JUDCon

JBoss Users & Developers Conference

2012:Boston

Infinispan
from POC to Production

Who am I?

Mark Addy, Senior Consultant

Fast, Reliable, Secure, Manageable

Agenda

Part 1

- An existing production system unable to scale

Part 2

- A green-field project unable to meet SLA’s

About the Customer

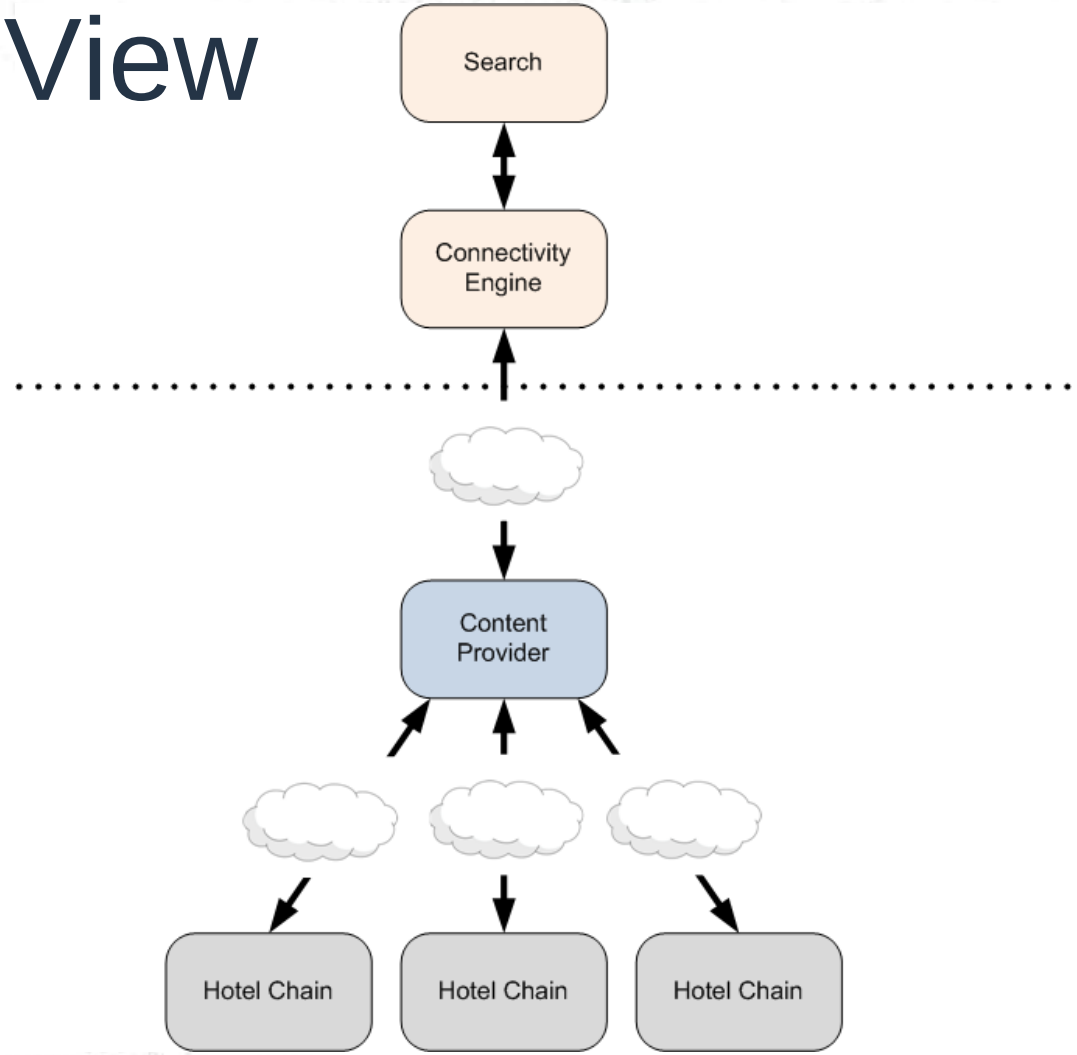
- Global on-line travel & accommodation provider
  - 50 million searches per day
- Our relationship
  - Troubleshooting
  - Workshops

Part 1 – Existing Application

Connectivity Engine

- Supplements site content with data from third parties (Content Providers)
  - Tomcat
  - Spring
  - EhCache
  - MySQL
  - Apache load-balancer / mod_jk

Logical View



Content Provider Challenges

- Unreliable third party systems
- Distant network communications
- Critical for generating local site content
- Response time
- Choice & low response time == more profit

Existing Cache

- NOT Hibernate 2LC
- Spring Interceptors wrap calls to content providers

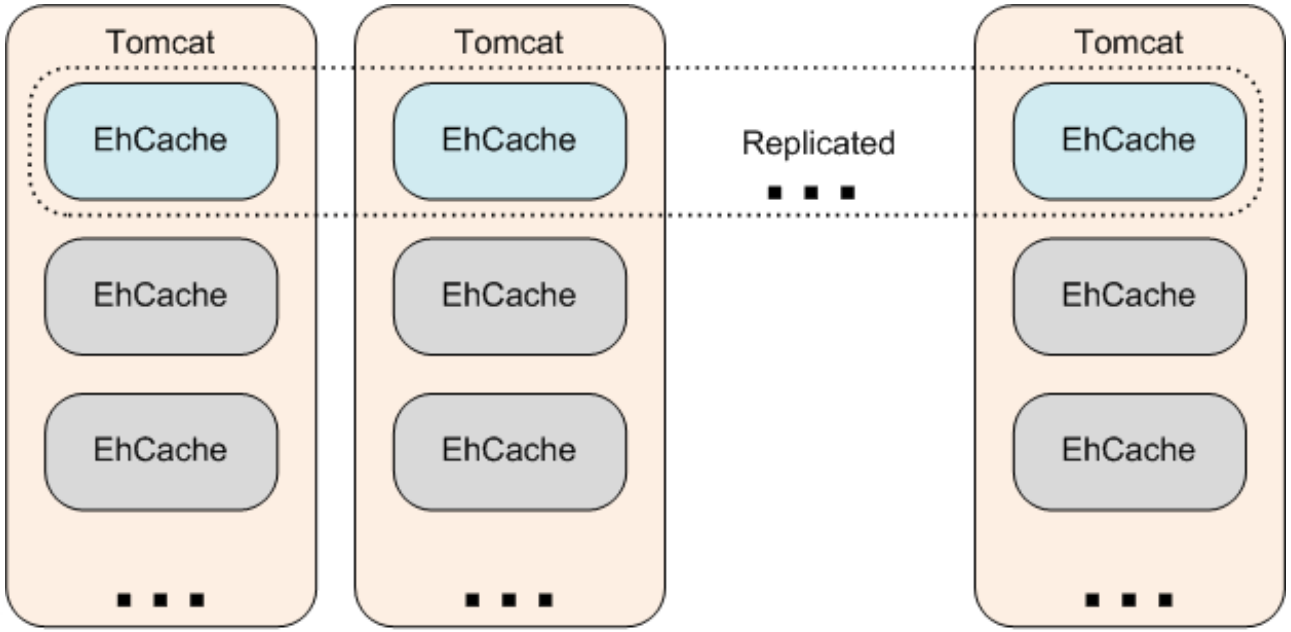
```
<bean id="searchService" class="org.springframework.aop.framework.ProxyFactoryBean">
   <property name="proxyInterfaces" value="ISearchServiceTargetBean"/>
   <property name="target" ref="searchServiceTargetBean"/>
   <property name="interceptorNames">
        <list>
                <value>cacheInterceptor</value>
        </list>
   </property>
</bean>

<bean id="searchServiceTargetBean" class="SearchServiceTargetBean">
   ...
</bean>
```

