Lab Report for PA 3

1. Measurement

All the measurements below are were exported in terminal while running the code on client side

- Measure the RTT while uploading and downloading the data. Select proper units and report results with appropriate levels of precision.
- Measure the download time for the block contents for each policy for both the Experiments
- Measure the download times with **1MB blocks and 64KB blocks**
- Run each experiment multiple times (3 times)
- Store these measurements in a file for visualization and analysis later

Store these measurements in a file for visualization and analysis later

```
[08:41:06.372] [error] [thread 12985] RTT #0 = 129
[08:41:06.503] [error] [thread 12985] RTT #1 = 130
[08:41:06.632] [error] [thread 12985] RTT #2 = 129
[08:41:06.762] [error] [thread 12985] RTT #3 = 129
[08:41:06.891] [error] [thread 12985] RTT #4 = 129
[08:41:07.021] [error] [thread 12985] RTT #5 = 129
[08:41:07.151] [error] [thread 12985] RTT #6 = 129
[08:41:07.281] [error] [thread 12985] RTT #7 = 129
[08:41:07.281] [error] [thread 12985] Average RTT is 129.125 for server #2...
[08:41:07.281] [info] [thread 12985] RTT standard dev: 0.330719 milliseconds for server #
[08:41:07.281] [info] [thread 12985] Getting block hashlist from server #2
[08:41:07.410] [info] [thread 12985] Calculating RTT for server #3...
[08:41:07.411] [error] [thread 12985] RTT #0 = 0
[08:41:07.411] [error] [thread 12985] RTT #1 = 0
[08:41:07.411] [error] [thread 12985] RTT #2 = 0
[08:41:07.412] [error] [thread 12985] RTT #3 = 0
[08:41:07.412] [error] [thread 12985] RTT #4 = 0
[08:41:07.412] [error] [thread 12985] RTT #5 = 0
[08:41:07.412] [error] [thread 12985] RTT #6 = 0
[08:41:07.413] [error] [thread 12985] RTT #7 = 0
[08:41:07.413] [error] [thread 12985] Average RTT is 0 for server #3...
[08:41:07.413] [info] [thread 12985] RTT standard dev: 0 milliseconds for server #3
[08:41:07.413] [info] [thread 12985] Getting block hashlist from server #3
[08:41:07.413] [info] [thread 12985] Getting FileInfoMap from server #3
[08:41:07.418] [error] [thread 12985] Download time of file file0.dat is 4 milliseconds.
[08:41:07.418] [info] [thread 12985] Reconstituting file 'file0.dat'
[08:41:07.419] [info] [thread 12985] File 'file0.dat' reconstitution successful
[08:41:07.424] [error] [thread 12985] Download time of file file1.dat is 4 milliseconds.
[08:41:07.424] [info] [thread 12985] Reconstituting file 'file1.dat'
[08:41:07.425] [info] [thread 12985] File 'file1.dat' reconstitution successful
[08:41:07.429] [error] [thread 12985] Download time of file file2.dat is 4 milliseconds.
[08:41:07.429] [info] [thread 12985] Reconstituting file 'file2.dat'
[08:41:07.430] [info] [thread 12985] File 'file2.dat' reconstitution successful
[08:41:07.434] [error] [thread 12985] Download time of file file3.dat is 4 milliseconds.
[08:41:07.434] [info] [thread 12985] Reconstituting file 'file3.dat'
[08:41:07.435] [info] [thread 12985] File 'file3.dat' reconstitution successful
[08:41:07.440] [error] [thread 12985] Download time of file file4.dat is 4 milliseconds.
[08:41:07.440] [info] [thread 12985] Reconstituting file 'file4.dat'
[08:41:07.440] [info] [thread 12985] File 'file4.dat' reconstitution successful
[08:41:07.445] [error] [thread 12985] Download time of file file5.dat is 4 milliseconds.
```

2.Data Visualization

Experiment 1

a. Report only the Average (Avg) RTT and the standard deviation (std) while uploading and downloading files for each policy in a table such as this.
RTT (in milliseconds):

Mumbai RTT(Upload) (random)

