

## SAILFIN CONVERGED LOAD BALANCER

A software interface for unified load balancing and failover of converged web and SIP applications deployed on the Java EE platform

White Paper  
January 2009

### Abstract

High-availability and scalability of applications and services are the cornerstones of enterprise deployments. Telecom service providers and enterprises that build and deploy telecommunication applications using the Java EE platform require a unified load balancing and failover mechanism for handling SIP and HTTP network traffic over their business-critical applications.



## Table of Contents

Introduction to the Converged Load Balancer.....	1
Deployment Topology.....	1
How to Create a Self-Load-Balancing Cluster .....	3
The <code>-dcrfile</code> option.....	3
The <code>-autocommit</code> option.....	3
Load Balancing Policy.....	3
Load Balancing Incoming Requests and Responses....	4
Incoming Requests (HTTP and SIP).....	4
Incoming Responses (SIP).....	5
Load Balancing Outgoing Requests and Responses....	5
Outgoing Requests (SIP).....	5
Outgoing Responses (HTTP and SIP).....	5
Health Monitoring of the SailFin Cluster .....	6
Upscale Example.....	7
.....	7
Downscale Example.....	8

**Converged Load Balancer helps telecom network companies drive down the cost of achieving their key goals of service reliability and availability.**

## Executive Summary

The telecommunication world is moving towards providing more enhanced and modernized Internet-based services to customers. The disparate worlds of the Web and telecom domains are converging for consumers and businesses. There is a convergence of communication channels over fixed and mobile networks onto an IP-based communication medium for developing and deploying services. IP Multimedia Subsystems (IMS) caters to the composite framework for achieving such convergence.

These converged services require load balancing and failover capabilities to guarantee seamless serviceability. The Converged Load Balancer addresses the need for high-availability for converged applications and services that are being developed and deployed on the Java EE platform.

The Converged Load Balancer is a software component that provides a competitive alternative to expensive, high-end, hardware-based solutions.

## Introduction to the Converged Load Balancer

JSR 289 defines the Session Initiation Protocol (SIP) Servlet specification for developing converged applications on the Java EE platform. This JCP-based standard articulates a standard mechanism for composing SIP servlets and Java EE components.

In some aspects, SIP request processing is similar to the HTTP-based Servlet specification for Web applications. However there are significant differences between SIP and HTTP servlet specifications. A web session is established over an HTTP client request-response pair. In a SIP application session, multiple client's request-responses are related to the same application session. There can be more than one response to a SIP request and a request can result in more than one SIP requests being created to service original request. Moreover, a SIP request can result in more than one application being invoked using application composition.

Thus, apart from handling SIP protocol, a load balancer based on HTTP protocol would need to be enhanced to cater such semantics of Converged, SIP applications. GlassFish offers an HTTP load balancer that is built on native stack. GlassFish HTTP load balancer provides advanced load balancing features for multi-tier deployments with a web server as the front-end. SailFin Converged Load Balancer focuses on single-tier deployments with the application server as the front-end.

## Deployment Topology

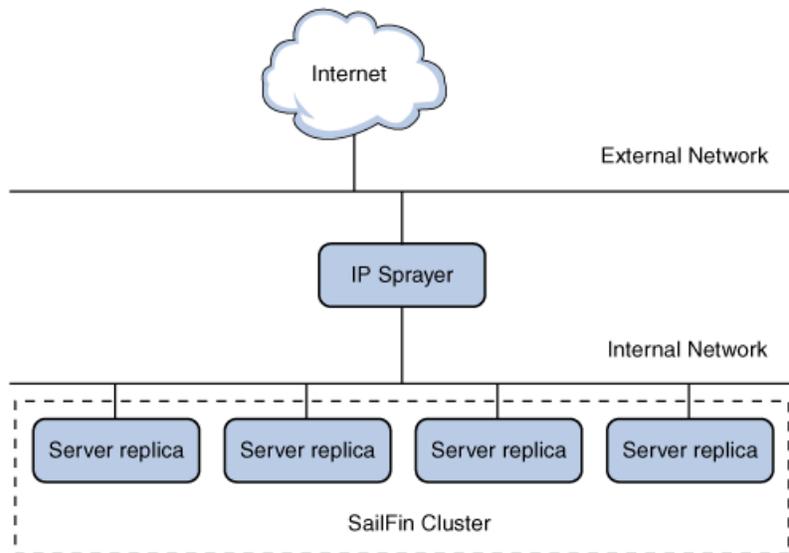
Converged Load Balancer is a Java-based component that runs in-process with SailFin (the supported version is called Sun GlassFish Communications Server).

