrol manager interval 5
  ```

- To set the level of logging to 0 for better performance:

  ```
  dscontrol manager loglevel 0
  ```

- To set the manager log size to 1,000,000 bytes:

  ```
  dscontrol manager logsize 1000000
  ```

- To specify that no more connections be sent to the server at 130.40.52.153:

  ```
  dscontrol manager quiesce 130.40.52.153
  ```

- To set the number of updating intervals before the weights are refreshed to 3:

```
dscontrol manager refresh 3
```

- To get a statistics snapshot of the manager:

  ```
  dscontrol manager report
  ```

- To restart all the servers to normalized weights and write a message to the manager log file:

  ```
  dscontrol manager restart Restarting the manager to update code
  ```

- To set the sensitivity to weight changes to 10:

  ```
  dscontrol manager sensitivity 10
  ```

- To set the smoothing index to 2.0:

  ```
  dscontrol manager smoothing 2.0
  ```

- To start the manager and specify the log file named ndmgr.log (paths cannot be set):

  ```
  dscontrol manager start ndmgr.log
  ```

- To display the current status of the values associated with the manager:

  ```
  dscontrol manager status
  ```

- To stop the manager:

  ```
  dscontrol manager stop
  ```

- To specify that no more new connections be sent to a server at 130.40.52.153 between 2:00 AM and 4:59 PM:

  ```
  dscontrol manager quiesce 130.40.52.153 daily 2 16
  ```

- To specify that the manager can begin to give a weight higher than 0 to a server at 130.40.52.153 that was previously quiesced:

  ```
  dscontrol manager unquiesce 130.40.52.153
  ```

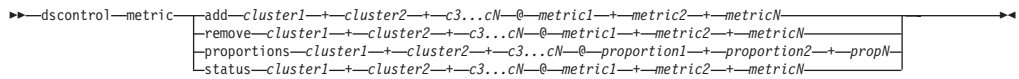
- To display the current version number of the manager:

  ```
  dscontrol manager version
  ```

dscontrol metric

You can configure system metrics with the `dscontrol metric` command.

Syntax



Parameters

add
Add the specified metric.

- *cluster*

  The address to which clients connect. The address can be either the host name of the machine, or the IP address notation format. Additional clusters are separated by a plus sign (+).

- *metric*

  The system metric name. This must be the name of an executable or script file in the metric server's script directory.

remove
Remove the specified metric.

proportions
Set the proportions for all the metrics associated with this object.

status
Display the current values of this metric.

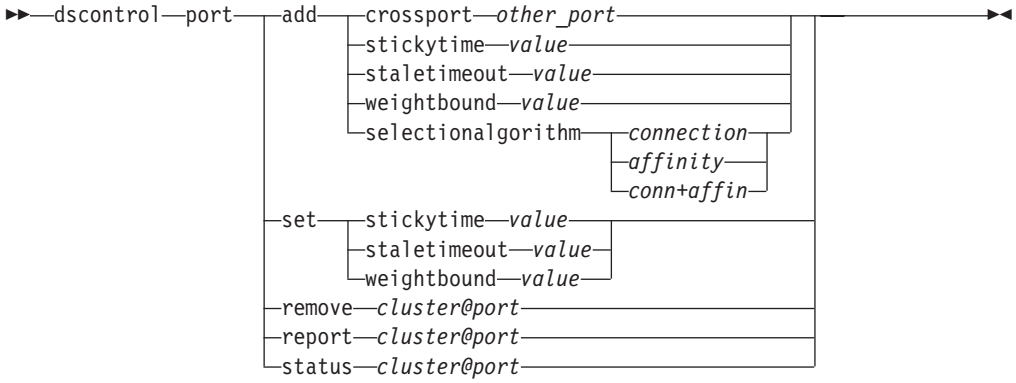
Samples

- To add a system metric:
  ```
  dscontrol metric add site1@metric1
  ```
- To set proportions for a sitename with two system metrics:
  ```
  dscontrol metric proportions site1 0 100
  ```
- To display the current status of values associated with the specified metric:
  ```
  dscontrol metric status site1@metric1
  ```

dscontrol port

Configure ports and port settings with the dscontrol port command.

Syntax



Parameters

add
Add a port to a cluster. You must add a port to a cluster before you can add any servers to that port. If there are no ports for a cluster, all client requests are processed locally. You can add more than one port at one time using this command.

- **crossport**

  Crossport affinity allows you to expand the affinity feature across multiple ports so that client requests received on different ports can still be sent to the same server for subsequent requests. For the crossport value, specify the *other_port* number for which you want to share the cross port affinity feature.

  In order to use this feature, the ports must:

  – Share the same cluster address
  – Share the same servers
  – Use the affinity or conn+aff selection algorithm
  – Have the same stickytime value, which is not zero

  To remove the crossport feature, set the crossport value back to its own port number.

– *other_port*: specifies the value of crossport. The default value is the same as its own port number.

- **stickytime** *value*

  The interval between the closing of one connection and the opening of a new connection during which a client will be sent back to the same server used during the first connection. After the sticky time value has elapsed, the client might be sent to a server different from the first.

  Note: The stickytime value is only valid for the conn+aff selection algorithm. For other selection algorithms, the stickytime value will be set to the same value as the staletimeout value.

  – *value* can be:

    - yes
    - no

- **staletimeout**

  The number of seconds during which there can be no activity on a connection before that connection is removed. For the Dispatcher component, the default value is 900 for port 21 (FTP) and 32,000,000 for port 23 (Telnet). For all other Dispatcher ports, the default is 300. Staletimeout can also be set at the executor or cluster level.

  – *value*

    The value of **staletimeout** in number of seconds.

weightbound
Set the maximum weight for servers on this port. This affects how much difference there can be between the number of requests the executor will give each server. The default value is 20.

- *weight*

  A number from 1–100 representing the maximum weight bound.

selectionalgorithm
Defines the method for selecting the next server.

- **affinity**

  Specifies that the server selection is based on client affinity.

- **connection**

  Specifies that the server selection is based on simple round-robin selection (default).

- **conn+affin**

  Note: Specifies that server selection is based on an existing connection. For new connections, the server selection is based on affinity.

set
Set the fields of a port.

remove
Remove this port.

report
Report on this port.

status
Show status of servers on this port. If you want to see the status on all ports, do not specify a port with this command, but remember to still include the @ symbol.

Sample

- To set the selection algorithm for a port:

  ```
  dscontrol port add cluster@port selectionalgorithm affinity
  ```

- To add port 80 and 23 to a cluster address 130.40.52.153:

  ```
  dscontrol port add 130.40.52.153@80+23
  ```

- To set the maximum weight of 10 to port 80 at a cluster address of 130.40.52.153:

  ```
  dscontrol port set 130.40.52.153@80 weightbound 10
  ```

- To set the stickytime value to 60 seconds for port 80 and port 23 at a cluster address of 130.40.52.153:

  ```
  dscontrol port set 130.40.52.153@80+23 stickytime 60
  ```

- To set the cross port affinity of port 80 to port 23 at a cluster address of 130.40.52.153:

  ```
  dscontrol port add 130.40.52.153@80 crossport 23
  ```

- To remove port 23 from a cluster address of 130.40.52.153:

  ```
  dscontrol port remove 130.40.52.153@23
  ```

- To get the status of port 80 at a cluster address of 9.67.131.153:

  ```
  dscontrol port status 9.67.131.153@80
  ```

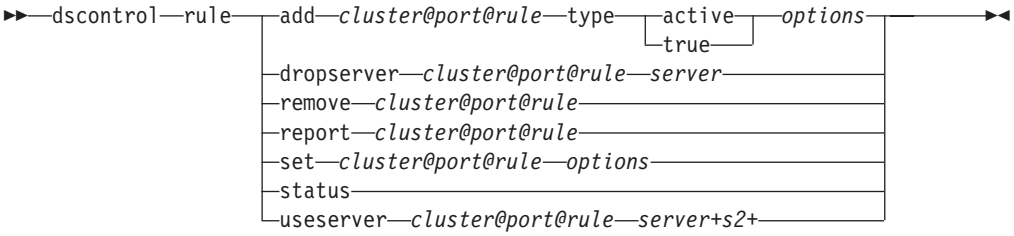
- To get the report of port 80 at a cluster address of 9.62.130.157:

