1. Introduction

1.1. Project/Component Working Name: Sun Java System Application Server Loadbalancer enhancements for GlassFish v2 (SJSAS 9.1)

1.2. Name of Document Author/Supplier:

Pankaj Jairath : pankaj.jairath@sun.com (Earlier inputs from Harsha R A, Nazrul Islam, Satish Viswanatham, Sanjeev Krishnan and Sreeram Duvur)

1.3. Date of This Document:

08/09/2006

2. Project Summary

2.1. Project Description:

Loadbalancer component of the application server is a webserver plugin which distributes the http requests to the application server instances. Currently it only supports simple round robin load balancing policy.

This one pager describes the enhancements that are proposed to this component that include rule based load balancing such as weighted round robin and user defined decision. It also describes ease of administration features.

2.2. Risks and Assumptions:

Increases the complexity of the component.

3. Problem Summary

3.1. Problem Area:

This could solve many of the customers problems related to load distribution and administration such as

1. Customer having hardware of differing capacities would like to distribute more load to the more powerful hardware's application server instance.

2. If one particular appserver instance is overloaded, customer would like to distribute less load to that instance and use the other less occupied instances instead.

3. Each time a new application is deployed, the administrator has to
manually generate the loadbalancer.xml file and copy it to the web tier.
Automatic update of the configuration from DAS to loadbalancer would
would ease this task.

4. Customer would like to implement a custom routing logic like identity
based routing or geographical location based routing.

3.2. Justification:

This would enhance customer satisfaction and make it more competitive.

4. Technical Description:

4.1. Details:

4.1.1. Load balancing enhancements

This section describes the enhancements that are proposed to the
component that include weighted round robin, user defined decision and
the ease of administration.

4.1.1.1. Weighted Round Robin

We introduce an optional attribute called weight for the instance
element of Loadbalancer.xml as given below. In addition, a new optional
attribute called policy for the cluster element would specify the
policy that would be applicable.

```
<loadbalancer name="loadbalancer1" >
 <cluster name="cluster1" policy="weighted-round-robin">
    <instance name="instance1" enabled="true" listeners="http://abc.com:80"
    weight="100" />
    <instance name="instance2" enabled="true"
    listeners="https://abc2.com:80" weight="400"/>
    <web-module
    context-root="fortune" enabled="true" error-url="error1.html"/>
    <health-checker url="/" interval-in-seconds="10" />
 </cluster>
</loadbalancer>
```

When this is specified, the loadbalancer would route the requests according to the weight. For every 500 requests, 100 will go to instance1 and 400 would go to instance2. The default weight will be 100.
The weight would be assigned to each instance from the admin gui/cli.
domain.xml will have an attribute for every instance indicating the weight. The weight has to be integer.

The disadvantage to this method is that the weights are static and the administrator has to calculate the instance weights appropriately.
Using the self management framework, user could write rules to alter the weights dynamically. The changes would be pushed to the loadbalancer using the automatic push feature.

4.1.1.2. User Defined LB Decision

This allows customers to have a custom logic for load balancing. Examples would be user identity based redirect, mime based load balancing etc. The user of this feature would have to develop a shared library which would be loaded by load balancer. The loaded custom shared library would implement the interface as defined in loadbalancer.h which will be placed in <appserver install dir>/lib/install/templates/.

The method lb_policy_init would be called by the load balancer whenever it starts up and has a list of active instances and also whenever this list changes either with a healthy instance becoming unhealthy or vice versa.

When a request arrives at loadbalancer, it first matches the request with the configured context roots. If there is no match, control is returned to the webserver as done at present.

The lb_decision would be called for every request which requires a selection of an instance. The lb_decision is not called for stuck requests.The method returns the name of the selected listener.

The loadbalancer configuration for this policy would look like

```
<loadbalancer name="loadbalancer1">
 <cluster name="cluster1" policy="user-defined"
```

```
                        policy-module="/path/lbmodule.so" >
 </cluster>
</loadbalancer>
```

A sample implementation will be shipped which will implement simple round robin policy.

4.1.2. Administration ease of use:

Currently the loadbalancer.xml has to be manually copied to the webserver's config directory. Enhancements to make it automatic between the appserver and the webserver can be done using the push approach.

(Advantages are marked with + and disadvantages with -)

DAS pushes the xml to load balancer
---> Can use SSL mutual authentication if SSL is configured on the webserver
--->(-) Need special virtual server/listener/NSAPI entry/admin port on the webserver side to accept the connection
--->(-) Needs a hole in the firewall for every LB instance for outgoing connection from DAS to LB. This can be prevented by configuring a proxy.
--->(+) There is no need of polling, can be event based(deploy,undeploy)

In the initial phase of the implementation, an asadmin command would push the configuration to the loadbalancer. Later, an integration with the loadbalancer SPI would allow automatic pushes based on the lb config change.

