

# Locking and Cache Improvements for eXist-db

eXist-db Community Meetup  
XML Prague 08/02/2018

**Adam Retter**

 [adam@evolvedbinary.com](mailto:adam@evolvedbinary.com)

 [@adamretter](https://twitter.com/adamretter)



EVOLVED BINARY

# Adam Retter

- **eXist-db Core Dev (13 years!)**
- **Consultant**
  - Concurrency and Databases
  - Scala / Java / C++ / XQuery / XSLT
- **Open Source Hacker**
  - NoSQL: eXist-db / RocksDB
  - CSV Validator / UTF-8 Validator / Shadoop
  - Many other smaller contributions...
- **W3C Invited Expert for XQuery WG**
- **Author of the "eXist" book for O'Reilly**



# We will talk about...

## 1. The last year of work at Evolved Binary

- <http://www.evolvedbinary.com/technical-reports/exist-db/locking-and-cache-improvements/>
- <http://www.evolvedbinary.com/technical-reports/exist-db/asymmetrical-locking/>
- PR #1719 - [https://github.com/evolvedbinary/exist/tree/locking-and-cache-improvements\\_report/](https://github.com/evolvedbinary/exist/tree/locking-and-cache-improvements_report/)

## 2. Concurrency in eXist-db

- Multi-user Transactions
- Sharded Caches
- Memory barriers - i.e. Locks

## 3. Problems identified with Locking in eXist-db

## 4. Improvements/Solutions



# How did this project start?

- **Corruptions in eXist-db became unbearable**
- **Evolved Binary start developing Granite (~2015)**
  - R&D project to build a better Database for structured information
  - Started with eXist-db, and replacing its BTree storage
- **Transaction Isolation differences**
  - eXist-db likely offers Repeatable Reads isolation level
  - Granite should offer at least Snapshot Isolation
- **eXist-db's Collection Cache not Transaction/Isolation safe**
  - Goal: We need a better Collection Cache
  - Problem: Replacing the Collection Cache opened up many concurrency problems

# Collection Cache Problems

- **Many operations are synchronized(collectionCache)**
  - Performance *effectively* single-threaded for Collection ops
  - Introduced to avoid *previous* deadlocks and corruptions
  
- **Shared mutable state between transactions**
  - Lack of transaction isolation
    - Fine for Repeatable Read in eXist-db (if you know)
    - Granite wants better Isolation support
  - Current approach restricts possible concurrency improvements
    - Unless you sacrifice consistency

