Infinispan’s Hot Rod Protocol

Galder Zamarreño
Senior Software Engineer, Red Hat
21st June 2010
Who is Galder?

• Core R&D engineer on Infinispan and JBoss Cache
• Contributor and committer on JBoss AS, Hibernate, JGroups, JBoss Portal,...etc
Agenda

• Infinispan peer-to-peer vs Infinispan client-server mode
• What is Hot Rod
• The motivations behind Hot Rod
• Hot Rod implementations and sample code
• Infinispan server comparison
• The path ahead for Hot Rod
• Demo
Infinispan Peer-To-Peer

- Infinispan is an in-memory distributed data grid
- Traditionally, deployed in peer-to-peer (p2p) mode
Infinispan Client-Server

- Sometimes client-server makes more sense
- E.g., access from non-JVM environment
- No Infinispan running on client
Infinispan Client-Server

- P2P data grids do not get along with elastic application tiers
Infinispan Client-Server

- Elastic application tiers work better with client-server
Infinispan Client-Server

- Multiple applications with data storage needs
- Starting a data grid per app is wasteful
Infinispan Client-Server

- Data service tier
- Keep a pool of data grid nodes as shared storage tier
Infinispan Client-Server

- More examples:
- Independent tier management
- E.g., upgrading AS without bringing down DB
- Contrasting JVM tuning needs - CPU vs Memory
- Security
Client-Server with Memcached

Server crashes, data is gone

Adding server requires manual update of server list

Static list of servers
Client-Server with Infinispan Memcached

- Infinispan Memcached
- Servers are clustered
- Server crashes but data survives
- Still, static list of servers
- Adding a new server still requires manual server list update
Client-Server with Infinispan Memcached

When servers are distributed, requests can land anywhere.
What is Hot Rod?

- Hot Rod is Infinispan’s binary client-server protocol
- Protocol designed for smart clients, which have the ability to:
  - Load balance and failover dynamically
  - Smartly route requests
Client Server with Hot Rod

- Infinispan Hot Rod
- Dynamic list of servers
- If Server crashes, data survives and list of servers is updated dynamically
- After adding a new server, server list updated automatically

Servers are clustered
Client Server with Hot Rod

When servers are distributed, clients can smartly route requests.

(1) get(car)
(2) ferrari

Infinispan Hot Rod
K=car, V=ferrari

Infinispan Hot Rod
K=car, V=ferrari

App

Infinispan Hot Rod

The Hot Rod Protocol

- Transmitted keys and values treated as byte[]
- To ensure platform neutral behaviour
- Each operation prepended with cache name
- Basic operations:
  - put, get, remove, containsKey, putIfAbsent, replace, clear
  - stats, ping
Data Consistency

• Concurrently accessed structures can suffer data consistency issue
• Normally solved with JTA
• No JTA in Hot Rod (yet)
• Versioned API as solution
Data Consistency Problem

(1) Show account balance

(2) get(balance)

(3) 500

(4) Show account balance

(5) get(balance)

Infinispan
Hot Rod

K=balance,
V=500

ATM 1

ATM 2

(6) 500
Data Consistency Problem

(1) Retrieve 400

ATM 1

(2) put(balance, 100)

Infinispan Hot Rod

K=balance, V=500

K=balance, V=100

ATM 2

(1) Retrieve 200

ATM 1

(2) put(balance, 300)

Infinispan Hot Rod

K=balance, V=100

K=balance, V=300
Data Consistency in P2P

(1) Retrieve 400

ATM 1

(2) replace(balance, 500, 100)

Embedded Infinispan

K=balance, V=500

K=balance, V=100

ATM 2

JVM

(2) replace(balance, 500, 300)

Embedded Infinispan

K=balance, V=100

ATM 1

JVM

(1) Retrieve 200

ATM 2

Error, old value != 500

JVM
Hot Rod Versioned API

(1) Show account balance

ATM 1

(2) getVersioned (balance)

(3) 500, 1

Infinispan Hot Rod

K=balance, V=500, Version=1

(4) Show account balance

ATM 2

(5) getVersioned (balance)

(6) 500, 1
Hot Rod Versioned API

(1) Retrieve 400

ATM 1

(2) replaceIfUnmodified (balance, 100, 1)

