# PojoCache: Cluster Your POJOs With Annotations

Ben Wang, Ph. D., Lead PojoCache
&
Brian Stansberry, Lead AS Clustering
JBoss, Inc.



## Topic

- What is JBoss Cache?
- Problems solved by PojoCache
- PojoCache annotations
- Usage in JEE
- Performance numbers
- Conclusion



## What is JBoss Cache?

- A replicated, transactional, and fine-grained cache system
- Two modules
  - ✓ TreeCache – plain cache
  - ✓ PojoCache – POJO (plain old Java object) cache
- TreeCache
  - ✓ Stores and replicates values from a tree structure (hence the name)
  - ✓ Each value is associated with a path and key
- PojoCache (formerly known as TreeCacheAop)
  - ✓ Object-oriented fine-grained cache



## Problems Solved by PojoCache

- Automatic fine-grained replication by operating on POJO directly
  - ✓ Transparency
  - ✓ Performance

```
Person joe = (Person) cache.getObject(fqn, key);
joe.setAge(51); // Will trigger replication automatically
// No need to do another put(fqn, key, joe) to trigger replication!!
```

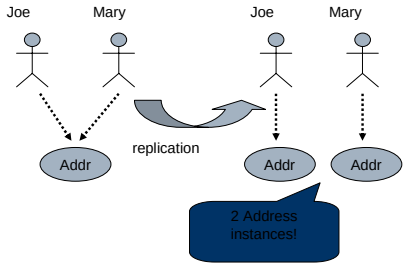


## Problems Solved by PojoCache

- Replication and persistence don't preserve object relationships
  - ✓ Object sharing (or multiple references)
    - Husband (joe) and wife (mary) share the same address
  - ✓ Circular reference
    - Father and son relationship
  - ✓ Multiple keys
    - Item belongs to multiple category keys



## Object Splitting During Replication



## Use of Annotation by JBoss Aop

**PojoCacheable.java**

```
package org.jboss.cache.aop.annotation;
// Annotation marker interface
public @interface PojoCacheable {}
```

**Person.java**

```
@org.jboss.cache.aop.annotation.PojoCacheable
public class Person {
  String name;
  int age;
  Address addr;
  int visit;
...
}
```

**jboss-aop.xml**

```
<aop>
<prepare expr="field(*
$instanceof{org.jboss.cache.aop.annotation.PojoCacheable}->*)" />
</aop>
```



## PojoCache Annotation

- Class level
  - ✓ @PojoCacheable
    - Declare that this POJO will be instrumented by the underlying JBoss Aop framework
  - ✓ @InstanceOfPojoCacheable
    - Same as above except all sub-classes will be "instrumented" as well.
- Field level
  - ✓ @Serializable
    - Will treat this field as Serializable even it is PojoCacheable.
  - ✓ @Transient
    - Will treat field like modifier "transient". No field replication takes place.



## Instrumenting Classes

- PojoCache utilizes JBoss AOP's Dynamic AOP feature
- Classes that will be fine grain replicated must be instrumented (bytecode enhancement)
- Enhancement can be done at
  - ✓ Compile time
    - Use JBoss AOP's aopc Ant task
  - ✓ Load time
    - Need special bootstrap classloader
    - See Chapter 10 of JBoss AOP Ref Guide



## Standalone PojoCache Code Snippet

```
// Configuring the cache via xml
PojoCache cache = new PojoCache();
PropertyConfigurator configurator = new PropertyConfigurator();
configurator.config(cache, "META-INF/replSync-serivce.xml");
cache.start();

// Putting an object under cache management
Person joe = new Person("Joe Black", 51);
cache.putObject("/person/joe", joe);

// Fine grained replication here. Only the age field is replicated!
joe.setAge(61);

// Remove the object from cache management
cache.removeObject("/person/joe");

cache.stop();
```



## JEE Uses of PojoCache

- HttpSession replication
- EJB3 Stateful Session Beans



## HttpSession Replication

- Replication Granularity
  - ✓ How much gets replicated when session is changed?
    - SESSION – entire session object
    - ATTRIBUTE – only the changed attributes
    - FIELD – individual fields within changed attributes
      - ✓ Uses PojoCache under the covers
      - ✓ New in JBoss AS 4.0.4

## Configuration

- Replication granularity is configured per webapp in the jboss-web.xml deployment descriptor

**jboss-web.xml**

```
<jboss-web>
  <replication-config>
    <replication-granularity>FIELD</replication-granularity>
  </replication-config>
</jboss-web>
```

