Fedora Performance for AIHT

By Johnny Graettinger

Abstract

This report details my experiences while evaluating Fedora with respect to the ingestion of large numbers of digital objects. Particular attention is given to performance of ingests into an already large repository, numbering around fifty thousand items. There are three primary obstacles I encountered during this process that hampered my efforts, related both to the command-line client and server side portions of the repository system.

Methodology

Needing to build a large base repository but lacking an ample digital library of objects, I settled on replicating the demo objects found within $fedoraHome/client/batch-demo a large number of times. This set of 10 objects includes as datastreams a collection of 10 images, averaging about 26 kilobytes in size apiece.

To perform the actual ingestion I created a bash script 'mass-import' that in a loop of configurable length assigns a new, unique PID to each object and calls 'fedora-batch-ingest'. This bash script, part of the Fedora distribution, in turn starts an instance of 'fedora.client.batch.AutoBatchIngest', the workhorse of the endeavor.

Once the repository numbered around fifty thousand objects, I leveraged the open source projects p6spy and sql-profiler (both available: www.p6spy.com) to profile SQL performance during additional object ingests.

p6spy is a JDBC driver wrapper that transparently sits between the client application (DSpace) and the actual PostgreSQL JDBC driver. Without requiring any code modifications, p6spy allows for the logging, timing, and tracing of database queries dispatched by the client application, relying on the implementing driver to fulfill the actual query. Logs can be written to disk, or sent over a network in real-time via Log4J.

sql-profiler is a utility that receives p6spy logging information in real time over a network, allowing for interactive viewing of database activity. Also included is the ability to profile over query structure, independent of the actual parameter arguments used. Query statistics presented in this report are derived from sql-profiler.

The environment used during this process was a Macintosh Dual 2GHz PowerPC G5 with 2GB SDRAM, running OS 10.3.8 and JDK 1.4.2.

Performance Observations

I collected statistics from a number of trial ingestion runs, where each run consisted of 10 ingests of 10 objects (100 objects total), being ingested by a newly started fedora-server instance. The

following table shows the most expensive SQL queries from a median run, listed by the percentage of time MySQL spent evaluating queries of that kind:

21.53%	"SELECT 1"
9.50%	"SELECT dsBindKeyDbID FROM dsBindSpec WHERE bMechDbID =16 AND dsBindSpecName = 'THUMBRES_IMG'"
9.10%	"INSERT INTO dsBind (doDbID, dsBindKeyDbID, dsBindMapDbID, dsBindKeySeq, dsID, dsLabel, dsMIME, dsLocation, dsControlGroupType, dsCurrentVersionID, policyDbID, dsState) VALUES (...)"
8.64%	"SELECT doDbID FROM do WHERE doPID='demo:80001'"
4.15%	"SELECT bMechDbID FROM bMech WHERE bMechPID = 'demo:2'"
3.92%	"SELECT doPID FROM doRegistry WHERE doPID='demo:80001'"
3.58%	"SELECT riMethodPermutation.permutation, riMethodMimeType.mimeType FROM riMethodPermutation, riMethodMimeType, riMethodImpl WHERE riMethodPermutation.methodId = riMethodImpl.methodId AND riMethodImpl.methodImplId = riMethodMimeType.methodImplId AND riMethodImpl.bMechPid = 'demo:2'"
3.42 %	"SELECT path FROM objectPaths WHERE token='demo:80001'"

Elapsed MySQL Time was 3003 ms

I performed several other 100 object ingests, and a 200 object ingest, and saw little deviation from these values. As far as database tuning goes, Fedora looks to be in pretty good shape--a similarly sized, tuned DSpace repository spends three times as much execution in SQL queries (though it makes no additional use of a Triplestore). Indeed, my qualitative experience is that ingests committed by a newly started server instance are quite snappy.

Issues Encountered

While performance for individual ingests was quite good, I encountered a number of other problems both while manipulating settings and over the course of large ingestions.

AWT Initialization

AutoBatchIngest? is a command line tool, but java's Abstract Windowing Toolkit is initialized during execution. This is in no way a show-stopping issue, but proves to be an annoyance as it grabs focus from any other open windows. I admit to being somewhat mystified as to where this is happening; though Fedora's client library contains a number of Swing-aware classes, tracing the execution of AutoBatchIngest? shows no explicit initialization of Swing components that I can find.

One workaround is to modify the fedora-batch-ingest script included with the distribution such that the -Djava.awt.headless=true flag is applied to the started instance of fedora.client.batch.AutoBatchIngest.

Shutdown Race Condition

It seems there is a race condition that occurs during the shutdown phase of AutoBatchIngest? which can cause a deadlock. Deadlocks occur infrequently, perhaps once or twice over the course of ingesting 2,500 objects, but cause difficulties for large batch ingestions.

Attaching JDB to a deadlocked client reveals that the threads involved are not explicitly those of Fedora, but rather the AWT. Preventing AWT initialization for all command line Fedora tools would likely kill two birds with one stone.

I implemented a simple workaround for this issue via a script that killed any instance of AutoBatchIngest? that ran for an inordinate amount of time. As the deadlock occurs during virtual machine shutdown, the termination never leaves the repository in an inconsistent state, has no impact on the just-completed ingest, and allows the overall ingest to continue.

JDB output whilst debugging deadlocked client:

```
> threads
Group system:
 (java.lang.ref.Reference$ReferenceHandler)0x4f7  Reference Handler   cond. waiting
 (java.lang.ref.Finalizer$FinalizerThread)0x4f8   Finalizer           cond. waiting
 (java.lang.Thread)0x4f9                          Signal Dispatcher   running

Group main:
 (java.util.logging.LogManager$Cleaner)0x4fb      Thread-0            unknown
 (java.lang.Thread)0x4fc                          AWT-AppKit          running
 (java.lang.Thread)0x4fd                          AWT-Shutdown        cond. waiting
 (java.lang.Thread)0x4fe                          DestroyJavaVM?      running


> where all
DestroyJavaVM:
```

AWT-Shutdown:

1. java.lang.Object.wait (native method)
2. sun.awt.AWTAutoShutdown.run (AWTAutoShutdown.java:278)
3. java.lang.Thread.run (Thread.java:552)

AWT-AppKit:
Signal Dispatcher:
Finalizer:

1. java.lang.Object.wait (native method)
2. java.lang.ref.ReferenceQueue.remove (ReferenceQueue.java:111)
3. java.lang.ref.ReferenceQueue.remove (ReferenceQueue.java:127)
4. java.lang.ref.Finalizer$FinalizerThread.run (Finalizer.java:159)

Reference Handler:

1. java.lang.Object.wait (native method)
2. java.lang.Object.wait (Object.java:429)
3. java.lang.ref.Reference$ReferenceHandler.run (Reference.java:115)

JDBC Driver Logging

Exceptions thrown during JDBC driver initialization are not trapped by Fedora's logger, and result only in a rather cryptic "No Suitable Driver Found" message. This proved a problem when configuring Fedora to work with the specialized p6spy passthrough JDBC driver, designed to sit between Fedora and the native MySQL driver.

This will likely only affect those planning on using databases other than MySQL, but I created modifications, included in diff format, which increase the verbosity of logging:

```
Index: src/java/fedora/server/storage/ConnectionPool.java
@@ -280,7 +280,7 @@
    try
    {
      // Load database driver if not already loaded
-     Class.forName(driver);
+      Class.forName(driver).newInstance();
      // Establish network connection to database
      Connection connection =
        DriverManager.getConnection(url, username, password);
@@ -291,7 +291,13 @@
      // throwing only one exception type.
      throw new SQLException("Can't find class for driver: " +
                             driver);
-   }
+    } catch(InstantiationException ie )
+      {
+        throw new SQLException("Couldn't instantiate driver class: " + ie.toString() );
```

```
+       } catch( IllegalAccessException iae )
+       {
+         throw new SQLException("Illegal Access while instantiating driver class: " +
iae.toString() );
+       }
   }
```

Kowari Triplestore

Early on in my experimentations with Fedora, I began encountering an inordinate number of unprovoked exceptions being thrown by long running instances of fedora.client.batch.AutoBatchIngest. In particular, numerous SAX xml and invocation target exceptions were thrown, as well as the occasional OutOfMemoryError?.

Suspecting these errors would be mirrored on the server side, I looked into nohup.out of the Fedora server instance, and found that a large number of OutOfMemoryErrors? were being thrown. Almost all of these exceptions were thrown within the triplestore implementation (org.kowari).

My initial guess was that the server instance was under too much memory pressure from continuous ingests, and the triplestore was merely the point of failure. Indeed, increasing the maximum heap size from the default of 128M to 512M did have an impact, allowing me to finish building the repository to its target size. However, though it reduced the frequency of thrown exceptions, the problem was clearly still present.

Still under the impression Kowari was merely the weakest link in the memory usage patterns of a stressed server instance, I tried experimenting with the frequency of Kowari buffer flushes via settings in fedora.fcfg. Requiring Kowari to flush its buffers more periodically ought to have relieved stress, but of course had no effect.

