CS553 Homework #3

Benchmarking Storage

Instructions:

- *Assigned date: Friday March 13th, 2020*
- *Due date: 11:59PM on Friday March 27th, 2020*
- *Maximum Points: 100%*
- *This homework can be done in groups up to 3 students*
- *Please post your questions to the Piazza forum*
- *Only a softcopy submission is required; it will automatically be collected through GIT after the deadline; email confirmation will be sent to your HAWK email address*
- *Late submission will be penalized at 10% per day; an email to the TA with the subject "CS553: late homework submission" must be sent*

1 Your Assignment

This project aims to teach you how to benchmark storage systems. You can be creative with this project. You must use either the C or C++ programming languages. Libraries such as PThreads will be necessary to complete the assignment. Libraries such as STL could be used if necessary. Other programming languages will not be allowed due to the variability of difficulty between languages, and increased complexity in grading. Do not write code that relies on complex libraries (e.g. boost), as those will simplify certain parts of your assignments, and will increase complexity in grading. If you are not sure if certain libraries are allowed, ask the TAs.

You can use any Linux system for your development, but you must use the Chameleon testbed [https://www.chameleoncloud.org]; more information about the hardware in this testbed can be found at https://www.chameleoncloud.org/about/hardware-description/, under Standard Cloud Units. Even more details can be found at https://www.chameleoncloud.org/user/discovery/, choose "Compute", then Click the "View" button. You are to use "Compute Haswell" node types; if there are no Haswell nodes available, please use "Compute Skylake" node types. You are to use the advanced reservation system to reserve 1 bare-metal instance to conduct your experiments. You will need to assign your instance a floating IP address so that you can connect to your instance remotely.

In this project, you need to design a benchmarking program that evaluates the storage system. You will perform strong scaling studies, unless otherwise noted; this means you will set the amount of work (e.g. the number of objects or the amount of data to evaluate in your benchmark), and reduce the amount of work per thread as you increase the number of threads. The TAs will compile (with the help of make) and test your code on Chameleon bare-metal instances (Haswell or Skylake). If your code does not compile and the TAs cannot run your project, you will get 0 for the assignment.

1. **Disk:**
   a. Implement: MyDiskBench benchmark; ***Hint:*** *there are multiple ways to read and write to disk, explore the different APIs, and pick the fastest one out of all them; also make sure you are measuring the speed of your disk and not your memory (you may need to flush your disk cache managed by the OS)*
   b. Dataset: 10GB data split up in 7 different configurations (note these are similar to the way IOZone deals with multi-threading and multiple concurrent file access):

    i. D1: 1 file of 10GB size for a total of 10GB of data
    ii. D2: 2 files of 5GB size each for a total of 10GB of data
    iii. D3: 4 files of 2.5GB size each for a total of 10GB of data
    iv. D4: 8 files of 1.25GB size each for a total of 10GB of data
    v. D5: 12 files of 833.33MB size each for a total of 10GB of data
    vi. D6: 24 files of 416.67MB size each for a total of 10GB of data
    vii. D7: 48 files of 208.33MB size each for a total of 10GB of data

c. Workload:
    i. Operate over 10GB data (in 7 different configurations D1, D2, D3, D4, D5, D6, and D7) 1X times with various access patterns and various record sizes
    ii. Access pattern
        - WS: write with sequential access pattern
        - RS: read with sequential access pattern
        - WR: write with random access pattern
        - RR: read with random access pattern
    iii. Record size
        - 64KB, 1MB, 16MB

d. Concurrency:
    i. Use varying number of threads (1, 2, 4, 8, 12, 24, 48) to do the I/O operations

e. Measure:
    i. throughput, in terms of MB per second; report data in MB/sec, megabytes (10^6) per second; these experiments should be conducted over 10GB of data

f. Run the Disk benchmark IOZone benchmark (http://www.iozone.org/) with varying workloads and measure the disk performance:
    i. Threads:
        - 1, 2, 4, 8, 12, 24, and 48 threads
    ii. Record:
        - Throughput (MB/sec): 64KB, 1MB, 16MB
        - Latency (OPS/sec): 4KB
    iii. Access Patterns:
        - 0 (sequential write), 1 (sequential read), and 2 (random read/write)
    iv. All other parameters for IOZone should be kept as close as possible to your own implementation
    v. You are likely to use the following command line arguments for iozone:
        - -T -t -s -r -F -I -+z -O

g. Compute the theoretical bandwidth of your disk; you can use information from Chameleon and the manufacturer of the disk to figure this out (the number should be a constant value that you list in the entire column Theoretical Throughput. What efficiency do you achieve compared to the theoretical performance? Compare and contrast your performance to that achieved by IOZone, MyDiskBench**,** and to the theoretical performance.

h. Fill in the table 1 below for Disk Throughput; table 1a table should be for workload WS, table 1b should be for workload RS, table 1c should be for workload WR and table 1d for workload RR:

Work-load	Con-currency	Record Size	MyDiskBench Measured Throughput (MB/sec)	IOZone Measured Throughput (MB/sec)	Theoretical Throughput (MB/sec)	MyDiskBench Efficiency (%)	IOZone Efficiency (%)
WS/RS WR/RR	1	64KB					
WS/RS WR/RR	1	1MB					
WS/RS WR/RR	1	16MB					
WS/RS WR/RR	2	64KB					
WS/RS WR/RR	2	1MB					
WS/RS WR/RR	2	16MB					
WS/RS WR/RR	4	64KB					
WS/RS WR/RR	4	1MB					
WS/RS WR/RR	4	16MB					
WS/RS WR/RR	8	64KB					
WS/RS WR/RR	8	1MB					
WS/RS WR/RR	8	16MB					
WS/RS WR/RR	12	64KB					
WS/RS WR/RR	12	1MB					
WS/RS WR/RR	12	16MB					
WS/RS WR/RR	24	64KB					
WS/RS WR/RR	24	1MB					
WS/RS WR/RR	24	16MB					
WS/RS WR/RR	48	64KB					
WS/RS WR/RR	48	1MB					
WS/RS WR/RR	48	16MB					

i. Fill in the table 2 below for random disk operations per second (measured in OPS/sec); table 2a table should be for workload WR and table 2b for workload RR.

Work-load	Con-currency	Record Size	MyDiskBench Measured IOPS (OPS/sec)	IOZone Measured IOPS (OPS/sec)	Theoretical IOPS (OPS/sec)	MyDiskBench Efficiency (%)	IOZone Efficiency (%)
WR/RR	1	4KB					
WR/RR	2	4KB					
WR/RR	4	4KB					
WR/RR	8	4KB					
WR/RR	12	4KB					
WR/RR	24	4KB					
WR/RR	48	4KB					

Other requirements:

- You must write all benchmarks from scratch. Do not use code you find online, as you will get 0 credit for this assignment. If you have taken other courses where you wrote similar benchmarks, you are welcome to start with your codebase as long as you wrote the code in your prior class.
- All of the benchmarks will have to evaluate concurrency performance; concurrency can be achieved using threads. Use strong scaling in all experiments, unless it is not possible, in which case you need to explain why a strong scaling experiment was not done. Be aware of the thread synchronizing issues to avoid inconsistency or deadlock in your system.
- All benchmarks can be run on a single machine.
- Not all timing functions have the same accuracy; you must find one that has at least 1ms accuracy or better, assuming you are running the benchmarks for at least seconds at a time.
- Since there are many experiments to run, find ways (e.g. scripts) to automate the performance evaluation. Besides BASH scripts, it is possible to automate your experiments using the "parallel" tool in Linux.
- For the best reliability in your results, repeat each experiment 3 times and report the average and standard deviation. This will help you get more stable results that are easier to understand and justify.
- Don't forget to benchmark your disk, and not your memory. You may need to flush caches that might be stored in memory.
- You may find it more efficient to deal with binary data when reading or writing in this evaluation.
- No GUIs are required. Simple command line interfaces are required. Make your benchmark and iozone as similar as possible from a command line argument and how the program behaves.

3 What you will submit

When you have finished implementing the complete assignment as described above, you should submit your solution to your private git repository. Each program must work correctly and be detailed in-line documented. You should hand in:

1. **Source code (30%):** All of the source code; in order to get full credit for the source code, your code must have in-line documents, must compile (with a Makefile), and must be able to run a variety of benchmarks through command line arguments (matching relevant iozone command line arguments).

2. **Readme (10%):** A detailed manual describing how the program works. The manual should be able to instruct users other than the developer to run the program step by step. The manual should contain example commands to invoke the benchmark. This should be included as readme.txt in the source code folder.
3. **Report / Performance (60%):** Must have working code that compiles and runs on Chameleon assuming Ubuntu Linux 18.04 to receive credit for report/performance. If your code requires non-standard libraries to compile and run, include instructions on what libraries are needed, and how to install them. Furthermore, the code must match the performance presented in the report. A separate (typed) design document (named hw3-report.pdf) of approximately 1-3 pages describing the overall benchmark design, and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made). Since this is an assignment aimed at teaching you about benchmarking, this is one of the most important part; you must evaluate the disk benchmarks with the entire parameters space outlined. You must produce 6 tables (based on the tables included above) to showcase the results; we encourage you to create additional figures to highlight your performance evaluation based on the data in your tables. Please combine data and plot on the same graph wherever possible, for either compactness reasons, or comparison reasons. Don't forget to plot the average and standard deviation. Also, you need to explain each graph's results in words. Hint: graphs with no axis labels, legends, well defined units, and lines that all look the same, are likely very hard to read and understand graphs. You will be penalized if your graphs are not clear to understand. Please specify which student contributed to what benchmark experiments, as well as what part of the code or scripts each student contributed to.

Submit code/report through GIT. If you cannot access your repository contact the TAs. You can find a git cheat sheet here: https://www.git-tower.com/blog/git-cheat-sheet/

Grades for late programs will be lowered 10% per day late.