RogueWave SOFTWARE

Accelerating Great Code

INCREASE SYSTEM AVAILABILITY BY LEVERAGING APACHE TOMCAT CLUSTERING

Open source is the dominant force in software development today, with over 80 percent of developers now using open source in their software[1]. With an increase in use of open source software (OSS), support and knowledge of these packages becomes critical to companies. Apache Tomcat is one of the most popular open source software implementations in the industry, and one of the most requested support packages to Rogue Wave OSS support experts. Support requests related to Apache Tomcat cover many aspects of the software, but a popular topic is always high-reliability and clustering.

Running Apache Tomcat as an application container for revenue generating applications is a decision many companies make every day. Unfortunately, this is where the decision making stops. The design should continue to include a system configuration that will protect these cash generating applications even in the event of catastrophic system failure. This is where Tomcat clustering plays a role. By implementing a solid clustering design you protect your company from systems failure.

This paper will walk through what is clustering and why you should cluster your Tomcat application servers. It will look at the multiple options for clustering setups, and help you identify which one is the best for your IT infrastructure. Finally, you can review some detailed example configurations and common issues when clustering with Apache Tomcat.

[1] Results from Forrester Research – Forrsights Developer Survey Q2-2015.

WHAT IS CLUSTERING?

Clustering, when referring to information technology systems, is two or more independent interconnected systems (nodes), interlinked to provide reliability. Reliability can come in the form of high-availability, improved scalability, improved application availability, and ease of maintenance.

Independent interconnected systems sounds complicated, although it's not. In the case of this paper we are referring to Tomcat systems. An instance of Tomcat is an independent system. Clustering instances of Tomcat makes them interconnected. Tomcat instances in a Tomcat cluster are often referred to as a node. Individual components in any network configuration can normally be referred to as nodes, but for the duration of this paper, we are referring to nodes as Tomcat instances.

A Tomcat cluster is a group of Tomcat instances that are connected. There are different ways that they can be connected. The Tomcat instances can be running on the same physical device, same virtual device, or disparate systems. There are many different options when it comes to clustering Tomcat, and we will discuss these in detail further on in this paper.

WHY YOU SHOULD CLUSTER

Clustering can solve different problems. For instance, you have a web application, serving approximately five thousand concurrent requests, running on your server. Under this load, your single server is maxed out. New users are receiving 404 errors. Supporting larger numbers of concurrent requests is one of the advantages of high-availability clustering. The goal of high-availability clustering is 99.999 percent ("five nines").

Another example of a problem clustering is the solution for failover. If your business is running a web application that earns income for your business and this web application is running in a non-clustered environment, you are at risk. If your application is on a Tomcat server that is not clustered and the Tomcat server fails, that source of revenue stops generating money every second the system is down. Setting up a simple Tomcat cluster containing two instances, this issue is preventable. In a properly configured cluster, all requests to the failed server will be directed to the remaining working instance. This will preserve your revenue stream even if there is performance degradation from losing 50 percent of the nodes in the cluster.

These are just some examples of why it pays off to cluster Tomcat, or at least research a little more. In addition to these examples above, Tomcat improves your systems availability. High-availability is a goal that many companies seek to improve the appearance and availability of their services.

A normal system's yearly average uptime is called it's availability. High-availability is a pre-arranged, contracted level of performance that will be maintained during the contract length. Granted, that is not very easy to understand. An example of high-availability could be: your web server is guaranteed to be available "five nines." This means that in a given year the server will have a maximum of 5.26 minutes of unscheduled downtime a year.

To achieve high-availability you need to implement geographic separation. Geographic separation, in regards to our server configuration, is installing nodes of the cluster in geographically different locations. This provides safety against regional power outages and other locational risks like storms and floods.

DETERMINING THE BEST CLUSTERING SETUP FOR YOUR ARCHITECTURE

Everyone wants to build a reliable, stable, and available application container platform. But, in order to do so you need to determine which clustering configuration fits you and your business the best.

In determining your configuration you must evaluate the resources at hand. This section will discuss possible options for your resources, without actually taking your resources into consideration. The next section will make suggestions as to which configurations your company may leverage depending on the resources available.

