JBoss WORLD
CHICAGO 2009

FOLLOW US:

TWITTER.COM/REDHATSUMMIT

TWEET ABOUT US:

ADD #SUMMIT AND/OR #JBOSSWORLD TO THE END OF YOUR EVENT-RELATED TWEET

presented by

JBoss WORLD
CHICAGO 2009

Accelerate Your JBoss

Andrig (Andy) Miller
VP of Engineering, Red Hat
September, 2009

Agenda

Performance tuning basics.

Enterprise Application Platform (EAP) tuning.

Linux specific tuning.

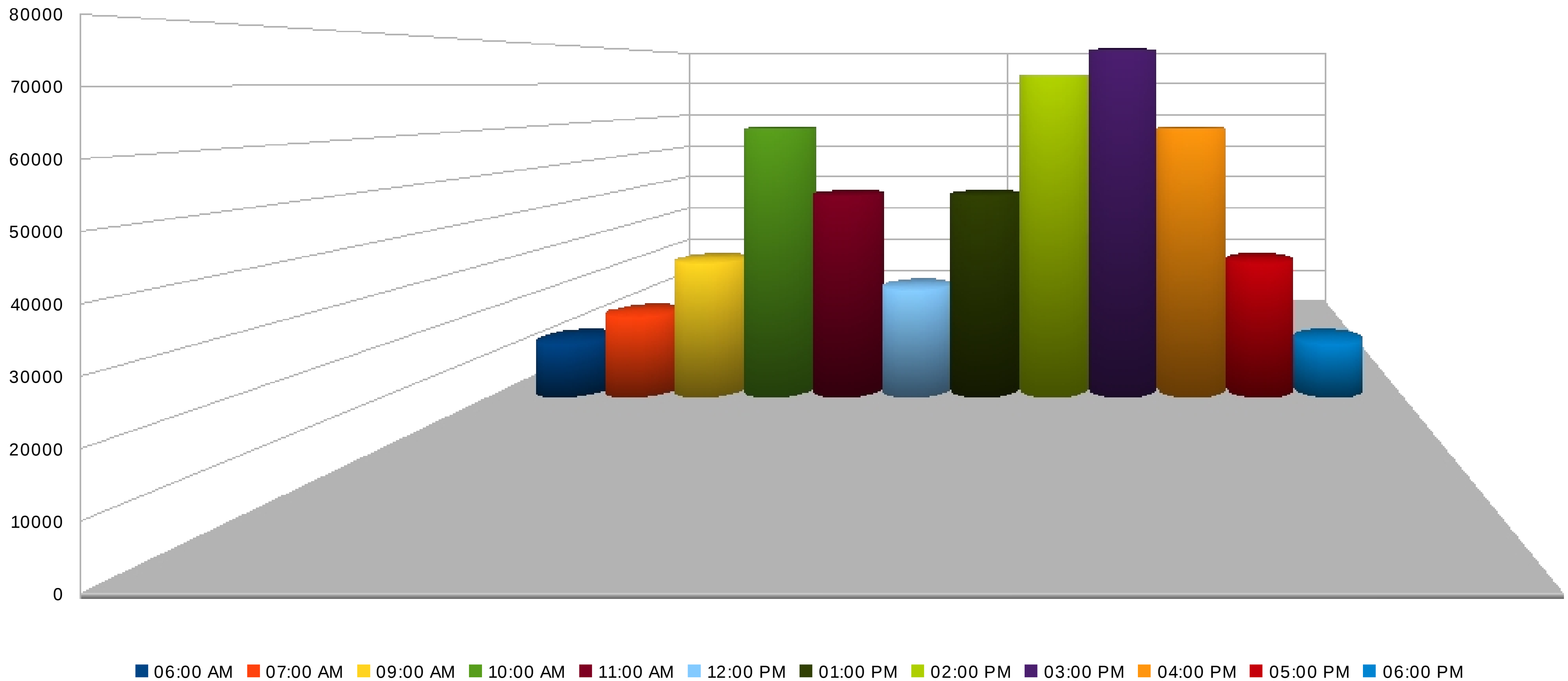
High-level database performance tuning.

Performance tuning as applied to an actual EJB 3 application.

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 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, or week, or month, such as end-of-month processing.
    - Peaks during quarters or years, such as end-of-quarter, or seasonal peaks for you business.

Example of Workload Curve



Don't Count on Averages!

One of the biggest mistakes developers make, is looking at the performance requirements as an average over a run-time 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 becomes 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 where 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.
    - JBoss Operations Network (JON).
      - Turn on call statistics in the containers, and Hibernate statistics.
    - The EAP already has these features available to you.
    - Commercial products.
    - For the adventurous, you can also use the JBoss AOP framework and roll your own instrumentation.

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 where the problem is occurring.

Modeling Results

Most companies cannot afford exact replicas of their production environment for load testing.

