JESSICA RIEGER OCTOBER 16TH, 2016

Bigtable: A Distributed Storage System for Structured Data

By: Fay Chang, Jeffery Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber

Google, Inc: OSDI '06

MAIN IDEA OF PAPER 1

- The main ideas of the paper I chose is that Bigtable is a good way to manage large amounts of structured data through its distributed storage system.
- Gives clients dynamic control over the data layout and format. Clients can decide, using the simple model, about the locality of data, such as the schema of their data and whether the data is in memory or on the disk.
- Wants to serve both throughput-oriented batch-processing jobs and latency sensitive jobs serving data to users as well as jobs that require huge amounts of data storage.
- Bigtable says that it strives to achieve wide applicability, high-performance, scalability, and high availability.
- Bigtable is a model that can be represented as a Tree structure that has 3 levels. A root tablet, other metadata tablets, and user tablets.

HOW ITS IMPLEMENTED

- Bigtable abandons the traditional use of the relational model and stores its data in a way that can be represented by as a “sparse, distributed, persistent, multi-dimensional map.”
- The Bigtable solution is a flexible, high-performance solution that is used by Google products as well as other companies. This solution is beginning to be sold as a service so that customers do not have to manage their own clusters of servers.
- Bigtable uses Google File System, MapReduce, and Google SSTable to supplement its operations as well as a lock service called Chubby.
- The data is also grouped by column name into column families that have related data and can be compressed to reduce space usage and can be grouped into locality groups readily available to the user and can reduce accesses to the disk. Access control and disk or memory accounting are done at this level.
- The data is spread across multiple tablet servers that are all connected to a master server and a library. Tablets can be added or deleted dynamically as needed.
- The structure of this model is a map that is indexed by row key, column key, and timestamp. Each row range for a table is called a tablet and is stored on its own server. By choosing the correct range, a client can keep relevant information easy to access.
- The user can use the API to delete tables or column values or manage the structure or access control rights. The user can also access data based on column values or row values.
- To improve its performance, uses 2 level compression, multiple types of caching, bloom filters(to as a client what data present before accessing disk), and a single log file of commits.

MY OPINION

- If chubby becomes unavailable then Bigtable becomes unavailable.
- Though the API may seem simple to the developers, it seems much more complex than the relational model that is so popular today.
- Why are we going back to showing the user where things are stored when the whole point of the relational model was abstraction into a simple table.
- Does seem like a flexible system that trades time for space so that is usually the way we want to go. Saving time by having the library prefetch other tablet locations when it goes to get one.
- Has a recovery system using memtable so that it can redo all changes recently made to recover the lost tablet. Seems fairly reliable because indexes are stored in so many places.
- New system though so even the developers admit they are having to deal with unexpected problems.
- Good because can be scaled as the company develops over time. (grows with the enterprise that it serves.)

THE COMPARISON PAPER'S MAIN IDEAS

- The main idea of the comparison paper was to test the performance of DBMS-X, Vertica, and MapReduce (Hadoop.)
- In the end, they found that overall, the DBMS were harder to install and configure initially however once they were up and running they were easier to manipulate.
- MapReduce program is inefficient due to lack of structure, has slow start-up time that hinders performance, is not easy to use among developers dues to the lack of database level rules, and uses a lot of CPU processing/energy unnecessarily.
- In terms of the user interface, Hadoop was easier to install and configure, but the DBMS were easier to interact with due to the several tools that go with them and the ability to query the system catalog.
- The DBMS (DBMS-X and Vertica) outperformed Hadoop in almost every test due to its short/non-existent start-up time, the fact that it doesn't need to combine results into one file, its use of compression effectively, and its execution strategies that reduce the amount of network traffic.
- DBMS was bad because must completely restart after failure, and the size of buffer pool and sort heaps were too small for current systems.

COMPARISON PAPER’S IMPLEMENTATION

- The implementation of the comparison of the three systems was executed through a series of trials. For each system they performed a certain test three times and took the average performance time. As well, they observed how the performance was altered as the test grew in size (namely through the number of nodes involved and the amount of data.)
- By playing with the configuration of each system and adjusting the capabilities, they tried to make each system perform its best. Whether it be writing code for Hadoop or adjusting the distribution of the processing, they did everything they could to improve the results.
- The tests that they ran were a join task, aggregation task, selection task, and the UDF aggregation task. MapReduce was slow due to the amount of data it must transfer, the map and reduce functions it has to execute, and its limitation to the speed it can read off the disk.
- The DBMS outperformed the MapReduce system in all except the UDF aggregation task. Same performance for the UDF aggregation task.

COMPARISON OF THE TWO PAPERS

- In my opinion, I think that the Database management system is still the way to go for a database model. The reason for this is because it has many more years of foundation and though it presented its own problems, it out performed the Map Reduce model Significantly.
- Perhaps once the Map Reduce model gains some more tools and a simpler way to interact with it, then I will consider it more thoroughly as a model to use.
- One thing that definitely needs to be addresses regarding the Database Management System is its lack of an adequate failure system. To start over after failure seems completely ridiculous.
- Overall, this experiment was useful to explore the faults in both systems and push both to make the necessary user interface and model changes.

COMPARISON OF THE TWO PAPERS

- Both stress the importance of a system's performance and the trade-offs between time and space.
- Emphasis is placed on the structure of the systems and the importance of this structure to make accessing the data easy and quick.
- One paper supports the use of MapReduce, while the other points out many of its flaws. Both acknowledge that MapReduce is far from a perfect system.
- In each of the papers, the user interface and ease of use is addressed. In the first, the user interface seems more complex than the Database Management System interface that the second promotes.
- Both papers present their ideas on how to address the ever increasing amounts of data that our world sees. Each approaches it from a clustered distributed server approach and uses compression.
- The first paper uses many different programs to manage the data whereas the DBMS uses only a single program with a user interface.

MAIN IDEAS OF STONEBRAKER TALK

- The Relational Database Management System can not be the "one size that fits all," instead it is in his opinion the "one size fits none." Essentially it is useless.
- RDBMS are too heavy-weight, too slow. SQL is too slow and incapable of doing complex analytics such as graph analytics.
- The popularity of the column store, the speed of the column stores, the NoSQL market (potpouri of different models very different from RDBMS), the battle for more complex data management and statistic programs combined are all forcing RDBMS into oblivion.
- Warehouse storage, Transaction processing, analytics, streaming, and graph analytics markets are all using column store and other technologies so row-store is becoming obsolete.
- Networking in DBMS are the bottleneck which conflicts with the needs of our world and its markets today. Other systems deal with networking much better so they are gaining popularity.

ADVANTAGES AND DISADVANTAGES OF CHOSEN PAPER

- The advantages of the Bigtable system in terms of the stonebreaker talk are that it is a model that is abandoning the traditional relational model.
- It implements a column-based storage and is abandoning much of the relational model's abstraction in favor of allowing the user to customize the locality and layout of their data.
- The model supports complex modifications by the user that are clearly being demanded by markets described in the talk.
- Disadvantages are that the Bigtable system still has row storage to some degree so that may hinder its ability to keep up with the complex and fast systems being developed today.
- Another disadvantage is that this model is built on several other Google Infrastructure. This means that if one of the many other products in the competitive market edges out Google, then their system could potentially not work.