Project 3 Experimental Report

Lab: Indexed vs Non-indexed Database Tables

CS525

12/2/2010

Team Members:
Patrick O'Shea
poshea@iit.edu
Role: Data generation
Hao Gao
hgao8@iit.edu
Role: Indexed range query lookups
Xuesi Jian
xjian1@iit.edu
Role: Experiment performer/Data reporter

INTRODUCTION

It is well known that indices can have both positive and negative effects on the performance of a database management system. In the following experiment we intend to learn what the benefits and disadvantages of an index are, in regard to the different operations of a database and the varying percentages of a given table returned by a query. In order to accomplish this goal, we set out to design a series of experiments that will further our understanding of the impact that indices have on the performance of our database for various types of operations.

APPROACH

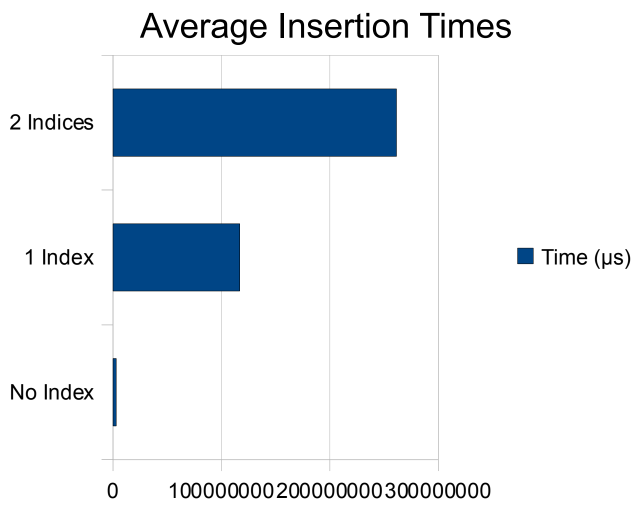
To begin, we formed hypotheses for each of the experiments we would be performing. In the case of insertions, our familiarity with our index implementation led us to believe that insertion performance would be much worse for an indexed table than for a non-indexed table, and this informed our experimental setup. In the case of point and range queries, with the intuition we learned in class that indices should only be used when 10% or less of a table is returned by a query, we assumed that performance would get exponentially worse as the percentage of the table returned increased. With this in mind, we decided we needed to create a number of tables that would return various increments of 5% (between 5% and 50%) for each type of query. In order to support this experiment we would need to generate a large and varying set of tables to perform the experiments on. For this we created a small Python script that, given certain arguments such as: tablename, number of columns in the table, number of rows required for the table, attribute to set percentage of, operation (=, <, <=, >, >=), a value to base the percentage of rows from, and the percentage desired, etc. would generate tables with pseudo-randomly generated values that would would contain rows with the specified column's values that satisfy the percentage criteria specified (with uniform distribution). We would then use these tables in a series of experiments in which we would run the appropriate queries on the appropriate tables several times to get a fair average of performance with and without indices on the tables. Using the data culled from these experiments we could then analyze the results to see if they match with our hypotheses.

EXPERIMENTAL SETUP

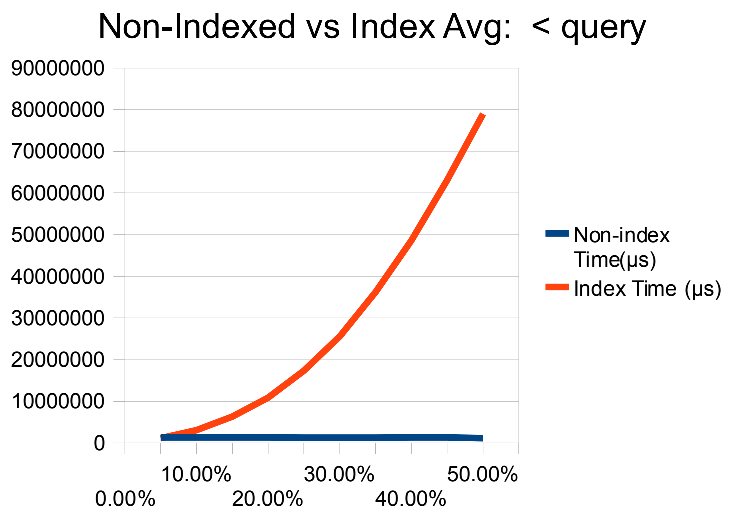
For our experiment we utilized a ThinkPad laptop with an Intel Core i3 M330 CPU at 2.13GHz processor, 2GB of RAM. The system was running Ubuntu Linux version 10.10. From our experience with Project 2, we knew that our implementation of the index was quite slow when doing insertions, so we decided our tables would not need to be very large to get a noticeable difference in performance. For this reason we created datasets to insert with only thousands of rows. We then performed several insertions of these datasets into database tables. One set was run without any index on the table, the second set was run with one index on the table and the third was run with two indices on the table. For the point and range queries we generated tables with hundreds of thousands of rows that satisfied the percentage requirements of 5% increments between 5% and 50% of the table for each type of query we would run (=, <, <=, >, >=). We then would use these tables in a series of experiments using the appropriate queries and tables several times to get a fair average of performance with and without indices. All of the related Python scripts, shell scripts, generated data files and experiment definitions can be found in the /datagen folder of our project. The pre-built tables used for point and range tests can be found in the /base folder of our project.