# Collection Cache for Granite

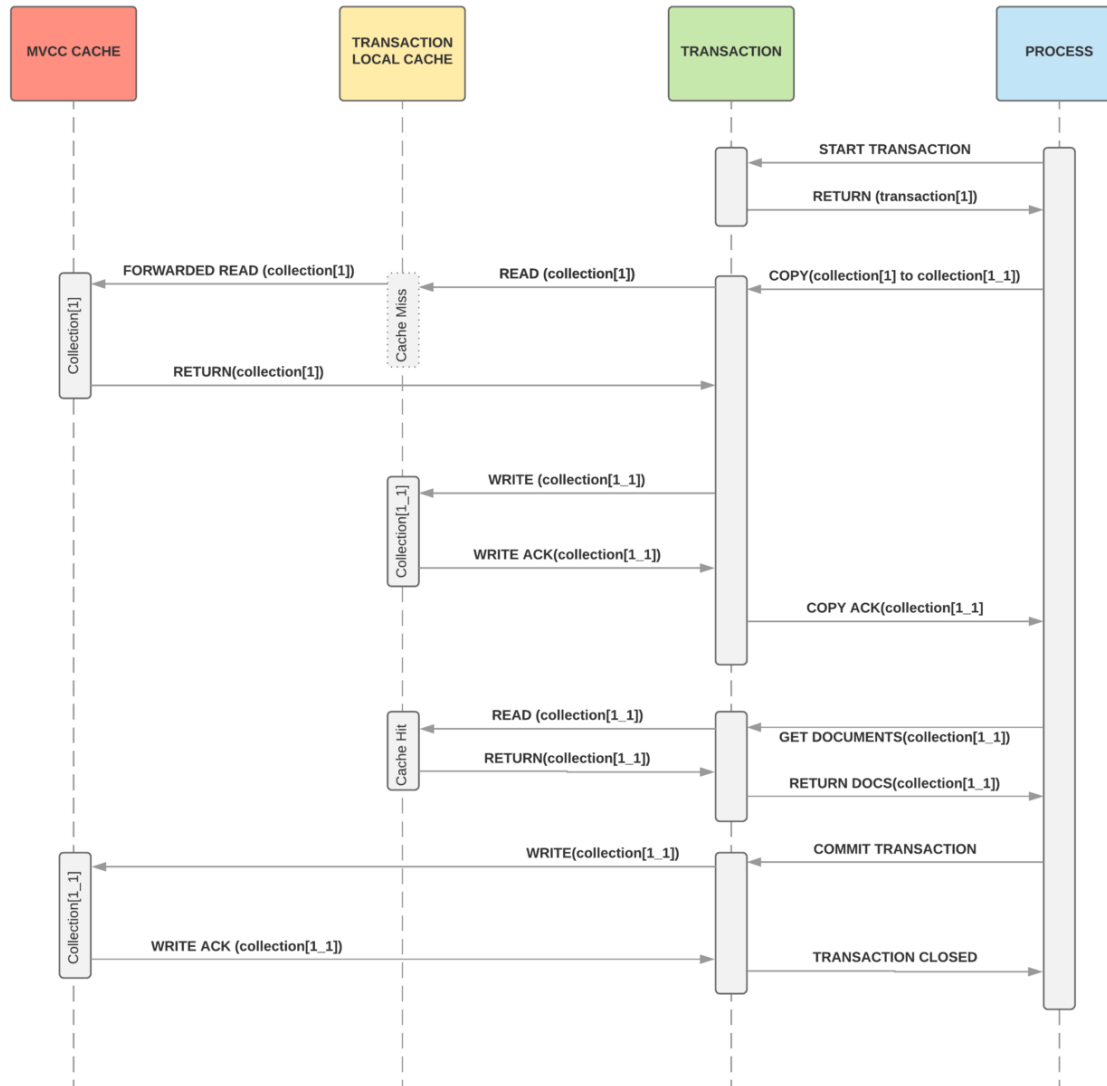
- Requirement: Transaction aware and Isolation safe
- Two Levels
  1. Transaction Local
    - Mutable
    - per-Transaction
    - Read-through to Global
    - Write version to Global on Commit
  2. Global
    - Immutable
    - Versioned and GC'd
- Remove synchronized(collectionCache) paths for performance



# Collection Cache for Granite

MVCC COLLECTION CACHE

Adam Retter | May 8, 2017



EVOLVED BINARY

# un-synchronized Collection Cache

- Revealed several deadlock scenarios
- Revealed further data corruption opportunities
- Showed inconsistent design and use of Collection/Document locks





# Locking issue categories

1. Inconsistent use of Locks
2. Inconsistent Lock Interleaving
3. Use of Incorrect Lock Modes - Read vs. Write
4. Lock Leaks
5. Accidental Lock Release
6. Insufficient Locking
7. Overzealous Locking
8. Correctness of Lock Implementations
9. Lack of Concurrency

# Collection Locks

- **One per in-memory Java Collection Object**
  - **should** only be zero-or-one Java Object in-memory per database Collection
  - Guards both mutable Java Object state and collections.dbx entry
- **Implementation:**  
**[org.exist.storage.lock.ReentrantReadWriteLock](#)**
  - Not actually Read/Write, really a Mutex!
  - "*modified*" `EDU.oswego.cs.dl.util.concurrent.ReentrantLock`
  - Exact Provenance is unclear
  - Correctness is unproven



# Document Locks

- **One per in-memory Java Document Object**
  - **should** only be zero-or-one Java Document in-memory per database Collection's Document
  - Guards both mutable Java Object state, and collections.dbx and dom.dbx entry
- **Implementation:**  
**`org.exist.storage.lock.MultiReadReentrantLock`**
  - Similar to Java SE's `ReentrantReadWriteLock`?
  - Writer Biased
  - Allows Lock upgrading, i.e.: `READ_LOCK` -> `WRITE_LOCK`
  - Adapted from Apache Turbine JCS project
  - Exact Provenance is unclear
  - Correctness is unproven

