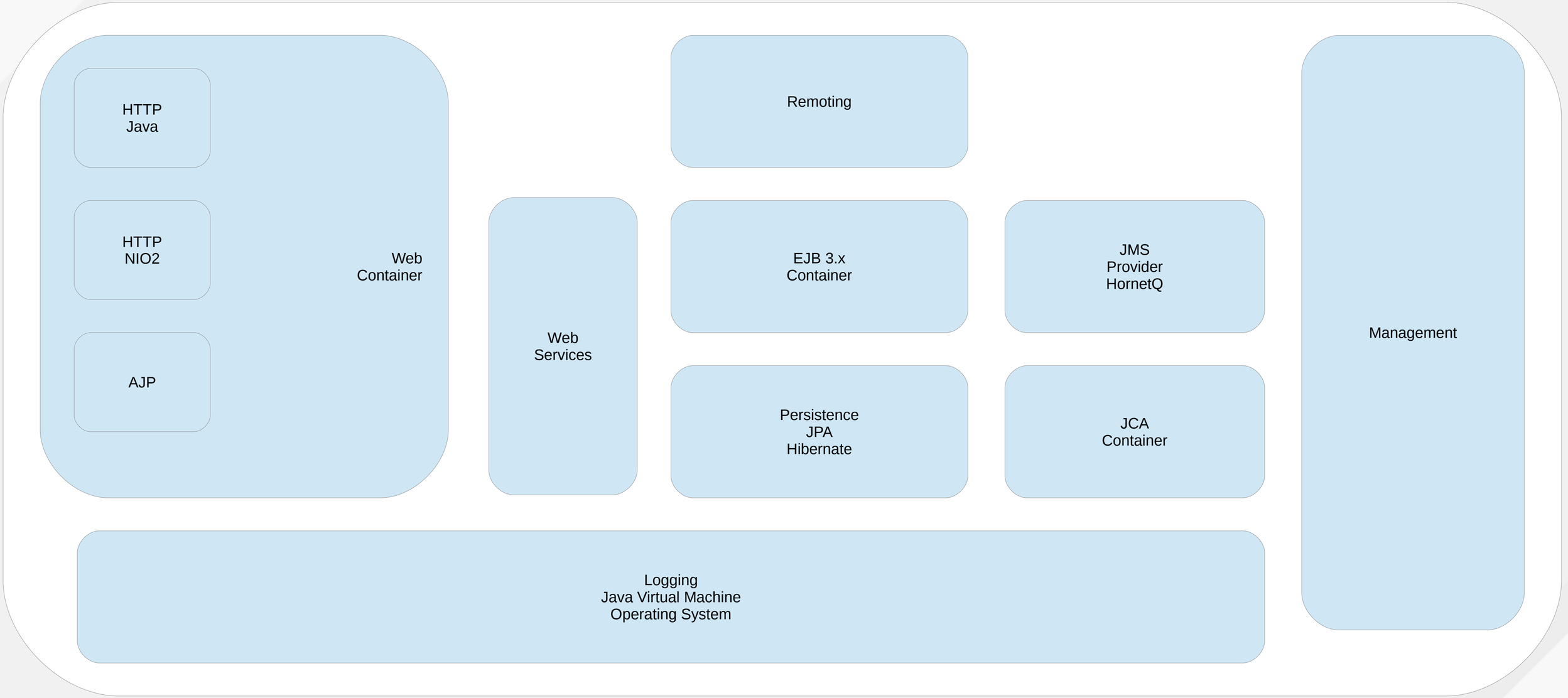
BOSTON, MA
JUNE 23-26, 2015

Performance Tuning Red Hat Enterprise Application Platform

Andrig T. Miller
Global Platform Director
June 26, 2015

redhat.

EAP 6.4 Specific Tuning


HTTP
Java
HTTP
NIO2
AJP
Web
Container
Remoting
Web
Services
EJB 3.x
Container
JMS
Provider
HornetQ
Persistence
JPA
Hibernate
JCA
Container
Management
Logging
Java Virtual Machine
Operating System

JBoss Web – Connectors/Thread Pools

- Three different use cases determine which web connector to use.
  - Low number of connections (i.e. in the hundreds) with high concurrency.
    - Use the Java I/O connector, which is synchronous blocking I/O based.
  - High number of connections (i.e. in the thousands, tens of thousands or even hundreds of thousands).
    - Use the NIO2 based connector (available since EAP 6.1), which is asynchronous non-blocking I/O based.
      - Requires Java 7.
  - Application server fronted by Apache HTTPD server.
    - Use AJP connector.
  - For all connectors use a defined executor, and never use the default thread pool.
  - Set the max connections on your connector configuration.
  - Set the protocol parameter to enable the NIO2 based connector.
  - Set up multiple connectors if your deployment or deployments have multiple http based points of connection.
    - For example, an application or applications deployed that have a web front end, but also a web service based, whether JAX-WS and/or JAX-RS, API.
    - Having multiple connectors defined will allow you to configure specific thread pools for each connector, and fine tune the resources consumed.