Seoul	Ireland	San Paulo
157.625 (0.484)	124(0)	319.375 (0.696)

Mumbai RTT(Download) (random)

Seoul	Ireland	San Paulo
158 (0)	121.375 (0.484)	320.5 (0.5)

Mumbai RTT(Upload) (tworandom)

Seoul	Ireland	San Paulo
157 (0)	119 (0)	319.125 (0.33)

Mumbai RTT(Download) (tworandom)

Seoul	Ireland	San Paulo
157 (0)	123.5 (0.5)	321 (0)

Mumbai RTT(Upload) (local)

Seoul	Ireland	San Paulo
158 (0)	123.75 (0.43)	322.5 (2.35)

Mumbai RTT(Download) (local)

Seoul	Ireland	San Paulo
157 (0)	123.25 (0.43)	321 (0)

Mumbai RTT(Upload) (localclose)

Seoul	Ireland	San Paulo
157.25 (0.43)	123 (0)	319 (0)

Mumbai RTT(Download) (localclose)

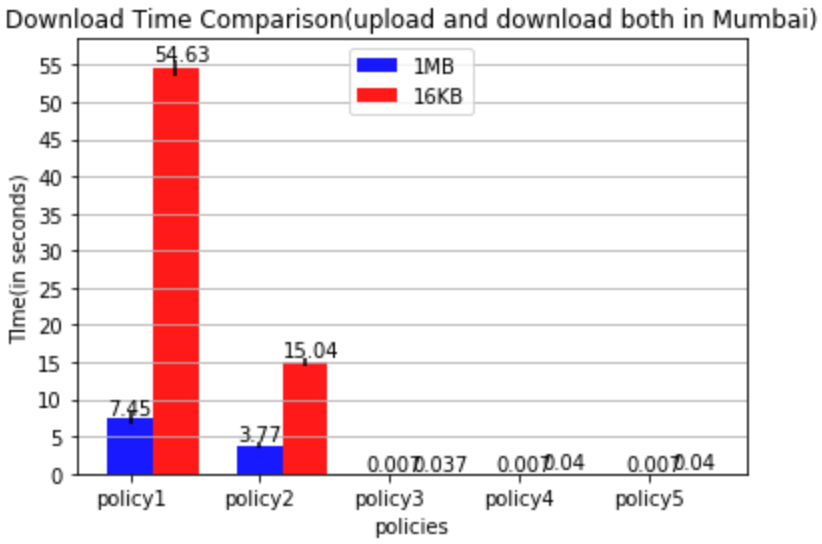
Seoul	Ireland	San Paulo
171.875 (3.79)	119.375 (0.48)	319.13 (0.33)

Mumbai RTT(Upload) (localfarthest**)**	**Mumbai RTT(Download) (**localfarthest**)**
<table><tr><td>Seoul</td><td>Ireland</td><td>San Paulo</td></tr><tr><td>157 (0)</td><td>133 (0)</td><td>326.5 (0.87)</td></tr></table>	<table><tr><td>Seoul</td><td>Ireland</td><td>San Paulo</td></tr><tr><td>193.5 (5.31)</td><td>126 (0)</td><td>322.625 (2.83)</td></tr></table>

Download Time (in seconds with its std in ()):
Notice that 64KB download is slower than 1MB download in overall, because the overhead of downloading 64 KB is slower.

Policy # \ File Size	1MB	64KB
1.random	7.45 (0.738)	54.63 (1.106)
2.Two random	3.77(0.393)	15.04 (0.429)
3.local	0.007(0.0002)	0.037 (0.0004)
4.Local close	0.007(0.0002)	0.04 (0)
5.Local farthest	0.010 (0.0004)	0.04 (0)

b. Plot the download times for each policy as bar plots.
(Policy number are correspond with the above diagram.)



Experiment 2

a. Report only the Average (Avg) RTT and the standard deviation (std) while uploading and downloading files for each policy in a table such as this.