  ```
  dscontrol port report 9.62.130.157@80
  ```

dscontrol rule

Control the executor function with the `dscontrol rule` command.

Syntax



The following options are available for this command:

Parameters

add

Add this rule to a port.

- *cluster*: specifies the address of the cluster as either a symbolic name or in IP address format. You can use a colon (:) to act as a wild card. For instance, the following command will result in adding RuleA to port 80 for all clusters:

  ```
  dscontrol rule add :80:RuleA type type
  ```

Separate additional clusters with a plus sign (+).

- *port*: specifies the number of the port. You can use a colon (:) to act as a wild card. For instance, the following command, dscontrol rule add clusterA::RuleA type type, will result in adding RuleA to all ports for ClusterA. Separate additional ports with a plus sign (+).
- *rule*: specifies the name that you choose for the rule. This name can contain any alphanumeric character, underscore, hyphen, or period. It can be from 1 to 20 characters and cannot contain any blanks. Separate additional rules with a plus sign (+).
- **type** *value*
  - **active**: based on the number of active connections total for the port. This rule will work only if the manager is running.
  - **true**: specifies that this rule will always evaluate as true.
- *beginrange*: specifies the lower value in the range used to determine whether or not the rule is true. This is an integer with a default value of 0.
- *endrange*: specifies the higher value in the range used to determine whether or not the rule is true. This is an integer with a default value of 2 to the 32nd power minus 1.
- **priority** *value*: The order in which the rules are reviewed, where *value* is an integer.

  If you do not specify the priority of the first rule you add, Load Balancer will set it to 1 by default. When a subsequent rule is added, by default its priority is calculated to be 10 + the current lowest priority of any existing rule. For example, assume you have an existing rule whose priority is 30. You add a new rule and set its priority at 25 (which, remember, is a higher priority than 30). Then you add a third rule without setting a priority. The priority of the third rule is calculated to be 40 (30 + 10).
- **evaluate** *value*: specifies whether to evaluate the rule's condition across all servers within the port or across servers within the rule. *Value* can be:
  - **port**: specifies to evaluate rule's condition across all the servers on the port. This is the default value.
  - **rule**: specifies to evaluate the rule's condition across the servers within the rule.

  Evaluate servers within the rule

  The option to measure the rule's condition across the servers within the rule allows you to configure two rules with the following characteristics:

  1. The first rule that gets evaluated contains all the servers maintaining the Web site content, and the evaluate option is set to rule (evaluate the rule's condition across the servers within the rule).
  2. The second rule is an always true rule that contains a single server that responds with a "site busy" type response.

  The result is that when traffic exceeds the threshold of the servers within the first rule, traffic is sent to the "site busy" server within the second rule. When traffic falls below the threshold of the servers within the first rule, new traffic continues once again to the servers in the first rule.

  Evaluate servers on the port

  Using the two rules described above, if you set the evaluate option to port for the first rule (evaluate rule's condition across all the servers on the port), when traffic exceeds the threshold of that rule, traffic is sent to the "site busy" server associated to the second rule. The first rule measures all server traffic (including the "site busy" server) on the port to determine whether

the traffic exceeds the threshold. As congestion decreases for the servers associated to the first rule, an unintentional result may occur where traffic continues to the ″site busy″ server because traffic on the port still exceeds the threshold of the first rule.

dropserver
Remove a server from a rule set.

- *server*: specifies the name of the server to remove. This is the IP address of the TCP server machine as either a symbolic name or in IP address format. Or, if you used server partitioning, use the logical server's unique name. See cprf_serverpart.dita for more information. Separate additional servers with a plus sign (+).

remove
Remove one or more rules, separated from one another by plus signs.

report
Display the internal values of one or more rules.

set
Set values for this rule.

useserver
Insert servers into a rule set.

status
Display the values that are configured of one or more rules.

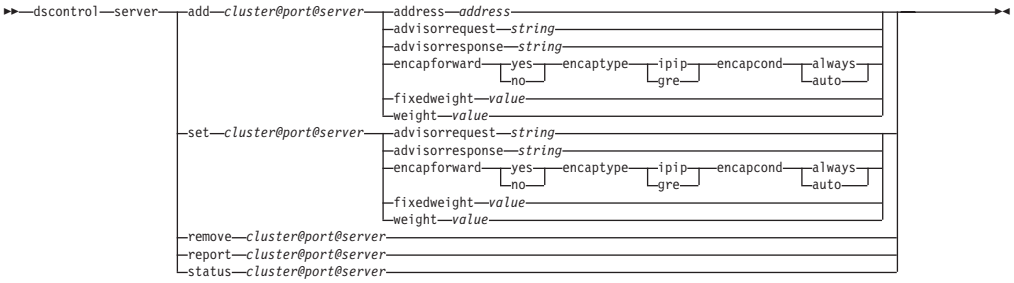
Samples

- For example, to route a range of connections to a certain cluster and port:
  ```
  dscontrol rule add 130.40.52.153@80@pool2 type active beginrange 250 endrange 500
  ```
- Create a rule that always evaluates as true with a priority of 100:
  ```
  dscontrol rule add 130.40.52.153@80@jamais type true priority 100
  ```

dscontrol server

Configure servers and modify existing server configurations with the dscontrol server command.

Syntax



Parameters

add
Add this server.

- *cluster*

The address of the cluster as either a symbolic name or in IP address format. You can use a colon (:) to act as a wild card. For instance, the following command, dscontrol server add :80:ServerA, will result in adding ServerA to port 80 on all clusters.

Note: Additional clusters are separated by a plus sign (+)
.

- *port*

  The number of the port. You can use a colon (:) to act as a wild card. For instance, the following command, dscontrol server add ::ServerA, will result in adding ServerA to all clusters on all ports.

  Note: Additional ports are separated by a plus sign (+).

- *server*

  The server is the unique IP address of the TCP server machine as either a symbolic name or in IP address format. Or, if you use a unique name that does not resolve to an IP address, you must provide the server address parameter on the dscontrol server add command.

  Note: Additional servers are separated by a plus sign (+).

- **address**

  The unique IP address of the TCP server machine as either a host name or in IP address format. If the server is not able to be resolved, you must provide the address of the physical server machine.

  – *address*

    Value of the address of the server.

- **advisorrequest**

  String

- **advisorresponse**

  String

- **encapforward** *value*

  Specifies to enable encapsulation forwarding. *Value* can be yes or no.

  Note: Use encapsulation forwarding when the back-end server is not located on the same network segment or if you are using virtualization technology and need to forward packets that are otherwise unable to be forwarded.

  – **encaptype** *value*

    Specifies the type of encapsulation forwarding. *Value* can be:

    - ipip
    - gre

  – **encapcond** *value*

    Specifies the conditions in which to enable encapsulation forwarding. *Value* can be:

    - always
    - auto

- **fixedweight**

  The fixedweight option allows you to specify whether you want the manager to modify the server weight or not. If you set the fixedweight value

to yes, when the manager runs it will not be allowed to modify the server weight. For more information, see "Managing traffic with server weights" on page 103.

– *value*

Specifies the value of fixedweight. The value can be yes or no. Default is no.

– *portvalue*

Value of the map port number. The default is the client request's destination port number.

- **weight**

A number from 0–100 (but not to exceed the specified port's weightbound value) representing the weight for this server. Setting the weight to zero will prevent any new requests from being sent to the server, but will not end any currently active connections to that server. The default is one-half the specified port's maximum weightbound value. If the manager is running, this setting will be quickly overwritten.

– *value*

Value of the server weight.

remove
Remove this server.

report
Report on this server. The report contains the following information per server: current number of connections per second (CPS), kilobytes transferred in a one second interval (KBPS), total number of connections (Total), number of connections that are in the active state (Active), number of connections that are in the FIN state (FINed), and number of completed connections (Comp).

set
Set values for this server.

status
Show status of the servers.

Samples

- To add the server at 27.65.89.42 to port 80 on a cluster address 130.40.52.153:

```
dscontrol server add 130.40.52.153@80@27.65.89.42
```

- To remove the server at 27.65.89.42 on all ports on all clusters:

```
dscontrol server remove @@27.65.89.42
```

- To set the weight to 10 for server 27.65.89.42 at port 80 on cluster address 130.40.52.153:

