JUDCon

JBoss Users & Developers Conference

Boston:2011

Optimizing Performance with JBoss Application Server 7

John Bailey
Red Hat Core Developer for JBoss Application Server

Agenda

- Aspects of Performance
- AS7 Architecture Background
- Application Server Trimming
- Thread Pool Tuning
- Connection Pools Tuning
- Questions / Comments

Aspects of Performance

- Boot time - time it takes to get up and running
  - Increases developer productivity
  - Improves on-demand scaling in cloud environments
  - Reduces cost in pay-for-usage environments
- Throughput - how much can be done at a given time
  - Increased throughput can reduce the number of instances required to run a workload
  - Can reduce the amount of time to run a workload through concurrency

Aspects of Performance cont.

- Memory footprint – how much memory does the system require to run
  - Leads to lower hardware costs
  - Decreases time required for garbage collection
- Disk space – how much disk space is required
  - Leads to lower hardware costs
  - Reduces the amount of time needed to provision servers by requiring less bandwidth to get media in place

AS7 Architecture

- Redesigned from the ground up with performance as a first class design goal
- Based on the MSC (Modular Service Container) – advanced dependency management system
- Utilizes a modular classloading system
- Extensible management of application server core facilities
- Centralized configuration – limited to a small number of configuration files

SHow of hands. How many know about AS7

MSC and Performance

- Supports multiple service modes allowing services to start and stop immediately, on-demand or lazily
- Service life-cycles are processed in parallel whenever possible
- Proper service definitions and creation will lead to improved performance out of the box by dynamically tuning required services
- Capable of managing extreme numbers of services with a linear performance cost

JBoss Modules and Performance

- No more flat classloader!
- Relies on fine-grained inter-module dependencies
- Capable of supporting complex module graphs with little overhead
- Modules are loaded on-demand and un-loaded when no longer needed
- Only JAR files which are currently in use will occupy runtime memory

AS Extensions

- Application server building blocks
- Backed by one or more modules
- Enable additional functionality to be loaded into the AS
- Provide custom schema support for enhancing the configuration files

Subsystems

- Configuration for a specific aspect of the application server (transactions, logging, security, etc)
- Utilize a subsystem specific XML schema
- Responsible for adding required services into the MSC
- Invoked by the application server during the bootstrap process

Profiles

- Grouping of related subsystems
- Can be extended to provide an inherited set of subsystems
  - eg. <profile name="default"><include profile="web"/></profile>
- Applied to a server group to establish a base set of services for a server or group of servers

Example Configuration

${jboss-server-root}/domain/configuration/domain

```
...
  <extensions>
    <extension module="org.jboss.as.logging"/>
     ...
  </extensions>
  <profiles>
    <profile name="default">
      <subsystem xmlns="urn:jboss:domain:logging:1.0">
        <console-handler name="CONSOLE" autoflush="true">
          <level name="INFO"/>
          <formatter>
            <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%E%n"/>
          </formatter>
        </console-handler>
        ...
      </subsystem>
      ...
    </profile>
    ...
</profiles>
...
```

Application Server Trimming

- Goal #1: Reduce the number of running services
- Goal #2: Reduce the amount of configuration
- Goal #3: Reduce the number of loaded extensions
- Goal #4: Reduce the number of modules

Reduce Running Services

- Reduce boot time by not starting unneeded services
- Reduce memory footprint by eliminating memory used by unneeded services
- Reduce the amount of configuration
- Accomplished by disabling unneeded configuration
- Can be isolated to configuration within a subsystem or a whole subsystem
- Will cause runtime dependency errors if services are removed which are depended on by other services

Reduce the Amount of Configuration

- Reduce boot time by eliminating the need for additional configuration parsing
- Reduce memory footprint by eliminating additional configuration maintained in-memory by the server
- Eliminate the need to load extensions when all services and configuration provided by an extension are no longer needed

Reduce the Number of Loaded Extensions

- Reduce boot time and memory footprint
  - No longer registering additional schemas
  - No longer registering additional subsystem configuration handlers
- Reduce the number of modules required

Example: Reduce Services

1. **Remove the unneeded datasources**

```
  <extension module="org.jboss.as.connector"/>
...
  <subsystem xmlns="urn:jboss:domain:datasources:1.0">
    <datasources>
      <datasource jndi-name="java:/H2DS" pool-name="H2DS" enabled="false"
          use-java-context="true">
        <connection-url>dbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
        <driver-class>org.h2.Driver</driver-class>
        <module>org.h2.Driver#1.2</module>
        <pool>
          <prefill>true</prefill>
          <use-strict-min>false</use-strict-min>
        </pool>
        <security>
          <user-name>sa</user-name>
          <password>sa</password>
        </security>
      </datasource>
    </datasources>
    <drivers>
      <driver module="com.h2database.h2"/>
    </drivers>
  </subsystem>
```

