Scaling up JBoss application server

**By Priyank Dhadhal**

* Introduction
* Methodology
* Hardware Tuning
* JVM Tuning
* Application Server Tuning
* Miscellaneous Tuning
* Conclusion

Introduction

* Better system performance depends on: Better design & implementation .

We need to defined performance objectives for tuning. Since JBoss tuning involves the environment on which JBoss is running , the JVM settings and OS settings on which JBoss can produce best results.

Methodology (Performance Improvement Cycle)

Hardware & operating System Tuning

Minimum configuration

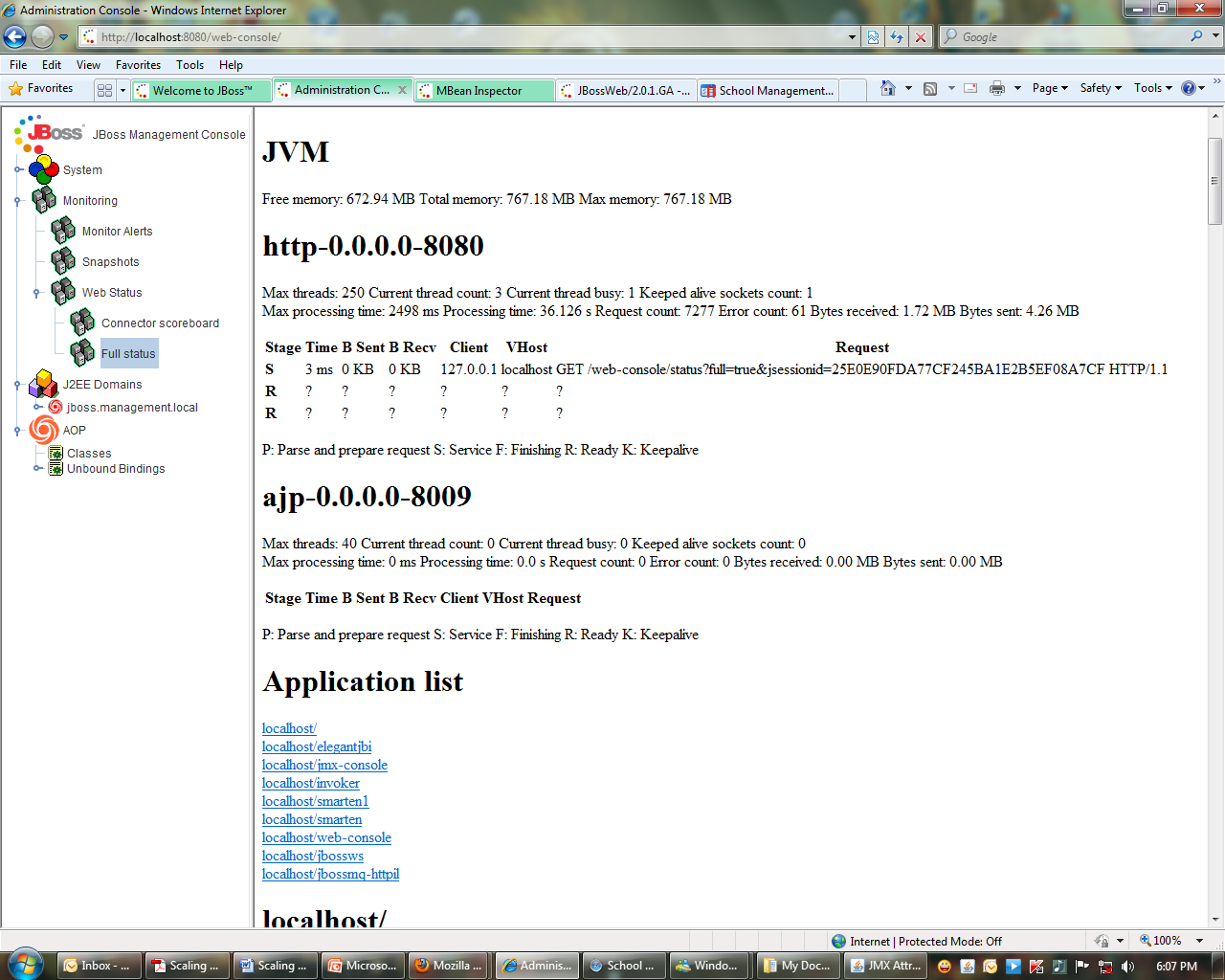
* 1000 Mhz Pentium IV
* 512 MB RAM minimum, 1 GB recommended
* 8 GB in the hard disk
* 100 MBPS Network connection
* Windows / linux 32 Bit operating system
* JAVA 32 bit

