Big Data Paper Summary

Papers, Concepts and Works of Art Summarized by Madeline Atwood
October 29, 2020

The Google File System

By Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Choosing A Cloud DBMS: Architectures and Tradeoffs

Junjay Tan[3], Thanaa Ghanem[2], Matthew Perron[3], Xiangyao Yu[3], Michael Stonebraker[3,6], David DeWitt[3], Marco Serafini[4], Ashraf Aboulnaga[5], Tim Krask[3]

[1]Brown University; [2]Metropolitan State University (Minnesota), CSC; [3]MIT CSAIL;
[4]University of Massachusetts Amherst, CICS; [5]Qatar Computing Research Institute, HBKU;
[6]Tamr, Inc.

Michael Stonebraker on his 10-Year Most Influential Paper Award

at ICDE 2015

Google’s DBMS provides fault tolerance while running on inexpensive commodity hardware, and it delivers high levels of performance to a large number of clients.

- The system is built from many inexpensive commodity hardware and components that often fail. It must constantly monitor itself and detect, tolerate, and recover promptly from component failures on a routine basis.
- Being that this file system has successfully met all storage needs, present file system interface extensions and features are designed to support the future distribution of more further advancing applications.
- The Database runs on four main values being:

  1→ Component failures are the norm rather than the exception.

  2→ Files are huge by traditional standards (optimize for huge files that are mostly appended to and then read sequentially).

  3→ Most files are mutated by appending new data rather than overwriting existing data (Once written, the files are only read, and often only sequentially, then dismissed).

  4→ Co-designing the applications and the file system API benefits the overall system by increasing flexibility.

- The system provides fault tolerance by constant monitoring, replicating crucial data, and fast and automatic recovery through Chunkservers and chunk monitoring.
- Overall: Meets storage needs and is widely used within Google as the storage platform for research and development as well as production data processing due to their observational processed thinking about data implementations.

Chunk Implementations

Chunk replication allows us to tolerate chunkserver failures. The frequency of these failures motivated a novel online repair mechanism that regularly and transparently repairs the damage and compensates for lost replicas as soon as possible.

Master involvement in common operations is minimized by a large chunk size and by chunk leases, which delegates authority to primary replicas in data mutations. This makes possible a simple, centralized master that does not become a bottleneck. We believe that improvements in our networking stack will lift the current limitation on the write throughput seen by an individual client.

Chunk size is one of the key design parameters. The chunk size is 64 MB, which is much larger than typical file system block sizes (which means small files may become hot spots if many clients are accessing the same file), but offers several important advantages:

1→ It reduces clients' need to interact with the master because reads and writes on the same chunk require only one initial request to the master for chunk location information, this is especially significant because applications mostly read and write large files sequentially.

2→ Since on a large chunk, a client is more likely to perform many operations on a given chunk, it can reduce network overhead by keeping a persistent TCP connection to the chunkserver over an extended period of time.

3→ It reduces the size of the metadata stored on the master. This allows us to keep the metadata in memory, which ultimately brings on more advantages such as the metadata being controlled directly by the master.

Since metadata is stored in memory, master operations are fast. It is also easy and efficient for the master to periodically scan through its entire state in the background. This periodic scanning is used to implement chunk garbage collection, re-replication in the presence of chunkserver failures, and chunk migration to balance load and disk space usage across chunk servers- again helping with the overall storage demands of data that Google faces everyday.

One potential concern for this memory-only approach is that the number of chunks and hence the capacity of the whole system is limited by how much memory the master has but if necessary to support even larger file systems, the cost of adding extra memory to the master is a small price to pay for the simplicity, reliability, performance, and flexibility gained by storing the metadata in memory.

Personal Analysis

- Since the design of this system has been driven by many different forms of observations from past systems and the current system, I feel that this more scientifically created than fully experiment based. After all, observations lead to hypotheses which lead to better scientific designs and trust worthy products released.

- Being that the files are organized hierarchically in directories and are identified by path names and usual operations such as create, delete, open, close, read and write files, it seems to be very user (or master) friendly and is easily manageable for larger groups of admins and system users.

- One of my favorite features of this database is the snapshot feature (just like our beloved CAP database), I feel that the snapshot feature is really important and convenient for people who are more visual data analysers. Also the snapshot operation makes a copy of a file or a directory tree so you can edit the snapshots without worrying about the loss of any wide scale data.

- Their data integrity system is interesting: “chunkservers will not propagate corruptions to other machines. If a block does not match the recorded checksum, the chunkserver returns an error to the requestor and reports the mismatch to the master. In response, the requestor will read from other replicas, while the master will clone the chunk from another replica. After a valid new replica is in place, the master instructs the chunkserver that reported the mismatch to delete its replica.” I think it is important to have a data integrity system like this to ensure no important data is lost but at the same time we are able to update data to match the needed standards for the users and the system at large.

