

Performance Testing

Testing Goals

Current instances of XTF used in production serve thousands to hundreds of thousands of documents. One of our investigation goals was to experiment with a much larger data set (millions of documents) to see how well XTF performs. This in turn helps to answer whether this retrieval system could realistically form the basis of a production union catalog.

The University of California's Melvyl union catalog must currently be able to serve a maximum of 600 simultaneous users, maintaining response times under 1 second for 95% of queries. So our goal was to determine whether XTF could meet that specification, and what hardware or software improvements might be required.

Test Methodology and Source Data

A software load testing rig was adapted from previous XTF testing for the purposes of this research. The rig attempts to simulate various loads on the system, beginning with a very light load and progressively increasing the number of simultaneous users to a specified upper limit. In our case, the load went from 1 up to 1000 users.

The questions naturally arise, "What is a user, and how do they use the system?" We attempted to answer this in two different ways, resulting in two source data sets:

1. First, we took log data gathered over the lifespan of the recommender system. The query URLs were gathered from Apache logs along with the source IP address and timestamp. The data had to be filtered to remove debugging queries and certain assessment pages. For this data set, a user was simulated by seeking to a random point in the log, and executing queries with the order and relative temporal spacing found there. After 5-10 queries, the load test pauses for 30 seconds, then seeks to a different random log position and resumes.
2. The other data set was taken from recent requests made to the live Melvyl catalog. Metadata fields were mapped to match the Recommender data set. Unfortunately, timestamps were not available, so they were synthesized to simulate a user requesting a page every 15 seconds. The results were formatted as Apache logs, and fed into the test rig that seeks to a random position as in #1 above.

Each test ran for 60 minutes, starting at 1 user and adding simultaneous users at even intervals up to 1000 users at the end of the test.

During each test, data on the latency of each request was collected. Latency is defined as the amount of time between issuance of a request and when the last byte of response data is received by the requestor. Additional data was collected on garbage collection activity

by the Java Virtual Machine (JVM) during the test runs. Tests were run when the prototype machine was otherwise idle.

Problems Encountered at High Loads

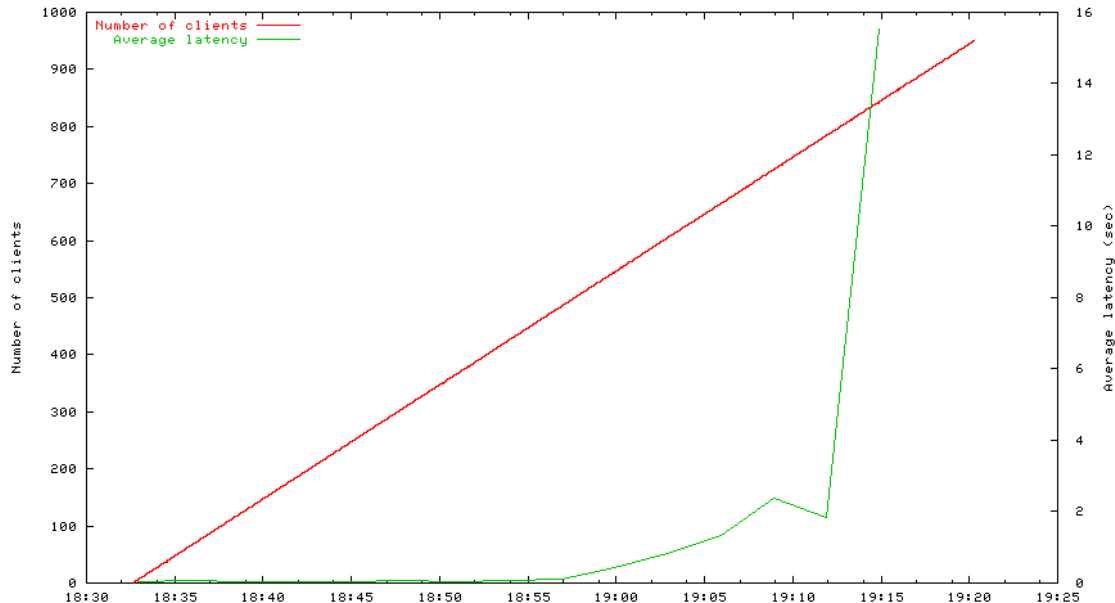
Early tests failed at fairly light loads, and debugging efforts led to discovery of limitations related mainly to memory use (our self-imposed JVM limit was two gigabytes):

