Performance Tuning JBoss AS 6
AS Tuning

- Connection pooling.
- Thread pools.
- Object/component pools.
- Logging.
- JVM Tuning.
Connection Pooling

Database connections are expensive to setup and tear down...

- I have seen applications that created new connections with every query or transaction, and then closed that connection.
- You should monitor your connection usage to determine proper sizing.
- You can monitor the connection pool utilization from the admin console as well as with database specific tools.
Example Data Source

<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>
    <prefill>true</prefill>
    <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-isolation>
    <prepared-statement-cache-size>100</prepared-statement-cache-size>
    <share-prepared-statements>true</share-prepared-statements>
  </local-tx-datasource>
</datasources>
Thread Pooling

Thread pools need to be sized appropriately for the workload...

- The httpd thread pool in JBoss Web is defined in server.xml file under <server>/deploy/jboss-web-sar.
  - 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.
- 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

Thread pools need to be sized appropriately for the workload...

- The JCA thread pool is used in conjunction with JMS.
  - This can be configured in `<server>/deploy/jca-jboss-beans.xml`, and is called WorkManager thread pool.

- There is also a TCP thread pool for remote clients.
  - For those remote clients, the thread pool is specified in `<server>/deploy/remoting3-jboss-beans.xml`.
  - There is now only one configuration for remote clients, versus one for each type of component, such as remote JMS clients, remote EJB 3 clients, etc.
Object/Component Pools

There are a variety of other pools that need to be sized...

- For EJB 3, there are pools defined in the ejb3-interceptors-aop.xml. You find 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 defaults for Message Driven Beans is the StrictMaxPool.
  - These pools can be monitored through the JMX console.
The default configuration is appropriate for development, but not for a production environment...

- In the default configuration, console logging is enabled.
- In a production environment, console logging is very expensive.
- Turn down the verbosity level of logging if it's not necessary.

- The less you log, the less I/O will be generated, and the better the overall throughput will be.
Logging Continued

• New JBoss Logging implementation.
  – Configured in deploy directory (not conf), and was designed to meet performance and usability requirements.
  – Most log statements don't have to be wrapped with ifXXXEnabled()...
• Those boolean checks are still there for the cases where the construction of the log statement is expensive.
JVM Tuning

- Use large memory page support (HugeTLB in Linux).
  - Default memory page size is typically 4KB.
  - 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.
  - Besides the system overhead of mapping so many memory pages, large memory pages on Linux cannot be swapped to disk.
JVM Tuning Continued

• Use large memory pages with the 64-bit JVM.
  – Use the 64-bit JVM whenever you have a more than 4GB of memory available to you.
• Large page memory is not available on the 32-bit JVM.
• RHEL does let you allocate large pages on the 32-bit OS, but you get an illegal argument when starting the JVM.

• The Sun JVM, as well as OpenJDK, requires the following option, passed on the command-line, to use large pages:
  – -XX:+UseLargePages
JVM Tuning Continued

- Turn on aggressive optimizations.
  - `-XX:+AggressiveOpts`.
    - This option on the Sun and OpenJDK 1.6 JVM's turns on additional HotSpot optimizations that have yet to be made default.
    - Reduced response times on my test workload by 8%.

- Use Escape Analysis with an EJB 3 workload.
  - `-XX:+DoEscapeAnalysis`.
    - Not a general optimization that I would rely on for anything outside of the EAP without testing it first.
      - Escape Analysis determines if objects are accessed in multiple threads or a single thread. If a single thread, then it “escapes” the object making it local.
      - This has the affect of reducing locking in many cases (improving multi-core throughput), and reducing garbage collection overhead.
      - Reduced response times by 40% on my test workload!!!
JVM Tuning Continued

• So, what about the 32-bit vs. 64-bit JVM?
  – So, I did some simple workload testing to compare the two.
  – Nehalem 8 core system with 16 GB's of RAM, and Hyper-threading turned on.
    • Looks like a 16 CPU system to the OS.
NUMA

• The latest x86_64 systems from Intel, and all the AMD systems are NUMA.
• JVM supports NUMA.
  – -XX:+UseNUMA
• On Linux start the JVM via numactl.
  – Lots of options to pin the JVM process to specific NUMA nodes, etc.
  – We have seen improvements in experiments.