Extreme Redundancy

800,000 elements

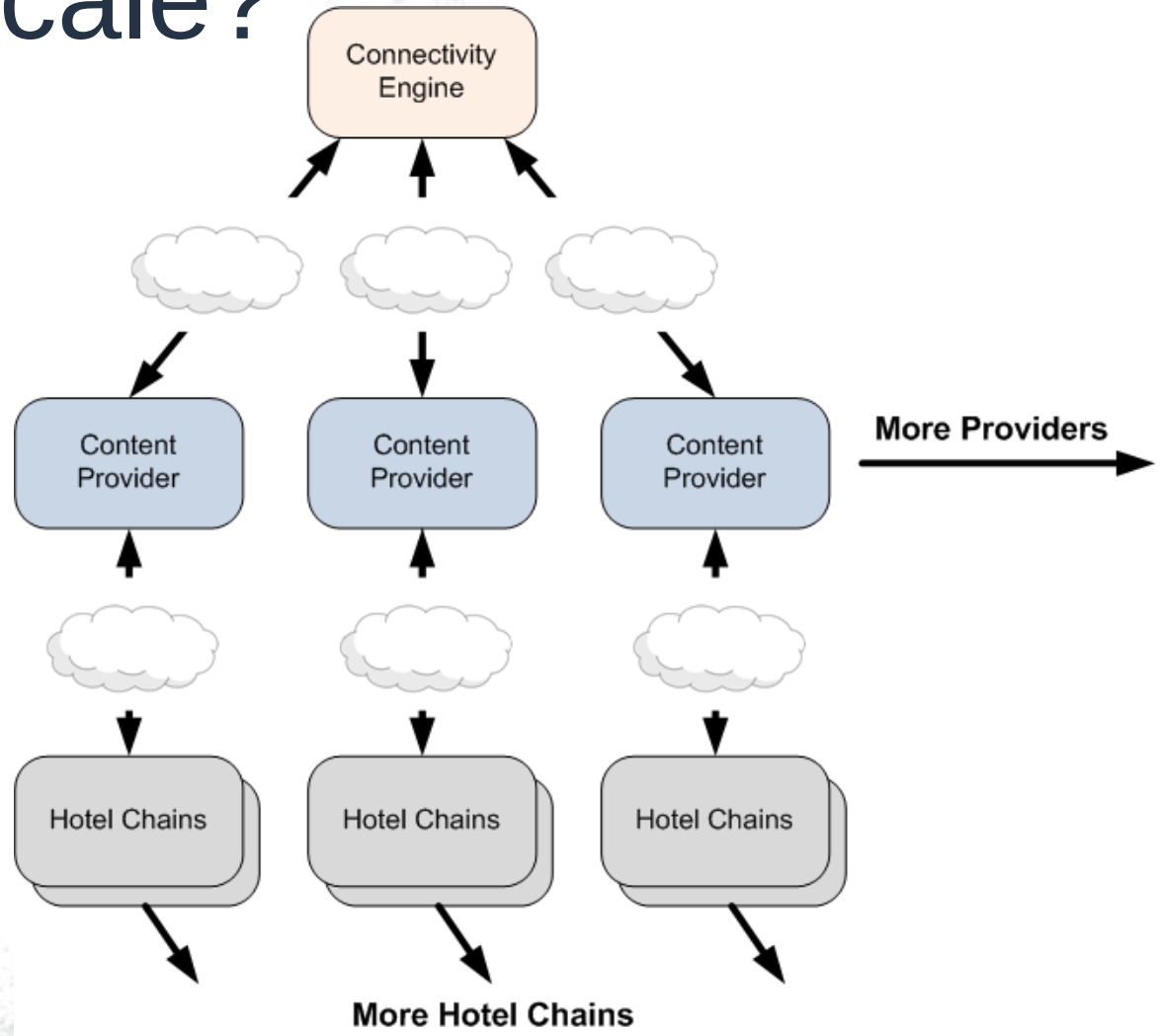
10 nodes = 10 copies of data



The Price

- 10G JVM Heap
  - 10-12 second pauses for major GC
  - Over 8G of heap is cache
- Eviction before Expiry
  - More trips to content providers
- EhCache expiry / eviction piggybacks client cache access

How to Scale?


Connectivity Engine
Content Provider
Content Provider
Content Provider
More Providers
Hotel Chains
Hotel Chains
Hotel Chains
More Hotel Chains

Objectives

- Reduce JVM Heap Size
  - 10 second pauses are too long
- Increase cache capacity
- Remove Eviction
  - Cache entries should expire naturally
- Improve Response Times
  - Latency decreases if eviction, GC pauses and frequency are reduced

Discussions

- Pre-sales workshop
  - Express Terracotta EhCache
  - Oracle Coherence
  - Infinispan

Why Infinispan?

- Open source advocates
- Cutting edge technology

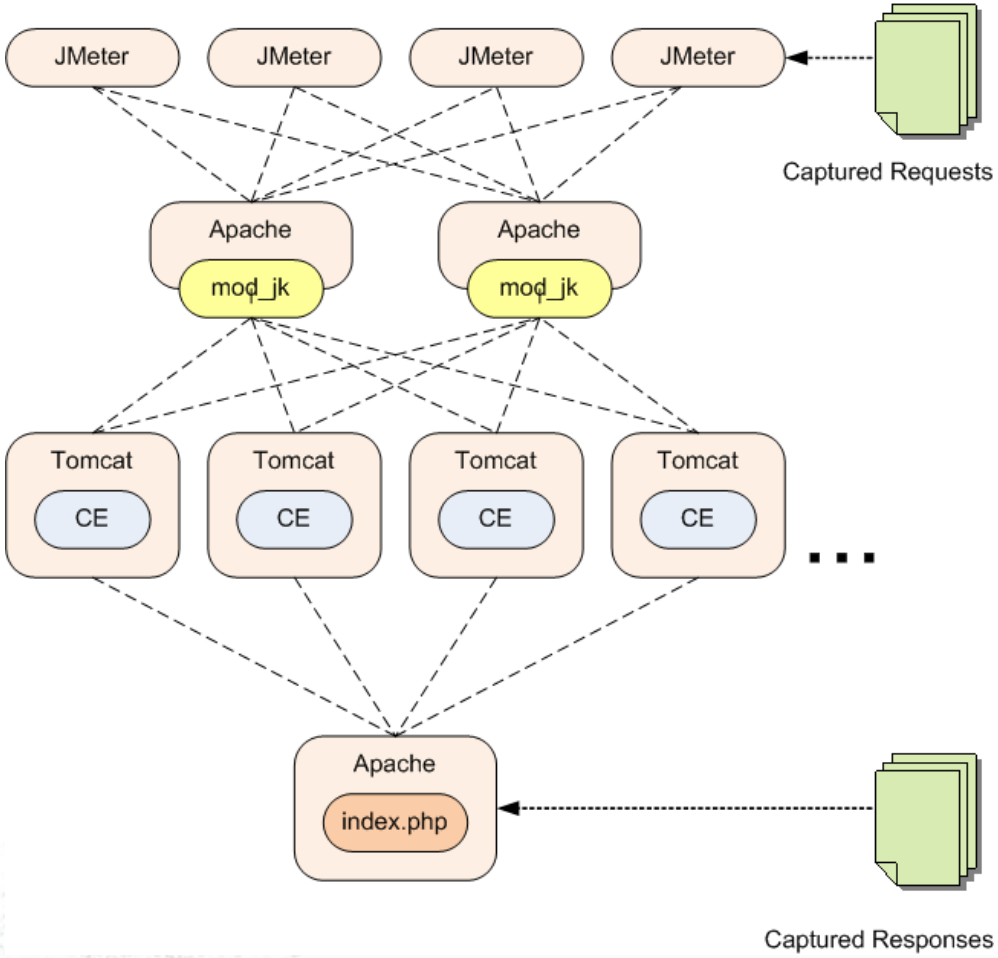
Benchmarking

- Must be reproducible
- Must reflect accurately the production load and data
  - 50 million searches / day == 600 / sec
- Must be able to imitate the content providers

Solution

- Replica load-test environment
- Port mirror production traffic
  - Capture incoming requests
  - Capture content provider responses
- Custom JMeter script
- Mock application Spring Beans

Benchmarking Architecture



Benchmarking Validation

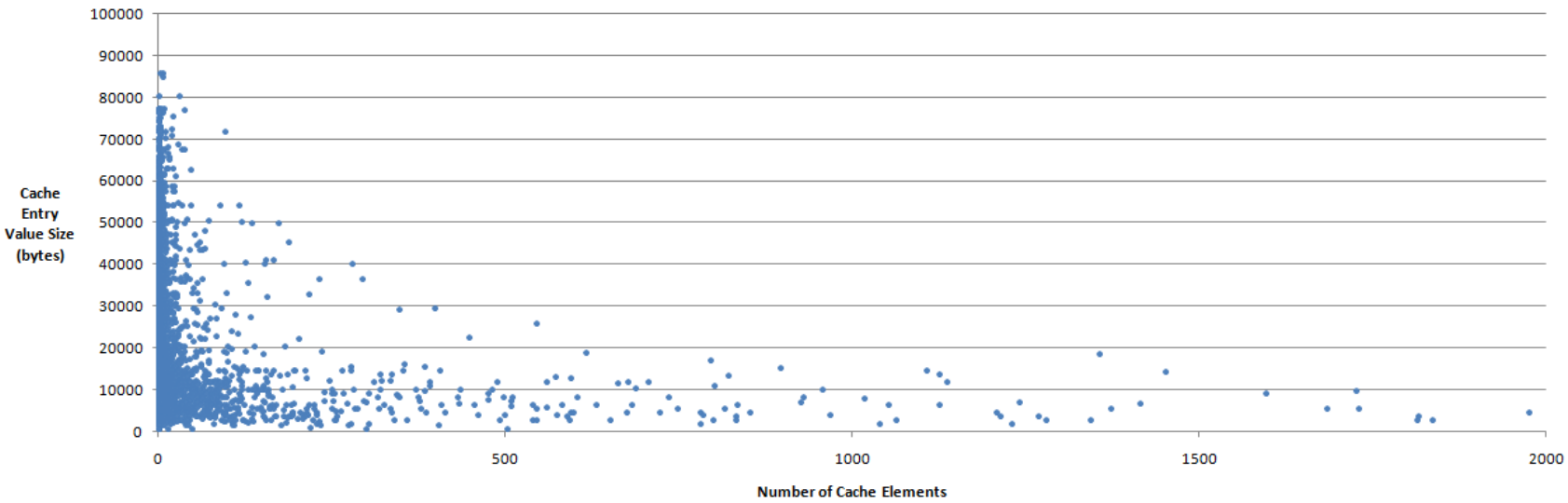
- Understand your cached data
  - jmap

```
jmap -dump:file=mydump.hprof <pid>
```

  - Eclipse Memory Analyzer
  - OQL

```
SELECT
toString(x.key)
, x.key.@retainedHeapSize
, x.value.@retainedHeapSize
FROM net.sf.ehcache.Element x
```

Benchmarking Validation



Extract cached object properties - you can learn a lot quickly

- creationTime
- lastAccessTime
- lastUpdateTime
- hitCount
- timeToLive
- timeToIdle
- etc
- etc

