Improve Apache Load Balancing with mod_cluster - Apaches to Tomcats Application servers Get Better Load Balancing

Author : admin

Earlier I've blogged on How to set up Apache to to serve as a Load Balancer for 2, 3, 4 etc. Tomcat / other backend application servers with mod_proxy and mod_proxy_balancer, **however though default Apache provided mod_proxy_balancer works fine most of the time,** ***If you want a more precise and sophisticated balancing with better load distribuion you will probably want to install and use mod_cluster instead.***

So what is Mod_Cluster and why use it instead of Apache proxy_balancer ?

Mod_cluster ***is an innovative Apache module for HTTP load balancing and proxying. It implements a communication channel between the load balancer and back-end nodes to make better load-balancing decisions and redistribute loads more evenly.***

Why use mod_cluster instead of a traditional load balancer such as Apache's mod_balancer and mod_proxy or even a high-performance hardware balancer?

Thanks to its unique back-end communication channel, mod_cluster takes into account back-end servers' loads, and thus provides better and more precise load balancing tailored for JBoss and Tomcat servers. Mod_cluster also knows when an application is undeployed, and does not forward requests for its context (URL path) until its redeployment. And mod_cluster is easy to implement, use, and configure, requiring minimal configuration on the front-end Apache server and on the back-end servers.

So what is the advantage of mod_cluster vs mod proxy_balancer ?

Well here is few things that turns the scales in favour for mod_cluster:

- ***advertises its presence via multicast so as workers can join without any configuration***
- ***workers will report their available contexts***
- ***mod_cluster will create proxies for these contexts automatically***
- ***if you want to, you can still fine-tune this behaviour, e.g. so as .gif images are served from httpd and not from workers...***
- ***most importantly: unlike pure mod_proxy or mod_jk, mod_cluster knows exactly how much load there is on each node because nodes are reporting their load back to the balancer via special messages***
- ***default communication goes over AJP, you can use HTTP and HTTPS***

1. How to install mod_cluster on Linux ?

You can use mod_cluster either with JBoss or Tomcat back-end servers. We'll install and configure mod_cluster with Tomcat under CentOS; using it with JBoss or on other Linux distributions is a similar process. I'll assume you already have at least one front-end Apache server and a few back-end Tomcat servers installed.

To install mod_cluster, first download the latest mod_cluster httpd binaries. Make sure to select the correct package for your hardware architecture – 32- or 64-bit.
Unpack the archive to create four new Apache module files: mod_advertise.so, mod_manager.so, mod_proxy_cluster.so, and mod_slotmem.so. We won't need mod_advertise.so; it advertises the location of the load balancer through multicast packets, but we will use a static address on each back-end server.

Copy the other three .so files to the default Apache modules directory (/etc/httpd/modules/ for CentOS). Before loading the new modules in Apache you have to remove the default proxy balancer module (mod_proxy_balancer.so) because it is not compatible with mod_cluster.

Edit the Apache configuration file (/etc/httpd/conf/httpd.conf) and remove the line

LoadModule proxy_balancer_module modules/mod_proxy_balancer.so

Create a new configuration file and give it a name such as **/etc/httpd/conf.d/mod_cluster.conf**. Use it to load mod_cluster's

modules:

```
LoadModule slotmem_module modules/mod_slotmem.so  LoadModule mana
ger_module modules/mod_manager.so  LoadModule proxy_cluster_modul
e modules/mod_proxy_cluster.so
```

In the same file add the rest of the settings you'll need for mod_cluster something like:

And for permissions and Virtualhost section

```
Listen 192.168.180.150:9999


  Order
deny,allow
    Allow from all 192.168
        ManagerBalancerName mymodcluster      EnableMCPMReceive
    ProxyPass / balancer://mymodcluster/
```

The above directives create a new virtual host listening on port 9999 on the Apache server you want to use for load balancing, on which the load balancer will receive information from the back-end application servers. In this example, the virtual host is listening on IP address 192.168.204.203, and for security reasons it allows connections only from the 192.168.0.0/16 network.
The directive ManagerBalancerName defines the name of the cluster – mymodcluster in this example. The directive EnableMCPMReceive allows the back-end servers to send updates to the load balancer. The standard ProxyPass and ProxyPassReverse directives instruct Apache to proxy all requests to the mymodcluster balancer.

That's all you need for a minimal configuration of mod_cluster on the Apache load balancer. At next server restart Apache will automatically load the file mod_cluster.conf from the /etc/httpd/conf.d directory. To learn about more options that might be useful in specific scenarios, check mod cluster's documentation.

While you're changing Apache configuration, you should probably set the log level in Apache to debug when you're getting started with mod_cluster, so that you can trace the communication between the front- and the back-end servers and troubleshoot problems more easily. To do so, edit Apache's configuration file and add the line LogLevel debug, then restart Apache.