# Solution. 1 - Lock Manager and Lock Table

- **Before solutions, we must understand the problems!**
  - Centralises all locking operations
  - Reports all locking events to the Lock Table
- **Lock Identity**
  - Now per-URI rather than per-Object
    - Impossible to have two in-memory Java Objects for the same database object
    - Can acquire in advance of creating the database object
- **Lock Table**
  - Registerable Event Listeners
  - JMX Output
  - Snapshots and Traces

# Solution. 1 - Lock Table JMX

pid: 26159 org.apache.tools.ant.launch.Launcher test

Overview Memory Threads Classes VM Summary **MBeans**

- JMImplementation
- com.sun.management
- java.lang
- java.nio
- java.util.logging
- org.apache.logging.log4j2
- org.exist.management
  - LockManager
  - LockTable
    - Attributes
      - Attempting
      - Acquired**
    - Operations
      - dumpToConsole
      - dumpToLog
  - SystemInfo
- org.exist.management.exist
- org.exist.management.exist.tasks

Attribute value

Name	Value
key	concurrencyTest-remove-0
value	1

Refresh

MBeanAttributeInfo

Name	Value
<b>Attribute:</b>	
Name	Acquired
Description	Acquired
Readable	true
Writable	false
Is	false
Type	javax.management.openmbean.TabularData

Descriptor

Name	Value
<b>Attribute:</b>	
openType	javax.management.openmbean.TabularType(name=java.util.Map<java.lang.String, java.util.Map<or...
originalType	java.util.Map<java.lang.String, java.util.Map<org.exist.storage.lock.Lock\$LockType, java.util.Map<or...



# Solution. 1 - Lock Table Snapshot

## Acquired Locks

---

```
/db/test
  COLLECTION
    READ_LOCK      concurrencyTest-remove-12 (count=1),
                  concurrencyTest-remove-23 (count=1),
                  concurrencyTest-remove-21 (count=1),
                  concurrencyTest-remove-1  (count=1),
```

```
/db
  COLLECTION
    INTENTION_WRITE concurrencyTest-remove-0 (count=1)
```

```
/db/test/test1.xml
  DOCUMENT
    WRITE_LOCK      concurrencyTest-remove-0 (count=1)
```

## Attempting Locks

---

```
/db/test
  COLLECTION
    WRITE_LOCK      concurrencyTest-remove-0
```

# Solution. 1 - Lock Table Trace

- Simply set locks.log to "trace" in log4j2.xml

```
2018-02-07 18:16:42,877 TRACE - Acquired COLLECTION#1133260707637130
                               (WRITE_LOCK) of /db/system/security/exist by ma
2018-02-07 18:16:42,891 TRACE - Attempt COLLECTION#1133260707637130
                               (WRITE_LOCK) of /db/system/security/exist/group
2018-02-07 18:16:42,891 TRACE - Acquired COLLECTION#1133260707637130
                               (WRITE_LOCK) of /db/system/security/exist/group
2018-02-07 18:16:42,891 TRACE - Attempt DOCUMENT#1133260707647983
                               (WRITE_LOCK) of /db/system/security/exist/group
2018-02-07 18:16:42,891 TRACE - Acquired DOCUMENT#1133260707647983
                               (WRITE_LOCK) of /db/system/security/exist/group
2018-02-07 18:16:42,891 TRACE - Attempt COLLECTION#1133260707653300
                               (INTENTION_READ) of /db by main at 113326070765
2018-02-07 18:16:42,891 TRACE - Acquired COLLECTION#1133260707653300
                               (INTENTION_READ) of /db by main at 113326070765
2018-02-07 18:16:42,891 TRACE - Attempt COLLECTION#1133260707653300
                               (INTENTION_READ) of /db/system by main at 11332
2018-02-07 18:16:42,891 TRACE - Acquired COLLECTION#1133260707653300
                               (INTENTION_READ) of /db/system by main at 11332
```



# Solution. 2 - Standard Java Locks

- **Are eXist's lock implementations trustworthy?**
  - We don't know the Provenance!
  - No known proofs of Correctness!
  - Likely, not used in other projects...
- **Replaced with Java SE's implementations**
  - Fixed paths which performed lock upgrading
  - Collections/Documents: Java SE's ReentrantReadWriteLock
    - Collections now Reader/Writer (not Mutex)
    - Still mutex on Collection Cache and collections.dbx!
  - Some Java SE deadlock detection support, e.g. jconsole
  - Acquired with `Lock#lockInterruptibly()`