Enable JMX for Infinispan

Enable CacheManager Statistics

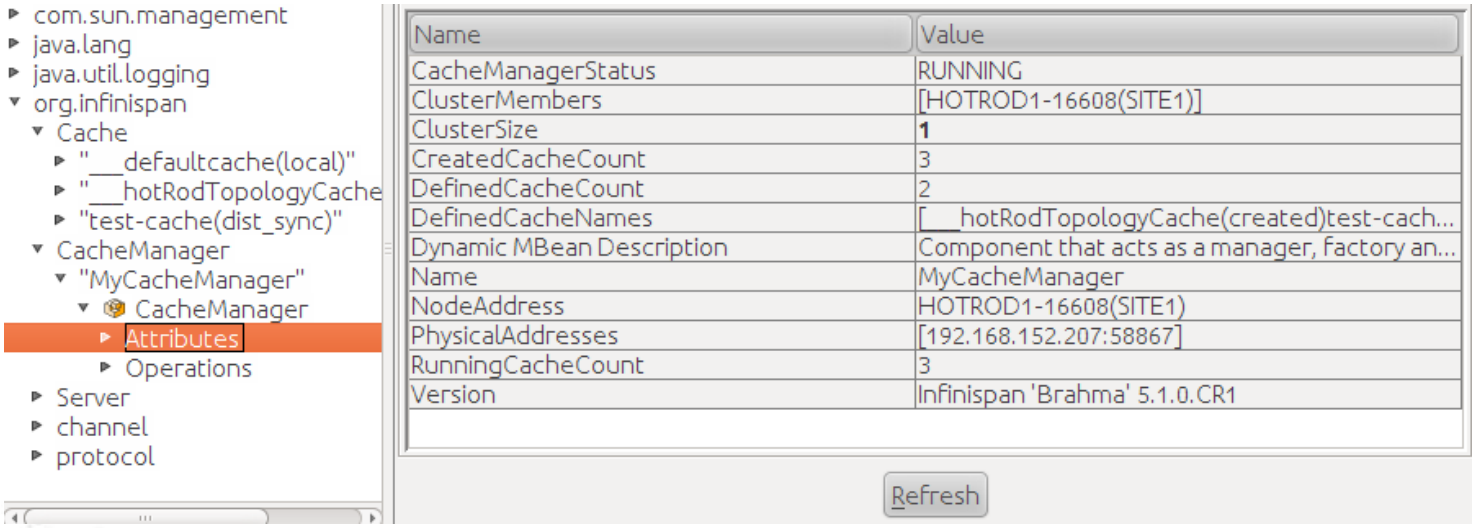
```
<global>
   <globalJmxStatistics
     enabled="true"
     jmxDomain="org.infinispan"
     cacheManagerName="MyCacheManager"/>
   ...
</global>
```

Enable Cache Statistics

```
<default>
     <jmxStatistics enabled="true"/>
     ...
</default>
```

