A Closer Look at Fedora's Ingest Performance

Kai Strnad, Matthias Razum
kai.strnad@fiz-karlsruhe.de, matthias.razum@fiz-karlsruhe.de

FIZ Karlsruhe
Development and Applied Research
Germany

Motivation

It is of paramount importance for large-scale applications that Fedora can handle huge amounts of data efficiently. While Fedora is generally known to be stable and reliable, there appears to be a lack of data and experience regarding large-scale installations and the performance implications thereof.

But performance and scalability are not solely dependent of the underlying repository software. Configuration, hardware, and other software systems (e.g., database server, Java virtual machine, triplestore) impact both scalability and performance. A thorough analysis of basic operations like ingest, updates, retrievals, and deletes help to identify bottlenecks and may lead to valuable hints regarding further optimization of Fedora.

FIZ Karlsruhe is currently working on several projects with large-scale Fedora repositories holding several million complex objects[1]. We conducted extensive performance and scalability tests with the current Fedora software (mostly version 3.0), focusing on ingest operations. Our goal was to prove that Fedora actually scales well enough for our use cases. Our test runs provided us with data which helped us identifying limits and constraints, and devising some optimization recommendations.

What To Measure?

Performance and scalability tests are a broad and potentially disputable topic. We tried to narrow down the area by defining a set of questions, mostly motivated by the requirements for our Fedora-based projects.

- **How many datastreams** can an object have? Is there a limit? If so, how does it manifest itself?
- Which **triplestore** is the **fastest** and most suitable for which use case? Is there an upper limit for the number of triples that can be stored in a triplestore?
- What is the **impact of external factors** (hardware, operating system, Java version?)
- Is it possible to **derive a recommendation for components** (like type of database, triplestore) in relation to the use case?

[1] Most projects are not (yet) public, but one well documented projects with a large-scale Fedora repository is eSciDoc (http://www.escidoc.org/)

- Is it possible to derive a **recommendation for hardware**?
- What are the bottlenecks in Fedora?
- Are there hidden limitations in the software?

Test Data

Our test data were 1,411,258 patents from the PCT database[2]. Each patent document has an XML full-text file and may additionally have a PDF version of the full-text and/or a first-page image file. Roughly half of the patent documents had all three file types. File sizes ranged from a few kilobytes to 13 megabytes. Each patent document has between 31 and 69 triples, with an average of 54 triples.

We created an additional metadata stream for each object synthetically by populating the mandatory DC datastream with 10 DC fields chosen arbitrarily (copied from fedora demo object), so each digital object contained the same metadata record.

How To Measure?

Finding a method that does not impair the performance of the monitored system is important, so that the resulting measurements are as accurate as possible. Several methods were considered, none yielded the desired results. Most existing Java profiling mechanisms have a noticeable effect on the execution performance. Therefore, we basically used two approaches:

- inserting `System.nanoTime()` directly in the code and
- using AOP for creating an aspect and weaving it into the code

The resulting execution times were aggregated using a buffer in a separate thread and periodically flushed over the network to a separate loghost using Log4J Socket Server in order not to impair IO performance on the ingesting host.

The JIT (just in time) compiler of the Java Virtual Machine (JVM) does lots of optimizations after the application is started. Classes aren't loaded by the JVM until the first time they are referenced, and code is compiled from bytecode to machine code only after it has been executed a number of times. We observed that it takes approximately 5000 - 10000 ingests for the process to stabilize. Only then the measured times can be considered representative.

Test Runs

All measurements were done on of-the-shelf PC hardware with a single Intel Core2 CPU (2.4GHz), 2GB RAM, locally attached SATA disk drives, and an Intel Pro 1000 network adapter. The machines were set up with Suse Linux 10 and ext3 file systems. As mentioned before, we laid our main focus on ingest operations. We conducted various test runs in order to identify areas in which configuration, tuning, or hardware setup impact the overall ingest times.

- Managed versus external content

[2] See http://www.wipo.int/pctdb/en/content.jsp for more information about the PCT database and its contents

- Different Java versions
- Java Heap and Garbage Collection tuning
- Triplestores
- Database servers
- Distributed ingests

Most of the tests were done with a limited number of objects (typically 50,000 – 100,000). After optimizing the configuration for our hardware setup, we did a large-scale ingest with about 14 million objects.

