CS 1632- DELIVERABLE 4: Performance Testing

Simran Gidwani

Alex Kim

https://github.com/adk67/D4

The most challenging part about this deliverable was ensuring that all the edge cases were met. At first we had thought to use multiple classes to make debugging and testing easier but eventually ended up sticking with using only two other classes. The two main failure cases we considered were whether the file existed and if the file was empty. Regarding the validity of transactions, we tested for invalid characters in the sequence of transactions, tested for valid to and from addresses, and also tested whether the timestamp of the second transaction was after the timestamp of the previous transaction. At first, after splitting the timestamp by the decimal, we were only testing whether the first half of the split number was larger or smaller than the previous. After testing and seeing that we were getting some incorrect output, we realized that we were forgetting to test the timestamp by the nanoseconds as well. Another thing that took some time was reaching the 90% coverage with minitest.

Although we knew it would not affect our run time by much, the first thing we decided to do to cut our run time down before running flamegraph, was making a slight change in the print\_outcome method. While first running the code we had not been sorting the results of the hash when printing the keys and values. After realizing that this might help optimize, we were able to change the method to sort the array by the key and then print the values. Even though this was a small change it decreased our runtime for long.txt from 58 seconds to 55 seconds.

After finalizing the rest of our code, we ran flamegraph using sample.txt to see which method was taking up the most time. As you can see below, it was the compare\_hashes method that is used to determine whether the output hash that is generated is correct. More specifically, the call to hash\_block is what took up the most time.

After running flamegraph and looking at the code to see what we could do, we were able to determine that in order to optimize this section of our code we would have to approach storing the calculations of the hashes in a different way. After some time and research we decided that instead of hashing each time and storing the value into a variable, we should read each character and store it into an array and then do the calculations. Just by making this optimization we were able to cut our runtime down from around 55 seconds for long.txt to 17 seconds.

Initial flamegraph:



After optimization flamegraph:

