Date	February 17, 2014
Team ID	cd-037
Team Members	Aryamman Jain (999554076) Pranav Mehndiratta (999480725) Vaibhav Vijay (1000073029)

Performance Evaluation Report

To evaluate our storage server's performance under different configurations (logging to a file, logging to stdout and logging disabled) we used the "Census" data to populate a single table in the database. Specifically, we measured the performance of get and set functions under varying workload. The table contained city names as keys and population data as values stored in a hash table. The hash function implemented, generates a unique index corresponding to the characters in the hash value. To handle collisions, we are using Linear Probing.

Methodology

The process used to measure both metrics, end-to-end processing time and server-side processing time, was similar as we primarily focused on testing our data structure implementation rather than the client-server communication. This allowed us to optimize our implementation of the storage server by immediately using our measured data. Since hash tables are generally O(1) for inserting, deleting and searching, we were testing for corner cases, i.e. to generate as many collisions as possible.

End-to-End Execution Time and Server Processing Time

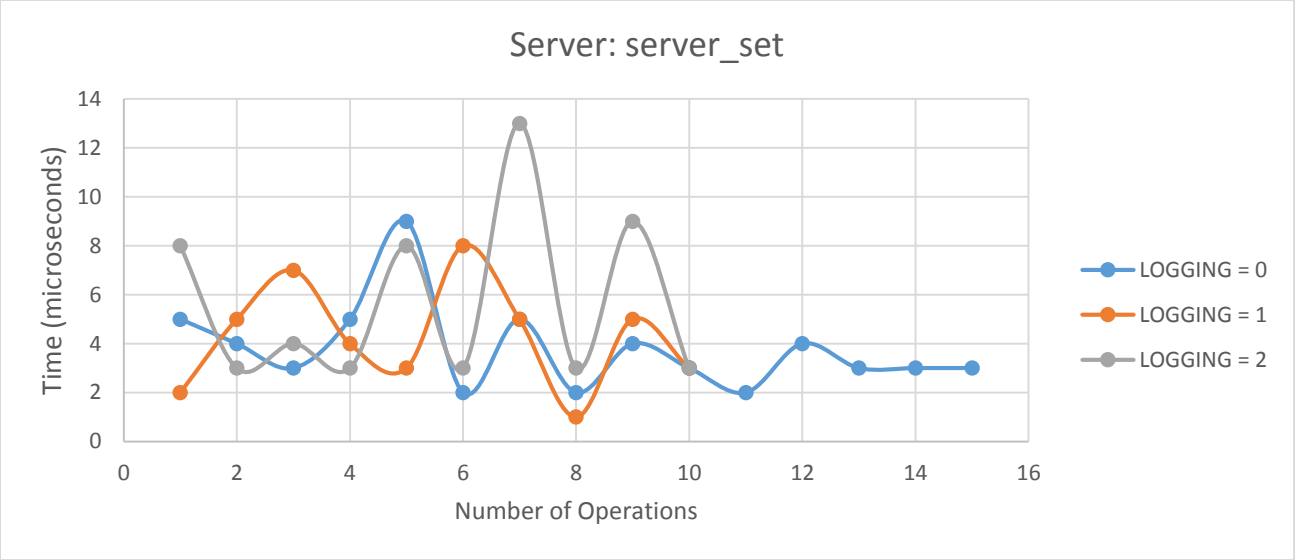
To maximize the number of collisions, we entered the entire "Census" population data into the storage server. Following the insertion, we measure the consequent performance of the database on accessing and modifying this table. There were a cumulative 692 entries in the census table and we performed approximately 20 sets and gets on this data. The sets and gets were chosen such that to generate a collision case in each operation.