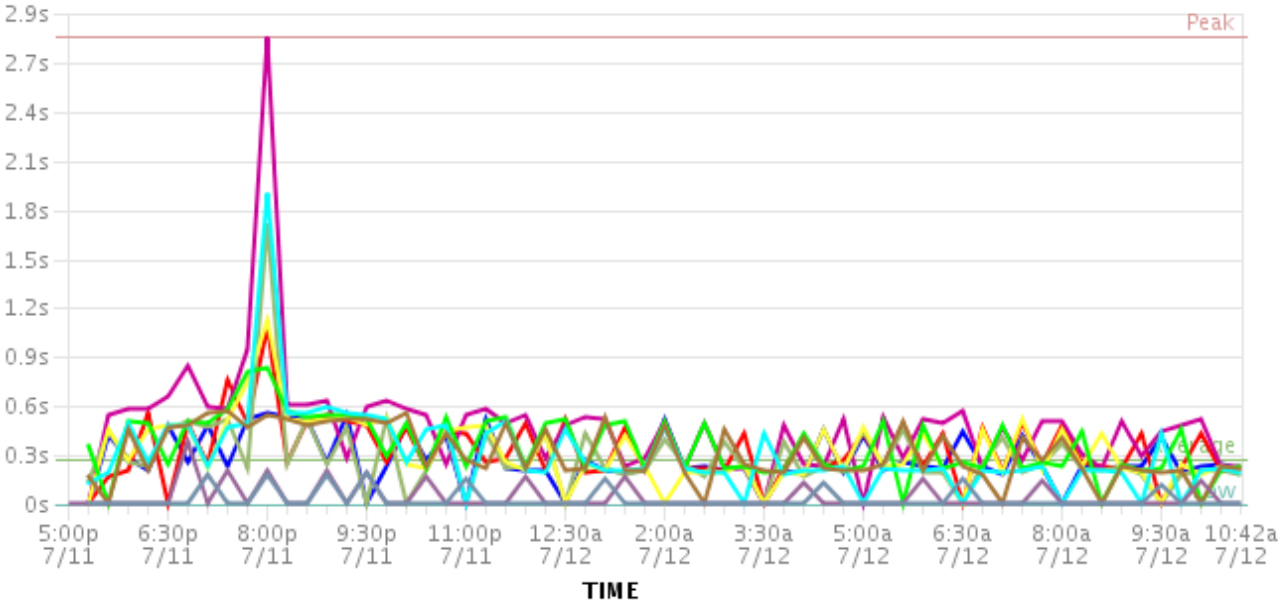
Enable Remote JMX

```
-Dcom.sun.management.jmxremote.port=nnnn
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

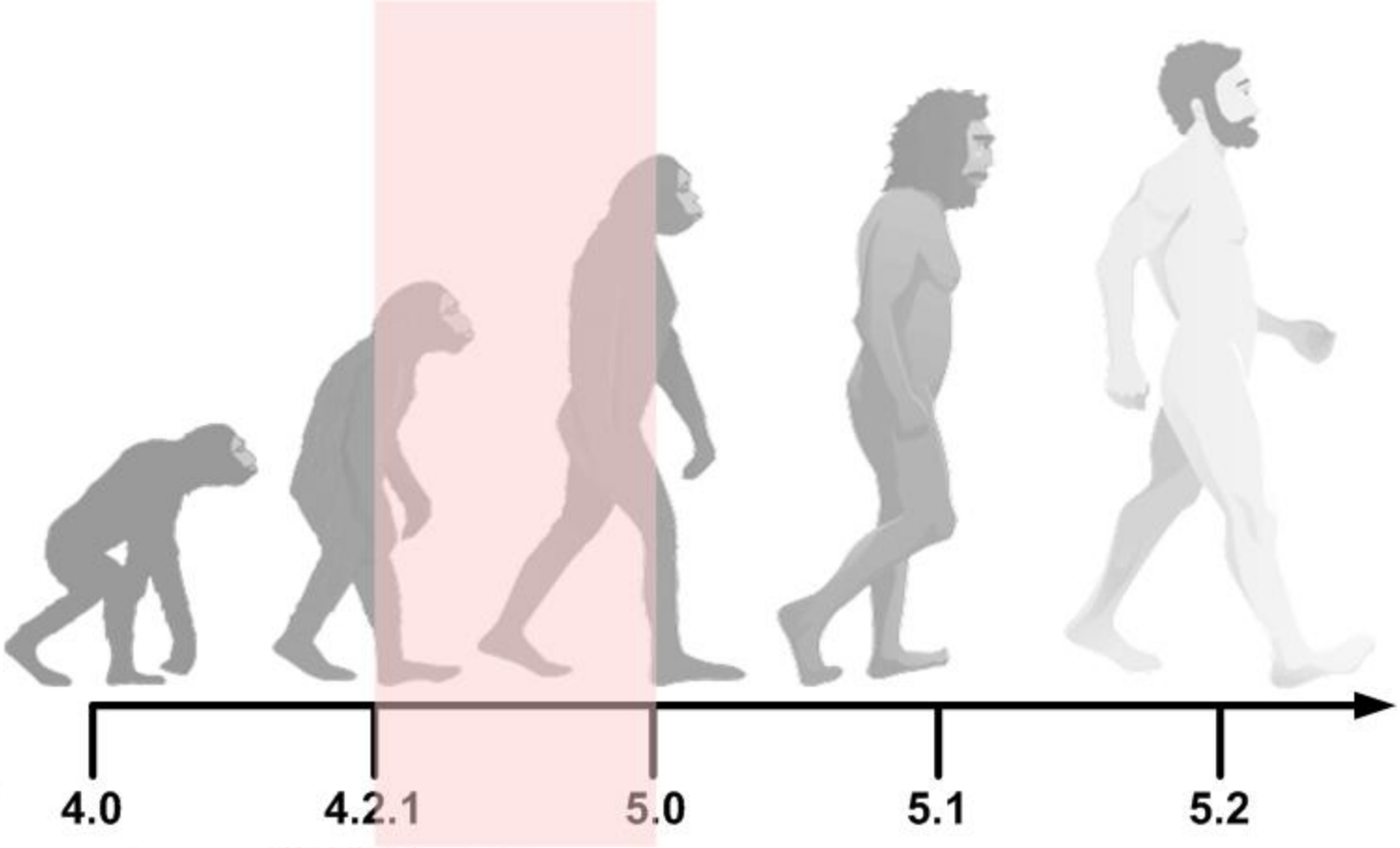


Record Performance

- RHQ http://rhq-project.org
  - JVM memory, GC profile, CPU usage
  - Infinispan plugin

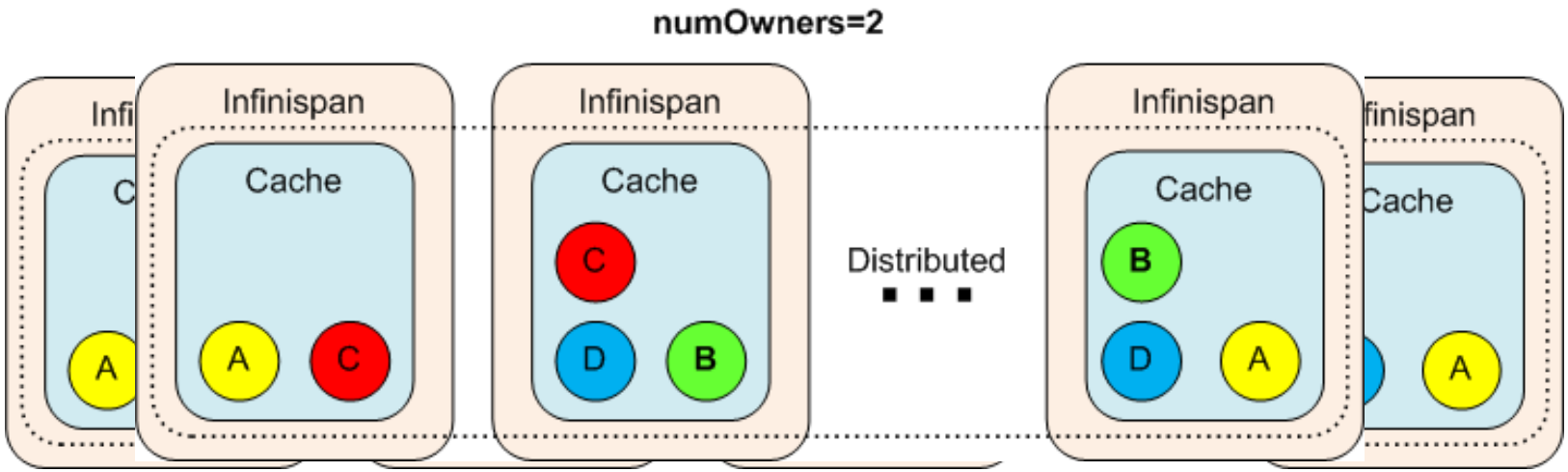


Infinispan



Distributed Mode

hash(key) determines owners



Distribution Features

- Configurable redundancy
  - numOwners
- Dynamic scaling
  - Automatic rebalancing for distribution and recovery of redundancy
- Replication (distribution) overhead does not increase as more nodes are added

Hotrod

- Client – Server architecture
  - Java client
  - Connection pooling
  - Dynamic scaling
  - Smart routing
- Separate application and cache memory requirements

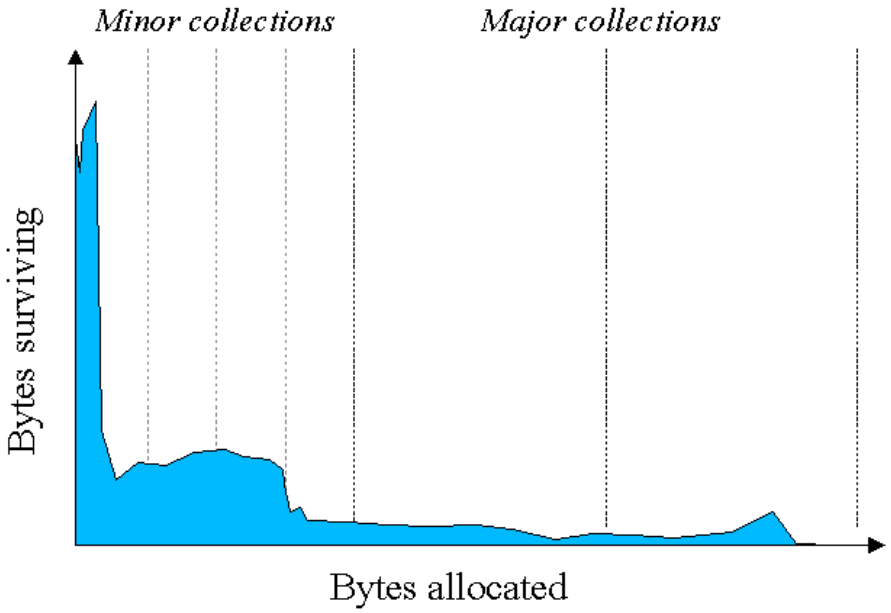
Application – Cache Separation

Application

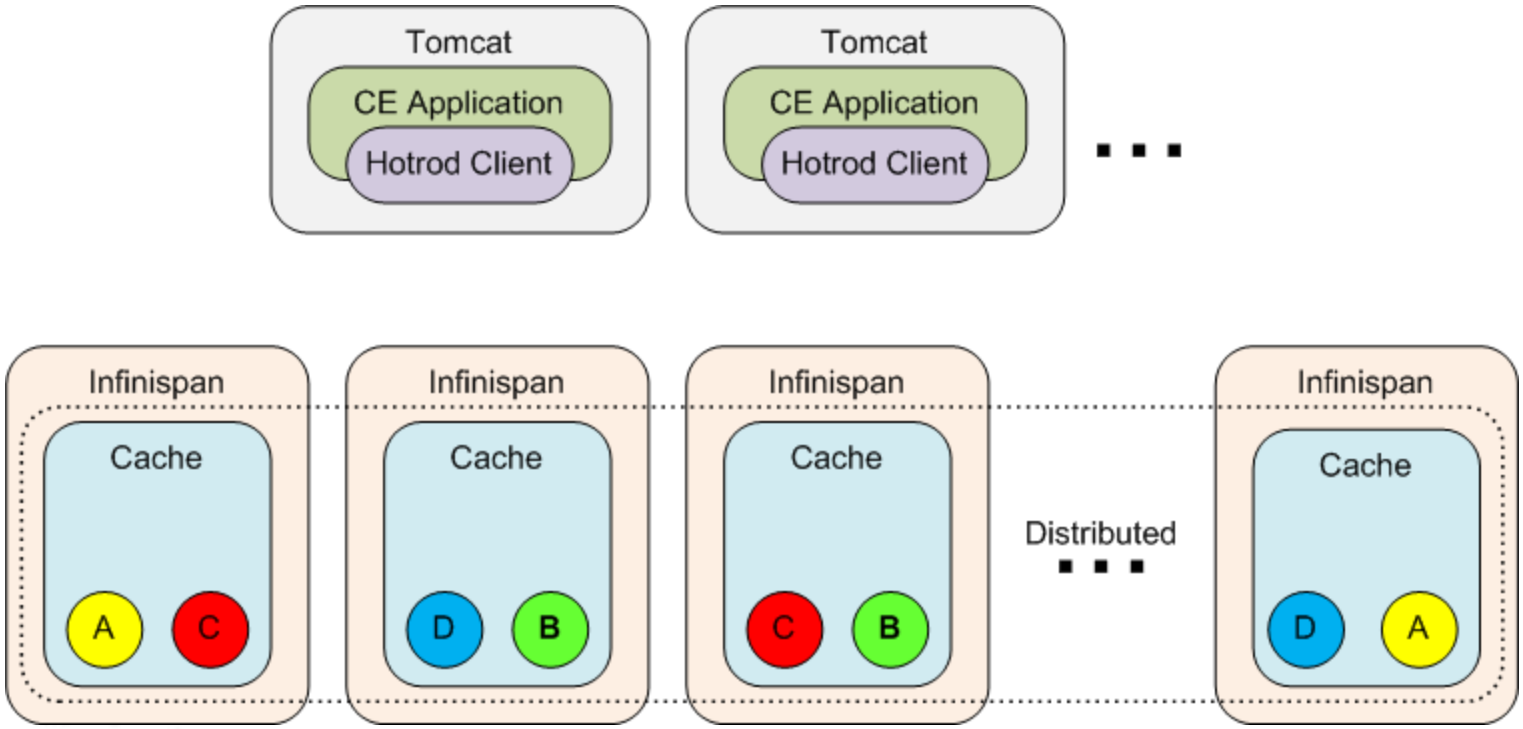
- CPU intensive
- High infant mortality

Cache

- Low CPU requirement
- Mortality linked to expiry / eviction



Hotrod Architecture



Remember this is cutting edge

- Latest final release was 4.2.1
- Lets get cracking...
  - Distributed mode
  - Hotrod client
  - What issues did we encounter...

Topology Aware Consistent Hash

- Ensure back-ups are held preferentially on separate machine, rack and site

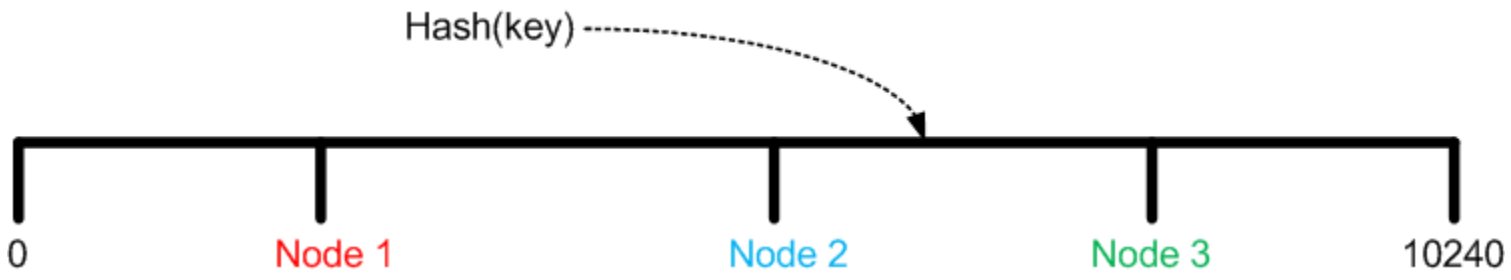
```
<infinispan
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:infinispan:config:5.0 http://www.infinispan.org/s
      xmlns="urn:infinispan:config:5.0">