Example: Reduce Services

2. **Remove the unneeded drivers**

```
<extension module="org.jboss.as.connector"/>
...
<subsystem xmlns="urn:jboss:domain:datasources:1.0">
  <drivers>
    <driver module="com.h2database.h2"/>
  </drivers>
</subsystem>
```

Example: Reduce Services

3. **Remove the unneeded subsystem**

```
<extension module="org.jboss.as.connector"/>
...
<subsystem xmlns="urn:jboss:domain:datasources:1.0">
  <datasources>
  </datasources>
</subsystem>
```

Example: Reduce Services

4. **Remove the unneeded extension**

```
<extension module="org.jboss.as.connector"/>
```

Demo Removing Services

Custom Profiles

- Preferable to removing existing subsystem configurations
- Custom built to include only the subsystem configurations needed
- Reusable configurations that can simplify the configuration of a new server
- Can be targeted to support a specific set of requirements

Example: Basic Web Profile

- Contains only the most basic subsystems used by web
- Can be easily applied to a new server to greatly reduce what the server starts up
- Additional services such as EJB can be enabled by adding the additional subsystem config

```
<profile name="web">
   <subsystem xmlns="urn:jboss:domain:logging:1.0">
      ...
   </subsystem>
   <subsystem xmlns="urn:jboss:domain:ee:1.0"/>
   <subsystem xmlns="urn:jboss:domain:naming:1.0"/>
   <subsystem xmlns="urn:jboss:domain:web:1.0">
      <connector name="http" protocol="http"
         socket-binding="http" scheme="http"/>
      <virtual-server name="localhost">
         <alias name="example.com"/>
      </virtual-server>
   </subsystem>
</profile>
```

Demo Custom Profile

Questions on AS Trimming?

Thread Pool Tuning

- Thread pools are primarily used to increase or decrease the number of concurrent tasks executing on an application server
- Gained throughput for an application server can be obtained by properly controlling the concurrent execution of tasks
- Thread pools reduce the cost associated with creating threads
- Thread pools have a number of tuning parameters which allow the thread pool to achieve desired performance characteristics

Thread Pool Types

- Unbounded Queue
  - Has a core and maximum size
  - Will create new threads until the core size is reached
  - Will queue tasks beyond the core size
- Bounded Queue
  - Core and maximum size and a specified queue length
  - Will create new threads until the core size is reached
  - Will queue tasks beyond the core size until queue length is reached

Thread Pool Types, cont.

- Queueless
  - No queue, but still maintains a maximum number of threads
    - Will create a new thread up until the max size is hit and then will either block or fail
- Scheduled
  - Has a max size
  - Allows tasks to be submitted on a scheduled basis

Thread Pool Attributes

- max-threads - The maximum threads this pool will have in use at any given time
- queue-length - The queue length for a bounded queue thread pool
- core-threads - The default number of threads to keep in a bounded queue pool to execute tasks
- keepalive-time - The amount of time to keep an unused thread alive in the pool before destroying it
- blocking - Determines whether the pool will wait for a thread to be returned to the pool when a thread is

Scaled Count

- All thread pool size attributes are configured as scaled counts
- Uses a base size and a per-CPU size to determine the actual size
- The actual size is determine by taking the count attribute and adding it to the per-cpu attribute times the number of CPUs in the system

Eg. <max-threads count="10" per-cpu="20"/>
For a two CPU system, the actual count would be 50

Tuning max-threads

Eg. <max-threads count="10" per-cpu="20"/>

- Sized using a scaled count
- The max threads count should be tuned when you want to limit the number of concurrent tasks executing
- A max threads count that is too low will result in tasks either failing to execute or blocking waiting for a thread, resulting in reduced throughput
- A max threads count that is too high will allow too many tasks to run currently and possible exhaust other resources (db connections, filesystem handles, etc)

Tuning core-threads

Eg. <core-threads count="10" per-cpu="20"/>

- Sized using a scaled count
- Represents the minimum pool size
- The core threads count should be tuned when you have a good idea of the typical number of concurrently executing tasks
- A core threads count that is too low will result in somewhat reduced concurrency
- A core threads count that is too high will keep unnecessary, idle threads in memory at a give time (wastes memory, adds overhead)

Tuning Queue Length

Eg. <queue-length count="10" per-cpu="20"/>

- Sized using a scaled count
- Represents the number of tasks that can be queued while waiting for a core thread
- A queue size that is too low will cause a unnecessary number of tasks blocking or failing to execute
- A queue size too large will cause delays in the task execution and will not be maintained by the blocking characteristics

