The Google File System, A Comparison of Approaches to Large-Scale Data Analysis, One Size Fits All – An Idea That Has Come And Gone

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, Google
October 29-22, 2003, Bolton Landing, New York, USA.

Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Michael Stonebraker
June29-July2, 2009, Providence, Rhode Island, USA.

Michael Stonebraker
Ugur Cetintimel

NICHOLAS MARENGO

CMPT 308

ALAN LABOUSEUR

The Google File System

The Google File System, or GFS for short, was created to cater to the needs of Google's large data processing.

Their main focuses in creating the GFS were performance, scalability, reliability, and availability.

Google recognized problems in other earlier file systems, and strived to make theirs perfect for their data processing. It is even stated that some design decisions were specific to their unique setting.

The GFS was built to accommodate and support large scale data processing and commodity hardware. The GFS system constantly monitors, replicates crucial data, and is capable of fast and automatic recovery. It also replicates its "chunks" of data regularly to ensure no data loss.

Implementation of the GFS

Implementation of the Google File System was deployed to accommodate processing data used by Google's service as well as research and development efforts that require large data sets.

It is implemented across thousands of disks on over a thousand machines accessed by hundreds of clients, and is capable of hundreds of terabytes of storage.

When designing the GFS, component failures were taken into account as being the norm, as they are not unusual when working with such a large system (thousands of storage machines built with inexpensive parts).

Also, files are divided into fixed-size chunks; each one 64MB. This is relatively large, but has several advantages: many operations can be performed on a single chunk, and metadata size is reduced considerably (to about 64 bits per chunk)

My Analysis of the GFS

Creating and implementing a file system that caters to their own needs was a very smart move for Google. Having almost unlimited resources has its advantages.

In my opinion, I find it interesting that they accommodated the system to work amidst various failures. It is very practical to make a system that can operate through a few hardware errors, human errors, etc. especially when working with such a large system.

Like I mentioned earlier, I also find it interesting that they divide their data into large chunks of around 64MB, but this also has its practical uses especially when implemented in a system as large as this one.

All in all, the GFS works well: without it, the millions of Google searches entered every day would not be able to be processed and stored correctly.

A Comparison of Approaches to Large-Scale Data Analysis

The main idea of the comparison paper is to compare MapReduce's (MR) model for large-scale data analysis with a very similar control flow of that framework that has existed in parallel SQL database management systems (DBMS) for twenty years.

The writers of the paper evaluated both systems by measuring each one's performance on a collection of tasks ran on an open source MR as well as two parallel DBMSs.

In the end, the observed performance of the parallel DBMSs were very similar to that of the MR.

It seemed that MR was more user-friendly, but behaved worse in some tasks when compared to the parallel DBMSs which have existed for years. In the end, each system had its ups and downs.

Implementation of the comparison tasks

In order to compare and contrast MapReduce's performance with that of a parallel DBMS, a few different experiments were conducted. A few of these include:

"Grep task" – each system must scan through a data set if 100-byte records looking for a three-character pattern.

Data loading – using the command line, a loader program, or by using SQL phrases. The parallel DBMS took the longest time to load 535MB/Node

Task Execution – each system executed some SQL commands and then were evaluated (it was proven that the systems performed about the same)

SQL Join – each system had to perform a complex calculation on two data sets. The Parallel DBMS excelled in this task by utilizing a hash join.

My analysis

Obviously, there is no better way to test two different systems than have them perform the same tasks while analyzing their speed and precision.

I also find it helpful that they had the systems perform a number of analytical tasks as well as critique other parts of the systems such as schema support, indexing, data distribution, flexibility, and ease of use – this shows that the two systems are being reviewed from a human perspective as well as a formal technological perspective.

MR seemed to be more attractive because it is simple to operate – it has only a few main functions.

In conclusion, both systems, and more, will remain popular in the long run. Both systems perform well; each has its ups and downs. It all comes down to personal preference and what specific tasks one will be carrying out.

Comparison of the two papers

The two papers have different themes: the GFS one goes into detail about what the system is made up of and how it is implemented while the Comparison paper compares and contrasts the MapReduce system other, older DBMSs.

Both papers explain their appropriate systems in great detail. What I most got out of reading them was that certain database systems can vary. Google's file system was made to handle large amounts of data that is stored across hundreds of disks and computers, or more. MR and the other parallel DBMSs that were compared were not made to handle such data, but do function very well when performing certain tasks.

All in all, is apparent that MR seems to be popular because of its ease of use – and because it is different. The traditional DBMSs carried out their tasks just as well as MR; a time difference in a measure of milliseconds does not mar the general user, and will barely affect a large database. Like I mentioned before, it comes down to a matter of personal preference: whether one wants to use the traditional DBMS or the new MR system is up to them – in the end they both perform well.

Stonebraker's talk summary

One size does not fit all – Relational DBMS can never be "universal". This further backs up my point mentioned before that each database comes down to personal preference as well as system performance needs based on what tasks need be performed.

He summarizes that column stores are superior and that all new database models will implement column stores in the future (as opposed to row stores). He also goes on to explain that there is a diverse market of database systems.

There also exist a market of NoSQL databases which are gaining popularity, but as of now there are no set standards.

After explaining that there is a lot of opportunity to create new database systems (thanks to new technologies), Stonebraker goes on to talk about the "Innovator's Dilemma" a book which explains how companies try to adapt their system without losing a market share.

To sum up his talk, the once believed notion that "One size fits all" when it comes to DBMSs has actually wounded the field. But, in the current market, this means that it is a great time to be a DBMS researcher!

Advantages/Disadvantages

The GFS, created by Google, was made to suite the needs of Google's processes. They could not implement a system that would not suit their needs optimally, so they created their own. This further reinforces Stonebraker's notion that "one size does not fit all". (*Advantage – If you want something done right, do it yourself!*)

After reading the GFS paper and the Comparison paper, it is clear that one size does not fit all. One needs a database that can do exactly what is required, to the best of that system's ability.

When comparing MR's system and the other parallel DBMSs, it is proven that one system does a few tasks better than the other and vice versa. You would then select which system you need based on what tasks need be performed.

The GFS is built for large data (it manages "chunks" at 64MB each). This would be impractical if implemented for a system that only uses a tenth of that data, for example. (*Disadvantage – but who, other than Google, uses the GFS? …Exactly.)*