   <global>
      <globalJmxStatistics
            enabled="true"
            jmxDomain="org.infinispan"
            cacheManagerName="MyCacheManager"/>
      <transport clusterName="infinispan-cluster"
            machineId="machine1"
            siteId="site1"
            rackId="rack1"
            nodeName="node1">
         <properties>
            <property name="configurationFile" value="jgroups-udp.xml" />
         </properties>
      </transport>
   </global>
```

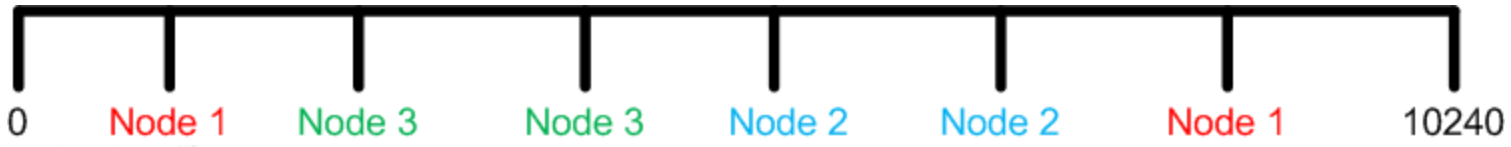
- https://community.jboss.org/thread/168236
- We need to upgrade to the latest 5.0.0.CR

Virtual Nodes

Sub-divides hash wheel positions

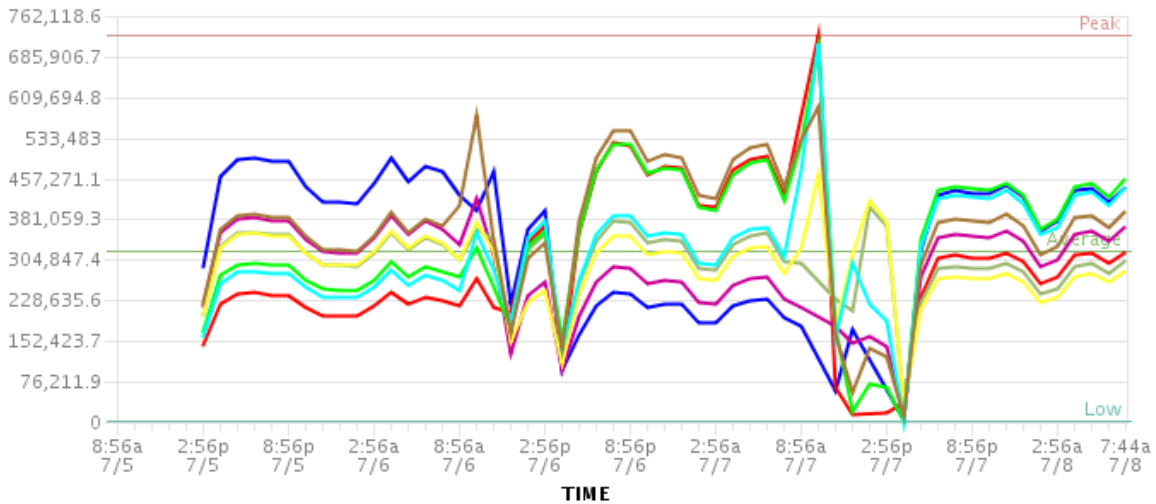



