Jabber Server Farming How−To

Ryan Eatmon

Jabber.org

reatmon@jabber.org

2002−06−06

Revision History

Revision 0.1	2002−06−06	Revised by: rwe
First draft.		

Provide Jabber admins a look into the present and future of Jabber Server Farming.

Table of Contents

Chapter 1. About this How−To

1.1. Purpose / Scope

This document was started on June 06, 2002 by Ryan Eatmon to explain how to set up a Jabber Server Farm using the Jabber.org Jabber Server (http://jabberd.jabberstudio.org).

This document is broken into two main sections. What is, and what will be. As of the writing of this document the Jabber Server does not have all of the pieces that are required to do full farming. There are steps that can be taken to get partial farming, and those will be covered. There are also steps that we are going to take, and those are covered to so that others can contribute and comment.

1.2. About Jabber

Jabber is an Open Source Instant Messaging System that uses XML as its base protcol. I could spend an entire How−To describing Jabber, so I am just going to point you to the Jabber.org web site. (http://www.jabber.org).

1.3. Feedback

Comments on this How−To may be directed to the author (`<reatmon@jabber.org>`).

1.4. Copyrights and Trademarks

1.5. Acknowledgments and Thanks

Thanks to everyone who gave me hints and tips on tuning Linux and structuring of the farm. This includes Thomas Muldowney, Jeremie Miller, and other members of the `<jabberd@jabberstudio.org>` list.

Chapter 2. Introduction to Jabberd and Farming

2.1. Jabberd – The Jabber.org Jabber Server

The Jabberd server is a reference implementation for the Jabber protocol as defined by the Jabber Software Foundation. It is written in C and is covered by both the Jabber Open Source License(JOSL) and the GNU Public License(GPL). The Jabberd project has its own documentation. (http://jabberd.jabberstudio.org/howto.html).

2.2. Farming

This is an interesting topic. When you think of a server you tend to think of a single machine running a single program. Farming is the idea of multiple machines running multiple programs that all act together to appear to be one machine/program.

Why farming? Two reasons with equal importance. Scalability and Reliability.

Scalability is the idea of allowing something to handle many many more connections/transactions at the same time. A typcial web server can handle say 1000 connections a second, but 10 web servers acting together can provide 10,000 connections per second.

Reliability is the idea that even if a piece of the system goes down, the system does not. There is no single point of failure, and there is built in redundancy to ensure that something is there to pick up the slack if something goes down. In the web server example, if one web server goes down you could not send any connections to it until it is back up, thus ensuring that your web site is always up.

Both of these ideas are crucial to a successful Farming strategy. Each wants to push the design into a different direction, but they can be reconciled into a single design that can scale and is reliable at the same time.

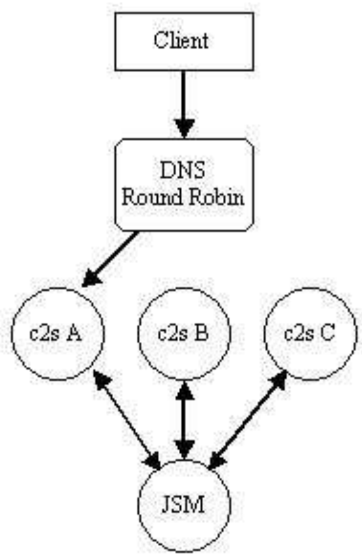
Chapter 3. Implementing the Farm

3.1. Client−to−Server Farming w/ Round Robin DNS

3.1.1. The Dream

The first thing that we can do is to split out the c2s from the server and make it a seperate process/component. Once that is done, we can replicate the c2s processes over multiple machines. Each one would connect back to the main server running the JSM and everything should work just fine.

Figure 3−1. c2s Farming Diagram



Pros:

- Easy to setup right now.

Cons:

- Round Robin will not provide true load balancing since there is no mechanism in place to check how many connections a server has when it forwards a new connection to it the next time it comes up in the Round Robin.

3.1.2. The Reality

This is doable right now. The 1.5 version of jabberd has sought to break all of the pieces out into seperate processes. This was not done for farming specifically, but we will not complain since the jadc2s component can handle upwards of 10k users. (The 1.4.x series c2s could only handle ~1024).

Currently there is a forked version of jadc2s that works with the 1.4.2 server. It is located in the jabberd14 CVS repository on JabberStudio. The following example is running with two jadc2s boxes, and one central jabberd box. To set this up do the following:

1. Get all of the source code for jabberd14. Build the server, configure/make.
2. Get the source for xdb_sql from IDEALX , build it, and setup the jabber.xml to use it.

