CS 1632 - DELIVERABLE 3: Performance Testing

Jordan Grogan & John Fahnestock

GitHub Usernames: jordangrogan & johnfcs778

https://github.com/jordangrogan/BillcoinVerifier

Summary

What was most challenging about this deliverable?

- One challenging aspect of the deliverable was coming up with a good object-oriented structure to hold all of the data in the blockchain efficiently. Once set up, populating the structures was also difficult specifically in the case of breaking apart the strings for each line and writing correct regular expressions to match and populate the objects.
- Efficient reading of the file/populating of the data structures was also a key part that required some clear planning.

What kind of edge cases and failure modes did you consider?

- Tests were written considering certain failure modes such as checking the regular expressions stripping transactions correctly and the addresses weren't greater than 6 alphabetic characters.
- The lines being in the correct order were verified through unit tests of in-order and out-of-order mocked blocks.
- The hash of the current line or the hash of the previous line not being correct was also verified.

Using the flame graph, what methods were taking up the most CPU time?

- Based on the flamegraph, the `hash` function was taking up the most CPU time. This makes sense since the function was doing taxing calculations at byte granularity.
- Checking user existence was also taking up more CPU time than it should have at first when we used an array for the user list instead of a hashmap.

Did you make any changes based on the flame graph or timing?

- The hash function was not changed at all as we deemed it made sense for it to be the most taxing on the CPU, and we determined there was not a good/feasible way to improve its performance as it was performing a specific set of calculations for the hash.
- The `user_exist?` function within the `user_list` object was modified to use a hashmap instead of an array to provide a constant performance for checking user existence. The hashmap `has_key` method easily allowed constant time access to a user's data. The time for this function was linear prior to this improvement.

Real Times for `long.txt`

Run 1: 0m39.004s

Run 2: 0m39.759s

Run 3: 0m39.152s

Mean: 0m39.305s

Median: 0m39.152s

Flamegraph Screenshot (sample.txt)