```
dscontrol server set 130.40.52.153@80@27.65.89.42 weight 10
```

- To allow the HTTP advisor to query an HTTP URL request HEAD / HTTP/1.0 for server 27.65.89.42 on HTTP port 80:

```
dscontrol server set 130.40.52.153@80@27.65.89.42 advisorrequest "\"HEAD / HTTP/1.0\""
```

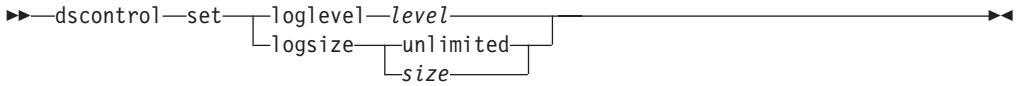
- To show the status for server 9.67.143.154 on port 80:

```
dscontrol server status 9.67.131.167@80@9.67.143.154
```

dscontrol set

Configure the settings for the server log file with the `dscontrol set` command.

Syntax



Parameters

loglevel
The level at which the dsserver logs its activities.

- *level*

  The default value of **loglevel** is 0. The range is 0–5. The following are the possible values: 0 is None, 1 is Minimal, 2 is Basic, 3 is Moderate, 4 is Advanced, 5 is Verbose.

logsize
The maximum number of bytes to be logged in the log file.

- *size*

  The default value of logsize is 1 MB.

dscontrol status

You can display status for the managers or advisors with the `dscontrol status` command.

Syntax

Parameters

There are no parameters for this command.

Sample

To see which managers and advisors are running:

```
dscontrol status
```

Examples

This section provides examples of code snippets, command syntax, and configuration values that are relevant to performing tasks with Load Balancer.

To open the information center table of contents to the location of this reference information, click the **Show in Table of Contents** button ( ) on your information center border.

Example: Sample advisor

This is a sample advisor file called ADV_sample.

```
/ * *
 * ADV_sample: The Load Balancer HTTP advisor
 *
 *
 * This class defines a sample custom advisor for Load Balancer. Like all
```

```
* advisors, this custom advisor extends the function of the advisor base,
* called ADV_Base. It is the advisor base that actually performs most of
* the advisor's functions, such as reporting loads back to the Load Balancer
* for use in the Load Balancer's weight algorithm. The advisor base also
* performs socket connect and close operations and provides send and receive
* methods for use by the advisor. The advisor itself is used only for
* sending and receiving data to and from the port on the server being
* advised. The TCP methods within the advisor base are timed to calculate
* the load. A flag within the constructor in the ADV_base overwrites the
* existing load with the new load returned from the advisor if desired.
*
* Note: Based on a value set in the constructor, the advisor base supplies
* the load to the weight algorithm at specified intervals. If the actual
* advisor has not completed so that it can return a valid load, the advisor
* base uses the previous load.
*
* NAMING
*
* The naming convention is as follows:
*
*  - The file must be located in the following Load Balancer directory:
*
*     ulb/servers/lib/CustomAdvisors/ (ulb\servers\lib\CustomAdvisors on Windows)
*
*  - The Advisor name must be preceded with "ADV_". The advisor can be
*    started with only the name, however; for instance, the "ADV_sample"
*    advisor can be started with "sample".
*
*  - The advisor name must be in lowercase.
*
* With these rules in mind, therefore, this sample is referred to as:
*
*     <base directory="">/lib/CustomAdvisors/ADV_sample.class
*
*
* Advisors, as with the rest of Load Balancer, must be compiled with the
* prerequisite version of Java. To ensure access to Load Balancer classes, make
* sure that the ibmlb.jar file (located in the lib subdirectory of the base
* directory) is included in the system's CLASSPATH.
*
* Methods provided by ADV_Base:
*
* - ADV_Base (Constructor):
*
*   - Parms
*     - String sName = Name of the advisor
*     - String sVersion = Version of the advisor
*     - int iDefaultPort = Default port number to advise on
*     - int iInterval = Interval on which to advise on the servers
*     - String sDefaultName = Unused. Must be passed in as "".
*     - boolean replace = True - replace the load value being calculated
*                                by the advisor base
*                         False - add to the load value being calculated
*                                 by the advisor base
*   - Return
*     - Constructors do not have return values.
*
* Because the advisor base is thread based, it has several other methods
* available for use by an advisor. These methods can be referenced using
* the CALLER parameter passed in getLoad().
*
* These methods are as follows:
*
* - send - Send a packet of information on the established socket connection
*          to the server on the specified port.
* - Parms
*   - String sDataString - The data to be sent in the form of a string
```

```
 * - Return
 *   - int RC - Whether the data was sucessfully sent or not: zero indicates
 *              data was sent; a negative integer indicates an error.
 *
 * - receive - Receive information from the socket connection.
 *   - Parms
 *     - StringBuffer sbDataBuffer - The data received during the receive call
 *   - Return
 *     - int RC - Whether the data was successfully received or not; zero
 *                indicates data was sent; a negative integer indicates
 *                an error.
 *
 * If the function provided by the advisor base is not sufficient,
 * you can create the appropriate function within the advisor and
 * the methods provided by the advisor base will then be ignored.
 *
 * An important question regarding the load returned is whether to apply
 * it to the load being generated within the advisor base,
 * or to replace it; there are valid instances of both situations.
 *
 * This sample is essentially the Load Balancer HTTP advisor. It functions
 * very simply: a send request--an http head request--is issued. Once a
 * response is received, the getLoad method terminates, flagging the advisor
 * base to stop timing the request. The method is then complete. The
 * information returned is not parsed; the load is based on the time
 * required to perform the send and receive operations.
 */

package CustomAdvisors;
import com.ibm.internet.nd.advisors.*;
  public class ADV_sample extends ADV_Base implements ADV_MethodInterface
  {
    String COPYRIGHT =
              "(C) Copyright IBM Corporation 1997, All Rights Reserved.\n";
    static final String ADV_NAME = "Sample";
    static final int ADV_DEF_ADV_ON_PORT = 80;
    static final int ADV_DEF_INTERVAL = 7;

    // Note: Most server protocols require a carriage return ("\r") and line
    // feed ("\n") at the end of messages. If so, include them in
    // your string here.

    static final String ADV_SEND_REQUEST =
       "HEAD / HTTP/1.0\r\nAccept:  */ *\r\nUser-Agent: " +
       "IBM_Load_Balancer_HTTP_Advisor\r\n\r\n";

    /**
     * Constructor.
     *
     * Parms: None; but the constructor for ADV_Base has several parameters
     *        that must be passed to it.
     *
     */
     public ADV_sample()
     {
       super( ADV_NAME,
          "2.0.0.0-03.27.98",
               ADV_DEF_ADV_ON_PORT,
               ADV_DEF_INTERVAL,
               "", // not used false);
       super.setAdvisor( this );
     }


     /**
      * ADV_AdvisorInitialize
      *
```

```
* Any Advisor-specific initialization that must take place after the
* advisor base is started. This method is called only once and is
* typically not used.
*/
public void ADV_AdvisorInitialize()
{
  return;
}

/**
 * getLoad()
 *
 * This method is called by the advisor base to complete the advisor's
 * operation, based on details specific to the protocol. In this sample
 * advisor, only a single send and receive are necessary; if more complex
 * logic is necessary, multiple sends and receives can be issued. For
 * example, a response might be received and parsed. Based on the
 * information learned thereby, another send and receive could be issued.
 *
 * Parameters:
 *
 * - iConnectTime - The current load as it refers to the length of time it
 *                  took to complete the connection to the server through
 *                  the specified port.
 *
 * - caller - A reference to the advisor base class where the Load
 *            Balancer-supplied methods are to perform simple TCP requests,
 *            mainly send and receive.
 *
 * Results:
 *
 * - The load - A value, expressed in milliseconds, that can either be added
 *   to the existing load, or that can replace the existing load, as
 *   determined by the constructor's "replace" flag.
 *
 *   The larger the load, the longer it took the server to respond;
 *   therefore, the lower the weight will become within the Load Balancer.
 *
 *   If the value is negative, an error is assumed. An error from an
 *   advisor indicates that the server the advisor is trying to reach is not
 *   accessible and has been identified as being down. Load Balancer will
 *   not attempt to load balance to a server that is down. Load Balancer will
 *   resume load balancing to the server when a positive value is received.
 *
 */
public int getLoad(int iConnectTime, ADV_Thread caller)
{
  int iRc;
  int iLoad = ADV_HOST_INACCESSIBLE; // -1