Observations

Managed vs External Datastreams

If digital objects contain datastreams that are of control group "M" (managed), they should be located during ingest either on the ingesting machine or on a machine with a very fast low-latency network connection in order to minimize retrieval time. Datastreams should be made available using a web server instead of sending them to the upload servlet.

Triplestores

Two triplestores were tested during the ingest: MPTStore and Mulgara (v 1.2)[3]. Both triplestores show similar performance characteristics during ingest. There are however a few thoughts to consider:

- MPTStore is not a true semantic store. It consists of multiple database tables used for subject and object and a lookup table that stores the predicates. However, even a database containing more than 100M triples performed reasonably well in terms of insertion and retrieval of triples.
- Mulgara is a semantic store built specifically for usage in semantic applications. However, as of version 1.2 there was a significant performance penalty using the `sync()` call which persists triples to the underlying physical storage. The `sync()` call took up to 500ms. For the ingest this is not an issue, because it is not necessary to call `sync()` often. However, in a scenario with CRUD operations depending on each other, `sync()` has to be called more often leading to severe performance penalties. This issue has not yet been tested for Mulgara 2.0.

Java

Even though the performance of Java 5 and Java 6 looks quite similar in this case, it is still advisable to use Java 6 if possible for the following reasons:

- The Java 6 VM has better self-tuning capabilities and therefore needs less attention.
- Java 6 provides more transparency through enhanced management and diagnostic capabilities.

For 64 bit architectures it is generally advisable to use the 64 bit version of Java. However, more memory is needed (up to 30%) due to the larger address space. Additionally if the

[3] For a more comprehensive comparison of MPT and Mulgara, see http://www.slideshare.net/cwilper/mptstore-a-fast-scalable-and-stable-resource-index

machine running Fedora Commons has more than 2G of RAM, the 64 bit version should be used in order to utilize the full amount of available memory. Tuning the Heap and Garbage Collector is, at least for Java 6, often not necessary. The heap should however be dimensioned appropriately in order to avoid frequent garbage collections and avoid Out-Of-Memory errors. Minimum and maximum heap size should be equal so that the Java VM doesn't have to re-adjust the heap size.

Database Servers

MySQL and PostgreSQL have been tested and were found to yield roughly equal performance when tuned properly. The usage of MySQL ISAM is discouraged because it lacks critical features necessary for this kind of use case. It is generally advisable to run the database engine on a separate machine or at least provide a separate disk for the database (especially WAL log). This will speed up database operations and improve performance up to 30% in some cases (see figure 1).

