Overview and Sample Results

Sunday, November 11, 2018 6:19 PM

Program 4 Overview

Figure out what you are required to do

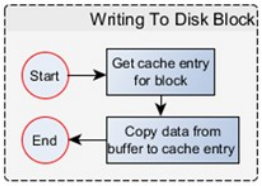
- Goal: Write a Disk cache to make disk operations more efficient and write a test to evaluate the performance improvement.
- How to start?
  - What is a cache?
  - Why do we need it? Think about memory hierarchy
  - **Bottom up design**: start your design with the test!
    - What **end tasks** are you supposed to accomplish?
    - What files, classes, concepts do you need to accomplish the **end tasks**?

Look at the requirements, focus on the test

- So, if we start by looking at what the test should do:
  - **Random accesses**
    - The selected blockID's are chosen at random using a random number generator
  - **Localized accesses**
    - The system will create a cache with a size of 10 entries, so selecting blockID's in such a small range allows nearly all of them to fit into cache
  - **Mixed accesses**
    - Mixes random accesses with localized accesses with a general rate of 90% localized and 10% random
  - **Adversarial accesses**
    - Purposefully pick a series of blockID's that won't be in the cache when you try to access them
- You should think about tests that use the entirety of DISK blocks, i.e., 1000
  - Suggested: Use read AND write
    - Write to a bunch of blocks first
    - Then use read to verify that what you wrote is correct
- You should also think about WHAT TO EXPECT from the performance results… understand what each test entails and how it affects performance, e.g., think about locality.

Cache functionality

- The only feature this cache really provides is the ability to read and write to disk blocks
  - Logic for read/write is the same



  - To find a particular block
    1. Check last block used
    2. Search cache for an entry associated with the block
    3. Use a free slot in the cache
    4. Find a victim, sync it to disk if needed, swap it out, and create a new entry for the desired block
- Disk Cache will use enhanced second-chance algorithm

The Second Chance Algorithm

- This algorithm uses dirty and reference bits to decide how viable of a candidate is as a victim:
  - Dirty – 0, Reference – 0: the best candidate for a victim
  - Dirty – 1, Reference – 0: the second best candidate for a victim
  - Dirty – 0, Reference – 1: the third best candidate for a victim
  - Dirty – 1, Reference – 1: the worst candidate for a victim
- To easily compare two cache entries, the states of the dirty and reference bits are used to calculate a victim level (from 0-3)
  - For each cache entry

- If there is a level 3 victim à choose it
- Else, is the victim level higher than the current best known victim?
  - Yes: use this entry as best known victim
- Mark cache entry as not recently used
- Choose best known victim

Sample Results

Disclaimer: The actual assignment description takes precedence over anything stated herein. Specifically, if something here contradicts the assignment description, that specification will determine what you should expect and do.

Note that best way to collect and summarize your results by taking several time measurements and then average them, do not use single time measurement.

The following results were submitted as part of a couple of former student reports. I'm giving you this information so that you can compare against your own results.

Example 1: pretty plots

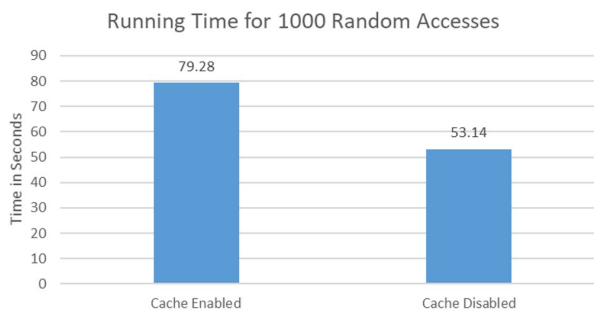
I chose these particular results because this is a great and very professional way to show and summarize your results. However, you are not allowed to copy the format or the numbers. In addition, the results presented are specific to that implementation and have a corresponding interpretation to explain "weird" results (e.g., the Random Access results may seem the opposite to what you get). If you decide to use plots, use your own formats, create your own charts, and **don't forget that you need to interpret the results -- a plot or table without interpretation and explanation will receive NO points.**

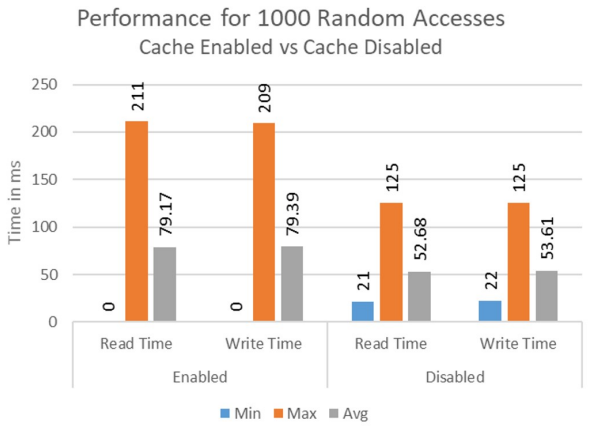
! **Scroll all the way down for Example 2!!**