# Solution. 2 - Standard Java Locks

- **Replaced with Java SE's implementations**
  - .dbx files: Java SE's ReentrantLock
    - Complex Relationship between BTree and BTreeCache
    - Existing functions often request the (overall) wrong lock mode
    - eXist's ReentrantReadWriteLock was (really) a mutex, so previously not a problem
    - Difficult to make Reader/Writer
- **Provenance and Correctness of Lock implementations is now well known and widely used**



# Solution. 3 - Managed Locks

- **Reduces: Lock Leaks and Accidental Lock Releases**
- **ARM constructs engage with syntax**
  - e.g. try-with-resources
  - Lock(s) are always correctly released
- **We provide:**
  - ManagedLock
  - ManagedCollectionLock
  - ManagedDocumentLock
  - LockedCollection
  - LockedDocument



# Solution. 3 - Managed Locks

- Example, before Managed Locks:

```
Collection collection = null;
try {
    collection = broker.openCollection("/db/x/y", LockMode.READ_LOCK);

    DocumentImpl resource = null;
    try {
        resource = collection.getDocumentWithLock(broker, "doc1.xml",
            LockMode.READ_LOCK);

        // now do something with the document

    } finally {
        if (resource != null) {
            resource.getUpdateLock().release(LockMode.READ_LOCK);
        }
    }
} finally {
    if (collection != null) {
        collection.release(LockMode.READ_LOCK)
    }
}
```

# Solution. 3 - Managed Locks

- Example, with Managed Locks:

```
try(final Collection collection = broker.openCollection("/db/x/y",
    LockMode.READ_LOCK);
    final LockedDocument resource = collection.getDocumentWithLock(broker,
        "doc1.xml", LockMode.READ_LOCK)
    ) {

        // now do something with the document
    }
```



# Solution. 4 - Lock Ordering

- **Deadlock Avoidance: Iterate objects in stable global order**
- **Modified Collection's sub-Collections iterator**
  - Previously unstable order - backed by a HashSet
  - Now backed by a LinkedHashSet, provides insertion order
- **Modified Collection's Documents iterator**
  - Previously unstable order, backed by a TreeMap... ordered by Document ID!
  - Now backed by a LinkedHashMap, provides insertion order
- **Modified DefaultDocumentSet's iterator**
  - Previously unstable order, backed by a Int2ObjectHashMap
  - Now backed by a LinkedHashSet, provides insertion order

# Solution. 5 - Explicit Lock Interleaving

- **Deadlock Avoidance: Always mix Collection/Document locks in same order**
- **Mainly two patterns previously:**
  - **Symmetrical**
    - i.e.: Lock Collection, Lock Document, Unlock Document, Unlock Collection
    - Easiest to provide managed constructs for e.g. Managed Locks
  - **Asymmetrical**
    - i.e. Lock Collection, Lock Document, Unlock Collection, Unlock Document
    - Most flexible
    - Offers best concurrency... can release Collection lock early!

# Solution. 5 - Explicit Lock Interleaving

- Explicitly settled on the Asymmetrical pattern
  - Refactored eXist-db to exclusively use Asymmetrical pattern
  - Commented code to remind developers of Asymmetrical Pattern at each site of use
  - Documented the pattern

```
try(final Collection collection = broker.openCollection("/db/x/y",
    LockMode.READ_LOCK)) {

    // ...do something with *just* the Collection

    try(final LockedDocument resource = collection.getDocumentWithLock(
        broker, "doc1.xml", LockMode.READ_LOCK)) {

        // ...do something with the Collection and Document

        // NOTE: early release of Collection lock inline with Asymmetrical Lock
        collection.close();

        // ...finally do something with *just* the Document
    }
}
```

# Solution. 6 - Ensure Locking Annotations

- **Reduces: Incorrect Lock Modes, Lock Leaks, Accidental Lock Releases and Insufficient Locking**
- **Explicitly Documents (and enforces) locking contracts**
- **We provide Java Annotations (for developers):**
  - @EnsureLocked / @EnsureUnlocked
    - Lock mode must/not be held on a parameter or return object
  - @EnsureContainerLocked / @EnsureContainerUnlocked
    - Lock mode must/not be held on the object of a method call
- **Using Aspect Oriented Programming:**
  - Can log violations to ensure-locking.log
  - Can throw an exception when a violation is detected
  - Designed to be used at test time (not production)