Vertical, horizontal, or hybrid cluster

A vertical cluster expands vertically. A horizontal cluster expands, you guessed it, horizontally. What does this mean? A vertically expanding cluster has limited horizontal layout. Horizontal layout would be multiple systems/resources.

A vertical cluster is on a single machine — a machine can be many things, including a physical device or a virtual host. As need increases, Tomcat instances are spawned on the same machine, using configuration tweaks that allow multiple instances to run on the same system.

A horizontal cluster contains Tomcat instances running on separate machines. If demand for processing increases and you had a pure, horizontal cluster configuration, the network technician (or you) would install a new machine, virtual or physical, and on that machine is a new Tomcat instance.

Real life is often very different from dictate. Companies rarely have a pure horizontal or vertical cluster configuration. Most systems are hybrids. A hybrid cluster is a mixture of vertical and horizontal clustering to facilitate a specific need and/or to match the hardware provided.

Homogeneous or heterogeneous cluster

Is your setup going to be for multiple applications, or just a few, or maybe just one? Do you have applications that require specific hardware? This determines whether or not you decide to use a heterogeneous or homogeneous setup. While this section defines your options, the next section will help you decide which option suits your needs.

A homogeneous setup is very common. Companies will often duplicate their Tomcat environment, launching servers on many devices with a simple copy of the Tomcat directory. A Tomcat cluster that has the same web applications deployed on all nodes is considered homogeneous.

Homogeneous setups can be hard to keep truly identical. Sometimes, especially after node failure and replacement, it can be hard to synchronize the Tomcat instances. The best way to do this is to create an image of the Tomcat setup from a node designated as the primary node. As long as this image stays up-to-date you can distribute it over as many Tomcat setups as you prefer.

Load balancing

Load balancing happens outside of the Tomcat cluster. The broad scope of load balancing will not be touched in this document. We are concerned with Apache Httpd server and the built-in load balancing/gateway features, as this is free and available, and because of this, it is a common solution in many enterprise systems.

To use Apache Httpd as a load balancer we will configure it as a gateway. Once it is aware of its nodes, it is able to balance traffic across these nodes. Further on in the paper, we will show an example configuration, using `mod_proxy_ajp`, of an Httpd gateway with "Round Robin" load balancing.

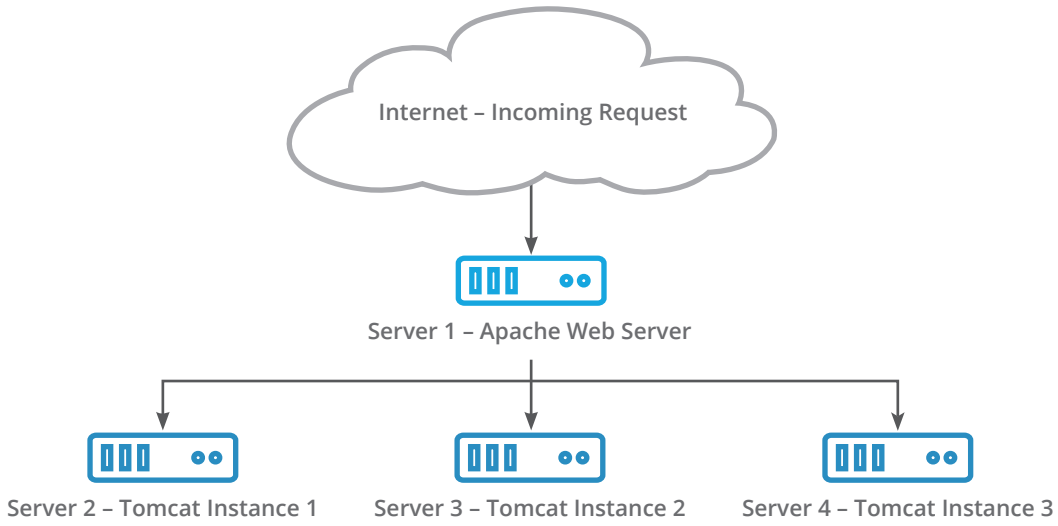
Another common enterprise configuration option for load balancing is the hardware load balancer. A hardware load balancer (HLB) performs the same tasks as a software balancer (like the one in the Apache Httpd server). The main difference between a software balancer and a hardware balance (besides price), is resources. A HLB has dedicated hardware resources (RAM), processor, network adapters, etc. This allows hardware balancers to perform at a much more efficient rate, while providing more features. This is also an infinitely more expensive method, as you can find many free open source load balancing solutions.

