



# JBoss Enterprise Application Platform Tuning

Andrig T. Miller – but you can call me Andy

JBoss, a division of Red Hat

VP of Engineering

January, 2008



# Agenda

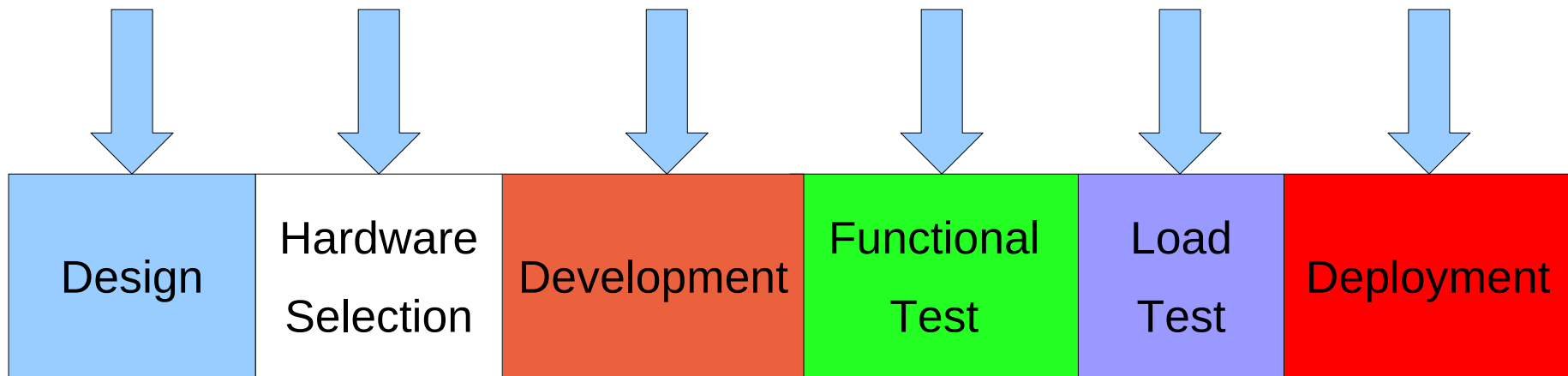
- Performance tuning basics.
- Enterprise Application Platform tuning.
- Linux specific tuning.
- Database and Storage performance tuning.
- Performance tuning as applied to an actual EJB 3 application.

# When should you consider performance tuning?

- When “performance is great”?
- When you asking yourself, why is it running so slow?
- When people are saying, we need to upgrade the hardware?
- When people are questioning the design meeting the load requirements?
- Or, when your down in production!

# Typical Life-cycle

- The short answer to the question posed in the previous slide is non-of-the-above.
- Looking at a typical project life-cycle, each step illustrated below has inputs, and without those inputs, you cannot be successful.

