CS 677 Lab 3 Performance Evaluation Report

Dan Cline, Kyle Stevens, Tian Xia *UMass Amherst, Amherst College*

This document provides a performance evaluation of our implementation.

Performance of Sequential Requests w/wo Caching

In this test, we perform 100 requests of each for all the items and topics, and measure the average latency for each combination of item/topic and requests. The test can be found in `tests/sequential`. The output can be found in `tests/output`. We tested the latency with and without caching by performing the following operations (we only list the lookup latency for the first four items):

Performance of the System with Caching

item/topic	Response Time of `buy`(s)	Response Time of `lookup`(s)	Response Time of `search`(s)
item1	0.145609	0.015043	N/A
item2	0.186815	0.017289	N/A
item3	0.169865	0.018120	N/A
item4	0.158762	0.014589	N/A
gradschool	N/A	N/A	0.016642
systems	N/A	N/A	0.016781

Performance of the System without Caching

item/topic	Response Time of `buy`(s)	Response Time of `lookup`(s)	Response Time of `search`(s)
item1	0.147895	0.047688	N/A
item2	0.158601	0.047280	N/A
item3	0.157561	0.048731	N/A
item4	0.164243	0.048143	N/A
gradschool	N/A	N/A	0.047666
systems	N/A	N/A	0.048458

From these numerical results, we can see that caching improves the end-to-end response time on `lookup` and `search` requests, increasing the speed by almost 3 times. There was almost no changes in the end-to-end response time on `buy` requests, which makes intuitive sense since the `buy` requests are not cached and goes through the order server and then the catalog server.

Cache Tests

In this test, we tested write operations w/wo invalidation triggers and read operations w/wo cache misses, sending 100 sequential requests for each and calculating the overhead.

Write Operations w/wo Invalidations:

Operations/Time(s)	Writes wo Invalidation(s)	Writes w Invalidation(s)	Overhead (s)
Time(s)	0.153312	0.150751	-0.002561

Read Operations w/wo Cache Hits:

Operations/Time(s)	Read without Cache Miss(s)	Read with Cache Miss(s)	Overhead (s)
Time(s)	0.029069	0.055862	+0.026793

From these results, we see that the overhead for writes that triggers the invalidation is negligible. This is probably because the cache invalidation simply deletes the record from the dictionary and is not a time-consuming operation. We do notice however that reads with cache hit takes less time than reads with cache misses. This makes sense because reads with cache misses forces the request to take another hop from the frontend server to the catalog server, which gives the request an overtime.

Fault Tolerance Test

Test Description: In this test, we kill the primary catalog replica and perform updates on the secondary replica for `item 1`. After some time, we bring back the primary catalog replica, and kill the secondary replica. Then we bring back the secondary replica and perform more updates on `item 1`. At the very end, we perform lookup requests on both catalog servers to make sure that the number of items sold is consistent. The local test can be found in `tests\fault_tolerance`. The remote test is packaged into the script `run_tests.sh`.

Test Outcome: After performing the above operations, the number of items remaining in stock for `item 1` is consistent on both servers and matches the number of original stock subtracted by the number of buy operations performed. On the frontend server, the catalog server, and the alive catalog server, messages show that the killed server is down. After the dead servers come back alive, it successfully notifies the other catalog server that the primary is now `catalog1` as intended. The frontend also adds the revived server to the list of available servers. All of these messages can be checked by sshing into the servers and viewing the log messages. We conclude that our system is fault tolerant.