DETERMINING THE BEST FIT FOR YOUR ORGANIZATION AND RESOURCES

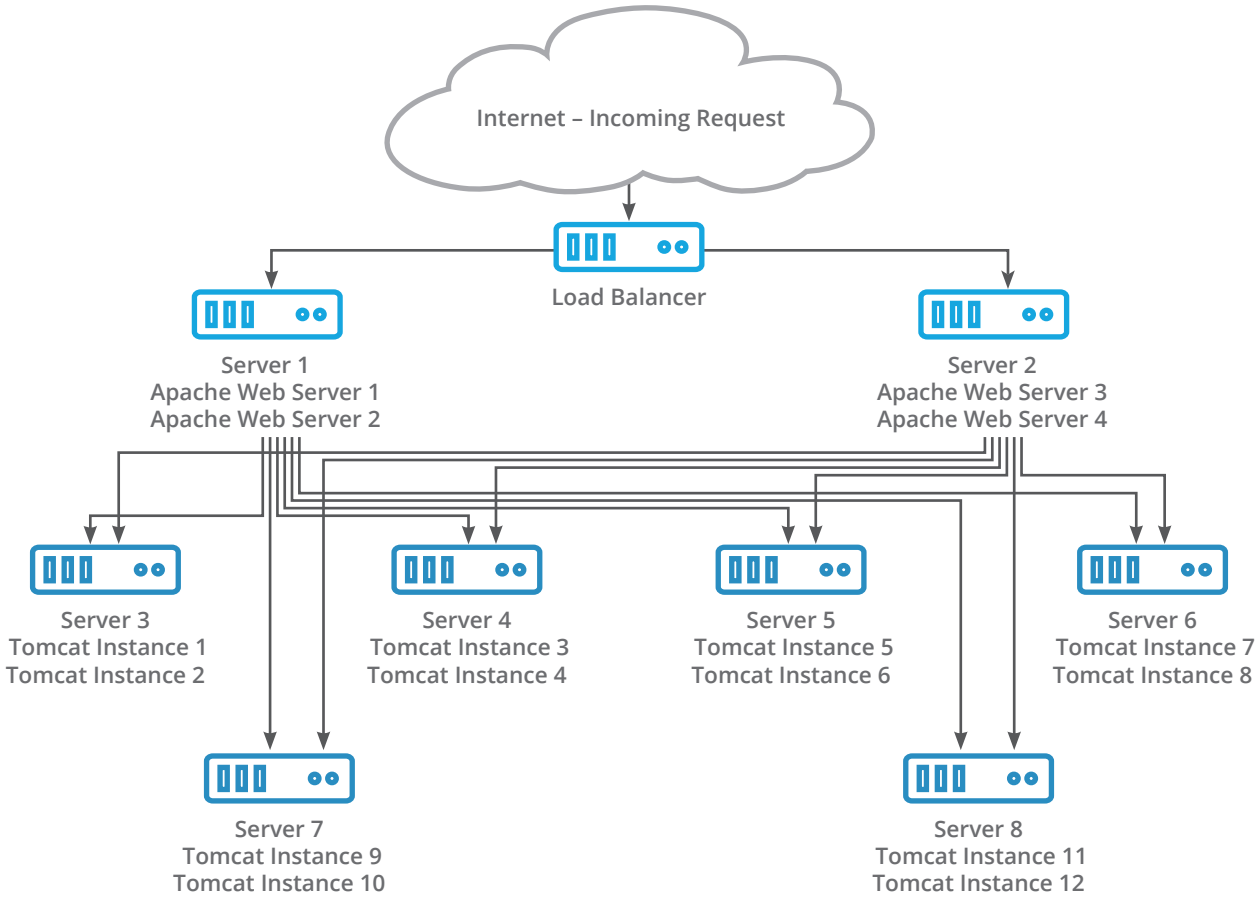
There are many factors in determining your cluster configuration. When choosing how to configure your cluster it's critical to examine the resources available before making a decision.