JBoss Web – Connector/Thread Pool Example

```
<subsystem xmlns="urn:jboss:domain:threads:1.1">
    <unbounded-queue-thread-pool name="JBossWeb">
        <max-threads count="62"/>
        <keepalive-time time="75" unit="minutes"/>
    </unbounded-queue-thread-pool>
    <unbounded-queue-thread-pool name="JBossWebWs">
        <max-threads count="38"/>
        <keepalive-time time="75" unit="minutes"/>
    </unbounded-queue-thread-pool>
</subsystem>
…

<subsystem xmlns="urn:jboss:domain:web:2.2" default-virtual-server="default-host" native="false">
    <configuration>
        <jsp-configuration trim-spaces="true" smap="false" generate-strings-as-char-arrays="true"/>
     </configuration>
     <connector name="http" protocol="org.apache.coyote.http11.Http11NioProtocol" scheme="http" socket-
binding="http" executor="JBossWeb" max-connections="56096"/>
     <connector name="httpWs" protocol="org.apache.coyote.http11.Http11NioProtocol" scheme="http" socket-
binding="httpWs" executor="JBossWebWs" max-connections="10240"/>
     <connector name="ajp" protocol="AJP/1.3" scheme="http" socket-binding="ajp"/>
     <virtual-server name="default-host" enable-welcome-root="true">
         <alias name="benchserver2"/>
         <alias name="benchserver2G1"/>
     </virtual-server>
</subsystem>
```

JBoss Web – Connector/Thread Pool Example (Cont'd)

```
 <socket-binding-group name="standard-sockets" default-interface="public" port-offset="$
{jboss.socket.binding.port-offset:0}">
        <socket-binding name="management-native" interface="management" port="$
{jboss.management.native.port:9999}"/>
        <socket-binding name="management-http" interface="management" port="$
{jboss.management.http.port:9990}"/>
        <socket-binding name="management-https" interface="management" port="$
{jboss.management.https.port:9443}"/>
        <socket-binding name="ajp" port="8009"/>
        <socket-binding name="http" port="8080"/>
        <socket-binding name="https" port="8443"/>
        <socket-binding name="jacorb" interface="unsecure" port="3528"/>
        <socket-binding name="jacorb-ssl" interface="unsecure" port="3529"/>
        <socket-binding name="httpWs" port="8081"/>
        <socket-binding name="httpsWs" port="8444"/>
...
```

JBoss Web - JSP

- If you use JSP, we have found the following JSP configuration parameters to optimize memory allocation, network usage and throughput.
  - trim-spaces="true" – tells the container to remove useless spaces from the response.
  - smap="false" - tells the container to no longer generate JSR 045 SMAP files (SMAP files enables better stack traces for debugging).
    - This can be useful to have on during development, but shouldn't be necessary in production environments, unless you are having trouble debugging something in production.
      - You can always turn it back on if needed.
  - generate-strings-as-char-arrays="true" – tells the container to use char arrays when it generates strings.
- The performance team has also enhanced Jasper for the use of scriplets. If you use, or have used, scriptlets, you can set a property that will turn on our optimizations, and it will improve the memory allocation for this use case too.
  - org.apache.jasper.compiler.Parser.OPTIMIZE_SCRIPTLETS=”true”

JBoss Web – JSP Example

```
<system-properties>
    <property name=”org.apache.jasper.compiler.Parser.OPTIMIZE_SCRIPTLETS” value=”true”/>
    ...
</system-properties>
```

```
<subsystem xmlns="urn:jboss:domain:web:2.2" default-virtual-server="default-host"
native="false">
            <configuration>
                <jsp-configuration trim-spaces="true" smap="false" generate-strings-as-
char-arrays="true"/>
            </configuration>

…
```

Web Services

- Web service calls, specifically JAX-WS calls, will be handled by the web container thread pool.
  - As in the previous example, its best to define a specific connector for your web service calls.
    - This will allow you to tune this in isolation of other web requests (e.g. JSP, JSF, etc.).
    - For @OneWay annotated web services there is an internal thread pool, but we have found that its better to use the calling thread for this instead of configuring another thread pool.
      - By using the calling thread you eliminate the hand-off from one thread pool to another.
      - You also eliminate the context switch by using another thread instead of the calling thread.
      - You enable this behavior through a CXF property called USE_ORIGINAL_THREAD.
    - For all other web services, message persistence is important, as any stream larger than 64k will be written to disk.
      - You configure this through a property org.apache.cxf.io.CachedOutputStream.Threshold
        - So, if you have a large stream, larger than 64k, you can prevent this from being cached in a file, and keep it in memory.
        - Of course the trade-off is memory used for this, and you may not want something that large to stay in memory. If you have plenty of memory though, it will be faster.
    - Finally, there are CXF bus strategies for servicing clients that should be considered.
      - In EAP, the TCCL_BUS (Thread Context Class Loader) bounds the number of buses instantiated to the number of applications deployed that have web services, limiting the memory footprint.
        - This has proved to be the best strategy for performance, especially latency, of JAX-WS requests.