- My biggest concern with the Google File System is that the master would somehow malfunction or not work properly and the GFS would loose large portions of their data or more importantly- shut down entirely. Placing large scales of trust in just function is scary but as long as it has proven itself reliable through research and their apparent extensive observations and history analysis, I have no other concerns with the GFS DBMS. I’m excited to see the future developments and evolution of the system.

Choosing a cloud DBMS is best done when using specific architecture tradeoffs such as multiple cloud node systems (Vertica, Redshift/Spectrum and Presto) which is most effective to make sure cloud speed and storage is optimized but also allows for maximum compatibility.

- As analytic (OLAP) applications move to the cloud, DBMSs have shifted from a locally attached storage design to a hybrid design that combines the use of shared-storage (e.g., AWS S3) with the use of shared query execution mechanisms- currently known as “the cloud”.

- As a result of evolving technology, many organizations are moving their applications to the cloud.

- For analytic applications running on the cloud, data resides on external shared storage systems (ex S3 or EBS) which are generally offered on Amazon Web Services (AWS).

- For the most part, the total system costs are composed of node compute costs, storage costs, and data scan costs.

- There is a large amount of data compatibility with other systems, this means that data used by one system can be accessed by another system (which is also very good for companies that are transitioning between systems or need multiple systems to run their systems). Otherwise, there will be significant extract, transform, and load (ETL) costs if one wants to switch to a different system for targeted workloads. Because the cloud offers the ability to easily launch new systems, using different systems for different workloads seems to be one of the biggest flexes of the entire cloud at large.

- The cloud provides the most flexibility for future system optimization over large storage formats used by DBMSs such as Redshift and Vertica (again, allowing for interchangeability).

Cloud Idea Implementations

- Prioritizing low-cost object stores like S3 for data storage and using system columnar formats like ORC (Oracle Recruiting Cloud) that allow easy switching to other systems for different workloads, and making features that benefit subsequent runs like caching remote data will make faster storage optional rather than required because they disadvantage ad hoc queries which is query that you loosely type out when you need it but it can end up being stored for longer periods of time and taking up more data.

- On AWS (Amazon's database migration service which is often used in the cloud), the main file systems are two block store options– Elastic Block Store (EBS) and Instance Store (InS)–and the Simple Storage Service (S3) object store. Block stores use a standard file system API and are offered as Solid State Disk (SSD) or Hard Disk Drive (HDD) which is another strong representation of all the mobility offered by cloud systems.

- EBS is remote network storage that is co-located in specified regions, while InS is physically attached to the compute node. In contrast to EBS, InS is not consistent and disappears once the compute node is shut down. As a result of these reasons, EBS is traditionally the more suitable option for DBMSs which is why it is more strongly implemented in cloud systems such as AWS.

- Different system architectures used that are offered in the cloud highlight interesting trade offs when they are compared to non-proprietary systems. For example, "the aggressive intraquery parallelism of Redshift can offer an order of magnitude performance advantage for single user workloads," but doing so causes significantly worse performance as concurrency increases. Similarly, query compilation to machine code performed by Redshift speeds up CPU-intensive queries but reduces scale out performance gains on largely diverse workloads.

Personal Analysis

My personal takeaway was how great the prices are considering the greatness of cloud systems being that they use so many different functions and cloud nodes. Bringing some focus to prices:

- S3 and EBS storage costs used the currently listed AWS costs: $0.023 per GB for S3, and $0.10 per GB for general purpose (gp2) EBS. Presto, Vertica, and Hive on S3 utilize a scratch EBS disk (512 GB per node) to support spill-to- disk → relatively cheap!

- The costs are calculated from the query runtimes and other associated node runtime processes (data snapshot load for Redshift, etc.) which are always something that can increase or decrease based on amount of storage being used and updates in software factors. There seems to be a lot of potential for the future of the cloud.

- To save money, time and resources (especially speed of processing data) I have learned that it is advantageous in the cloud to shut down compute resources when they are not being used, but there is then a query latency cost. All cloud nodes do require time to initialize and pass system checks before a DBMS can be started (systems using local storage like Redshift take the longest to reinitialize), which takes even longer when the system is shutdown- therefore using serverless offerings like Athena provide an alternative "instant on" query service.

- Overall, the main takeaway for the future of the cloud is positive and exciting. I can see big things coming from these implementations since there are so many different ways to complete tasks based off of the openness the cloud developers have created in the process of making this system. "Being able to choose the optimal configuration and leverage the full breadth of DBMS-as-a-service offerings can provide significant cost and performance benefits."

A comparison of the ideas and implementations of the two papers

Cloud:

Dealing with external storage:
Cloud providers offer multiple storage services with different semantics, performance, and cost due to their use of multiple nodes.
These data are accessible via a web-based REST API. Google also uses API web-based, it almost like the two systems are relying on each other!
Internal storage:

Block stores use a standard file system API and are offered as Solid State Disk (SSD) or Hard Disk Drive (HDD)

More external:
In contrast to block stores, S3 is an object store that runs on S3 nodes, and storage is accessed using a web-based REST API.