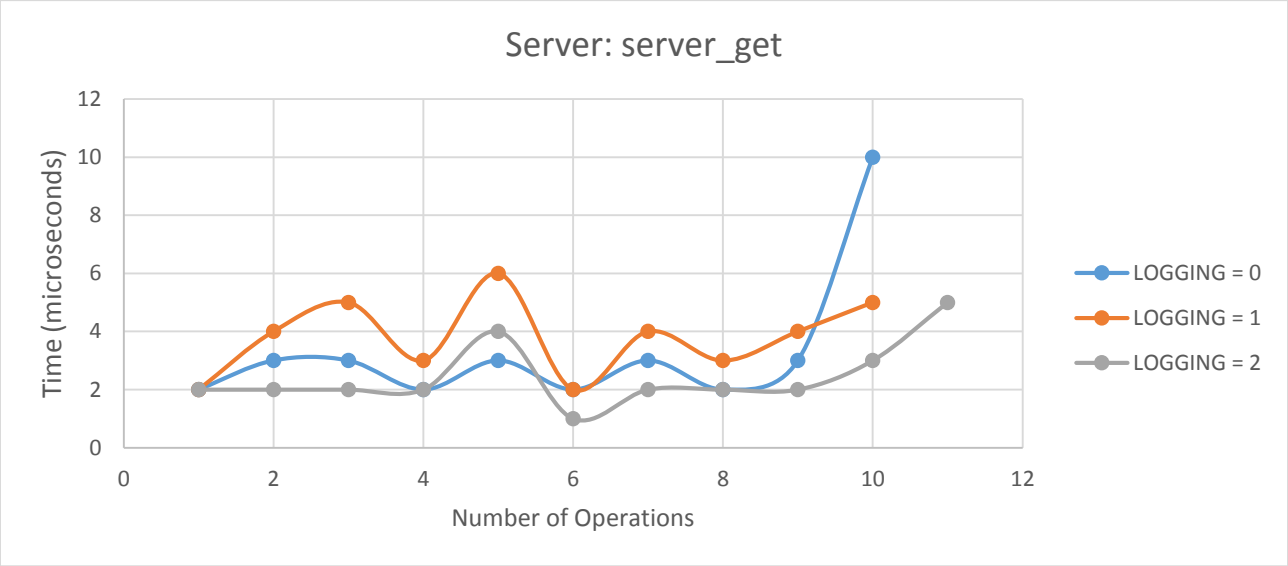
Results

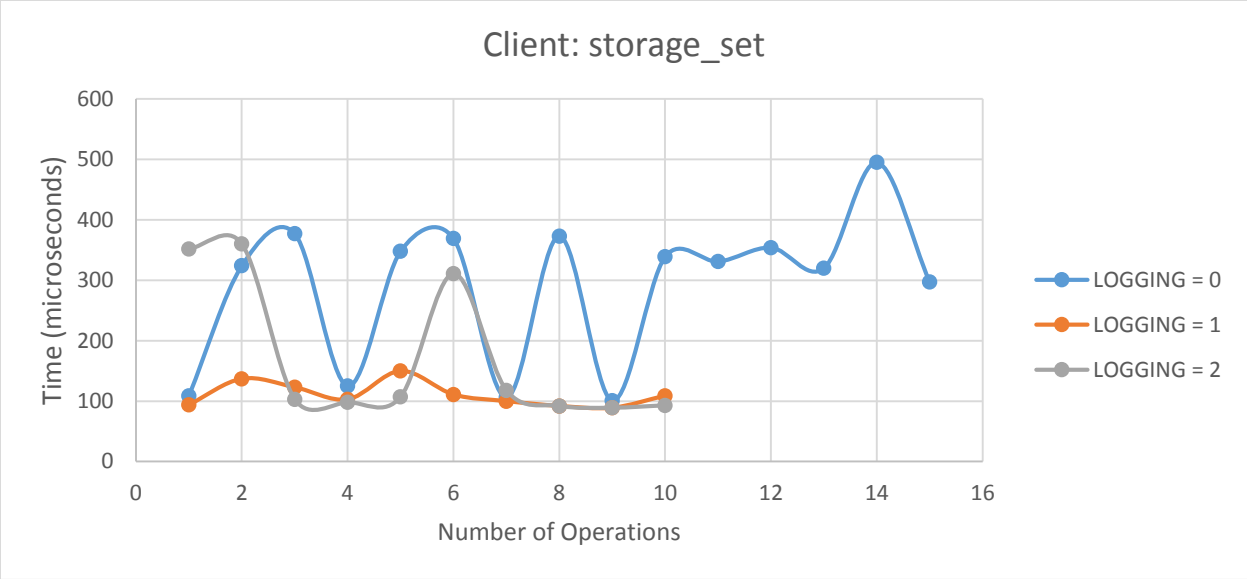
The measure metrics are shown clearly in the following figures. Generally the end-to-end execution time is 100-300 microseconds whereas the server-side processing time is 0-20 microseconds (figures later).