- When result sets are sorted other than by relevance, such as sort-by-title, Lucene loads the associated meta-data for each document and keeps it in memory. In the current XTF implementation, these strings are never accessed and thus the memory is wasted. As a workaround for this test, sorting was turned off.
- Facet-based queries (for Corpus Analysis) also require fairly large in-memory tables. But the main problem with these is the large memory requirement during the lengthy table loading process, because many threads attempting to load them at once can overflow available memory. To work around this, facet queries were filtered out of the source log data.
- Some of the queries gathered from the Recommender system's Apache logs requested results starting at document numbers around 500,000. This forced XTF to load and temporarily store all the documents prior to that point. If several of these queries happened simultaneously in different threads, the memory space was exhausted. A simple limit of 200 was placed on the start document to work around this problem for our load test.

There are fairly simple solutions for each of the problems encountered above, so we believe that the test results are not compromised by the workarounds we had to implement.

Test Results Explained: Set 1

The following graph shows the average latency (time required to serve each request) for a 60-minute test with the load increasing from 1 up to 1000 users. Each “user” was simulated using the collected Apache logs from the Recommender system.

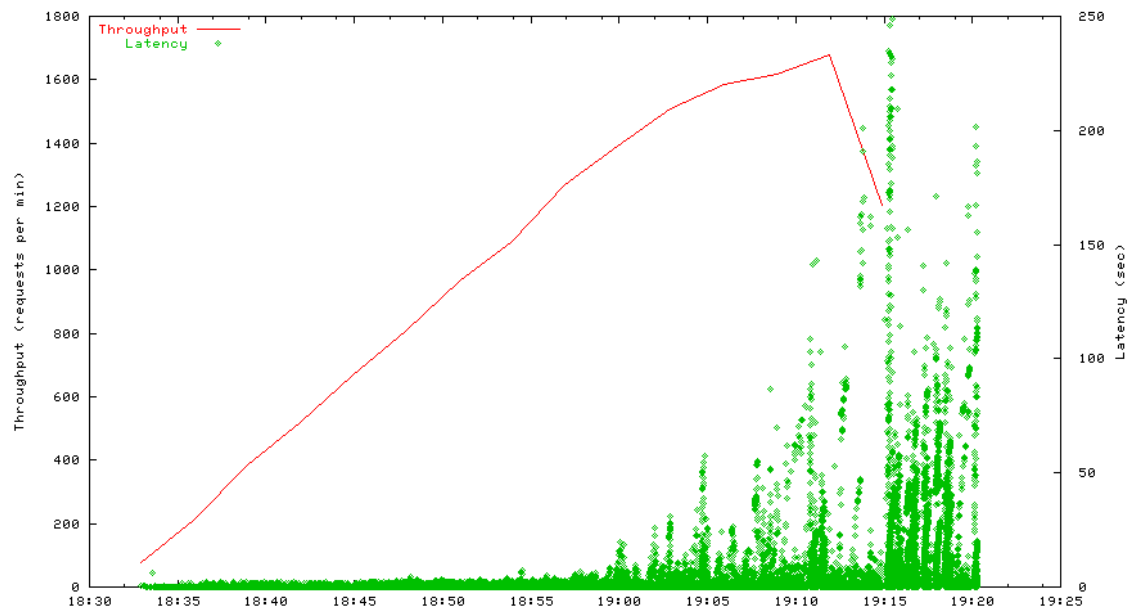


Number of Clients vs. Average Latency Graph

Looking at the graph, we can see that up until time 18:57, average latency is very low, meaning that requests are being served very quickly. This time corresponds to roughly 500 simultaneous users.

From that point, average latency begins to rise gently until time 19:12, or about 700 simultaneous users. During this period, requests start to take noticeably longer, with a typical request taking two seconds and some taking much longer. So for this data set, we can say that approximately 700 simultaneous users could be served with tolerable latency.