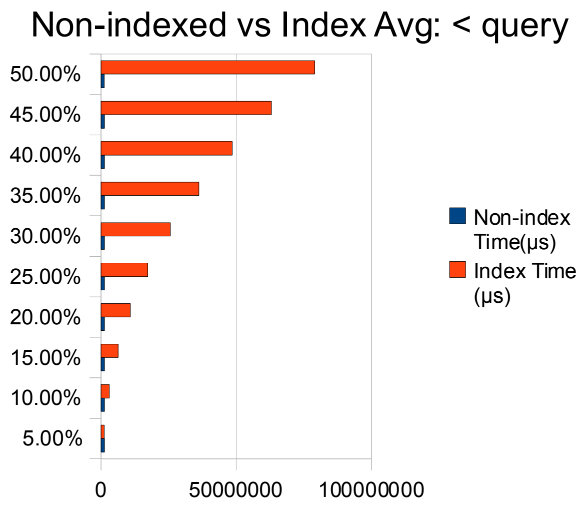
EXPERIMENTAL RESULTS

To begin with presenting our results, we shall start with the insertion performances.



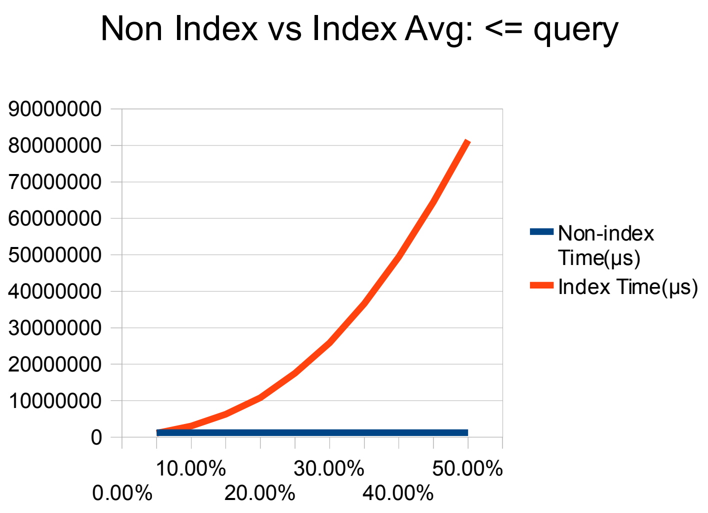
These results seem to be in line with our hypothesis. Moving on, we present the results of the < range queries.

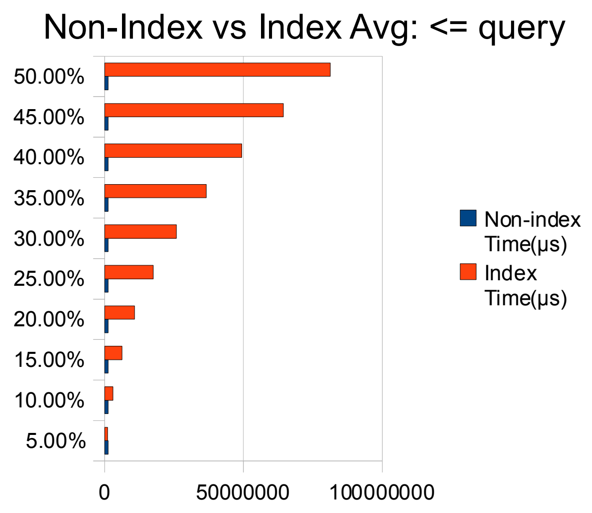




Again, this seems to fall in line with our initial assumption that the index times would increase exponentially.

