Google File System and BigTable

and tiny bits of HDFS (Hadoop File System) and Chubby

Not in textbook; additional information

DISCLAIMER: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Fall 2015 ; distribution outside this group of students is NOT allowed.*

(c) Copyright A. Gerbessiotis. CS 345 : Fall 2015 . All rights reserved.

Motivation

Fact 1. A cluster consists of 100s to 1000s of cheap commodity PCs (aka servers) that fail regularly (and irregularly) and often.

Fact 2. Multiple clusters worldwide, thousands of queries per second (3 billion queries per day).

Fact 3. One query reads doclists/hitlists extending to several 100s MB of data.

Fact 4. One query requires processing of 10s-100s of billions of CPU cycles per second. Desirable response time < 0.25 sec.

Fact 5. Multiple copies of the web, multiple copies of the index. Each copied stored with redundancy (copies of copies).

Fact 6. Billions of files (aka web-pages) of KBs of size. Total size is in TBs to PBs.

Fact 7. Large files of size several 100MBs (multi GB). Few million of files. Small files not supported (bulk-storage).

Fact 8. Mostly read operations or less-often massive write operations (periodically).

Fact 9 (READ operations). Lots of 100MBs at a time (index). Also, read few KBs at some specific location of a multi-GB file (web-page copy).

Fact 10 (WRITE). Write a new multi GB file (index), or append information to an existing file (web-page, new copy, previous coexists). Write-once files, data appended. Periodic clean-up aka garbage collection (old web-page copies to reduce overall size).

Fact 11 (Throughput vs latency). Throughput is more important than latency. Read operations deal with end-user (of Google) activity and are massive and to the index. Write operations are periodic events to reconstruct the index and are also massive or update of local copies of the Web (that involve few KBs of files at a time) that append information to existing large files.

(c) Copyright A. Gerbessiotis. CS 345 : Fall 2015 . All rights reserved.

Architecture

Intent. Try to build a system that works NOW, and is scalable for the next 2-4 years. Do not try to build the "perfect system" that would be scalable between now and 10 years from now.

Step 0. Global file-system, that spans the address-space of a single machine.

Step 1 (Master server). A single **master server**, and 2 **shadow servers** ready to replace it. (Shadows only allow read-only access to data.) NO DATA CACHING. Clients refer to master by domain name, not IP address.

Step 2 (Chunk-servers). Several **chunk-servers**. Any large file is split into a collection of 64MB-128MB (in 2004-2008, one chunk was 64MB) pieces called **chunks**. Every chunk is **replicated** usually **three times**, or a number of times that is user-defined otherwise. Chunk servers communicate with master in **heartbeat messages** to inform on chunk location, update chunk-leases (60 secs or so). One **primary chunk-server** that coordinates write/read operations on a chunk. Master initiates replication of data with chunk-server primary, but chunk-servers read/write/copy files. (Master never moves chunk data around, just initiates and monitors actions!) Checksumming for bit corruption detection. Checksum is generated for every 64KB (i.e. 1000 checksums per chunk) and maintained by chunk-server.

Step 3 (Master information). A single **master server**, stores the IP addresses of the chunk-servers storing a chunk. Every chunk is indexed/referenced by a **64-bit chunk-handle**. 1GB is 16 chunks, 1PB is 16million chunks or so. Each chunk is replicated 3 times, and uses metadata of approximately size 64B (< 100B). So 1PB requires $16M \times 64B = 1GB$. All info in main memory and copied to shadow servers. Every chunk is copied to replica nodes (3x). A chunk-lease (60sec) maintained by master to determine for how long chunk will be active and also version number assigned to chunk. Primary chunk-server takes over replication/copy. At end of period, chunk is revoked or gets renewed (heartbeat messages); if no response from replica, chunk-lease expires, and a new copy is replicated to other servers. Garbage collection initiated periodically. After a reboot all information retrieved and rebuilt from chunk-servers. Master stores three types of metadata.

- File and chunk namespaces,
- the mapping from files to chunks,
- the locations of each replica of a chunk (usually 3).

Replica information not persistent. At master reboot this information is generated from scratch through control probes to chunkservers.

(c) Copyright A. Gerbessiotis. CS 345 : Fall 2015 . All rights reserved.