SNAPSHOT: Redshift node snapshots are free, Redshift takes longest to initialize because “it must start the nodes and then load data from an S3 snapshot to the local NVMe drives”→ Redshift charges while data is being reinitialized from snapshots which something to keep in mind when considering cost.

COST: Consider query cost and storage cost separately because although they are related, they can be modified separately via “contractual pricing schemes.” In general, query cost is directly correlated with query performance so it can range in cost based off of many outside factors.

Interestingly, Redshift is both the best and worst performer on query cost. Standard shared-nothing Redshift, which reads data from local node storage, is cheapest because single user queries run extremely fast. However, Redshift Spectrum is about 3x the cost of other options even though its query performance is similar to other systems

Google:

GFS also uses the API file system which is linked into every application and communicates with the master and chunk servers to read or write data on behalf of the application. Clients interact with the master for metadata operations, but all data-bearing communication goes directly to the chunkservers. Although GFS does not utilize multiple systems like the cloud, it does put a lot of trust into the master which can be good when you have everything running on one system and don’t want to use seperate programs but can be bad if there are multiple tasks that need to all be completed differently.

SNAPSHOT: GFS has snapshot- snapshot is master heavy just as mostly everything else in GFS- when the master receives a snapshot request, it first “revokes any outstanding leases on the chunks in the files it is about to snapshot.” This ensures that any subsequent writes to these chunks will require an interaction with the master to find the lease hold. This adds additional time to the processes the master is completing but does not increase the price such as the cloud node “Redshift.”

COST: Snapshot creates a copy of a file or a directory tree at low cost. If necessary to support even larger file systems, the cost of adding extra memory to the master is “a small price to pay for the simplicity, reliability, performance, and flexibility” that is gained by storing the metadata in memory.

Stonebraker Talk Main Idea and Analysis

- The main idea of the Stonebraker talk is that Database Management Systems are currently in an extreme evolutionary stage. In the past, DBMS researchers have followed the philosophy that "one size fits all," but over time it has become apparent that one size fits none because the traditional row stores are obsolete and are proving to be good at nothing. Over time it has become apparent that most of the major markets now have a huge diversity in engines (traditional row stores from the past being good at none of these). Traditional vendors are the legacy code which was nice for a time but now there is a whole bunch of new ideas that are significant and good for running in the future. We can expect to see a lot more implementations in the future- especially more processor diversity which is a great help in accelerating DBMSs.

- To me it sounds like this developmental time for researching the best DBMS is kind of like watching a race. There are so many new ideas being implemented out there and it is going to be very interesting to see what market will come out on the top. Each market is showing strengths and each is showing weaknesses- it will be how they play off of each other's ideas and come together or break apart that will help find the ultimate DBMS.

- With new market leaders emerging, older ones are being dropped. SAP is Oracle's largest customer but SAP is unhappy because their databases are staged mainly on Oracle storage, so they have a system called hana which appears to be working a lot better. There is reasonable suspicion that SAP may soon stop supporting Oracle and develop their own lead in the market- making SAP being Oracle's new largest competitor.

Advantages and disadvantages of the main idea of the chosen paper in the context of the comparison paper and the Stonebraker talk

The google filing system (GFS) is a scalable file system that is currently on inexpensive commodity hardware that largely provides fault tolerance due to the large involvement of the master and high aggregate performance. By storing the metadata in the memory and using chunkservers to accomplish most program tasks there is a lot of room for increased speeds but also the data is a lot safer and reliable because it is all analyzed by the master. They treat failures as the norm and used extreme observation to develop the GFS.

The main advantages of using chunkservers are that it reduces interaction with the master so instead of the master scanning large bits of data, it is instead scanning the chunks. This increases the speed of the overall system which also reduces the metadata stored in the memory. A disadvantage to using server chunks is that there may be hotspots developed due to the different sizings and demand of any number of chunks. Since GFS provides POSIX-like interface but does not offer full support, in certain cases I would recommend using a system like the cloud since there are multiple nodes implemented which can do different tasks but still remain connected. On top of this, HDFS (Hadoop) uses GFS and is an open source implementation currently running in Java which follows the same overall design but supports random writes, supports appending to existing files and supports multiple concurrent writers which may suggest that HDFS may be taking the ideas GFS has released and turning them into a more compatible software while still using a similar data memory and storage processing system. Stormbraker would likely not be surprised because he predicted that there are many software engineers out here who are taking all that we currently know and blending it with what we previously thought would be the best there was to create something better. Software is constantly evolving and you can either work with many different softwares (such as having multiple cloud nodes) or take what we know and blend it into one program that relies only upon itself. The future is limitless and will continue to evolve until we’ve reached the unimaginable for technology.