- Of course, I'm always surprised at the number of companies that actually have duplicate test environments.
  - I was just never that fortunate to work for a company that did ;-).
- In most cases, you are going to have to model any results you get during load testing.
- 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/Component pools.
    - Logging.
      - Both verbosity and method.
      - Wrapping of debug log statements.
    - Caching.

JBoss WORLD CHICAGO 2009

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.
  - 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 admin console as well as with database specific tools.

Screenshot of Administration Console

File Edit View History Bookmarks Tools Help

http://192.168.1.22:8080/admin-console/secure/resourceInstanceMetrics.seam?path

Add to Wish List | Most Visited | Release Notes | Gmail - Inbox | Reader | Blogger: Create your ... | Zimbra | AXIOM - Sign In

iGoogle | Problem ... | Problem ... | Problem ... | Problem ... | Problem ... | Problem ... | JBos... | JBoss J... | MBean I...

JBoss AS Administration Console

Welcome admin [Logout]

- RPC Manager
- RegionManager
- Transaction Table
- Tx Interceptor
- Lock Manager
- jboss.cache:config=sfsb-cache
- jboss.cache:config=standard-s
- Applications
  - EARs
    - OrderManagerApp.ear
  - EJB2 JARs
  - EJB3 JARs
  - Embedded EJB3 JARs
  - Embedded WARs
  - RARs
  - WARs
- Resources
  - JBoss Web
    - Connectors
      - ajp://192.168.1.22:8009
      - http://192.168.1.22:8080
    - Virtual Hosts
  - Scripts
  - Connection Factories
  - Datasources
    - Local Tx Datasources
      - MySQLDS
    - No Tx Datasources
    - XA Datasources
  - JMS Destinations

: jbosstesting.miller.org : JBossAS Servers : JBoss EAP 5 (all) : Resources : Datasources : Local Tx Datasources : MySQLDS

MySQLDS

Summary | Configuration | Metrics | Control | Content

Status: Available

View the numeric metrics and traits for this resource.

Traits

Run State: RUNNING

Local Transaction: true

Pool JNDI Name: MySQLDS

Numeric Metrics

Name	Value	Description
Category: performance		
Available Connection Count	85	the maximum number of connections that are available
Connection Count	75	the number of connections that are currently in the pool
Connection Created Count	75	the number of connections that have been created since the datasource was last started
Connection Destroyed Count	0	the number of connections that have been destroyed since the datasource was last started
In Use Connection Count	15	the number of connections that are currently in use
Max Connections In Use Count	50	the most connections that have been simultaneously in use since this datasource was started
Max Size	100	Max Size
Min Size	75	Min Size

Refresh

JBoss AS Administration Console 1.3.0.GA (r613) - Powered by Embedded Jopr

© 2002-2009 Red Hat Middleware, LLC. All rights reserved. JBoss is a registered trademark of Red Hat, Inc.

http://192.168.1.22:8080/admin-console/secure/summary.seam?path=-8&conversationId=45&conversationPropagation=end

Example Data Source Definition

```
<datasources>
   <local-tx-datasource>
      <jndi-name>MySQLDS</jndi-name>
      <connection-url>jdbc:mysql://[host]:3306/[database]</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>
      <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-isolation>
      <prepared-statement-cache-size>100</prepared-statement-cache-size>
   <shared-prepared-statements>true</shared-prepared-statements>
   </local-tx-datasource>
</datasources>
```

Thread Pooling

- The EAP has robust thread pooling, that should be sized appropriately.
  - The EAP server has several thread pools, and depending on your application, different ones will get used.
    - The system thread pool defined in jboss-service.xml in the conf directory is for JNDI naming.
      - This rarely needs to be updated from the default.
    - The httpd thread pool in JBoss Web is defined in server.xml file under <server>/deploy/jboss-web-deployer.
      - Used when making HTTP requests directly to EAP.
    - The AJP thread pool is also defined in the same file, just in its connector section.
      - Used when making HTTP requests through mod_jk.
    - A word about the new load balancer, mod_cluster. When using mod_cluster, you setup a listener in JBoss Web, but it uses the AJP and/or HTTPD connector, and corresponding thread pool.

Thread Pooling (Cont'd)

EAP Thread Pools

- The JCA thread pool is used in conjunction with JMS, as JBoss Messaging uses JCA inflow as its integration into EAP.
  - This can be configured in <server>/deploy/jca-jboss-beans.xml, and is called WorkManager thread pool.
- JBoss Messaging also has its own thread pools, depending on whether you are invoking this via a remote client or in the same JVM there is a setting that you can tune.
  - For remote clients, you set a client thread pool that pools the TCP sockets via <server>deploy/messaging/remoting-bisocket-service.xml.
  - For in JVM clients, all the processing will occur on the JCA thread pool (WorkManager).
    - One note here, is that if you have a message driven bean, that invokes other beans, such as stateless session beans, those beans will also run on the JCA thread pool.

