SE 320 Homework 4 Write-up

1. A short (one paragraph) explanation of why the benchmark harness is structured as it is. Why is there an initial run that isn't counted in the min/max/average? Why are we measuring multiple runs?

The benchmark harness is structured as it is to run certain operations (proportionally distributed) across multiple runs. An initial run isn't counted in the min/max/average because it is a warmup run. As implied by the name, this run is meant to serve as an initial start-up for the application and subsequent benchmark. This allows for operations to be successfully run at least once and certain machine-level mechanisms to become accustomed with the application. Multiple runs are measured to test variability within the application — the same results should not be expected every time. By executing multiple runs, certain performance statistics can be calculated across all runs, such as min, max, and average run time.

2. Reasonable operational profiles for use of the RedBlackBST in the following use cases: logging data structure, read-heavy database, read-only database

NOTE: For the operational profiles, I will list the percentages in insert/delete/lookup format. So, for an operational profile that is 50% inserts, 25% deletes, and 25% lookups, I will list it as 50/25/25. This notation will be used throughout the write-up.

The operational profile I created for the logging data structure is 80/10/10. This allows for the majority of operations performed to be inserts, with a much smaller percentage going towards deletes and lookups.

For the read-heavy database, the operational profile I created is 10/10/80. This is a similar structure as the logging data structure, except this time, the majority of operations performed goes towards lookups.

Lastly, the operational profile I created for the read-only database is 0/0/10. This way, the only operations that are performed are lookups and no modification-based operations are performed at all.

3. A description of what steps you took in preparation for your performance measurements.

To prepare for measuring performance, I closed all applications that were not essential to editing and running the application benchmarks. I also kept my web browser open, since I am using an online document editor to create this write-up (there were 5 tabs open). This means that the only applications that were open on my PC while running the benchmarks were IntelliJ, Windows Terminal, and Firefox. This does not account for any background processes that were running at the time. According to Task Manager on Windows, my CPU and Memory usage were sitting around 15% and 55% respectively while not running the benchmarks. Furthermore, I had 166 background processes running.

4. For each of your operational profiles, report the following: warmup time, minimum iteration time, maximum iteration time, average iteration time

My times were the following for each operational profile:

- Logging Data Structure (80/10/10)
    - Warmup: 171,984,700ns
    - Minimum: 86,308,800ns
    - Maximum: 92,291,800ns
    - Average: 88,205,050ns
- Read-heavy Database (10/10/80)
    - Warmup: 106,371,900ns
    - Minimum: 51,411,000ns
    - Maximum: 118,261,900ns
    - Average: 62,055,340ns
- Read-only Database (0/0/100)
    - Warmup: 20,785,300ns
    - Minimum: 7,751,100ns
    - Maximum: 9,920,100ns
    - Average: 8,067,760ns

5. Which of your operational profiles has higher *throughput* (i.e., performs more work per unit time)? What about the red-black tree might explain this outcome?

NOTE: Upon reviewing my hypothesis after getting the results from VisualVM, I realized that I had misread the number that the read-only operational profile was outputting. I was reading it from the terminal output without commas and accidentally added an additional factor of 1,000 for each number, meaning that I was reading 80 million instead of 8 million. I kept what I had

written before I realized this discrepancy, as it is accurate to how I perceived the number at the time of writing my hypothesis.

The operational profile that has the highest throughput is the logging data structure (mostly writes). Close behind that is the read-only database (only reads), and the least throughput by a substantial margin comes from the read-heavy database (mostly reads). The fact that the read-only database has a higher throughput than the read-heavy database by a substantial amount tells me that reading itself is an expensive operation and that one of the two other operations are playing a part in read-heavy being less time-consuming. Since mostly-writes also had a high throughput, I would also assume that insert is also a time-consuming operation. Furthermore, when the delete and insert operations were included (compared to absent in read-only), the throughput was at its lowest. Since delete was included in the operational profile with the least throughput, this leads me to believe that it is the least expensive operation. I will test this hypothesis using VisualVM and add the results below.

After running VisualVM, the results I got were:

- Logging Data Structure (80/10/10)
    - Insert
        - % Time Spent: 74.9
        - Invocations: 136,850
    - Delete
        - % Time Spent: 22.5
        - Invocations: 17,009
    - Lookup
        - % Time Spent: 0.5
        - Invocations: 16,947
- Read-heavy Database (10/10/80)
    - Insert
        - % Time Spent: 38
        - Invocations: 76,454
    - Delete
        - % Time Spent: 44.3
        - Invocations: 76,797
    - Lookup
        - % Time Spent: 11.9
        - Invocations: 612,458
- Read-only Database (0/0/100)
    - Lookup
        - Invocations: 4,014,621

Here are the operational profile percentages compared to the invocations and actual time spent using VisualVM:

- Logging Data Structure
    - Operational Profile: 80/10/10
    - Invocations: 80/10/10
    - Actual Time Spent: 75/23/1
- Read-heavy Database
    - Operational Profile: 10/10/80
    - Invocations: 10/10/80
    - Actual Time Spent: 38/45/12

