The Google File System
A Comparison of Approaches to Large-Scale Data Analysis
"One Size Fits All:" An Idea Whose Time has Come and Gone

Andrew Rokoszak

March 15th 2016

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS Operating Systems Review. Vol. 37. No. 5. ACM, 2003.

Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., Dewitt, D. J., Madden, S., & Stonebraker, M. (2009). A comparison of approaches to large-scale data analysis. Proceedings of the 35th SIGMOD International Conference on Management of Data - SIGMOD '09. Retrieved March 12, 2016.

Stonebraker, M. (Adapter). (n.d.). One Size Fits All- An Idea Whose Time has Come and Gone [Video file]. Retrieved March 13, 2016, from http://kdb.snu.ac.kr/data/stonebraker_talk.mp4

Main Concepts of the The Google File System (GFS)

The Google File System is a scalable distributed file system that has successfully met the storage needs of Google. The file system is used for large distributed data intensive applications, providing fault tolerance while running on inexpensive hardware. It provides high availability as well as data integrity for a large client base. Some of the main features include:

- Critical Data Replication
- Automatic and efficient data recovery
- High aggregate throughput
- High availability

Implementation of the GFS

Interface	Architecture	Single Master	Chunk Size	Metadata
- Includes a familiar file system interface - Files are stored hierarchically in directories; identified by pathnames - GFS has a snapshot and record append operations - Supports the usual operations such as Create, Delete , Open , Close	- A cluster consists of one single master and multiple chunkservers - Files are divided into fixed-size chunks - For reliability, each chunk is replicated on multiple chunkservers	- Allows master to make smart chunk placement and replication decisions - Clients never read and write file data through the master as this would affect the entire system - Clients typically ask for multiple chunks in the same request, and the maser can also include the information for chunks immediately	- The large chunk size reduces the clients need to interact with the maser - Uses a 64MB chunk size which is larger than a typical file system - Each chunk replica is stored as a plain Linux file on a chunkserver - Lazy Space allocation is used to avoid wasting space due to internal fragmentation	- All the metadata is kept is the master's memory - Master does not hold chunk location information persistently - Master scans through entire state periodically to implement "chunk garbage collection" - An operation log is kept that contains historical log of critical metadata changes and the masters uses it to recover its state when need be

Analysis of the idea and implementation

Design:
The file system is efficient, very well organized and extremely reliable. The design assumptions made early on were a key part of the success of the file system.

Interactions:
Pushing the flow of data linearly is a major factor, greatly increasing the efficiency of the file system.

Fault Tolerance:
The file system provides a lot of fault tolerance using fast recovery as well as replication. The ability of the system to replicate and quickly recover files allows the system to be consistent and efficient.

Main Ideas of the Comparison Paper

There are two approaches to large-scale data analysis:

Map/Reduce is a distributed file system that has two main functions. The map function “reads a set of ‘records’ from an input file, does any desired filtering and/or transformations, and then outputs a set of intermediate records in the form of new key/value pairs.”

Parallel DBMS consists of standard relational tables where data is partitioned over cluster nodes. These “systems all support standard relational tables and SQL, and thus the fact that the data is stored on multiple machines is transparent to the end-user.” Here is an example of how it works through Join Processing: T1 Joins T2

If T2 is small, copy T2 to all the machines
If T2 is large, then hash partition T1 and T2 and send partitions to different machines (this is similar to the split-copy in MapReduce

Implementation of Comparison Paper

Schema Support	Programming Model & Flexibility	Indexing	Execution Strategy	Data Transfers
- MapReduce: -Flexible, programmers can write code to understand input data -Not effective if data is shared by multiple applications	- **MapReduce:** -very flexible -Low level of modeling	**- MapReduce:** -no native indexing support found -programmers can implement their own index support in Map/Reduce code -difficult to share customized indexes among applications	**- MapReduce:** - results are saved to local files -if a node fails, you can run the node-task again on some other node -possible poor performance from multiple reducers reading multiple local files simultaneously	**- MapReduce:** -programmer must perform tasks manually -decide where to schedule map instances
-Parallel DBMS: -Relational schemas are required -Good for data shared by multiple applications	**-Parallel DBMS:** -SQL -User defined functions, stored procedures, user-defined aggregates	**-Parallel DBMS:** -Hash and B-tree indexes are well supported	**-Parallel DBMS:** - intermediate results are pushed across the network - if a node fails, must re-run the entire query	**-Parallel DBMS:** -a parallel query optimizer balances computational workloads while minimizing the amount data transmitted over the network connecting the nodes of the cluster

Analysis of Map/Reduce and Parallel DBMS

Parallel DBM systems performed better than MR systems like Hadoop when “executing data intensive analysis benchmarks.” The Map Reduce system wastes vast amount of energy when at work. On the oter hand MR systems are certainly better regarding recovery of files after equipment failures. At the same time because an “MR system needs 1,000 nodes to match the performance of a 100 node parallel database system, it is ten times more likely that a node will fail while a query is executing.” Overall the Parallel DBMS seems to be a more efficient system for data management.

Comparison of Paper Ideas

After evaluating both papers it is clear that The Google File System and Map/Reduce systems have many similarities. Parallel DBMSs are much more difficult to install and configure appropriately compared to Map/Reduce systems. The GFS places files into chunks of data that are stored, while parallel DBMSs still use the basic SQL functions to store data. The GFS still seems to be in general a superior system despite using inexpensive, and often faulty hardware to support the system. Additionally the basis of Map/Reduce focuses on data recovery which is an important feature of the GFS.

Stonebraker Discussion: One Size Fits All

The premise of the paper focuses on providing users of these data management systems with the most efficient, accurate and speedy access to the data. Stonebraker discusses the idea of Relational Database Management Systems as essential to all firms. Specifically, on-line transaction processing will make a major shift in the way it manages data. He then discusses the ideas of data warehouses where firms store all of the information to be used by employees. He projected that all major vendors would have column stores, twice as fast as row stores. Stonebraker also discusses the idea that databases must be”read-optimized” as most of the action on the system is looking up historical data in the system. He ends with the outlook on the future, predicting the DBMS market to be expanding and evolving as more applications need more efficient data management solutions.

Comparison of GFS, M/R and Parallel, & Stonebraker

The Google File System is an adapted version of the basics of a Map/Reduce data management system. Stonebrakers states “the commercial world will fracture into a collection of independent database engines” which reflects the evolution of the GFS from basic concepts of Map/Reduce. The GFS’s ability to provide stable fault tolerance while running on inexpensive commodity hardware is the future of data management. While Parallel DBMSs offer some benefits, I reiterate Stonebrakers idea that firms will continue to develop and implement variations of systems to fit their needs. As his saying goes “one size fits all” is not true anymore.

The main advantages of the GFS are the systems reliability, pushing data linearly in an attempt to push data as fast as possible, and high availability regarding fast recovery for the master and chunkservers. On the other hand, the system is not ideal for smaller files as many users accessing those files can create a hotspot. Furthermore frequent component failures are a norm rather than an infrequent occurrence. This makes the system often unreliable and difficult to maintain.