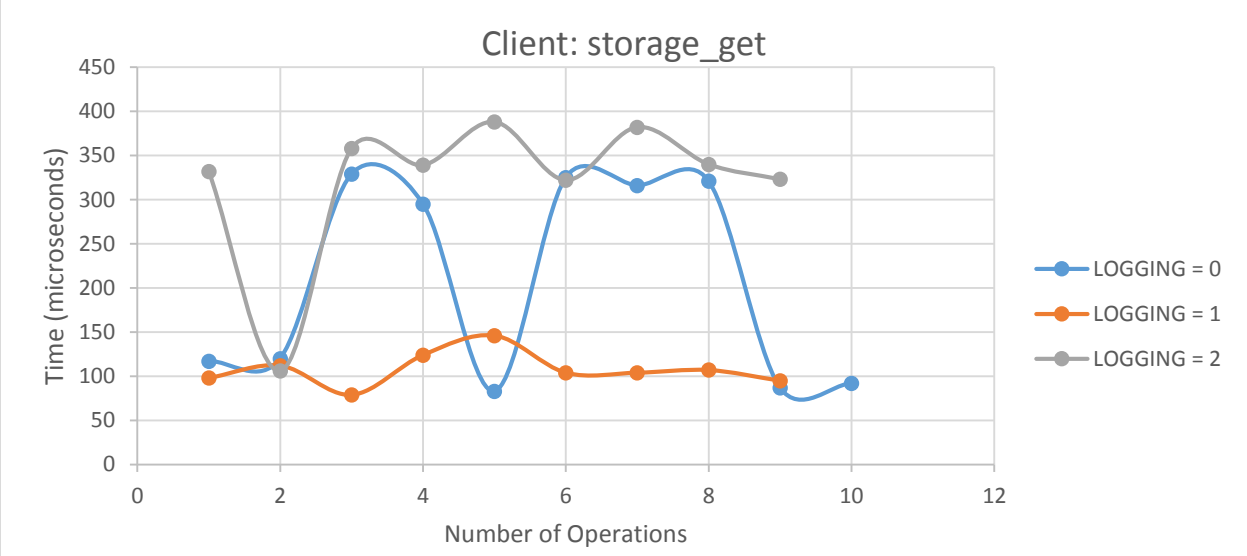
Conclusions

The evaluation clearly identified the areas where performance was lacking in the storage server. They clearly show that approximately 90% of the time required for each set and get is spent on the communication between the client and the server. The server side responses, i.e. the hash table processing is negligible compared to the time require to parse and send requests/responses between the client and server. This means that we must focus on improving the communication methods, particularly the implementations of recvline() and sendall() in the utills.c file.


Server: server_set
Time (microseconds)
Number of Operations
LOGGING = 0
LOGGING = 1
LOGGING = 2


Server: server_get
Time (microseconds)
Number of Operations
LOGGING = 0
LOGGING = 1
LOGGING = 2


Client: storage_set
Time (microseconds)
600
500
400
300
200
100
0
0
2
4
6
8
10
12
14
16
Number of Operations
LOGGING = 0
LOGGING = 1
LOGGING = 2


Client: storage_get
Time (microseconds)
450
400
350
300
250
200
150
100
50
0
0
2
4
6
8
10
12
Number of Operations
LOGGING = 0
LOGGING = 1
LOGGING = 2