After 19:12, latency shoots up, indicating the system has reached a breaking point. The next graph shows this breaking point in another way:

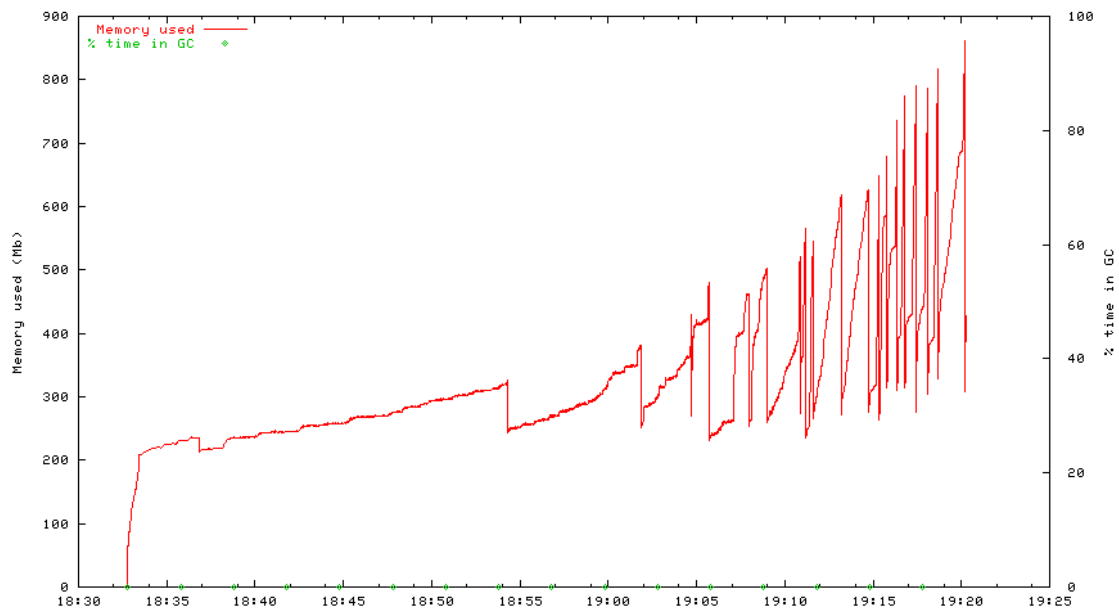


Graph of latency and throughput

The green dots reflect the latency of individual requests (rather than the average we saw on the previous graph). The red line is “throughput”, defined as the number of requests served by the system per minute.

We can see that, up until time 19:12, throughput rises roughly linearly in proportion to the number of simultaneous users. Then something happens, and it drops off very sharply.

What causes the sudden drop-off? Without further investigation we don't know for sure, but a clue might lie in the following graph:

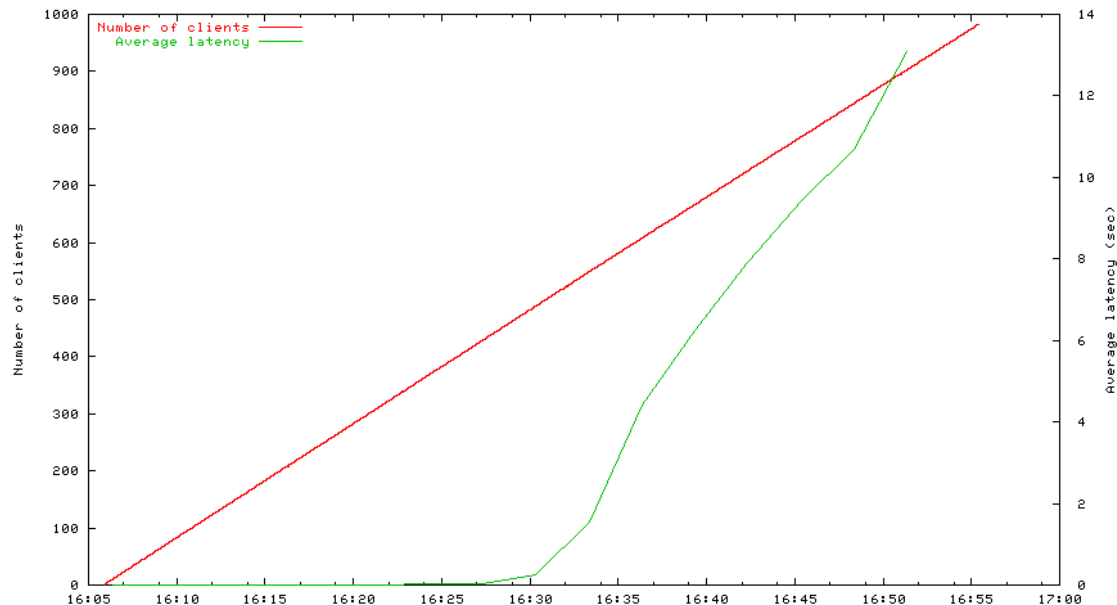


Graph of Java garbage collection activity

Around the time of the sudden drop-off, garbage collection events begin to occur very frequently, indicating that some part of XTF or Lucene is allocating and then discarding massive amounts of data.

Test Results Explained: Set 2

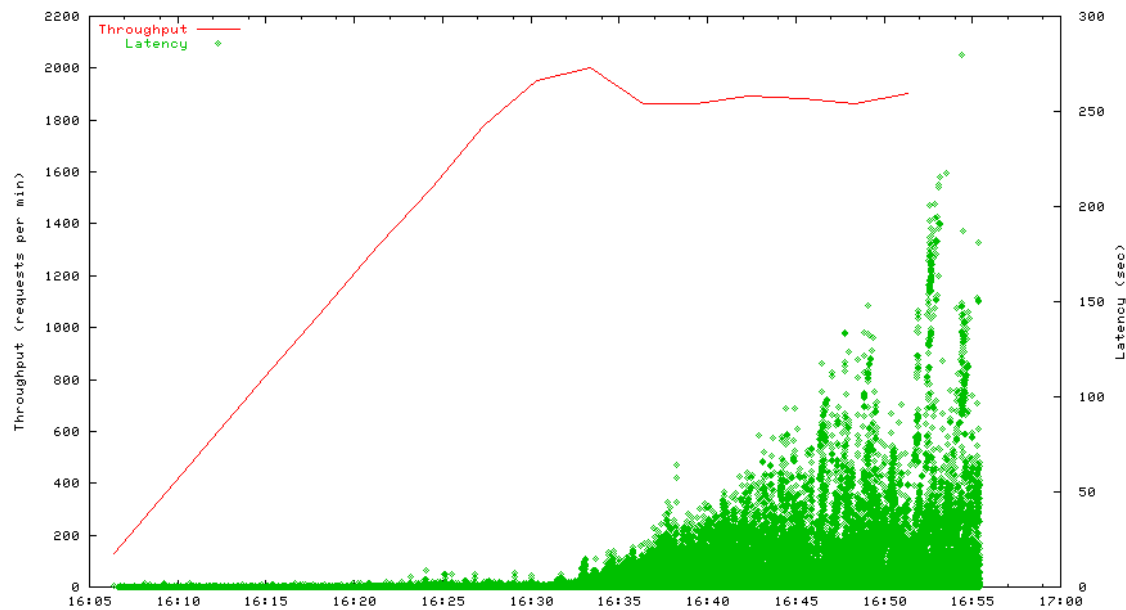
The following graph shows the average latency for a 60-minute test with the load increasing from 1 up to 1000 users. This time users were simulated using real Melvyl queries, but with synthetic timing.



Graph of load and latency

Up until time 16:30, or about 400 simultaneous users, average latency stays very low. Then the load begins to climb steeply though linearly.

Graphing throughput yields a different, possibly clearer, picture of how the system performs under load:



Graph of throughput and latency

Here we see that overall throughput (number of requests served per minute) increases steadily until roughly time 16:33, or about 500 simultaneous users. Then it declines slightly and levels off, indicating that the system has reached its maximum throughput. So from this data set, we would guess that the system can handle 500 simultaneous users with tolerable latency.

Extrapolating to a Bigger System

With this data set and test methodology, we can conclude with fair confidence that the system could handle 500-700 simultaneous users on the existing hardware. This could meet the requirement of the existing Melvyl catalog. Of course a real system would likely require performance testing and tweaking.

A significant difference between the test system and a real catalog is the number of records. The test system contains approximately 4.5 million records; the full Melvyl catalog is close to 30 million, about 7 times that size. How would XTF scale to a much larger data set?

Based on study of the Lucene code base and index format, we believe performance will degrade linearly with the number of data records. Thus, to serve seven times the number of records, a production version might need to split the load over seven machines similar to that used for the Recommender prototype.

Also, memory requirements might increase sevenfold, but this doesn't seem to present a problem as we artificially limited the prototype system to two gigabytes, and ended up

only using one gigabyte during these tests. The prototype machine contains 16 gigabytes of RAM.

Possibilities for Further Exploration

The logs used to construct the load tests were small in one case, and missing timestamps in the other case. Obtaining better log data from Melvyl would facilitate a much more realistic test.

In addition, loading the full set of ~30 million Melvyl records and running more tests could prove or disprove our hypothesis that performance will degrade linearly with the number of records.

Finally, collecting and graphing CPU and IO usage during the tests could help identify and illuminate the nature of bottlenecks, potentially allowing us to improve performance.