Coherence Implementation Patterns

Ben Stopford

The Royal Bank of Scotland
Some Ideas

Nothing More
Why do we use Coherence?

Fast?

Scalable?

Application layer?
Simplifying the Contract
• We don’t want ACID all of the time
• We want to pick the bits we need when we need them
• We want to use the context of our business requirement to work our way around the ones we don’t need.
Version your Objects
Why do we care?

Without versioning it’s a free-for-all.

• What changed?
• Was something overwritten?
• How can you prevent concurrent updates?
• What did the system look like 10 seconds ago.
• How can I provide a consistent view?
• How to I ensure ordering of updates in an asynchronous system?
Versioning your Objects

Versioned Cache

- A.1
- A.2
- B.1
- C.1
- C.2
- D.1
Versioning your Objects

Cache

A.1

A.2

Coherence Trigger

New Version = Old Version + 1 ??
Running a Coherence Filter

Client

cache.entrySet(query)

Node

Node

Node

Node
Using Key-Based Access
Latest / Versioned Pattern

Write

Latest Cache

A
B
C
D

Versioned Cache

A.2
B.1
C.1
C.2
D.1

Coherence Trigger
Latest / Versioned Pattern

Latest Cache Key = [Business Key]
Versioned Cache Key = [Business Key][Version]
Suffers from data duplication
Latest Marker Pattern

- **Write**
- **Versioned Cache**
  - A.1
  - A.L
  - B.L
  - C.1
  - C.L
  - D.L

**Coherence Trigger**

**Well Known Marker Version**
However our trigger can’t use `cache.put()`

Why?
Need to consider the threading model
So we’ll need to use the backing map directly

```java
public void copyObjectToVersionedCacheAddingVersion(MapTrigger.Entry entry) {
    MyValue value = (MyValue)entry.getValue();
    MyKey versionedKey = (MyKey)value.getKey();

    BinaryEntry binary = (BinaryEntry)entry;
    Binary binaryValue = binaryEntry.getBinaryValue();

    Map map = binary.getContext().getBackingMap("VersionedCacheName");
    map.put(toBinary(versionedKey), binaryValue);
}
```
A third approach
The Collections Cache

collectionsCache.put(key, val);

collectionsCache.invoke(key, new LastValueGetter());

...or override backing store
So we have 3 patterns for managing versioning whilst retaining key based access
Using versioning to manage concurrent changes

Multi Version Concurrency Control (MVCC)
Cache

Version 1

Version 2

Coherence Trigger

New Version = Old Version + 1 ??
Concurrent Object Update
(2 Clients update the same object at the same time)
Concurrent Object Update

(Client2 fails to update dirty object)

Client1

Versioned Cache

Client2

A.1

X

A.2
Concurrent Object Update

(Client 2 updates clean object)

Client1

Versioned Cache

Client2

A.1

A.2

A.3
So a concurrent update results in an error and must be retried.
What’s going to happen if we are using putAll?
Reliable PutAll

We want putAll to tell us which objects failed the write process
Reliable PutAll

Invocable:
- Split keys by member
- Send appropriate values to each member
- Collect any exceptions returned

Invocable:
- Write entries to backing map (we use an EP for this)
This gives us a reliable mechanism for knowing what worked and what failed
Synthesising Transactionality
The Fat Object Method
The Single Entry Point Method
(objects are stored separately)

Collocate with key association
All writes synchronize on the primary object.
All **reads** synchronize on the primary object.
Writing Orphaned Objects

Write read entry point object last

Write orphaned objects first

This mechanism is subtly flawed
Reading several objects as an atomic unit

aka Joins
The trivial approach to joins

- Get Cost Centers
- Get Ledger Books
- Get Source Books
- Get Transactions
- Get MTMs
- Get Legs
- Get Cost Centers

Network

Time
Server Side, Sharded Joins

Use KeyAssociation to keep related entities together
Server Side, Sharded Joins

Aggregator joins data across cluster
So we have a set of mechanisms for reading and writing groups of related objects.
Cluster Singleton Service
A service that automatically restarts after failure
A service that automatically restarts after failure
What is the cluster singleton good for

- Adding indexes
- Loading data
- Keeping data up to date
- Updating cluster time
- You can probably think of a bunch of others yourselves.
Code for Cluster Singleton

//run in a new thread on every Cache Server
while (true) {
    boolean gotLock = lockCache.lock("singletonLock", -1);
    if (gotLock) {
        //Start singletons
        wait();
    }
}

Implementing Consistent Views and Repeatable Queries
public interface MyBusinessObject{
  //data
  public Date getBusinessDate();
  public Date validFrom();
  public Date validTo();
}
Where does the System Time come from?
You can’t use the
System.currentTimeMillis() in a
distributed environment!
You need a cluster synchronised clock
Repeatable Time: A guaranteed Tick

Singleton Service

Write first

Write second

Write Time

Read Time

Replicated Caches (pessimistic)
As we add objects we timestamp them with Write Time
When we read objects we use Read Time.
Repeatable Time: A guaranteed Tick

Singleton Service

Write first

Write second

Write Time

Read Time

Replicated Caches (pessimistic)
Event Based Processing

cache.put(key, val);
Messaging as a System of Record

Messaging System (use Topics for scalability)
Messaging as a System of Record

JMS TOPIC

Trigger

cache.put(key, val);

A
B
C
D
Easy Grid Implementation in GUIs
CQCs on a CQC

Define this in config
How do you release quickly to a Coherence cluster?
Rolling Restart?
Disk-Persist
Final Thoughts
Data is the most important commodity that you have

Keep it safe
Use a Partition Listener
Have Proactive Monitoring of Memory
Version your Objects
Thanks

Slides & related articles available at:

http://www.benstopford.com