With application and load balancing layers being hosted on a single SailFin cluster, Converged Load Balancer provides for a Self-Load-Balancing cluster deployment. In this

**Converged Load Balancer's salient points:**

- Load balancing and failover of messages
- Open Source component delivered with SailFin
- Solution for addressing request processing semantics defined by JSR 289.

type of deployment, a SailFin instance or server replica is part of a cluster in which all participating server instances have homogenous configurations. Each server replica acts as a fronting load balancing and failover system and hosts the applications on the same JVM, so that it can also participate in serving the load over the cluster.



*Figure 1: Self-load-balancing cluster*

The Self-Load-Balancing cluster can then be fronted by an IP sprayer such as the Linux Virtual Server (LVS), which is capable of handling HTTP and SIP protocols and acts as the gateway to the SailFin cluster for the network traffic.

Figure 1 provides a view of a Self-Load-Balancing SailFin cluster that uses virtual IP mechanism to front-end the network traffic and load balance the traffic over the back-end SailFin cluster. In this deployment, the messages are tunneled through the IP sprayer over to a SailFin server replica, which then responds directly to the originating client of the message.

Converged Load Balancer also provides for heterogeneous application deployment. In a heterogeneous application deployment, converged applications and SIP applications can be deployed, load balanced, and failed over alongside pure web applications. The Converged Load Balancer detects the presence of such application types and accordingly applies an appropriate load balancing algorithm.

You can use the Admin Console or the Command-Line Interface to create a self-load balancing cluster.

## How to Create a Self-Load-Balancing Cluster

Create a SailFin cluster called `my-cluster`.

Thereafter, the you can use the following CLI command to configure Converged Load Balancer in Self-Load-Balancing mode.

```
<install-dir>/bin> asadmin create-converged-lb --target my-cluster -selfloadbalance=true --dcrfile=<location-of-dcr-file> --autocommit=true my-clb
```

This command creates a Converged Load Balancer on the target, `my-cluster`. In addition, this command specifies that `my-cluster` is the target for load balancing. Effectively, this implies that `my-cluster` is the host for creating the load balancer in addition to being the target for load balancing the network traffic.

### The `--dcrfile` option

The command specifies the `--dcrfile` option, which defines the Data Centric Rules to be used by Converged Load Balancer for applying the load balancing policy. These rules are explained in detail in the Data Centric Rules section.

### The `--autocommit` option

Another interesting option is the `--autocommit` option, which is used as a flag to indicate to the administration framework, if every configuration change to `my-cluster` that results in a change in configuration as viewed by the Converged Load Balancer, should generate a Converged Load Balancer configuration. This flag can be judiciously set to false during the establishing phase of configuration, and then set to true, once the configuration is established and before staging the cluster. Converged Load Balancer records the changes to its configuration by promoting another configuration image captured in its configuration file.

Changes are recorded by using a version-based configuration file. The naming convention for Converged Load Balancer configuration file is: `converged-load-balancer.xml.<vn>`, where `n = {1,2,.....}`.

## Load Balancing Policy

The load balancing algorithm or policy at application level defines the following:

- Pure web applications are load balanced using Round Robin algorithm
- SIP applications are load balanced using Consistent Hash algorithm. Data Centric Rules are used, if specified.