# Solution. 6 - Ensure Locking Annotations

- Example lock contract violation(s) log:

```
FAILED: Constraint to require lock mode WRITE_LOCK on Collection: /db/test
<- org.exist.storage.lock.EnsureLockingAspect.
    enforceEnsureLockedParameters(EnsureLockingAspect.java:161)
<- org.exist.storage.NativeBroker.removeCollection(NativeBroker.java:1665)
<- org.exist.dom.persistent.NodeTest.tearDown(NodeTest.java:239)
<- sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
FAILED: Constraint to require lock mode READ_LOCK on Document: /db/test/test.xml
<- org.exist.storage.lock.EnsureLockingAspect.
    enforceEnsureLockedContainer(EnsureLockingAspect.java:303)
<- org.exist.dom.persistent.DocumentImpl.getDocId(DocumentImpl.java:197)
<- org.exist.indexing.range.RangeIndexWorker.removeCollection(RangeIndexWorker.java:100)
<- org.exist.indexing.IndexController.removeCollection(IndexController.java:100)
FAILED: Constraint to require lock mode READ_LOCK on Document: /db/test/test.xml
<- org.exist.storage.lock.EnsureLockingAspect.
    enforceEnsureLockedContainer(EnsureLockingAspect.java:303)
<- org.exist.dom.persistent.DocumentImpl.getDocId(DocumentImpl.java:197)
<- org.exist.storage.structural.NativeStructuralIndexWorker.
    getQNamesForDoc(NativeStructuralIndexWorker.java:540)
<- org.exist.storage.structural.NativeStructuralIndexWorker.
    removeDocument(NativeStructuralIndexWorker.java:505)}
```

# Solution. 7 - Collection Locking Strategy

- Attempt to find a Deadlock free Collection Locking scheme
- Many options investigated!
  - Collection hierarchy in eXist-db is a tree!
  - Adopted a Hierarchical Locking Scheme
  - Granularity of Locks in a Shared Data Base - Gray et al. 1975
    - Lock from the tree's root node to the most granular node of interest
    - Locking a node in the tree implies locking descendants
    - Multiple lock modes: IS, S, IX, SIX, and X
    - Uses weaker intention locks are used at higher levels
    - Not deadlock free under all conditions



# Solution. 7 - Collection Locking Strategy

- **Our modified implementation: Granularity of Locks in a Shared Data Base**
  - Mode 1: Multi-Writer / Multi-Reader
    - Better performance
    - Not deadlock free... unless user designs Collection hierarchy suitably
  - Mode 2: Single-Writer / Multi-Reader
    - Deadlock free
    - Restricts writes to any single Collection at once (likely happened previously)
    - Long running writes can block reads (likely happened previously)
    - The Default
  - Does not consider Documents!
    - Deadlocks can still occur between Collection and Documents
    - Could easily be extended to incorporate Documents