Scalability

How do you choose your scalability options? This relies heavily on the availability of hardware resources. For instance, you have 4 low-end servers, meaning they have one processor with 1-4 gigabytes of RAM. This would be an ideal situation for a horizontal cluster. Each member of the cluster would be able to run one instance of Tomcat efficiently. One of the servers could be used as a balancer running Apache Httpd server. Here is a drawing of the architecture.



If your situation was a bit different, and you had better servers, you could consider a hybrid cluster. If there are servers available with two or more processors and a large amount of ram (8 gigabytes or more) this would be ideal for multiple Tomcat instances. In this configuration you can setup a hybrid cluster by running multiple instances of Tomcat on multiple machines, and multiple instances of Apache Httpd to handle the load of load balancing. This configuration could look something like this:

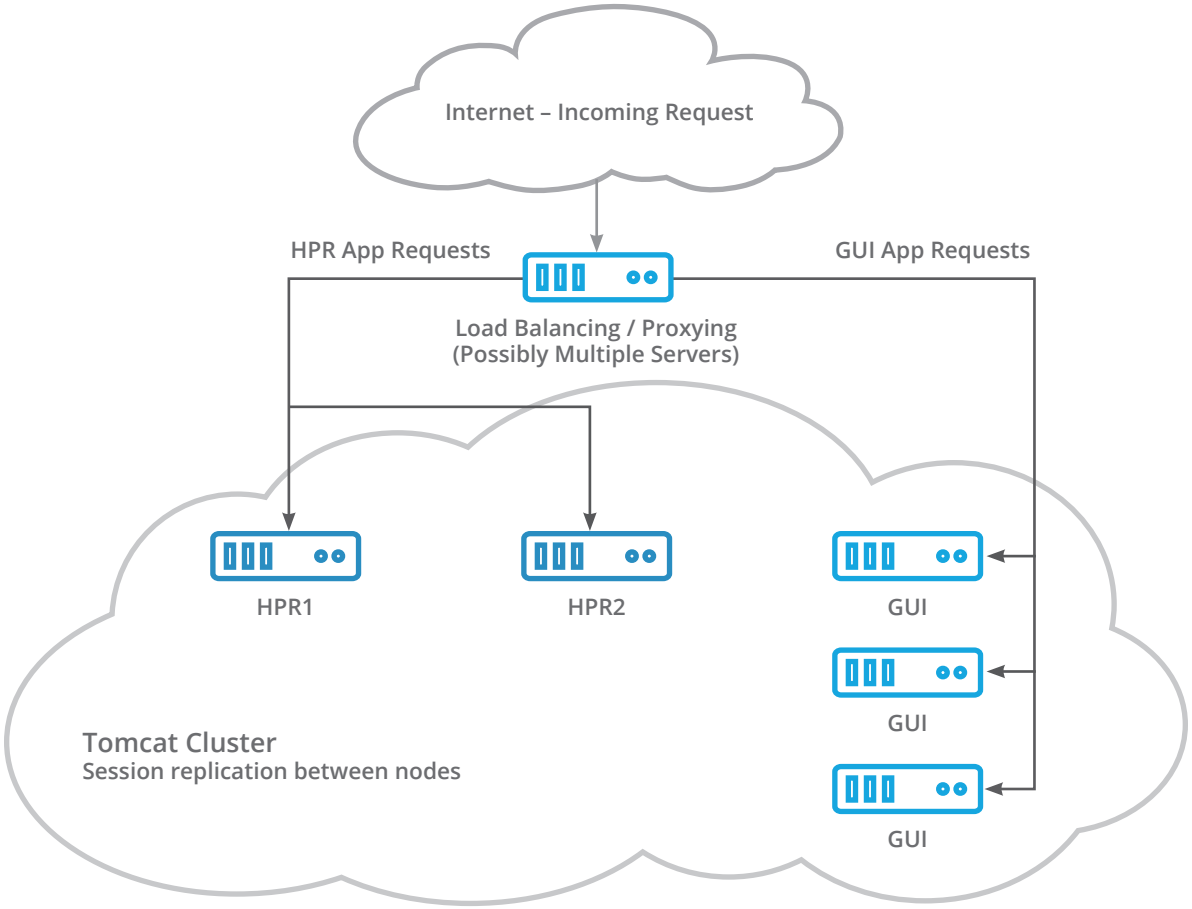


Heterogeneous or homogeneous Tomcat configuration

Determining heterogeneous or homogeneous setup can be simple in some situations. The easiest situation is one with a single web application. If there is only one application to deploy, you deploy it to all members in the cluster. This is a very straightforward homogeneous configuration.

Unfortunately, most companies do not have one single web application; however, this situation is not overly complicated unless the company has an extremely large number of applications.

The division of your applications over Tomcat nodes will be your choice. The Tomcat configuration of the nodes will be a little more complicated and we review this further in the examples section. If the company has an application that requires heavy processing and large amounts of RAM (HPR1,) you can setup this application on two nodes by itself. After this, take the remaining applications (GUI) and place them on two different nodes in the cluster. This will prevent the GUI application from being bogged down when HPR1 is consuming the CPU and RAM. This cluster might look like this:



There are many things to take into consideration when designing and building your cluster. If a large company is relying on you to provide a reliable, highly-available application implementation, then clustering and load balancing is the right choice. Regardless of if you are new to clustering, or an old hand, purchase a support contract. There are companies that will provide open source software support for your Tomcat and Apache Httpd configuration. This will allow you to offer your customers an extremely reliable, available service while at the same time providing someone to turn to if you run into problems.

ENTERPRISE APACHE TOMCAT 8 CLUSTERING — CONFIGURATIONS AND COMMON ISSUES

This is not a complete step-by-step tutorial on cluster creation, but we will provide you with the tools you can use to implement a cluster rapidly and effectively. Whether you have created many clusters in the past or this is your first attempt, we hope that you will be able to learn something, whether it be basic or advanced, from the information discussed in this paper. To limit the liability of your attempt at creating a cluster, you can set up a machine, virtual or physical, just for this task.

