JBoss Performance Tuning and Clustering – Quick Guide

Author: Renato Pioker

April, 15th 2015

SUMMARY

Contents

JBoss Performance Tuning and Clustering

The goal of this guide is to set up JBoss with optimal performance settings, guiding the configuration to quick and easy steps. This guide will also explain how to set up JBoss clustering.

This guide has three sections:

- Section I – Performance Tuning
- Section II – Clustering
- Section III – General CA product considerations

1. Pools

A pool is a group of objects equivalent between each other. While using JBoss as the application server to host CA IAM framework, the most relevant pools are the database connection pools.

It is important that the connection pool be set up with a small initial number (for example, five connections) but with a high maximum limit, for example two hundred connections. The reason for the low initial number is to avoid a high memory and CPU usage in low activity hours, while the high maximum limit allows the server to process properly during peaks.

A crucial consideration – sometimes ignored – is to check the maximum connection limit for the database user, the database server and the database itself. The sum of all JBoss connection pools maximum limits must be equal or lower the maximum connection limit from the database side, otherwise JBoss will trigger connection errors in peak usage times.

Connection pool configuration is done in JBoss XML files, like the "objectstore-ds.xml" file located in jboss-4.2.3/server/default/deploy path.

1.1. Configuration Example

```
<min-pool-size>5</min-pool-size>

<max-pool-size>200</max-pool-size>

<idle-timeout-minutes>5</idle-timeout-minutes>

<blocking-timeout-millis>10000</blocking-timeout-millis>
```

- "min-pool-size" is the minimum size of the pool. It is also the initial connection count.
- "max-pool-size" is the maximum size of the pool.
- "idle-timeout-minutes" is the amount of time in minutes JBoss will keep a connection opened without use, before closing it. After this amount of time, the connection is closed.
- "blocking-timeout-millis" is the amount of time in milliseconds JBoss will allow an application to wait for a connection when all connections from the pool are in use. After that amount of time the Java Virtual Machine triggers a connection timeout exception to the application and the request is discarded.

2. Java Virtual Machine Memory

The first thing to consider when configuring the JVM parameters is that there is no single configuration model that suits all environments. Each situation is unique, so it must be analyzed carefully. There are, in the other hand, certain concepts that must be applied to guarantee the better use of the hardware and of JBoss.

The JVM memory is divided, basically, in three major areas:

a) Process memory – it is used by JVM binary and it's not used by the applications;
b) Heap space – memory space available to applications;
c) Permanent Generation – reserved space to load the bytecode of each class.

2.1. Heap Space

The Heap Space is divided in Young Generation (objects that were recently loaded) and Tenured Generation (objects that are in use, so they stay longer in memory). Young Generation is also divided in smaller areas: Eden (where the objects are loaded at first) and the Survival Spaces (where the objects move during their use until being transferred to Tenured Generation). The lifespan of an object is its runtime – how long this object is being used by the application.

2.1.1. Heap Space considerations

To set up the Heap Space, you must consider the following:

- Total amount of RAM on the server;
- Other services and applications installed and in use on the same server;
- Available amount of RAM after OS startup;
- RAM that will be used by JVM binary and by Permanent Generation.

To achieve better performance:

- The server must be dedicated to JBoss. This will allow a better resource allocation, as explained later in this guide;
- The amount of RAM to be set up for JBoss JVM Heap Space must not be larger than 4GB. With memory allocation larger than 4GB, the Garbage Collector become slow and loses efficiency. This way is better to have several JBoss instances with 4GB RAM each, than one JBoss instance with 16GB.
  Important note regarding Garbage Collector: while it is running, a pause is enforced to all active objects, to allow the GC to perform the cleanup. The larger is the Heap Space to be processed by GC, the slower it will run – and the longer the objects will remain paused, causing application slowdown.
- Avoid using configurations that use dynamic memory allocation, because it uses CPU power and can slow down the applications.

2.1.2. Configuring Heap Space

On Java command line used to start JBoss there are two parameters that define the initial and maximum sizes of the Heap Space:

- -Xms – initial size;
- -Xmx – maximum size.

The recommendation is to set the same value in both parameters (for example: -Xms2048m -Xmx2048m to set a Heap Space of 2GB). With this setting, the whole memory is allocated at JVM startup, avoiding the dynamic allocation and improving performance on JBoss execution.

According with the example above, the Heap Space was set with 2GB RAM. Considering a server with 4GB RAM:

A – B – C – D = Heap Space

4 – 1,7 – 0,1 – 0,2 = 2GB where:

- A = Total amount of RAM
- B = RAM used by OS right after startup, together with other server resources like antivirus, agents, etc.
- C = Memory used by the JVM binary;
- D = Permanent Generation configured memory (explained next).