```
<hash numVirtualNodes=2/>
```



Virtual Nodes

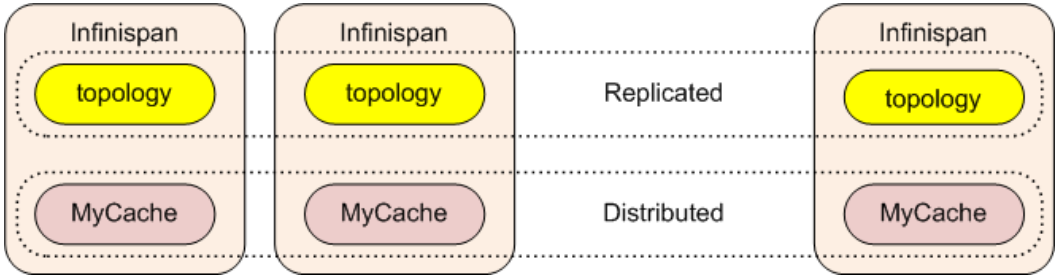
- Improves data distribution



- But didn’t work at the time for Hotrod
- https://issues.jboss.org/browse/ISPN-1217

Hotrod Concurrent Start-up

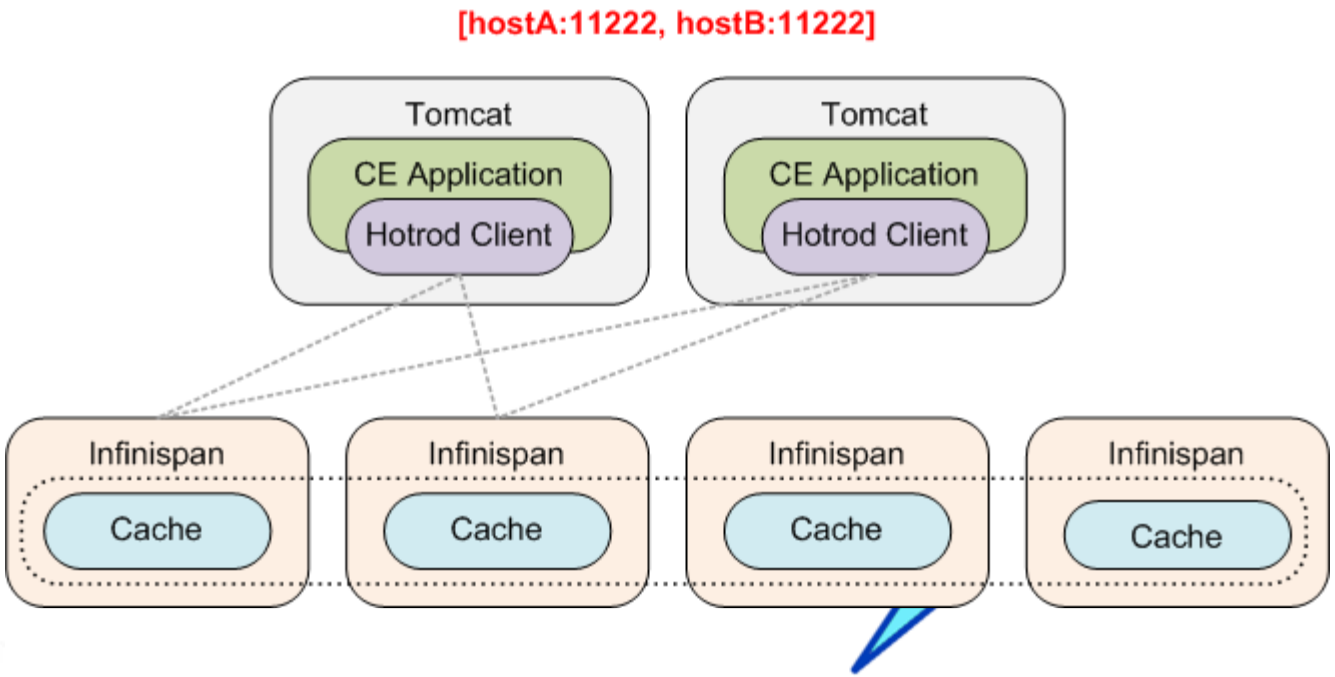
- Dynamic scaling
  - Replicated ___hotRodTopologyCache holds current cluster topology



  - New starters must lock and update this cache to add themselves to the current view
  - Deadlock!
  - https://issues.jboss.org/browse/ISPN-1182
- Stagger start-up

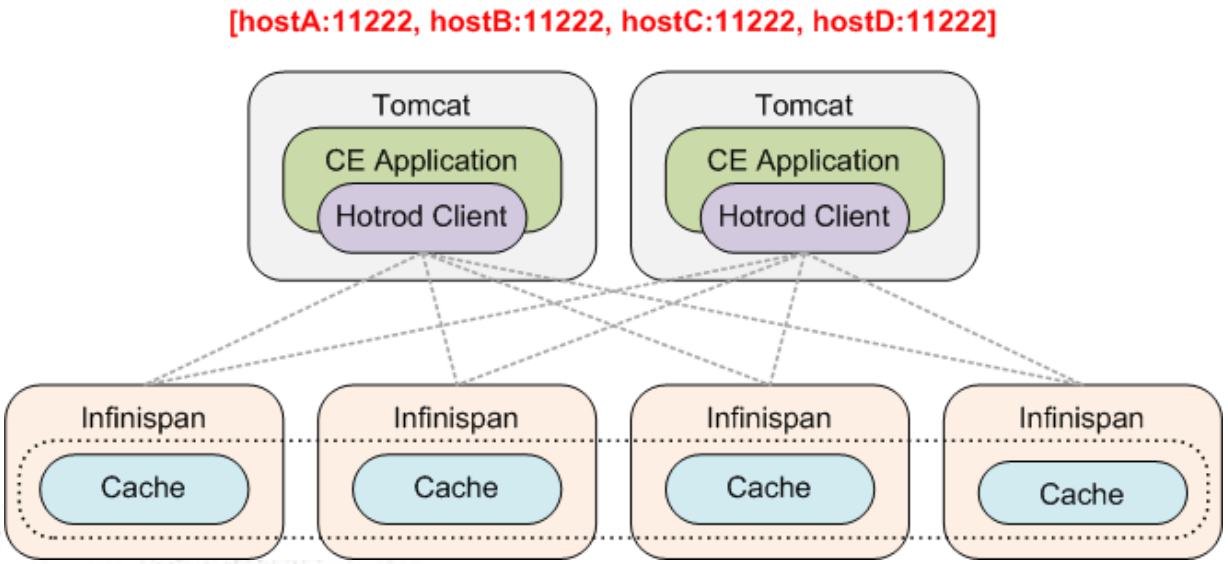
Hotrod Client Failure Detection

Unable to recover from cluster splits



Hotrod Client Failure Detection

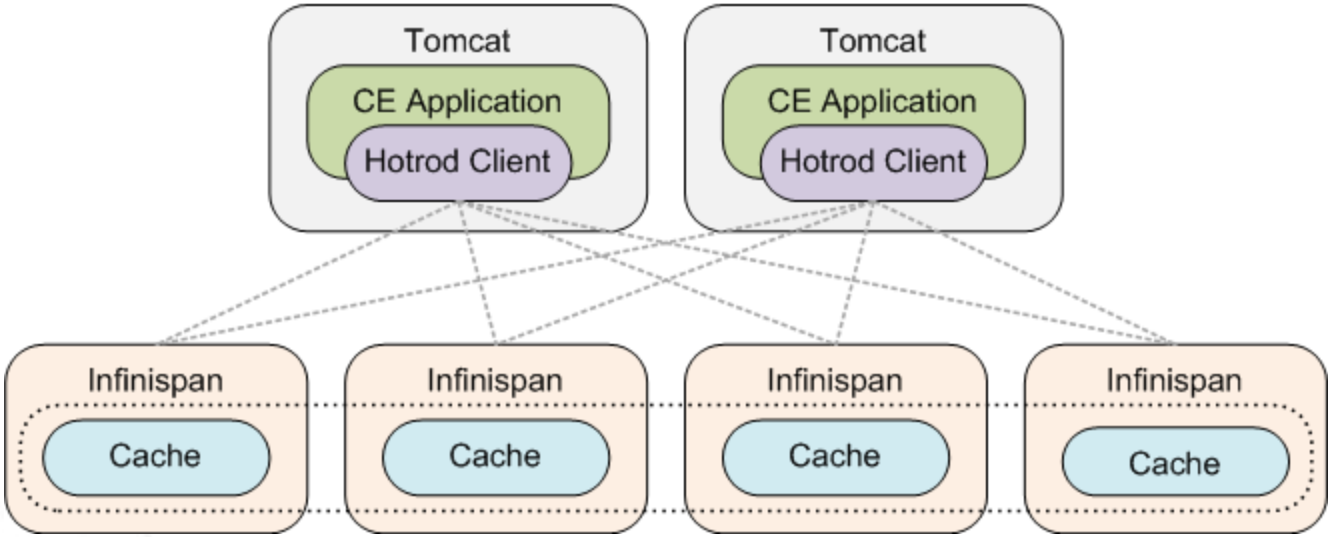
- New servers only added to ___hotRodTopologyCache on start-up
- Restart required to re-establish client topology view



Hotrod Server Cluster Meltdown

Static Server List from Configuration
infinispan.client.hotrod.server_list [hostA:11222, hostB:11222]

[hostA:11222, hostB:11222, hostC:11222, hostD:11222]

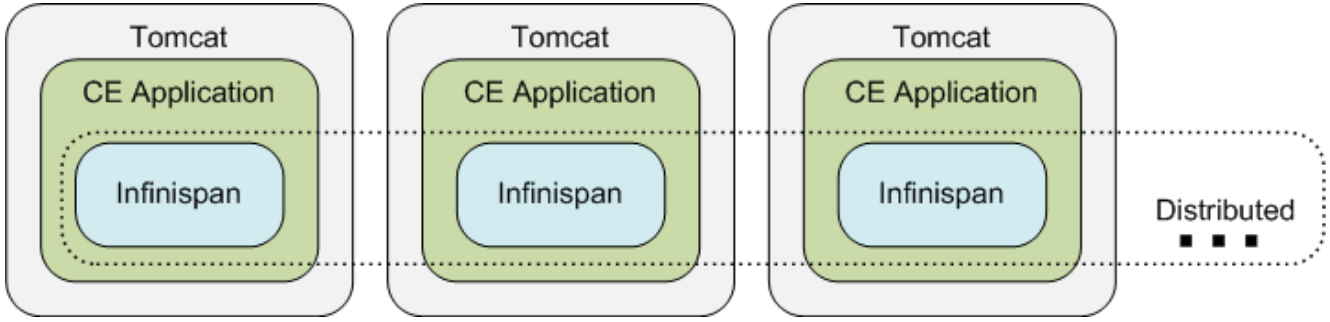


Hotrod Server Cluster Meltdown

- Clients can’t start without an available server
- Static Configuration is only read once
- To restart client-server communications either
  - Restart last “known” server
  - Restart the client

Change of tack

- Hotrod abandoned, for now
  - Data distribution
  - Concurrent start up
  - Failure detection
  - Unacceptable for this customer
- Enter the classic embedded approach



- How did we get this to work...

Dynamic Scaling

- Unpredictable under heavy load, writers blocked
  - Unacceptable waits for this system

    `<hash numOwners="2" rehashEnabled="false" />`
  - Accept some data loss during a leave / join
- Chunked rehashing / state transfer (5.1)
  - https://issues.jboss.org/browse/ISPN-284
- Non-blocking state transfer
  - https://issues.jboss.org/browse/ISPN-1424
- Manual rehashing
  - https://issues.jboss.org/browse/ISPN-1394

Cache Entry Size

- Average cache entry ~6K
  - 1 million entries = 6GB
  - Hotrod stores serialized entries by default
- JBoss Marshalling
  - Default Infinispan mechanism
  - Get reference from ComponentRegistry
- JBoss Serialization
  - Quick, easy to implement

Compression Considerations

- Trade
  - Capacity in JVM vs Serialization Overhead
- Suitability
  - Assess on a cache by cache basis
  - Very high access is probably too expensive
- Average 6K reduced to 1K

Advanced Cache Tuning

cache.getAdvancedCache.withFlags(Flag... flags)

- **Flag.SKIP_REMOTE_LOOKUP**
  - Prevents remote gets being run for an update `put(K key, V value)`

~~`DistributionInterceptor.remoteGetBeforeWrite()`~~

`DistributionInterceptor.handleWriteCommand()`

`DistributionInterceptor.visitPutKeyValueCommand()`

  - We don’t need to return the previous cache entry value

JGroups

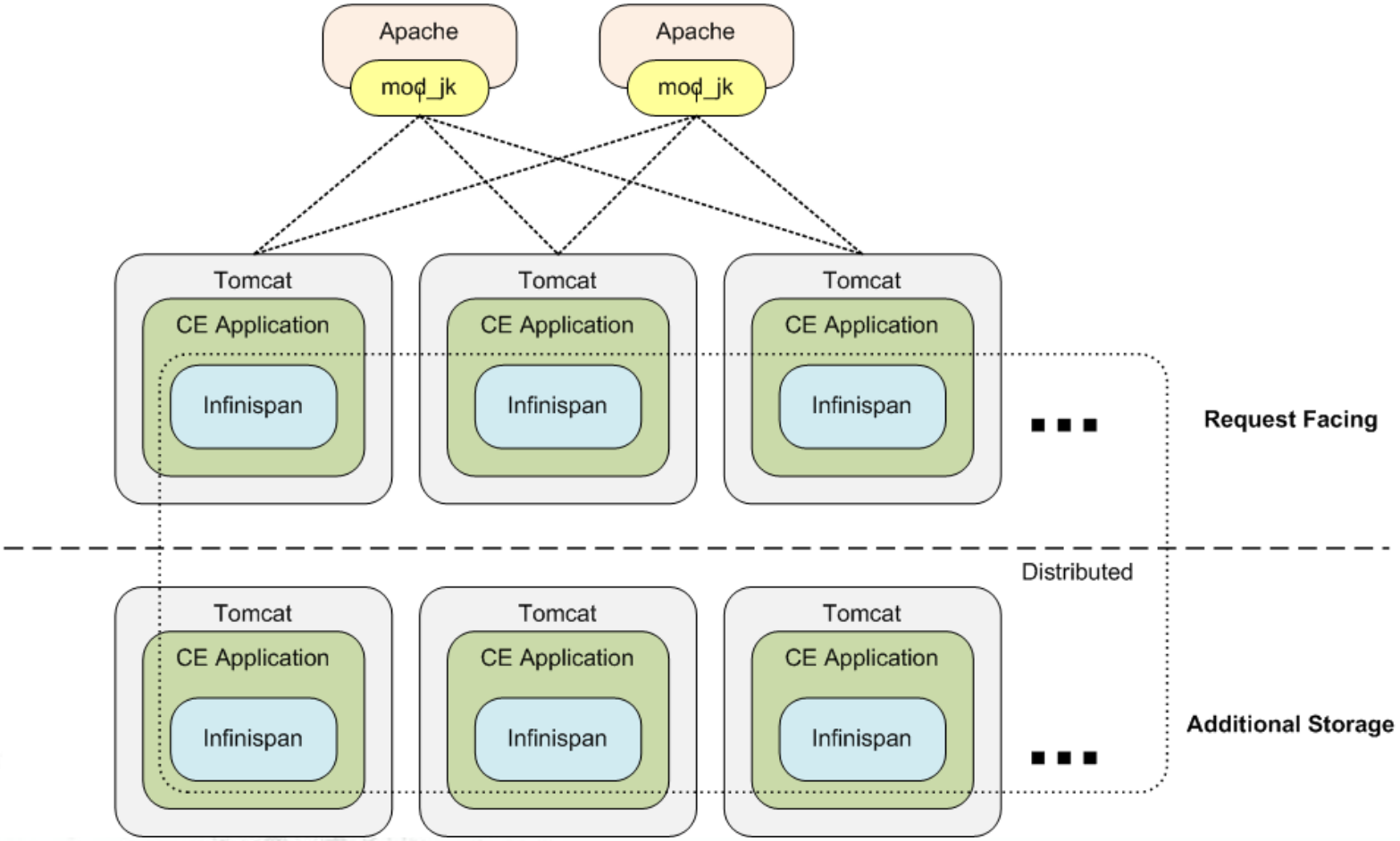
- UDP out-performed TCP (for us)
- Discovery
  - For a cold, full cluster start-up avoid split brain / merge scenarios