### The Converged Load Balancer's load balancing policy applies :

- Round Robin algorithm
- Consistent Hash algorithm
- ➔ using over out-of-box request entities : from-tag, call-id
- OR
- ➔ Data-centric rules

- Converged applications are load balanced using Consistent Hash algorithm over both HTTP and SIP messages, if Data Centric Rules are specified. Otherwise, Converged Load Balancer uses the Round Robin algorithm, by default, for HTTP requests and Consistent Hash for SIP messages. It is critical that you specify Data Centric Rules for such deployments to ensure that HTTP and SIP messages for the application are load balanced to the appropriate server replica in the cluster.

If a Data Centric Rules file is not specified, and the application is a converged or SIP application, Converged Load Balancer applies Consistent Hash policy over SIP messages by using the concatenation of the following message entities as the key:

*SIP message : call-id, from-tag*

This format uniquely identifies a call leg over SIP invocation. Round Robin algorithm is used to distribute the load over HTTP invocations. Although this policy definition provides an out-of-the-box, simple-to-specify solution, it does not necessarily provide the the correct distribution based on the kind of application deployed. For example, in the case of a peer-to-peer SIP call, it would be fine to associate the call-id, from-tag concatenation as the key to determine the server replica to direct the message to. However, this policy would not suit a converged or SIP conferencing application in which multiple callers are expected to join the same real-time session. If you use this policy, each fresh invocation would be directed to different server replica over web path and probably over SIP path of invocation.

As a result, it is critical to define the Data Centric Rules to ensure the correct semantics of load balancing and failing over of messages over converged and SIP invocation.

## Load Balancing Incoming Requests and Responses

A converged application involves incoming requests and responses, based on whether deployment is playing the role of Proxy, of User Agent Service (UAS)/ User Agent Client (UAC) or both, or of a B2BUA. In the case of Proxy, B2BUA or UAC, SailFin cluster also receives incoming responses.

If Data Centric Rules are specified, Converged Load Balancer applies the rules on the requests to determine the key to be used for the Consistent Hash algorithm. Based on the application protocol, the Converged Load Balancer applies the HTTP or SIP rules appropriately. If the incoming message contains the sticky information queried for by the Converged Load Balancer, this information is used for selecting the server replica to dispatch the message to.

### Incoming Requests (HTTP and SIP)

These requests are verified against the presence of following :

- BERoute (applicable over HTTP): The encoded value of the server instance to

which this message needs to be dispatched.

- **BEKey (applicable over HTTP and SIP):** The string that represents the key, which is queried for and obtained if the Data Centric Rule was to be applied again.

Stickiness attributes indicate to the Converged Load Balancer that the incoming message has an affinity to a specific server replica as part of an ongoing request-response message processing. Such requests that already contain sticky information are commonly referred to as *subsequent* requests.

### Incoming Responses (SIP)

These responses arrive directly from the UAS to the server replica that dispatched the request. They are directed to the correct replica if the connection was lost between UAC (replica) and UAS. In such a scenario, the UAS, in accordance to principles of RFC 3261, dispatches the response to the topmost *VIA*. This *VIA* points to the entry point of the SailFin cluster, through which all the network traffic reaches the system. Converged Load Balancer stamps a sticky information on this *VIA* as part outgoing request and retrieves this information as part of directing the incoming response to correct replica. This sticky information is encoded as a parameter called *BERoute*.

## Load Balancing Outgoing Requests and Responses

### Outgoing Requests (SIP)

These requests are dispatched directly from the server replica, with the Converged Load Balancer stamping stickiness details to ensure that responses to the message are load balanced back to the correct server replica. This scenario occurs if the connection between UAC (server replica) and UAS is lost. *BERoute* is stamped on topmost *VIA* for routing of incoming responses.

