CS 550

Sanchayan Maity
CWID A20340174
Section 03 - IITC, Bangalore

Assignment3: Performance Evaluation Report

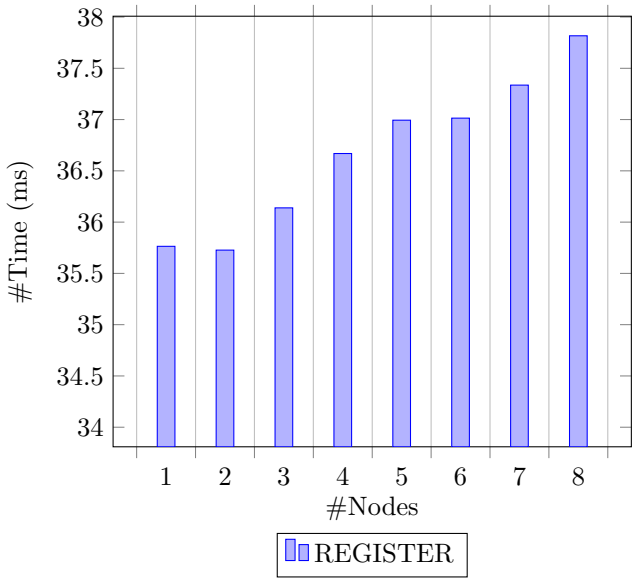
November 5, 2015

Note: Note that all results themselves are included in the *Results.txt* file.

The setup for the experiment comprised of eight m4.xlarge EC2 instances. Each instance had one instance of the application running. A screenshot of the running EC2 instances from the Amazon AWS management console is included in the file *aws.png*. The ping latency response between the instances was observed to be 0.28s on an average. The communication between the interfaces was done through the public IP interface of the EC2 instances. Ping latency from own network to AWS was 320ms. All of the below experiments were done with AWS EC2 instances running Ubuntu.

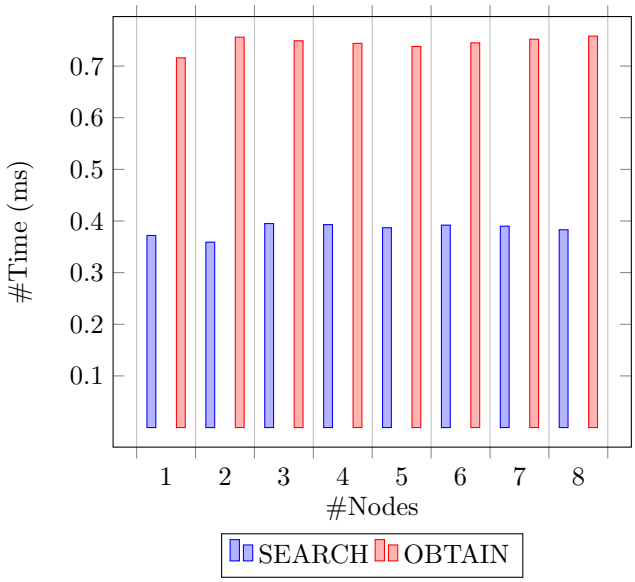
1 Experiment with varying nodes

In this part of the experiment, ten thousand operations were run for REGISTER, SEARCH and OBTAIN operations. Each instance had a directory comprising of 50 files of 1KB size generated using the included *file.sh* script.



Note that the graph seems increasing but notice the scale. The difference between the first and last node times is just about 2ms.

From the *Results.txt* file, we have 37.817ms as the average response time when eight clients were running. Note that for us this is related to the number of files as well which were 50 in this case. So the aggregate throughput in operations per second of all eight clients with 50 files, when put together is 1000ms/37.817ms is equal to roughly 26 operations per second multiplied by 8 clients viz 208 operations per second for all eight clients put together for register operation.