```
<PING timeout="3000" num_initial_members="10"/>
```

- Heartbeat
  - Ensure failure detection is configured appropriately

```
<FD_ALL interval="3000" timeout="10000"/>
```

Extending Embedded



Current Production System

- Over 20 nodes
  - 8 Request facing, remainder storage only
- Over 15 million entries
  - 7.5 million unique
  - 20GB cached data
  - Nothing is evicted before natural expiration
- 5GB JVM Heap, 3-4 second GC pauses
- 30% reduction in response times

Summary

- Don’t compromise on the benchmarking
  - Understand your cached data profile
  - Functional testing is NOT sufficient
  - Monitoring and Analysis is essential
- Tune Virtual Nodes for best distribution
- Mitigate memory usage of embedded cache
  - Consider compressing embedded cache entries
  - Non request facing storage nodes
- Distributed Infinispan out performs EhCache
- Don’t rule Hotrod out
  - Not acceptable for this customer
  - Many improvements and bug fixes

Part 2 – Green Field SLA's

New Pricing Engine

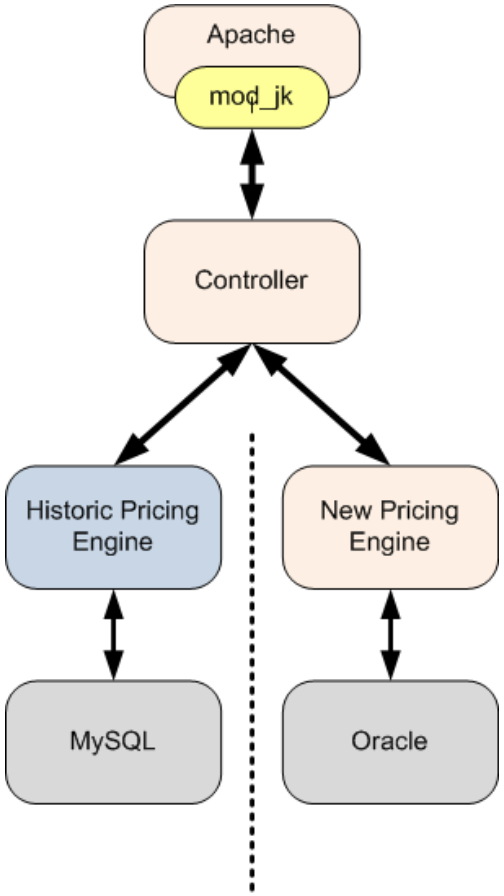
- Tomcat
- Spring & Grails
- Infinispan
- Oracle RAC
- Apache load-balancer / mod_jk

Historical Pricing Engine

- EhCache
- MySQL
- 2 second full Paris Query

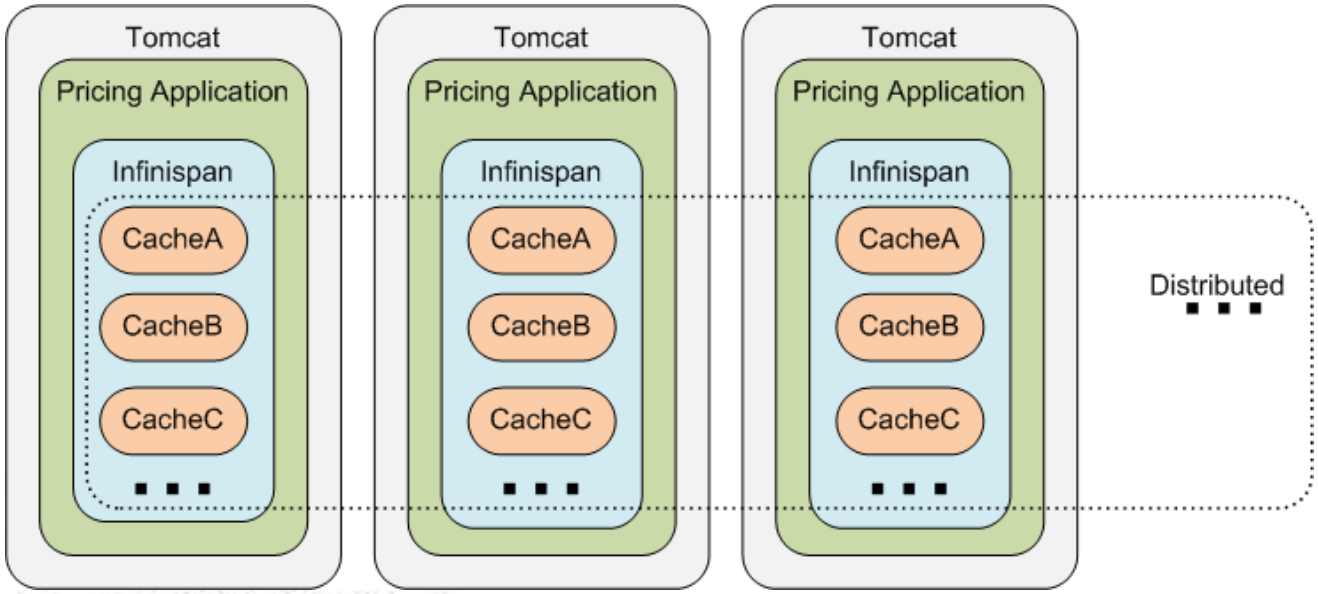
Logical View

- New Pricing Engine
  - Side by side rollout
  - Controller determines where to send requests and aggregates results
  - NOT Hibernate 2LC
  - Spring Interceptors containing logic to check / update cache wrap calls to DB that extract and generate cache entries



Proposed Caching

- Everything distributed
  - It worked before so we just turn in on, right?

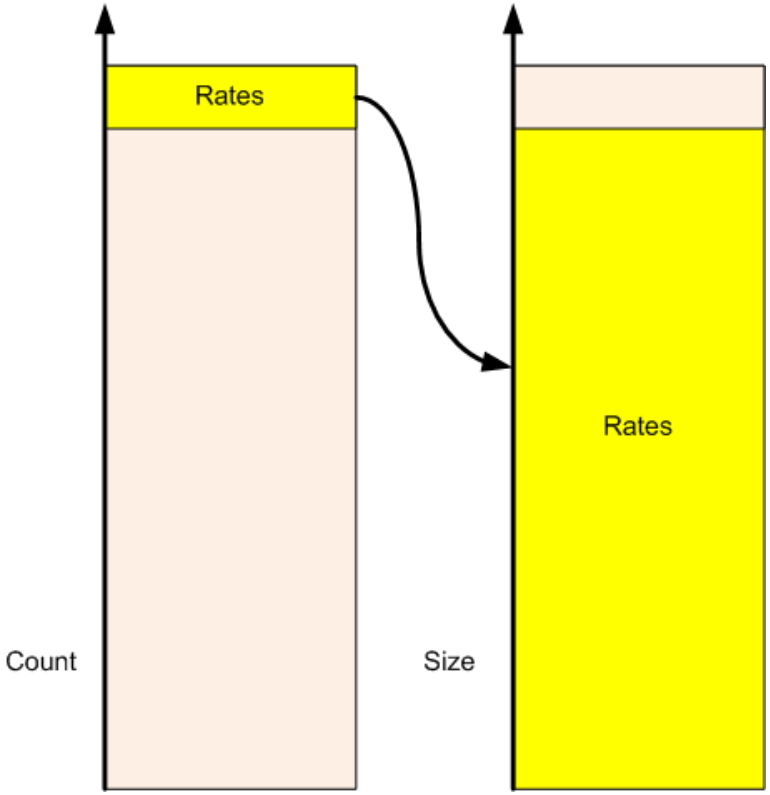


The Pain

- Distributed Mode
  - Network saturation on 1Gb switch (125MB/second) under load
  - Contention in org.jgroups
- Performance SLA's
  - Caching data in Local mode required 14G heap & 20 second GC pauses
- Aggressive rollout strategy
  - Struggling at low user load

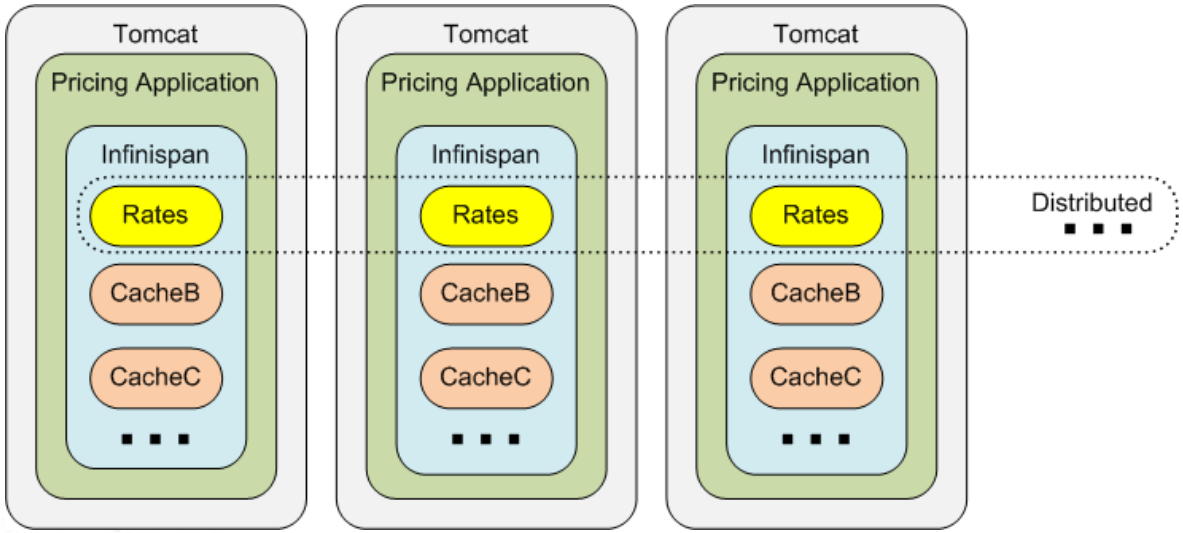
Cache Analysis

- Eclipse Memory Analyzer
  - Identify cache profile
  - Small subset of elements account for almost all the space
  - Paris “Rates” sizes 20K – 1.6MB
  - Paris search (500 rates records) == 50MB total
  - 1Gb switch max throughput = 125MB/second



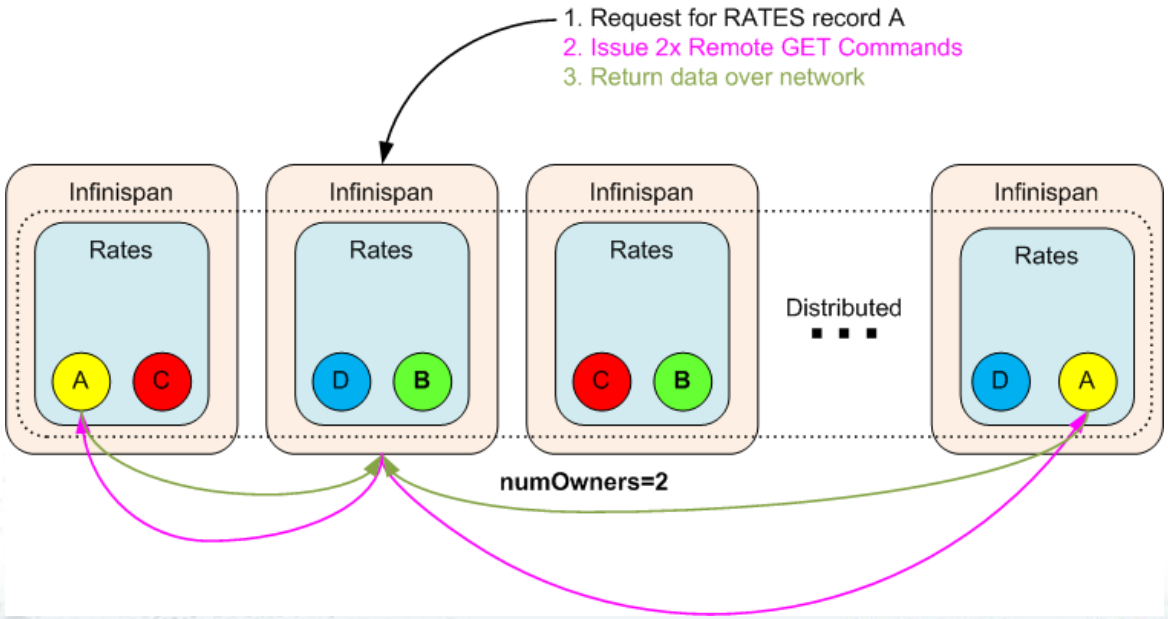
Revised Caching

- Local caching for numerous “small” elements
- Distributed for “large” expensive elements



Distributed Issue

- Here's why normal distributed doesn't work
  - One Paris request requires 500 rates records (50MB)
  - 10 nodes distributed cluster = 1 in 5 chance data is local
  - 80% remote Gets == 40MB network traffic



Options

- Rewrite the application caching logic
  - Significantly reduce the element size
- Run Local caching with oversized heap
  - Daily restart, eliminate full GC pauses
  - Large memory investment and careful management
- Sacrifice caching and hit the DB
  - Hits response times and hammer the database
- Distributed Execution?
  - Send a task to the data and extract just what you need

Change in Psychology...

If the mountain will not come to Muhammad, then Muhammad must go to the mountain

Distributed Execution

- DefaultExecutorService
  - http://docs.jboss.org/infinispan/5.1/apidocs/org/infinispan/distexec/DefaultExecutorService.html
- Create the Distributed Execution Service to run on the cache node specified