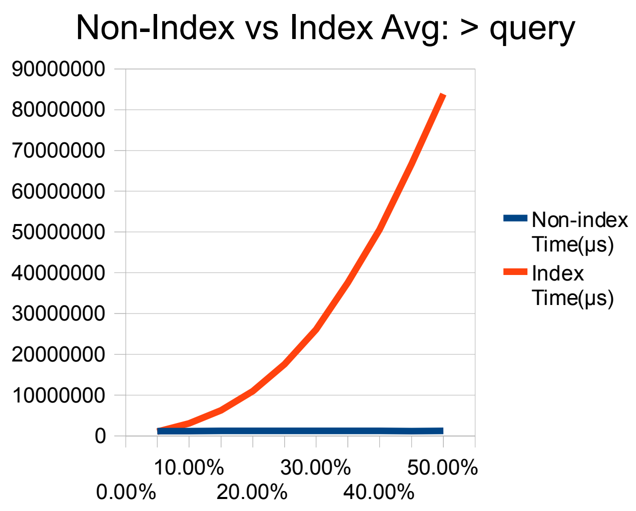
Next, we present the results of the <= range queries.

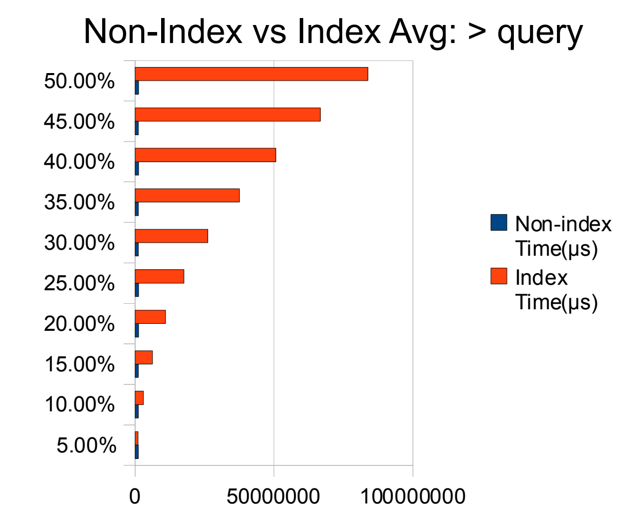




The exponential increase, once more, seems to fit with our initial hypothesis.

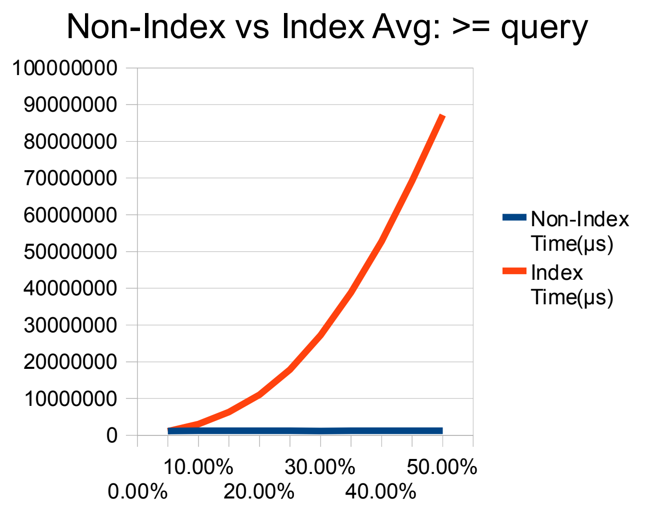
Here, we present the results of the > range queries.

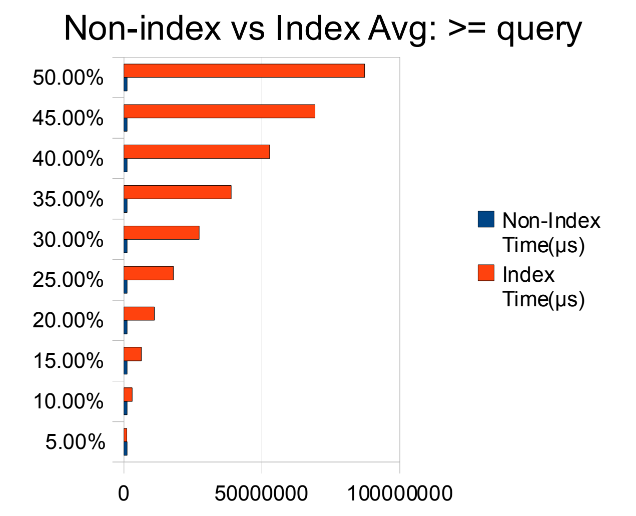




Once more, note the exponential increase in the index times as the number of rows increases.