RTT (in milliseconds):

Mumbai RTT(Upload) (random)	**Ireland RTT(Download) (random)**

Seoul	Ireland	San Paulo
157.625 (0.484)	124(0)	319.375 (0.696)

Seoul	Mumbai	San Paulo
252 (0)	123 (0)	183 (0)

Mumbai RTT(Upload) (tworandom)	**Ireland RTT(Download) (tworandom)**

Seoul	Ireland	San Paulo
157 (0)	119 (0)	319.125 (0.33)

Seoul	Mumbai	San Paulo
250 (0)	121 (0)	183.5 (0.5)

Mumbai RTT(Upload) (local)	**Ireland RTT(Download) (local)**
Seoul | Ireland | San Paulo 158 (0) | 123.75 (0.43) | 322.5 (2.35)	Seoul | Mumbai | San Paulo 251.25 (0.43) | 119 (0) | 183.375 (0.49)

Mumbai RTT(Upload) (localclose)	**Ireland RTT(Download) (localclose)**
Seoul | Ireland | San Paulo 157.25 (0.43) | 123 (0) | 319 (0)	Seoul | Mumbai | San Paulo 245.125 (0.33) | 123.125 (0.33) | 184 (0)

Mumbai RTT(Upload) (localfarthest**)**	**Ireland RTT(Download) (**localfarthest**)**
Seoul | Ireland | San Paulo 157 (0) | 133 (0) | 326.5 (0.87)	Seoul | Mumbai | San Paulo 245.125 (0.33) | 123.125 (0.33) | 184 (0)

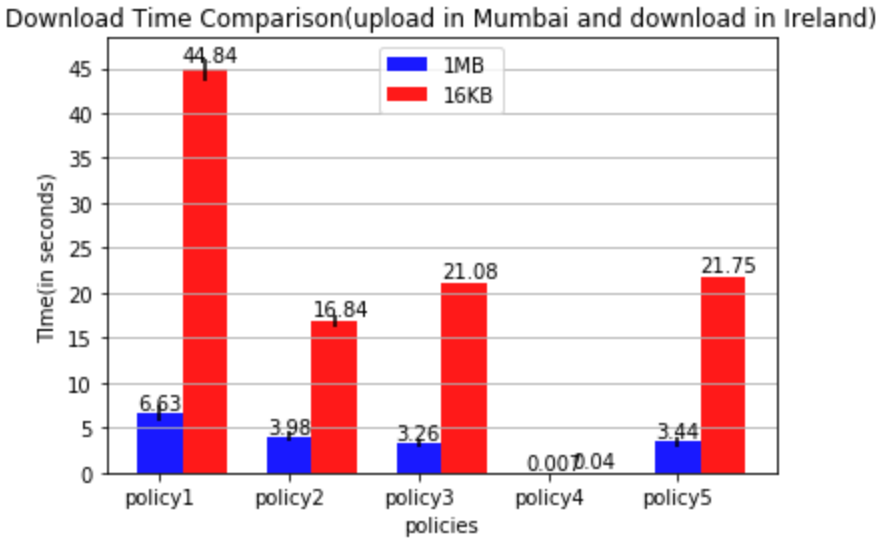
Download Time (in seconds with its std in ()):

Notice that 64KB download is slower than 1MB download in overall, because the overhead of downloading 64 KB is slower.

Policy \ File Size	1MB	64KB
1.random	6.63(0.862)	44.84 (1.234)
2.Two random	3.98(0.565)	16.84 (0.665)
3.local	3.26(0.469)	21.08 (0.03)
4.Local close	0.007(0.0006)	0.04 (0)

5.Local farthest	3.44(0.507)	21.75 (0.042)

b. Plot the download times for each policy as bar plots.
(Policy number are correspond with the above diagram.)



3. Analysis

a. Is there a correlation between observed RTT and throughput and bandwidth ?

Throughput is correlated to RTT while bandwidth is not.

Throughput = download data amount / download time, and observed RTT is directly related to the distance between two nodes. According to the table, the farther away the distance, the lower the throughput.

Bandwidth is the capacity of the link or inherent property of the network, it is not correlated to observed RTT.

b. For the 1-site experiments, comment on the relative performance of the policies and give 1-2 sentences explaining why you think each policy performed the way it did

The **local policies (local, localclosest, local_farthest)** have the best performance, with download duration orders of magnitudes lower than the random policies. This is **because**

blocks are always downloaded from the same node where it is uploaded and stored with local policy.

