JBoss Cache In-depth

Past, Present, and Future

Manik Surtani
Lead, JBoss Cache
JBoss Inc.

Agenda

- History
- Current architecture and API
- Aspects
  - ✓ Local vs. replicated cache, buddy replication
  - ✓ Locking - Optimistic vs. Pessimistic
  - ✓ Transactional support
  - ✓ Eviction and cache loading
- Pojo Cache
- The future
  - ✓ Grids
  - ✓ API-less model

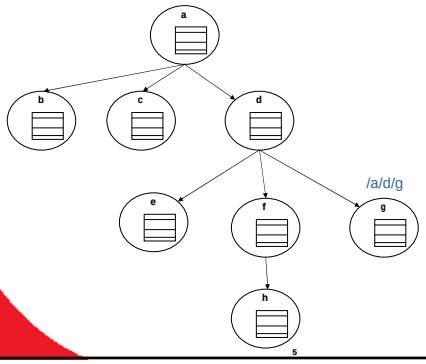
Humble beginnings

- Started as a demonstration of replicating a HashMap using JGroups.
- Added aspects including
  - transaction support
  - persistence and passivation
  - concurrency with both optimistic and pessimistic locking
  - partial replication
  - the list goes on
- Used in a number of open source and commercial products as a mechanism for replicating state as well as a local data cache.

What is JBoss Cache ?

- Two implementations
  - ✓ TreeCache
    - Stores and replicates values in a tree structure
    - Each value is associated with a path and key
  - ✓ PojoCache (fka TreeCacheAop)
    - Manages caching, replication and persistence of Plain Old Java Objects (Pojos)
    - Object Oriented cache
- Standalone or embeddable (MBean)
  - ✓ PojoCache requires JBossAOP libraries
  - ✓ Used in many different containers including
    - JBoss AS, Tomcat
    - BEA WebLogic, IBM WebSphere
    - and others
  - ✓ Used in many standalone environments to cluster custom components

Organising your data



API

- `put(Fqn path, Object key, Object value)`
  - ✓ Adds a key and value to a given node
  - ✓ If the node doesn't exist, it will be created
- `put(Fqn path, Map data)`
  - ✓ Adds a new node to the tree and sets its data. If the node already has data, then the new data will overwrite existing data
- `Object get(Fqn path, Object key)`
  - ✓ Finds a node given its name and returns the value associated with a given key in its data map

API

- `remove(Fqn path)`
  - ✓ Removes a node and its children from the tree
- `remove(Fqn path, Object key)`
  - ✓ Removes key from the node's data map

7

Sample code

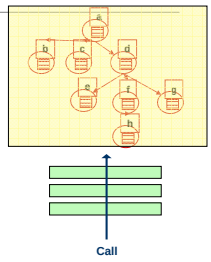
```
try
{
    tx = (UserTransaction) new
InitialContext().lookup("UserTransaction");
    TreeCache c = new TreeCache("test", null, 10000);
    c.setMode(TreeCache.REPL_ASYNC);
    c.startService();
    tx.begin();
    c.put("/a/b/c", "age", new Integer(29));
    c.put("/a/b/c", "age", new Integer(30));
    tx.commit();
    assertEquals(new Integer(30), c.get("/a/b/c", "age"));
}
catch(Throwable t)
{
    if(tx != null) tx.rollback();
}
```

8

Architecture

Based on interceptors
- ✓ Locking
- ✓ Replication
- ✓ Cache loading/cache storing

- Interceptor chain built at startup
  - ✓ Based on config of cache
- Add your own interceptors
  - ✓ Metrics, hit/miss ratio
  - ✓ Auditing



9

Aspects of JBoss Cache

- Local or replicated
- Locking
- Eviction
- Cache loading
- Transactions
- Monitoring

10

Local versus replicated

- LOCAL
  - ✓ Modifications are not replicated
- REPL_ASYNC
  - ✓ Asynchronous replication, on a separate thread
- REPL_SYNC
  - ✓ Synchronous replication, caller blocks until modifications have been applied at every cache in the cluster
- INVALIDATE_ASYNC
  - ✓ Invalidation message sent to remote caches, so remote caches evict the node invalidated. Asynchronous.
- INVALIDATE_SYNC
  - ✓ As above, except the messages are synchronous.