Finally, we present our findings regarding the >= range queries.





Finally, these results also seem to support the exponential nature of our hypothesis. Now, we shall move on to discuss the results of these experiments.

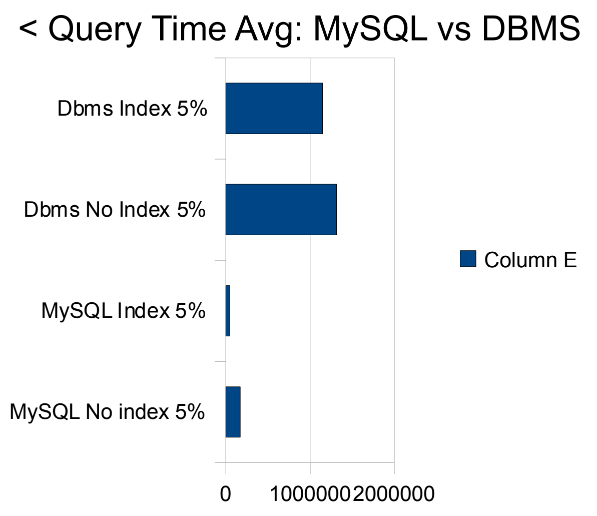
DISCUSSION

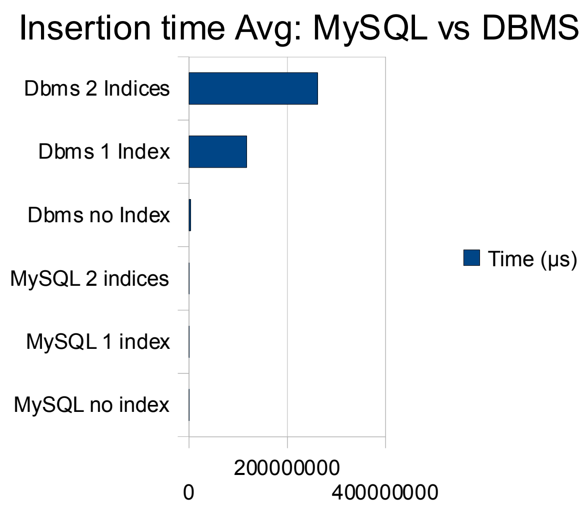
In many ways these results are as we expected from our hypotheses, but in other ways they differ from our expectations. In regard to the insertion times into tables both indexed and non-indexed, we were sure that we would see a marked increase in the time it takes to insert records. Our results show this to be just the case. When looking over the results of the point and range queries, it is clear that our assumption of exponential increase when using indices is, in fact, also the case. Unexpectedly, however, the “sweet spot” at which indexed queries outperform non-indexed queries was not where we would have assumed it to be. From looking at these results it is clear that the only percentage at which our index outperforms (or as in our case, nearly equally performs) the unindexed queries is when the percentage of rows returned is 5%. Clearly our implementation of the B+ tree index leaves something to be desired in the way of performance. As a further investigation, we decided to perform additional experiments with the MySQL database system, as part of the extra-credit assignment, to see how our implementation stacks up against a commercially viable database management system.

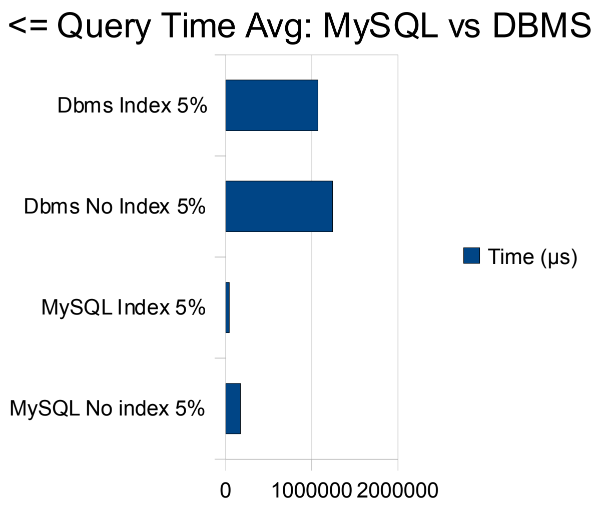
(All experimental results, both raw and aggregate, are located in the folder /datagen/results of the project folder.)

