Big Data Paper Summary

By Nicholas Russell

12 December 2014

Ghemawat, S., Gobioff, H., & Leung, S. (2003). The Google file system. *ACM SIGOPS Operating Systems Review*, 29-29. Retrieved December 3, 2014.

Pavlo, A., Paulson, E., Rasin, A., Abadi, D., DeWitt, D., Madden, S., & Stonebreaker, M. (2009). A Comparison of Approaches to Large-Scale Data Analysis. *SIGMOD*

The Main Idea of GFS

- When designing the system, the assumptions for the system were as follows:
  - Components will often fail and must monitor themselves
  - Optimize the multi-GB files
  - Workloads consist of large streaming reads and small random reads
  - Workloads will have writes that append to data files
  - Atomicity with the minimal synchronization overhead is essential
  - High bandwidth is much more important than low latency
- Has snapshot and record append operations which implements "mutli-way merge results and producer-consumer queues that many clients can simultaneously append to without additional locking".
- The system will have one master and multiple chunkserves to keep track of all of the different long term writes that the file system will need to contain.
  - Want to minimize master's involvement in all operations.
- Chunking and chunk replicas allows the system to create more chunks and store data more effectively, re-replicate the data at will for any users, and to maximize data reliability and availability.
- Needs to treat component failures as "the norm rather than the exception, optimize for huge files that are mostly appended to… and then read, and both extend and relax the standard file system interface to improve the overall system.

Implementation of GFS

- The architecture of the system allows each file to be divided up into chunks, and each chunk is stored on multiple chunkservers (default is 3 servers). Each chunk is 64 MB.
- The master will maintain all of the metadata which includes all of the mapping from the files to chunks and all of the meta data is kept in the master's memory.
- Mutations of data, which is "an operation that changes the content or metadata of a chunk", are frequent and by GFS, each file namespace mutation is atomic and is exclusively handled by the master. By namespace locking, this guarantees atomicity and correctness.
- The client pushes all of the data to the replicas and will send a write request to the primary. After the primary replica forwards the request to all of the secondary replicas, if they reply to the primary completeness, then the primary will reply to the client to state if the modified region has in fact been modified or if some error has occurred.
- GFS decouples the data flow by pushing data linearly through a chain of chunkservers rather than going through a tree of some sort. Also, each machine forwards their data to the machine that is closest to them that has not received the data as of yet. The GFS minimizes latency by "pipelining the data transfer over TCP connections", which allows the chunkserver to immediately forwarding.
- GFS does not have a directory data structure that shows all of the files in the directory. The file system represents its namespaces logically as a "lookup table mapping full pathnames to metadata". Every single node in the namespace has a read-write lock that is unique to itself.
- For garbage collection, the system does not immediately reclaim the space once something is deleted. The file will be renamed to a hidden name which includes the exact time and date that it was deleted. Then, on the master's regular scan of the namespace, it will just remove the hidden files that have existed for more than three days. Once the hidden file is removed, its metadata will be erased and will effectively sever any links to the chunks.
- To implement snapshots (which users use to quickly create copies of huge data sets), GFS uses the standard copy-on-write techniques which will give the master an opportunity to create a new copy of the chunk first.

Analysis of GFS

- The GFS is extremely efficient and allows the system to problem solve itself. The system is also very reliable since data is backed up onto multiple chunkservers and deleted is not removed from the system until three days have gone by.
- The large chunk size reduces the client's need to interact with the master, reduces the network overhead by keeping a persistent TCP connection to the chunk server over an extended period of time, and allows the metadata to be stored on memory which makes it easy and more efficient for the master to scan through all of the namespaces and files.
- The atomicity and correctness of the data allows for clients to retrieve data accurately and efficiently. This also allows the clients just to see what the mutation has written.
- The snapshots allow the users to checkpoint the current state of their data before experimenting with changes that can be eventually rolled back or committed.
- The code is written in Linux, which causes problems by corrupting data. However, the reliability and accessibility of Linux has let the GFS administrators to explore and understand the system's behaviors.
- For the large-scale data processing that Google requires, the file system that they have implemented provides efficiency in storage, reliability of data retrieval, automatic recovery, and provides unparalleled fault tolerance that allows the master and chunkservers to control themselves with regards to repairing, creating, writing, reading, and deleting.

Comparison of the MapReduce to Parallel DBMS

- In the second paper, Pavlo et al. talk about the newest revolution of "cluster computing", just as Google implemented in their file system. The paper goes on to talk about the advantages and disadvantages of MapReduce (more specifically Hadoop in this paper) and parallel DBMS (Vertica).
- Where parallel DBMSs use the relational model of rows and columns, MapReduce does not require the relational model and that allows the programmer to structure their data as they please.
- Parallel DBMSs uses a parallel query optimizer which strives to balance the workloads of the computer while minimizing the amount of data transmitted over the network connecting all of the nodes. The MapReduce programmer will have to write all of these programs manually and it will take much more time to complete since the filtration is done and the statistics are computed.
- The MapReduce provides a much more sophisticated model for fault tolerance since for Parallel DBMS, if a single node fails during a query, the entire query must be completely restarted.
- Parallel DBMS can be challenging to grasp, while MapReduce systems are easy to grasp because it provides a simple model through which the users can express sophisticated programs with relative ease.

Advantages and Disadvantages of GFS

- The GFS, which uses a MapReduce to run, takes time before all of the chunkservers and nodes are running at full capacity whereas the parallel DMBS were always waiting for a query to execute and therefore start-up time is much shorter for the DMBS than the GFS.
- The multi-thousand-node cluster in the Google File System runs an exorbitant amount of energy and wastes large amounts of energy. They state that even though "it is rumored that the Google version of [MapReduce] is faster than the Hadoop version," which the paper was testing on, "…We are doubtful… that there would be a substantial difference in the performance of the two versions as MR is always forced to start a query with a scan of the entire input file," which causes lots of energy to be wasted.
- MapReduce is extremely easy to set up and use in comparison to the Parallel DMBS, which allows the administrators and programmers lots of freedom and user-friendly programming.
- The maintenance of the MapReduce programs, over time, will be a pain for the programmers over time because the need to constantly modify code can be time-exhausting and very repetitive.
- Both the MapReduce and Parallel DBMS share performance advantages
  - "B-tree indices to speed the execution of selection operations"
  - "Novel storage mechanisms"
  - "Aggressive compression techniques with ability to operate on compressed data"
  - "Sophisticated parallel algorithms for querying large amounts of relational data"
- Extensibility in the MapReduce programs were much more developed and efficient than the Parallel DBMS
- MapReduce does a much superior job of minimizing the amount of work that is lost when a hardware failure occurs, but the performance penalty can be staggering.