Khuzaima Mushtaq
CSS 430

Project 3 Report

Cache Specs/ Design:

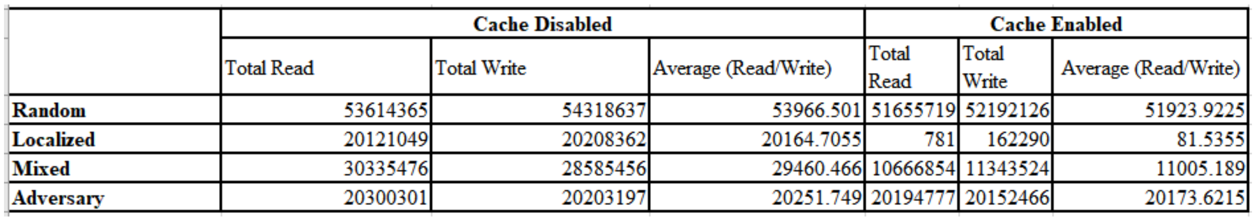
This section explains my Cache design and specification, including the Enhanced Second Chance algorithm. To start off, my Cache uses a CacheEntry private subclass to hold each CacheEntry. Each CacheEntry has a byte[] DATA, an int frameNo for corresponding disk block, and Booleans for useBit and modifiedBit. Using the CacheEntry class I made a CacheEntry[] Array that holds all the CacheEntry's. I also have private ints pageSize for blocksize, cacheSize for total number of CacheEntry's, and scPtr which is the Second Chance pointer.

For the Read and Write Methods my algorithm is very similar. Firstly, I check to see if the blockId is out of bounds, then I check to see if the blockId is in the Cache already, if Entry is already in cache I perform the action Read/Write and update useBit/ modifiedBit as needed. If entry is not in Cache then I Check if there is an open CacheEntry and use that, if not, I use my findVictim() Method to find an entry to remove as specified by the Second Chance algorithm. When overwriting CacheEntry's I check if the modifiedBit is set and write to disk if needed.

My Enhanced Second Chance algorithm is based on sets of 2 while loops. In the First while loop I check if there are any entry's that are u = 0 and m = 0 I also do not change any u = 1 to 0. IF this loop does not find an entry to replace then I hop into second loop. In the second while loop I check for u = 0 and m =1 and also set any 1's to 0 as I go through. IF, even after this I do not have an entry to replace I reset to the starting second chance pointer and do loops 1 and 2 again which are guaranteed to find a match because the second loop puts 1's to 0's.

Performance Tests:

This section discusses the results of using a Cache vs not in regard to the 4 different test cases. For all my test I do 1000 writes and 1000 reads. I create a byte double array as byte[1000][512] and use that reading/writing For all tests the enabled and disabled tests are the same except that I use cread vs rawread and cwrite cs rawwrite. The times below are in Microseconds. For comparison 53614365 microseconds is ~53.614 seconds.



	Cache Disabled			**Cache Enabled**		
	Total Read	Total Write	Average (Read/Write)	Total Read	Total Write	Average (Read/Write)
Random	53614365	54318637	53966.501	51655719	52192126	51923.9225
Localized	20121049	20208362	20164.7055	781	162290	81.5355
Mixed	30335476	28585456	29460.466	10666854	11343524	11005.189
Adversary	20300301	20203197	20251.749	20194777	20152466	20173.6215

ONE: Random Access

For my Random test I use Math.random() *999 + 0 to randomize the blockId for the reads and the writes and do this 1000 times. As for the performance Evaluation, the results are as expected. Since the Random Access really does not use the Cache it is obvious that the performance would not be increased much if at all.

TWO: Localized

For the Localized test I go through a loop 1000 times and read/write to only blockId's 1-10 using I %10. This leads to almost 100% Cache hit rate. The performance evaluation for this test was the most surprising. The speed increase is insane, many degrees faster. Up to 10000 time faster. This also makes sense though because the OS has to never access the Disk it has an almost 100% hit rate on the Cache aside from the first 10 lookups, which account for the increased time for the Total Write.

THREE: Mixed

For my mixed test I combine my Random and Localized such that 10% of the time I do a random access and the rest of the time I access BlockId's 1-10. The performance is as predicted for this test as well. The use of the cache in the localized hits ~90% of the time show a 10-20x faster speed.

