MP-3 Report by Colin Zhou (colinz2) and Keyang Xuan (keyangx3)

Design: Our design involved having a leader machine, and client machines sending requests to the leader machine, who will then help execute various operations. We only used TCP for exchanging messages between machines. We used a FileTable class that contained two python dictionaries; one to track which files are stored in every machine, and the other to store the most recent versions of a file. In terms of replication, we had a function on the leader machine to detect whether or not a machine was removed from the membership list. Once that happened, the function would figure out what files were on that failed machine, and calculate the target machines to replicate the files to. The target replicas are randomly chosen out of a pool of machines (machines in that pool cannot already have that file, and cannot be failed). Once that happens, the leader will replicate to those target machines and send messages to everyone else with the updated FileTable. **Our replication level was 4, to account for 3 failures at the same time.**

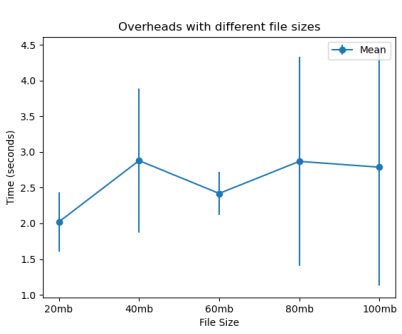
Past MP Use: We used the recommended solution for MP2 (python) as our failure detector and to maintain membership lists. MP2 especially helped us with replication, as it helped detect failed machines in time, for replication to occur. We utilized MP1 logging to keep track of logs for observing the state of the queue, and to check whether machines have failed.

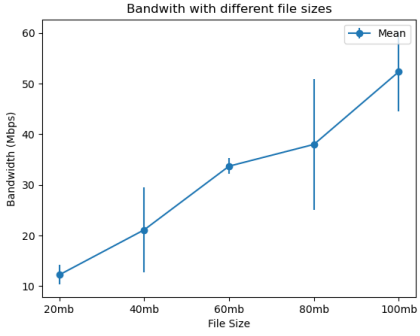
In order to maintain the sequential order of requests' execution, we designed a queue which executes the request based on following rules: Firstly, for the several continuous requests that each one points to a unique sdfs file, those requests will be executed in parallel. Secondly, if two continuous requests point to the same sdfs file, they will be executed parallelly if both are read requests, otherwise we will follow the FIFO rule that only executes the first request. Furthermore, to make our system starvation-free, we don't allow that one read/write to wait for more than four write/read requests which point to the same sdfs file. Therefore, we will forcibly switch the order between the fourth read/write request to the waiting write/read operation.

Measurements:

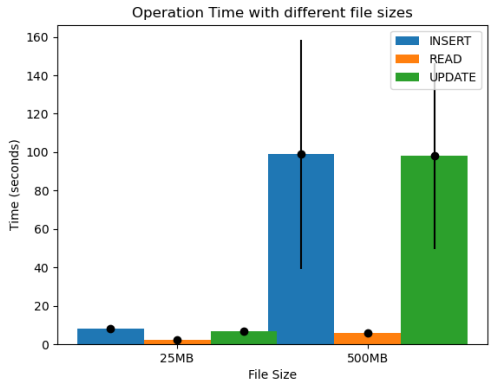
Overheads:

From the plot, we see that the overheads for rereplication is relatively stable regardless of file sizes, which may be indicative of the behavior of the "scp" command in python and the fact that we are launching threads to execute the replicate requests. The bandwidth increased alongside the file size, which makes sense since more data is being transferred over to different machines.

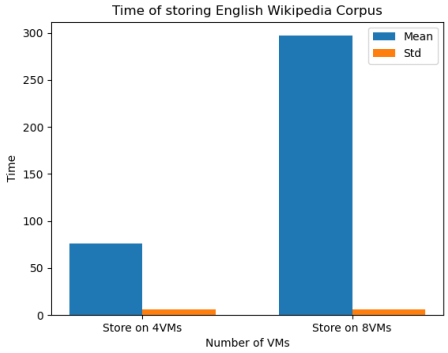




Op times:
From the plot, we can see that when the file size increases, the time for all operations are increased. Also, the larger file size is, the larger variance of execution time is, which makes sense since large file sizes are more likely to lead to result in larger bandwidth.

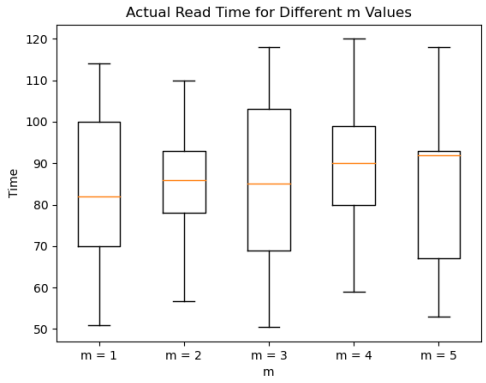


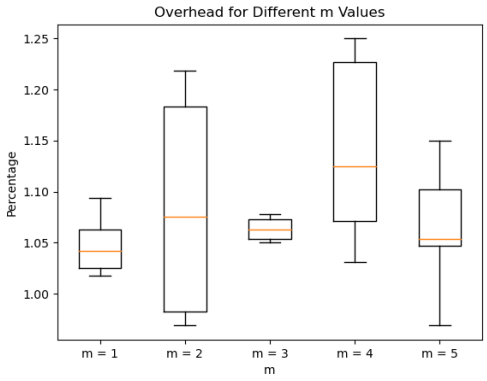
Large dataset:
From the plot, we can notice that the time to store English Wikipedia corpus on 8 VMs is much longer than the time to store it on 4 VMs. However, every round for storing is relatively consistent on both the 4 VM case and 8 VMs case. It makes sense that storing on more machines is going to take longer.



Read-Wait:

We used a 1GB file to test. The actual read time is varying between 80 and 90 seconds depending on the M readers. We think that this kind of variability may be due to us launching parallel threads to allow 2 readers to read a file at the same time. We see that the % above the ideal time above the baseline using the formula given is substantial, because of network latencies, and other possible environmental factors. In the overhead for different m values graph, the percentage is referring to the percentage above the baseline (calculated using the formula) provided in the spec sheet.





Write-Read

We use a file whose size is 1.56GB to test (the write time is 85s). From the result, we can see that the average execution time is increasing with the increment of the number of writers. Also, the variance of time is also increasing with the increase of the number of writers. This makes sense due to that more requests will be more likely to cause the stagnation in bandwidth, which leads to the execution time being further then the ideal execution time.