JBoss Communications SIP
Load Balancer User Guide

The Guide to the JBoss Communications SIP Load Balancer

by Douglas Silas, Jean Deruelle, Vladimir Ralev, Ivelin Ivanov, and Jared Morgan

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts* [https://fedorahosted.org/liberation-fonts/] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight key caps and key-combinations. For example:

> To see the contents of the file `my_next_bestselling_novel` in your current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key cap, all presented in Mono-spaced Bold and all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the hyphen connecting each part of a key-combination. For example:

> Press **Enter** to execute the command.
>
> Press **Ctrl**+**Alt**+**F1** to switch to the first virtual terminal. Press **Ctrl**+**Alt**+**F7** to return to your X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in `Mono-spaced Bold`. For example:

> File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialogue box text; labelled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System > Preferences > Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications > Accessories > Character Map** from the main menu bar. Next, choose **Search > Find** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit > Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in Proportional Bold and all distinguishable by context.

Note the **>** shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar' approach.

`Mono-spaced Bold Italic` or ***Proportional Bold Italic***

Whether Mono-spaced Bold or Proportional Bold, the addition of Italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type `ssh` *`username@domain.name`* at a shell prompt. If the remote machine is `example.com` and your username on that machine is john, type `ssh john@example.com`.
>
> The `mount -o remount` *`file-system`* command remounts the named file system. For example, to remount the `/home` file system, the command is `mount -o remount /home`.
>
> To see the version of a currently installed package, use the `rpm -q` *`package`* command. It will return a result as follows: *`package-version-release`*.

Note the words in bold italics above username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules* (*MPMs*). Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

1.2. Pull-quote Conventions

Two, commonly multi-line, data types are set off visually from the surrounding text.

Output sent to a terminal is set in `Mono-spaced Roman` and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in `Mono-spaced Roman` but are presented and highlighted as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }

}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

A note is a tip or shortcut or alternative approach to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring Important boxes won't cause data loss but may cause irritation and frustration.

Warning

A Warning should not be ignored. Ignoring warnings will most likely cause data loss.