Thread Pooling (Cont'd)

- EAP Thread Pools
  - For EJB 3 remote clients there is a thread pool defined in <server>/ejb3.deployer/META-INF/jboss-service.xml.
    - If your clients are in the same JVM they will run on whatever thread pool they are already on.
      - For example, a web request comes through the AJP connector, when it calls an EJB 3 bean, it will continue executing on the AJP connector thread pool.
    - You can monitor thread pool usage for the JBoss Web connectors (httpd and ajp) through the new administration console.
    - The other thread pools can be monitored through the JMX console.

Screenshot of Administration Console

JBoss AS Administration Console - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://192.168.1.22:8080/admin-console/secure/resourceInstanceMetrics.seam?path

Add to Wish List | Most Visited | Release Notes | Gmail - Inbox | Reader | Blogger: Create your ... | Zimbra | AXIOM - Sign In

iGoogle | Problem ... | Problem ... | Problem ... | Problem ... | Problem ... | Problem ... | JBos... | JBoss J... | MBean I...

JBoss AS Administration Console

Welcome admin [Logout]

- jboss.cache:config=mvcc-entity,s
  - Cache
  - Interceptor
  - Data Container
  - RPC Manager
  - RegionManager
  - Transaction Table
  - Tx Interceptor
  - Lock Manager
- jboss.cache:config=sfsb-cache,s
- jboss.cache:config=standard-ses
- Applications
  - EARs
    - OrderManagerApp.ear
  - EJB2 JARs
  - EJB3 JARs
  - Embedded EJB3 JARs
  - Embedded WARs
  - RARs
  - WARs
- Resources
  - JBoss Web
    - Connectors
      - ajp://192.168.1.22:8009
      - **http://192.168.1.22:8080**
    - Virtual Hosts
  - Scripts
  - Connection Factories
  - Datasources
    - Local Tx Datasources

: jbosstesting.miller.org : JBossAS Servers : JBoss EAP 5 (all) : Resources : JBoss Web : Connectors : http://192.168.1.22:8080

http://192.168.1.22:8080

Summary | Configuration | **Metrics** | Control | Content

Status: ✔ Available

View the numeric metrics and traits for this resource.

Traits

There are currently no traits available.

Numeric Metrics

Name	Value	Description
Category: utilization		
Request Count	20,305	the total number of requests processed since the last restart
Error Count	1	the number of errors while processing requests since the last restart
Current Active Threads	51	the number of threads for this connector that are currently active
Current Thread Count	65	the number of threads for this connector that currently exist
Category: performance		
Maximum Request Time	13.5s	the maximum time it took to process a request since the last restart

Refresh

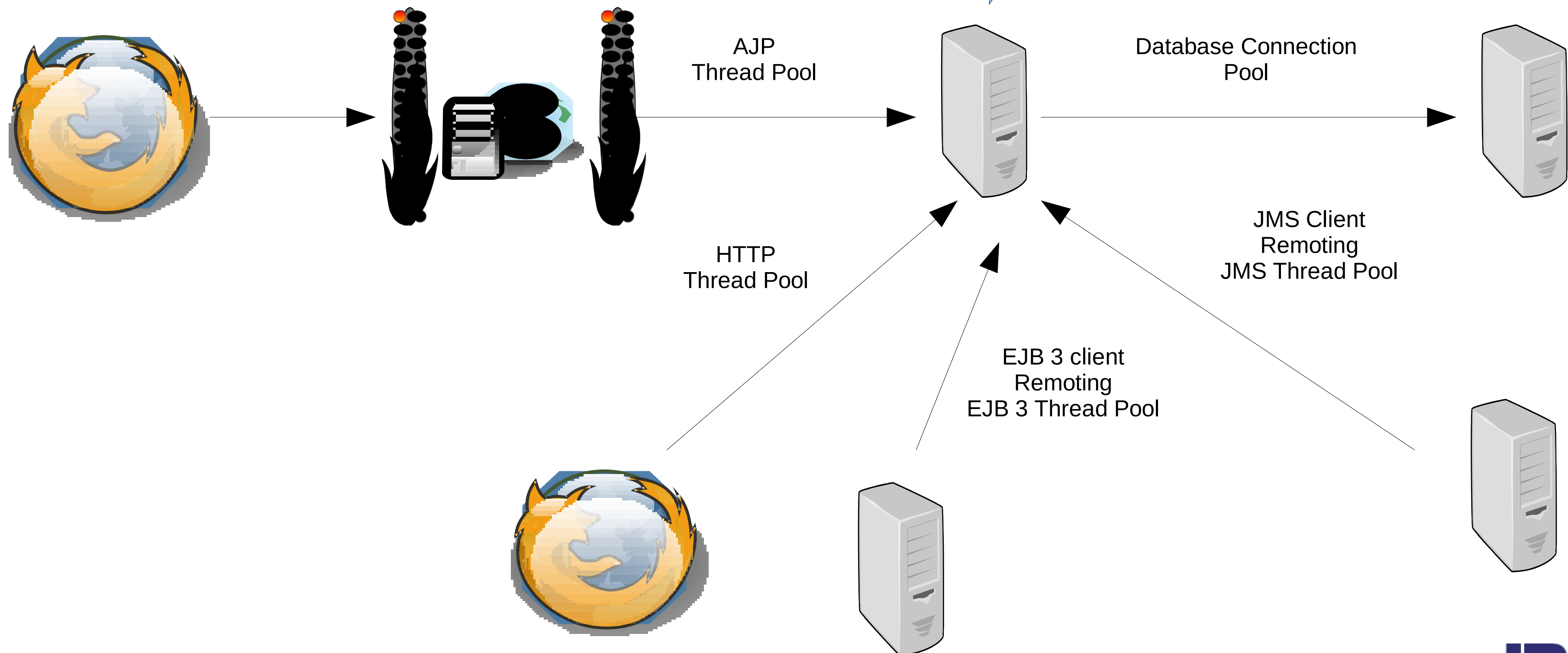
JBoss AS Administration Console 1.3.0.GA (r613) - Powered by Embedded Jopr

© 2002-2009 Red Hat Middleware, LLC. All rights reserved. JBoss is a registered trademark of Red Hat, Inc.

http://192.168.1.22:8080/admin-console/secure/summary.seam?path=-6&conversationId=42&conversationPropagation=end

Thread Pool Relationships

When calls are in JVM they execute on the callers thread pool.



Example of Thread Pools

HTTP Thread Pool

```
<Connector port="8080" address="${jboss.bind.address}"
         maxThreads="250" maxHttpHeaderSize="8192"
         emptySessionPath="true" protocol="HTTP/1.1"
         enableLookups="false" redirectPort="8443" acceptCount="100"
         connectionTimeout="20000" disableUploadTimeout="true" />
```

AJP Thread Pool

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
    <Connector port="8009" address="${jboss.bind.address}" protocol="AJP/1.3"
         emptySessionPath="true" enableLookups="false" redirectPort="8443"
         maxThreads="200" />
```

JCA Thread Pool

```
     <!-- THREAD POOL -->
     <bean name="WorkManagerThreadPool" class="org.jboss.util.threadpool.BasicThreadPool">

        <!-- Expose via JMX -->

  <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name="jboss.jca:service=WorkManagerThreadPool",
  exposedInterface=org.jboss.util.threadpool.BasicThreadPoolMBean.class)</annotation>

        <!-- The name that appears in thread names -->
        <property name="name">WorkManager</property>

        <!-- The maximum amount of work in the queue -->
        <property name="maximumQueueSize">1024</property>

        <!-- The maximum number of active threads -->
        <property name="maximumPoolSize">100</property>

        <!-- How long to keep threads alive after their last work (default one minute) -->
        <property name="keepAliveTime">60000</property>

     </bean>
```

Object and Component Pools

- There are a variety of other pools that are all configurable within the EAP.
  - For EJB 3, there are pools defined in the ejb3-interceptors-aop.xml. You fine this file in <server>/deploy.
    - There are two types of pools for EJB 3, one is called the ThreadLocalPool, and the other is called the StrictMaxPool.
    - The defaults are for Stateless and Stateful Session Beans to use the ThreadLocalPool.
      - The ThreadLocalPool is backed by an InfinitePool, which has no maximum size. Therefore, it grows according to volume in your application.
        - This has the distinct advantage of not needing to be tuned.
      - The defaults for Message Driven Beans is the StrictMaxPool.
        - This pool actually obeys a maximum, will queue up requests when that maximum has been reached, and will time out anything in the queue if there is not an available reference from the pool.
    - These pools can be monitored through the JMX console.

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.
    - Turn off console logging in production.
    - In the EAP, there is a new configuration, called production, and the 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, that generate lots of log data, this can make a real difference.
    - Of course, if your application does spew out log messages all the time, it won't really make a difference.
    - The time to write log messages 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 define cache provider in persistence.xml that you deploy with your EJB 3 application, and you use the @Cache annotation on the beans you want cached. On the @Cached annotation, you specify the usage being one of the following:
      - CacheConncurrencyStrategy.READ_ONLY, READ_WRITE, NONSTRICT_READ_WRITE, or TRANSACTIONAL.
    - You define the cache size and eviction policy in <server>/deploy/cluster/jboss-cache-manager.sar/META-INF/jboss-cache-manager-jboss-beans.xml.
  - **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 xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
   http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
             version="1.0">
    <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.region.factory_class"
                value="org.hibernate.cache.jbc2.JndiMultiplexedJBossCacheRegionFactory"/>
                <property name=”hibernate.cache.region.jbc2.cachefactory"
                          value="java:CacheManager"/>
                <property name="hibernate.cache.use_second_level_cache"
                <property name="hibernate.cache.region.jbc2.cfg.entity" value="mvcc-entity"/>
                <property name="hibernate.cache.region_prefix" value="services"/>
                     ...
        </properties>
    </persistence-unit>
</persistence>
```

There are actually four different cache region factory classes. Besides JndiMultiplexedJBossCacheRegionFactory, there is also MultiplexedJBossCacheRegionFactory, SharedJBossCacheRegionFactory and JndiSharedJBossCacheRegionFactory. You should consult the Hibernate documentation about the differences between the three.

Caching (Cont'd)

- Besides JBoss Cache, there is another form of caching, I would like to discuss, which is prepared statement caching.
  - Many applications can benefit from prepared statement caching.
    - This can be configured in your data source configuration, just like the the pool size we discussed earlier.
    - I have seen applications that experience a 20% improvement in throughput by this simple change.
    - The example data source definition has the parameters in bold.
      - You should do some analysis about how large to make the pool (unique prepared statements you have in your application for that data source).

Linux Specific Tuning

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, as well as OpenJDK, requires the following option, passed on the command-line, to use large pages:

`-XX:+UseLargePages`

The Sun instructions leave it at that and 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 value 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 you system will look and act like it has that amount of memory removed from it!

Large Page Support Example

Example configuration from a server with 8GB of memory and 6GB of memory configured as large page memory.

Page size is 2MB (2048 KB), as shown in `/proc/meminfo`

```
Hugepagessize: 2048KB
```

`/etc/sysctl.conf`

```
kernel.shmmax = 8589934592
vm.hugetlb_shm_group = 501
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)

`/etc/security/limits.conf`