Web Services - Example

```
<subsystem xmlns="urn:jboss:domain:webservices:1.2">
            ...
             <endpoint-config name="Standard-Endpoint-Config">
                 <property
name="org.apache.cxf.interceptor.OneWayProcessorInterceptor.USE_ORIGINAL_THREAD"
value="true"/>
             </endpoint-config>
             <endpoint-config name="Recording-Endpoint-Config">
                 <pre-handler-chain name="recording-handlers" protocol-
bindings="##SOAP11_HTTP ##SOAP11_HTTP_MTOM ##SOAP12_HTTP ##SOAP12_HTTP_MTOM">
                     <handler name="RecordingHandler"
class="org.jboss.ws.common.invocation.RecordingServerHandler"/>
                 </pre-handler-chain>
             </endpoint-config>
             <client-config name="Standard-Client-Config"/>
         </subsystem>

<system-properties>
    <!-- CXF message handling –>
    <property name=”org.apache.cxf.io.CachedOutputStream.Threshold” value=”4096000"/>
    <property name=”org.jboss.ws.cxf.jaxws-client.bus.strategy” value=”TCCL_BUS”/>
</system-properties>
```

EJB 3.x Container

- The EJB container has changed much since 6.0, but in 6.4, we changed the implementation of the stateless session bean and message driven bean pool, to improve concurrency.
  - There is nothing you have to do to get this new pool implementation, as it replaced the existing strict max pool, so configuration is the same as before.
- Key configuration parameters are:
  - maxPoolSize (bean instance pools – MDB/SLSB).
    - InstanceAcquisitionTimeout.
  - ChannelCreationOptions.
    - WORKER_READ_THREADS, WORKER_WRITE_THREADS, MAX_INBOUND_MESSAGES, MAX_OUTBOUND_MESSAGES.
  - Thread pool.
    - Max-threads.
    - Keepalive-time.
  - In-vm-remote-interface-invocation.

EJB 3.x – Container Example

```
<pools>
     <bean-instance-pools>
         <strict-max-pool name="slsb-strict-max-pool" max-pool-size="1300" instance-acquisition-timeout="1"
instance-acquisition-timeout-unit="MILLISECONDS"/>
         <strict-max-pool name="mdb-strict-max-pool" max-pool-size="180" instance-acquisition-timeout="1"
instance-acquisition-timeout-unit="MILLISECONDS"/>
         </bean-instance-pools>
 </pools>
...
<remote connector-ref="remoting-connector" thread-pool-name="default">
    <channel-creation-options>
        <option name="WORKER_READ_THREADS" value="1" type="xnio"/>
        <option name="WORKER_WRITE_THREADS" value="1" type="xnio"/>
        <option name="MAX_INBOUND_MESSAGES" value="165" type="remoting"/>
        <option name="MAX_OUTBOUND_MESSAGES" value="165" type="remoting"/>
        </channel-creation-options>
</remote>
...
<thread-pools>
    <thread-pool name="default">
        <max-threads count="165"/>
        <keepalive-time time="75" unit="minutes"/>
    </thread-pool>
</thread-pools>
...
<in-vm-remote-interface-invocation pass-by-value="false"/>
```

EJB 3.x Container (Cont'd)

- Important notes:
  - Default maxSession for Message Driven Beans (MDB), so regardless of how large you set the pool size, only 15 will execute concurrently, unless the maxSession is changed.
    - Of course, that is per MDB, so you could have a pool that is larger, and encompasses all the MDB's in the application, each only needing 15 or less to run concurrently.
  - The pool for stateless session beans needs to be sized, based on the number of stateless session beans in the application, and the concurrency rate in which those beans are invoked.
    - e.g. Your application has 10 unique stateless session beans, and all 10 are invoked at the same rate, and that rate is 10 per second, and the response times of those invocations are 1 second each.
      - This yields a pool size of at least 100.
      - Decreases in response times, or increases in concurrency is what drives the size.
    - One other optimization, is you can define independent pools for different stateless session bean.
      - You do this through the use of annotations on the beans (or alternatively ejb-jar.xml), and then define those pools in the XML.

EJB 3.x – Separate SSB Pool Example

```
<pools>
     <bean-instance-pools>
         <strict-max-pool name="CustomerManagerPool" max-pool-size="300" instance-
acquisition-timeout="1" instance-acquisition-timeout-unit="MILLISECONDS"/>
         <strict-max-pool name=”ProductManagerPool” max-pool-size=”500” instance-
acquisition-timeout=”1” instance-acquisition-timeout=unit=”MILLISECONDS”/>
         </bean-instance-pools>
 </pools>
...


@Stateless
@Pool(value="CustomerManagerPool")
public class CustomerManagerBean implements CustomerManager {

…

}
```

Persistence / JPA

- In the latest version of EAP 6, 6.4, we have made some internal improvements to Hibernate, along with an improved feature and a new feature that applies to certain use case.
  - Most of the changes are internal, but there is an enhanced feature and a new feature.
  - Key topics for persistence:
    - Bytecode enhanced entities.
    - Second-level cache (based on Infinispan since EAP 6.0):
      - Entity Caching.
        - Read only immutable entity optimization.
      - Query Caching.
    - Batching.
    - Fetch sizes.
    - Batch inserts.
    - Reflection optimization.