2.2. Permanent Generation

On Java command line used to start JBoss there are two parameters that define the initial and maximum sizes of the Permanent Generation memory:

- -XX:PermSize – initial size;
- -XX:MaxPermSize – maximum size.

The same recommendation from Heap Space parameters are valid to Permanent Generation parameters: set them up using the same value on initial size and maximum size (for example: -XX:PermSize=256m and -XXMaxPermSize=256m).

2.3. Additional JVM Parameters

2.3.1. Garbage Collector

There are three types of GC that can be used:

- Serial: standard in non-server machines. Performs the cleanup using only one thread, keeping the applications paused while running. The parameter to use it is: -XX:+UseSerialGC. Good for machines with only one CPU and low RAM;
- Parallel: standard for servers, it executes the cleanup using one thread per CPU/core, keeping the applications paused while processing the cleanup. Configuration parameters: -XX:+UseParallelGC and -XX:ParallelGCThreads=*n*, where "n" is the number of threads (the standard is one thread per CPU). Efficient in servers with two or more processors and/or CPU cores;
- Concurrent: runs the cleanup without pausing the applications and can only be executed by servers with multiple processors. Slower than the parallel GC and less efficient in releasing memory. Configuration parameter: -XX:+UseConcMarkSweepGC.

2.3.2. Young Generation

It is possible to configure, as well as Heap Space, its subdivision Young Generation, as follows:

- -XX:NewRatio=*n*: sets the size of Young Generation using the formula 1/(*n*+1), increasing or reducing the proportion between Young and Tenured;
- -Xmn: sets the value in bytes to the size of Young (for example, -Xmn512m). This parameter does not affect the total size of Heap Space, but how much of Heap Space will be allocated to Young Generation.

2.4. JVM Monitoring

It is possible to monitor the JVM usage by using an application that comes with JVM – the JVisualVM (at bin directory on JDK installation, jvisualvm.exe).

To allow JVisualVM to work properly, JBoss cannot be executed as a service because, in this case, JVisualVM cannot connect to JVM process. To monitor JBoss with this utility, stop the JBoss service and start JBoss using the run script. This utility allow to monitor RAM and CPU usage, as well as other kinds of information to help analyzing performance issues.

3. Cluster Configuration

To configure JBoss cluster, edit the run script (on jboss/bin directory):

- Replace all entries "-c default" by "-c all";
- Remove the comment from the line below:
  ```
  set IDM_OPTS=%IDM_OPTS% -Djava.net.preferIPv4Stack=true
  ```
- At the end of the line "set ARGS=-b 0.0.0.0" add the parameters "-g clusterPartition -Djboss.messaging.ServerPeerID=*PeerID*" (where *PeerID* is an integer between 0 and 255, and must be unique for each JBoss node). The line will look like this, considering PeerID=0:
  ```
  set ARGS=-b 0.0.0.0 -g clusterPartition -Djboss.messaging.ServerPeerID=0
  ```

4. Testing the Cluster

After configuring all required JBoss nodes, follow the steps below:

a) Access the folder jboss\server\all\lib on the first node, using command line;
b) Execute the following command on the first node:
   ```
   java -cp jgroups.jar org.jgroups.tests.McastSenderTest -mcast_addr 224.10.10.10 -port 5555
   ```
c) Access the folder jboss\server\all\lib on the remaining nodes, using command line;
d) Execute the following command on the remaining nodes:
   ```
   java -cp jgroups.jar org.jgroups.tests.McastReceiverTest -mcast_addr 224.10.10.10 -port 5555
   ```
e) Send a message (any text) from the first server;
f) Confirm that in the first node there is a reply from each cluster node;
g) Confirm that the sent message is visible on all nodes;
h) Close the test application on all nodes;
i) Start JBoss using the run script;
j) Confirm that all nodes are reporting to the cluster. This information is available at JBoss server log.

Section III – General CA product considerations

Each application, especially CA products, have special recommendations and procedures that are specific to JBoss clustering. It is crucial to read and follow the product documentation guidelines to allow a successful cluster implementation.

For example, CA IdentityMinder has a specific chapter for JBoss cluster on its Implementation Guide: “Installation on a JBoss Cluster”.

It is important to remember that, in a cluster deployment, any changes that are made directly to file system (performance settings, data sources, resource bundles, layout, BLTH, LAH, etc.) have to be executed manually in all cluster nodes.