|  |  |
| --- | --- |
| **Date** | April 6, 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 (concurrency set to 1), we inputted the data manually to populate a single table in the database. Specifically, we measured the performance of get and set functions under varying workload. The table contained random keys and integers (with column name) 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 Transaction Abort Rate**

To maximize the number of collisions, we inputted a lot of 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 100 entries in the census table and we performed approximately 20 sets and gets each on this data. These requests were being sent to the server by 10 different clients simultaneously. The sets and gets were chosen such that to generate a collision case in each operation and also to cause some aborts during the get-set transactions. Figure 4 below shows the increase in the transaction abort rate (ratio of number of transaction aborts to total number of transactions) as the number of clients increases. The total number of transactions attempted was around 40.

Results

The measure metrics are shown clearly in the following figures. Generally the average end-to-end execution time (time/number of client requests) is 100-600 microseconds with up to 10 clients running simultaneously (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. There is also a slight delay due to the creation of threads and also the time required to destroy them. We must focus on improving the communication methods, particularly the implementations of recvline() and sendall() in the utills.c file.

Figure : End-to-end processing time with concurrency set to 1

Figure : End-to-end processing time of the Get command

Figure : End-to-end processing time of the Set command

Figure : Occurrences of transaction aborts