Persistence / JPA – Bytecode Enhanced Entities

- Bytecode enhanced entities.
  - These are entities that have been modified at the bytecode level.
    - These bytecode changes enable more efficient processing, because the entity itself can monitor its state.
    - Primary use case, today, is dirty checking of entities (will be expanded to more capabilities in the future – EAP 7).
    - Today it works at build time, NOT at deployment time.
      - There are Ant, Maven and Gradle tasks in Hibernate to assist in building bytecode enhanced entities.
      - There is no configuration parameter.
        - Hibernate will detect at class load time that the entity has been enhanced and behave accordingly.
    - Using bytecode enhanced entities will reduce the CPU consumption of Hibernate and enhance scalability.
    - One word of caution with bytecode enhanced entities.
      - If you entity has mutable fields that are modified outside the mutator methods of the bean itself, bytecode enhanced entities will not detect that the state has changed of the bean, and that mutation will not be persisted.
      - Hibernate's default dirty checking, while expensive, does compare all fields, so can detect a change to an entity, even though a method of that entity did not make the change.

Persistence / JPA – Bytecode Enhanced Entities Example

```
…

@Entity
public class Order implements Serializable {

   ...
   private Date orderDate;

     ...

     public Date getOrderDate() {

         return orderDate;          ←-  Should be return orderDate.clone();

     }

     ...



}
```

Persistence / JPA – Entity Cache

- Entity Cache Keys:
  - Read/Write Ratio.
    - Mostly read, with very little writes (inserts and/or updates).
      - Would mostly recommend caching read only entities (read and delete).
      - If your read only entities are also immutable, there is an additional optimization that you can configure.
  - Query type.
    - entitymanager.find(class, pk).
    - Cacheable query.
      - A cacheable query is one that always returns the exact same result.
  - Cache Concurrency Strategy.
    - READ_ONLY.
      - The read only strategy applies to entities that are only read, or read and inserted, but not updated.
        - When you have read only entities, there is a new additional optimization that can be employed.
    - TRANSACTIONAL.
      - The transactional strategy is required if there are updates to the entity being cached.
  - Internal Infinispan optimization.
    - There is a setting that allows Infinispan to use the more efficient JDK 8 of the ConcurrentHashMap while running on JDK 6 or 7.
  - Data Size.
    - There is only so much heap space to play with, and extremely large sets of entities may suffer from low cache hit rates just because of the number of entities involved.
  - Access Pattern.
    - You may have a large set of entities, but if the access pattern is such that a small subset of them are accessed very often, you may still derive benefits from caching them.

Persistence / JPA – Read Only Immutable Entity Cache

- When you have a read only immutable entity, there is a new feature you can take advantage.
  - This new feature has some caveats.
    - First, the entity cannot have any associations to other entities.
    - Second, it must be read only (obviously).
    - Third, it must be immutable, which may sound obvious, but its not because of the way Hibernate cheats for developers.
      - Hibernate, when doing dirty checking does some interesting work, where if you have a mutable object, like if your entity has a Date field.
        - In this case, it is possible for the Date object to be modified outside the control of the entity. Hibernate's default dirty checking will detect this because it does a comparison of all fields, and will automatically generate the update that is needed for the entity.
        - So, immutability needs to be guaranteed by the way the entity is designed (we saw this in the bytecode enhanced entity example).
          - In our example using Date, you would want to make sure that any accessor method that can grab Date (or any mutable object) is actually given a copy of the object, and not the direct reference!
  - What does this really do?
    - When Hibernate puts an entity into the 2nd level cache, it actually does not store the entity, but the array of data fields that make up the entity from the result of the query that retrieved it.
      - So, this means, every time you get a cache hit, Hibernate goes through its object/entity inflation code to build the entity every time its accessed from the cache.
      - This new feature actually stores the entity, fully inflated, so there is no longer a need to inflate the entity from its raw data anymore!

Persistence / JPA – Example Read Only Immutable Entity Cache

hbm.xml

```
<class name="org.spec.jent.ejb.orders.entity.Item" dynamic-update="true" dynamic-
insert="true"  mutable="false" table="O_ITEM">
```

persistence.xml

```
<property name="hibernate.ejb.classcache.org.spec.jent.ejb.orders.entity.Item"
value="read-only"/>
```

Persistence / JPA – Entity Cache

- Key configuration parameters:
  - Entity Cache.
    - Entity cache types.
      - local-cache, invalidation-cache, replicated-cache.
      - transaction mode.
      - eviction-strategy.
        - Defaults to LRU (Least recently used), but Infinispan gives a new algorithm called LIRS (Low Inter-reference Recency Set), which performs better in our tests.
      - max-entries.
        - How many entries the cache can hold. Sizing this is based on the number of entities to be cached, and perhaps a subset of them based on the access pattern.
      - expiration.
        - max-idle and lifespan.
          - lifespan causes eviction regardless of whether it has been idle or not.
          - max-idle, without specifying lifespan, will cause entities that have not been accessed in that time, to be eligible to be evicted.

Persistence / JPA – Example Entity Cache

```
<cache-container name="hibernate" default-cache="local-query"
module="org.jboss.as.jpa.hibernate:4">
    <local-cache name="entity">
        <transaction mode="NONE"/>
        <eviction strategy="LIRS" max-entries="17030000"/>
        <expiration max-idle="1200000" lifespan="1200000"/>
    </local-cache>
...
</cache-container>
```

Persistence / JPA – Query Cache

- Query Cache Keys:
  - Query type.
    - Cacheable query.
      - A cacheable query is one that always returns the exact same result.
      - Most of the time this means the application has either created a query that is using the primary key, or the query results are always the same because the entity is only read, and there are no inserts and/or deletes.
  - The query cache only stores the keys of the result set!
    - This means you “MUST” also cache the entity in the entity cache.
  - While a query may be cacheable, if the entity is not a good candidate to be cached, then the query cache should not be used.
  - Sizing of the query cache is based on the number of unique combinations of parameters in the query (all still must result in the same result set).