There will be an element for configuring the actual loadbalancers in domain.xml . The administrator has to configure the loadbalancer's endpoint details like host, port, ssl certificates, proxy host, proxy port.

The loadbalancer will trap the special context root configured for configuration update. For eg, it could be /lbconfigupdate . It will accept the contents of the loadbalancer.xml in the post body. It will verify the credentials of DAS before accepting the contents. Then it will parse the incoming xml file, and if found to be valid, it will take a backup of existing xml file and replace it with the updated file.

Configuration Steps:

Documentation would be provided to enlist the steps required to manually install and configure the loadbalancer plugin. For GlassFish v2, Sun's Web Server would be the supported platform.

The Web Server could be configured to enforce client authentication only for the path "/lbconfigupdate" which is what DAS uses to post config updates and which is the only path from which LB accepts the push.

4.1.3   Monitoring

The following diagram shows the hierarchical tree structure of the Load Balancer Statistical information.

```
load-balancer
|
+---Cluster 1
| +--Instance 1
| | +----health
| | +----num-active-requests
| | +----num-total-requests
| | +---Application 1
|         +-- ContextRoot 1
|               +------------average-response-time
|               +------------max-response-time
|               +------------min-response-time
|               +------------num-failover-requests
|               +------------num-error-requests
|               +------------num-idempotent-url-requests
|               +------------num-active-requests
|               +------------num-total-requests
...
| +--Instance 2
| | +----health
| | +----num-active-requests
| | +----num-total-requests
| | +---Application 2
|         +-- ContextRoot 2
|               +------------average-response-time
|               +------------max-response-time
|               +------------min-response-time
|               +------------num-failover-requests
|               +------------num-error-requests
|               +------------num-idempotent-url-requests
|               +------------num-active-requests
```

```
   |               +------------num-total-requests
```

Monitoring is enabled per Load Balancer configuration. It is turned on by
setting required-monitoring-data to true. However setting Log verbose option
is not required.
Monitoring data can be obtained from DAS using GUI. These monitoring
information can also be obtained programmatically using AMX
Monitoring APIs.

PE/EE Impact

Impacts EE.

4.2. Bug/RFE Number(s):

4.3. In Scope:

4.4. Out of Scope:

4.4.1 64 bit support
32 bit version is provided with GlassFish v2. As part of
Sun Java System Application Server 9.1 EE, 64-bit support would be
considered.

4.4.2 Response time based round robin

This is based on the response time of just the URL that
established the session, so it is not quite powerful.

4.4.3 Load metric based load balancing

This will not be done in this release.

4.4.4 Prevention of Stale Session modification

The session persistence layer will initiate a takeover of the
session by updating the owner column of the HADB table for the
sessions that have failed over. The details of this is out of
scope of this one pager.

The loadbalancer will not initiate any action to alter the state
of the session or the instance.

4.5. Interfaces:

The interfaces for SJSAS 7.0EE and 7.1EE LB are applicable for GlassFish v2 as well.
Hence in this section we document the interfaces that have changed
and the new interfaces that have been added.

4.5.1 Exported Interfaces

Interface: sun-loadbalancer_1_2.dtd
Stability: Evolving
A new optional attribute called policy and policy-module is introduced for the cluster element. A new attribute called weights is added to instance element. As these are an optional
attribute, there will be no backward compatibility issues. The
other differences are listed below.

```
---------------------------------------------------------------------------
---
+<!ENTITY % policy "(round-robin | weighted-round-robin | user-defined )">


-<!ATTLIST cluster name CDATA #REQUIRED>
+<!ATTLIST cluster name CDATA #REQUIRED
+       loadbalancer policy %policy; "round-robin"
+       policy-module CDATA "">

 <!ATTLIST instance   name             CDATA     #REQUIRED
                      enabled          %boolean; "true"
                      disable-timeout-in-minutes  CDATA     "31"
-                     listeners        CDATA     #REQUIRED>
+                     listeners        CDATA     #REQUIRED
+                     weight        CDATA     "100">
```

Interface: User Defined LB Policy Interface (loadbalancer.h)
Stability: Unstable
The C interface implemented by the user's shared library will
be documented and supported, but the interface is not stable and could
undergo changes.

```
        #ifndef LOADBALANCER_H
        #define LOADBALANCER_H
        struct http_listener {
                char * name;
                char * url;
                int weight;
        };
        struct header {
                char * name;
                char * value;
        };

        #ifdef __cplusplus
        extern "C" {
        #endif
        int lb_policy_init(struct http_listener[] listeners, int size);
        char* lb_decision(int secure,char *url, struct header[] headers,int
size);

        #ifdef __cplusplus
        }
        #endif
        #endif  // LOADBALANCER_H
```