Recommended configuration

* Intel/AMD 64 bit processor
* 4 GB
* 20 GB in the hard disk space
* 1 GBPS Network connection
* Windows / linux 64 Bit operating system
* JAVA 64 bit

JVM & Performance Tuning: - Best Practices

Better performance in production servers is possible with proper configuration of JVM parameters, particularly those related to memory usage and garbage collection.



|  |
| --- |
| * **The JBoss Web console**      * **JMX console**   Managing run.bat file remotely |
|  |

* **Tune the garbage collector**

se the the command line option **-verbose:gc** causes information about the heap and garbage collection to be printed at each collection. For example, here is output from a large server application:

**It's demonstrated that an application that spends 10% of its time in garbage collection can lose 75% of its throughput when scaled out to 32 processors**

<http://java.sun.com/javase/technologies/hotspot/gc/gc_tuning_6.html>

**Turn off distributed gc**

The RMI system provides a reference counting distributed garbage collection algorithm. This system works by having the server keep track of which clients have requested access to remote objects running on the server. When a reference is made, the server marks the object as "dirty" and when a client drops the reference, it is marked as being "clean.". However this system is **quite expensive** and by default runs every minute.  
  
Set it to run every 30 minute at least  
-Dsun.rmi.dgc.client.gcInterval=1800000  
-Dsun.rmi.dgc.server.gcInterval=1800000

**Turn on parallel gc**

If you have **multiple proessors** you can do your garbage collection with multiple threads. By default the parallel collector runs a collection thread per processor, that is if you have an 8 processor box then you'll garbage collect your data with 8 threads. In order to turn on the parallel collector use the flag -XX:+UseParallelGC. You can also specify how many threads you want to dedicate to garbage collection using the flag -XX:ParallelGCThreads=8.

* **Don't choose an heap larger than 70% of your OS memory**

Choose a maximum heap size not more than 70% of the memory to avoid excessive page faults and thrashing.

* **Set -Xms and -Xmx to the same value**

**Configuring JVM Memory Settings**

The run.bat file in the bin folder contains JVM parameters including memory configuration. In the production server, ensure that these values are correctly set. For example, the following sets the minimum and maximum heap size as 1GB. It is better to keep them identical for performance reasons. The actual heap size setting will depend on your application requirements and also on the RAM size of the server machine.

set JAVA\_OPTS=%JAVA\_OPTS% -Xms1024m -Xmx1024m

**for X-86 ( 32 bit processor would support only up to 3 GB)**

**for X-64 ( 32 bit processor would support only up to xx GB) where xx is 70 % of your physical h/w ram .**

In the **JVM, PermGen** holds the **metadata** about classes that have been loaded/created. This information is garbage collected like the other parts of the heap, however there are rough edges that can prevent this from happening, class loaders in particular (which I won’t discuss here.) Generally, the amount of PermGen space needed is small in relation to the rest of the heap and default JVM values should work for you

Also ensure that adequate permgen space is set. Permgen space is the fixed memory required such as the code footprint. For large applications the default value of 64m may not be sufficient. Following sets the permgen space to 512MB.

set JAVA\_OPTS=%JAVA\_OPTS% -XX:PermSize=512m -XX:MaxPermSize=512m

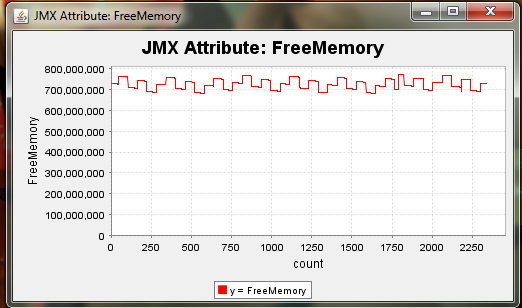
### Use server VM The server JVM is better suited to longer running applications. To enable it simply set the -server option on the command line.

### <http://java.sun.com/docs/hotspot/HotSpotFAQ.html#64bit_selection>

* **Monitor Memory alerts with JBoss to optimize your configuration**

<http://localhost:8080/web-console/>

Now Expand the System icon and then the JMX MBeans tree.  
jboss.system domain: expand it and look for the MBean *jboss.system:type=ServerInfo*.  
  
Beneath this Mbean you can find all the monitoring options available: you should be able to right click on in and get a few menu options: Now if we want to draw a graph of the attribute FreeMemory, simply right click on FreeMemory and you'll see a real-time graph of your Free Memory.



* **Remove unnecessary JBoss service & applications to optimize your configuration**

JBoss comes with a lot of services and your enterprise applications may not need all of them. Removing these unwanted services can boost application server performance. Following are some of the JBoss services you can remove if your application is not using them. Delete the files/folders given in brackets to remove these services completely.