Read, Write, Communication

```
1. Client/Application generates from ordinary file position a ChunkIndex and sends it to Master
   (FileName, PositionInFile)  --->  (FileName,  ChunkIndex)            by Application
              [ ChunkIndex = PositionInFile/ 64MB ]                     to Master

2. Master's reply with chunk-handle, chunk-versionnumber and location of replicas

   (FileName,  ChunkIndex)     --->  (ChunkHandle, ChunkLocation)       by Master
                                                                        to Application

3. Client determines nearest and recent chunk version and chunk-server
   and communicates with it
   (ChunkHandle, ByteRange)    --->  (ChunkServer,ChunkHandle,ByteRange) by Application
                                                                        to Chunk-server

4. Chunkserver replies with data back to client
   (Chunkserver, ....,ByteRange)===>  (Application, Chunk Data)         by Chunk-server
                                                                        to Application
5. Client sends data to primary chunk-server that then daiychain
   copies to 2nd, 3rd replica locations. Secondaries reply to primary
    and only primary replies to client.
    Copy from Client to ChunkServers                                    by Application
                                                                        to PrimaryChunkserver
                                                                        to SecondaryChunkserver
-------------------------------------------------------------------------------------------------
6.  Master sends/receives Hearbeat messages to update leases
    ControlMessage             <--->                                    by Master
                                                                        to Chunkserver

    ChunkserverState            --->                                    by Chunkserver
                                                                        to Master
```

Master and Chunkserver recovery

Master maintains a log of all activity in a disk file with periodic check-points This is also periodically copied to shadow servers. Master also rebalances chunks among chunservers and with that thus probes the health of the cluster (eg unavailability of chunkservers, dead machines, dead disks). Garbage collection of (stale) replica copies by the master. When the master reboots it uses this log to recover disk-related activity that was initiated between last check-point and reboot but also to retrieve all mapping between (user) files and namespaces and chunk mappings (but not location). It then initiates communication with the list of chunkservers to determine replica information. Chunk-leases and version numbers of every chunk, determine the state of a chunk and chunk-server. Out-of-date version numbered chunks are garbage collected by the master (and actually disposed by the chunkserver).

- Replays log from disk to recover namespace (directory) information, and file to chunkhandle mapping,
- recovers state information from chunkservers (replica information), and
- reconciles version numbers to determine stale copies; initiates copy activity.

If a chunk-server does not respond all its chunks are assumed lost. A copy is initiated with preference to those chunks that have the smaller number of replicas active. If multiple replicas have different version numbers the most recent version number chunk is being used in copy operations. A new primary chunkserver may be selected as well, before the copy starts. When the chunkserver fails

- master realizes that heartbeats are missing, and
- master replicates missing replicas with priority assigned to chunks missing most number of replicas; a new primary master may be assigned.

(c) Copyright A. Gerbessiotis. CS 345 : Fall 2015 . All rights reserved.

Filename namespace

GFS does not use a data structure that indicate/show/describe the files/directories under a given directory, ie. a Unix-like structure is not being used. Because of this no aliases are allowed: no hard or symbolic links are being used. Its namespace is a lookup table mapping pathnames to metadata. It uses prefix compression and stores it into main memory. This is allowed as the number of filenames is usually small (millions) though each one can have size in the 100sMB to GBs.

Each master operation locks the file by locking the directories leading to that file.

```
  /d1/d2/d3/filename
acquires read locks for
  /d1
  /d1/d2
  /d1/d2/d3
and read or write lock for
  /d1/d2/d3/filename
as needed.
```

Quick Overview

It is a variant and simplification of the Google File System. Here is a quick overview of it.

- Master is named **NameNode**. Opening, closing, renaming of files/directories by NameNode. One machine runs NameNode software. NameNode assigns replicas to "chunk-servers" using RackAwareness, i.e. to distribute files so that it improves reliability and throughput.

- Chunkservers are **DataNodes**, and chunks are known as **blocks**. Chunkservers communicate with Heartbeats with NameNode and periodically send BlockReport (i.e. chunk maintained by DataNode).

- Does not support multi-writers per file (and no record append). Every other machine (other than NameNode) runs a copy of DataNode software.

Motivation

Its development began in 2004 and is used for indexing, by MapReduce to generate data stored in BigTable, Gmail, etc. The BigTables uses the Google File System for storage.