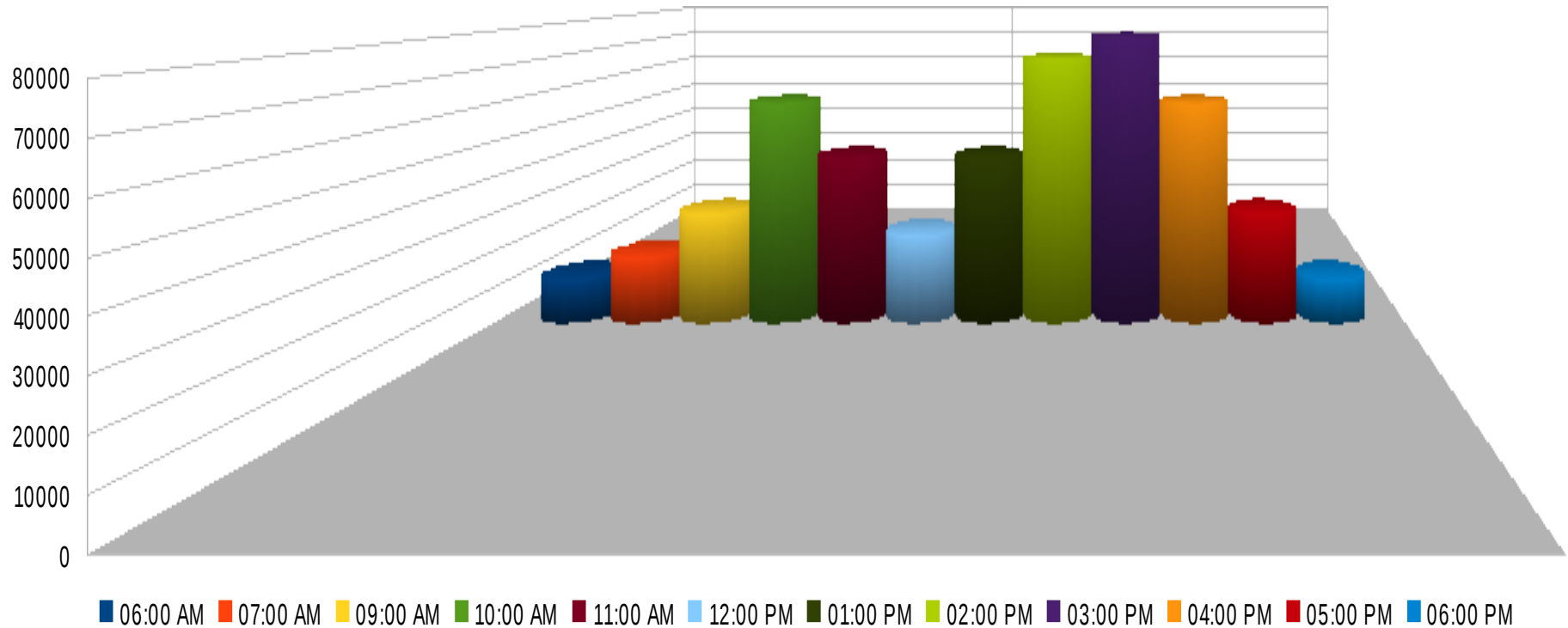


# Performance Tuning Basics

- Understand your performance requirements.
  - Is this new software replacing an existing solution?
    - In this case, you should have metrics from the current solution to base the requirements on.
  - Is this a totally new solution, with no past history?
    - Under this scenario, it becomes important to understand the business case surrounding the solution.
  - In both cases, you need to understand the peak time periods of the workload, and how that differs from the workload in non-peak times.
    - Peaks during the day.
    - Peaks during the week.
    - Peaks during the month, such as end-of-month processing.
    - Peaks during quarters or years, such as end-of-quarter, or seasonal peaks for your business.

# Example of Workload Curve

Order Transaction Volume by Hour



# Don't Count on Averages!

- One of the biggest mistakes developers make, is looking at their performance requirements as an average over a runtime period.
  - Your requirements are bounded by the peaks in the workload curve.
  - In the previous example, the peak is 6.5 times the low point during a typical day.
    - The average is just under 2 times the peak.
    - It become obvious that if you shoot for the average you will never be able to support the peak workload.
  - This kind of data may be difficult to come by, because it may be influenced by factors unknown to you as the developer.

# Instrument

- Applications should be instrumented for performance analysis.
  - In many cases, it will prove that your performance requirements, and the peak workloads assessed before production were incorrect.
  - Without instrumentation, you will not have accurate data to track.
  - Also, workloads can change over time, as business models, or business conditions change.
  - In the past, instrumentation would have had to be embedded in the application.
    - Today, there are many solutions for this that don't require developers to code.
      - Commercial products, and the JBoss AOP framework can be used for just this purpose.
      - Turn on call statistics in the containers, and Hibernate statistics.
      - The EAP already has these features available to you.

# Understand Where Time is Being Spent

- Along the lines of instrumentation, it is important to understand where your application spends its time.
  - So, besides the fact that workload curves change over time, you need the information of where time is being spent in your various transactions.
  - This will help you to avoid, what I call the “shotgun method” of performance tuning.
    - What I mean by that, is you start shooting a spread all over the place, but you don't necessarily hit the target of any performance problem you have.
    - I have seen this countless times, and in almost every case, performance issues linger for weeks, months, or even years because you don't know the where the problem is occurring.