```
jboss soft memlock   6291456
jboss hard memlock   6291456
```

Calculation is as follows:

3072 large pages * 2048 KB page size – 3072*2048 = 6291456

The jboss user is also added 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 EAP, you should see something like this:
  - `HugePages_Rsvd: 1182`
- If you don't see a non-zero value in `/proc/meminfo` for `HugePages_Rsvd`, than you are not using large pages.
  - Besides the log for EAP, you should check `/var/log/messages` for an `avc denied` messages.
    - I have run into SE Linux policies that don't allow applications like the JVM to access the shared memory segment.
- Good luck!

High-Level Database Tuning Tips

- With databases cache is king!
  - Modern databases are extremely efficient at caching data.
    - 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 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.
- Use batch fetching and inserting!
  - Can see order of magnitude improvements on many applications!

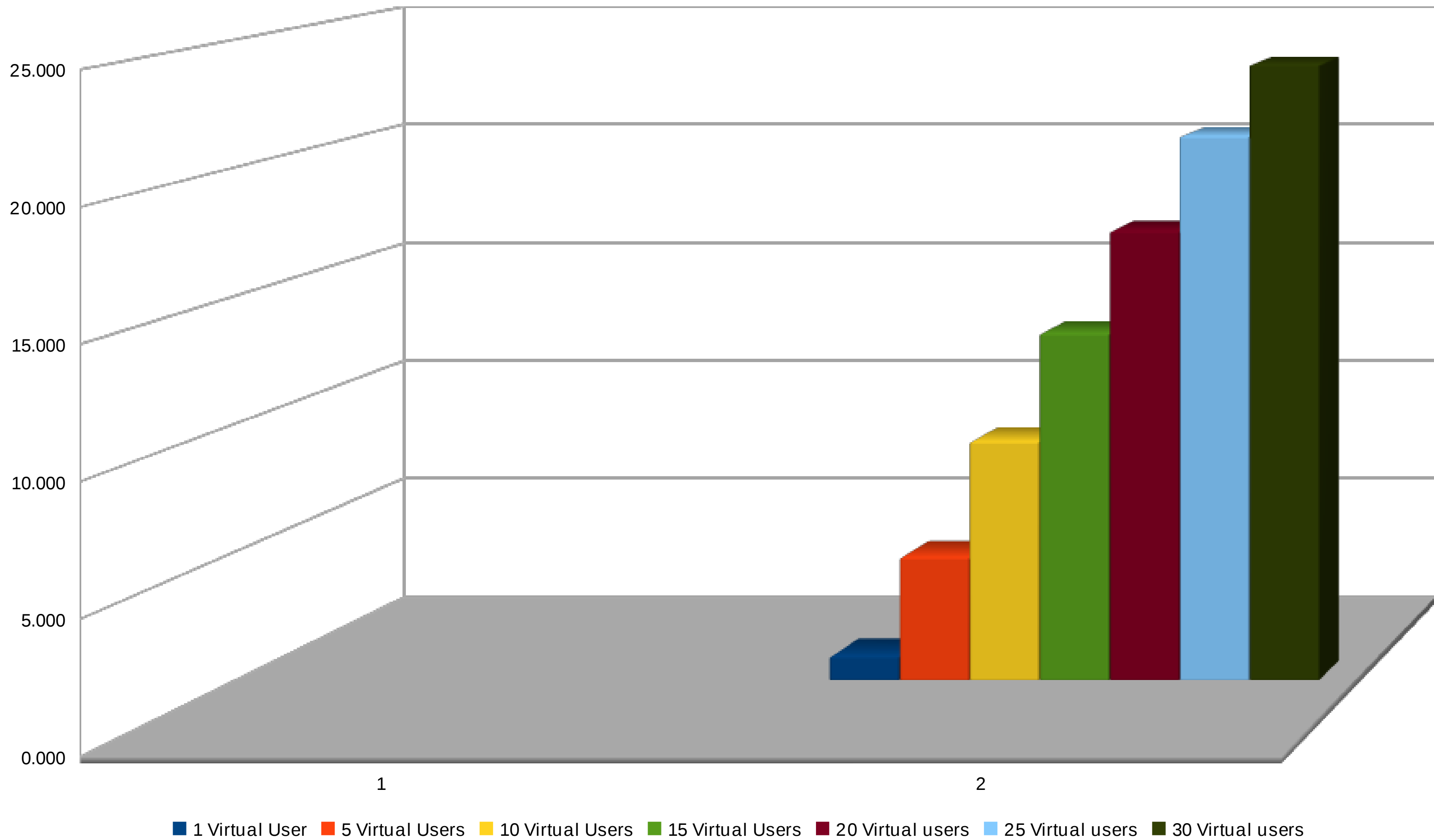
Databases and I/O

- Use DIRECT_IO of you database supports it!
  - With a large cache or a write intensize workload, you should be avoiding double buffering with the file system buffer cache all together.
    - I have seen DIRECT_IO reduce CPU utilization by as much as 70%, and improve throughput dramatically.
- Use asynchronous I/O if your database supports it.
  - As a side note, make sure you have enough threads or processes for reading and writing on your database engine.
    - While asynchronous I/O helps scale, you don't enough processes or threads configured for the workload, you will throttle your application.

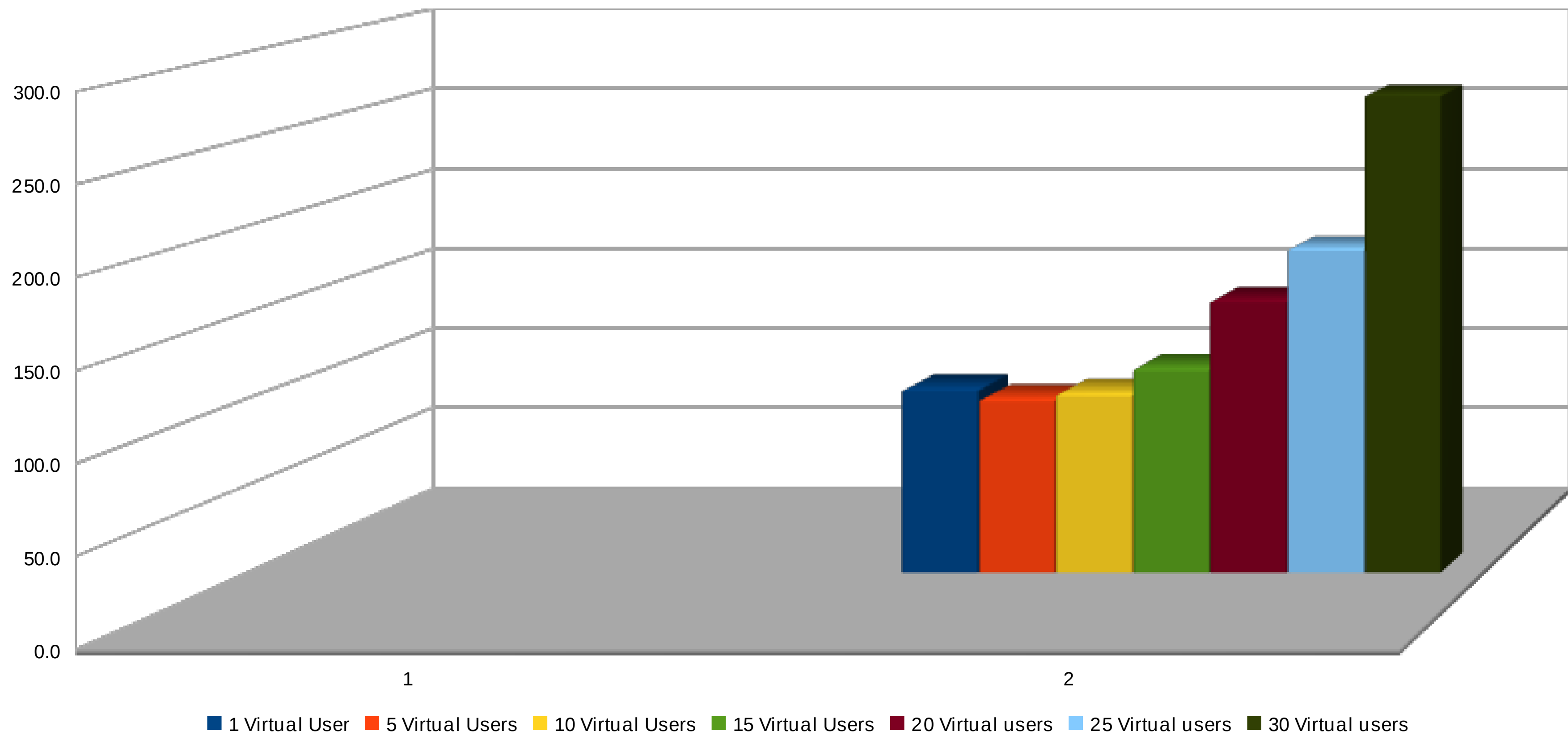
Example Application

