White Paper

JBoss Enterprise Application Platform (EAP) → Performance Tuning

Table of Contents

IT architects should be aware of certain principles and best practices to achieve better performance with JBoss EAP.

Introduction and Background Information

Performance tuning is a vital part of deploying and maintaining profitable business applications. You are supposed to tune the application, middleware and database while building or deploying custom, commercial or off-the-shelf solutions. Around 75 percent of performance related issues arise from the application itself.

We may usually assume that an application has the ability to function well straight out of the box. But this need not be true. Applications can have a variety of characteristics. Some applications can function properly with default settings and others may not.

To reap the benefits of company's middleware investments, IT architects should be aware of certain principles and best practices to achieve better performance with JBoss EAP.

This white paper explains the best practices in JBoss Enterprise Application Platform (EAP) performance tuning that can help you avoid the pitfalls when building your application for production.

Performance Priorities

Performance was once regarded as one of the other features of an application. Now-a-days it is considered as an important characteristic of an application. Performance has a significant impact on the productivity of any enterprise.

User experience need not be the primary reason for performance tuning. A well performing solution must normally use fewer software and hardware resources. If performance tuning is done appropriately, companies can optimize their hardware investments. This may involve using fewer systems overall, buying modestly sized new systems or using existing systems for a long period of time.

As far as software is concerned, a well performing solution might use fewer software licenses or CPU counts irrespective of the type of software being used in enterprises. So cost reduction in software packages may save significant money over time.

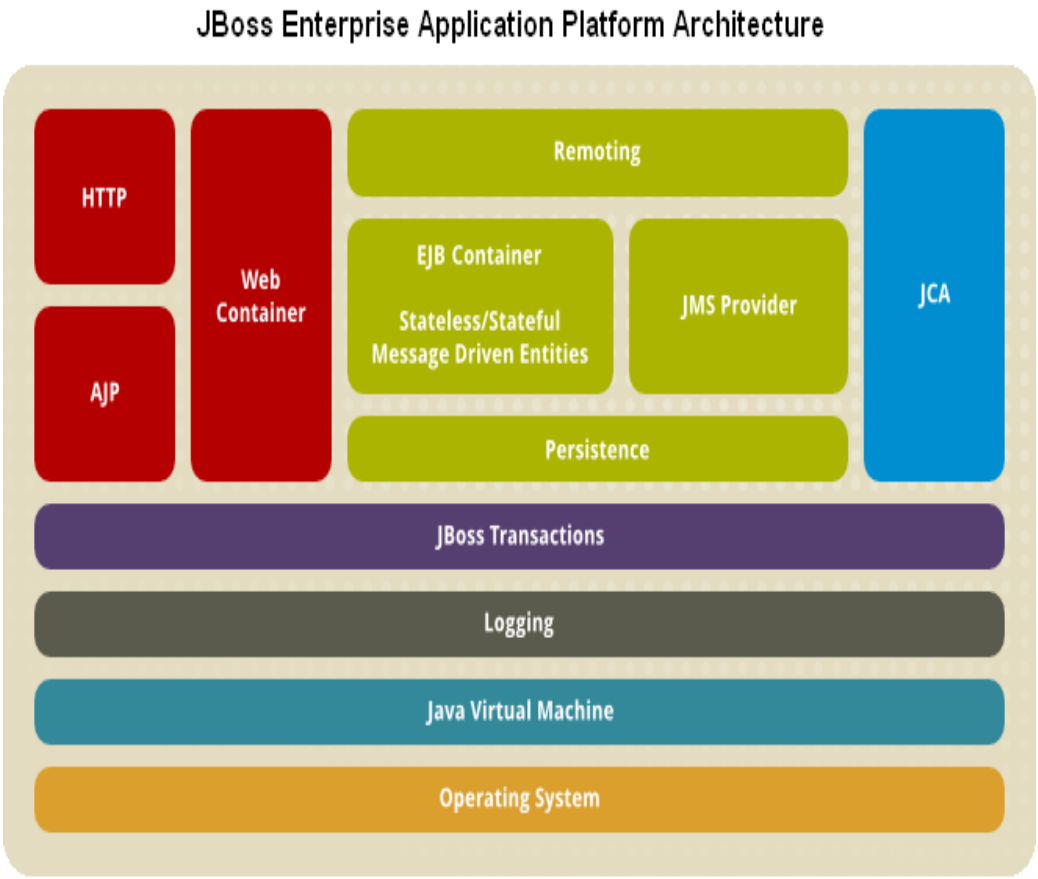
Various layers of the application stack contribute to superior performance. In many instances, the design of an application, database connectivity and other components may have a major impact on overall performance of applications. Many companies spend more time in tuning their applications and databases than the application servers. Note: A best performing application server might have a minimal impact on the total performance of an application.