Based on these results, the number of invocations for each operation roughly correlates to its weighted percentage in the operational profile. However, given these results, it seems that the delete operation spends much more time than its weighted percentage in each operational profile, which disproves my hypothesis. Furthermore, the lookup operation spends less time than its weighted percentage in each operational profile, also disproving my hypothesis. I did correctly hypothesize that the insert operation is expensive, given that the results show it takes up quite a lot of time even when its weighted percentage in the operational profile is low. It seems that when the weight is on inserts, it will spend roughly the same amount of time performing said operation as its operational profile percentage, whereas delete will spend much more time than lookup. Furthermore, when the weight is on lookup, it will spend significantly less time performing lookup than its operational profile percentage, whereas insert and delete will both take up more time than their operational profile percentages. This leads me to conclude that delete is the most expensive operation, followed by insert, and then a large gap with lookup being the least expensive operation.

6. Are your warmup times noticeably different from the "main" iteration times? Most likely yes, but either way, why might we *expect* a significant difference?

The warmup times are noticeably different from the “main” iteration times — they take longer. Specifically, the warmup takes roughly twice as long compared to the average time for the main runs for each operational profile. This is due to certain resources being “set up” while the program is being started. The program will create configuration files, things like the CPU, OS, and language will set up a cache for the program, and so on. Once these are created and re-used, the program can then run at a more efficient average speed.

7. When I run the test for either of my operational profiles with instrumentation-based profiling enabled, my measurements slow down by a factor of 10 or more. Why?

Instrumentation-based profiling will, in essence, record the start and end times plus the caller for every method call, which means it is very detailed. However, by having this level of detail, it will slow down the overall execution. In the current program, 1 million operations are done 10 times plus one warmup run for each operational profile. This means that 11 million operations are performed. If the start and end times are logged as well as the caller, this is three operations for each of the 11 million program operations. Furthermore, getting the specific start and end times of each call requires resources from the system to retrieve this information. Adding all of this on top of doing 11 million operations is bound to slow down the measurements by a significant factor.

8. Assume each run (each call to runOps) simulates the activity of one remote request. Based on the average execution time for each operational profile, what would the throughput be for each of your profiles? (i.e., requests/second)

If each call to runOps is meant to simulate the activity of one remote request, then the throughput for each operational profile based on the average execution time would be:

- Logging Data Structure (80/10/10): 11.34 requests/second
- Read-heavy Database (10/10/80): 16.11 requests/second
- Read-only Database (0/0/100): 123.95 requests/second

This was calculated by dividing 1 (request) by the small number of seconds it takes to perform that request to get how many requests can be done per 1 second.

9. Assuming each remote user makes 5 requests/minute, your program's resource usage scales linearly, and we are only interested in CPU execution time, how many concurrent remote users could you support on your machine (again, do this for each operational profile) without degrading performance or overloading the system?

To calculate how many concurrent remote users can be supported without degrading performance for each operational profile, the requests/second calculated in the last question needs to be converted into minutes. This can be done simply by multiplying each number by 60 to see what the throughput is in minutes instead of seconds:

- Logging Data Structure (80/10/10): 680.4 requests/minute
- Read-heavy Database (10/10/80): 966.6 requests/minute
- Read-only Database (0/0/100): 7,437 requests/minute

Since a user will make 5 requests a minute, and the throughput is now in units of minutes, the number of concurrent remote users can be calculated by dividing the throughput in minutes by 5:

- Logging Data Structure (80/10/10): 136.08 concurrent users
- Read-heavy Database (10/10/80): 193.32 concurrent users
- Read-only Database (0/0/100): 1,487.4 concurrent users

10. What aspects of load are we not testing, which could possibly reduce the capacity of your machine to service requests?

One aspect not calculated that could reduce the capacity of my machine to service requests is network connection. Every benchmark tested here has been isolated to my machine only — I ran all the tests locally. If users were to expect to connect into my machine and perform these operations, both their network connection (as a client) and my network connection (as a server) would need to be taken into account. Furthermore, I had executed these tests with only a set of essential applications running. This is not the common use case of my machine, meaning that the capacity could potentially be lower if I were using it as I usually do. Furthermore, only three specific operational profiles were tested, simulating a very narrow window of how the program can be used. I had determined that delete operations are quite expensive, however they only accounted for 10% or 0% of the total operations done by the operational profiles. If, let's say, one set of concurrent users' requests are very delete-heavy, the capacity of my machine and efficiency of the overall program would be much lower. This could also go hand-in-hand with stress testing, since a lot of the tests and calculations were done in regard to linear scaling of resource usage and taking max load into account rather than overload. Furthermore, each test was done for a brief window of time, meaning longevity testing was not taken into account. Lastly, a sense of capacity was not taken into account — it was determined how many concurrent users could potentially operate on my machine, but there is no guarantee that the calculated performance will maintain its average (it could dip after a certain number of concurrent users).