Tuning keepalive-time

Eg. <keepalive-time time="10" unit="SECONDS"/>

- The keep alive parameter should be tuned to help keep threads alive as needed based on current work-loads
- A keep alive time that is too low will cause threads to be destroyed earlier and possible miss the opportunity to reuse a thread for the next task
- A keep alive time that is too high will keep threads open longer than necessary and can possible keep the pool full of unused threads

Thread Pool Example

- Example uses a queue to hold onto tasks when core threads are not available
- The core threads will be 5 on a dual core machine
- The queue length will be 50 on a dual core machine
- The max threads will be 50 on a dual core machine
- The pool will hold onto a thread for 10 seconds once not in use

```
<bounded-queue-thread-pool name="jca-short-running"
                           blocking="true" allow-core-timeout="false">
  <core-threads count="1" per-cpu="2"/>
  <queue-length count="10" per-cpu="20"/>
  <max-threads count="10" per-cpu="20"/>
  <keepalive-time time="10" unit="SECONDS"/>
</bounded-queue-thread-pool>
```

Questions on Thread Pool Tuning

Connection Pool Tuning

- Used to control the number of active connections to a database
- Proper configuration of a connection pool can increase application server throughput
- Can reduce the time it takes for an application to gain access to a database
- Can also restrict the number of active connections to a database

Connection Pool Parameters

- min-pool-size - The minimum number of connections to keep in the pool
- max-pool-size - The maximum number of connections to keep in the pool
- prefill- Whether the pool should be pre-filled with the minimum number of connections
- use-strict-min - Whether idle connections below the min-pool-size should be close

Connection Pool Parameters cont.

- blocking-timeout-millis - the maximum time in milliseconds to block while waiting for a connection before throwing an exception
- idle-timeout-minutes - maximum time in minutes a connection may be idle before being closed
- allocation-retry - the number of times that allocating a connection should be tried before throwing an exception
- allocation-retry-wait-millis - time in milliseconds to wait between retrying to allocate a connection

Tuning min-pool-size

- Controls the minimum number of connections managed by the pool
- A minimum size too low can cause increased connection acquisition time for applications
- A minimum size that is too high will hold unneeded connections and waist resources on both the application server and the database server

Tuning max-pool-size

- Controls the maximum number of connections managed by the pool
- A maximum size too low can cause callers to block or receive and exception while waiting for a connection to be available
- A maximum size that is too high will can cause the number of connections to overrun the available resources on the application or database server

Tuning prefill and use-strict-min

- Enable prefill if the pool should be filled with the minimum number of connections upon creation
- Prefilling can reduce the time it takes for the initial requesters to get a connection
- Enable use-strict-min if the pool should never drop below the minimum number of connections
- Use strict minimum can reduce the time it takes to get a connection after an idle period
- Both prefill and use-strict-min can cause unneeded connections to be maintained

Tuning blocking-timeout-millis

- Lower blocking timeout can cause a requests to fail more frequently, but can give additional control back to the application to maintain a higher responsiveness under heavy load
- A higher blocking timeout can allow more requests to succeed, but cause the application to have lower responsiveness when a high number of requests start blocking

Tuning idle-timeout-minutes

- The amount of time a connection is allowed to be idle before being closed and removed from the pool
- A longer idle timeout will allow less re-connections by keeping more connections alive
- A shorter idle timeout will help reduce the number of application and database server resources in use at a given time

Tuning allocation-retry

- Number of times that allocating a connection should be tried before throwing an exception.
- A higher number of allocation retries will help reduce the number of connection failures if short outages occur
- A lower number of allocations will give control back to the application faster when failures occur

Tuning allocation-retry-wait-millis

- Time in milliseconds to wait between retrying to allocate a connection
- A longer retry wait will reduce the number of attempts to retry when a outage occurs, but will cause a longer connection time for short outages
- A shorter wait time will acquire a connection faster when an outage is resolved, but will use more resources attempting to connect

Connection Pool Example

- Example datasource that will maintain at least one connection throughout its lifetime
- The datasource will not allow more than ten concurrent connections
- If the pool is exhausted requesters will block for up to 30 seconds
- Each connection can remain idle in the pool for up to 15 minutes
- A connection failure will be retried up to two times 5 seconds apart

```
<datasource jndi-name="java:/H2DS" pool-name="H2DS">
  <pool>
    <min-pool-size>1</min-pool-size>
    <max-pool-size>10</max-pool-size>
    <prefill>true</prefill>
    <use-strict-min>false</use-strict-min>
  </pool>
  <timeout>
    <blocking-timeout-millis> 30000</blocking-timeout-millis>
    <idle-timeout-minutes>15</idle-timeout-minutes>
    <allocation-retry>2</allocation-retry>
    <allocation-retry-wait-millis>5000</allocation-retry-wait-millis>
  </timeout>
</datasource>
```

Questions on Connection Pool Tuning

Thanks!