Daniel Wojtowicz
A20349975
CS 550: Advanced Operating Systems
February 22nd, 2018

Dapster Performance Evaluation

Table of Contents

I. Introduction

The performance evaluation performed as part of this project tests the response time of the indexing server as seen by peers. My first test will simply make 1000 sequential write requests and measure the average time per request. My second test will have multiple peers simultaneously making requests. The number of peers will vary from 1 to 3.

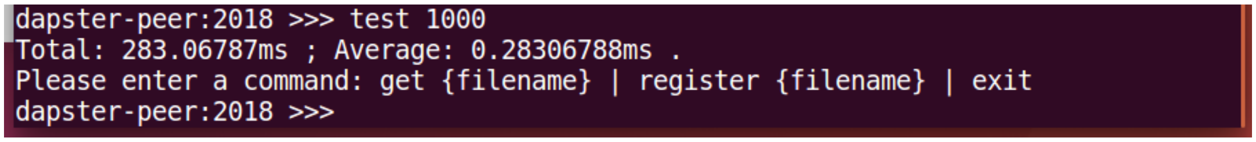
The test function in the peer simply runs many search requests on the server. The Peer contains a command line function to invoke the test with a varied number of requests. The server will have some files registered on it, but the main factor in timing will not be from the data structure used but the fact that many threads will be attempting to read/write from the index at the same time. Since I used the Java synchronized methods to employ a monitor, threads will need to wait their turn. I expect that in the multiple peers test, the more peers there are the more time a request will take on average, although not necessarily linearly.

Testing Environment

- Operating System: Ubuntu 16.04 64-bit on VirtualBox
- Processor: Ryzen 1700
- Memory: 2GB

II. Single Peer

This test was simple: do 1000 sequential requests to the server, asking for lookup for various files. The result of this test is shown below:



```
dapster-peer:2018 >>> test 1000
Total: 283.06787ms ; Average: 0.28306788ms .
Please enter a command: get {filename} | register {filename} | exit
dapster-peer:2018 >>>
```

Figure 1:Single Peer Test

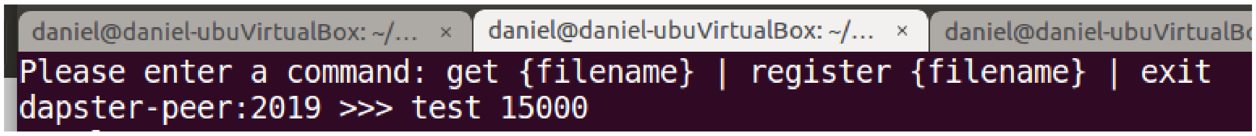
As you can see, the average response time was 0.28ms, which is between 1/3 and 1/4 of a second.

III. Multiple Peer

With multiple peers introduced, it is expected that requests will take more time due to the server's synchronization mechanisms. In this experiment, I will vary the number of simultaneous peers making search requests from N = 1 to N=3. I will take the result of the Single Peer test as N = 1, so the remaining tests will be N = 2 and N = 3.

In order to perform the test, I will have the secondary peer(s) doing a large number of sequential searches (15000) in separate terminal threads so that the 1000-sequential request test could be done while it is ensured that the server is already processing requests from another peer(s).

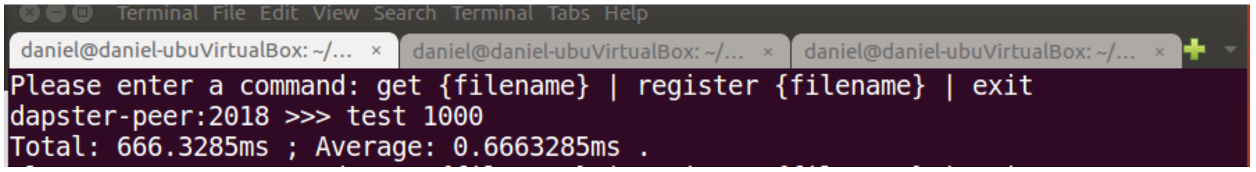
Testing N = 2



daniel@daniel-ubuVirtualBox: ~/...
daniel@daniel-ubuVirtualBox: ~/...
daniel@daniel-ubuVirtualB

```
Please enter a command: get {filename} | register {filename} | exit
dapster-peer:2019 >>> test 15000
```

Figure 2:Running the secondary peer



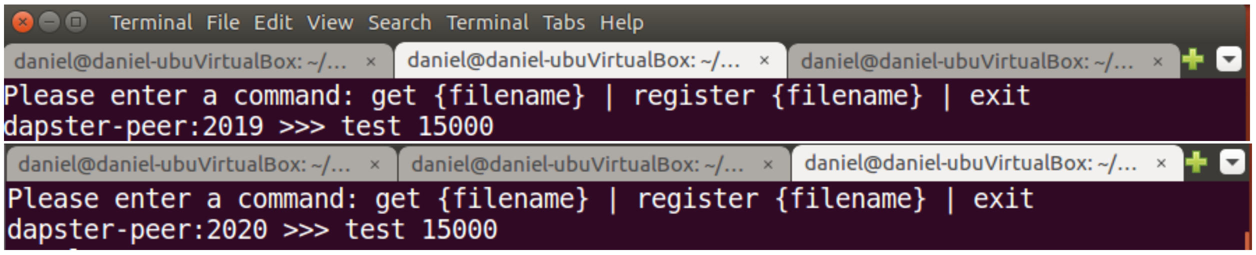
Terminal File Edit View Search Terminal Tabs Help
daniel@daniel-ubuVirtualBox: ~/...
daniel@daniel-ubuVirtualBox: ~/...
daniel@daniel-ubuVirtualBox: ~/...

```
Please enter a command: get {filename} | register {filename} | exit
dapster-peer:2018 >>> test 1000
Total: 666.3285ms ; Average: 0.6663285ms .
```