Principles of Performance Tuning

The design of an application, database connectivity and other components may have a major impact on overall performance of applications.

JBoss EAP is an integrated Java Enterprise Edition application and service hosting platform. It actually enhances the Java standard and provides strong base for Java EE. JBoss EAP runs on multiple virtual machines and operating systems such as Windows, Linux and Unix and is compatible with many industry leading databases. JBoss EAP has to be performance tuned from the start for business critical production environments.

The basic components of JBoss EAP are shown in the following figure:



The workload profile of your application may depend on the specifics of your business, but you must always pay attention to peak workload periods.

This section focuses on the five important principles of performance tuning.

1. Understanding the performance requirements

The initial step in performance tuning is to evaluate the conditions under which it needs to perform. If your application is just a replacement for an existing one then your company already has significant experience with that software application. Precise metrics such as number of transactions per day, number of users and transaction load variations are quickly and effortlessly available.

In case of a new application deployment, you must understand the business context of the application very cautiously. The more you understand the usage of the application, the more successful your tuning will be. Mere assumptions and estimations may not reflect the reality. In one scenario, the IT team tested its application based on an assumption that the workload may not exceed sixty thousand transactions per day. But surprisingly six million transactions occurred on the first production day.

2. Planning for peaks and not averages

One of the major mistakes developers make is to rely on averages. But averages are just not sufficient to make sure that your application performs extremely well during peak load periods. So, your primary goal should be to build application workload profile with special focus on peaks. For instance, many applications might witness daily peaks in the morning and afternoon with a valley during the middle of the day during lunch sessions. Some applications experience peaks at the end of the month or quarter. The workload profile of your application may depend on the specifics of your business, but you must always pay attention to peak workload periods.

3. Instrument your application

It would be better to instrument all applications in order to provide information for performance analysis. Business conditions such as workload curves and customer behaviors can alter over time. If your application has not been instrumented for performance, and if you run into trouble then it is very difficult to trace the existing problems. But if it is instrumented appropriately then you will be in a position to monitor all the business changes very easily and tune your application to match before any problems occur.

To instrument an application in the past, developers used to embed the code within the application. Today many solutions give the required information without the necessity of code. By using JBoss EAP, the container's call statistics can show you the total number of calls, minimum

and maximum call time, concurrent calls and the call-time standard deviation. The JBoss EAP Admin Console can display all such information. Many third party performance management products are also available from Red Hat partners that help in providing the information related to application performance.

In some cases, if the tool does not provide the required information then you can have your own instrumentation by using a framework such as JBoss AOP. This framework is equipped with all features that help you to see all the runtime events. Your decision to write your own instrumentation actually depends on the application, your skills and the significance of the performance characteristics to overall success of an application.

4. Understanding where your application spends time

In some organizations test environments are exact replicas of the production environments. As developers need not worry about scalability issues, this is considered as an ideal approach.

One important motive to instrument your application is to know where the time is being spent. JBoss EAP tools can help you to know the time spent across each application stack layer. With these tools, you can actually understand some part of the equation. For instance, if your application spends more time in the database, you should focus on the statistics of your database to resolve the issue. By just knowing the time spent by your application, you can definitely avoid the shotgun method of tuning the performance. So it is very essential to follow an objective method to understand the time spent.

5. Replicating the production environment

To understand the performance of your application in production, a test environment is essential to run the application. In some organizations test environments are exact replicas of the production environments. As developers need not worry about scalability issues, this is considered as an ideal approach.

Replication cost is still an obstacle for many enterprises because their environments can only employ small systems. In such circumstances, you can create a model that helps you in defining the relationship between the test and production environments. Here you need to consider two factors as mentioned below:

- You are not supposed to assume a linear relationship.
- Use historical data whenever appropriate and you should be conservative in the creation of your model.

JBoss EAP Performance Tuning

Connection pooling can reduce the cost of data source connections by creating a pool of data source connections that are available to be shared by all applications.

To improve the performance and throughput, you need to consider the following five areas and tune the settings:

1. Connection pooling
2. Thread pooling
3. Object pools
4. Logging
5. Caching

1. Connection Pooling

If you wish to maximize the throughput on modern hardware then connection pooling is the most important area to consider.

While connecting to a data source, JBoss EAP has to allocate and de-allocate the resources for every connection. But this is very expensive in terms of time and resources. Connection pooling can reduce the cost of data source connections by creating a pool of data source connections that are available to be shared by all applications. So pooling data source connections is much more efficient than allocating resources.

The activities to be performed include:

- You can begin by changing connection pool settings on the data source definitions, which could be fixed in the deploy directory.
- You need to set the minimum pool size to your desired tuning level.
- Set the maximum pool size at least 30% higher.
- No need to set the maximum size too high. If you don't require that many connections, the pool can shrink automatically.