Note – You can run multiple Tomcat instances on a single virtual/physical machine by tweaking just a few settings, mainly port numbers so the instances don't interfere with each other. The configuration of these Tomcat instances is well outside the scope of this document, although it is not difficult to accomplish. Settings for the connectors in your server configuration files can be found at the Apache Tomcat 8 website. Below are two server configurations that you can use to run a simple cluster, just start with two instances of Tomcat 8, and replace the corresponding server.xml file with the .xml information provided below.

Server.xml 1

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="50005" shutdown="SHUTDOWN">
 <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
 <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
 <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
 <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
 <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

 <GlobalNamingResources>
   <Resource name="UserDatabase" auth="Container"
             type="org.apache.catalina.UserDatabase"
             description="User database that can be updated and saved"
             factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
             pathname="conf/tomcat-users.xml" />
 </GlobalNamingResources>

 <Service name="Catalina">
   <Connector port="51111" protocol="HTTP/1.1"
              connectionTimeout="20000"
              redirectPort="51113" />

   <Engine name="Catalina" defaultHost="localhost">
     <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
     <Realm className="org.apache.catalina.realm.LockOutRealm">
          <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
                 resourceName="UserDatabase"/>
     </Realm>

     <Host name="localhost"  appBase="webapps"
           unpackWARs="true" autoDeploy="true">

          <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
                 prefix="localhost_access_log" suffix=".txt"
                 pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />

     </Host>
   </Engine>
 </Service>
</Server>
```

Server.xml 2

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="50006" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

  <GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database that can be updated and saved"
              factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>

  <Service name="Catalina">
    <Connector port="51112" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="51114 " />

    <Engine name="Catalina" defaultHost="localhost">
      <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
      <Realm className="org.apache.catalina.realm.LockOutRealm">
          <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
                resourceName="UserDatabase"/>
      </Realm>

      <Host name="localhost"  appBase="webapps"
            unpackWARs="true" autoDeploy="true">

          <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
               prefix="localhost_access_log" suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />

      </Host>
    </Engine>
  </Service>
</Server>
```

Configurations

Create a Tomcat cluster

Tomcat clustering is very simple to setup. However, if you wish to leverage clustering in your enterprise environment the default configuration is not going to be the best route for you. To turn on clustering in your Tomcat server all you have to do is add one line of code to your server.xml.