Two-random takes less than half time to download than random. That's because it's much more likely to have some blocks stored on closer nodes from the client in two-random storage policy. Pure random policy performs the worst.

Generally the download time standard deviations are in proportion to download time values.

c. For the 2-site experiments, comment on the relative performance of the policies and give 1-2 sentences explaining why you think each policy performed the way it did

Similar to the 1-site experiment, **the local policies have much better download time performance than random policies, and two_random is better than pure random.** Generally the download time standard deviations are in proportion to download time values.

However, there are differences in the performance of difference local policies since this time the upload client and the download client are on separate machines. **The download performance of local policies is very dependent on which client the upload/download client actually is.** For example, in the local policy, the Mumbai client uploads all its blocks to the Mumbai node, which is not local to the Ireland download client. So downloading performance at the Ireland client isn't even close to that of localhost. Also, local_closest policy gives the best performance in the 64KB block size category. **That's because besides the Mumbai node itself, Ireland is the second closest node to Mumbai (not in the geographic sense) in the virtual network space. Since the local_closest policy uploads a second copy of data to the Ireland node (second closest to the Mumbai node), the download speed or the Ireland client parallels downloading from localhost.**

d. Also comment on how the block size affects the download time for each policy.

Increasing block size can reduce download time, but the effect is more dependent on the policy we choose.

Random:

For both experiment 1 and 2, having a much larger block size can reduce the download time significantly. This because increasing block size can effectively reduce the overhead in downloading data, and storing data in random location makes downloading very slow.

Two random:

For both experiment 1 and 2, having a much larger block size can reduce the download time significantly. And the reason is same as the above.

Local:

For experiment 1, having a much smaller block size can reduce the download time a little bit, this is because uploading and downloading from the same server make downloading very fast and we can not observe a significant improvement by increasing block size.

However, for experiment 2, we can see a more significant deduction in download time because we download data from a different location, so larger block size makes downloading faster.

Local Closest:

For experiment 1, we can not see very significant influence since we are uploading and downloading in the same server, as the overhead of downloading from local is small, the overall download time is small.

For experiment 2, by applying the local closest policy, we the second copy is uploaded files to its closest node(which is Ireland), therefore, the effect of increasing block size is not significant as experiment 1 (the download client is also on the Ireland node).

Local Farthest:

For experiment 1, we can not see very significant influence since we are uploading and downloading in the same server, as the overhead of downloading from local is small, the overall download time is small.

For experiment 2, since the Ireland node is closest to the Mumbai node, a first copy is stored to the Mumbai and second copy is stored to a non-Ireland node. So downloading closest blocks from the Ireland equates to downloading blocks from the Mumbai, which is equivalent to local policy performance. Therefore, the effect of increasing block size is not significant as experiment 1 (the download client is also on the Ireland node).

However, despite the fact that the download time is at variance with the block size, its standard deviation isn't affected as much.

e. Based on these results, what are your overall comments on what you learned in this experiment (1-2 sentences)

It takes a lot of time to test, debug, and analyze distributed file system. Making multiple copies of data available at different locations enables faster download time by increasing the likelihood of having a copy of data available right next to the client. The best performance is to download from localhost.

f. Given this experience, what one or two additional experiments might you perform? (you don’t actually have to do them, just describe what they would look like and why you’d be interested in the results)

Consider the fetch feature: any node in the system now could fetch blocks from another node given their hash values. It will fetch from the closest available node if the client requests a block it has never seen before. With the same 1-site and 2-site upload/download client setup, investigate the download performance under the existing five policies as compared to not enabling this fetching feature.

I'm interested in this experiment under the assumption that the communication with a distributed storage system generally experience much less lag and much higher throughput. So it's assumed to be faster if the client sends request to the closest node and have that node help him fetch the data. But this assumption may not always be true, especially when there might be millions of such requests coming at once. So I want to investigation the tradeoff of this practice.