### Outgoing Responses (HTTP and SIP)

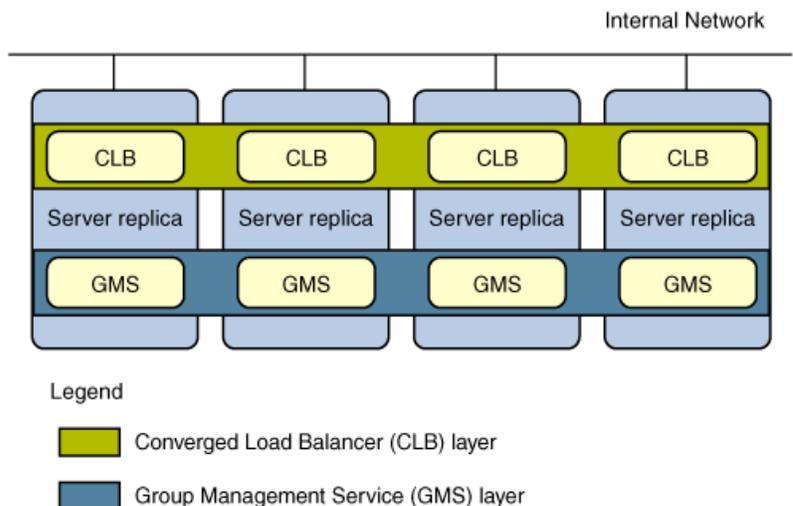
These messages are dispatched to the client / UAC by the replica that originally received the request. This is the replica on which the request first landed. Converged Load Balancer recognizes the scenario in which the incoming request of the request-response pair was serviced by another server replica than the one that first received it. This scenario occurs as part of the task of load balancing the request in a self-load balanced cluster. The Converged Load Balancer dispatches the response to the original server replica, which, in turn, dispatches it to the client. *BEKey* is stamped on this response to ensure that subsequent requests are sticky to the server replica that serviced the first request. *BEKey* is used when the HTTP and or SIP message, gets load balanced using the Consistent Hash policy. *BERoute* would be stamped in case the HTTP request was load balanced using Round Robin algorithm.

## Health Monitoring of the SailFin Cluster

Converged Load Balancer uses the services of Group Management Service (GMS), for tracking the availability of server replicas in the cluster. The Converged Load Balancer registers itself with the GMS service on the cluster to source the membership changes that occur in the cluster. While doing so, Converged Load Balancer views the replicas from the prism of the states that the replicas are in, in order to determine whether they can join in load balancing of network traffic.

For a server replica to join load balancing, it needs to be in GMS READY or READYANDALIVE state, as part of a `JoinNotificationSignal`, `JoinAndReadyNotificationSignal` notification from GMS.

These states are indication of the fact that the replica has successfully started and is now ready to service messages. This state is achieved upon startup of server replica and further successful startup of all deployed applications .



**Figure 2: Health monitoring in a self-load-balancing cluster**

Figure 2 is in continuation to Figure1. This figure specifically showcases Converged Load Balancer over the Self-Load-Balancing SailFin cluster and GMS, conceptually working as a single substrate over the cluster.

Converged Load Balancer also listens for notification dispatched by GMS over the cluster indicating departure on a replica and these include `FailureNotificationSignal` and `PlannedShutdownSignal`.

GMS is a subsystem of the SailFin server. GMS manages the clustering of server replicas. It detects join, failure, and removal of server replica as part of dynamic membership changes to a SailFin cluster.

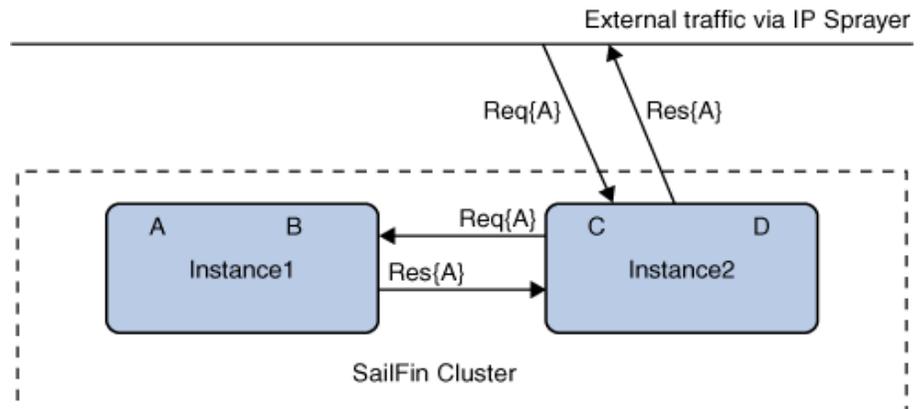
## Upscaling or Downscaling SailFin Cluster