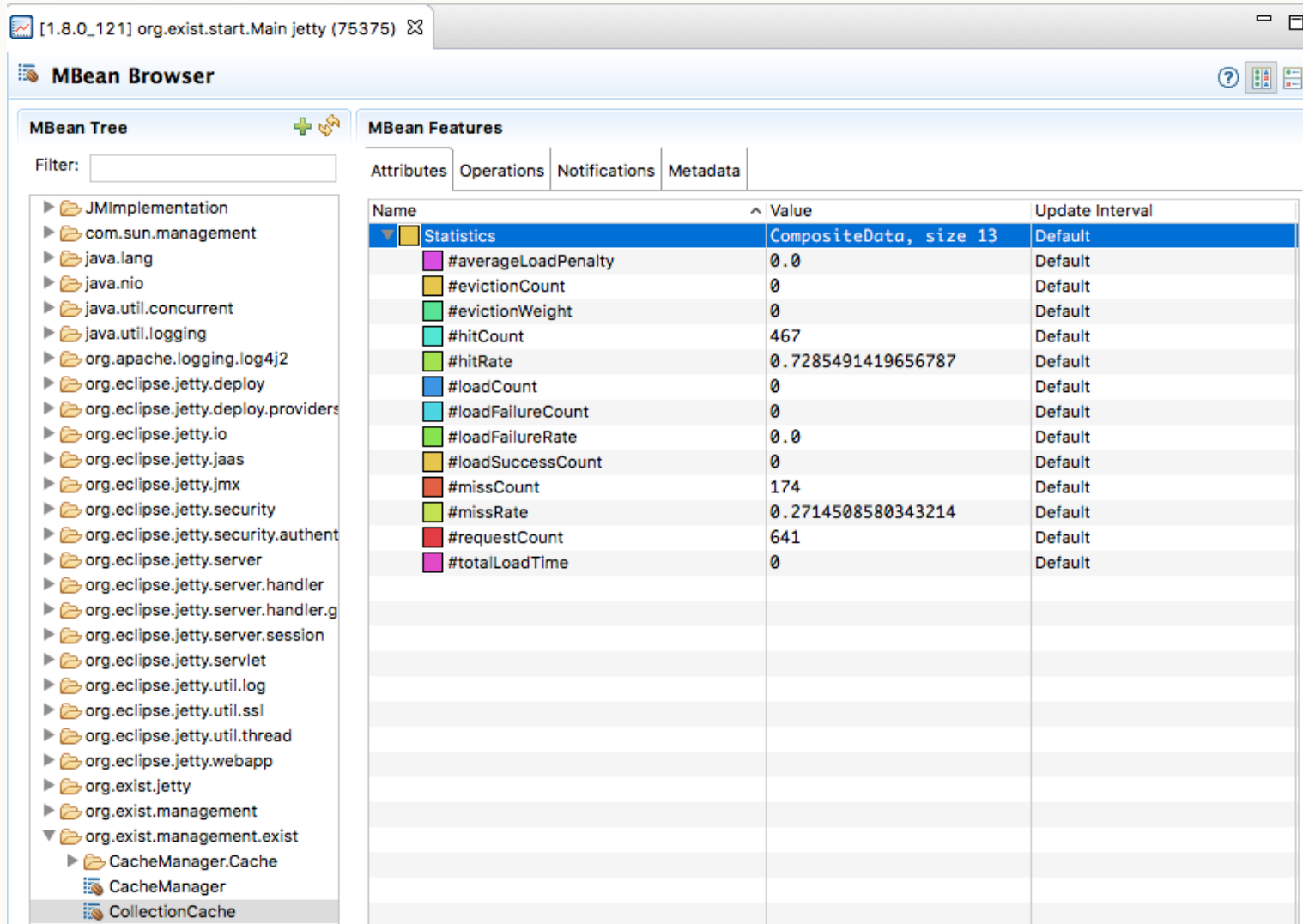
# Solution. 8 - Concurrent Collection Cache

- **Previously: synchronized(collectionCache)**
  - But... We have now addressed the locking issues!
- **Replaced eXist's Collection Cache:**
  - Previously HashMap with LRU Policy
  - Adopted Caffeine from Ben Manes
  - Provides both size and age bounds
  - Now TinyFLU policy - more performant
  - ConcurrentHashMap like interface
  - Comprehensive Cache Statistics available through JMX



# Solution. 8 - Concurrent Collection Cache

- Example Collection Cache JMX:



The screenshot shows the MBean Browser interface. The left pane displays the MBean Tree with a filter box. The right pane shows the MBean Features for the selected MBean, with the 'Statistics' tab selected. The table below lists the statistics and their values.

Name	Value	Update Interval
Statistics	CompositeData, size 13	Default
#averageLoadPenalty	0.0	Default
#evictionCount	0	Default
#evictionWeight	0	Default
#hitCount	467	Default
#hitRate	0.7285491419656787	Default
#loadCount	0	Default
#loadFailureCount	0	Default
#loadFailureRate	0.0	Default
#loadSuccessCount	0	Default
#missCount	174	Default
#missRate	0.2714508580343214	Default
#requestCount	641	Default
#totalLoadTime	0	Default

# Conclusion

- **Many Improvements to eXist-db**
  - Standard Java Locks
  - Improved Deadlock Avoidance
  - Managed Locks offer safety through syntax
  - Documented Locking Patterns
  - Corrected various lock use problems in the code base
  - Tools: EnsureLocked Annotations, LockTable tracing
- **Deadlocks Happen!**
  - eXist-db cannot yet abort a Transaction without risking corruption
- **Provides a good foundation for future work...**