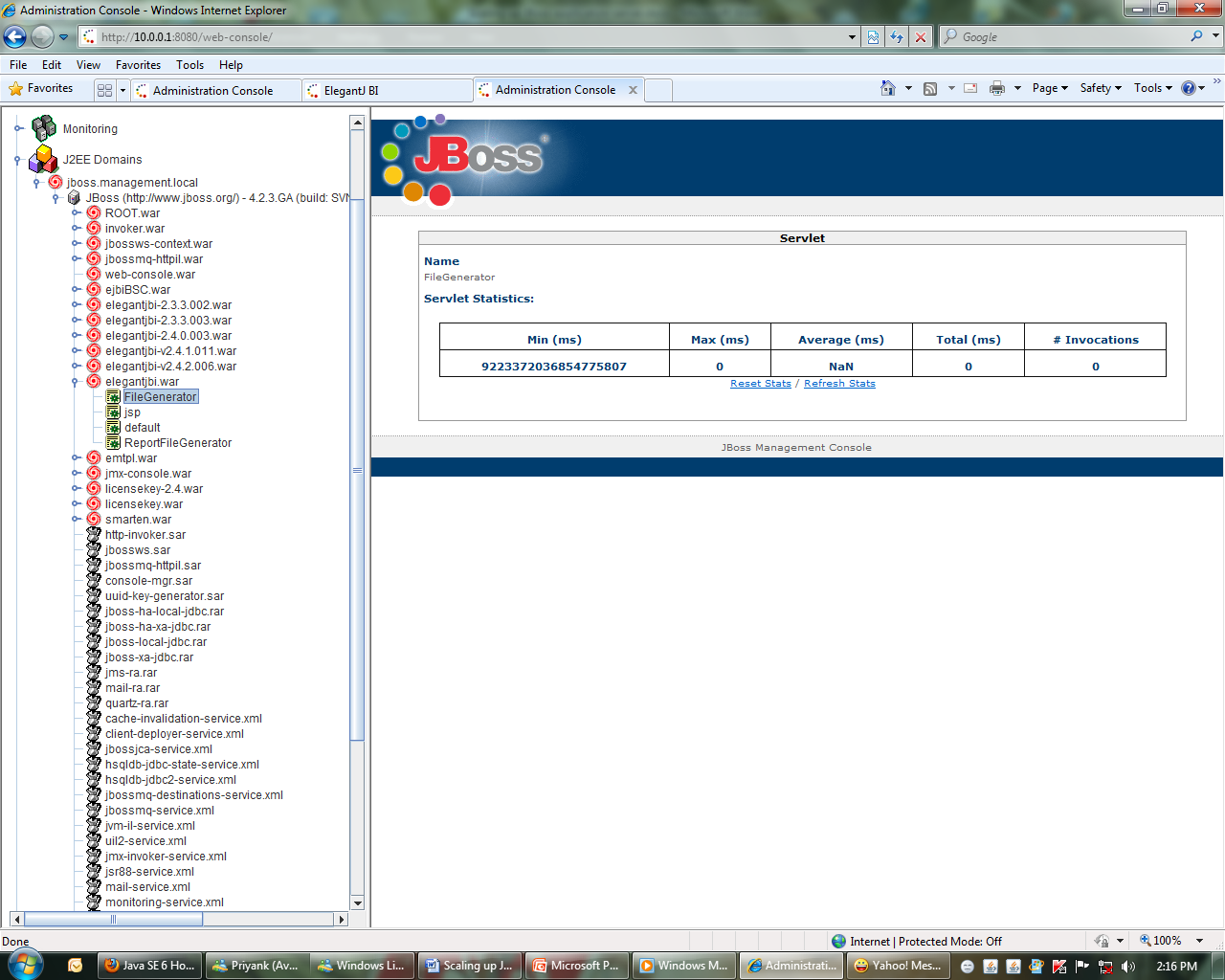
(a) Home page server- (deploy/ROOT.war)(b) JMX Console server – (deploy/jmx-console.war)(c) Web Console server – (deploy/management)(d) Unique ID key generator – (deploy/uuid-key-generator.sar, lib/autonumber-plugin.jar)(e) HTTP Invoker service – (deploy/http-invoker.sar)(f) Quartz scheduler service – (deploy/quartz-ra.rar)(g) Mail service – (deploy/mail-service.xml, lib/mail\*.jar)(h) Monitoring service – (deploy/monitoring-service.xml,lib/jboss-monitoring.jar)(i) Scheduler service – (deploy/scheduler-service.xml, deploy/schedule-manager-service.xml,lib/scheduler-plugin\*.jar)(j) Messaging (JMS) service – (deploy/messaging, deploy/jms-ds.xml, deploy/jms-ra.rar, lib/jboss-messaging\*.jar)