# Modeling Results

- Most companies cannot afford exact replicas of their production environment for load testing.
  - In most cases, you are going to have to model any results you get during load testing for the actual deployment environment.
  - Be conservative, be conservative, be conservative!
    - If you take a given result, and let's say the production environment is twice (not necessarily capacity) the size of the load testing environment, don't double your numbers and compare that to your requirements.
    - In all cases, you will not get linear results.
      - Vendors may tell you that they get linear results, but don't believe them.
      - Also, they may be able to show you linear, or near linear results on some benchmark, but that benchmark is not your application!
    - If you have past experiences, retain that data and feed it into your model.

# EAP Tuning

- Seventy-five percent of all performance problems are the result of the application, not the middle-ware or the operating system.
  - With that said, there are things that affect performance and throughput in the middle-ware, and may need some attention.
  - Let's look at the following areas:
    - Connection Pooling.
    - Thread Pools.
    - Object and Component Pools.
    - Logging.
      - Both verbosity of logging and method.
      - Wrapping of debug log statements.
    - Caching.
    - Clustering and Replication.

# Connection Pooling

- Database connections are expensive to setup and tear down.
  - I have seen applications that created new connections to the database with every query or transaction, and then closed that connection.
    - This adds a great deal of overhead, and throttles the application.
    - Rely on the data source definitions you can setup in the deploy directory of the EAP, and utilize the connection pool settings.
  - The EAP has robust connection pooling, and you should monitor your connection usage from the database to determine proper sizing.
    - Too small a pool will also throttle the application as the EAP will queue the request for a default of 30,000 milliseconds (30 seconds) before giving up and throwing an exception.
  - You can monitor the connection pool utilization from the JMX Console, as well as with database specific tools.

# Example Data Source Definition

```
<datasources>
  <local-tx-datasource>
    <jndi-name>MySQLDS</jndi-name>
    <connection-url>jdbc:mysql://<host>:3306/Schema</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>someuser</user-name>
    <password>somepassword</password>
    <exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter</exception-sorter-class-name>
    <min-pool-size>75</min-pool-size>
    <max-pool-size>100</max-pool-size>
    ...
    <!-- corresponding type-mapping in the standardjbosscomp-jdbc.xml -->
    <metadata>
      <type-mapping>mySQL</type-mapping>
    </metadata>
  </local-tx-datasource>
</datasources>
```

# Thread Pooling

- The EAP has robust thread pooling, that should be sized appropriately.
  - The EAP server has a file called `jboss-service.xml` in the `conf` directory that defines the system thread pool.
    - There is a setting that defines the behavior if there isn't a thread available in the pool for execution.
      - The default is to allow the calling thread to execute the task.
    - You can monitor the queue depth of the system thread pool through the JMX Console, and determine from that if you need to make the pool larger.
  - In `server.xml` for JBossWeb, there is a `maxThreads` setting.
    - There are other thread pools defined such as:
      - In `cluster-service.xml` `JRMPInvoker`, in `jbossjca-service.xml` there is a `WorkerManager` thread pool, and in `deploy-hasingleton`, there is a `jbossmq-service.xml` thread pool.

# Example Thread Pool

```
<!-- A Thread pool service -->

<mbean code="org.jboss.util.threadpool.BasicThreadPool" name="jboss.system:service=ThreadPool">

  <attribute name="Name">JBoss System Threads</attribute>

  <attribute name="ThreadGroupName">System Threads</attribute>

  <!-- How long a thread will live without any tasks in MS -->

  <attribute name="KeepAliveTime">60000</attribute>

  <!-- The max number of threads in the pool -->

  <attribute name="MaximumPoolSize">10</attribute>

  <!-- The max number of tasks before the queue is full -->

  <attribute name="MaximumQueueSize">1000</attribute>

  <!-- The behavior of the pool when a task is added and the queue is full.
  abort - a RuntimeException is thrown
  run - the calling thread executes the task
  wait - the calling thread blocks until the queue has room
  discard - the task is silently discarded without being run
  discardOldest - check to see if a task is about to complete and enqueue
               the new task if possible, else run the task in the calling thread -->

  <attribute name="BlockingMode">run</attribute>

</mbean>
```



