“The Google File System”

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google File System." *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles - SOSP '03* (2003): n. pag. Web. 12 Mar. 2016.

“A Comparison Approaches to Large-Scale Data Analysis”

Pavlo, Andrew, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. Dewitt, Samuel Madden, and Michael Stonebraker. "A Comparison of Approaches to Large-scale Data Analysis." *Proceedings of the 35th SIGMOD International Conference on Management of Data - SIGMOD '09* (2009): n. pag. Web. 13 Mar. 2016.

Rafael Marmol
3/12/16

Main Idea of “The Google File System”

- Explains why the Google File System was created and how it works
  - Clients and Google needed a file system that fit their application workloads, provided fault tolerance, scalability, security, and supported large-scale data processing.
  - Shares many of same goals as previous distributed file systems, but changes have been made after some observations
  - States the GFS works successfully with meeting storage needs
  - Presents extensions designed to support applications
  - Talks about aspects of the design of GFS
  - Reports measurements taken from experiments and real world use

How is the idea implemented?

- GFS is implemented through

- **A single master:** having a single master simplifies design and makes chunk placement/replication decisions easier; instead of accessing chunks, client accesses master, which makes the system faster; maintains all file system metadata

-**Metadata:** 3 types (file and chunk namespaces, mapping from files to chunks, locations of each chunk's replicas); master does not keep constant record of which chunkservers have a replica of a given chunk, rather it gets it from the chunkservers at startup; includes operation log

-**Operation log:** historical record of metadata changes; CENTRAL to GFS; serves as a logical time line that defines order of concurrent operations; files and chunks are uniquely defined by operation log; replicated on multiple machines to not lose data; master can recovers its file system state by replaying operation log; log must be kept small

- **Shadow masters:** provide read-only access to file system, even if master is down; enhance readability for files not being mutated; keeps itself updated by reading operation log

- **Chunks:** files are broken down into smaller chunks (64 MB) which are replicated for fault tolerance; large chunk size is good because it only requires one call to the master to get chunk location, which helps get load off of master;

-**Chunkservers:** store chunks on local disks as Linux files; chunks data replicated on multiple chunkservers; master periodically communicates through HeartBeat messages to give it instructions and collect its state

-**Leases:** whenever a mutation occurs, a lease is given from master to "primary" chunk (chosen replica chunk by master); the primary then chooses a serial order for all mutations to the chunk and replicas follow this order; leases help take effort away from master

-**Garbage Collection:** after a file is deleted, GFS doesn't immediately let it go and free up physical storage; instead, the file is logged as deleted, is renamed to a hidden name, and during a regular scan by the master, any hidden named files older than 3 days are deleted (this helps with recovering accidentally deleted data) and afterwards, the storage is freed; anything intentionally deleted twice is immediately removed

-**Checksums:** used to detect corruption of stored data; regularly experienced disk failures cause data corruption/loss; chunk replicas are spread throughout multiple chunkservers on multiple racks with their own checksum values; each chunkserver must independently verify integrity of its own copy by maintaining checksums

-**Record appends:** GFS used record appends to preserve Atomicity; instead of creating new files, additional data is appended to it

My analysis of idea and implementation

-In my own words, the Google File System is a hierarchical system that requires minimal memory, covers instances of failure, damage, and data corruption/loss, provides quality and quantity usage to clients, and learns from older file system methods to better itself.

-I believe that the implementation of breaking down files into replicated chunks was a great idea since losing data can be a monumental loss.

-Also, I think that keeping deleted files as hidden is another good implementation since people are often stupid and will make errors and the fact that the GFS covers that stupidity is great

-The idea of having a master control everything was good since it controls almost everything with minimal memory usage by assigning jobs to chunkservers

-In addition, I believe the GFS may not work as well for other applications that are not made by Google since the article states that it was mainly developed for their own applications and if clients want to use it more efficiently, they would have to use Google applications

Main ideas of Comparison Paper

-Comparison of database management systems (DBMS) to MapReduce (MR)

-Believe DBMS to be the better system

-MR can make adjustments to be better

-Evaluate both DBMS and MR on performance and execution

-Show DBSM to be the better system on various tasks

-Exemplify how MR may be easier to learn but over the haul, DBMS may be the better choice

How are ideas implemented?

