CS351 Lab #4

Benchmarking Storage

Instructions:

- *Assigned date: Monday March 21st, 2022*
- *Due date: 11:59PM on Friday April 1st, 2022*
- *Maximum Points: 40*
- *This lab must be done individually*
- *Please post your questions to BB*
- *Only a softcopy submission is required; it will automatically be collected through GIT at the deadline; email confirmation will be sent to your HAWK email address; late submission will be penalized at 10% per day; your submission will not be graded until you have submitted a single page PDF file with your name, email, A#, assignment number, and URL to the gitlab repo*

1 Your Assignment

This project aims to teach you about file I/O and benchmarking. You must use the C programming languages. You can use stdio.h, stdlib.h, unistd.h, wait.h, time.h, and string.h; no other libraries or header files can be used to complete the assignment, without express permission. Libraries such as STL or Boost cannot be used. You can use any Linux system for your development, but you must make sure it compiles and runs correctly on fourier. The performance evaluation should be done on your own computer in a Linux environment (e.g. think back at Lab #0). Given the large class size, we encourage all students to avoid using fourier for anything but testing functionality on small datasets.

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 amount of data to evaluate in your benchmark), and reduce the amount of work per process as you increase the number of processes. You are to create a makefile that helps build your benchmark, as well as run it through the benchmark bash script.

1. **Disk:**
    a. Implement: FileIO benchmark
    b. Datasets: 4MB and 1GB data sets split up in 4 different configurations each:
        i. D1: 1 file of 4MB size for a total of 4MB of data
        ii. D2: 2 files of 2MB size for a total of 4MB of data
        iii. D3: 4 files of 1MB size for a total of 4MB of data
        iv. D4: 8 files of 500KB size for a total of 4MB of data
        v. D5: 1 file of 1GB size each for a total of 1GB of data
        vi. D6: 2 files of 500MB size each for a total of 1GB of data
        vii. D7: 4 files of 250MB size each for a total of 1GB of data
        viii. D8: 8 files of 125MB size each for a total of 1GB of data
    c. Workload:
        i. Operate over 1GB data (in 4 different configurations D5, D6, D7, and D8) 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 processes (1, 2, 4, 8) to do the I/O operations

e. Measure:
   i. latency, in terms of operations per second; report data in ops/sec; these experiments should be conducted over 1GB of data; these experiments should be conducted over 4MB data set (D1, D2, D3, and D4)
   ii. throughput, in terms of MB per second; report data in MB/sec, megabytes (10^6) per second; these experiments should be conducted over 1GB data set (D5, D6, D7, and D8)

f. Run the Disk benchmark IOZone benchmark (http://www.iozone.org/) with varying workloads and measure the disk performance:
   i. Processes:
      - 1, 2, 4, 8
   ii. Record:
      - Latency (OPS/sec): 4KB
      - Throughput (MB/sec): 64KB, 1MB, 16MB
   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. Compare and contrast your performance to that achieved by IOZone and FileIO.

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:

Workload	Concurrency	Record Size	FileIO Measured Throughput (MB/sec)	IOZone Measured Throughput (MB/sec)
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		

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.

Workload	Concurrency	Record Size	FileIO Measured IOPS (OPS/sec)	IOZone Measured IOPS (OPS/sec)
WR/RR	1	4KB		
WR/RR	2	4KB		
WR/RR	4	4KB		
WR/RR	8	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.
- All of the benchmarks will have to evaluate concurrency performance; concurrency can be achieved using processes or threads; you must use processes for this assignment. Use strong scaling in all experiments.
- All benchmarks must be compiled and tested for functionality on fourier.
- All benchmarks can be run on your own Linux system (preferred), or on fourier (discouraged).
- 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, you must use a bash script to automate the performance evaluation.
- There are many ways to create files for your datasets (e.g. truncate, fallocate, head, dd, etc); pick a method, it's not important which approach you use.
- 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).
- You may find it more efficient to deal with binary data when reading or writing in this evaluation. Either one is fine.
- You may need to install IOZone (http://www.iozone.org); you can compile and install it on fourier from http://www.iozone.org/src/current/iozone3_490.tar, as a regular user without sudo permissions; for Ubuntu Linux systems you have sudo permissions on, you can use the *apt* manager to install it.
- No GUIs are required. Simple command line interfaces are required.

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 and compilation (70%):** All of the source code in C and Bash; 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. It must have a working bash script for FileIO benchmark, and a separate bash script for IOZone benchmark. Must have working code that compiles and runs on fourier.
2. **Report / Performance (30%):** A separate (typed) design document (named lab3-report.pdf) describing the results in a table format. You must evaluate the disk benchmarks with the entire parameters space outlined; there are 6 tables (tables 1a, 1b, 1c, 1d, 2a, and 2b) to showcase the results. You must summarize your findings, in terms of scalability, concurrency, performance, and

comparison between your FileIO benchmark and the IOZone benchmark. For your particular setup (everyone will be different since you are running on different hardware), what seems to be the best configuration and worst configuration for each workload, in terms of concurrency level and record size? Highlight the best one in green, and worst one in red for each table, for each benchmark. Can you explain in words why they are the best, or the worst, and why the data you have makes sense.

To submit your work, simply commit all your changes to the Makefile, fileio.c, test-fileio.sh, and test-iozone.sh files and push your work to Github. You can find a git cheat sheet here: https://www.git-tower.com/blog/git-cheat-sheet/. Your solution will be collected automatically at the deadline. If you want to submit your homework later, you will have to push your final version to your GIT repository and you will have let the TA know of it through email. There is no need to submit anything on BB for this assignment. If you cannot access your repository contact the TAs.

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