Analyzing Enron Data: Bitmap Indexing Outperforms MySQL Queries by Several Orders of Magnitude

Kurt Stockinger, Doron Rotem, Arie Shoshani, Kesheng Wu

Computational Research Division
Lawrence Berkeley National Laboratory
University of California
1 Cyclotron Road, Berkeley, CA 94720, USA

Abstract

FastBit [1] is an efficient, compressed bitmap indexing technology that was developed in our group. In this report we evaluate the performance of MySQL and FastBit for analyzing the email traffic of the Enron dataset. The first finding shows that materializing the join results of several tables significantly improves the query performance. The second finding shows that FastBit outperforms MySQL by several orders of magnitude.

1 Introduction

The Enron dataset was used by various researchers in the area of social network analysis to discover patterns in the data. These patterns are usually visualized by complex graph algorithms. However, due to the large amount of social network data, the pattern finding and visualization algorithms often take a long time to terminate. In order to reduce the time complexity of these algorithms, it is often important to pre-filter the results based on multi-dimensional criteria such as "Retrieve all emails that were sent by person P at time T". In this report we will show that multi-dimensional bitmap indices significantly improve the performance of these types of queries.

For our performance evaluation we used the Enron dataset that was prepared by Shetty and Adibi [2]. All the data is stored in MySQL containing the following four tables: *employeeList*, *message*, *recipientInfo*, *referenceInfo*. In total, the dataset contains some 250,000 message from 151 Enron employees that were recorded over a few years. For further details about the dataset we refer the reader to [2].

In this report we compare the performance of MySQL with FastBit [1], an efficient, compressed bitmap indexing technology that was developed in our group.

2 Performance Results - Original Dataset

In our first set of experiments we measured the performance of searching for specific *senders* and *receivers* of the emails. We built an index for each of these two attributes. Since both senders and receivers are in different database tables, this kind of search requires an expensive join operation. The next step was to materialize the join and store the results in an additional table that we call *materialized table*. The newly created table contains some 2 million records. Remember, the number of original messages was 250,000 which indicates that, on average, each message contains 8 recipients. We also built indices for *sender* and *receiver* on the materialized table.

In order to build bitmap indices for the materialized table, we needed to export the date into binary files. In particular, we stored each attribute in a separate file and then built a bitmap index for the attributes *sender* and *receiver*.

Next we measured the performance of queries of the form "Retrieve the recipients of all emails that were sent by person *P*". For these experiments we randomly selected 100 persons from the table *employeeList* and executed a query for each person. In total, we ran 100 queries and measured the retrieval time including the time to extract the result after the search.

Figure 1 shows the performance of three different access plans, namely *MySQL - Join*, *MySQL - Materialized* and *FastBit*. We can see that the query that is based on joining two tables takes the most time. We can also see that the response time is independent of the number of hits. *FastBit* shows the best query response time and is a factor of 10 to 100 faster than *MySQL - Materialized*.

Next we measured the performance of queries of the form "Retrieve all senders of emails that were received by person *P*". Similar to the previous experiments, we randomly selected 100 persons. Figure 2 shows that this time the difference between *MySQL - Join* and *MySQL - Mate-*
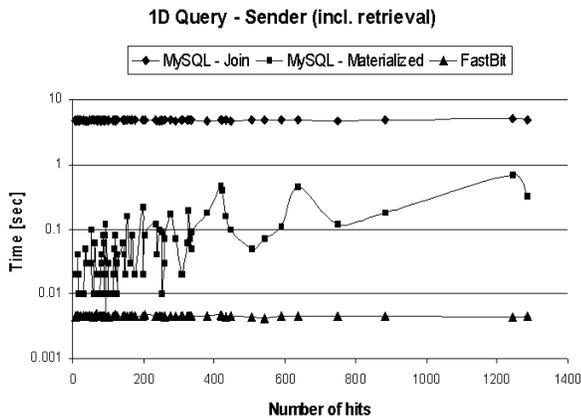
1D Query - Sender (incl. retrieval)

Figure 1. Performance of query: "Retrieve the recipients of all emails that were sent by person P".

rialized is much smaller. The reason is that the number of hits is much smaller than in the previous experiments and thus the join operation is less expensive. However, *FastBit* is again up to a factor of 100 faster than *MySQL - Materialized*.
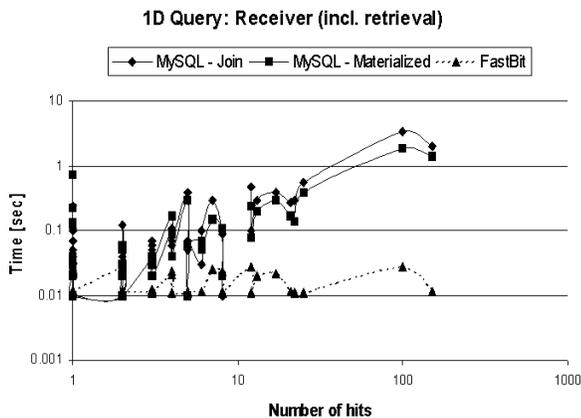
1D Query: Receiver (incl. retrieval)

Figure 2. Performance of query: "Retrieve the senders of all emails that were received by person P".