   This is a very important step. xdb_file, the default xdb that comes with jabberd, is limited to open file descriptors too. You can play the same shell games that we are going to play with jadc2s later, but if you want a server that can handle millions or users, then you need something other than xdb_file. Enter xdb_sql, which only uses one file descriptor to connect to the mysql server.

   For more information on how to configure xdb_sql, please see the README that is distributed in the release.
3. Build jadc2s and distribute the binaries to the boxes where they will run.
4. Setup the main jabberd to accept the jadc2s component connections. In the jabber.xml config file, add the following XML:

```
<service id="jadc2s-1">
    <accept>
        <ip/>
        <port>5111</port>
        <secret>secret</secret>
    </accept>
</service>

<service id="jadc2s-2">
    <accept>
        <ip/>
        <port>5112</port>
        <secret>secret</secret>
    </accept>
</service>
```

   Now you can run the main jabberd and get it listening for the jadc2s to connect to it.
5. Configure the jadc2s.xml on each box to connect to the SM, where to listen, etc...

```
<!--  session manager configuration                                    -->
<sm>
    <!--  host and port of the SM                                      -->
    <host>localhost</host>
    <port>5111</port>
    <!--  shared secret, for authenticating us                         -->
    <secret>secret</secret>

    <!--  our ID, for authenticating us to the sm                      -->
    <id>jadc2s</id>

    <!--  how many times to try to connect to the sm (default: 5)      -->
    <retries>5</retries>
</sm>

<!--  local network settings                                           -->
<local>
    <!--  who we identify ourselves as. This should correspond to the  -->
    <!--  ID (host) that the session manager thinks it is.             -->
    <id>localhost</id>

    <!--  IP address to bind to (default: 0.0.0.0)                     -->
    <!-- <ip>0.0.0.0</ip> -->
```

```
    <!--  port to bind to (default: 5222)                              -->
    <port>5222</port>

    <!--  SSL configuration                                            -->
    <!--  Specify the port to listen on, and the key file to use for   -->
    <!--  the certificate.                                             -->
    <!--  <port/>      (default: 5223)                                 -->
    <!--  <pemfile/>  (default: ./server.pem)                          -->
    <!--
    <ssl>
        <port>5223</port>
        <pemfile>./server.pem</pemfile>
    </ssl>
    -->
</local>
```

For more information on how to configure jadc2s, please see the README in the jadc2s source directory.

6. Open a shell where you can change file system parameters (root usually) and execute the following command:

```
bash$ echo "24000" > /proc/sys/fs/file−max
```

   This bumps the upper bound on the number of allowed file descriptors that can be open at one time.
7. Set the limit for the shell you are in to use more than the default 1024 file descriptors.

```
bash$ ulimit −n 11000
```

8. Tell jadc2s how many file descriptors it is allowed to use:

```
<!--  maximum number of file descriptors. Should be a prime        -->
<!--  number. At least four will be used by jadc2s itself,         -->
<!--  usually around six or seven (default: 1023)                  -->
<!--  For a list of prime numbers:                                 -->
<!--    http://www.utm.edu/research/primes/lists/small/10000.txt   -->
<max_fds>10007</max_fds>
```

   It is important that the number you pick is a prime number. To make it easy to find the prime you want you can visit this page http://www.utm.edu/research/primes/lists/small/10000.txt.
9. All that's left is to run the server, and the jadc2s processes. Everything should work fine. If it doesn't, then PLEASE tell me at <reatmon@jabber.org> so that I can fix this document.

3.2. Connection Redirection

3.2.1. The Dream

Now that we have the c2s processes distributed across machines, we need to get them to work together. We need two things, the ability to communicate how many connections a c2s has to the others, and the ability to know who the others are.

[22:21] Jer: well, the "protocol" could just be an auth error of 3xx for redirect w/ the ip/port to connect to

Maybe we need to talk about the router layer sooner...

Pros:

- Load balances nicely.

Cons:

- Requires client support for connection redirection. This is not hard to do, it is just not part of any client right now.

3.3. Router Layer

3.3.1. The Dream

Pros:

-

Cons:

-

3.4. Moving JSM State data into a DB

3.4.1. The Dream

Pros:

-

Cons:

-

3.5. Multiple JSMs

3.5.1. The Dream

Pros:

-

Cons:

-

3.6. JSM Communications

3.6.1. The Dream

Pros:

-

Cons:

-