# Object and Component Pools

- There are a variety of other pools that are all configurable within the EAP. For example:
  - In standardjboss.xml there is a JMSPool.
    - Besides the pool, there is also a retry parameter that defaults to 10. This is a lot of retries, so you may want to change this to a smaller number.
  - In that same file are specific J2EE 1.4 container configurations, that have pool sizes.
    - Examples are “Standard CMP 2.x Entity” and “Standard Stateless Session Bean”.
  - For EJB 3, there are pools defined in ejb3-interceptors-aop.xml.
    - This file is in the deploy directory, unlike standardjboss.xml which is in the conf directory.
  - Again, all these pools can be monitored through the JMX Console. The JMX Console is your friend when performance tuning.

# Logging

- The default log4j configuration is appropriate for development, but not necessarily for a production environment.
  - In the default configuration, console logging is enabled.
    - This is great for development, especially within the IDE, as you get all the log messages to show in the IDE console view.
  - In a production environment, console logging is very expensive. Many applications can get away with it, but many can't.
    - Turn off console logging in production.
    - In the EAP, there is a new configuration, called production, and console logging is already turned off in that configuration.
  - Turn down the verbosity level of logging if its not necessary.
    - The less you log, the less I/O will be generated, and the better the overall throughput will be.

# Logging (Cont'd)

- Use asynchronous logging.
  - In my experience, with high throughput applications, this can make a real difference.
    - Locking overhead is reduced, and the time to log is not added to your transaction times.
- Wrap debug log statements with:
  - If (debugEnabled())
  - Your application will create all the string objects for each of the log statements and Log4j creates the LoggingEvent object for each log statement, regardless of the log level that is set.
    - I have seen this lead to thousands and thousands of temporary String and LoggingEvent objects, causing garbage collection issues, and reducing throughput dramatically.
    - If your applications are anything like what I have seen, there are lots of debug log statements in your code!

# Caching

- JBoss Cache is an integral part of the EAP, and can be used directly by your application to cache anything you want.
  - I have personally seen it used to cache product catalog search results, with dramatic performance improvements.
    - It's especially useful where results don't change much, but are expensive to generate in the first place.
  - By far, one of the easiest potential performance enhancements you can make is caching of EJB 3 entities.
    - You simply define what entities you want cached in the persistence.xml that you deploy with your EJB 3 application.
    - You define the cache size and eviction policy in ejb3-entity-cache-service.xml found in the deploy directory.
  - **WARNING:** be careful with cache size and eviction policy, you only have so much heap space.
  - **WARNING:** caching is not a silver bullet, and can sometimes reduce throughput.

# Example persistence.xml

```
<persistence>

<persistence-unit name="services" transaction-type="JTA">

<provider>org.hibernate.ejb.HibernatePersistence</provider>

  <jta-data-source>java:/MySQLDS</jta-data-source>

  <properties>

    <property name="hibernate.default_catalog" value="EJB3"/>

      ...

    <property name="hibernate.cache.provider_class"

      value="org.jboss.ejb3.entity.TreeCacheProviderHook"/>

    <property name="hibernate.treecache.mbean.object_name"

      value="jboss.cache:service=EJB3EntityTreeCache"/>

    <property name="hibernate.ejb.classcache.services.entities.Customer" value="read-only"/>

    <property name="hibernate.ejb.classcache.services.entities.Inventory"

value="transactional"/>

      ...

  </properties>

</persistence-unit>

</persistence>
```

# Clustering and Replication

- One simple statement here – Use buddy replication!
  - In our own internal testing buddy replication scales well, regardless of cluster size, for HTTP session replication.
    - You might want to also consider have a dedicated network for cluster communication, and use jumbo frames (assuming gigabit ethernet).
  - Analyze your HTTP session data carefully to see if fined grained replication is a fit for your application.
    - If you have a very small amount of field level changes, you could get benefits out of fined grained replication.
    - Fine grained replication can be much slower if you have lots of fields changing in the objects that you have in your HTTP session.
  - In the future, buddy replication will be available for Stateful Session beans as well.
  - You configure buddy replication in jboss-service.xml that is in the server/<configuration>/deploy/jboss-web-cluster.sar/META-INF.