Load Balancer

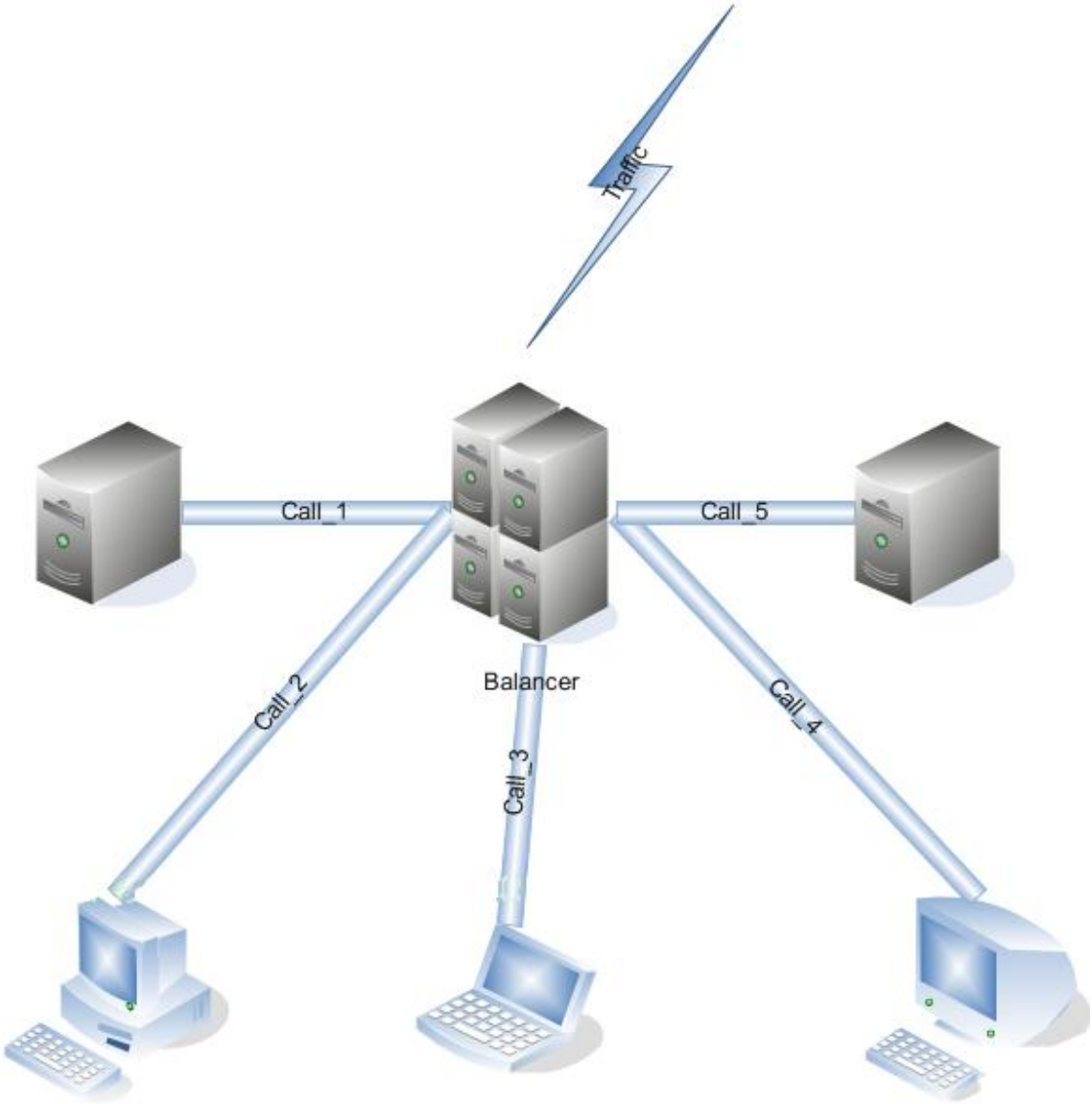

Figure 1.1. Star Cluster Topology.

The JBoss Communications SIP Load Balancer is used to balance the load of SIP service requests and responses between nodes in a JBoss Communications SIP Server cluster, such as JBoss Communications JAIN SLEE or SIP Servlets. Both JBCP servers can be used in conjunction with the SIP load balancer to increase the performance and availability of SIP services and applications.

In terms of functionality, the JBoss Communications SIP load balancer is a simple stateless proxy server that intelligently forwards SIP session requests and responses between User Agents (UAs) on a Wide Area Network (WAN), and SIP Server nodes, which are almost always located on a Local Area Network (LAN). All SIP requests and responses pass through the SIP load balancer.

1.1. SIP Load Balancing Basics

All User Agents send SIP messages, such as `INVITE` and `MESSAGE`, to the same SIP URI (the IP address and port number of the SIP load balancer on the WAN). The load balancer then parses, alters, and forwards those messages to an available node in the cluster. If the message was sent as a part of an existing SIP session, it will be forwarded to the cluster node which processed that User Agent's original transaction request.

The SIP Server that receives the message acts upon it and sends a response back to the SIP load balancer. The SIP load balancer reparses, alters and forwards the message back to the original User Agent. This entire proxying and provisioning process is carried out independent of the User Agent, which is only concerned with the SIP service or application it is using.

By using the load balancer, SIP traffic is balanced across a pool of available SIP Servers, increasing the overall throughput of the SIP service or application running on either individual nodes of the cluster. In the case of a JBCP server with `</distributed>` capabilities, load balancing advantages are applied across the entire cluster.

The SIP load balancer is also able to fail over requests mid-call from unavailable nodes to available ones, thus increasing the reliability of the SIP service or application. The load balancer increases throughput and reliability by dynamically provisioning SIP service requests and responses across responsive nodes in a cluster. This enables SIP applications to meet the real-time demand for SIP services.

1.2. HTTP Load Balancing Basics

In addition to the SIP load balancing, there are several options for coordinated or cooperative load balancing with other protocols such as HTTP. Typically, a JBoss Application Server will use apache HTTP server with mod_jk, mod_proxy, mod_cluster or similar extension installed as an HTTP load balancer. This apache-based load balancer will parse incoming HTTP requests and it will look for the session ID of those requests in order to ensure all requests from the same session arrive at the same application server. By default, this is done by examining the `jsessionid` HTTP cookie or GET parameter and looking for the `jvmRoute` assigned to the session. The typical `jsessionid` value is of the form `<sessionId>.<jvmRoute>` (e.g. `mysessionid323424.node1` where `node1` is the `jvmRoute` component). The very first request for each new HTTP session do not have any session ID assigned, thus apache routes the request to a random application server node. When the node responds it assigns a session ID and `jvmRoute` to the response of the request in a HTTP cookie and this response goes back to the client through apache, which keeps track of which node owns which `jvmRoute`. Once, the very first request is served this way, the subsequent requests from this session will carry the assigned cookie and the apache load balancer will always route the requests to the node, which advertised itself as the `jvmRoute` owner.

Instead of using apache, an integrated HTTP load balancing is also available. The SIP load balancer has an HTTP port where you could direct all incoming HTTP requests. The integrated HTTP load balancer behaves exactly like apache by default, but this behaviour is extensible and can be overridden completely with the pluggable balancer algorithms. The integrated HTTP load

balancer is much easier to configure and generally requires no effort, because it reuses most SIP settings ans assumes reasonable default values.

Unlike the native apache, the integrated HTTP load balancer is written completely in Java and does not support AJP, thus a performance penalty should be expected when using it. However, the integrated HTTP balancer has an advantage when related SIP and HTTP requests must stick to the same node.

1.3. Pluggable balancer algorithms

The SIP/HTTP load balancer exposes an interface to allow users to customize the routing decision making for special purposes. By default there are three built-in algorithms. Only one algorithm is active at any time and it is specified with the `algorithmClass` property in the configuration file.

It is completely up to the algorithm how and whether to support distributed architecture or how to store the information needed for session affinity. The algorithms will be called for every SIP and HTTP request and other significant events to make more informed decisions.