    Interface: domain.xml
    Stability: Evolving

```
        <!ATTLIST server
        ....
+        lb-weight CDATA "100">
```

For load-balancing policy, we introduce policy and policy-module attributes for cluster-ref element of lb-config.

```
        <!ATTLIST cluster-ref
        ref CDATA #REQUIRED
+       lb-policy ("round-robin" | "weighted-round-robin" | "user-
defined" )
                        "round-robin"
+       lb-policy-module CDATA #IMPLIED>
```

Configuration Support for Physical Load-balancers

```
-       <!ELEMENT domain (applications?, .....,..)>
+       <!ELEMENT domain (applications?, ....., load-balancers?...)>
+       <!ELEMENT load-balancers (load-balancer*)>
+       <!ELEMENT load-balancer (property*)>
+       <!-- load-balancer attributes
+
+       name - name of the load balancer
+       config-ref - name of the lb-config used by this load balancer
+       automatic-lb-apply-enabled - immediately push changes to lb config
to
+               the physical load balancer
+       properties:
+       device-host - Host name or IP address for the device
+       device-admin-port - Device administration port number
+       ssl-proxy-host - proxy host used for outbound HTTP
+       ssl-proxy-port - proxy port used for outbound HTTP
+
+       -->
+
+   <!ATTLIST load-balancer
+   name CDATA #REQUIRED
+   config-ref CDATA #REQUIRED
+   automatic-lb-apply-enabled %boolean; "false">
---------------------------------------------------------------------------
----

            Interface: Loadbalancer screens in Admin GUI
            Stability: Evolving
            Comments: Admin GUI would provide new screens to support the
ease of
                      use enhancements.

            Interface: Loadbalancer commands in Admin CLI
            Stability: Evolving
            Comments: CLI would provide new commands to support the ease of
use
                      enhancements.

        New Commands:

        asadmin create-http-lb
                --config lb_config_name
                [--autoapplyenabled=false]
                --devicehost device_host_or_ip
                --deviceport device_port
                [--sslproxyhost proxy_host]
                [--sslproxyport proxy_port]
                [--property (name=value)[:name=value]*]
                <load-balancer-name>

        asadmin delete-http-lb <load-balancer-name>

        asadmin list-http-lbs

        asadmin apply-http-lb-changes <lb-name>

        asadmin configure-http-lb-config
                [--responsetimeout=60]
                [--httpsrouting=false]
                [--reloadinterval=60]
                [--monitor=false]
                [--routecookie=true]
                [--healthcheckerurl url]
                [--healthcheckerinterval=30]
                [--healthcheckertimeout timeout]
                [--target target]
                [--config config_name]
                [xml-file-name]

        asadmin configure-lb-weight
                --cluster cluster_name
                <instance-name=weight[:instance-name=weight]>
```

Changes to Existing Commands:
create-http-lb-ref will have the following new options

```
[--lbpolicy lb_policy] [--lbpolicymodule lb_policy_module]
[--healthcheckerurl url] [--healthcheckerinterval=30]
[--healthcheckertimeout=10] [--lbEnableAllInstances] [--
lbEnableAllApplications]
```

create-http-lb-config will have the following new option
--property

Interface: Loadbalancer interfaces in AMX
Stability: Evolving
Comments:
The following are the new classes in AMX for load balancer.

```
LoadBalancer
LoadBalancerConfig
LBConfig
LBConfigHelper
LoadBalancerApplicationMonitor
LoadBalancerApplicationStats
LoadBalancerClusterMonitor
LoadBalancerClusterStats
LoadBalancerConfigKeys
LoadBalancerContextRootMonitor
LoadBalancerContextRootStats
LoadBalancerMonitor
LoadBalancerServerMonitor
LoadBalancerServerStats
```

4.5.2 Imported interfaces

4.5.3 Other interfaces (Optional)

4.6. Doc Impact:
The loadbalancer admin guide, Error Reference Manual will be

impacted.

4.7. Admin/Config Impact:
Changes to GUI and CLI to support the administration of the enhancements.

4.8. HA Impact:
It increases the usability options available for HA.

4.9. I18N/L10N Impact:
None

4.10. Packaging & Delivery:
Loadbalancer component would be avialable as part of the build and
steps would be provided to manually install and configure it for Sun Java System Web Server.

4.11. Security Impact:
The authentication between the LB and the DAS/instance needs to be reviewed.

4.12. Compatibility Impact

4.13. Dependencies:
The admin CLI/GUI need to support the new features.

5. Reference Documents:

6. Schedule:

6.1. Projected Availability:
With GlassFish v2 (SJSAS 9.1)