# Linux Specific Tuning

- For 64-bit systems, use Linux's large memory page support (HugeTLB).
  - Default memory page size is typically 4KB. When you are addressing large amounts of memory this quickly adds up to lots of memory pages.
    - Even just one gigabyte of memory, requires 262,144 memory pages!
  - Large memory page support usually starts with 2MB memory pages, and can be as large as 256MB on some architectures.
  - All the major JVM's support large memory pages on Linux, but its a little trickier to setup than one would think.
  - Besides the system overhead of mapping so many memory pages, large memory pages on Linux cannot be swapped to disk.
    - Obviously, having your heap space swap to disk will reek havoc on the performance of your application.

# Large Page Support

- The Sun JVM requires the following option, passed on the command-line, to use large pages:
  - `-XX:+UseLargePages`
    - The Sun instructions leave it at that you will most likely get the following error:
      - Failed to reserve shared memory (error-no=12).
  - Next, you set the following in `/etc/sysctl.conf`
    - `kernel.shmmax = n`
      - Where *n* is equal to the number of bytes of the maximum shared memory segment allowed on the system. You should set it at least to the size of the largest heap size you want to use for the JVM, or alternatively you can set it to the total amount of memory in the system.
    - `vm.nr_hugepages = n`
      - Where *n* is equal to the number of large pages. You will need to look up the large page size in `/proc/meminfo`.
    - `vm.huge_tlb_shm_group = gid`
      - Where *gid* is a shared group id for the users you want to have access to the large pages.

# Large Page Support (Cont'd)

- Next, set the following:
  - In `/etc/security/limits.conf`:
    - `<username>`                    `soft`            `memlock`            `n`
    - `<username>`                    `hard`            `memlock`            `n`
      - Where **<username>** is the runtime user of the JVM.
      - Where **n** is the number of pages from `vm.nr_hugepages` \* the page size in KB from `/proc/meminfo`.
  - You can now enter the command `sysctl -p`, and everything will be set, and survive a reboot.
    - You can tell that the large pages are allocated by looking at `/proc/meminfo`, and seeing a non-zero number for `HugePages_Total`.
      - This may fail without a reboot, because when the OS allocates these pages, it must find contiguous memory for them.
  - **WARNING:** when you allocate large page memory, it is not available to applications in general and your system will look and act like it has that amount of memory removed from it!

# Large Page Support Example

- I have a server with 8GB of memory and will allocate 6GB to be shared by the EAP JVM and a MySQL database.
  - Page size is 2MB (2048 KB), as shown in /proc/meminfo
    - Hugepagesize: 2048KB
  - Here's my /etc/sysctl.conf
    - # Change maximum shared memory segment size to 8GB
    - kernel.shmmax = 8589934592
    - # Add the gid to the hugetlb\_shm\_group to give access to the users
    - vm.hugetlb\_shm\_group = 501
    - # Add 6GB of in 2MB pages to be shared between the JVM and MySQL
    - vm.nr\_hugepages = 3072
  - Calculations are as follows:
    - $1024 * 1024 * 1024 * 8 = 8589934592$
    - $(1024 * 1024 * 1024 * 6) / (1024 * 1024 * 2)$  or  $6GB / 2MB = 3072$  pages

# Large Page Support Example (Cont'd)

- Here is my `/etc/security/limits.conf`:
  - `# Add the limits for memlock to allow the JVM and MySQL to access the large`
  - `# page memory.`
  - `jboss soft memlock 6291456`
  - `jboss hard memlock 6291456`
  - `mysql soft memlock 6291456`
  - `mysql hard memlock 6291456`
  - `root soft memlock 6291456`
  - `root hard memlock 6291456`
- Calculation is as follows:
  - `3072 large pages * 2048 KB page size – 3072 *2048 = 6291456`
- I also added the `jboss` and `mysql` users to the `501` group in `/etc/group`, which is called `hugetlb` (you can call this anything you want). This gives those users permission to attach to the shared memory segment.