2. How to set up Tomcat appserver for mod_cluster ?

Mod_cluster **works with** ***Tomcat version 6, 7 and 8*****, to set up the Tomcat back ends you have to deploy a few JAR files and make a change in Tomcat's server.xml configuration file.**
The necessary JAR files extend Tomcat's default functionality so that it can communicate with the proxy load balancer. You can download the JAR file archive by clicking on "Java bundles" on the mod cluster download page. It will be saved under the name mod_cluster-parent-1.2.6.Final-bin.tar.gz.

Create a new directory such as `/root/java_bundles` **and extract the files from** `mod_cluster-parent-1.2.6.Final-bin.tar.gz` ***there*****. Inside the directory** `/root/java_bundlesJBossWeb-Tomcat/lib/*.jar` **you will find all the necessary JAR files for Tomcat, including two Tomcat version-specific JAR files –** `mod_cluster-container-tomcat6-1.2.6.Final.jar` **for Tomcat 6 and mod_cluster-container-tomcat7-1.2.6.Final.jar for Tomcat 7. Delete the one that does not correspond to your Tomcat version.**

Copy all the files from /root/java_bundlesJBossWeb-Tomcat/lib/ to your Tomcat lib directory – thus if you have installed Tomcat in

/srv/tomcat

run the command:

```
cp -rpf /root/java_bundles/JBossWeb-Tomcat/lib/* /srv/tomcat/lib/.
```

Then edit your Tomcat's *server.xml* file

/srv/tomcat/conf/server.xml.

After the default listeners add the following line:

This instructs Tomcat to send its mod_cluster-related information to IP 192.168.180.150 on TCP port 9999, which is what we set up as Apache's dedicated vhost for mod_cluster.
While that's enough for a basic mod_cluster setup, you should also configure a unique, intuitive JVM route value on each Tomcat instance so that you can easily differentiate the nodes later. To do so, edit the server.xml file and extend the Engine property to contain a **jvmRoute**, like this:

.

Assign a different value, such as node2, to each Tomcat instance. Then restart Tomcat so that these settings take effect.

To confirm that everything is working as expected and that the Tomcat instance connects to the load balancer, grep Tomcat's log for the string "modcluster" (case-insensitive). You should see output similar to:

```
Mar 29, 2016 10:05:00 AM org.jboss.modcluster.ModClusterService i
nit  INFO: MODCLUSTER000001: Initializing mod_cluster ${project.v
ersion}  Mar 29, 2016 10:05:17 AM org.jboss.modcluster.ModCluster
Service connectionEstablished  INFO: MODCLUSTER000012: Catalina c
onnector will use /192.168.180.150
```

This shows that mod_cluster has been successfully initialized and that it will use the connector for 192.168.204.204, the configured IP address for the main listener.
Also check Apache's **error log**. You should see confirmation about the properly working back-end server:

```
[Tue Mar 29 10:05:00 2013] [debug] proxy_util.c(2026): proxy: ajp
: has acquired connection for (192.168.204.204)  [Tue Mar 29 10:0
5:
00 2
013] [de
bug] proxy_util.
c(2082): proxy: connecting ajp:/
/192.168.180.150:8009/ to
192.168.180.150:8009  [Tue Mar 29 10:05:00 2013] [debug] proxy_ut
il.c(2209): proxy: connected /
to
192.168.180.150:8009  [Tue Mar 29 10:05:00 2013] [debug] mod_prox
y_cluster.c(1366): proxy_cluster_try_pingpong: connected to backe
nd  [Tue Mar 29 10:05:00 2013] [debug] mod_proxy_cluster.c(1089):
 ajp_cping_cpong: Done  [Tue Mar 29 10:05:00 2013] [debug] proxy_
util.c(204
4): proxy: ajp: has released connection for (192.168.180.150)
```

This Apache error log shows that an AJP connection with

192.168.204.204 was successfully established and confirms the working state of the node, then shows that the load balancer closed the connection after the successful attempt.

You can start testing by opening in a browser the example servlet SessionExample, which is available in a default installation of Tomcat.
Access this servlet through a browser at the URL http://*balancer_address*/examples/servlets/servlet/SessionExample. In your browser you should see first a session ID that contains the name of the back-end node that is serving your request – for instance, **Session ID: 5D90CB3C0AA05CB5FE13121E4B23E670.node2**.

Next, through the servlet's web form, create different session attributes. If you have a properly working load balancer with sticky sessions you should always (that is, until your current browser session expires) access the same node, with the previously created session attributes still available.

To test further to confirm load balancing is in place, at the same time open the same servlet from another browser. You should be redirected to another back-end server where you can conduct a similar session test.
As you can see, **mod_cluster** is easy to use and configure. *Give it a try to address sporadic single-back-end overloads that cause overall application slowdowns*.