Persistence / JPA – Query Cache

- Key configuration parameters:
  - Query Cache.
    - Entity cache types.
      - local-cache is really the only cache type that makes sense for a query cache. You don't need to invalidate a cached query on another node, as the result set is always the same. You also don't want the overhead of replicating the cache, as other nodes will execute the query once, and cache it anyway.
      - eviction-strategy.
      - max-entries.
      - expiration.
        - max-idle and lifespan.
    - Use minimal puts.
      - This tells Hibernate to deal with only the end state, and not intermediate states within a given unit of work, and will potentially eliminate unnecessary puts to the cache.

Persistence / JPA – Example Query Cache

```
<cache-container name="hibernate" default-cache="local-query"
module="org.jboss.as.jpa.hibernate:4">
...
    <local-cache name="local-query">
        <transaction mode="NONE"/>
        <eviction strategy="LIRS" max-entries="180"/>
        <expiration max-idle="1200000" lifespan="1200000"/>
    </local-cache>
...
</cache-container>
```

Persistence / JPA – Query Cache Example (Cont'd)

```
<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.cache.use_second_level_cache" value="true"/>
             <property name="hibernate.cache.use_query_cache" value="false"/>
             <property name="hibernate.cache.use_minimal_puts" value="true"/>
             <property name="hibernate.cache.infinispan.statistics" value="false"/>
             ...
         </properties>
         </persistence-unit>
</persistence>
```

Persistence / JPA – Batching

- Batching.
  - What is is?
    - Well, this is Hibernate's ability to batch a set of SQL statements to be executed and send them together to the database.
    - This reduces latency, and optimizes the network between the application server and the database server.
    - Is specified through a Hibernate property:
      - <property name="hibernate.jdbc.batch_size" value="20"/>
    - Sizing this parameter requires good knowledge of how many inserts, deletes, updates, etc. there typically are in your transactions.
    - You can put this property in your persistence.xml.

Persistence / JPA – Batching (Cont'd)

- Fetch sizes.
  - In Hibernate you can specify how many rows to return from the database at one time for queries that return more than one row.
  - If you return one row at a time, and there are tens, hundreds, or even thousands, its going to increase response times, and, obviously, add lots of latency for the rounds trips to and from the database.
  - You specify this through a Hibernate property:
    - <property name="hibernate.jdbc.fetch_size" value="20"/>
  - Sizing of this parameter requires knowledge of the application queries and the typical usage of the result set.
    - e.g., in one of our test applications we set the fetch size is set to 20, even though one of the main queries may return as many as 500 rows.
      - The reason for 20, instead of 500, is that the result set is managed by a stateful session bean that paginates by 20 rows at a time. Understanding the frequency in which a user will even go past the first 20 rows is important!
  - Like the batch size, this can be set in the persistence.xml.

Persistence / JPA – Batch Inserts

- Batch Inserts.
  - What do I mean by batch inserts?
    - This is when you can take multiple insert statements, that would normally be sent to and executed by the database one at a time, and make it into a single insert statement.
    - e.g.
      - Insert into Table (id, val1, val2, val3) values ('x', 'x', 'x', 'x);
      - Insert into Table (id, val1, val2, val3) values ('y', 'y', 'y', 'y');
      - Insert into Table (id, val1, val2, val3) values ('z', 'z', 'z', 'z');
      - And turn it into:
        - Insert into Table (id, val1, val2, val3) values('x', 'x', 'x', 'x'), ('y', 'y', 'y', 'y'), ('z', 'z', 'z', 'z');
      - This feature is dependent on two things. This first being a Hibernate property:
        - <property name="hibernate.order_inserts" value="true"/>
      - The second being the JDBC driver's capabilities to rewrite the statement:
        - e.g., the MySQL JDBC driver has a connection property called:
          - RewriteBatchedStatements
        - We have a prototype of the PostgreSQL JDBC driver with this capability in our lab for testing.
          - We have engaged the community, and we believe this change will be accepted once we finish our testing, and shows the performance improvements.

Persistence / JPA – Example Batch Inserts

```
...
<datasource jndi-name="java:/MySqlDS" pool-name="MySqlDS" use-ccm="false">
    <connection-url>
        jdbc:mysql://localhost:3306/EJB3
     </connection-url>
     <connection-property name="maintainTimeStats">
         false
     </connection-property>
     <connection-property name="rewriteBatchedStatements">
         true
     </connection-property>
...
```

Persistence / JPA – Batch Inserts (Cont'd)

- **Important notes on batch inserts:**
  - You must both specify the ordered inserts parameter to Hibernate, and have a JDBC driver that can rewrite the statements into one.
  - The Hibernate property enables the JDBC driver to detect that fact that all the inserts are to the same table, and can be rewritten.
  - The Hibernate property orders the inserts statements by their primary key values (actually uses the entities hash code).
  - If your JDBC driver does not have the ability to rewrite the statements into one, then throughput will suffer.
    - You are adding the sort overhead before executing the statements, so without benefit of the rewrite of the statement, that shortens the insert time, you just make things take longer.
  - This capability does show benefits in OLTP applications, as well as batch applications, but careful testing is in order for OLTP workloads.