To determine the proper sizing, you can always monitor your connection usage. A pool that is too small can throttle the application. The connection pool utilization could be monitored from EAP JMX console or JBoss ON.

2. Thread Pooling

Thread pooling is another area to be considered. Before deciding to size thread pools, you have to evaluate their usage and their impact on the performance of your application. Certain application attributes may decide which pools could be used and which ones could be estimated as bottlenecks. But this may vary from one application to the other.

Few thread pools and their usage:

- System thread pool is used for JNDI naming
- HTTPd thread pool is used in case of direct HTTP requests to JBoss EAP.
- JCA thread pool can be used along with JMS.
- JBoss Messaging thread pool usually pools TCP sockets.

Removing connectors

If you can make sure that a particular connector is not necessary for your application then you may remove it. For instance, applications normally use httpd thread pool or the mod_jk thread pool. You can just consider any one connector and remove the other.

Monitoring thread pools

By using the admin console of JBoss EAP, you can monitor the thread pools. For each pool, the console displays the total number of active threads and the size of the queue. You can also define and adjust persistent thread pools by using EAP console as well as JBoss ON.

3. Object Pools

By using the admin console of JBoss EAP, you can monitor the thread pools. For each pool, the console displays the total number of active threads and the size of the queue.

Object pool settings denote the total number of object instances. Two types of pools could be defined for EJB. They are ThreadLocalPool and StrictMaxPool.

- ***ThreadLocalPool:*** Generally, Stateless and Stateful beans use ThreadLocalPool. This is actually supported by an InfinitePool that has no maximum capacity and it can grow according to your application volume.

- ***StrictMaxPool:*** A Message Driven bean uses StrictMaxPool that has a maximum limit. Requests are queued up when that maximum limit is reached. As failed transactions can affect your business, you must consider monitoring StrictMaxPool carefully through the JMX console.

4. Logging

Developers can take advantage of logging in the application lifecycle development and testing. You need to ensure that logging provides useful and relevant information without troubling the throughput. While promoting your application to production, you have to consider the following changes:

- Turning off console logging in production.
- Turning down the logging verbosity.
- Using asynchronous logging.
- Wrapping all debug log statements by using If(debugEnabled()).

5. Caching

JBoss Cache is an intrinsic component of JBoss EAP. So your application can utilize it to cache anything you like. You can enhance the performance by caching EJB entities. Apart from JBoss Cache, you have prepared statement caching that could be set in the data source configuration. With minor changes, an application may experience substantial improvement in throughput.

Results of a Sample Application

Developers can take advantage of logging in the application lifecycle development and testing. You need to ensure that logging provides useful and relevant information without troubling the throughput.

In order to evaluate the performance results with JBoss EAP tuning, let us consider a sample application. Here the sample application is an EJB application along with a couple of servlets for user interface, a stateless and a stateful bean for business logic, a message driven bean for processing asynchronously and few entities for persistence.

Initially, for non optimized tests, we use default JBoss EAP configuration (i.e. thread pool of 64 and default heap of 1.3 GB).

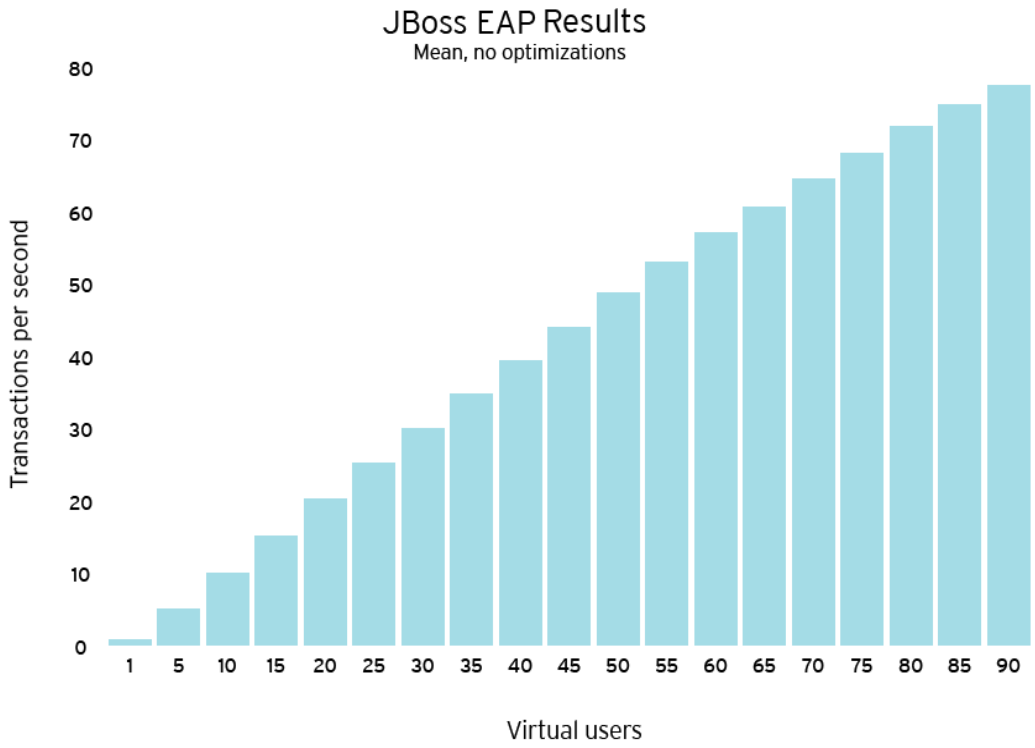
Later, for optimized test, let us adjust the heap size to 3.7 GB with large page memory and the thread pool is increased to 150 along with database connection pool at 150.