Due to the better performance of the access plan *MySQL - Materialized*, for the remaining experiments we only use this access plan and compare it with *FastBit*.

Our next experiments evaluated the performance of the following queries: a) "Count the number of emails that were sent every day before time T. b) "Count the number of emails that were sent before date D". For performance reasons we split the attribute *date* of the original table *message* into the basic components of *date* and *time* and built indices.In order to emphasize on the index operations, the following experiments only count the number of emails the fulfill a certain search criterion.

The performance of these queries is shown in Figures 3 and 4. Again we see that *FastBit* shows better performance characterstics than MySQL. In particular, we observe that the performance of *MySQL - Materialized* depends on the number of hits whereas the performance of *FastBit* is about constant.
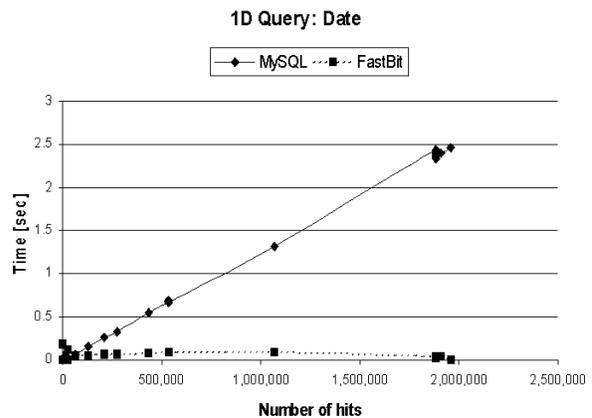
1D Query: Date

Figure 3. "Count the number of emails that were sent every day before time T.

3   Performance Results - Duplicated Dataset

In the next experiments we measured the query performance of a larger dataset. We thus duplicated the Enron dataset 10 times. The resulting materialized table contains some 20 million records.

Figures 5 through 7 show the performance of queries with one specific search criterion. Similar to the previous experiments, FastBit is up to a factor of 100 faster than MySQL.

In our last set of experiments we measured the performance of queries with multiple search criteria (*multi-dimensional queries*). A typical query of this kind is "Count the number emails that were sent by person P in the time interval T before date D". The results of two and three dimensional queries are shown in Figures 8 and 9. We notice that as the number of query dimensions increases, the relative performance improvement of FastBit over MySQL
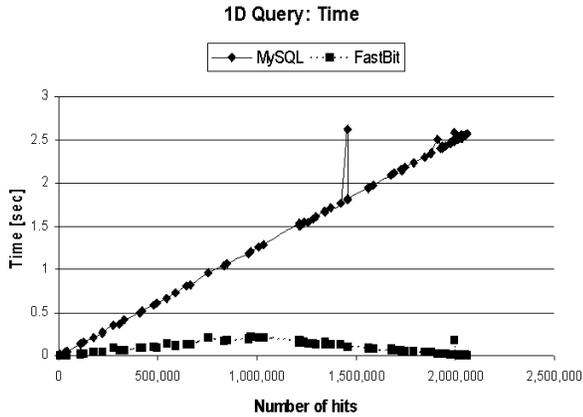
1D Query: Time

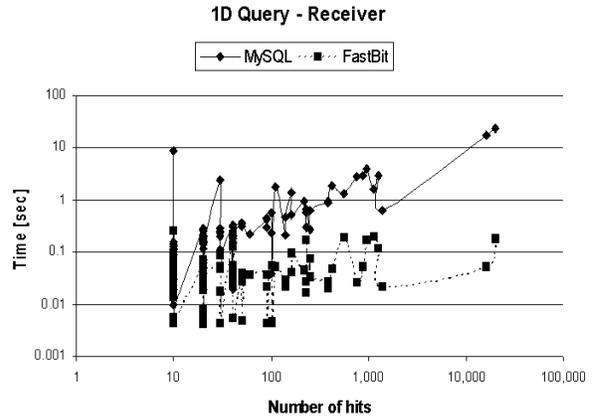**Figure 4. "Count the number of emails that were sent before date D".**

1D Query - Receiver

Figure 6. "Count the number of emails that were received by person P".