- I have an example application that I load tested using the EAP's default configuration (with one minor exception), along with all Linux parameters at their defaults.
  - I load tested 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, (EAP, database and OS) and measured the throughput that I could achieve.
  - Specifically, large page memory support, logging, entity caching and prepared statement caching.
- This application is an EJB 3 application, with two servlets for the UI, stateless and stateful session beans for most of the business logic, and a message driven bean for some asynchronous processing, and entities for the persistence.

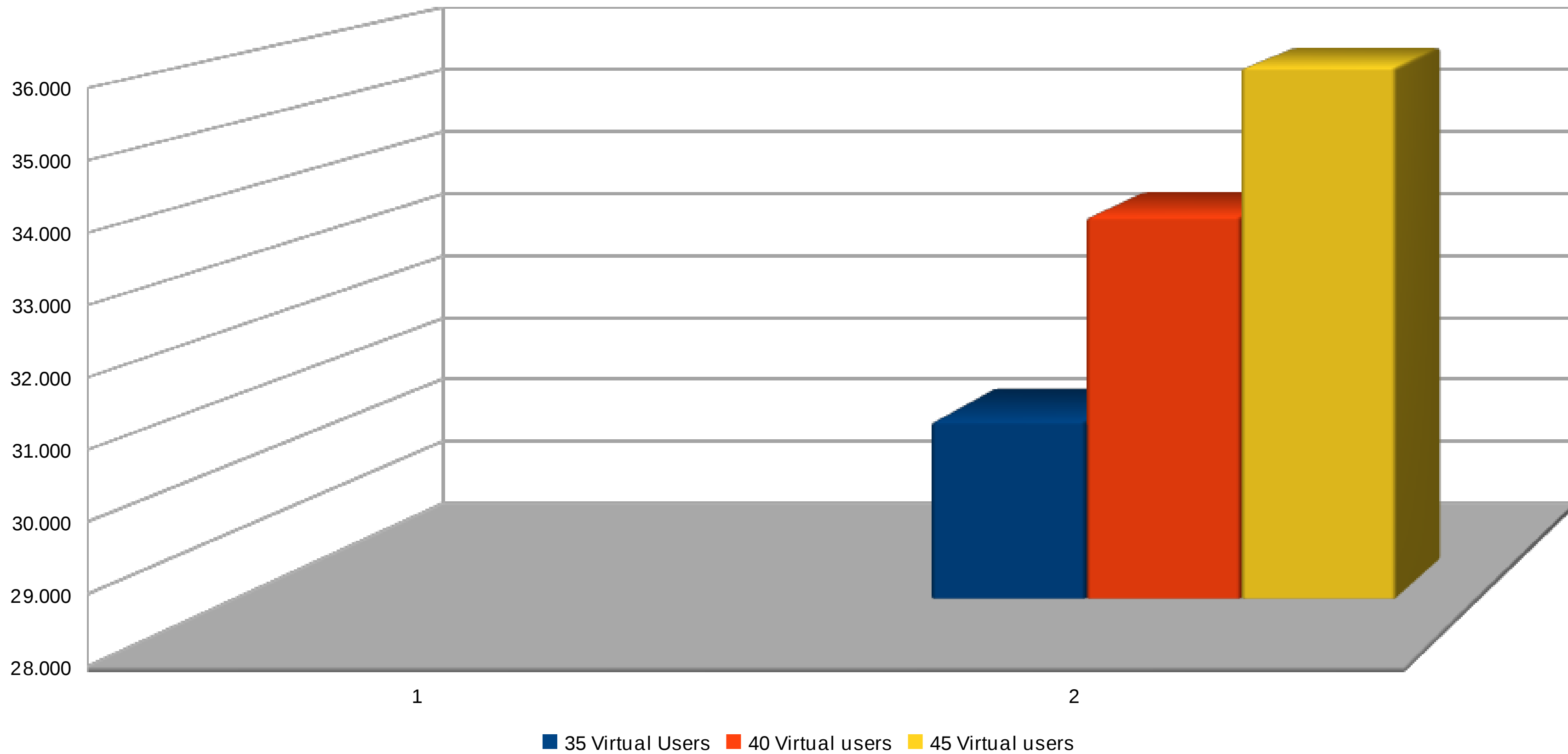
Results!



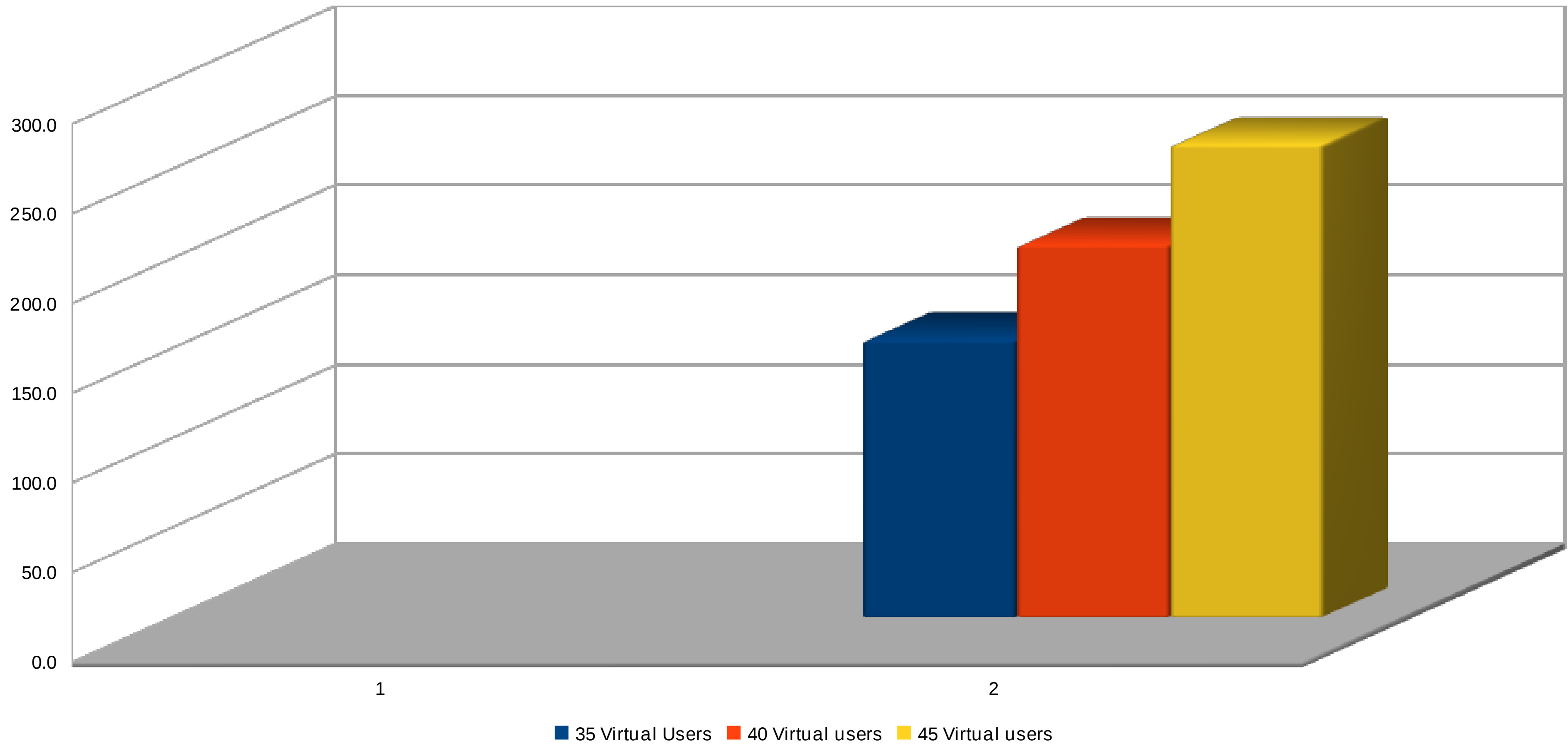
Results!



Results!



Results!



QUESTIONS?

TELL US WHAT YOU THINK:
REDHAT.COM/JBOSSWORLD-SURVEY