From the *Results.txt* file, we have 0.383ms and 0.758ms as the average response

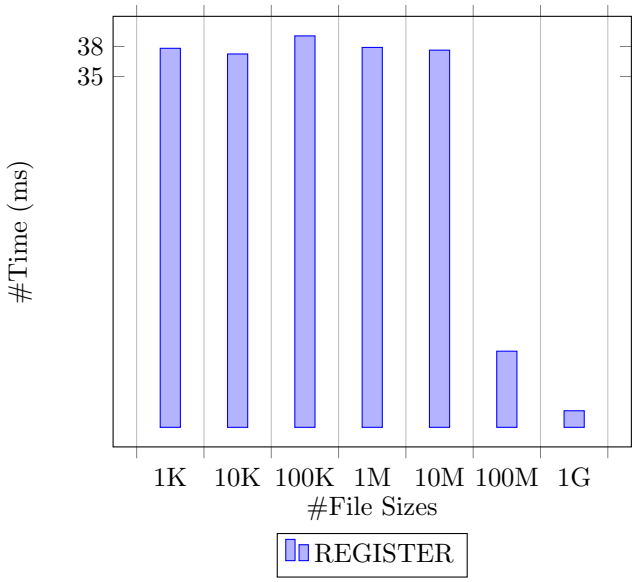
time for a single search and obtain operation respectively. Note that obtain operation time is related to the file size which was 1K in this scenario. So the aggregate throughput is 1000ms/0.383ms = 2611 operations per second multiplied by 8 clients which is 20888 operations per second for all eight clients put together for search operation. Similarly for obtain operation, we have 1000ms/0.758ms = 1319 multiplied by 8 clients viz. 10552 operations per second.

The results for register requests are more or less constant with node increase. Variation is only 2ms from the first node to the last eighth node. This is as expected from the distributed nature of the system. The register operation for a single file takes 0.7ms and since we call the register operation for each file, the time is as per the number of files being shared by a peer. Since 50 files were shared by each peer, 0.7ms multiplied by 50 is the observed time for registration requests viz. roughly 35ms.

The search and obtain requests also remain more or less constant again because of the distributed nature of the system.

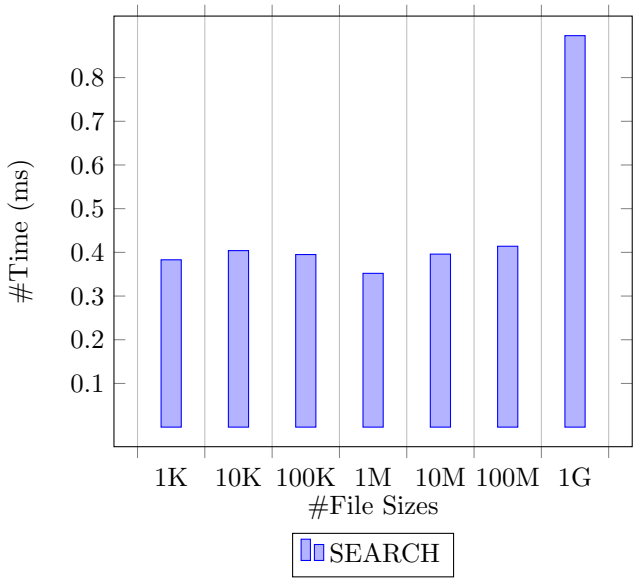
2 Experiment with varying file sizes

This experiment comprised of running the above experiments but by changing the file sizes available with each node. Each node was made to have identical file size of 1KB, 10KB, 100KB, 1MB, 10MB, 100MB and 1GB and then the above experiments were carried out being repeated each time for each file size considered. All eight nodes were made to participate in the experiment.