Persistence / JPA –Reflection Optimization

- Reflection optimization has been around in Hibernate for a long time, and it used to be the default.
  - It is now turned off by default, because it was thought with the newer JVM's that this was no longer needed.
  - Our testing, in our performance lab, has shown that it still provides benefit, even up to JDK 7.
  - If you are using JDK 8, and EAP 6.4.1 is now supported for JDK 8, your mileage may vary.
    - We have not tested this with JDK 8 as of yet, so up to JDK 7, you can turn this on and you will see improvement in overall performance for persistence using JPA.

Persistence / JPA – Example Reflection Optimization

persistence.xml

```
<properties>
            ...
    <property name=”hibernate.bytecode.use_reflection_optimizer” value=”true"/>
            ...
</properties>
```

JCA

- **For the JCA container we are going to concentrate on data sources.**
  - The JCA container is responsible for the integration of our data sources into the application server and provides services for them.
  - The three areas we will concentrate on are:
    - Database connection pooling, the Cached Connection Manager, and Prepared Statement Caching.
    - For database connection pooling, we have created an entirely new connection pool that performs and scales much better.
      - We recommend its use over the default for all applications.
  - The size of the database connection pool is directly related to the concurrent execution of queries across your application.
    - To small a pool, and you add to your response times.
    - The default timeout for a database connection is 30 seconds!
      - This is a long time to wait, and you won't get the log message of a timeout until you have real problems with application performance.
  - The Cached Connection Manager provides a debugging capability for leaked database connections.
    - Unless you are doing your own JDBC code, this is typically not needed.
  - There are also some JCA container internals that have some property settings that can improve performance and scalability, reduce contention and CPU consumption too.

JCA - Example

```
...
<datasource jndi-name="java:/MySqlDS" pool-name="MySqlDS" use-ccm="false">
...
<pool>
    <min-pool-size>200</min-pool-size>
    <max-pool-size>250</max-pool-size>
    <prefill>true</prefill>
</pool>
...
<statement>
    <prepared-statement-cache-size>100</prepared-statement-cache-size>
    <share-prepared-statements>true</share-prepared-statements>
</statement>
…
standlone*.xml or domain.xml

<system-properties>
  <property name=”ironjacamar.mcp”
value=”org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreConcurrentLinkedQueueManagedConnectionPo
ol"/>
    <property name=”ironjacamar.disable_enlistment_trace”  value=”true”/>
    <property name=”ironjacamar.disable_runtime_statistics” value=”true”/>
</system-properties>
```

JCA (Cont'd)

- **Important notes on data source configuration:**
  - The shared-prepared-statements parameter, when set to true, along with the cache being non-zero, will reuse the same prepared statement if its executed more than once in the same transaction.
    - This may or may not happen in your application, but I have seen lots of applications that do this.
  - Sizing of the prepared statement cache is based on the number of prepared statements your application has within it.
    - Of course, this is by data source, so if your application uses multiple data sources (in JPA, multiple persistence units), then you would configure a cache for each one, and it would be sized based on the prepared statement count for that individual data source.

JMS

- **Our JMS provider is HornetQ.**
  - HornetQ is the successor to JBoss Messaging, which was the default provider in EAP 5.x, but was discontinued in the EAP 6.x series.
  - HornetQ is a very high performance, and highly reliable JMS provider.
    - In fact, it holds the world record SPECjms2007 result:
      - http://www.spec.org/jms2007/results/res2011q2/
    - For those of you in the know, you know that Red Hat acquired Fuse Source, and that brought ActiveMQ into the fold.
    - So, just as an FYI, what happens to our JMS provider in the future?
      - HornetQ's code was donated to the Apache Foundation (it was already ASL 2.0 licensed).
      - That code base makes up the foundation for Apache ActiveMQ Artemis.
        - Apache ActiveMQ Artemis combines the functionality of both ActiveMQ 5.x and HornetQ into a single, high performance, highly scalable messaging solution.
          - This new solution will become the basis for our JMS provider in the EAP 7 series.
            - You will see this in the early releases of Wildfly 10 upstream.
  - There is no longer a database backend for persistence. Instead, there is a high performance journal maintained on disk.

JMS (Cont'd)

- **Key configuration parameters:**
  - journal-type
    - ASYNCIO, NIO
      - The ASYNCIO option specifies using native ASYNC I/O capabilities, plus opens the file using DIRECT I/O, which bypasses the file system buffer cache.
      - The NIO option uses the JDK's NIO API's to write to the journal.
  - journal-directory
    - The placement of the journal files is important, as the default will be relative to the install of the application server, and that may not be the best performing file system to place your persistent messages on.
  - Pooled connection factory:
    - transaction mode
      - Whether to use XA transactions or local transactions.
    - min-pool-size, max-pool-size
      - The session pool size.
      - The sizing of this depends on the number of concurrent MDB's your application may be executing, and relates to the maxSession on those MDB's, or if you are using the JMS api directly, the number of concurrent messages being processed.