  // Send tcp request iRc = caller.send(ADV_SEND_REQUEST);
  if (iRc >= 0)
  {
      // Perform a receive
      StringBuffer sbReceiveData = new StringBuffer("");
      iRc = caller.receive(sbReceiveData);

   /**
    * In the normal advisor mode ("replace" flag is false), the load
    * returned is either 0 or 1 indicating the server is up or down.
    * If the receive is successful, a load of zero is returned
    * indicating that the load built within the base advisor is to be used.
    *
    * Otherwise ("replace" flag is true), return the desired load value.
    */

    if (iRc >= 0)
```

```
          {
             iLoad = 0;
          }
        }
        return iLoad;
      }
  } // End - ADV_sample
```

Example: Implementing custom advisors

The following examples show how custom advisors can be implemented.

Examples for the following types of custom advisors are provided:

- Standard advisor
- Side stream advisor
- Two-port advisor
- WebSphere Application Server (WAS) advisor

Example: Using data returned from advisors

Whether you use a standard call to an existing part of the application server or add a new piece of code to be the server-side counterpart of your custom advisor, you possibly want to examine the load values returned and change server behavior.

The Java StringTokenizer class, and its associated methods, make this investigation easy to do. The content of a typical HTTP command might be

```
GET /index.html HTTP/1.0 90
```

A typical response to this command might be the following:

```
HTTP/1.1 200 OK
Date: Mon, 20 November 2000 14:09:57 GMT
Server: Apache/1.3.12 (Linux and UNIX)
Content-Location: index.html.en
Vary: negotiate
TCN: choice
Last-Modified: Fri, 20 Oct 2000 15:58:35 GMT
ETag: "14f3e5-1a8-39f06bab;39f06a02"
Accept-Ranges: bytes
Content-Length: 424
Connection: close
Content-Type: text/html
Content-Language: en

<!DOCTYPE HTML PUBLIC "-//w3c//DTD HTML 3.2 Final//EN">
<HTML><HEAD><TITLE>Test Page</TITLE></HEAD>
<BODY><H1>Apache server</H1>
<HR>
<P><P>This Web server is running Apache 1.3.12.
</P>
<P><IMG SRC="apache_pb.gif" ALT="">
</P></P>
</HR>
</BODY></HTML>
```

The items of interest are contained in the first line, specifically the HTTP return code. The HTTP specification classifies return codes that can be summarized as follows:

- 2xx return codes are successes
- 3xx return codes are redirections

- 4xx return codes are client errors
- 5xx return codes are server errors

If you know precisely what codes the server can possibly return, your code might not need to be as detailed as this example. However, keep in mind that limiting the return codes you detect might limit the future flexibility of your program.

The following example is a stand-alone Java program that contains a minimal HTTP client. The example invokes a simple, general-purpose parser for examining HTTP responses.

```
import java.io.*;
import java.util.*;
import java.net.*;

public class ParseTest {
   static final int iPort = 80;
   static final String sServer = "www.ibm.com";
   static final String sQuery = "GET /index.html HTTP/1.0\r\n\r\n";
   static final String sHTTP10 = "HTTP/1.0";
   static final String sHTTP11 = "HTTP/1.1";

   public static void main(String[] Arg) {
     String sHTTPVersion = null;
     String sHTTPReturnCode = null;
     String sResponse = null; int iRc = 0;
     BufferedReader brIn = null;
     PrintWriter psOut = null;
     Socket soServer= null;
     StringBuffer sbText = new
     StringBuffer(40);

   try {
     soServer = new Socket(sServer, iPort);
     brIn = new BufferedReader(new InputStreamReader(
                                  soServer.getInputStream()));
     psOut = new PrintWriter(soServer.getOutputStream());
     psOut.println(sQuery);
     psOut.flush();
     sResponse = brIn.readLine();
     try {
       soServer.close();
     } catch (Exception sc) {;}
   } catch (Exception swr) {;}

   StringTokenizer st = new StringTokenizer(sResponse, " ");
   if (true == st.hasMoreTokens()) {
     sHTTPVersion = st.nextToken();
     if (sHTTPVersion.equals(sHTTP110) || sHTTPVersion.equals(sHTTP11)) {
       System.out.println("HTTP Version: " + sHTTPVersion);
     } else {
       System.out.println("Invalid HTTP Version: " + sHTTPVersion);
     }
   } else {
     System.out.println("Nothing was returned");
     return;
   }

   if (true == st.hasMoreTokens()) {
     sHTTPReturnCode = st.nextToken();
     try {
       iRc = Integer.parseInt(sHTTPReturnCode);
     } catch (NumberFormatException ne) {;}

     switch (iRc) {
     case(200):
```

```
      System.out.println("HTTP Response code: OK, " + iRc);
      break;
    case(400): case(401): case(402): case(403): case(404):
      System.out.println("HTTP Response code: Client Error, " + iRc);
      break;
    case(500): case(501): case(502): case(503):
      System.out.println("HTTP Response code: Server Error, " + iRc);
      break;
    default:
      System.out.println("HTTP Response code: Unknown, " + iRc);
      break;
    }
  }

  if (true == st.hasMoreTokens()) {
    while (true == st.hasMoreTokens()) {
      sbText.append(st.nextToken());
      sbText.append(" ");
      }
    System.out.println("HTTP Response phrase: " + sbText.toString());
  }
 }
}
```

Example: Implementing a side stream advisor

The following example demonstrates how a side stream advisor can be implemented. This sample illustrates suppressing the standard socket opened by the advisor base. Instead, this advisor opens a side stream Java socket to query a server. This procedure can be useful for servers that use a different port from normal client traffic to listen for an advisor query.

In this example, a server is listening on port 11999 and when queried returns a load value with a hexadecimal int ″4″. This sample runs in replace mode, that is, the last parameter of the advisor constructor is set to true and the advisor base code uses the returned load value rather than the elapsed time.

Note the call to supressBaseOpeningSocket() in the initialization routine. Suppressing the base socket when no data will be sent is not required. For example, you might want to open the socket to ensure that the advisor can contact the server. Examine the needs of your application carefully before making this choice.

```
package CustomAdvisors;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Date;
import com.ibm.internet.lb.advisors.*;
import com.ibm.internet.lb.common.*;
import com.ibm.internet.lb.server.SRV_ConfigServer;

public class ADV_sidea extends ADV_Base implements ADV_MethodInterface {
  static final String ADV_NAME = "sidea";
  static final int ADV_DEF_ADV_ON_PORT = 12345;
  static final int ADV_DEF_INTERVAL = 7;

  // create an array of bytes with the load request message
  static final byte[] abHealth = {(byte)0x00, (byte)0x00, (byte)0x00,
                                  (byte)0x04};

  public ADV_sidea() {
    super(ADV_NAME, "3.0.0.0-03.31.00", ADV_DEF_ADV_ON_PORT,
          ADV_DEF_INTERVAL, "",
          true);                    // replace mode parameter is true
```

```
    super.setAdvisor( this );
  }

//--------
// ADV_AdvisorInitialize
  public void ADV_AdvisorInitialize()
  {
    suppressBaseOpeningSocket();        // tell base code not to open the
                                        // standard socket
    return;
  }

//--------
// getLoad
  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;    // -1
    int iControlPort = 11999;             // port on which to communicate
                                          // with the server
    String sServer = caller.getCurrentServerId(); // address of server to query
    try {
      socket soServer = new Socket(sServer, iControlPort);   // open socket to
                                                             // server
      DataInputStream disServer = new DataInputStream(
                                                  soServer.getInputStream());
      DataOutputStream dosServer = new DataOutputStream(
                                                  soServer.getOutputStream());
      int iRecvTimeout = 10000;           // set timeout (in milliseconds)
                                          // for receiving data
      soServer.setSoTimeout(iRecvTimeout);
      dosServer.writeInt(4);              // send a message to the server
      dosServer.flush();
      iLoad = disServer.readByte();       // receive the response from the server

    } catch (exception e) {
      system.out.println("Caught exception " + e);
    }
    return iLoad;                         // return the load reported from the server
  }
}
```

Example: Implementing standard advisors

The following example demonstrates how to use a standard custom advisor.

This sample source code is similar to the standard Load Balancer HTTP advisor. It functions as follows:

1. A send request, a ″HEAD/HTTP″ command, is issued.
2. A response is received. The information is not parsed, but the response causes the getLoad method to terminate.
3. The getLoad method returns 0 to indicate success or -1 to indicate a failure.

This advisor operates in normal mode, so the load measurement is based on the elapsed time in milliseconds required to perform the socket open, send, receive, and close operations.