-Series of tests and comparisons are run through 2 DBMS (DBMS-X, Vertica) and an MR (Hadoop)

-**Fault tolerance:** MR frameworks provide a better failure model than DBMS; MR is far better at handling node failures since if a node fails, another one is used, but in DBMS, if a node fails, the entire query must be restarted

-**Data Loading:** Loading times for each system increases in proportion to number of nodes used; Hadoop ran faster since it directly loads files in its internal storage system, while DBSM execute a UDF that processes documents on each node at runtime and loads data on a temporary table

-**Selection Task:** both parallel DBMS outperform Hadoop by a significant amount; this is because DBMS use indexes, while MR's do not

-**Aggregation Task:** both Vertica and DBMS-X outperform Hadoop; it is more advantageous to use a column-store for this task, but MR's don't use tables and schemas so it is at a disadvantage here

-**Join Task:** this task results in biggest performance difference between Hadoop and parallel DBS; MR does not have the JOIN capability used in SQL and needs to make up for by doing a complete table scan which takes up significantly more time

-**UDF Aggregation Task:** both DBMS-X and Vertica execute most of the tasks faster than Hadoop at scaling levels

-**Task Start-Up:** DBSM is considered "warm" since nodes are ready at boot time and are always waiting for a query to run, while Hadoop and also GFS are "cold" which means it takes time before all nodes are running at full capacity

-**Compression:** all parallel DBMS (including DBMS-X and Vertica) allow for optional compression of stored data, while Hadoop supports block and record level, but it requires more effort to do so

-**Ease of use:** MR programs are written primarily in Java, which is easier for programmers to adhere to, while SQL commands need to be taught; MR does not follow a set schema like DBMS so a programmer needs to structure the data on MR and that set structure they made needs to be shared with everyone else, which makes large-scale use of MR very difficult since everyone using it needs to be informed constantly

-**Additional tools**: SQL databases have tons of existing tools and applications, while Hadoop only has a rudimentary web interface with no addtional tools that would have to be developed

Analysis of those ideas and implementations

-In my own words, the main idea of the comparison paper was to show that although DBMS may be older than the newer, more attractive MR system, it still holds significant value and is tested and proven to be better than MR in many ways

-After analyzing the way both are set up, I realized that the fact that MR's do not have a set schema and are unstructured unsettles me. Probably, because to me it seems that a lack of structure leaves room for a lot of errors, which is what the paper points out

-After seeing the results of each test, I believe that DBMS's are significantly better even if SQL is more annoying than Java because in the long run DBMSs are far better. Also, as stated in the paper combining aspects of both systems will most definitely create a better overall system.

Comparison of ideas/implementations in both

-After reading the comparison paper, I believe that the DBMS is better than MR type systems like GFS. Although the GFS has its strengths, especially in data corruption, deletion, replication, and concurrent mutations, it doesn’t hold up well to the tests that were performed in the comparison paper. However, like stated in the comparison paper, I believe that the GFS can be improved through implementing SQL and other DBMS aspects like set schemas and having a set program that isn’t one that needs to be updated, changed, and is one that everyone knows instead of having to cipher what someone else’s work. In addition, I believe that the MR of GFS works better for Google since they develop applications specifically for their GFS and vice versa. Even though MR systems like Hadoop and GFS are attractive and fancy, nothing beats the basics of DBMS; it holds up reliability and structure and has held its value for over two decades.

Main Ideas of Stonebreaker Talk

-Used to believe in “one size fits all for row stores”, but then realized later that “one size fits none” as row store is becoming obsolete in place of the better implemented column store

-Data Warehouse: all vendors use column stores

-OLTP: moving towards main memory deployments

-NoSQL Market: small, but increasing; more ideas with no standards

-Complex Analytics Market: use data warehouses to run business projects; data scientists will soon replace business analysts; can simulate this in SQL, but it is very slow; most likely to be taken over by column store rather than row store

-Streaming Market: not based on traditional row stores; OLTP engines have a greater market share which is extremely fast

-Graph Analytics: Facebook is an example; can be used through column stores, array engines, and graph engines; no dominant processing yet

-Overall, there are a huge variety of engines, row stores basically suck for all markets, and all markets are geared towards vertical/column stores and applications

-Great opportunity for new ideas in data!

-Vendors moving from old stuff to new stuff will most likely experience difficulties

-In the 80s and 90s DBMS was “dead on our feet” because they believed in “one size fits all” which is completely dead

-Its a great time to be a DBMS researcher

Advantages/Disadvantages of main idea of chosen paper in context of comparison paper and the Stonebreaker Talk

-**<u>Advantages:</u>**

-New ideas are good for the market (GFS is new and Google is a new attractive company so they will definitely make more money)

-Ideas change over time: DBMS were shown to be better than MR models like GFS, but just like Stonebreaker changed his mind from “one size fits all” to “one size fits none”, MR models could be more prominent and dominant in the next decade or so

-More people are moving away from older methods and clinging onto newer ideas

-GFS is easier to understand for programmers than regular SQL

-Everyone loves Google so it will probably take hold vastly and quick

-**<u>Disadvantages:</u>**

-GFS is not a column store DBMS (instead it follows the MR model) which takes up the majority of the markets Stonebreaker was mentioning

-GFS was made mainly for Google applications and in order to reach those markets, changes need to be made

-GFS does not follow the DBMS schema, so it has no structure leaving it without relational rules and is thus prone to errors

-Since GFS is similar to MR, it lacks the execution that DBMS was shown to have in the task analyses of the comparison paper