The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

A Comparison of Approaches to Large-Scale Data Analysis

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

Victoria Confeiteiro

March 7, 2017

Database Management – Labouseur

Main Idea of GFS

- Created with problems of old file systems and both current and anticipated technological environment in mind
  - Treat hardware problems as a norm and account for them as such
    - GFS is built on inexpensive hardware, therefore it must be assumed that at any given time some machines may not be functional
  - Most files are mutated by appending new data, rather than overwriting existing data
    - System can account for small overwrites, but appending files is priority
  - System mostly stores large multi-GB files
    - Managing larger files is priority, smaller files are supported but not the focus
  - High bandwidth is more important than low latency
    - Most applications do not have a time requirement for an individual read or write, so it's more important to be able to process large amounts of data quickly

How GFS is Implemented

- All files are divided into chunks
  - 64 MB size
  - Identified by an immutable and globally unique 64 bit chunk handle
  - Stored on minimum of 3 chunkservers, incase of a machine problem
- System is broken up into Clusters
  - Network of machines, made up of three parts
    - Single Master- maintains all file system metadata, controls system-wide activities to manage chunks
      - Receives requests from clients, replies with chunk handle and location of replicas
    - Clients- request files by their file name and chunk index to perform various operations (read, append, etc.)
      - After receiving the chunk handle and replica location from Master, then send a request to the closest replica located in a chunkserver
    - Chunkserver- stores chunks on local disks using chunk handles as the identifier
      - Once the request is received from the client, the chunkserver responds with the corresponding chunk data

Analysis of GFS

- Dividing the system into large clusters that each contain three parts keeps the design simple and organized
- Anticipating hardware failures is a smart, proactive approach
  - HeartBeat messages are a perfect example, occurs when the master periodically communicates with the chunkserver to learn its status and any possible problems
  - Both the master and the chunkserver are designed to restore their state on their own if they were shutdown for any reason
- Although this is a great system, it is designed specifically for Google and many things would have to be changed if it were to be implemented elsewhere
  - Would have to be adjusted to possible give more of a priority to smaller files rather than larger ones
  - Similarly, adjustments may have to be made to improve how system handles overwrites
- By their own admission, hot spots on chunkservers will most likely occur when system is first put into place and changes need to be made accordingly
  - Extra replications of files that many clients are trying to access must be made
  - Don't know what files these are until the hotspot has already occurred

Main Idea of Comparison Paper

- Compares design and performance of MapReduce and Parallel DBMS
  - MapReduce is simpler, more flexible, and responds to machine failures more efficiently
    - Only has two functions; Map and Reduce
    - Schema is not required, programmer can design their own or have none at all, greater flexibility
    - If a node fails during execution, the scheduler can automatically restart the task on a different node
  - Parallel DBMS is rigid but supports greater functionality
    - All data must fit into traditional relational model consisting of tables, made up of rows and columns
    - Indexing is built-in and multiple indexes can be used on a single table
    - SQL is straight forward and easy to understand; able to execute complex queries using many functions with ease

Implementation of Comparison Paper

- Ran six different tasks using Hadoop, DBMS-X, and Vertica
  - "Grep task" – scan through a plain text file looking for a three-character pattern
    - Hadoop has slowest return time while Vertica had the fastest
  - Data Loading – amount of time it took to load specific data set
    - In every instance, it took longer to load data using DBMS-X, Hadoop had the fastest time
  - Selection Task – lightweight filter to find the pageURLs in the Rankings table with a pageRank above a userdefined threshold
    - Hadoop had the longest return time, Vertica had the shortest
    - Both DBMS are able to outperform Hadoop because they use an index on the pageRank column, as required
  - Aggregation Task – calculate the total adRevenue generated for each sourceIP, grouped by the sourceIP column
    - Hadoop had longest return times in every instance, Vertica had shortest times for the most part
  - Join Task – execute two sub-tasks that perform a complex calculation on two data sets
    - Hadoop took significantly longer and was more time consuming to implement
  - UDF Aggregation Task – compute the inlink count for each document in the data set
    - DBMS-X had the slowest return time, Hadoop was second, and Vertica was the fastest

Analysis of Comparison Paper

- The tests were a great way to compare functionality of these two systems and yielded expected results
  - Even though Hadoop is simpler, this comes at the cost of slower return times, more difficulty in performing complex tasks (Join Task)
  - SQL is more difficult to learn, but there are more functions than in MapReduce
  - Shorter time to load data using Hadoop (Data Loading Task)
- Even though the DBMS model is rigid, the functionality it offers is above MapReduce
  - Requiring all data to be stored in a relational model makes it easier to navigate when writing queries
  - Indexing allows data to be reached faster, which explains the slower return times of Hadoop

Comparison of Ideas and Implementation

- Two papers clearly have opposing views
  - Google File System strays away from the traditional DBMS model
  - Comparison paper proves through various tests that the DBMS model is most functional overall
- Hard to compare because the Google File System was created specifically for the needs of Google
  - Google File System puts a huge emphasis on accounting for hardware errors, other databases with different machines may not have to do this
  - GFS also accounts for many clients trying to access the same file, other databases may not have this problem
  - DBMS can be efficiently used in the creation of almost any database
- Chunkserver vs. Tables
  - Each chunk can be identified by a unique chunk handle
  - Similar to identifying a row in a table based on a primary key
  - Chunks are replicated, while tables are not
    - May cause some inconsistencies

Main Idea of Stonebraker Talk

- DBMS was thought to be "one size fits all", this is not the case anymore
  - Many different data models such as OLTP, NoSQL, and Data Warehouses illustrate this shift
  - Different models are all designed for specific verticals or applications
- Traditional row-stores will soon be obsolete
  - Column stores are much faster and are currently being implemented in new Data Warehouse models
  - Predicts that within the next 10 years, all major vendors will have column store
  - None of the new models mentioned use row-stores
- Looking to the future, expect big changes
  - NVRAM will completely eliminate flash memory, creating greater process diversity
- Existing systems will try to adapt without losing their market share
  - Systems will change underlying processes, but try and keep UI the same

Advantages and Disadvantages

- Advantages
  - Utilizes column stores on chunkservers, which is faster and will be widely implemented in the future
  - Understands that DBMS is not a one size fits all model and made changes to fit specific needs
    - Part of NoSQL market, giving it greater flexibility
    - Provides a good reference model for others looking to break away from DBMS
  - Cluster design is simple and easy to implement
- Disadvantages
  - Part of the NoSQL market
    - No standards, similar to Hadoop, takes longer to perform simple tasks
    - Don't get to manipulate data using powerful SQL functions
    - People familiar with SQL functions might not want to switch
  - Specifically designed with Google in mind
    - As stated before, GFS was designed with the needs of Google in mind and may not suit the needs of any other companies
      - DBMS was created and later added to with the idea in mind to make it universal thus making it easier to use with different types of data systems