```
package CustomAdvisors;
import com.ibm.internet.lb.advisors.*;
public class ADV_sample extends ADV_Base implements ADV_MethodInterface {
  static final String ADV_NAME ="Sample";
  static final int ADV_DEF_ADV_ON_PORT = 80;
  static final int ADV_DEF_INTERVAL = 7;
  static final String ADV_SEND_REQUEST =
    "HEAD / HTTP/1.0\r\nAccept: */*\r\nUser-Agent: " +
    "IBM_Load_Balancer_HTTP_Advisor\r\n\r\n";
```

```
  //--------
  // Constructor
    public ADV_sample() {
      super(ADV_NAME, "3.0.0.0-03.31.00",
            ADV_DEF_ADV_ON_PORT, ADV_DEF_INTERVAL, "",
            false);
      super.setAdvisor( this );
   }

  //--------
  // ADV_AdvisorInitialize
    public void ADV_AdvisorInitialize() {
      return;                                // usually an empty routine
    }

  //--------
  // getLoad

    public int getLoad(int iConnectTime, ADV_Thread caller) {
      int iRc;
      int iLoad = ADV_HOST_INACCESSIBLE;           // initialize to inaccessible

      iRc = caller.send(ADV_SEND_REQUEST);         // send the HTTP request to
                                                   // the server
      if (0 <= iRc) {                              // if the send is successful
        StringBuffer sbReceiveData = new StringBuffer("");      // allocate a buffer
                                                                // for the response
      iRc = caller.receive(sbReceiveData);         // receive the result

      // parse the result here if you need to

      if (0 <= iRc) {                              // if the receive is successful
        iLoad = 0;                                 // return 0 for success
      }                                            // (advisor's load value is ignored by
    }                                              // base in normal mode)
    return iLoad;
  }
}
```

Example: Implementing the WAS advisor

The following examples show how custom advisors can be implemented.

A sample custom advisor for WebSphere Application Server is included in the *install_root*/servers/samples/CustomAdvisors/ directory. The full code is not duplicated in this document. Ensure that the following will be implemented:

- ADV_was.java is the advisor source code file that is compiled and run on the Load Balancer machine.
- LBAdvisor.java.servlet is the servlet source code that must be renamed to LBAdvisor.java, compiled, and run on the WebSphere Application Server machine.

The complete advisor is only slightly more complex than the sample. It adds a specialized parsing routine that is more compact than the StringTokenizer example shown in the topic “Example: Using data returned from advisors” on page 80.

The more complex part of the sample code is in the Java servlet. Among other methods, the servlet contains two methods required by the servlet specification: init() and service(), and one method, run(), that is required by the Java.lang.thread class.

- init() is called once by the servlet engine at initialization time. This method creates a thread named _checker that runs independently of calls from the advisor and sleeps for a period of time before resuming its processing loop.
- service() is called by the servlet engine each time the servlet is invoked. In this case, the method is called by the advisor. The service() method sends a stream of ASCII characters to an output stream.
- run() contains the core of the code execution. It is called by the start() method that is called from within the init() method.

The relevant fragments of the servlet code appear below:

```
...
  public void init(ServletConfig config) throws ServletException {
    super.init(config);
    ...
    _checker = new Thread(this);
    _checker.start();
  }

  public void run() {
    setStatus(GOOD);

    while (true) {
      if (!getKeepRunning())
        return;
      setStatus(figureLoad());
      setLastUpdate(new java.util.Date());

    try {
      _checker.sleep(_interval * 1000);
    } catch (Exception ignore) { ; }
  }
}

public void service(HttpServletRequest req, HttpServletResponse res)
                    throws ServletException, IOException {
  ServletOutputStream out = null;
  try {
    out = res.getOutputStream();
  } catch (Exception e) { ... }
  ...
  res.setContentType("text/x-application-LBAdvisor");
  out.println(getStatusString());
  out.println(getLastUpdate().toString());
  out.flush(); return;
}
...
```

Example: Implementing a two-port advisor

The following example shows how to implement a two-port advisor. This custom advisor sample demonstrates the capability to detect failure for one port of a server based upon both its own status and on the status of a different server daemon that is running on another port on the same server machine.

For example, if the HTTP daemon on port 80 stops responding, you might also want to stop routing traffic to the SSL daemon on port 443.

This advisor is more aggressive than standard advisors, because it considers any server that does not send a response to have stopped functioning, and marks it as down. Standard advisors consider unresponsive servers to be very slow. This advisor marks a server as down for both the HTTP port and the SSL port based on a lack of response from either port.

To use this custom advisor, the administrator starts two instances of the advisor: one on the HTTP port, and one on the SSL port. The advisor instantiates two static global hash tables, one for HTTP and one for SSL. Each advisor tries to communicate with its server daemon and stores the results of this event in its hash table. The value that each advisor returns to the base advisor class depends on both the ability to communicate with its own server daemon and the ability of the partner advisor to communicate with its daemon.

The following custom methods are used.

- ADV_nte() is a simple container object to hold information about a server. These objects are stored in the hash table as table elements. Each object has a time stamp that is used to determine whether the element is current.
- putNte() and getNte() are synchronized methods that ensure that the two advisor instances access the hash table in a controlled fashion.
- getLoadHTTP is a method that queries the responsiveness of an HTTP server. It is a low-level routine and does not gather or use information about SSL.
- getLoadSSL() is a method that queries the responsiveness of an SSL server. It is a low-level routine and does not gather or use information about HTTP.
- getLoad() is the entry point routine for this custom advisor. It can handle both protocols and can store and fetch information from the hash table. This is the routine that links the two ports.

The following error conditions are detected:

- Unresponsive server machine - The base advisor classes periodically send a ping signal to the server address. If the address is not reachable, the base advisor classes marks the server down. Neither of the two instances of the custom advisor is called, and both servers on that machine are marked down.
- One daemon on a server machine becomes unresponsive, but the other is working - When the base code attempts to open a socket with the server, the connection is refused, and the base advisor for this protocol marks the server as down. The custom advisor code for that protocol is not called. Although the custom advisor for the other protocol continues communicating with its server, it learns from the hash table that the other custom advisor cannot communicate with its server daemon. Therefore, the second protocol's advisor also marks its server as down.
- One daemon does not send a response, but the other daemon does - The custom advisor for the unresponsive protocol detects the failure to communicate, marks the server as down, and stores the data in the hash table. The custom advisor for the other port learns that information from the hash table and marks its server as down.

This sample is written to link ports 80 for HTTP and 443 for SSL, but it can be tailored to any combination of ports:

```
package CustomAdvisors;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Date;
import com.ibm.internet.lb.advisors.*;
import com.ibm.internet.lb.common.*;
import com.ibm.internet.lb.manager.*;
import com.ibm.internet.lb.server.SRV_ConfigServer;

//--------
// Define the table element for the hash tables used in this custom advisor
```

```
class ADV_nte implements Cloneable {
  private String sCluster;
  private int iPort;
  private String sServer;
  private int iLoad;
  private Date dTimestamp;

//--------
// constructor

  public ADV_nte(String sClusterIn, int iPortIn, String sServerIn,
                 int iLoadIn) {
    sCluster = sClusterIn;
    iPort = iPortIn;
    sServer = sServerIn;
    iLoad = iLoadIn;
    dTimestamp = new Date();
  }

//--------
// check whether this element is current or expired
  public boolean isCurrent(ADV_twop oThis) {
    boolean bCurrent;
    int iLifetimeMs = 3 * 1000 * oThis.getInterval();     // set lifetime as
                                                          // 3 advisor cycles
    Date dNow = new Date();
    Date dExpires = new Date(dTimestamp.getTime() + iLifetimeMs);

    if (dNow.after(dExpires)) {
      bCurrent = false;
    } else {
      bCurrent = true;
    } return bCurrent;
  }