# Final Word on Large Pages

- Finally, after starting the applications (in my case the EAP JVM and MySQL), you should see something like this:
  - HugePages\_Rsvd: 1182
- If you don't see a non-zero value in /proc/meminfo for the above parameter, than you are not using the large pages, and something is not configured correctly.
  - With MySQL I ran into a problem where the SE Linux policy was preventing it from accessing the large pages, so check /var/log/messages for avc denied messages (error-no=13 “permission denied” in the mysqld.log).
- Good Luck!

# Tuning the VM

- In Linux, the virtual memory manager is tunable, and in some cases you can get benefits with changing the settings.
  - In `/etc/sysctl.conf` you can set `vm.swappiness` to 1. This will prevent applications from being swapped to disk when there is memory pressure.
    - On a server running the EAP, and/or a database, the last thing you need is to have file system buffer cache to be favored over your application code.
    - This setting also relates to the database and storage topic, later in this presentation, so don't set this without understanding that section first! Your settings there will determine whether it makes sense to set this.

# Database and Storage Tuning

- With databases cache is king!
  - Modern database, especially 64-bit, are extremely efficient at caching data.
    - I have seen OLTP applications with Oracle buffer caches as large as 15GB with very good results.
      - In the early days of 64-bit databases large buffer sizes would slow performance due to elongated search times, but this is not true anymore.
    - The more read intensive you are, the more cache helps.
    - Of course, if you are write heavy, or the data set is so large (e.g. data warehouse), then a large cache won't help, and most likely will be slower.
      - Understand your read to write ratios!
    - Most applications I have seen over the past 10 years are read intensive.
      - This tends to be true, due to the fact that most applications drive their business logic by reading data from the database, versus being hard coded.

# Database and I/O

- Use `DIRECT_IO` if your database supports it!
  - With a large cache or a write intensive workload, you should be avoiding double buffering with the file system buffer cache all together.
    - There is a note in the MySQL 5 documentation, that states that queries may slow down by up to three times when using `DIRECT_IO`, but this is simply not the case.
      - If you have a properly sized buffer pool, this simply will not happen. On the other hand, if your buffer pool is too small, you do gain benefits on reads from the file system buffer cache, but you should size the buffer pool large enough to begin with.
    - I have seen `DIRECT_IO` reduce CPU utilization by as much as 70%, and improve throughput dramatically.
  - This setting is related to the `vm.swappiness` kernel parameter we discuss earlier. If you are using `DIRECT_IO`, you don't want the virtual memory manager to favor the file system buffer cache, since you will not be using it!
- Use asynchronous I/O if your database supports it.

# Database and I/O (Cont'd)

- Storage layout for your database is very important!
  - Utilize your databases capability to divide the tables, tablespaces, etc., in different files so you can spread them out over more physical disks.
    - Look to keep I/O rates between 100 and 120 I/O's per second on 10,000 RPM drives, and between 120 and 150 I/O's per second on 15,000 RPM drives.
    - Separate database log files and data/index files onto different physical disks.
- If your workload is read intensive, use large block or page sizes (depending on your databases capabilities and terminology).
  - Large block or page sizes will mean less I/O, and more efficient index scans for your queries.

# Storage

- Where storage is concerned, you are looking for capabilities to maximize I/O's per second, with the lowest possible latency.
  - I would target 4 ms average access times as an upper bounds for your database.
  - In order to achieve this, depending on your applications needs, you may have to have a storage system that can stripe data over many drives.
    - Different storage systems provide different limits in this area, but a minimum would be striping over at least seven physical disks.
    - There are storage systems that can stripe over many more disks than seven, and they aren't necessarily more expensive.
  - If you select a NAS or iSCSI SAN solution, then isolate the storage network from the rest of the network.
    - In this case, you should also use jumbo frames (9,000 byte MTU) for the storage network.

# Storage and Writes

- There is usually a lot of concern with storage and write caching.
  - I would encourage you to use storage solutions that support write caching, as it improves performance considerably, with the following caveats:
    - Favor storage solutions that have non-volatile write caches, mirrored caches, and/or battery backed write caches.
    - Without these fail safes in place, you can lose your data!