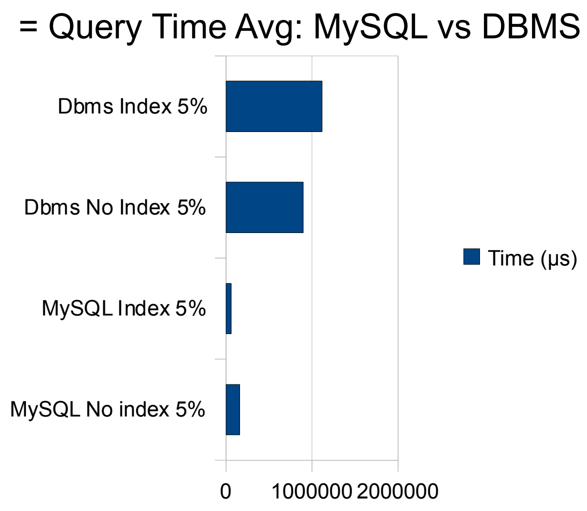
EXTRA CREDIT – MySQL COMPARISON

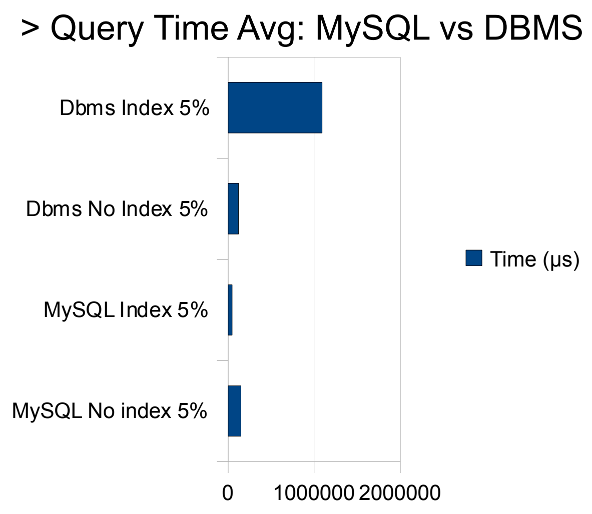
In order to compare the results from our database management system with that of MySQL's, we intended to run all of our experiments identically on both systems. What became obvious as we were conducting the experiments was that MySQL only uses its indices for queries that returned 5% or less of the rows in a given table. Given this discrepancy between our two systems (which makes sense considering a commercially viable DMBS should not use indices in this case) we decided to only compare our results for point and range queries that return 5% of the tables. Our insertion experiment, however, was conducted in the same fashion. The following were our findings:

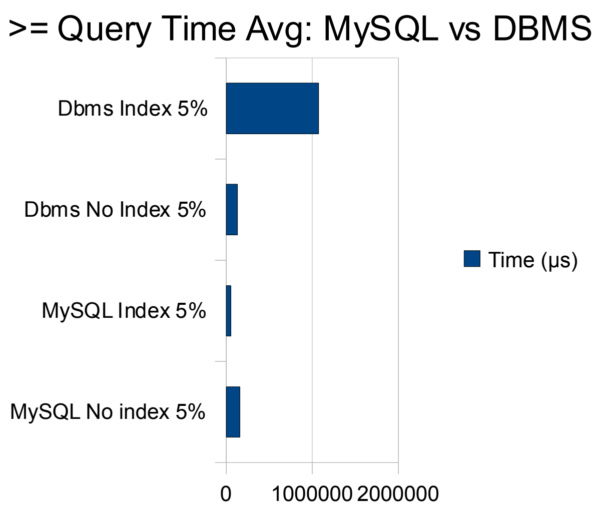












Clearly the MySQL database management system bests ours on every front. We expected as much from a system that has been in development since 1995 and has also been widely adopted in enterprise situations the world over. Though we would have liked our performance to be somewhat closer to that of another database management system, we believe that the amount of knowledge that we have gained over the course of these three projects far outweighs the shortcomings themselves.