//--------
// value accessor(s)

 public int getLoadValue() { return iLoad; }

//--------
// clone (avoids corruption between threads)

 public synchronized Object Clone() {
   try {
     return super.clone();
   } catch (cloneNotSupportedException e) {
     return null;
    }
  }

}

//--------
// define the custom advisor

public class ADV_twop extends ADV_Base
   implements ADV_MethodInterface, ADV_AdvisorVersionInterface {
   static final int ADV_TWOP_PORT_HTTP = 80;
   static final int ADV_TWOP_PORT_SSL = 443;

   //--------
   // define tables to hold port-specific history information

   static HashTable htTwopHTTP = new Hashtable();
   static HashTable htTwopSSL = new Hashtable();
   static final String ADV_TWOP_NAME = "twop";
```

```
  static final int ADV_TWOP_DEF_ADV_ON_PORT = 80;
  static final int ADV_TWOP_DEF_INTERVAL = 7;
  static final String ADV_HTTP_REQUEST_STRING =
     "HEAD / HTTP/1.0\r\nAccept: */*\r\nUser-Agent: " +
     "IBM_LB_Custom_Advisor\r\n\r\n";

  //--------
  // create byte array with SSL client hello message

  public static final byte[] abClientHello = {
    (byte)0x80, (byte)0x1c,
    (byte)0x01,                                 // client hello
    (byte)0x03, (byte)0x00,                     // SSL version
    (byte)0x00, (byte)0x03,                     // cipher spec len (bytes)
    (byte)0x00, (byte)0x00,                     // session ID len (bytes)
    (byte)0x00, (byte)0x10,                     // challenge data len (bytes)
    (byte)0x00, (byte)0x00, (byte)0x03,         // cipher spec
    (byte)0x1A, (byte)0xFC, (byte)0xE5, (byte)0x20,    // challenge data
    (byte)0xFD, (byte)0x3A, (byte)0x3C, (byte)0x18,
    (byte)0xAB, (byte)0x67, (byte)0xB0, (byte)0x52,
    (byte)0xB1, (byte)0x1D, (byte)0x55, (byte)0x44, (byte)0x0D, (byte)0x0A };

 //--------
 // constructor

 public ADV_twop() {
   super(ADV_TWOP_NAME, VERSION, ADV_TWOP_DEF_ADV_ON_PORT,
         ADV_TWOP_DEF_INTERVAL, "",
         false);                          // false = load balancer times the response
   setAdvisor ( this );
 }

 //--------
 // ADV_AdvisorInitialize

 public void ADV_AdvisorInitialize() {
   return; }

 //--------
 // synchronized PUT and GET access routines for the hash tables

 synchronized ADV_nte getNte(Hashtable ht, String sName, String sHashKey) {
   ADV_nte nte = (ADV_nte)(ht.get(sHashKey));
   if (null != nte) {
     nte = (ADV_nte)nte.clone();
   }
   return nte;
 }
synchronized void putNte(Hashtable ht, String sName, String sHashKey,
                         ADV_nte nte) { ht.put(sHashKey,nte); return;
}


 //--------
 // getLoadHTTP - determine HTTP load based on server response

 int getLoadHTTP(int iConnectTime, ADV_Thread caller) {
   int iLoad = ADV_HOST_INACCESSIBLE;
   int iRc = caller.send(ADV_HTTP_REQUEST_STRING);   // send request message
                                                     // to server
   if (0 <= iRc) {                                   // did the request return a failure?
     StringBuffer sbReceiveData = new StringBuffer("") // allocate a buffer
                                                       // for the response
     iRc = caller.receive(sbReceiveData);             // get response from server

     if (0 <= iRc) {                               // did the receive return a failure?
       if (0 < sbReceiveData.length()) {           // is data there?
```

```
          iLoad = SUCCESS;                          // ignore retrieved data and
                                                    // return success code
        }
      }
    }
    return iLoad;
}


//--------
// getLoadSSL() - determine SSL load based on server response

int getLoadSSL(int iConnectTime, ASV_Thread caller) {
  int iLoad = ADV_HOST_INACCESSIBLE;
  int iRc;

  CMNByteArrayWrapper cbawClientHello = new CMNByteArrayWrapper(
                                                  abClientHello);
  Socket socket = caller.getSocket();

  try {
      socket.getOutputStream().write(abClientHello); // Perform a receive.
      socket.getInputStream().read();                // If receive is successful,
                                                     // return load of 0. We are not
                                                     // concerned with data's contents,
                                                     // and the load is calculated by
                                                     // the ADV_Thread thread.
      iLoad = 0;
  } catch (IOException e) {           // Upon error, iLoad will default to it.
  }
  return iLoad;
}


//--------
// getLoad - merge results from the HTTP and SSL methods

public int getLoad(int iConnectTime, ADV_Thread caller) {
  int iLoadHTTP;
  int iLoadSSL;
  int iLoad;
  int iRc;

  String sCluster = caller.getCurrentClusterId();  // current cluster address
  int iPort = getAdviseOnPort();
  String sServer = caller.getCurrentServerId();
  String sHashKey = sCluster = ":" + sServer;      // hash table key

  if (ADV_TWOP_PORT_HTTP == iPort) {                // handle an HTTP server
    iLoadHTTP = getLoadHTTP(iConnectTime, caller);  // get the load for HTTP

    ADV_nte nteHTTP = newADV_nte(sCluster, iPort, sServer, iLoadHTTP);
    putNte(htTwopHTTP, "HTTP", sHashKey, nteHTTP);     // save HTTP load
                                                       // information
    ADV_nte nteSSL = getNte(htTwopSSL, "SSL", sHashKey);  // get SSL
                                                          // information
      if (null != nteSSL) {
        if (true == nteSSL.isCurrent(this)) {              // check the time stamp
          if (ADV_HOST_INACCESSIBLE != nteSSL.getLoadValue()) {    // is SSL
                                                                   // working?
            iLoad = iLoadHTTP;
          } else {                      // SSL is not working, so mark the HTTP server down
            iLoad= ADV_HOST_INACCESSIBLE;
          }
        } else {                // SSL information is expired, so mark the
                                // HTTP server down
          iLoad = ADV_HOST_INACCESSIBLE;
```

```
      }
    } else {                  // no load information about SSL, report
                              // getLoadHTTP() results
      iLoad = iLoadHTTP;
    }
  }
  else if (ADV_TWOP_PORT_SSL == iPort) {            // handle an SSL server
    iLoadSSL = getLoadSSL(iConnectTime, caller);    // get load for SSL

    ADV_nte nteSSL = new ADV_nte(sCluster, iPort, sServer, iLoadSSL);
    putNte(htTwopSSL, "SSL", sHashKey, nteSSL);     // save SSL load info.

    ADV_nte nteHTTP = getNte(htTwopHTTP, "SSL", sHashKey);   // get HTTP
                                                             // information
    if (null != nteHTTP) {
      if (true == nteHTTP.isCurrent(this)) {                 // check the timestamp
        if (ADV_HOST_INACCESSIBLE != nteHTTP.getLoadValue()) {   // is HTTP
                                                                 // working?
          iLoad = iLoadSSL;
        } else {            // HTTP server is not working, so mark SSL down
          iLoad = ADV_HOST_INACCESSIBLE;
        }
      } else {     // expired information from HTTP, so mark SSL down
        iLoad = ADV_HOST_INACCESSIBLE;
      }
    } else {                // no load information about HTTP, report
                            // getLoadSSL() results
      iLoad = iLoadSSL;
    }
  }

