Beyond Peer-to-Peer Data Grids with Infinispan Servers

Galder Zamarreño
Senior Engineer, Red Hat
4th October 2010
Who is Galder?

- R&D engineer (Red Hat Inc):
  - Infinispan developer
  - JBoss Cache developer
- Contributor and committer:
  - JBoss AS, Hibernate, JGroups, JBoss Portal,...etc
- Blog: zamarreno.com
- Twitter: @galderz
Agenda

- Introduction to Infinispan
- Peer-to-peer (P2P) data grids vs client-server
- Infinispan server comparison
- The path ahead for Infinispan servers
- Demo
What is Infinispan?

- A data grid platform which is:
  - Open Source (LGPL)
  - In-memory
  - Distributed
  - Elastic
  - Highly Available
Brief feature overview

- Seamless data distribution, linearly scalable data structures
- Implicit eviction
- Write through and write behind to persistent storage
- JTA and XA transactions
- Listeners and notifications
- Querying and indexing
- Alternative JPA-like API
- JMX management and GUI console
- Multiple endpoints for remote invocations...
Peer-to-Peer Setup
Non-JVM access
Elasticity problems with P2P
Client-Server brings Elasticity

Diagram showing the transition from one JVM to multiple JVMs for improved scalability and elasticity.
Data Grid per Application?
Shared Data Grid
Independent Tier Management

- Independently deploy new app version
- Security
- Incompatible JVM tuning requirements
Infinispan Server Modules

- Protocols supported in 4.1:
  - REST
  - Memcached
  - Hot Rod
  - Websocket
REST Server

- RESTful HTTP access
- HTTP PUT/POST to store
- HTTP GET to retrieve
- Available since 4.0
  - In ‘all’ distribution
# Sample python code using the standard http lib only
#

import httplib

#putting data in
conn = httplib.HTTPConnection("localhost:8080")
data = "SOME DATA HERE !" #could be string, or a file...
conn.request("POST", "/infinispan/rest/Bucket/0", data, 
    {"Content-Type": "text/plain"})
response = conn.getresponse()
print response.status

#getting data out
import httplib
conn = httplib.HTTPConnection("localhost:8080")
conn.request("GET", "/infinispan/rest/Bucket/0")
response = conn.getresponse()
print response.status
print response.read()
Memcached not good enough
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
Routing not so smart

When servers are distributed, requests can land anywhere.
Essential Information

- To run, execute:
  - `startServer.sh -r memcached`
- New in 4.1
- Only text protocol supported
- Works with any Memcached client
Hot Rod to the rescue!

- What is Hot Rod?
  - Wire protocol for client-server communications
  - Open and language independent
  - Built-in dynamic failover and load balancing
  - Smart routing
Dynamic routing with Hot Rod

- Infinispan Hot Rod
- Hot Rod Protocol
- App
- Servers are clustered
- If Server crashes, data survives and list of servers is updated dynamically
- After adding a new server, server list updated automatically
Smart routing with Hot Rod

When servers are distributed, clients can smartly route requests.

1. get(car)
2. bmw
Essential Information

- To run:
  - `startServer.sh -r hotrod`
- New in 4.1
- Only a Java client available:
  - Supports smart routing and dynamic load balancing
Java Hot Rod Client

//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", "bmw");
assert cache.get("car").equals("bmw");

//remove the data
cache.remove("car");
assert !cache.containsKey("car") : "Value must have been removed!";
Websocket Server

- Exposes Infinispan Cache instance over Websocket
- To run it:
  - `startServer.sh -r websocket`
- Accessible via Javascript API
Server Comparison

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Client Libraries</th>
<th>Clustered</th>
<th>Smart Routing</th>
<th>Load balancing / Failover</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST</td>
<td>Text</td>
<td>N/A</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Any HTTP load balancer</td>
</tr>
<tr>
<td>Memcached</td>
<td>Text</td>
<td>Plenty</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Only with predefined server list</td>
</tr>
<tr>
<td>Hot Rod</td>
<td>Binary</td>
<td>Currently only Java</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Dynamic</td>
</tr>
<tr>
<td>Websocket</td>
<td>Text</td>
<td>Javascript</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Any HTTP load balancer</td>
</tr>
</tbody>
</table>
The path ahead

- Hot Rod improvements:
  - Remote querying
  - Event handling
- Submit Hot Rod protocol to standards body (maybe)
- Others:
  - Memcached binary protocol won’t be implemented
Prototype Hot Standby Demo

(*) Embedded Tomcat running within Arquillian
Summary

• Accessing data grids in client-server mode makes sense
• Infinispan 4.1 comes with a range of server modules
• Each server fits one type of use case
• We need your help to build more Hot Rod clients!
Questions?

infinispan.org
blog.infinispan.org
twitter.com/infinispan
#infinispan