FOUR: Adversary

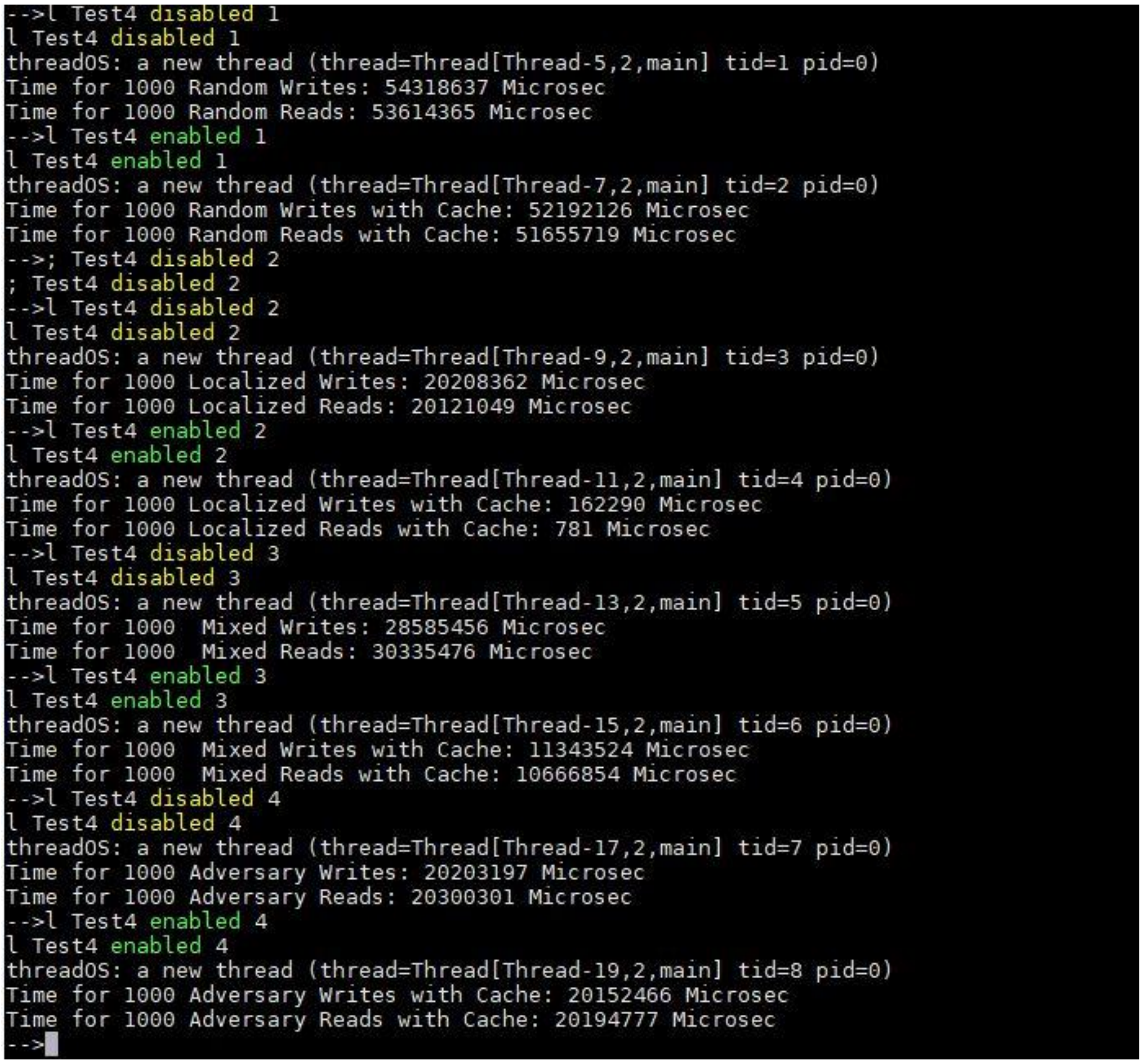
For my Adversary Test I do a sequential read/Write such that there is never a Cache Hit because we are trying to access a different BlockId Everytime 1-1000. For this one there isn't a significant difference between the enabled and disabled tests again because the Cache is not used since we do not hit it at all.

OVERALL Performance:

Overall, I was surprised by Localized test and the Random Test the most. The random test takes significantly longer than even the Adversary Test, I can only imagine that this is because of the Random element adding overhead to the reads/writes. Since the program has to calculate random numbers a lot of times. Otherwise the results are as expected between the disabled and enabled versions of the tests!

Linux Test Screen Shot Below!

Results Linux Screenshot:



```
-->l Test4 disabled 1
l Test4 disabled 1
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)
Time for 1000 Random Writes: 54318637 Microsec
Time for 1000 Random Reads: 53614365 Microsec
-->l Test4 enabled 1
l Test4 enabled 1
threadOS: a new thread (thread=Thread[Thread-7,2,main] tid=2 pid=0)
Time for 1000 Random Writes with Cache: 52192126 Microsec
Time for 1000 Random Reads with Cache: 51655719 Microsec
-->; Test4 disabled 2
; Test4 disabled 2
-->l Test4 disabled 2
l Test4 disabled 2
threadOS: a new thread (thread=Thread[Thread-9,2,main] tid=3 pid=0)
Time for 1000 Localized Writes: 20208362 Microsec
Time for 1000 Localized Reads: 20121049 Microsec
-->l Test4 enabled 2
l Test4 enabled 2
threadOS: a new thread (thread=Thread[Thread-11,2,main] tid=4 pid=0)
Time for 1000 Localized Writes with Cache: 162290 Microsec
Time for 1000 Localized Reads with Cache: 781 Microsec
-->l Test4 disabled 3
l Test4 disabled 3
threadOS: a new thread (thread=Thread[Thread-13,2,main] tid=5 pid=0)
Time for 1000  Mixed Writes: 28585456 Microsec
Time for 1000  Mixed Reads: 30335476 Microsec
-->l Test4 enabled 3
l Test4 enabled 3
threadOS: a new thread (thread=Thread[Thread-15,2,main] tid=6 pid=0)
Time for 1000  Mixed Writes with Cache: 11343524 Microsec
Time for 1000  Mixed Reads with Cache: 10666854 Microsec
-->l Test4 disabled 4
l Test4 disabled 4
threadOS: a new thread (thread=Thread[Thread-17,2,main] tid=7 pid=0)
Time for 1000 Adversary Writes: 20203197 Microsec
Time for 1000 Adversary Reads: 20300301 Microsec
-->l Test4 enabled 4
l Test4 enabled 4
threadOS: a new thread (thread=Thread[Thread-19,2,main] tid=8 pid=0)
Time for 1000 Adversary Writes with Cache: 20152466 Microsec
Time for 1000 Adversary Reads with Cache: 20194777 Microsec
-->
```