JMS – Example

```
<hornetq-server>
...
    <journal-type>ASYNCIO</journal-type>
    <journal-directory>
       <path>/some/absolute/path</path>
    </journal-directory>
...
<pooled-connection-factory name="hornetq-ra">
    <transaction mode="xa"/>
    <min-pool-size>180</min-pool-size>
    <max-pool-size>198</max-pool-size>
...
```

Java Virtual Machine Specific Tuning

Java Virtual Machine

- The Java Virtual Machine is our key piece of software, as the entire platform is completely dependent on it.
  - Tuning the JVM can be difficult.
  - Start simple!
  - Test one thing at a time!
  - Understand your goals!
  - We will directly discuss Large Page Memory.

Java Virtual Machine – Large Page Memory

- What is large page memory?
  - The normal memory page size in the OS is usually 4k. When you consider the memory footprints of modern servers, there are an awful lot of memory pages that the OS has to manage.
  - Large page memory, is a larger memory page than 4k.
  - Typically, on X86_64 systems, this is 2 MB.
  - Newer X86_64 systems can support 1 GB page sizes.
    - If using 1GB pages on a Linux system, and using JDK 7 or earlier, you will have to set the minimum and maximum permanent generation sizes to at least 1GB!
  - Another attribute of large pages (called HugeTLB in Linux), is that they are locked into physical memory, and cannot be swapped to disk.
  - This is a great attribute for the JVM.
    - If you have ever experienced the JVM heap being swapped to disk, you know this is a situation that often leads to the JVM crashing.
- How do we take advantage of large page memory?

Java Virtual Machine – Large Page Memory

- The Oracle JVM, as well as OpenJDK, requires the following option, passed on the command-line, to use large pages:
  - -XX:+UseLargePages
- There is also something in Linux called “Transparent huge pages”.
  - Transparent huge pages allows the operating system (Linux) to evaluate memory usage of processes and dynamically move from regular pages to large pages (consolidates many 4k pages into a large page).
  - For some workloads this may be sufficient. For other, the static configuration, and usage will be better.
  - For transparent huge pages, you do not set the JVM argument.
  - If you have transparent huge pages turned on in the Linux kernel, you should not specify the JVM argument to use large pages. Only if you opt for the static configuration, which I'll walk through next.
- ***From my testing so far, I would not recommend having both statically defined large pages, and transparent huge page support on at the same time!***
  - ***To turn off transparent huge pages, you can set a boot parameter in grub.conf.***
  - ***Some recent testing on RHEL 7.1 may indicate considerable improvement in transparent huge page support, but at this time, we have not had the time to verify, so your mileage may vary, and if you are on RHEL 7.1, it is worth investigating.***

Java Virtual Machine – Example Turning Off Transparent Hugepages

• To turn off transparent huge pages, you can set a boot parameter in grub.conf as follows:

title Red Hat Enterprise Linux (2.6.32-220.el6.x86_64)
root (hd0,0)
kernel /vmlinuz-2.6.32-220.el6.x86_64 ro root=UUID=f8196a3a-1f1a-47f3-9141-1d33e3da4454 rd_NO_LUKS rd_NO_LVM LANG=en_US.UTF-8 rd_NO_MD quiet SYSFONT=latarcyrheb-sun16 rhgb crashkernel=auto KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM **transparent_hugepage=never**
initrd /initramfs-2.6.32-220.el6.x86_64.img

Java Virtual Machine – Example Large Page Memory

- The Oracle JVM, as well as OpenJDK, requires the following option:
  - -XX:+UseLargePages
    - The Oracle 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 to perhaps 3 times the amount of physical memory.
        - Setting this value smaller, may result in error-no=22 on startup of the JVM. This error, is “no space left on device”, and is a rather new phenomenon on the Linux kernel.
    - 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**.
      - Newer systems can support 1GB page sizes, but you can only have one page size configured at a time.
        - So, you can either have the 2MB (most x86_64 systems), or the 1GB pages, but not both.
    - 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.

Java Virtual Machine – Example Large Page Memory (Cont'd)