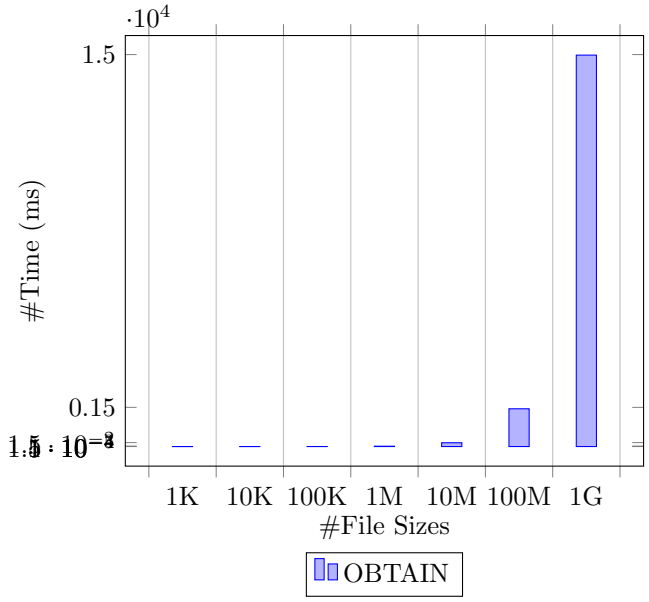


For the 1K, 10K, 100K, 1M, 10M case fifty files were made available with each peer. For the 100M case ten files were made available with each peer and for the 1G case one file was made available with each peer. As can be seen for the first five cases, registration times are same as observed in Part 1 and dependent

on the number of files. Since only ten files were used for the 100M case and one file was used for the 1G case, we observed the time for one single request roughly multiplied by the number of files as explained earlier and in the design document. The registration times are basically not dependent on the file size at all and as per the application logic are dependent on the number of file being registered.



We have results almost similar to Part 1 as the search operation also does not depend on the file size either. The last slightly higher response time for 1G case can be considered a slight anomaly perhaps arising due to the network state at that moment.



From the *Results.txt* file we know that that it takes 0.758ms for 1K file on average. So in 1 second we can send (1000ms * 1K) / 0.758 = 1319.2KB. For eight clients this is 1319.2KB * 8 = 10554 KBps. On similar lines, so we aggregate throughput in bytes per second for eight clients as follows as per file size:

1KB: 10.3 MBps
10KB: 70 MBps
100KB: 412 MBps
1MB: 590.6 MBps
10MB: 550 MBps
100MB: 554 MBps
1G: 550 MBps

The times for getting the file from another peer increase as the file size increases. This is expected as a higher file size means more number of bytes required transferring over the network. Note the fact that the transfer time is almost a linear multiplicative increase from 1M to 10MB, 10MB to 100MB and 100MB to 1GB.

Also it takes more time to write and read bigger file sizes. The writes can be checked for example as below:

time dd if=/dev/zero bs=1M count=10 of=tmpfile
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.0121604 s, 862 MB/s
dd if=/dev/zero bs=1M count=10 of=tmpfile 0.00s user 0.01s system 71

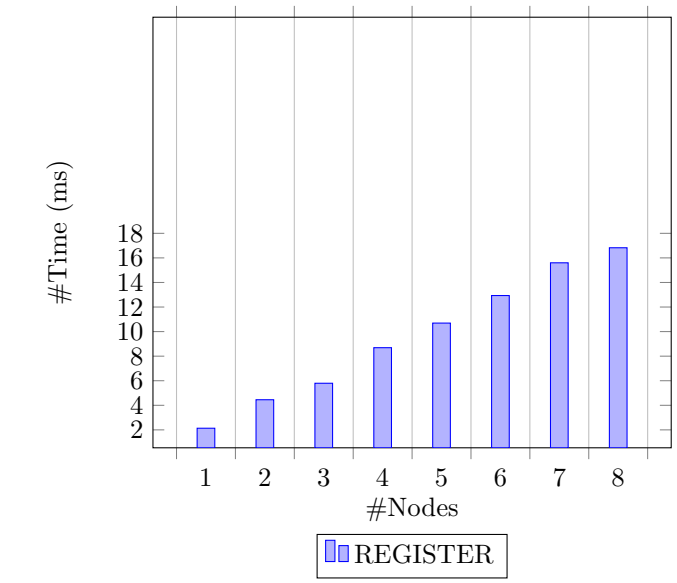
time dd if=/dev/zero bs=1M count=100 of=tmpfile
100+0 records in
100+0 records out
104857600 bytes (105 MB) copied, 1.31721 s, 79.6 MB/s
dd if=/dev/zero bs=1M count=100 of=tmpfile 0.00s user 0.11s system 8