Note

Users must be aware that, by default requests explicitly addressed to a live server node passing through the load balancer will be forwarded directly to the server node. This allows for pre-specified routing use-cases, where the target node is known by the SIP client through other means. If the target node is dead, then the node selection algorithm is used to route the request to an available node.

The following is a list of the built-in algorithms:

org.mobicents.tools.sip.balancer.CallIDAffinityBalancerAlgorithm
: This algorithm is not distributable. It selects nodes randomly to serve a give Call-ID extracted from the requests and responses. It keeps a map with `Call-ID -> nodeId` associations and this map is not shared with other load balancers which will cause them to make different decisions. For HTTP it behaves like apache.

org.mobicents.tools.sip.balancer.HeaderConsistentHashBalancerAlgorithm
: This algorithm is distributable and can be used in distributed load balancer configurations. It extracts the hash value of specific headers from SIP and HTTP messages to decide which application server node will handle the request. Information about the options in this algorithms is available in the balancer configuration file comments.

org.mobicents.tools.sip.balancer.PersistentConsistentHashBalancerAlgorithm
: This algorithm is distributable and is similar to the previous algorithm, but it attempts to keep session affinity even when the cluster nodes are removed or added, which would normally cause hash values to point to different nodes.

1.4. Distributed load balancing

When the capacity of a single load balancer is exceeded, multiple load balancers can be used. With the help of an IP load balancer the traffic can be distributed between all SIP/HTTP load balancers based on some IP rules or round-robin. With consistent hash and `jvmRoute`-based balancer algorithms it doesn't matter which SIP/HTTP load balancer will process the request, because they would all make the same decisions based on information in the requests (headers, parameters or cookies) and the list of available nodes. With consistent hash algorithms there is no state to be preserved in the SIP/HTTP balancers.

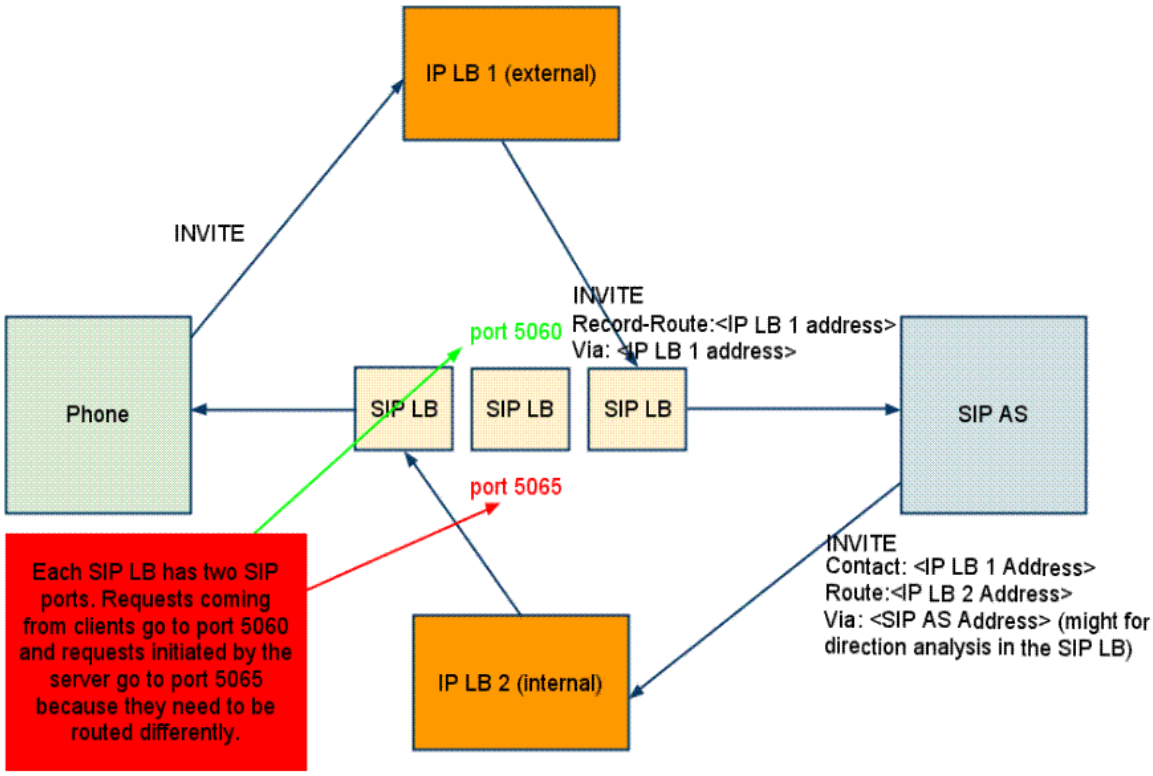

Figure 1.2. Example deployment scenario with IP load balancers serving both directions for incoming and outgoing requests in a cluster

1.5. Implementation of the JBoss Communications Load Balancer

Each individual JBoss Communications SIP Server in the cluster is responsible for contacting the SIP load balancer and relaying its health status and regular "heartbeats".