 //--------
 // error handler

  else {
    iLoad = ADV_HOST_INACCESSIBLE;
  }
  return iLoad;
 }
}
```

Glossary

ACK A control bit (acknowledge) occupying no sequence space, which indicates that the acknowledgment field of this segment specifies the next sequence number the sender of this segment is expecting to receive, hence acknowledging receipt of all previous sequence numbers.

address
The unique code assigned to each device or workstation connected to a network. A standard IPv4 address is a 32-bit address field containing two parts. The first part is the network address, and the second part is the host number. An IPv6 address is a 128-bit address field that supports a much higher number of addresses than IPv4, and IPv6 addresses also support additional features like multicast and anycast addressing.

advisor
Advisors collect and analyze feedback from individual servers, and inform the manager function.

agent In systems management, a user that, for a particular interaction, has assumed an agent role.

An entity that represents one or more managed objects by (a) emitting notifications regarding the objects and (b) handling requests from managers for management operations to modify or query the objects.

alias An additional name assigned to a server. The alias makes the server independent of the name of its host machine. The alias must be defined in the domain name server.

API Application programming interface. The interface (calling conventions) by which an application program accesses operating system and other services. An API is defined at source code level and provides a level of abstraction between the application and the kernel (or other privileged utilities) to ensure the portability of the code.

backup
In a high availability configuration, this is the partner of the primary machine. It monitors the status of the primary machine and takes over if necessary. See also high availability, primary.

bandwidth
The difference between the highest and lowest frequencies of a transmission channel. This is the amount of data that can be sent through a given communication circuit per second.

begin range
In rules-based load balancing, a lower value specified on a rule. The default for this value depends on the type of rule.

binary logging
Allows server information to be stored in binary files, and then be processed to analyze the server information that is gathered over time.

Caching proxy
A caching proxy server that can help speed up end-user response time through highly-efficient caching schemes. Flexible PICS filtering helps network administrators control access to Web-based information at one central location.

CGI Common Gateway Interface. A standard for the exchange of information between a Web server and an external program. The external program can be written in any language supported by the operating system, and performs tasks not usually done by the server, such as forms processing.

CGI script
A CGI program written in a scripting language such as Perl or REXX that uses the Common Gateway Interface to perform tasks not usually done by the server, such as forms processing.

client A computer system or process that requests a service of another computer system or process. For example, a workstation or personal computer requesting HTML documents from a Lotus® Domino® Go Webserver is a client of that server.

cluster
A group of TCP or UDP servers that are used for the same purpose and are identified by a single hostname. See also cell.

cluster address
The address to which clients connect.

clustered server
A server that Load Balancer groups with other servers into a single, virtual server. Load Balancer balances TCP or UDP traffic among these clustered servers.

consultant
Collects server metrics from the servers that are being load balanced and sends server weight information to the switch that performs the load balancing.

controller
A collection of one or more consultants.

cross port affinity
Cross port affinity is the affinity (sticky) feature expanded to cover across multiple ports. See also sticky time.

daemon
Disk And Execution Monitor. A program that is not involved explicitly, but lies dormant waiting for some condition(s) to occur. The idea is that the perpetrator of the condition need not be aware that a daemon is lurking (though often a program will commit an action only because it knows that it will implicitly invoke a daemon).

default
A value, attribute, or option that is assumed when none is explicitly specified.

destination address
The address of the high availability partner machine to which heartbeats and responses are sent.

Dispatcher
A component of Load Balancer that efficiently balances TCP or UDP traffic among groups of individual linked servers. The Dispatcher machine is the server running the Dispatcher code.

domain name server
DNS. A general-purpose distributed, replicated, data query service chiefly used on Internet for translating hostnames into Internet addresses. Also, the style of hostname used on the Internet, though such a name is properly called a fully qualified domain name. DNS can be configured to use a sequence of name servers, based on the domains in the name being looked for, until a match is found.

dotted-decimal notation
The syntactical representation for a 32-bit integer that consists of four 8-bit numbers, written in base 10 and separated by periods (dots). It is used to represent IPv4 addresses.

dscontrol
Provides the interface to the Dispatcher component of Load Balancer.

dsserver
Handles the requests from the command line to the executor, manager, and advisors.

end range
In rules-based load balancing, a higher value specified on a rule. The default for this value depends on the type of rule.

Ethernet
A standard type of local area network (LAN). It allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and transmission. Software protocols used by Ethernet systems vary, but include TCP/IP.

executor
One of several functions. The executor routes requests to the TCP or UDP servers, and also monitors the number of new, active, and finished connections and does garbage collection of completed or reset connections. The executor supplies the new and active connections to the manager function.

FIN A control bit (finis) occupying one sequence number, which indicates that the sender will send no more data or control occupying sequence space.

FIN state
The status of a transaction that has finished. When a transaction is in FIN state, the garbage collector can clear the memory reserved for the connection.

Firewall
A computer that connects a private network, such as a business, to a public network, such as the Internet. It contains programs that limit the access between two networks. See also proxy gateway.

FQDN
Fully Qualified Domain Name. The full name of a system, consisting of its local hostname and its domain name, including a top-level domain (tld). For example, ″venera″ is a hostname and ″venera.isi.edu″ is an FQDN. An FQDN should be sufficient to determine a unique Internet address for any host on the Internet. This process, called ″name resolution″, uses the Domain Name System (DNS).

FTP (File Transfer Protocol)
An application protocol used for transferring files to and from network computers. FTP requires a user ID and sometimes a password to allow access to files on a remote host system.

gateway
A functional unit that interconnects two computer networks with different architectures.

GRE Generic Routing Encapsulation. A protocol which allows an arbitrary network protocol A to be transmitted over any other arbitrary protocol B, by encapsulating the packets of A within GRE packets, which in turn are contained within packets of B.

heartbeat
A simple packet sent between two machines in high availability mode used by the standby machine to monitor the health of the active machine.

high availability
A feature in which one Load Balancer machine can take over the function of another, if the primary machine is no longer available.

host A computer, connected to a network, that provides an access point to that network. A host can be a client, a server, or both simultaneously.

host name
The symbolic name assigned to a host. Host names are resolved to IP addresses through a domain name server.

HTML (Hypertext Markup Language)
The language that is used to create hypertext documents. Hypertext documents include links to other documents that contain additional information about the highlighted term or subject. HTML controls the format of text and position of form input areas, for example, as well as the navigable links.

HTTP (Hypertext Transfer Protocol)
The protocol used to transfer and display hypertext documents.

HTTPS (Hypertext Transfer Protocol, Secure)
The protocol used to transfer and display hypertext documents using SSL.

ICMP Internet Control Message Protocol. A message control and error-reporting protocol between a host server and a gateway to the Internet.

IMAP Internet Message Access Protocol. A protocol allowing a client to access and manipulate electronic mail messages on a server. It permits manipulation of remote message folders (mailboxes), in a way that is functionally equivalent to local mailboxes.

Internet
The worldwide collection of interconnected networks that use the Internet suite of protocols and permit public access.

intranet
A secure, private network that integrates Internet standards and applications (such as Web browsers) with an organization's existing computer networking infrastructure.

IP Internet Protocol. A connectionless protocol that routes data through a network or interconnected networks. IP acts as an intermediary between the higher protocol layers and the physical layer.

IP address
Internet Protocol address. The unique address that specifies the actual location of each device or workstation in a network. It is also known as an Internet address.

IPSEC Internet Protocol Security. A developing standard for security at the network or packet processing layer of network communication.

LAN Local Area Network. A computer network of devices connected within a limited geographical area for communication and which can be connected to a larger network.

loopback alias
An alternative IP address associated with the loopback interface. The alternative address has the useful side affect of not advertising on a real interface.

loopback interface
An interface that bypasses unnecessary communications functions when the information is addressed to an entity within the same system.

MAC address
Media Access Control address. The hardware address of a device connected to a shared network medium.

managed node
In Internet communications, a workstation, server, or router that contains a network management agent. In the Internet Protocol (IP), the managed node usually contains a Simple Network Management Protocol (SNMP) agent.

manager
Sets weights of servers based on internal counters in the executor and feedback that is provided by the advisors. The executor then uses the weights to perform load balancing.

mark down
To break all active connections to a server and stop any new connections or packets from being sent to that server.

mark up
To allow a server to receive new connections.

metric A process or command that returns a numeric value that can be used in load balancing on the network, for example, the number of users currently logged on.

metric address
The address where the metric server connects.

metric collector
Resides in the consultant and is responsible for collecting a metric or metrics.

Metric Server