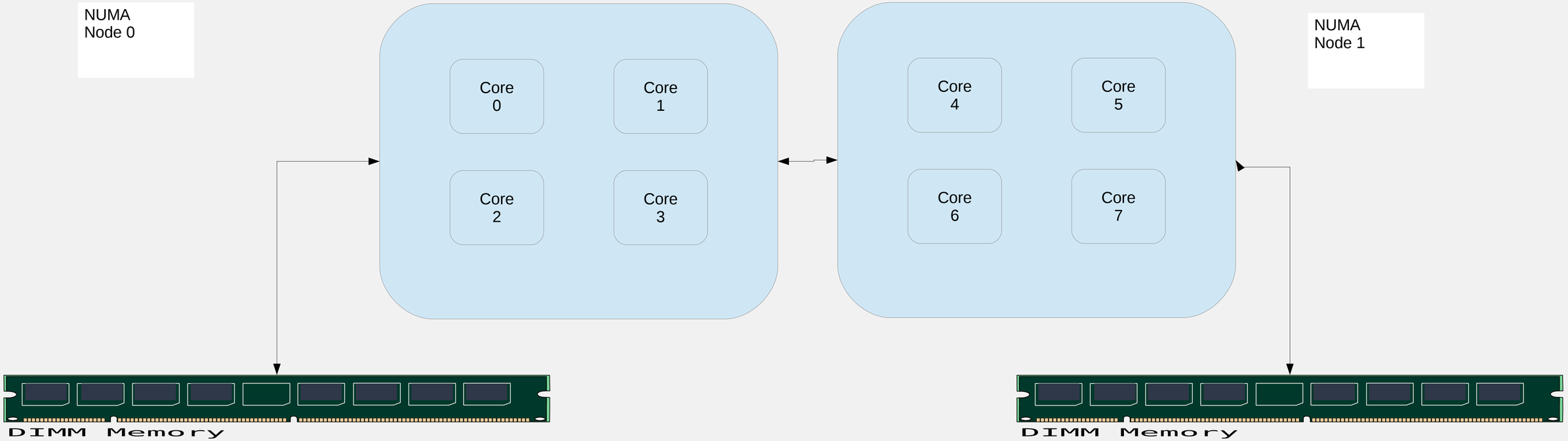
- Next, set the following:
  - In **/etc/security/limits.conf**
    - <username> soft memlock n
    - <username> hard memlock n
      - Where ***<username>*** is the runtime user of the JVM.
      - Where ***n*** is the number of pages from vm.nr_hugepages * the page size in KB from /proc/meminfo.
        - The value for ***n*** can also be **unlimited**.
  - 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!

Red Hat Enterprise Linux Specific Tuning

Red Hat Enterprise Linux - NUMA

- What is NUMA?
  - It stands for non-uniform memory architecture.
  - Why is it important?
    - It's important because all newer x86 architecture servers are based on NUMA.
  - The memory architecture of a NUMA system is laid out so that each socket (with its cores) is attached directly to a single bank of memory.
    - Accessing this bank of memory has the lowest latency of all memory accesses.
    - To access memory from any other bank of memory, it has to go through the other sockets to access the memory adding significant latency.
    - The best performance comes from keeping processing running on a single socket, and having its memory needs satisfied through the local memory bank attached to that socket.

NUMA


NUMA
Node 0
Core
0
Core
1
Core
2
Core
3
Core
4
Core
5
Core
6
Core
7
NUMA
Node 1
DIMM Memory
DIMM Memory

Red Hat Enterprise Linux – NUMA (Cont'd)

- How do you take advantage of NUMA?
  - First, you have to understand your NUMA hardware layout.
  - Second, you have to start the JVM with numactl.
  - Third, you have to supply numactl the policy information necessary to bind the process and its threads to the NUMA node, and its memory accesses to the local memory of that NUMA node.
  - Fourth, if you are using large page memory, you need to understand how many large pages are on each NUMA node.
  - Fifth, you have to understand the number of threads to use for GC, as the default JVM ergonomics will not apply.
    - In the test example I show, I set the number of GC threads to the number of virtual cores (including hyper-threading) that were on the NUMA node I bound the JVM too.
- So, let's take a look at the commands you have to issue to accomplish all five points above.

Red Hat Enterprise Linux – Example NUMA

```
[root@jbosstesting ~]# numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 2 4 6 8 10 12 14
node 0 size: 12277 MB
node 0 free: 710 MB
node 1 cpus: 1 3 5 7 9 11 13 15
node 1 size: 12287 MB
node 1 free: 225 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

Red Hat Enterprise Linux – Example NUMA

```
if [ "x$LAUNCH_JBOSS_IN_BACKGROUND" = "x" ]; then
      # Execute the JVM in the foreground
      eval numactl –-membind 0 –cpunodebind 0 \"$JAVA\" -D\"[Standalone]\" $JAVA_OPTS \
         \"-Dorg.jboss.boot.log.file=$JBOSS_LOG_DIR/boot.log\" \
         \"-Dlogging.configuration=file:$JBOSS_CONFIG_DIR/logging.properties\" \
         -jar \"$JBOSS_HOME/jboss-modules.jar\" \
         -mp \"${JBOSS_MODULEPATH}\" \
         -jaxpmodule "javax.xml.jaxp-provider" \
         org.jboss.as.standalone \
         -Djboss.home.dir=\"$JBOSS_HOME\" \
         -Djboss.server.base.dir=\"$JBOSS_BASE_DIR\" \
         "$@"
      JBOSS_STATUS=$?
   else
      ...
```

Red Hat Enterprise Linux – Example NUMA (Cont'd)

```
 [root@jbosstesting hugepages-2048kB]# pwd
/sys/devices/system/node/node0/hugepages/hugepages-2048kB
[root@jbosstesting hugepages-2048kB]# cat nr_hugepages
5376
[root@jbosstesting hugepages-2048kB]#

...

[root@jbosstesting hugepages-2048kB]# pwd
/sys/devices/system/node/node1/hugepages/hugepages-2048kB
[root@jbosstesting hugepages-2048kB]# cat nr_hugepages
5376
[root@jbosstesting hugepages-2048kB]#
```

Q&A

LEARN. NETWORK.
EXPERIENCE OPEN SOURCE.