11

Replication with transactions

- If transaction is present, replication occurs at transaction commit time
  - ✓ Multiple changes can be made to the cache within a TX
  - ✓ Single replication message is sent as part of the commit handling
  - ✓ If there is a rollback, we have zero replication cost
- Changes without a transaction are replicated immediately
  - ✓ If there are 1000 changes, there will be 1000 replication messages
  - ✓ Compare to 1000 changes in a TX, 1 replication message

12

Replication vs.Invalidation

- Replication may incur a heavy network overhead penalty since all the data in a node is replicated
- Invalidation incurs a very small network penalty since no data is transmitted with the invalidation message
  - ✓ Only makes sense if the cache does not store the only copy of the data in the system
  - ✓ E.g., if the data is backed up via a shared cache loader
  - ✓ Or the cache is used as a forward-cache, with data originating from another system such as a database

Buddy Replication

- Instead of replicating everything to everyone, we replicate only to N backups
- Example
  - ✓ 10 caches, every cache has, on average, 100MB data
    - Default replication: every cache has 1GB of data
    - BR (where N=1): every cache has 200MB of data
- BR allows us to scale, with memory requirements and network traffic no longer a function of cluster size
- Enabled via simple configuration setting
- New in JBoss Cache 1.4.0

Locking

- Concurrent access is guarded
- Optimistic locking
  - ✓ Workspaces and data versioning, commit fails if data modified by other TX
- Pessimistic locking
  - ✓ Locks at node level
  - ✓ Isolation levels define locking policy
    - NONE
    - READ_UNCOMMITTED
    - READ_COMMITTED
    - REPEATABLE_READ
    - SERIALIZABLE

Eviction

- Elements are evicted from cache
  - ✓ Keeps cache size bounded
- Time- or size-based
- Eviction policy pluggable
  - ✓ Implement your own
- Eviction policies apply to a cache region
  - ✓ /myshop: evict after 10 minutes inactivity
  - ✓ /myshop/products/pricelist: no eviction
  - ✓ /myshop/shoppingCarts: evict after 30 mins inactivity

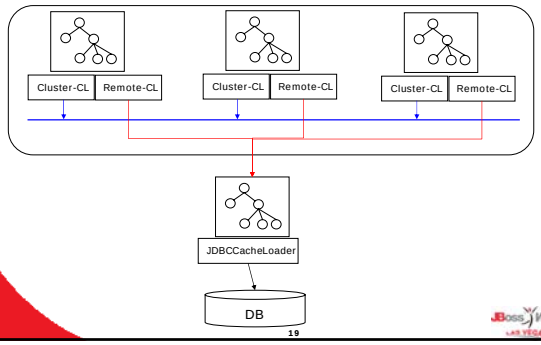
Cache loaders

- Opposite of eviction
- Also pluggable
- Loads elements from store into cache
- Stores elements from cache into store
  - ✓ On put(), or on eviction (passivation/activation)
- Implementations
  - ✓ FileCacheLoader: serializes data to file system
  - ✓ JDBCCacheLoader: DB (Oracle, MySql, MS-SQL server, PostgreSQL tested)
  - ✓ BdbjeCacheLoader: Berkeley DB

Cache loaders

- Hierarchical CacheLoader
  - ✓ Uses TCP/JGroups/RMI to access remote cache
- ClusteredCacheLoader
  - ✓ Uses lookup across cluster to find elements of data
- ChainingCacheLoader
  - ✓ Allows for CacheLoader chaining
  - ✓ ClusteredCacheLoader, followed by RemoteCacheLoader, followed by JDBCCacheLoader

Example of hierarchical setup



19

Transactions

- Transaction support pluggable
  - ✓ Just let JBoss Cache know how to find your TransactionManager
  - ✓ JTA interfaces
- If not defined, JBoss Cache runs without TXs
- Implementations
  - ✓ `JBossTransactionManagerLookup`
  - ✓ `GenericTransactionManagerLookup` (Sun, BEA, IBM, etc.)
  - ✓ Standalone (`DummyTransactionManagerLookup`)
  - ✓ Custom implementations

20

Configuration

- Via XML file
- Same syntax as for JBoss MBeans
  - ✓ But also works standalone, or in other application servers and containers.
- Programmatic access
  - ✓ Setters

21

What is PojoCache ?