```
public DefaultExecutorService(Cache masterCacheNode)
```

- Run task on primary owner of Key input

```
public Future<T> submit(Callable<T> task, K... input)
```

  - Resolve primary owner of Key then either
    - Run locally
    - Issue a remote command and run on the owning node

Pricing Controller

- Callable task
  - Contains code to
    - Grab reference to local Spring Context
    - Load required beans
    - Spring interceptor checks cache at the owning node (local get)
    - If not found then goto database, retrieve and update cache
    - Extract pricing based on request criteria
    - Return results

Existing Code

Pricing Controller

- Create a new DefaultExecutorService
  - Create callable tasks required to satisfy request
  - Issue callable tasks concurrently

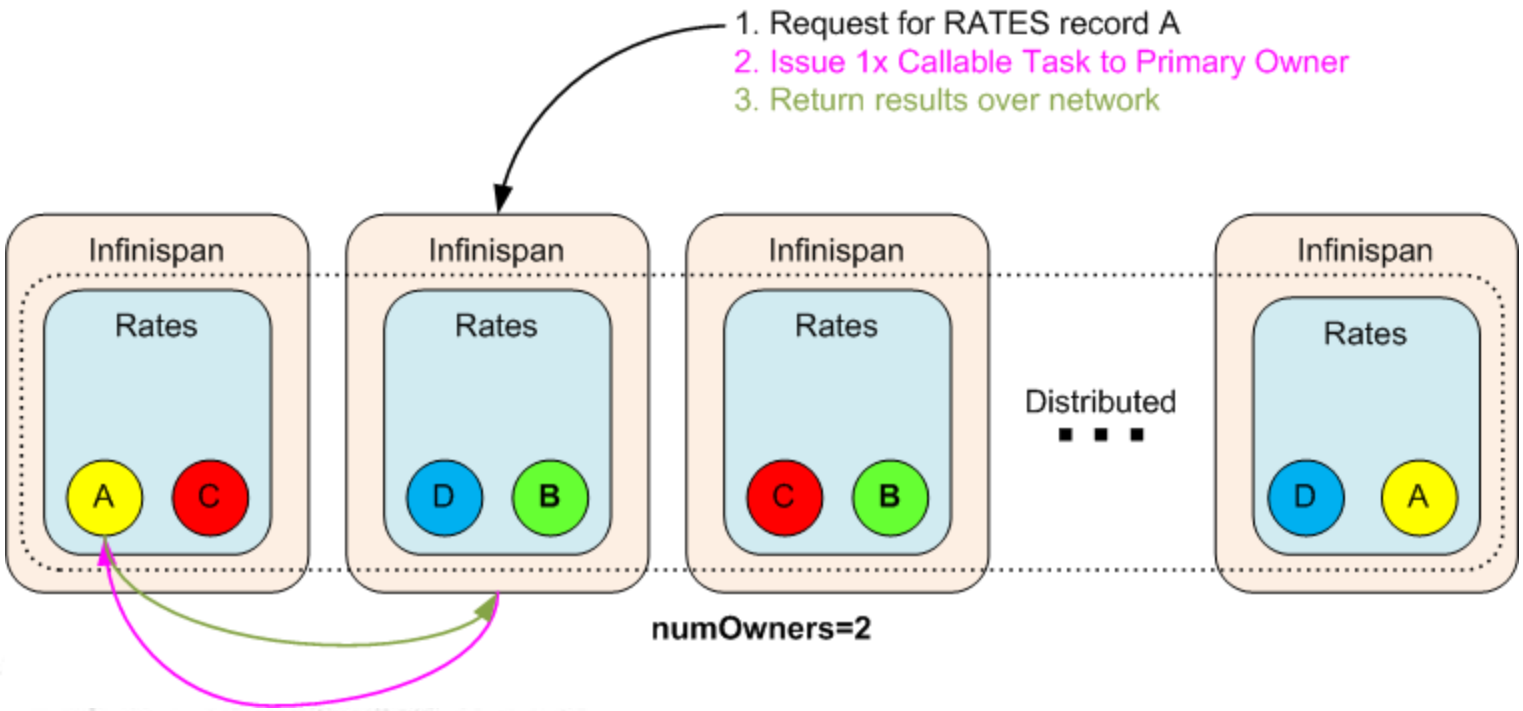
```
while (moreKeys) {
   Callable<T> callable = new MyCallable<T>(...);
   Future<T> future = distributedExecutorService.submit(callable, key);
   ...
}
```

  - Collate results and assemble response

```
while (moreFutures) {
   T result = future.get();
}
```

Distributed Execution

- Only the relevant information from the cache entry is returned



Results

- Latency – Paris search
  - Historic Engine 2 seconds
  - Dist-Exec 200ms
- JVM
  - 5GB Heap
  - 3-4 second pauses

Limitations

- Failover
  - Task sent to primary owner only
  - https://community.jboss.org/wiki/Infinispan60-DistributedExecutionEnhancements
  - Handle failures yourself
- Hotrod not supported
  - This would be fantastic!
  - https://issues.jboss.org/browse/ISPN-1094
- Both in 6.0?

Summary

- Analysis and re-design of cached data
- Accessing large data sets requires an alternative access pattern
- Dramatically reduced latency
  - Parallel execution
  - Fraction of data transferred across the wire
- Execution failures must be handled by application code, at the moment...

Thanks for Listening!

Any Questions?