Infinispan Hot Rod

K=balance, V=500, Version=1

K=balance, V=100, Version=2

ATM 1

(2) replaceIfUnmodified (balance, 300, 1)

Infinispan Hot Rod

K=balance, V=100, Version=2

ATM 2

Error, version numbers do not match

(1) Retrieve 200
Hot Rod Client Intelligence

• Different client intelligence levels supported:
  • Basic clients
  • Topology-aware clients
  • Hash-distribution-aware clients
Infinispan Hash Functions

- Infinispan uses language independent hash functions
- Used for smart routing
- Enables smart client implementations in any language
- So far, MurmurHash 2.0 implemented
Topology Information Delivery

Client

topology: id=1,[nodeA:11311]

Infinispan
Hot Rod

K=car,
V=ferrari

topology: id=1, [nodeA:11311]
Topography Information Delivery

1. Start Hot Rod server in nodeB

2. Get(car), id=1, [nodeA:11311, nodeB:11311]

3. Ferrari, id=2, [nodeA:11311, nodeB:11311]

Infinispan Hot Rod

- K=car,
  V=ferrari
  topology: id=2,
  [nodeA:11311, nodeB:11311]
Hot Rod Implementations

- Server implementation included in 4.1.0.Beta2
  - Uses high performance Netty socket framework
  - Start via script: `startServer.sh -r hotrod`
- Java client reference implementation also available
  - Supports all client intelligence levels
- Volunteers for writing clients in other languages welcomed :)  
  - If interested, join us at the Cloud Hackfest!
Hot Rod Client Basic API

//API entry point, by default it connects to localhost:11311
CacheContainer cacheContainer = new RemoteCacheManager();

//obtain a handle to the remote default cache
Cache<String, String> cache = cacheContainer.getCache();

//now add something to the cache and make sure it is there
cache.put("car", "ferrari");
assert cache.get("car").equals("ferrari");

//remove the data
cache.remove("car");
assert !cache.containsKey("car") : "Value must have been removed!";
Hot Rod Client Versioned API

//API entry point, by default it connects to localhost:11311
CacheContainer cacheContainer = new RemoteCacheManager();

//obtain a handle to the remote default cache
RemoteCache<String, String> remoteCache = cacheContainer.getCache();

//put something in the cache
remoteCache.put("car", "ferrari");

//retrieve the value and the version
RemoteCache.VersionedValue value = remoteCache.getVersioned("car");

//replace it with a new value passing the version read
assert remoteCache.replace("car", "mclaren", value.getVersion());
# Infinispan Servers Comparison

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Client Availability</th>
<th>Clustered</th>
<th>Smart Routing</th>
<th>Load Balancing / Failover</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hot Rod</td>
<td>Binary</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, dynamic via Hot Rod client</td>
</tr>
<tr>
<td>Infinispan Memcached</td>
<td>Text</td>
<td>Yes</td>
<td>No</td>
<td>Only with predefined list of servers</td>
</tr>
<tr>
<td>Infinispan REST</td>
<td>Text</td>
<td>Yes</td>
<td>No</td>
<td>Any Http Load Balancer</td>
</tr>
</tbody>
</table>
The path ahead for Hot Rod

• Within Hot Rod:
  • Clients in other languages
  • Querying
  • Event handling...
• Submit protocol to a standards body (maybe)
Hot Rod as base for new functionality
Demo
Summary

• Infinispan client-server architectures are needed
• Hot Rod is Infinispan’s binary client-server protocol
• Designed for load balancing, failover and smart routing
• Server and java client available now
• We need your help to build more clients!
• Hot Rod as foundation for interesting new functionality
Questions?

• Project: www.infinispan.org
• Blog: blog.infinispan.org
• Twitter:
  • @infinispan, @galderz
  • #infinispan #judcon
• Join us at the Cloud Hackfest!!!
• JBoss Asylum Podcast recording - panel discussion
  • Tonight, 8.30pm community room
Learn more about Infinispan!

- **Storing Data on Cloud Infrastructure in a Scalable, Durable Manner** - Wed 23rd
- **Using Infinispan for High Availability, Load Balancing, & Extreme Performance** - Thu, 24th
- **How to Stop Worrying & Start Caching in Java** - Thu 24th
- **Why RESTful Design for Cloud is Best** - Fri 25th