From these health status reports and heartbeats, the SIP load balancer creates and maintains a list of all available and healthy nodes in the cluster. The load balancer forwards SIP requests between these cluster nodes, providing that the provisioning algorithm reports that each node is healthy and is still sending heartbeats.

If an abnormality is detected, the SIP load balancer removes the unhealthy or unresponsive node from the list of available nodes. In addition, mid-session and mid-call messages are failed over to a healthy node.

The SIP load balancer first receives SIP requests from endpoints on a port that is specified in its Configuration Properties configuration file. The SIP load balancer, using a round-robin algorithm, then selects a node to which it forwards the SIP requests. The load balancer forwards all same-session requests to the first node selected to initiate the session, providing that the node is healthy and available.

1.6. SIP Message Flow

The JBoss Communications SIP load balancer appends itself to the `Via` header of each request, so that returned responses are sent to the SIP Balancer before they are sent to the originating endpoint.

The load balancer also adds itself to the path of subsequent requests by adding Record-Route headers. It can subsequently handle mid-call failover by forwarding requests to a different node in the cluster if the node that originally handled the request fails or becomes unavailable. The SIP load balancer immediately fails over if it receives and unhealthy status, or irregular heartbeats from a node.

In advanced configurations, it is possible to run more than one SIP load balancer. Simply edit the balancers connection string in your SIP Server - the list is separated with semi-colon.

Figure 1.3, "Basic IP and Port Cluster Configuration" describes a basic IP and Port Cluster Configuration. In the diagram, the SIP load balancer is the server with the IP address of `192.168.1.1`.