My next attempt was to add an explicit triplestore commit after each ResourceIndex? update, but found this caused every attempted ingest to fail. A new pattern of exceptions was occurring within nohup.out, as well:

ERROR 00:16 TripleWriteThread> Exception in TripleWriteThreadjava?.lang.OutOfMemoryError
ERROR 00:16 XAStatementStoreImpl> Prepare failed.
WARN 00:16 SimpleXAManager> RuntimeException? during prepare-commit.
java.lang.RuntimeException: Exception in TripleWriteThread: /Users/johng/fedora-2.0/triplestore/graph.g_1203

- at org.kowari.store.statement.xa.TripleWriteThread.checkForException(TripleWriteThread.java:194)
- at org.kowari.store.statement.xa.TripleWriteThread.drain(TripleWriteThread.java:242)

- at org.kowari.store.statement.xa.TripleAVLFile$Phase.<init>(TripleAVLFile.java:684)

...
Caused by: java.lang.OutOfMemoryError
WARN 00:17 SimpleXAManager> An exception occurred during prepare-commit. Attempting to roll back.
org.kowari.query.QueryException: Commit failed

- at org.kowari.store.AbstractDatabaseSession.endTransaction(AbstractDatabaseSession.java:3541)
- at org.kowari.store.AbstractDatabaseSession.insert(AbstractDatabaseSession.java:1310)
- at org.trippi.impl.kowari.KowariSession.doTriples(KowariSession.java:94)

...
Caused by: org.kowari.store.StoreException: Commit failed

- at org.kowari.store.xa.XADatabaseImpl.commit(XADatabaseImpl.java:327)
- at org.kowari.store.AbstractDatabaseSession.endTransaction(AbstractDatabaseSession.java:3539)

...
Caused by: java.lang.RuntimeException: Exception in TripleWriteThread: /Users/johng/fedora-2.0/triplestore/graph.g_1203

- at org.kowari.store.statement.xa.TripleWriteThread.checkForException(TripleWriteThread.java:194)
- at org.kowari.store.statement.xa.TripleWriteThread.drain(TripleWriteThread.java:242)
- at org.kowari.store.statement.xa.TripleAVLFile$Phase.<init>(TripleAVLFile.java:684)

...
Caused by: java.lang.OutOfMemoryError
WARN 00:17 ConcurrentTriplestoreWriter> Error auto-flushing update buffer: org.trippi.TrippiException: Error adding triples: org.kowari.query.QueryException: Commit failed

Curious that TripleWriteThread? would name a particular file in reference to the RuntimeException? it threw. I investigated the file structure of the triplestore and found, surprisingly, that files of the kind graph.g_1203 had not been written to in days, though I had been running Fedora consistently in that time. I began to suspect that the OutOfMemoryErrors? Kowari was throwing were really just the louder cousin of a silent file I/O error, and that somehow, early on, the Kowari datastore had gotten itself in an inconsistent state (perhaps from an abrupt server kill).

Shortly after I stumbled upon this bugzilla entry at http://scripta.lib.virginia.edu/bugs/show_bug.cgi?id=83 that highly parallels my experiences. It, and a duplicate entry at http://scripta.lib.virginia.edu/bugs/show_bug.cgi?id=110, include fedora outputs that closely match those I saw, though both claim only Windows machines are susceptible.

Taking a queue from the bugzilla report, I wiped out and had fedora rebuild the triplestore/ directory. This completely resolved the issue--I have since ingested over 2,500 objects without an error of any kind.

Recommendation

This last one is not so much an issue but a suggestion. After running for a period, ingestions in Fedora will sometimes stutter as the JVM Garbage Collector catches up to memory load. I added an explicit call System.gc() after object ingest in the hopes of combating aformentioned triplestore issues, but found that it instead impacted more on the qualitative (though not neccesarily quantitative) ingest speed.

As object ingestion is expensive in terms of memory (on the order of 7-8 Mbytes per object) and because upon ingest completion is a logical place to free that memory, I kept the addition.

```
Index: src/java/fedora/server/management/DefaultManagement.java
@@ -170,8 +170,17 @@
             if (w != null) {
                 m_manager.releaseWriter(w);
             }
+
+             // recover memory from ingest
             Runtime r=Runtime.getRuntime();
-            getServer().logFinest("Memory: " + r.freeMemory() + " bytes free of " +
r.totalMemory() + " available.");
+            long memAfter, memBefore;
+
+            memBefore = r.freeMemory();
+            System.gc();
+            memAfter = r.freeMemory();
+
+            getServer().logFinest("GC'd " + (memAfter - memBefore) + " bytes; " + memAfter + "
bytes free of " + r.totalMemory() + " available.");
+
             getServer().logFinest("Exiting DefaultManagement.ingestObject");
         }
     }
```