Formerly known as Server Monitor Agent (SMA). Metric server provides system specific metrics to the manager.

MIB Management Information Base. A collection of objects that can be accessed by means of a network management protocol.

A definition for management information that specifies the information available from a host or gateway and the operations allowed.

netmask
For IPv4, a 32–bit mask used to identify the subnetwork address bits in the host portion of an IP address.

network
Hardware and software data communication system. Networks are often

classified according to their geographical extent, local area network (LAN), metropolitan area network (MAN), wide area network (WAN) and also according to the protocols used.

Network Address Translation
NAT, or Network Address Translator, Virtual LAN. A hardware device currently being developed and used to extend the Internet addresses already in use. It allows duplicate IP addresses to be used within a corporation and unique addresses outside.

Network Address Port Translation
NAPT, also known as port mapping. This allows you to configure multiple server daemons within one physical server to listen on different port numbers.

network management station
In the Simple Network Management Protocol (SNMP), a station that runs management application programs that monitor and control network elements.

network proximity
The proximity of two networked entities, such as a client and server, which determines by measuring round-trip time.

NFA (nonforwarding address)
The primary IP address of the machine, used for administration and configuration.

NIC Network Interface Card. An adapter circuit board installed in a computer to provide a physical connection to a network.

NNTP Network News Transfer Protocol. A TCP/IP protocol for transferring news items.

packet The unit of data that is routed between an origin and a destination on the Internet or any other packet-switched network.

PICS Platform for Internet Content Selection. PICS-enabled clients allow the users to determine which rating services they want to use and, for each rating service, which ratings are acceptable and which are unacceptable.

ping A command that sends Internet Control Message Protocol (ICMP) echo-request packets to a host, gateway, or router with the expectation of receiving a reply.

POP3 Post Office Protocol 3. A protocol used for exchanging network mail and accessing mailboxes.

port A number that identifies an abstracted communication device. Web servers use port 80 by default.

primary
In a high availability configuration, the machine that starts out as the machine actively routing packets. Its partner, the backup machine, monitors the status of the primary machine and takes over if necessary. See also backup, high availability.

priority
In rules-based load balancing, the level of importance placed upon any given rule. The evaluates rules from the first priority level to the last priority level.

private network
A separate network on which Load Balancer communicates with clustered servers for performance reasons.

protocol
The set of rules governing the operation of functional units of a communication system if communication is to take place. Protocols can determine low-level details of machine-to-machine interfaces, such as the order in which bits from a byte are sent; they can also determine high-level exchanges between application programs, such as file transfer.

Quality of Service (QoS)
The performance properties of a network service, including throughput, transit delay and priority. Some protocols allow packets or streams to include QoS requirements.

quiesce
To end a process by allowing operations to complete normally.

reach An advisor that issues pings to a given target and reports whether that target is responding.

reach address
In a high availability configuration, the address of the target to which the advisor should issue pings to see if the target is responding.

return address
A unique IP address or hostname. It is configured on the Load Balancer machine and used as its source address when load balancing the client's request to the server.

RMI Remote Method Invocation. Part of the Java programming language library which enables a Java program running on one computer to access the objects and methods of another Java program running on a different computer.

root user
The unrestricted authority to access and modify any part of the AIX, Red Hat Linux, or Solaris operating system, usually associated with the user who manages the system.

route The path of network traffic from origin to destination.

router A device which forwards packets between networks. The forwarding decision is based on network layer information and routing tables, often constructed by routing products.

RPM Red Hat Package Manager.

rule In rules-based load balancing, a mechanism for grouping servers such that a server can be chosen based on information other than the destination address and port.

rule type
In rules-based load balancing, an indicator of the information that should be evaluated to determine whether a rule is true.

scalable
Pertaining to the capability of a system to adapt readily to a greater or lesser intensity of use, volume, or demand. For example, a scalable system can efficiently adapt to work with larger or smaller networks performing tasks of varying complexity.

server A computer that provides shared services to other computers over a network; for example, a file server, a print server, or a mail server.

server address
The unique code assigned to each computer that provides shared services to other computers over a network; for example, a file server, a print server, or a mail server. The server address can be either the IP address or the host name.

server machine
A server that Load Balancer groups with other servers into a single, virtual server. Load Balancer balances traffic among the server machines. Synonymous with clustered server.

service
A function provided by one or more nodes; for example, HTTP, FTP, Telnet.

shell The software that accepts and processes command lines from a user's workstation. The bash shell is one of several UNIX® shells available.

site name
A site name is an unresolvable host name that the client will request. For example, a web site has 3 servers (1.2.3.4, 1.2.3.5, and 1.2.3.6) configured for site name *www.dnsload.com*. When a client requests this site name, one of the three server IP addresses will be returned as the resolution. The site name must be a fully qualified domain name, for example: *dnsload.com*. An unqualified name, for example, *dnsload* is invalid for a site name.

Site Selector
A DNS-based load balancing component of . Site Selector balances the load on servers within a wide area network (WAN) using measurements and weights that are gathered from the Metric Server component running on those servers.

SMTP Simple Mail Transfer Protocol. In the Internet suite of protocols, an application protocol for transferring mail among users in the Internet environment. SMTP specifies the mail exchange sequences and message format. It assumes that the Transmission Control Protocol (TCP) is the underlying protocol.

SNMP
Simple Network Management Protocol. The Internet standard protocol, defined in STD 15, RFC 1157, developed to manage nodes on an IP network. SNMP is not limited to TCP/IP. It can be used to manage and monitor all sorts of equipment including computers, routers, wiring hubs, toasters and jukeboxes.

source address
In a high availability configuration, the address of the high availability partner machine that sends heartbeats.

SPARC
Scalable processor architecture.

sscontrol
Provides the interface to the Site Selector component of .

SSL Secure Sockets Layer. A popular security scheme developed by Netscape Communications Corp. along with RSA Data Security Inc. SSL allows the client to authenticate the server and all data and requests to be encrypted. The URL of a secure server protected by SSL begins with https (rather than HTTP).

sticky time
The interval between the closing of one connection and the opening of a new connection during which a client will be sent back to the same server used during the first connection. After the sticky time, the client may be sent to a server different from the first.

strategy
In a high availability configuration, a keyword for specifying how recovery takes place following the failure of the active machine.

subnet mask
For IPv4, a 32–bit mask used to identify the subnetwork address bits in the host portion of an IP address.

SYN A control bit in the incoming segment, occupying one sequence number, used at the initiation of a connection, to indicate where the sequence numbering will start.

TCP Transmission Control Protocol. A communications protocol used on the Internet. TCP provides reliable host-to-host exchange of information. It uses IP as the underlying protocol.

TCP/IP
Transmission Control Protocol/Internet Protocol. A suite of protocols designed to allow communication between networks regardless of the communication technologies used in each network.

TCP server machine
A server that links with other servers into a single, virtual server. balances TCP traffic among the TCP server machines. Synonymous with clustered server.

Telnet Terminal emulation protocol, a TCP/IP application protocol for remote connection service. Telnet allows a user at one site to gain access to a remote host as if the user's workstation were connected directly to that remote host.

timeout
The time interval allotted for an operation to occur.

TOS Type of service. A one byte field in the IP header of the SYN packet.

TTL A DNS TTL (time to live) is the number of seconds a client can cache the name resolution response.

UDP User Datagram Protocol. In the Internet suite of protocols, a protocol that provides unreliable, connectionless datagram service. It enables an application program on one machine or process to send a datagram to an application program on another machine or process. UDP uses the Internet Protocol (IP) to deliver datagrams.

URI Universal Resource Identifier. The encoded address for any resource on the Web, such as HTML document, image, video clip, program, and so forth.

URL Uniform Resource Locator. A standard way of specifying the location of an object, typically a web page, on the Internet. URLs are the form of address used on the World-Wide Web. They are used in HTML documents to specify the target of a hyperlink which is often another HTML document (possibly stored on another computer).

VPN Virtual Private Network (VPN). A network comprised of one or more secure IP tunnels connecting two or more networks.

WAN Wide Area Network. A network that provides communication services to a geographic area larger than that served by a local area network or a metropolitan area network, and that may use or provide public communication facilities.

WAP Wireless Application Protocol. This is an open international standard for applications that use wireless communication. For example, this standard includes internet access from a mobile phone.

WAS WebSphere® Application Server.

Web The network of HTTP servers that contain programs and files, many of them hypertext documents that contain links to other documents on HTTP servers. Also World Wide Web.

wizard
A dialog within an application that uses step-by-step instructions to guide a user through a specific task.

WLM Workload Manager. An advisor that is provided with Load Balancer. It is designed to work only in conjunction with servers on OS/390® mainframes running the MVS™ Workload Manager (WLM) component.