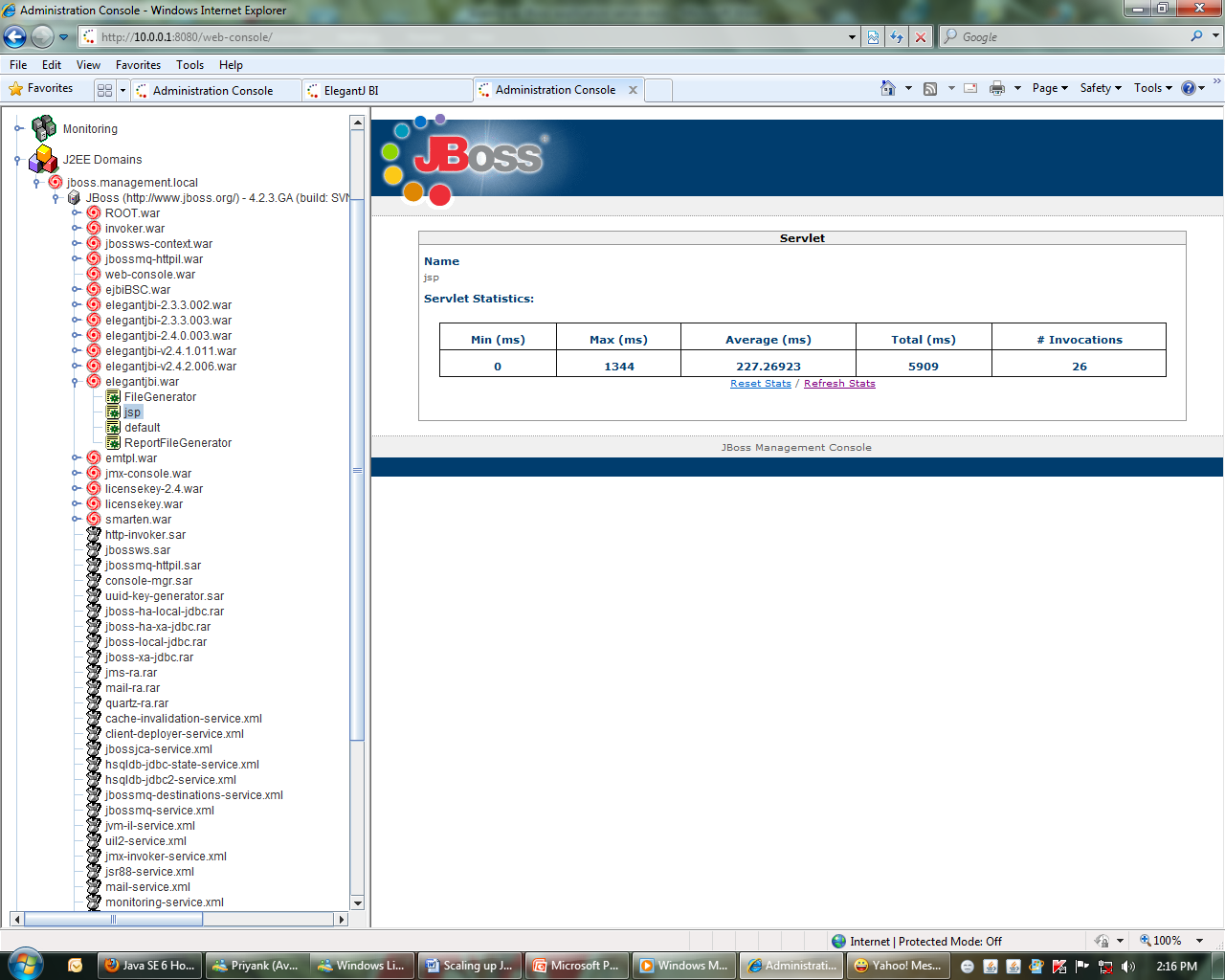
Application level Tuning

The first step is to measure the different components of your app to see where the degradation is.  Is it an external resource (database, message server, etc.)?  Is it internal?  Where is the app spending all its time?

So the first step could be to to **use JBoss JMX agents** and **monitor** the components deployed to the application server. Once it's **clear which component or library takes most of the time** or most of resource you can use a more specialized tool like JProbe and examine the single method or the single objects loaded in memory.

<http://10.0.0.1:8080/web-console/>

J2EE domains🡪 jboss🡪 elegantjbi



**Remove** LOG4J

The JBoss application server uses Log4J, which is an open-source logging **...** If you, instead, wanted to remove just the transaction-related **...**

**It will help to reduce i/o operation**

* **Miscellaneous Tuning**

Monitor and measure Operating system performance and CPU usage under normal condition

Remove all unnecessary programs and applications