The Google File System

By Thomas McArdle
May 5, 2014

Ghemawat,, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google File System." N.d. MS. Google. Web.

Pavlo, Andrew, Erik Paulson, Alexander Rasin, Daniel Abadi, David DeWitt, Samuel Madden, and Michael Stonebraker. "A Comparison of Approaches to Large-Scale Data Analysis." N.d. MS. Web.

The Main Idea of GFS

- GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability
- It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.
- It must constantly monitor itself and detect, tolerate, and recover promptly from component failures on a routine basis.
- The system stores a modest number of large files.
- The system must efficiently implement well-defined-semantics for multiple clients that append to the same file at the same time
- High sustained bandwidth is more important than low latency
- Doesn’t implement a standard API, Files are organized hierarchically in directories and identified by path names.
- Has a snapshot and record append. Snapshots create a copy of file directory tree. Record append allows multiple clients to access the same file at the same time

How it is Implemented

- Master Server
  - Coordinates Clusters
  - Updates Operation Log
  - Stores meta Data
- Files
  - are divided into chunks
  - Each chunk is identified by an immediate and globally unique 64 bit chunk handle assigned at the time of the chunk creation by the master
- Chunk Server
  - Normally multiple chunk servers per master server
  - Contains 64bit data files
  - Accessed by mutilpe clients at the same time
- Chunkserver and user generally run on the same machine, as long as machine resources permit
- Fault Tolerance
  - Recovers Extremely fast due to snap shots and record appends

My Analysis

- I think the google file system is very intricate for how simple its design is
- It was built with the intentions of scaling to an extremely large size
- Even though components may crash often, there are many back ups that are then accessed and the client would never be able to tell that anything went wrong
  - each chunk is replicated on multiple chunkservers on different racks
- Fast Recovery
  - Both the master and the chunkserver are designed to restore their state and start in seconds no matter how they terminated
- Clients and applications can
  - Read, Write and append in parallel
- Very fast and efficient
- Data Integrity
  - Each chunkserver uses checksumming to detect corruption of stored data. Given that a GFS cluster often has thousands of disks on hundreds of machines, it regularly experiences disk failures that cause data corruption or loss on both the read and write paths

Comparison

- Google's file system is essential a type of MapReduce Large Scale Data Anaylsis.
- GFS splits there files up amongst many machines, while MapReduce are spread across multiple nodes.
- GFS create its kind of own architecture compared to other MRs, using a master server and then multiple chunkservers.
- Fault Tolerance
  - MR models, along with the GFS, provide more sophisticated failure models compared to DBMSs.
- Results
  - Loading Data MRs excelled by far compared to the others
    - Especially as the data sizes increased, MRs time showed just how much more efficient they can be
  - Task Execution MRs Fell behind
    - Complete opposite of loading data; MRs took the

Advantages and Disadvantages

Advantages	Disadvantages
• Reliable • Quick • High reliability • Can handle lots of data • Can be accessed by multiple people at the same time and still be fast • Takes into account scalability so it can constantly grow and still be efficient • Very Well built in Fault Tolerance to handle any failures that go wrong • Don't have to conform to any specific schema with arbitrary format.	• More programming needs to go into the initial set up • Programmer often times need to write custom parser in order to derive the appropriate data • Because of the simplicity, often times MRs don't provide their own built in indexes so a programmer would have to implement any if he want to speed up the application • Are designed more for scalabilty than size. • Tests show that MRs are generally slower in retrieving data and joing data