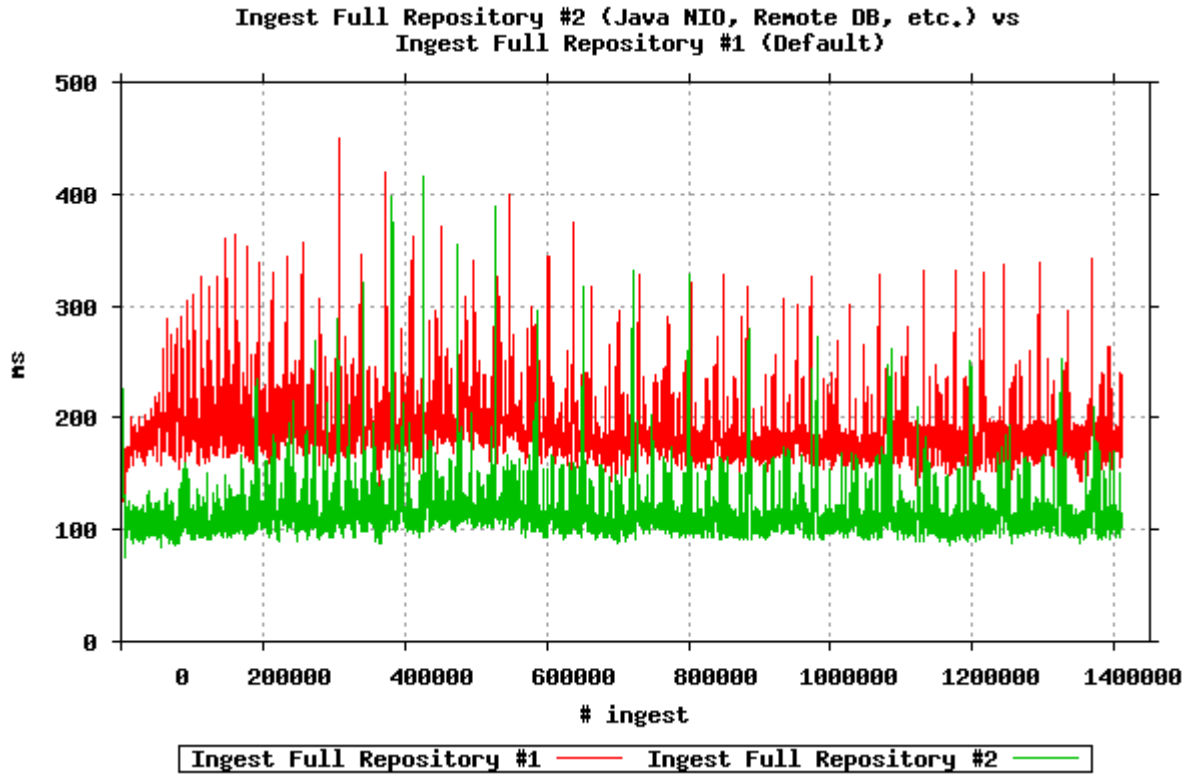

Figure 1: Comparison between two test runs, both using Fedora 3.0b2 with MPTStore, Java 1.6, PostgreSQL 8.3, and managed content. Test run #1 with database and Fedora on the same machine, test run #2 with remote database and Java NIO.

Conclusion

We haven't found yet a good answer to all of the above mentioned questions, but we think our intermediate results are interesting enough to share with the community.

We have successfully loaded 14 million objects with roughly 750 million triples into Fedora. The total amount of data loaded as FOXML and managed content summed up to roughly two terabytes. The ingest rate of about 10 objects/second remained stable throughout the whole ingest process, which took roughly 21 days (see figure 2). We stopped loading more objects after running out of disk space.

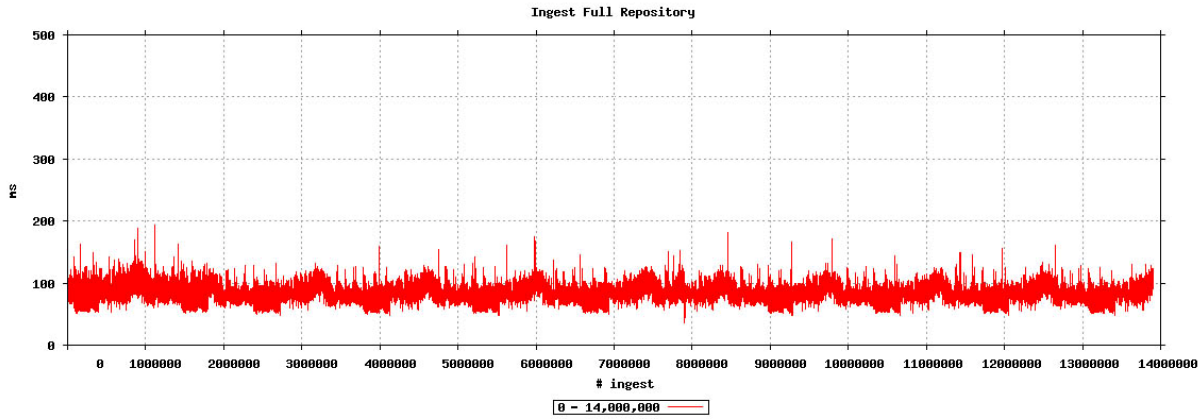

Figure 2: A test run with 14 million patent documents, 50 million binary content files as managed content, and 750 million triples showed stable ingest times of about 100ms per object.

We documented our various setups, all test runs, and our results and recommendations in a dedicated Fedora Performance and Scalability Wiki[4] and invite the community to contribute to this endeavor by providing comments, test scenarios, own measurements, and additional test data.

[4] http://fedora.fiz-karlsruhe.de/docs/