13

## Configuration

- The cache used for session replication is configured in the tc5-cluster.sar/META-INF/jboss-service.xml file
  - ✓ Make sure "UseMarshalling" and "InactiveOnStartup" are "true"

**tc5-cluster.sar/META-INF/jboss-service.xml**

```
<mbean code="org.jboss.cache.aop.TreeCacheAop"
name="jboss.cache:service=TomcatClusteringCache">
  .....
  <attribute name="UseMarshalling">true</attribute>
  <attribute name="InactiveOnStartup">true</attribute>
  .....
</mbean>
```

14

## Annotation of Classes

**Person.java**

```
@org.jboss.cache.aop.annotation.InstanceOfPojoCacheable
public class Person {
  String name;
  int age;
...
}
```

**Address.java**

```
@org.jboss.cache.aop.annotation.PojoCacheable
public class Address {
  String street, city, state;
  int zip;
...
}
```

15

## Servlet Use Example

Object sharing

```
Person user = (Person) session.getAttribute("user");
user.setName("joe"); // Only this field gets replicated.

Person spouse = (Person) session.getAttribute("spouse");
spouse.setAge(41); // Only this field gets replicated.

Address addr = new Address("San Jose", 95123);
user.setAddress(addr);
user.setAddress(addr);  // addr only gets replicated once; shared reference is maintained

// By default, all field changes are replicated in a batch at the end of the request
```

16

## EJB3 Annotations

- Basic EJB3 SFSB Annotations
  - ✓ @Remote
  - ✓ @Stateful
  - ✓ @Remove
  - ✓ Lifecycle callbacks
- Clustering Annotations

17

## Clustering Annotations

- @Clustered
  - ✓ Specifies you want load balancing and failover for your SFSB
  - ✓ Used whether or not you want fine-grained replication
  - ✓ Optional attributes
    - "partition" – identity of the cluster
    - "loadBalancePolicy"
- Fine-grained replication
  - ✓ @PojoCacheable
  - ✓ @Transient
  - ✓ @Serializable

18

## Clustering Annotations

```
import org.jboss.annotation.ejb.Clustered;
import org.jboss.cache.aop.annotation.PojoCacheable;
import org.jboss.ha.framework.interfaces.*;

@Stateful
@Clustered(partition="ShoppingPartition"
   loadBalancePolicy=FirstAvailableIdenticalAllProxies.class)
@PojoCacheable
public class ShoppingCartBean implements ShoppingCart {
        Map items;
        // No replication of processor field
        @Transient CreditCardProcessor processor;
}
```



## Performance Numbers

- PojoCache
  - ✓ PojoCache characteristics
  - ✓ Comparison to TreeCache
- Http session replication
  - ✓ Effect of different replication granularities



## PojoCache performance characteristics

- putObject to attach the POJO to cache is expensive because of initial field mapping
- Subsequent field update/replication is very efficient, since it usually involves only replicating a primitive type
  - ✓ Successive field updates can also be batched as well
- Performance characteristics of PojoCache therefore depend on the POJO lifetime, i.e. how often a new POJO is attached to the cache system



## PojoCache vs. TreeCache

- TreeCache operation has a constant cost, i.e. each put(fqn, key, pojo) performs the same if pojo size is equal.
- PojoCache performance depends on the POJO lifetime, i.e. how often is putObject called to attach a new POJO
  - ✓ Note that putObject with the same POJO is almost a no-op



## PojoCache Performance

Test setup

- Hardware
  - ✓ 4 node cluster
  - ✓ Intel® Pentium® 4 3.0GHz CPU x 2
  - ✓ 4GB memory
  - ✓ 1Gbps Network
- Software
  - ✓ JBoss Cache 1.4.0.Beta, JGroups 2.2.9.1
  - ✓ JDK1.5.0_05, heap size is 512MB
  - ✓ 800 clients total, no sleep between requests



## Test Load Patterns

- Stores a Student POJO with Address and list of Courses

```
@PojoCacheable
public class Student {
  String name;
  int age;
  Address addr;
  List courses; // We change the list size to study repl message size
}
```

```
@PojoCacheable
public class Course {
  String name;
  String instructor;
}
```

## Test Load Patterns

- TreeCache load pattern
  - ✓ cache.put(fqn, key, pojo)
    - Constant cost each time
- PojoCache load pattern
  - ✓ Mix of:
    - cache.putObject(fqn, pojo)
      - ✓ More expensive POJO attachment
    - pojo.getCourses().get(0).setInstructor("Ben Wang")
      - ✓ Efficient fine-grained replication



## Test Load Patterns

- PojoCache load pattern
  - ✓ Has 3 different mixes: 100-0, 10-90, 5-95
    - To study effects of performance wrt pojo lifetime
    - E.g., 10-90 means 10% putObject and 90% field update, etc.