The following results were collected in 2018

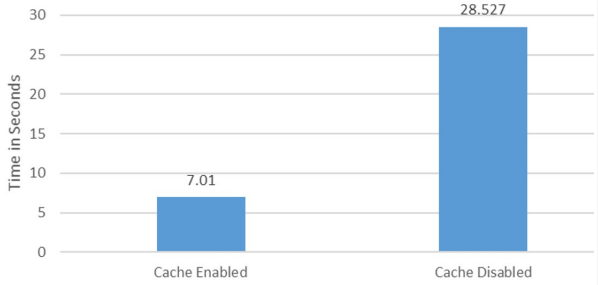
Random Access

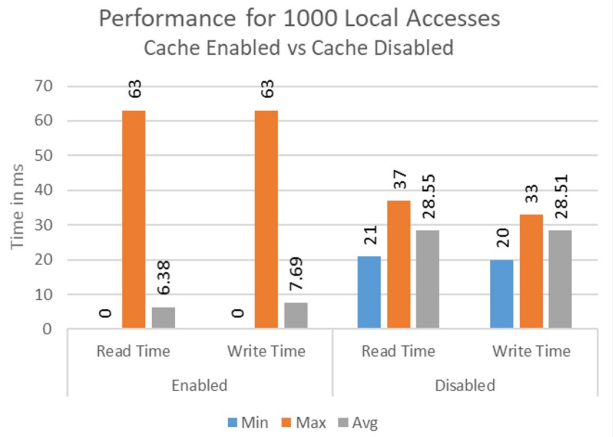
In most implementation you would expect opposite measurements (this is implementation specific, make sure you are able to explain YOUR results).



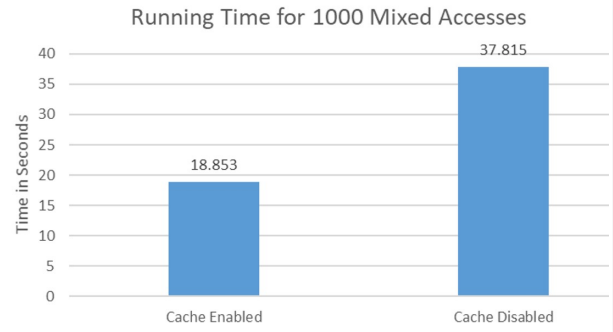


Local Access

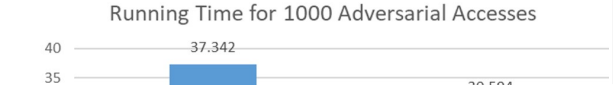


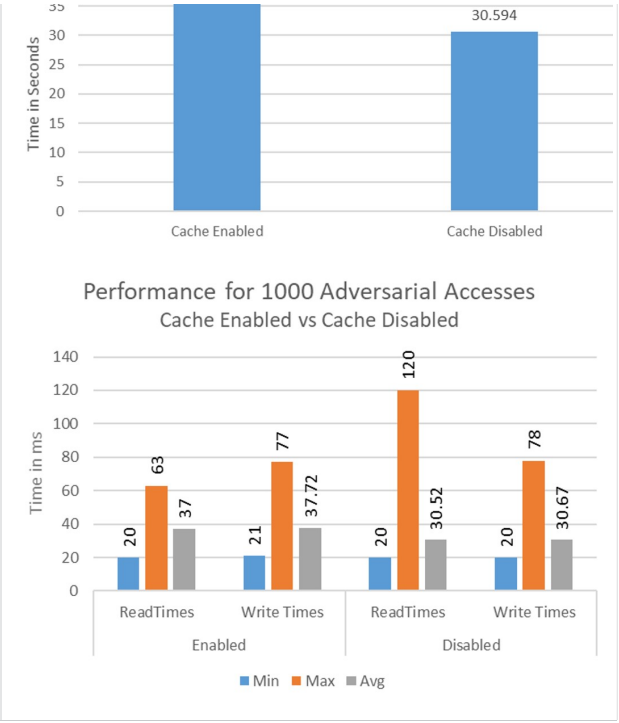


Mixed Access



Adversarial Access





Example 2: summary table

The following results were collected in 2017

More concise, yet less appealing (user friendly) way of showing results… also *less* professional but still nice and clean. This is still a way better format to present your results than just showing screenshots of your runs -- ***don't forget that you are required to have screenshots in addition to summary results!***

Also, Random access results are more in line to what you would expect.

	Cache Enabled		Cache Disabled	
	Average Write	Average Read	Average Write	Average Read
Random Access	13,965,618.6 ns	14,167,452.55 ns	23,574,175.2 ns	23,266,153.51 ns
Localized Access	2,366,686.3 ns	2,437,238.6 ns	23,333,433.9 ns	23,121,311.59 ns
Mixed Access	5,088,115.95 ns	14,500,747.53 ns	23,879,607.8 ns	24,184,063.6 ns
Adversary Access	15,944,454.75 ns	24,365,830.19 ns	23,962,449.45 ns	23,107,879.93 ns