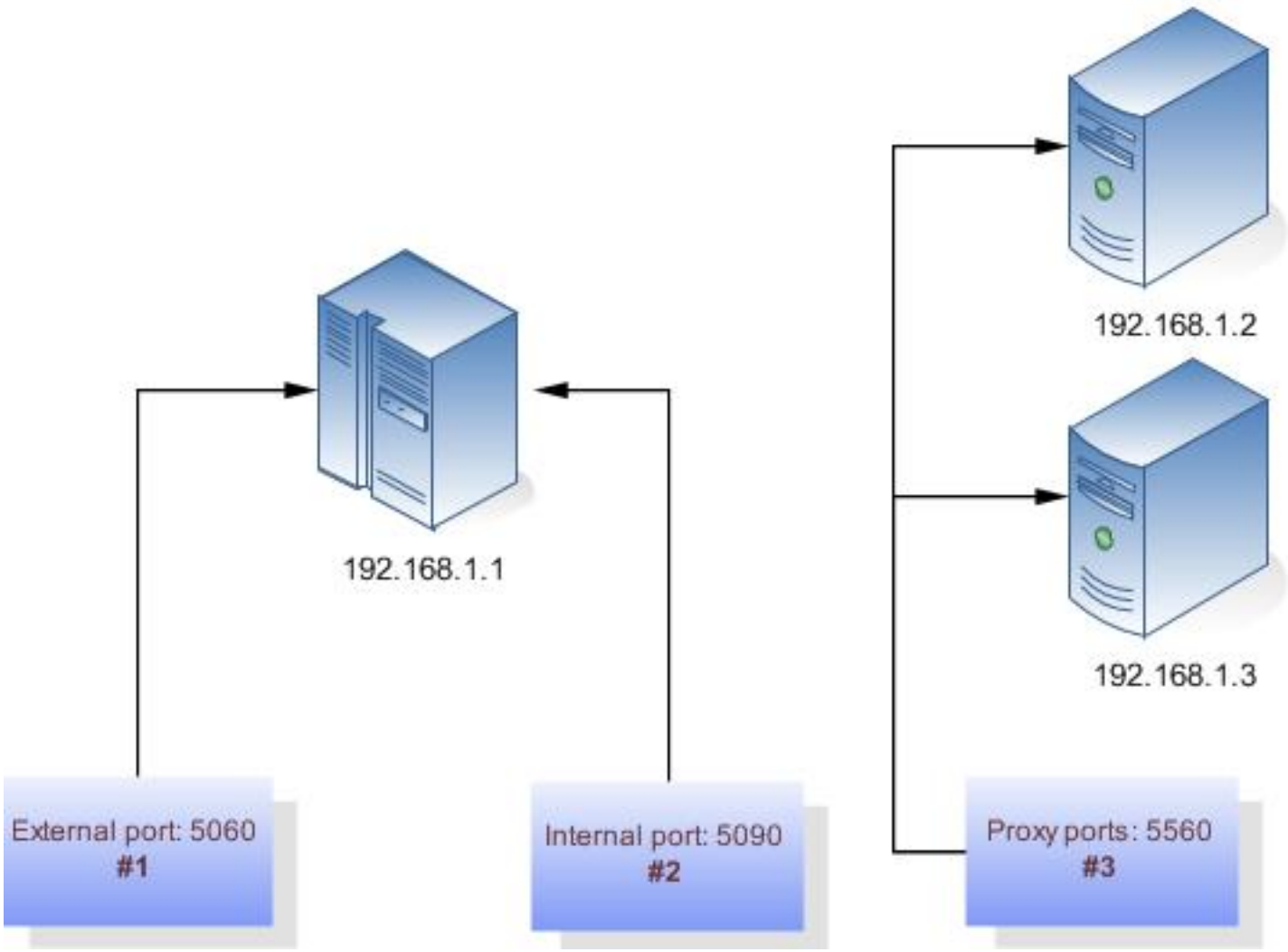

Figure 1.3. Basic IP and Port Cluster Configuration

1.7. SIP Load Balancer: Installing, Configuring and Running

1.7.1. Pre-Install Requirements and Prerequisites

Software Prerequisites

A JAIN SIP HA-enabled application server such as JBoss Communications JAIN SLEE or JBoss Communications SIP Servlets is required.

Running the SIP load balancer requires at least two instances of the application server as cluster nodes nodes. Therefore, before configuring the SIP load balancer, we should make

sure we've installed a the SIP application server first. The JBoss Communications SIP load balancer will work with a SIP Servlets-enabled JBoss Application Server *or* a JAIN SLEE application server with SIP RA.

SIP Servlets containers based on Tomcat are also supported but the session replication is not available there, thus mid-call failover will not work.

1.7.2. Downloading

The load balancer is located in the `sip-balancer` top-level directory of the JBCP distribution. You will find the following files in the directory:

SIP load balancer executable JAR file
: This is the binary file with all dependencies

SIP load balancer Configuration Properties file
: This is the properties files with various settings

1.7.3. Installing

The SIP load balancer executable JAR file can be placed anywhere in the file system. It is recommended that the file is placed in the directory containing other JAR executables, so it can be easily located in the future.

1.7.4. Configuring

Procedure 1.1. Configuring the JBoss Communications SIP Load Balancer and SIP Server Nodes

1. **Configure lb.properties Configuration Properties File**

   Configure the SIP load balancer's Configuration Properties file by substituting valid values for your personal setup. *Example 1.1, "Complete Sample lb.properties File"* shows a sample `lb.properties` file, with key element descriptions provided after the example. The lines beginning with the pound sign are comments.

