The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-tak Leung

- and -

A Comparison of Approaches to Large-Scale Data Analysis

Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Michael Stonebraker

Sara Ogorzalek

March 7, 2017

The Google File System
Main Idea

- The Google File System provides **fault tolerance**, runs on inexpensive commodity hardware, and delivers **high aggregate performance** to a large number of people.
- GFS goals are performance, scalability, reliability, and availability.
- Exhibits all the qualities essential to support large-scale data processing workloads on commodity hardware.

The Google File System
Idea Implementation

- For design, the Google File System is driven by key observations:
  - Component failures are the norm rather than the exception.
    - File system consists of storage machines build from inexpensive parts and is accessed by many client machines.
    - Constant monitoring, error detection, fault tolerance, and automatic recovery is crucial to the system.
  - Files are huge by traditional standards.
    - Multi-GB files are common, and therefore should be managed effectively.
  - Most files are mutated by appending new data rather than overwriting existing data.
    - Primarily consist of large streaming reads and small random reads.
  - Co-designing the applications and the file system API increases flexibility.
    - Large sequential writes to append data to files.
    - Efficiently implement semantics for multiple clients that concurrently append to the same file.
  - High sustained bandwidth.

The Google File System
Analysis

- The Google File System achieves its goals by separating file system control. This passes through the master from data transfer. Then, this passes between the chunkservers and clients.
  - Large chunk size and chunk leases minimize master involvement in common operations, therefore it does not become a bottleneck.
- Data integrity is ensured by checksumming that detects the corruption of stored data.
- I think that the Google File System can successfully meet any storage needs. Because of the key observations, the system runs better for more needs such as research and development tasks.

A Comparison of Approaches to Large-Scale Data Analysis
Main Idea

- Comparison of **performance** and **development complexity** of two parallel DBMS, Vertica and a system from a major relational vendor, and an open source version of MapReduce, Hadoop
- Both parallel database systems showed a large performance advantage over Hadoop MR in executing a wide variety of data analysis benchmarks.
  - This is largely because database systems have B-tree indices to speed execution, novel storage mechanisms, aggressive compression techniques, and sophisticated parallel algorithms.
- Hadoop was easy to set up and use in comparison to the databases, and is best at saving work that is lost when a hardware failure occurs.
- Overall, there is a lot to learn about both kinds of systems.

A Comparison of Approaches to Large-Scale Data Analysis
Idea Implementation

- Measured systems by performance benchmarks of five tasks used to compare performance. They executed each task three times and reported the average.
- The parallel DBMSs outperformed Hadoop almost in all tasks. Some of this was due to Hadoop's increased start-up costs, the parallel DBMS use clustered indexes, where the MR program has to completely scan the table, and the MR model cannot join two or more disparate data sets inherently.
- Then, the systems were measured for development complexity. This included the installation process, task start-up, and compression.
  - DBMS-X was straightforward, but hard to configure. Vertica was easy to install. Overall, the parallel DBMSs were much more challenging than Hadoop to install and configure properly.
  - For start-up, MR programs took a lot of time before all nodes were running at full capacity. The parallel DBMSs are started at OS boot time
  - Many parallel DBMS have optimal compression of stored data. This is not true with Hadoop.
  - MR is able to recover from faults in the middle of a query, unlike parallel database systems

A Comparison of Approaches to Large-Scale Data Analysis
Analysis

- MapReduce is attractive because of its simplicity. There are two functions, Map and Reduce that are written by the user. It is easy to get started, but maintenance is hard to upkeep.
  - All Map instances use same hash function, therefore all output records with same hash value are stored in the same output file.
  - Better suited for development environments with a small number of programmers and limited application domain because of lack of constraints.
  - Does not require that all data files adhere to a schema – MR programmer is free to structure data however they please, or have no structure at all. Therefore, all developers must agree on a single schema.
    - This not only causes the programmer more work, but a MR framework allows input data to be easily corrupted with bad data.
- Parallel DBMSs have most tables partitioned over the nodes in a cluster and the system uses an optimizer that translates SQL commands into a query plan.
  - They require data to fit into the relational paradigm of rows and columns.
  - Use hash or B-tree indexes to accelerate access to data.
  - Uses a push approach as opposed to a pull.

Comparison of Ideas

- The Google File System is a way to go about large-scale data processing. It is extremely efficient, and serves a large amount of people with high performance.
- MapReduce and parallel database systems are used for different kinds of needs. Overall, the parallel database systems can perform better than MR, but MR is better when a hardware failure occurs.
- Fault tolerance is a big thing that sets Google File System apart. They constantly monitor, replicate crucial data, and they have fast automatic recovery. Their chunk replication allows them to tolerate chunkserver failure. Compared to parallel database systems, failure is extremely intolerable.
- Google File System differs from Hadoop in that it does not support random writes, appending to existing files, and multiple concurrent writers.

One Size Fits All – An Idea Whose Time Has Come and Gone (2005) – Michael Stonebraker Talk

Main Idea

- Relational Database Systems used to be the answer in the 1970s-2000s
  - Attempted to make RDBMS "universal" – one size fits all.
- By 2005, one size did not fit all. Streaming applications could not be put into traditional row-store RDBMS.
  - Column-stores were completely different than row-stores
- In 2015, one size fit none. Row-stores are now obsolete.
  - Many different markets that show row-stores are not good for anything. Column-stores are faster, but depending on the market, there is a large **diversity of engines** oriented toward specific verticals or specific applications.
- New implementations are being developed.

Advantages & Disadvantages

- Advantages to the Google File System include file recovery, chunks replicated on multiple chunkservers for reliability, and reduced interaction with master.
- Disadvantages include small files of small chunks may become hot spots.
- Different markets require different engines depending on the need. It is important to have a variety of engines where each market can choose the one best catered to their needs. Some markets may find MapReduce more useful than parallel database systems, and some may find Google File System to be the most useful.