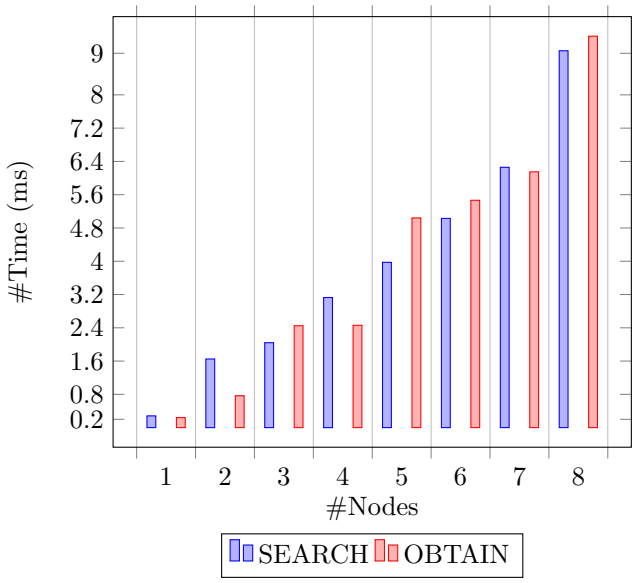
time dd if=/dev/zero bs=1M count=1000 of=tmpfile
1000+0 records in
1000+0 records out
1048576000 bytes (1.0 GB) copied, 11.9666 s, 87.6 MB/s
dd if=/dev/zero bs=1M count=1000 of=tmpfile 0.00s user 0.92s system 7

Note the increasing times for writes of 10MB, 100MB and 1GB as 0.012s, 1.317s and 11.97s respectively.

3 Experiment with centralised server

This part consisted of running similar tests with the centralised server design approach from assignement 1. 50 files of 1K size were used with each peer and the nodes doing the request were gradually increased.





As can be seen from the graphs, with the addition of each node, the response time of each operation increases. Comparing these same graphs from the part 1 of the experiment, we see that while the response time for each of the operations in the distributed design remained more or less same throughout with addition of nodes, for the centralised approach we have a gradual increasing response time.

If the Results.txt file is observed closely, the first node generally seems to have a lower response time than the rest of the nodes. We attribute this to the fact that there is delay in trying to simulaneously start the experiment from all nodes and centralised design is being followed here. Assignment 1 had a sqlite based

approach along with the use of global lock during any update operation. So this gives rise to slightly skewed results due to the fact whoever gets the lock first, will get better repsonse times.

For the purposes of this experiment, the assignment 1 code had to be slightly modified as issues were observed with respect to sqlite accesses with these kind of tests and they were not carried out during assignment 1 as a result of which these issues were not observed then. The delete operation had a lock missing which resulted in errors with delete operation from concurrent accesses. A lock was added in the same way as was used for registration operation.

No experiments were performed with varying file sizes in this part. This is because we expect similar results here as for the distributed approach. Getting a file is a peer to peer operation same as in a distributed system. It is not bound by the centralised approach or distributed approach at least for our use case here since we do not retrieve file in parts from multiple peers like torrent based approach does with it's Kademlia DHT. For increasing file sizes we expect increasing times just like the distributed case.

4 Conclusion

From all the experiments above we can conclude that the distributed approach provides better scalability since even at 8 node scales we observe the differences with the centralised approach having increasing response times with each addition. The exact absolute times are not to be compared here but rather the observed trend or nature of graphs which are observed. For example, in the first part of the experiment register operation take around 35ms while here with the centralised approach we have 16ms max. But the absolute times themselves are of not much significance and can vary depending on the implementation. So while we may have 35ms in the distributed approach and 16ms maximum in the centralised approach, the former stays same more or less throughout as the nodes are added and demonstrates scalability while the actual performance number is only due to the particular nature of the implementation/design being followed there.

However it must also be added that this is not exactly an apples to apples comparison. Different designs are followed for the server in centralised and distributed implementation. The centralised design had a per thread accept and a pre allocated thread pool design while the distributed one uses a workqueue based epoll event based design.