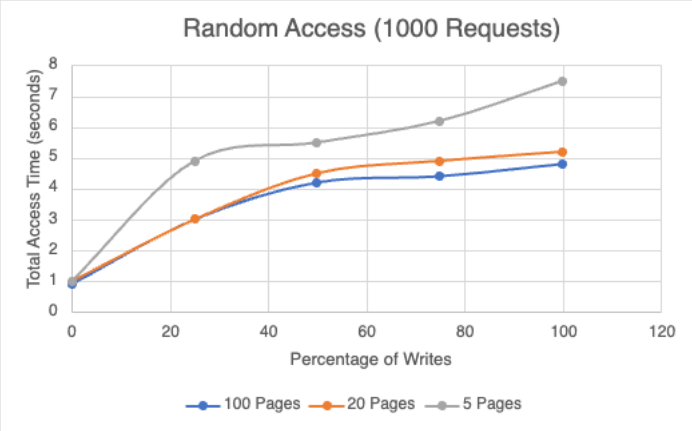
CS739 P3: DISTRIBUTED SHARED MEMORY

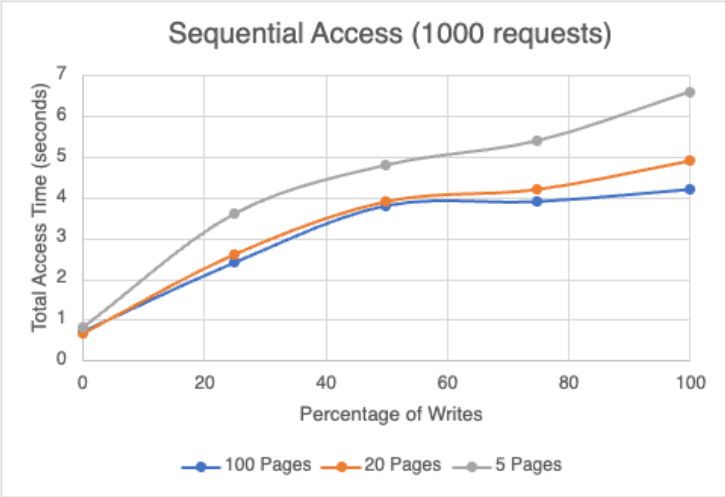
Deep Jiten Machchhar, Vishnu Ramadas, Selvaraj Anandaraj

We build a distributed shared memory service spread across different physical machines. Our service enables applications to access and remotely operate on data residing on different machines. We use a master service distributed across all the machines in our system to track the pages being shared across machines. This master service handles client page requests and grants them access after ensuring that the other clients currently using it are duly notified. A page can be accessed in either an exclusive state, a shared state, or an invalid state. Any access to a page in invalid state raises a fault which is caught by a fault handler. This fault handler requests the master service to grant itself the required level of access to the page it accesses. If any client wants to update the data stored on a page, it sends a request to the master asking for exclusive access. The master invalidates all other page copies, if any, and provides the client with write permission. Similarly, if client requests read access, the master service grants it read permission even if other clients have access to the page. If any client has exclusive access and it gets a read request, it is demoted to a shared state while clients with read access already remain in the shared state. This model ensures that the distributed shared memory service guarantees sequential consistency, caching and coherency to all its clients.

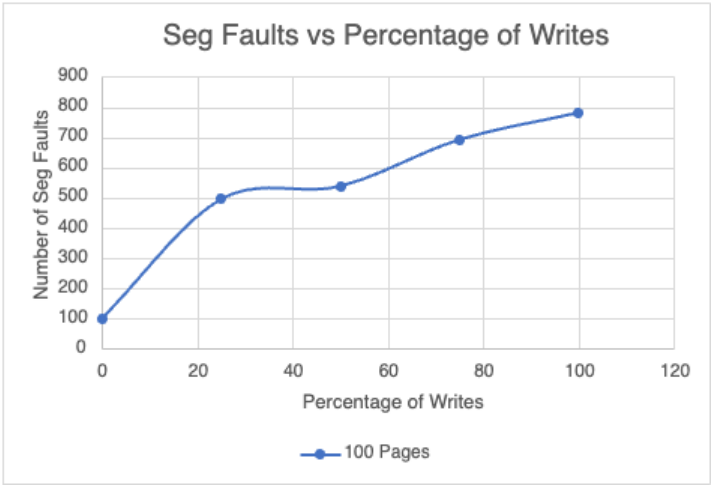
EXPERIMENTAL RESULTS:



The above graph shows the performance variation using the total time taken for access. We make this study for various percentage of writes. We also vary the number of pages taken by the application to scatter its variables. We use a random-access pattern so that we would make the duration of a page being non-dirty on a client is very less, this can be done by another page writing this page between two random accesses. So, more the page count, it's more probable the accesses are spread across pages and spread across in terms of duration too. This helps us emulate the above behavior.



We try to do the same thing with sequential access. We define sequential access as an access with more locality with a page. This workload pattern helps us retain a page non-dirty for a longer duration. We do the same study for this workload. We also do note that the access within a page is random, but strictly local to the page. So, the term sequential might be a bit confusing.



We wanted to correlate the relation of the number of segmentation faults with the time taken to execute for random accesses. We try to perform that experiment on accesses spread across 100 pages. We see that the number of segmentation faults increases as we increase the write percentage, this directly correlates to the increase in the execution time shown above for random accesses. This is because each segmentation fault causes a network call.

CONCLUSION:

Hence, we successfully built and tested a distributed shared memory system.