The user can dynamically add or remove instances from the cluster by issuing the following commands :

- `asadmin create-instance`
- `asadmin delete-instance`

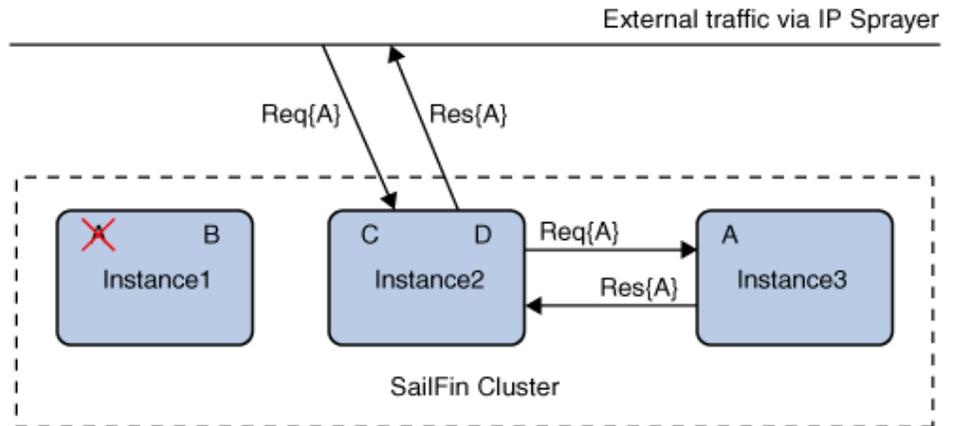
Such a change in cluster membership has impact on the traffic being load balanced for converged and SIP applications using the Consistent Hash policy. In this scenario, the load gets redistributed in the as explained in the following examples , In these examples, Instance1, Instance12, and Instance3 represent server replicas in a SailFin cluster.

### Upscale Example



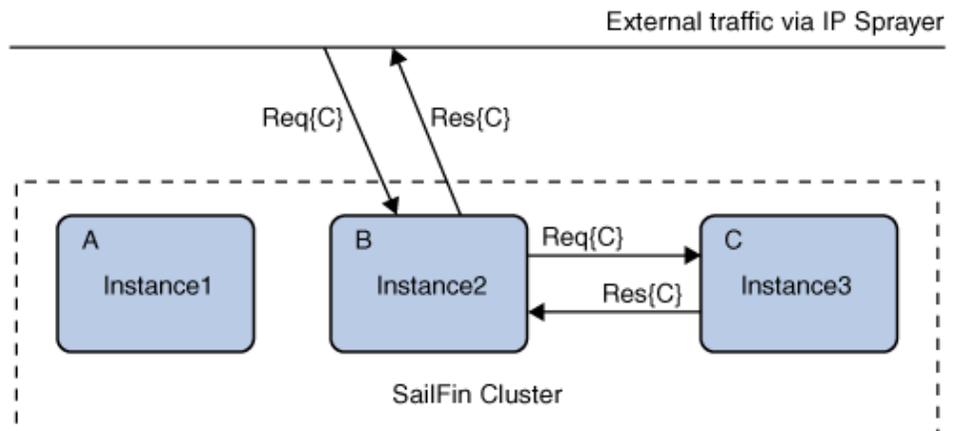
*Figure 3:SailFin cluster with hashkey distribution before upscale*

Consider a cluster, as shown in Figure 3, with instance Instance1, servicing and bound to messages with hash keys A and B. This cluster has another instance called Instance2, which services hash keys C and D. See Figure 3, where Req{A} and Res{A} represent the request associated with hash key A and the related response respectively. To this cluster, add a third instance, Instance3. A could move to Instance3 but it never moves to Instance2. Refer Figure 4.