Example 1.1. Complete Sample lb.properties File

```
# Mobicents Load Balancer Settings
# For an overview of the Mobicents Load Balancer visit http://docs.google.com/present/view?id=dc5jp5vx_89cxdvtxcm
# The Load balancer will listen for both TCP and UDP connections



# The binding address of the load balancer. This also specifies the
```

```
# default value for both internalHost and externalHost if not specified separately.
host=127.0.0.1

# The binding address of the load balancer where clients should connect (if the host property
 is not specified)
#externalHost=127.0.0.1

# The SIP port from where servers will receive messages
# delete if you want to use only one port for both inbound and outbound)
internalPort=5065

# The SIP port used where clients should connect
externalPort=5060

# The binding address of the load balancer where SIP application servers should connect (if
 the host property is not specified)
#internalHost=127.0.0.1

# The RMI port used for heartbeat signals
rmiRegistryPort=2000

# The HTTP port for HTTP forwarding
# if you like to activate the integrated HTTP load balancer, this is the entry point
httpPort=2080
#If no nodes are active the LB can redirect the traffic to the unavailableHost specified in this
 property,
#otherwise, it will return 503 Service Unavailable
#unavailableHost=google.com

# If you are using IP load balancer, put the IP address and port here
#externalIpLoadBalancerAddress=127.0.0.1
#externalIpLoadBalancerPort=111

# Requests initited from the App Servers can route to this address (if you are using 2 IP load
 balancers for bidirectional SIP LB)
#internalIpLoadBalancerAddress=127.0.0.1
#internalIpLoadBalancerPort=111

# The addresses in the SIP LB Via headers can be either the real addresses or those specified
 in the external and internal IP LB addresses
useIpLoadBalancerAddressInViaHeaders=false

# Designate extra IP addresses as serer nodes
#extraServerNodes=222.221.21.12:21,45.6.6.7:9003,33.5.6.7,33.9.9.2
```

```
# Call-ID affinity algortihm settings. This algorithm is the default. No need to uncomment it.
#algorithmClass=org.mobicents.tools.sip.balancer.CallIDAffinityBalancerAlgorithm
# This property specifies how much time to keep an association before being evitcted.
# It is needed to avoid memory leaks on dead calls. The time is in seconds.
#callIdAffinityMaxTimeInCache=500
#The following attribute specified the policy after failover. If set to true all calls from the failed
 node
#will go to a new healthy node (all calls to the same node). If set to false the calls will go to
 random new nodes.
#callIdAffinityGroupFailover=false

# Uncomment to enable the consistent hash based on Call-ID algorithm.
#algorithmClass=org.mobicents.tools.sip.balancer.HeaderConsistentHashBalancerAlgorithm
# This property is not required, it defaults to Call-ID if not set, cna be "from.user" or "to.user"
 when you want the SIP URI username
#sipHeaderAffinityKey=Call-ID
#specify the GET HTTP parameter to be used as hash key
#httpAffinityKey=appsession

# Uncomment to enable the persistent consistent hash based on Call-ID algorithm.
#algorithmClass=org.mobicents.tools.sip.balancer.PersistentConsistentHashBalancerAlgorithm
# This property is not required, it defaults to Call-ID if not set
#sipHeaderAffinityKey=Call-ID
#specify the GET HTTP parameter to be used as hash key
#httpAffinityKey=appsession

#This is the JBoss Cache 3.1 configuration file (with jgroups), if not specified it will use default
#persistentConsistentHashCacheConfiguration=/home/config.xml

# Call-ID affinity algortihm settings. This algorithm is the default. No need to uncomment it.
#algorithmClass=org.mobicents.tools.sip.balancer.CallIDAffinityBalancerAlgorithm
# This property specifies how much time to keep an association before being evitcted.
# It is needed to avoid memory leaks on dead calls. The time is in seconds.
#callIdAffinityMaxTimeInCache=500

# Uncomment to enable the consistent hash based on Call-ID algorithm.
#algorithmClass=org.mobicents.tools.sip.balancer.HeaderConsistentHashBalancerAlgorithm
# This property is not required, it defaults to Call-ID if not set, cna be "from.user" or "to.user"
 when you want the SIP URI username
#sipHeaderAffinityKey=Call-ID
#specify the GET HTTP parameter to be used as hash key
#httpAffinityKey=appsession
```

```
# Uncomment to enable the persistent consistent hash based on Call-ID algorithm.
#algorithmClass=org.mobicents.tools.sip.balancer.PersistentConsistentHashBalancerAlgorithm
# This property is not required, it defaults to Call-ID if not set
#sipHeaderAffinityKey=Call-ID
#specify the GET HTTP parameter to be used as hash key
#httpAffinityKey=appsession

#This is the JBoss Cache 3.1 configuration file (with jgroups), if not specified it will use default
#persistentConsistentHashCacheConfiguration=/home/config.xml


#If a node doesnt check in within that time (in ms), it is considered dead
nodeTimeout=5100
#The consistency of the above condition is checked every heartbeatInterval milliseconds
heartbeatInterval=150


#JSIP stack configuration.....
javax.sip.STACK_NAME = SipBalancerForwarder
javax.sip.AUTOMATIC_DIALOG_SUPPORT = off
# You need 16 for logging traces. 32 for debug + traces.
# Your code will limp at 32 but it is best for debugging.
gov.nist.javax.sip.TRACE_LEVEL = 0

// Specify if message contents should be logged.
gov.nist.javax.sip.LOG_MESSAGE_CONTENT=false

gov.nist.javax.sip.DEBUG_LOG = logs/sipbalancerforwarderdebug.txt
gov.nist.javax.sip.SERVER_LOG = logs/sipbalancerforwarder.xml
gov.nist.javax.sip.THREAD_POOL_SIZE = 64
gov.nist.javax.sip.REENTRANT_LISTENER = true
```

host
: Local IP address, or interface, on which the SIP load balancer will listen for incoming requests.

externalPort
: Port on which the SIP load balancer listens for incoming requests from SIP User Agents.

internalPort
: Port on which the SIP load balancer forwards incoming requests to available, and healthy, SIP Server cluster nodes.

rmiRegistryPort
: Port on which the SIP load balancer will establish the RMI heartbeat connection to the application servers. When this connection fails or a disconnection instruction is received, an application server node is removed and handling of requests continues without it by redirecting the load to the lie nodes.

httpPort
: Port on which the SIP load balancer will accept HTTP requests to be distributed across the nodes.

internalTransport
: Transport protocol for the internal SIP connections associated with the internal SIP port of the load balancer. Possible choices are `UDP`, `TCP` and `TLS`.

externalTransport
: Transport protocol for the external SIP connections associated with the external SIP port of the load balancer. Possible choices are `UDP`, `TCP` and `TLS`. It must match the transport of the internal port.

externalIpLoadBalancerAddress
: Address of the IP load balancer (if any) used for incoming requests to be distributed in the direction of the application server nodes. This address may be used by the SIP load balancer to be put in SIP headers where the external address of the SIP load balancer is needed.

externalIpLoadBalancerPort
: The port of the external IP load balancer. Any messages arriving at this port should be distributed across the external SIP ports of a set of SIP load balancers.

internalIpLoadBalancerAddresst
: Address of the IP load balancer (if any) used for outgoing requests (requests initiated from the servers) to be distributed in the direction of the clients. This address may be used by the SIP load balancer to be put in SIP headers where the internal address of the SIP load balancer is needed.

internalIpLoadBalancerPort
: The port of the internal IP load balancer. Any messages arriving at this port should be distributed across the internal SIP ports of a set of SIP load balancers.

extraServerNodes
: Comma-separated list of hosts that are server nodes. You can put here alternative names of the application servers here and they will be recognized. Names are important, because they might be used for direction-analysis. Requests coming from these server will go in the direction of the clients and will not be routed back to the cluster.