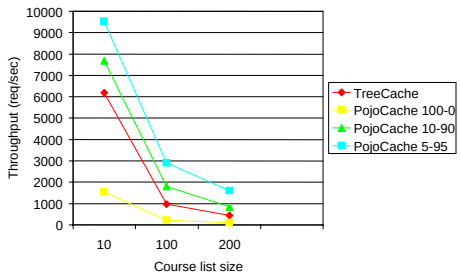
## Message size vs. Course list

- ✓ Uses 3 different course list sizes

| Course list size | Replication size (bytes) |
|---|---|
| 10 | 1.2K |
| 100 | 8.4K |
| 200 | 16.3K |

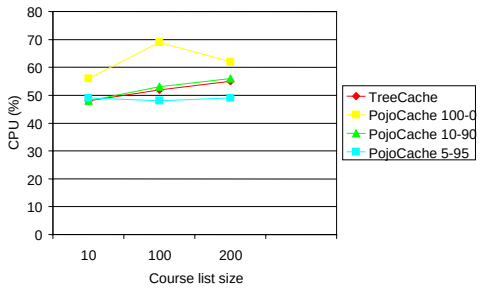


## PojoCache Throughput






## CPU Utilization






## HttpSession Replication

- How much gets replicated when session is changed?
  - ✓ SESSION – entire session object
  - ✓ ATTRIBUTE – only the changed attributes
  - ✓ FIELD – individual fields within changed attributes
    - Uses PojoCache under the covers
- Tradeoff
  - ✓ Size of payload vs. # of change instructions
  - ✓ A change instruction has overhead
    - Changes are batched; 1 replication message per request

## Effect of Replication Granularity

Test setup

- Hardware
  - ✓ Intel® Pentium® 4 3.0GHz CPU x 2
  - ✓ 4GB memory
  - ✓ 1Gbps Network
- Software
  - ✓ JBoss AS 4.0.4.GA, JBoss Cache 1.4.0.Beta, JGroups 2.2.9.1
  - ✓ JDK1.5.0_05, heap size is 756MB
  - ✓ 4 servers in the cluster
  - ✓ 200 clients, no sleep between requests
  - ✓ Sticky sessions



## Effect of Replication Granularity

Test design

- Change a varying number of attributes per request, see how different replication granularities perform
- Session
  - ✓ Session has 10 attributes
  - ✓ Each attribute is a List of 16 elements
  - ✓ List element is a simple JavaBean with a 100 byte "name" (String) and an "age" (int)
  - ✓ Attribute change == change the name of 1 bean in the list

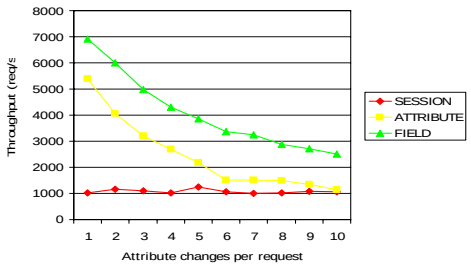


## Message size

| Granularity | Replication size |
|---|---|
| SESSION | ~ 16Kb |
| ATTRIBUTE | ~ 1.6Kb * number of attributes changed |
| FIELD | ~ 100 bytes * number of attributes changed |



## Effect of Replication Granularity






## Conclusion

- PojoCache
  - ✓ Has instantaneous clustering/persistence for POJOs
  - ✓ Fine-grained operation
    - Also can be batched
  - ✓ Works with object graph even when distributed
- Performance
  - ✓ Fine-grained replication can increase your performance throughput when your object size is significant



## Additional info

- Main Wiki Page: http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossCache
- JBoss Cache Home Page: http://labs.jboss.com/portal/index.html?ctrl:id=page.default.info&project=jbosscache (JBoss.org landing page)
- OnJava article: http://www.onjava.com/pub/a/onjava/2005/11/09/jboss-pojo-cache.html
- A recent Wiki: http://wiki.jboss.org/wiki/Wiki.jsp?page=WhatShouldWeExpectOfThePojoCachePerformance