Netid: Joey Stack(jkstack2) Hanliang Jiang(hj33)

a) **Design:**

Our design for this MP included one leader node where all the file operations from the other VMS (nodes) went through. We decided to ensure that the load on the leader was reduced by having the leader respond with the addresses of VMs rather than the actual contents of a certain file. For example, if another node wanted to *put* a file into the SDFS, it would send the put request to the leader, and the leader would hash this filename and respond with a list of 4 replicas (if less nodes in membership list use all nodes as replicas) because there could be at most 3 simultaneous failures. The original node that made the file operation will then send *putfile* requests – along with the file content – directly to the 4 replicas. For a *get* operation, the original node will do the same and send the request to the leader, and the leader will respond with the list of 4 vm addresses that are containing the 4 replicas. The original node will then send a request to those addresses until it receives the file and then store that file locally.

In order to avoid starvation for both reads and writes, we utilized queues and kept track of the number of consecutive reads and writes we had. If there were 4 consecutive reads or writes, the leader will ensure that the other action will be performed, so starvation won't be present in our system.

For our leader election, we essentially elected the VM with the lowest VM number (ex. 9501) out of the currently alive nodes in the membership list, which every VM has the same membership list. When the leader either leaves the group or has failed, through the heartbeat failure detection protocol, another node will detect its failure and initiate an election run. It will check its VM number to see if it's the lowest. If it is, it will declare itself as leader and send an "elected" message out to all the nodes. If not, it will send an "election" message to all the nodes, which will acknowledge and the lowest VM number will then send out the "elected" message and declare itself leader.

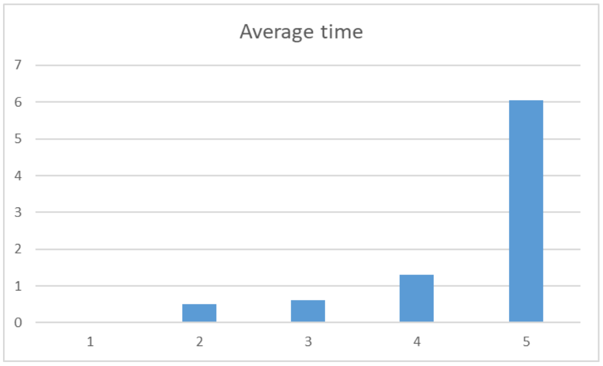
b) **Past MP Use:**

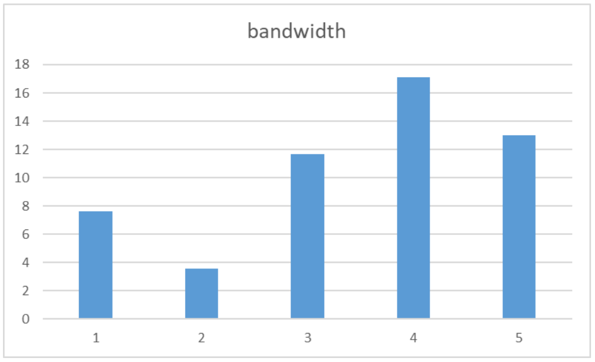
We utilized both MP1 and MP2 to help debug our problems and obstacles we ran into during MP3. Instead of having to constantly use print statements to debug what values certain variables were currently set to, we could just use the log files that were simultaneously being written to as the VM nodes were running, and search these log files (MP1). In addition to this, we also used the membership list in MP2 to assist with our MP3 implementation. For instance, we used the heart beating membership list protocol with our leader election algorithm. If the time before the next heartbeat received from a certain node was longer than the threshold, we mark the node as failed and remove it from the list. If this node was the leader, then we initiate an election run.

c) **Measurements:**

* run 5 trials for each and *plot trial # by read/write time* (5 plots total)

i. (Overheads) Re-replication time and bandwidth upon a failure (you can measure for different filesizes ranging up to 100 MB size, at least 5 different sizes)
Average time = [0.005618095,0.50340147,0.614497089,1.297905445,6.051176882]
Bandwidth = [7.600440536,3.555810036,11.68435152,17.10448175,13.02226022]





