The Google File System

Harris Nagle

CMPT 308 Database Management

April 13th, 2015

Ghemawat, Sanjay, Gobioff, Howard & Leung, Shun-Tak. “The Google File System.” Google 2003

Main Ideas About Google File System

1) The Creation of a scalable distributed proprietary file system made in house by Google

- Separate computers are connected together by a network
- Will use large clusters of commodity hardware

2) This system is designed to …

- Meet the ever-increasing demands of Google
  - Data storage
  - Usage needs
- Be very fault tolerant
- Handle & process multi-TB data
  - Having files be read and written by multiple clients at the same time
  - Manage and organize all files that need to be dealt with on a daily occurrence
- Always favor high bandwidth over low latency!

Implementation

1) Google File System's architecture is composed of the following:
    - One master and multiple check servers
    - Accessed by multiple clients
2) Master handles all metadata and the operations log into memory
    - Makes sure chunk placements and replication decisions using global knowledge.
    - Stores three types of metadata:
        1) File & Chunk namespaces
        2) Mapping from a file to chunks
        3) Locations of chunk's replica's
    - Files will never be overwritten
    - New information will be appended to the file
    - Operation Log: Is the logical timeline that defines the sequence of concurrent operations; only persistent record of metadata
3) Divides Files into 64 MB Chunks
    - Most file systems chunks sizes very between 8KB or 1MB
4) Utilizes lazy space allocation
    - Chunks are allocated only when necessary

My Take on the Google File System

1) Commodity Hardware

- The use of inexpensive components to power such a widely used search engine on the worlds most visited website.
- Cost Effective
    - Google can continue to add more inexpensive Linux Servers for added storage.

2) Having One Master

-Having only one master simplifies the overall design significantly

- Helps with consistency in respective chunks servers
- One master key is risky, but it's Google, so everything will be fine should things hit the fan.
    - Keeps a history of important and critical metadata changes and replicating each chunk in multiple chunk-servers
    - The master and the chunk-server are both designed to recover itself in seconds; does not matter how either were terminated or exited

Advantages

1) Optimized for large scale data processing

-Larger Chunk Size

- Reduces clients' need to interact with master
- Reduces network overhead
- Reduces the size of the metadata stored in the master

- Metadata is stored in the master's memory

- It's easy and efficient to periodically scan through its entire background and state.

2) Redundancy

- By default, chunk-servers store three replicas, as discussed earlier, of every chunk
- Provides higher fraud tolerance

3) Inexpensive

- Uses commodity hardware

- Off the shelves Linux servers

- Does not resort to purchasing new hardware or components

Disadvantages

1) Not optimized for smaller work loads

    - Can still handle small files and random reads/ writes

2) Single master may cause lesser performance

    - May cause a bottleneck

    -Everything is reliant on the master

    - If the master goes down, so will the entire cluster!

3) Redundancy

    - Wastes disk space

    - All replicas may not be up to date

    - Because files are not replaced when new data is appended, it causes bigger file size

Comparisons

1) The large-scale data analysis paper focuses primarily on the map reduce paradigm and then comparing that to the parallel database management systems.
2) They both handle large scale data processing abilities
3) MapReduce is great because of its simplicity, having only two functions: map and reduce
    - Map: from an input file reads, transforms, then outputs a new key/value pair
    - Reduce: combines the records assigned and then writes the records to an output file

4) Parallel DBMS's have all tables partitioned over the nodes in a cluster, and then the system uses an optimizer that translates SQL into execution over multiple nodes