IT IS NOT an RDBMS. A BigTable is a map that maps two arbitrary string values, (a) a row key (think of URLs such as "cs.njit.edu"), and (b) a column key (think of attributes such as "contents:', 'anchor:www.njit.edu'), and (c) a timestamp (i.e. it is a three-dimensional mapping) into an arbitrary size byte/char array. It scale into PBs and it is easy to add mahines, and start/restart it with minimum reconfiguration. The timestamp allows for versioning and also to assist in cleanup (eg garbage collection).

A collection of rows is also known as a **tablet** and should be roughly 200MB in size i.e. roughly 4 GFS chunks (with some space available for extending it). A tablet may be split into tablets along a row, or can be resized by compressing it (with Zippy or open-sourced Snappy which is LZ77-based).

Around 2006, a webcrawl of 2 billions pages generated 45-50TB of uncompressed pages, that after compression resulted into a 4.5TB size (10% of original). Links and anchors generate compression savings of 85%, slightly worse.

If a tablet is receiving many user queries it can migrate to another less busy machine. If a machine that maintains x tablets goes down, x other machines can host each one one copy of the x tablets going down. The migration cost is minimal.

Location information of a tablet is available in special **META** tablets.

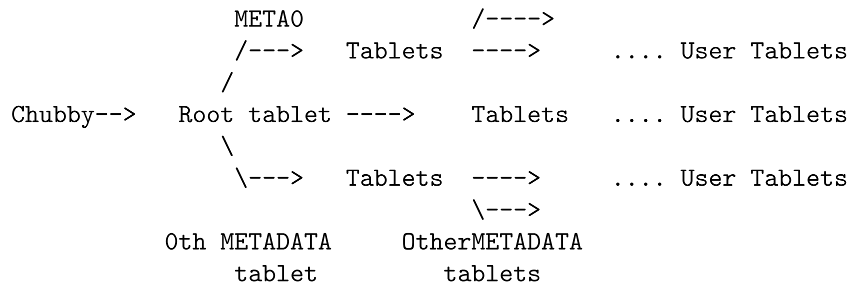
There is one **META0** tablet pointing to multiple **META1** tablets. A client (user) gets a pointer (information) to the only **META0** tablet (think of root node of a tree, in fact a B-tree-like structure). This tablet keeps track of **META1** tablets, and it is a META1 tablet that records information about tablets in general. META-tablets are cached. There are 100-1000 tablets per machine of a cluster.

(c) Copyright A. Gerbessiotis. CS 345 : Fall 2015 . All rights reserved.

(a) Row key. About 10-100B strings. Lexicographic order of row keys. Pages in the same domain are close to each other. Row key for
`http://www.njit.edu/alexg/courses/cs345/index.html` becomes
`edu.njit.www/alexg/courses/cs345/index.html`
(b) Column family. It's a combination of a family name/field and a qualifier such as `family:qualifier`. Thus a "contents:' family points to the contents of the row key URL, and 'anchor:www.njit.edu' points to the anchor text of the URL `www.njit.edu` that points to row-key URL, or "language:" to the language used.
(c) Timestamps. They are 64-bit integers. They can represent time in microseconds. Most recent version is read first.

All tablets of a single machine share a log file. All tablets are built on top of GFS (i.e. think triple replication). When a machine goes down, META1 entries indicate the tablets that need to be "replicated", and along the log-file which is a tablet itself, the maintenance takes place and gets completed.

A three-level hierarchy of tablets storing information about other tablets allows for a modest 128MB METADATA tablet and approximately 1KB of METADATA row size. Thus $128K$ entries per tablet (2^{17}) allows for a two-level hierarchy $2^{17} \times 2^{17}$ tablets of 128MB each or roughly $2^{17+17+27} = 2^{61}$B.



Note that a higher level entity Chubby that is a persistent distributed lock service, points to the root tablet. Chubby is five-replicated and active if at least three of its replicas are active and communicating with each other; one copy becomes the master. It uses the Paxos algorithm to maintain consistency of the replicas. In Chubby BigTable Schema information is maintained (eg column information for every tablet).

(c) Copyright A. Gerbessiotis. CS 345 : Fall 2015 . All rights reserved.