The Average time for getting the file from replicas increase as file size increases. The file sizes are 0.0427 KB, 1.79 MB, 7.18 MB, 22.2 MB and 78.8 MB. The average time for five restoring operations are 0.00609 s, 1.43424 s, 0.95231 s, 1.93788 s, 7.61355 s. The bandwidth can be calculated by file sizes divided by average time, which varies a lot across different file sizes.

ii. (Op times) Times to insert, read, and update, file of size 25 MB, 500 MB (6 total data points), under no failure

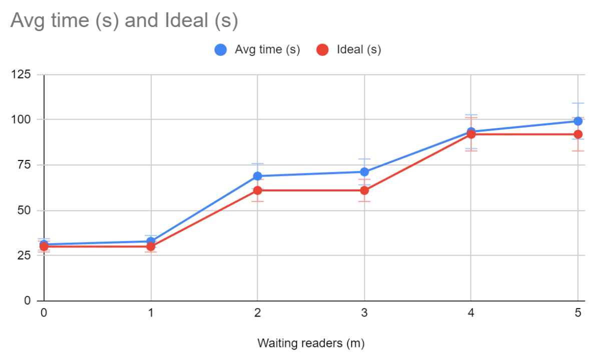
25MB:
Insert = 0.6473s; Read = 0.1738s; Update = 0.4728s
500 MB:
Insert = 8.2934s; Read = 1.8876s Update = 4.8426s

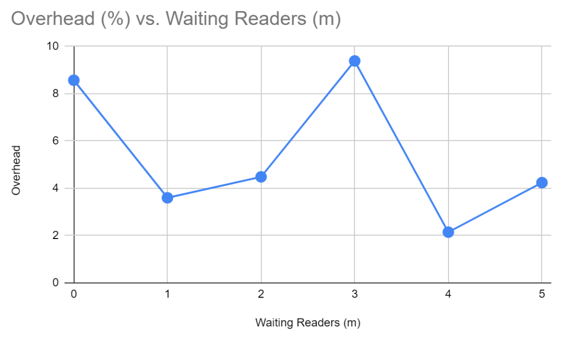
iii. (Large dataset) Time to store the entire English Wikipedia corpus into SDFS with 4 machines and 8 machines (not counting the leader)

8 machines:
Avg time = 110.3824s
Standard deviation = 5.821s
4 machines:
Avg time = 101.9921s
Standard deviation = 3.282s

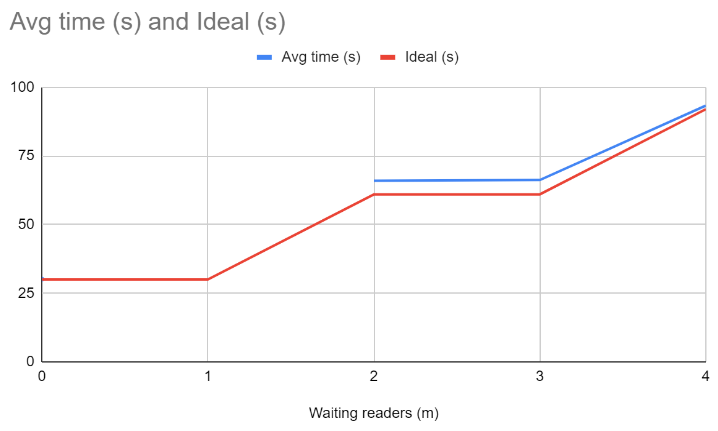
iv. (Read-Wait) Pick a large file that takes at least 30 s to read/write. Specify the file size, and the typical read time.

When several readers arrive at the same time as a file read starts, the time it takes for the last reader to finish shows a varying increase as more waiting readers are involved, as demonstrated by the actual completion times graphed against the number of waiting readers. This variability is also apparent in the overhead percentages, which measure how much the actual read times differ from an idealized linear increase in read times. The actual completion times initially begin slightly above the ideal times, even in the absence of waiting readers, and continue to rise as more readers join, but not in a strictly linear manner.





v. (Write-Read) Pick a large file that takes at least 30 s to read/write. Specify the file size, and the typical time to read and time to write.



*overlapping line above