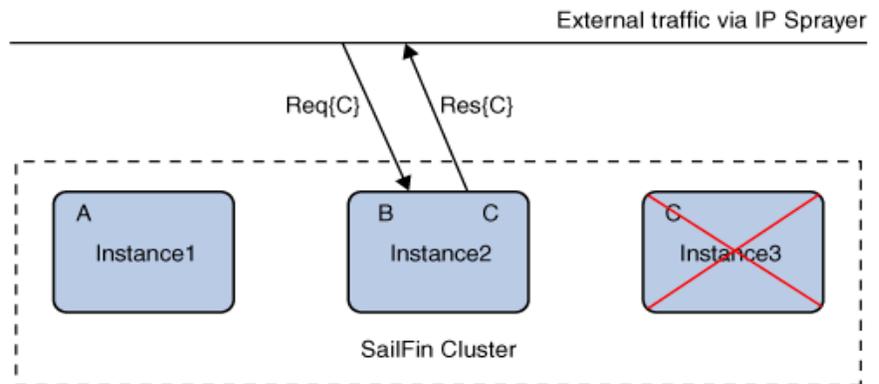
*Figure 4: SailFin cluster with hashkey distribution after upscale*

#### Downscale Example



*Figure 5: SailFin cluster with hashkey distribution before downscale*

Hash keys A, B, and C are bound to instances, Instance1, Instance2, and Instance3, respectively. See Figure 5, where Req{C} and Res{C} represent the request associated with hash key C and the related response respectively. If Instance3 fails, C is bound to Instance1 or Instance2. However, A would never move to Instance2, nor would B move to Instance1. Refer Figure 6, where C is shown to be bound to Instance2 upon failover. This example showcases the consistent behavior of the Converged Load Balancer.



*Figure 6: SailFin cluster with hashkey distribution after downscale*

## Data Centric Rules

Converged Load Balancer can act as a front-end for various types of applications deployed on SailFin. Applications can vary from pure web applications to pure SIP applications to converged applications (having both SIP and Web components). If Data Centric Rules are not provided, HTTP requests are load-balanced using round-robin policy and SIP requests are load-balanced using Consistent Hash policy. So it is essential to provide Data Centric Rules, in the form of an XML file before deploying converged applications. Providing Data Centric Rules enforces Consistent Hash policy to be used for SIP and HTTP messages meant for converged applications. Data Centric Rules are required even if only pure SIP applications are deployed, because the default headers that are used to generate hash key utilized by the Consistent Hash policy may not provide desired distribution of SIP requests.

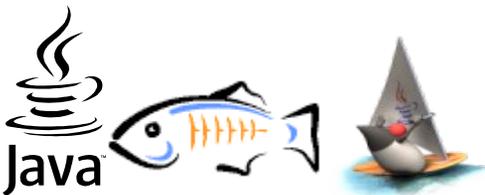
Consistent Hash policy distributes requests based on the generated hash key using headers or parameters of SIP/HTTP requests. Using Data Centric Rules, you can define the headers or parameters that are used to generate the hash key. If none of the rules match the SIP/HTTP request, default headers are used to generate hash key from the request.

A sample DCR file is provided here:

```
<?exempt version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE user-centric-rules PUBLIC "-//Sun Microsystems Inc.//DTD
SailFin 1.0//EN" "http://www.sun.com/software/appserver/dtds/sun-
data-centric-rule_1_0.dtd">
<user-centric-rules>
  <!--
    Extract hash key from sip requests using following rules
    1. If Conference-Name header exists in request , then use
```



applications. The Converged Load Balancer facilitates the development and deployment of reliable and scalable business-critical, JSR 289 -compliant telecommunications applications.



Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA Phone 1-650-960-1300 or 1-800-555-9SUN (9786) Web [sun.com](http://sun.com)