1D Query: Sender
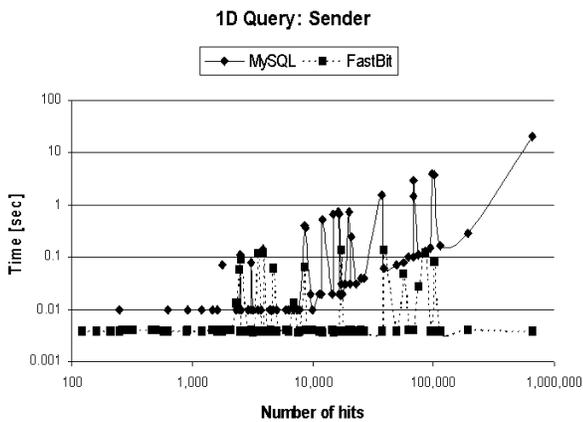
Figure 5. "Count the number of emails that were sent by person P".
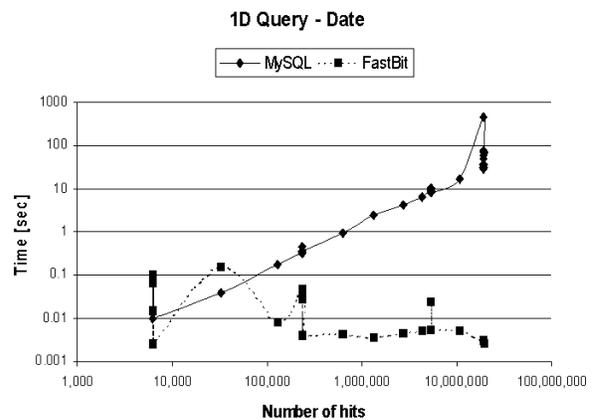
1D Query - Date

Figure 7. "Count the number of emails that were sent before date D".

increases even more. For these types of queries, FastBit is even up to a factor of 1000 faster than MySQL.

4   Conclusions

In this report we evaluated the performance of MySQL and FastBit for queries on the Enron dataset. Our first finding shows that queries on materialized tables provide a significant performance improvement since expensive join operations are avoided. We also demonstrated that Fast-Bit outperforms MySQL up to a factor of 1000 for multi-dimensional queries.

In the future we will work on *neighborhood queries* that are of particular importance for analyzing message flows/chains within groups. Typical queries are "Find all the emails that person A sent to person B". Next, find all emails that person B received from A and sent to person C". By analyzing this kind of messages one can discover indirect relationships between person A and person C. Moreover, the message frequency and the message date might also reveal some important characteristics. In order to quickly search through this information, the use of efficient, multi-dimensional indexing as described in this report is very important.
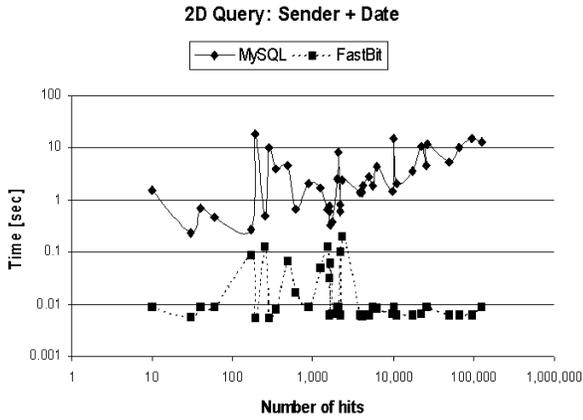
http://www.isi.edu/~adibi/Enron/Enron_
Dataset_Report.pdf, Jan. 2006

2D Query: Sender + Date

Figure 8. "Count the number of emails that were sent by person P before data D".

3D Query: Sender + Date + Time
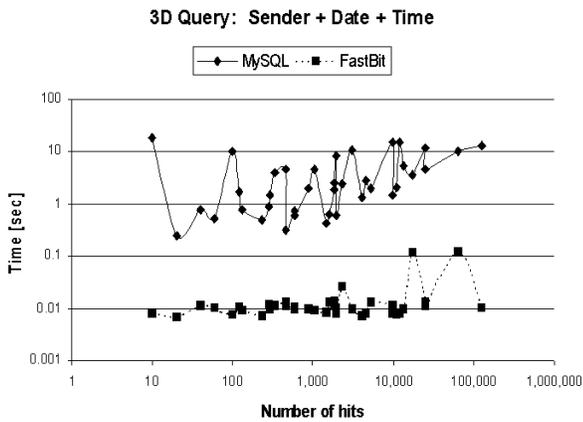
Figure 9. "Count the number of emails that were sent by person P before data D and time T".

Acknowledgment

References

[1] FastBit, http://sdm.lbl.gov/fastbit. Jan. 2006.

[2] J. Shetty, J. Adibi, The Enron Email Dataset, Database Schema and Brief Statistical Report, Retrieved from