- Extends TreeCache. Formerly known as TreeCacheAop
- Manages Pojos rather than key/value pairs
- Pojos are inserted to the cache, cache
  - ✓ keeps track of modifications to Pojos
  - ✓ replicates modifications across cluster (possibly atomically with TXs)
  - ✓ persists modifications to DB (via CacheLoader, if configured), possibly also at TX commit
- We use JBossAOP to instrument Pojos and keep track of state changes

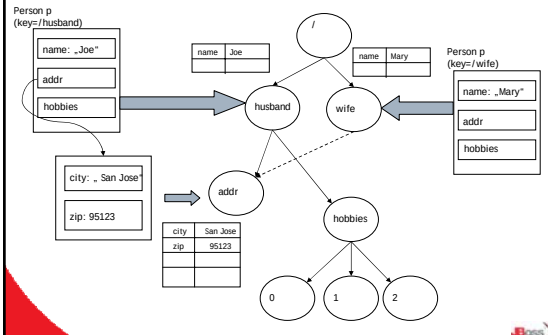
22

API

- `putObject(Fqn path, Object pojo)`
  - ✓ Adds a Pojo, manages it from now on
  - ✓ All modifications are replicated/persisted, with ACIDity if TXs are used
- `Object getObject(Fqn path)`
- `void removeObject(Fqn path)`
  - ✓ Removes an object, PojoCache doesn't manage it anymore
  - ✓ All modifications are written directly into the Pojo, no replication/persistence
- After adding Pojos to cache, Pojos are accessed directly

23

putObject() Mapping



24

How are changes to Pojos detected?

- Tag Pojos with a marker annotation
- Byte code for tagged Pojos are instrumented, either:
  - ✓ At compile-time: aopc
  - ✓ At runtime: loader / java.lang.instrument
- Instrumentation adds an interceptor stack
  - ✓ PojoCache adds a CacheInterceptor
  - ✓ All access to Pojo is through the CacheInterceptor
  - ✓ CacheInterceptor redirects reads/writes to Pojos to PojoCache
  - ✓ Modifications are written back to Pojo at TX commit

Instrumentation of Pojos

- Needed to detect state changes
  - ✓ Online (runtime) or offline (compile-time using aopc)
  - ✓ putObject() adds interceptor to your Pojo's stack
  - ✓ removeObject() removes it again
- JDK 5: use of Java annotations
- JDK 1.4: use of javadoc and annotation compiler

```
@AopMarker // JDK5.0 annotation
public class Person
{
}
```


Sample code

```
PojoCache cache = new PojoCache();
PropertyConfigurator config = new PropertyConfigurator();
config.configure(cache, "META-INF/replySync-service.xml");
cache.startService(); // kick start TreeCacheAop
Person joe = new Person("Joe Black", 30);  // Just a Pojo
cache.putObject("/person/joe", joe); // ask cache to manage joe
joe.setAge(40); // set to cache (will replicate as well)
Person p = cache.getObject("/person/joe"); // An alias for joe
p.getAge(); // Should be 40
cache.removeObject("/person/joe"); // detach from cache
joe.setAge(50); // not replicated, not persisted (not managed)
```


The future

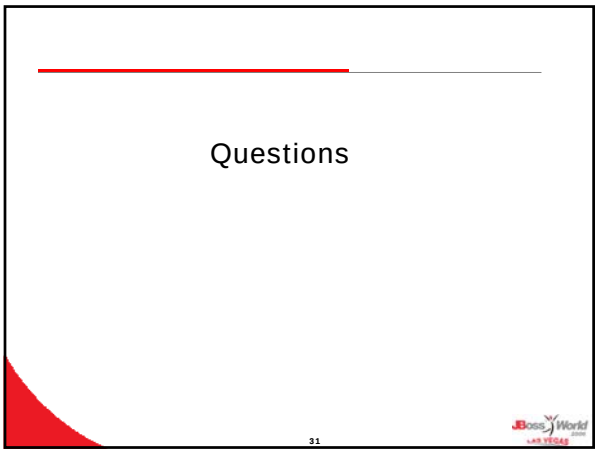
- Grid computing
  - ✓ Partnering with research projects on large data grids
  - ✓ Making use of buddy replication
  - ✓ POCs of distributed data stores
  - ✓ Feeds back into highly scalable and performant clustering capabilities of JBoss Cache
  - ✓ Will enable JBoss Cache to efficiently scale to clusters of over a hundred nodes

The future

- API-less programming model
  - ✓ Make better use of AOP and Java annotations to provide transparent caching
    - With replication, persistence, concurrency, transactions
  - ✓ Based on PojoCache
  - ✓ Completely transparent
    - Absolutely no compile-time dependencies in user code, except on a pojocache-annotations library.
    - No explicit construction of a PojoCache and explicit attachment of Pojos
    - Use of annotations to mark Pojos as cached, replicated, etc.
    - No complex XML configuration or mappings

Links

- http://labs.jboss.org/jbosscache
- http://www.jboss.com/products/jbosscache


Questions
31