algorithmClass
: The fully-qualified Java class name of the balancing algorithm to be used. There are three algorithms to choose from and you can write your own to implement more complex routing behaviour. Refer to the sample configuration file for details about the available options for each algorithm. Each algorithm can have algorithm-specific properties for fine-grained configuration.

nodeTimeout
: In milliseonds. Default value is 5100. If a server node doesnt check in within this time (in ms), it is considered dead.

heartbeatInterval
: In milliseconds. Default value is 150 milliseonds. The hearbeat interval must be much smaller than the interval specified in the JAIN SIP property on the server machines - `org.mobicents.ha.javax.sip.HEARTBEAT_INTERVAL`

Note

The remaining keys and properties in the configuration properties file can be used to tune the JAIN SIP stack, but are not specifically required for load balancing. To assist with tuning, a comprehensive list of implementing classes for the SIP Stack is available from the *Interface SIP Stack page on nist.gov* [http://snad.ncsl.nist.gov/proj/iptel/jain-sip-1.2/javadoc/javax/sip/SipStack.html]. For a comprehensive list of properties associated with the SIP Stack implementation, refer to *Class SipStackImpl page on nist.gov* [http://snad.ncsl.nist.gov/proj/iptel/jain-sip-1.2/javadoc/gov/nist/javax/sip/SipStackImpl.html].

2. **Configure logging**

   The SIP load balancer uses *Java Logging as a logging mechanism* [http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html]. As such you can configure it through a property file and specify the property file to be used by using the following command `-Djava.util.logging.config.file=./lb-logging.properties`. Please refer to JDK logging for more informationon how to configure the Java logging.

1.7.4.1. Converged Load Balancing

1.7.4.1.1. Apache HTTP load balancer

The JBCP SIP load balancer can work in concert with HTTP load balancers such as `mod_jk`. Whenever an HTTP session is bound to a particular node, an instruction is sent to the SIP load balancer to direct the SIP calls from the same application session to the same node.

It is sufficient to configure `mod_jk` to work for HTTP in JBoss in order to enable cooperative load balancing. JBCP will read the configuration and will use it without any extra configuration. You can read more about configuring `mod_jk` with JBoss in your JBoss Application Server documentation.

1.7.4.1.2. Integrated HTTP load balancer

To use the integrated HTTP load balancer, no extra configuration is needed. If a unique `jvmRoute` is specified and enabled in each application server, it will behave exactly as the apache balancer. If `jvmRoute` is not present it will use session ID as a hash value and attempt to create sticky session. The integrated balancer can be used together with the apache balancer at the same time.

In addition to the apache behaviour, there is a consistent hash balancer algorithm that can be enabled for both HTTP and SIP messages. For both HTTP and SIP messages, there is a configurable affinity key, which is evaluated and hashed against each unassigned request. All requests with the same hash value will always be routed to the same application server node. For example, the SIP affinity key could be the callee user name and the HTTP affinity key could the “appsession” HTTP GET parameter of the request. If the desired behaviour group these requests, we can just make sure the affinity values (user name and GET parameter) are the same.