Figure 3: Running the primary testing peer simultaneously

For N = 2, it seems that the average is about .67 seconds, or 2/3 of a second.

Testing N = 3

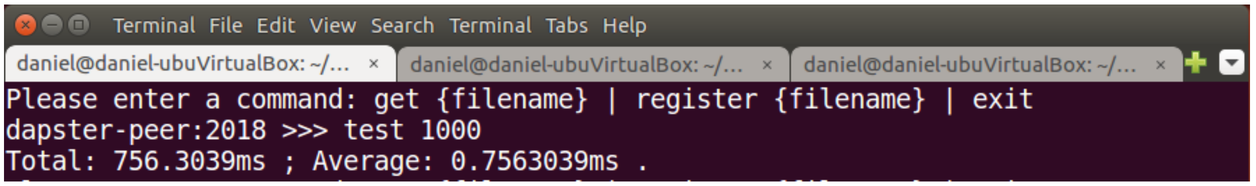


Terminal File Edit View Search Terminal Tabs Help
daniel@daniel-ubuVirtualBox: ~/...
daniel@daniel-ubuVirtualBox: ~/...
daniel@daniel-ubuVirtualBox: ~/...

```
Please enter a command: get {filename} | register {filename} | exit
dapster-peer:2019 >>> test 15000
```

```
Please enter a command: get {filename} | register {filename} | exit
dapster-peer:2020 >>> test 15000
```

Figure 4: Starting secondary concurrent peers 2 and 3



Terminal File Edit View Search Terminal Tabs Help
daniel@daniel-ubuVirtualBox: ~/...
daniel@daniel-ubuVirtualBox: ~/...
daniel@daniel-ubuVirtualBox: ~/...

```
Please enter a command: get {filename} | register {filename} | exit
dapster-peer:2018 >>> test 1000
Total: 756.3039ms ; Average: 0.7563039ms .
```

Figure 5: Running the primary testing peer alongside 2 and 3.

For N = 3 peers requesting the server simultaneously, the average response time is about 0.76ms, or 3/4 of a second.

Graphed Results

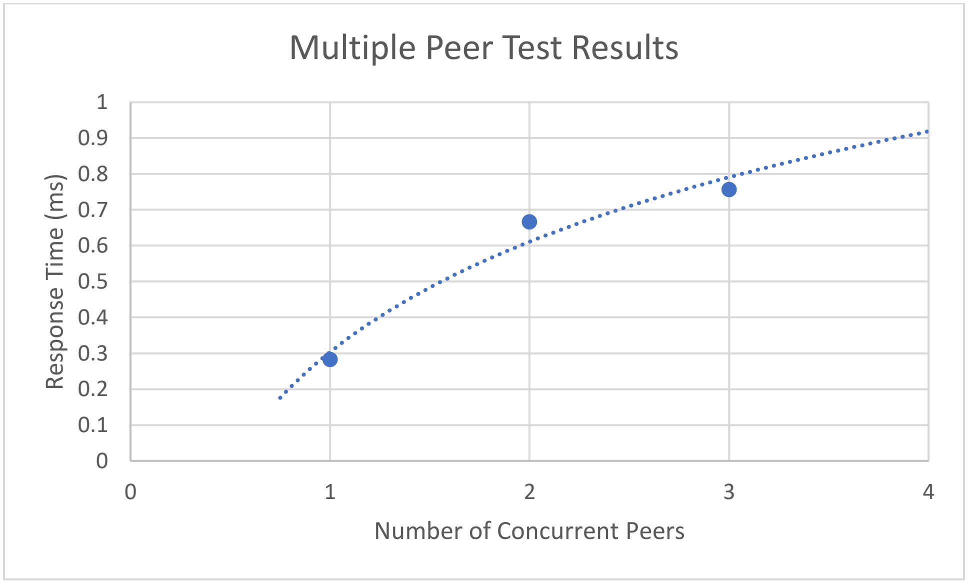


Figure 6: The three data points with a logarithmic curve approximation.

Conclusion

The tests show that with 2 simultaneous peers, requests in fact take more than double the time. However, when a third peer is introduced, the response time from the server only increases slightly.

One way to explain this is that when there is 1 peer, the server doesn't have to manage locks and provide concurrency protection. However, the concurrency via monitors that Java employs seem to take a large hit in performance, because the average time increased so much.

Once a third peer is introduced, response times grew only a little; to me, this shows that once the concurrency mechanisms are in place, their overhead is not increased very much as we scale the number of concurrent peers. I would expect 4, 5, 6, and more peers to only slightly increase the average response time.