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
```

Adding this line to your configuration enables clustering with all of the default settings. This would be great if you were not in an enterprise setting.

You have created a clustered Tomcat instance, but you only have one instance, so it is not a very big cluster. Before we create the next instance we should install the application we want to test on this cluster.

Make your web application distributable

With your cluster running, placing a normal application on one server will not trigger propagation to other servers. The idea behind propagation is: an application is placed on one node in the cluster, it is migrated (copied) automatically to other nodes in the cluster. To achieve this we add the following code to the web.xml:

```
<distributable/>
```

This tells Tomcat that this application is designed to run on multiple nodes in this cluster.

Set up session replication

The default session replication mode is "All to All," meaning any session data created on a server will be duplicated to all other servers in the cluster. If your application creates session data for a user, and you have a heterogeneous cluster, the session data will still be replicated across the other nodes. A heterogeneous configuration is one that does not have all of the same applications on every node. Therefore, if application A stores session data for a user, and application A is running on server A, but not server B, session data will replicate to server B, even though there is no use for it there.

Configure multicast setup

The cluster is discovered and maintained via multicast heartbeats. The server will be set up with a default multicast IP address of 228.0.0.4 and a multicast port of 45564. This means that any other nodes that are using the same multicast address and port will see this cluster/node. It is important to ensure your network supports multicast. This is commonly blocked for security reasons.

Additional considerations

The Manager object

After creating the cluster object and making your web applications distributable, we need to move on to configuring other settings.

The Manager object controls session replication.

```
<Manager
 className="org.apache.catalina.ha.session.DeltaManager".../>
```

The DeltaManager replicates all changed session data to all nodes of the cluster. The BackupManager backs up session data to a specific backup node. For large clusters the BackupManager is the option to go with, for smaller clusters it is common to just use the default DeltaManager.

In Tomcat 5, you couldn't choose the specific session manager for your application. In Tomcat 8, you can define a manager in the cluster configuration, as you could in earlier versions, but you can also define a manager in a web application's context.

Defining the Manager in your clustering configuration provides a default setting for applications that do not provide their own Manager configuration. For instance, the following code will set all applications in your cluster to use the BackupManager for session replication.

```
<Manager
  className="org.apache.catalina.ha.session.BackupManager".../>
```

Channel send options

After setting the Manager, you might need to apply a non-default channel send options value. Channel send options is a setting specified on the cluster object. For example:

```
<Cluster
 className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
                channelSendOptions="6">
```

Channel send options controls how messages are sent between cluster nodes. Are these message sent synchronously, or, in layman's terms, does the thread that sends the message have to wait until the message has sent before continuing to work, in turn, potentially making the users request wait on this message to be sent? Sending the messages asynchronously is when the thread generates and sends the message, but does not stop and wait for this to happen. It does this by spawning a worker thread.

As you can see, this is just one aspect of the channel send options, and it is a lot of information. To go over channel send options in detail will require a whole default channel send mode is asynchronous.

CONCLUSION

This paper barely scratches the surface of clustering. What we have provided is a starting point for your cluster. With the information provided here you can start a cluster containing two or one thousand nodes, it is just a matter of determining your company's needs.

The cluster configuration in Tomcat can be simple. As demonstrated here we were able to configure a cluster in a small amount of time. Problems like nodes not joining cluster, session information being lost, random node crashes, and configuration issues are normally resolved with little effort. Tomcat clustering is a powerful tool that can provide the high availability, reliability, and dependability that your company requires and all of it can be setup with little effort.