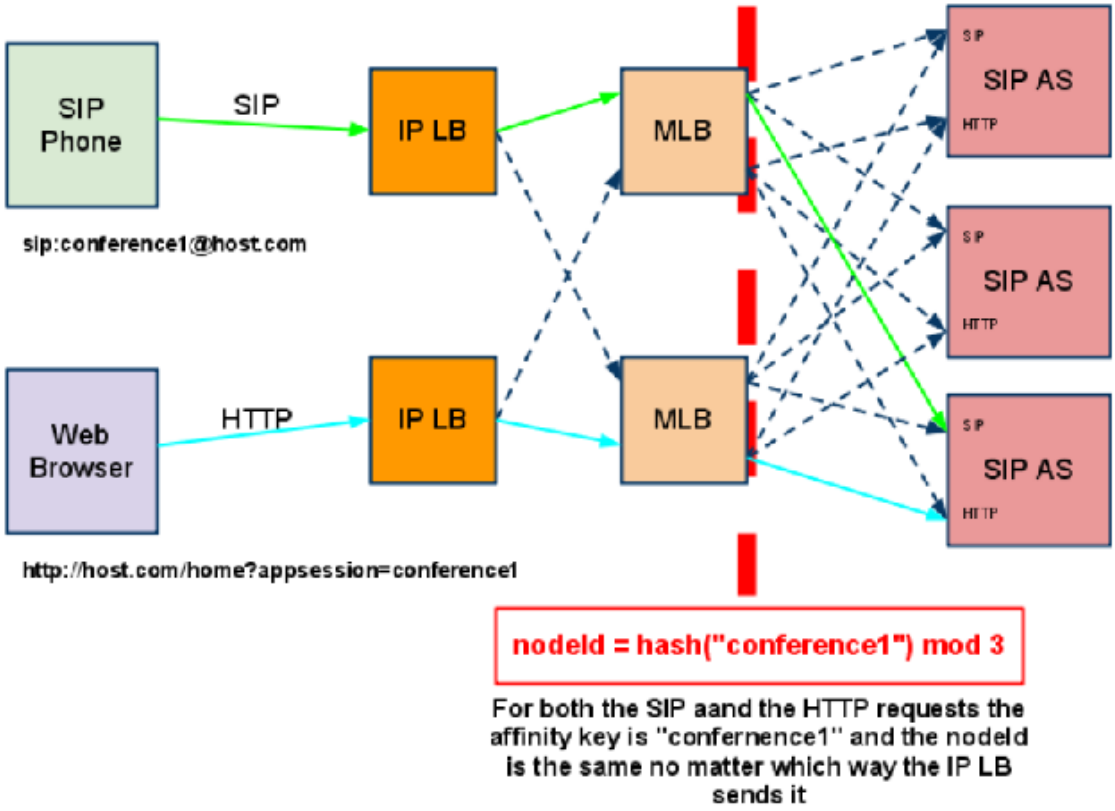

Figure 1.4. Ensuring SIP and HTTP requests are being grouped by common affinity value.

1.7.5. Running

Procedure 1.2. Running the SIP Load Balancer and SIP Server Nodes

1. **Start the SIP Load Balancer**

   Start the SIP load balancer, ensuring the Configuration Properties file (`lb.properties` in this example) is specified. In the Linux terminal, or using the Windows Command Prompt, the SIP Load Balancer is started by issuing a command similar to this one:

   ```
   java -jar sip-balancer-jar-with-dependencies.jar lb-
   configuration.properties
   ```

   Executing the SIP load balancer produces output similar to the following example:

   ```
   home]$ java -jar sip-balancer-jar-with-dependencies.jar lb-
   configuration.properties
   Oct 21, 2008 1:10:58 AM
    org.mobicents.tools.sip.balancer.SIPBalancerForwarder start
   INFO: Sip Balancer started on address 127.0.0.1, external port : 5060,
    port : 5065
   Oct 21, 2008 1:10:59 AM
    org.mobicents.tools.sip.balancer.NodeRegisterImpl startServer
   INFO: Node registry starting...
   Oct 21, 2008 1:10:59 AM
    org.mobicents.tools.sip.balancer.NodeRegisterImpl startServer
   INFO: Node expiration task created
   Oct 21, 2008 1:10:59 AM
    org.mobicents.tools.sip.balancer.NodeRegisterImpl startServer
   INFO: Node registry started
   ```

   The output shows the IP address on which the SIP load balancer is listening, as well as the external and internal listener ports.

2. **Configure SIP Server Nodes**

   The information about configuring your SIP Server, SIP Servlets or JAIN SLEE, is in the respective server User Guide.

3. **Start Load Balancer Client Nodes**

   Start all SIP load balancer client nodes.

1.7.6. Testing

To test load balancing, the same application must be deployed manually on each node. Two SIP Softphones must be installed.

Procedure 1.3. Testing Load Balancing with Sip Servlets

1. **Deploy an Application**

   Ensure that for each node, the DAR file location is specified in the `server.xml` file.

   Deploy the Location service manually on both nodes.

2. **Start the "Sender" SIP softphone**

   Start a SIP softphone client with the SIP address of **`sip:sender@sip-servlets-com`**, listening on port 5055. The outbound proxy must be specified as the sip-balancer (http://127.0.0.1:5060)

3. **Start the "Receiver" SIP softphone**

   Start a SIP softphone client with the SIP address of **`sip:receiver-failover@sip-servlets-com`**, listening on port 5090.

4. **Initiate two calls from "Sender" SIP softphone**

   Initiate one call from **`sip:sender@sip-servlets-com`** to **`sip:receiver-failover@sip-servlets-com`**. Tear down the call once completed.

   Initiate a second call using the same SIP address, and tear down the call once completed. Notice that the call is handled by the second node.

Procedure 1.4. Testing Load Balancing with JAIN SLEE and SIP RA

1. Deploy SIP RA
2. Configure the JAIN SIP HA properties for load balancing according to the JAIN SLEE User Guide
3. Deploy a sample application
4. Run the sample scenario for the application using the SIP Load Balancer

1.7.7. Stopping

Assuming that you started the JBoss Application Server as a foreground process in the Linux terminal, the easiest way to stop it is by pressing the **Ctrl**+**C** key combination in the same terminal in which you started it.

This should produce similar output to the following:

```
^COct 21, 2008 1:11:57 AM
 org.mobicents.tools.sip.balancer.SipBalancerShutdownHook run
```

```
INFO: Stopping the sip forwarder
```

1.7.8. Uninstalling

To uninstall the SIP load balancer, delete the JAR file you installed.

Appendix A. Revision History

Revision History

Revision 3.0 Thu Jun 11 2009 JaredMorgan<jmorgan@redhat.com>

Second release of the "parameterized" documentation.

Revision 2.0 Fri Mar 06 2009 DouglasSilas<dhensley@redhat.com>

First release of the "parameterized", and much-improved JBCP documentation.