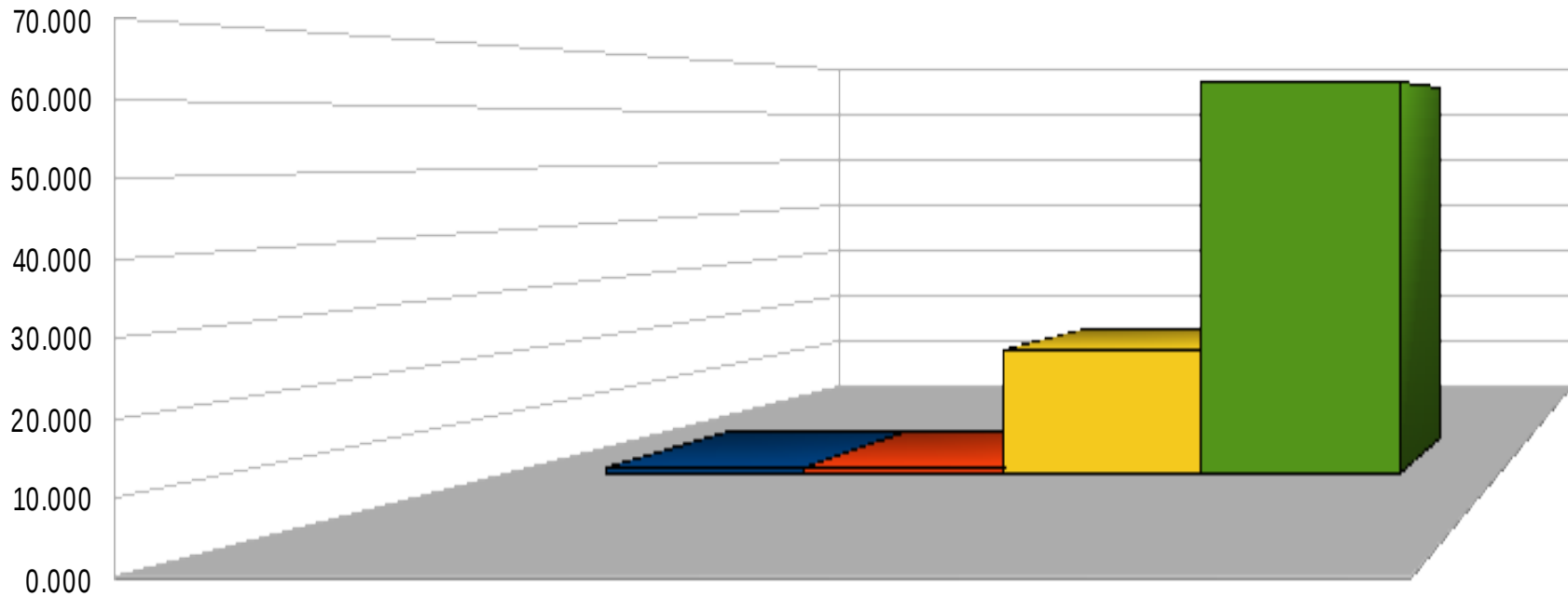
# Example Application

- I have an example application that I load tested using the EAP's default configuration values (with one minor exception), along with all Linux parameters at their defaults.
  - I loadtested this application, using Grinder, and measured the most throughput that I could achieve with all of these settings at the default.
- I took the same application, and applied many of the optimizations discussed here, both to the EAP and the OS, and measured the throughput that I could achieve.
- This application is an EJB 3 application, with two servlets for the UI, stateless and stateful session beans for most of the business logic, a message driven POJO for some asynchronous processing, and entities for the persistence.
- Without further ado, here are the results!

# Results!

## Results!

All tests were done with 3.5GB heap, and a data source definition with sufficient connections in the pool.



- Baseline EJB 3 Application with 1 Virtual User - TPS (Mean) - No Optimizations
- Baseline EJB 3 Application with 1 Virtual User - TPS (Mean) - Optimized
- Top Throughput Test - 25 Virtual Users - TPS (Mean) - No Optimizations
- Top Throughput Test - 75 Virtual Users - TPS (Mean) - Optimized

# Question and Answers!