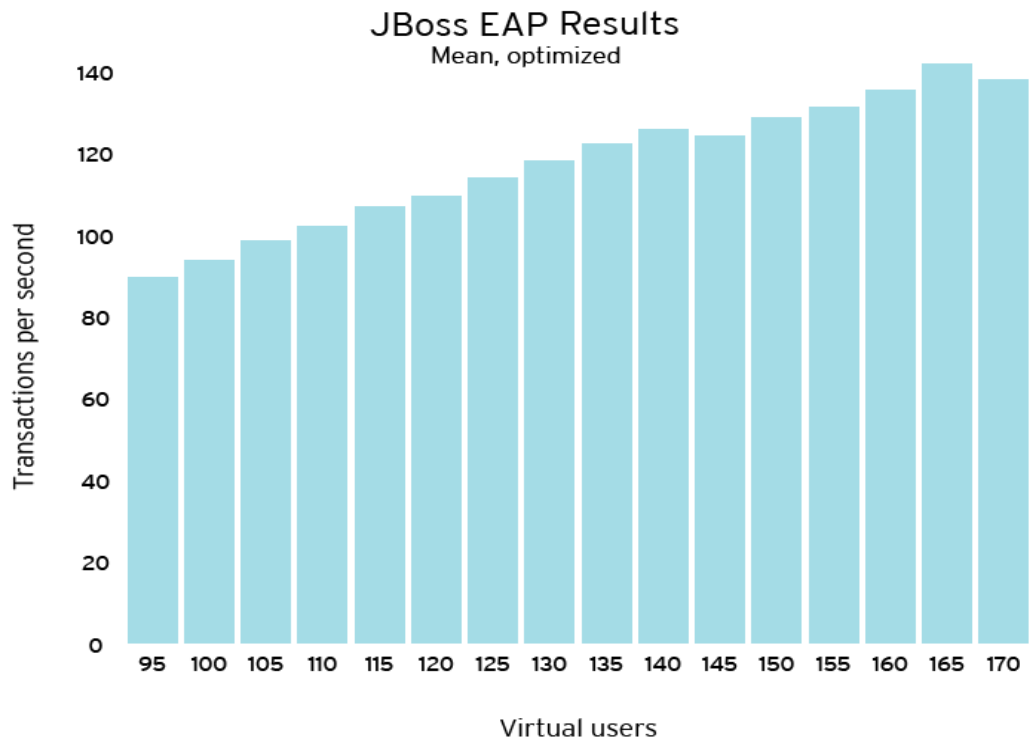
The results for both the scenarios are illustrated as follows:

Here we measure the scalability by using mean transactions per second (TPS). The first test was conducted with default and non optimized configuration as shown in the first graph below:

The second test was conducted with optimized configuration as discussed above. This is shown in the second graph.

On comparing the two graphs, we can notice dramatic changes. The first test scaled up to 90 virtual users and achieved above 75,000 peak TPS. But with optimized JBoss EAP configuration, we almost doubled the number of virtual users. We also doubled the peak number of transactions at above 140,000 TPS.


JBoss EAP Results
Mean, no optimizations
Transactions per second
0
10
20
30
40
50
60
70
80
1
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
Virtual users


JBoss EAP Results
Mean, optimized
Transactions per second
0
20
40
60
80
100
120
140
95
100
105
110
115
120
125
130
135
140
145
150
155
160
165
170
Virtual users

JBoss Cache is an intrinsic component of JBoss EAP. Your application can utilize it to cache anything you like.

Conclusion

Tuning performance is not a one time task. It is actually a cyclical phenomenon that involves monitoring the performance, changing the configuration and reviewing. Each change should have a well defined objective.

As all workloads are not the same, all changes need not enhance the performance. So it is always advised to apply one or two changes at a time and observe the results.