BigTable

Implementation details

Implementation

- BigTable comprises
  - client library
    - linked with users code, gives the user an interface to interact with BigTable
  - master server
    - activities coordinator
  - many slave servers
    - called **tablet servers**
    - store the tablets
    - can be added / removed dynamically

Implementation

- Master server
  - assignes tablets to tablet servers
  - takes care of the load balancing
  - manages schema changes of tables (column families and columns)
- Tablet Server
  - each server manages a set of tablets (usually 10 - 1,000)
  - handle read/write of tablets, it manages
    - client does not go throw the master, communicate directly with tablet server holding the data (using **SSTable**, (BigTable internal file format))
  - takes care of splitting a tablet when it gets large (based on how it is configured)

Implementation: Supporting Services

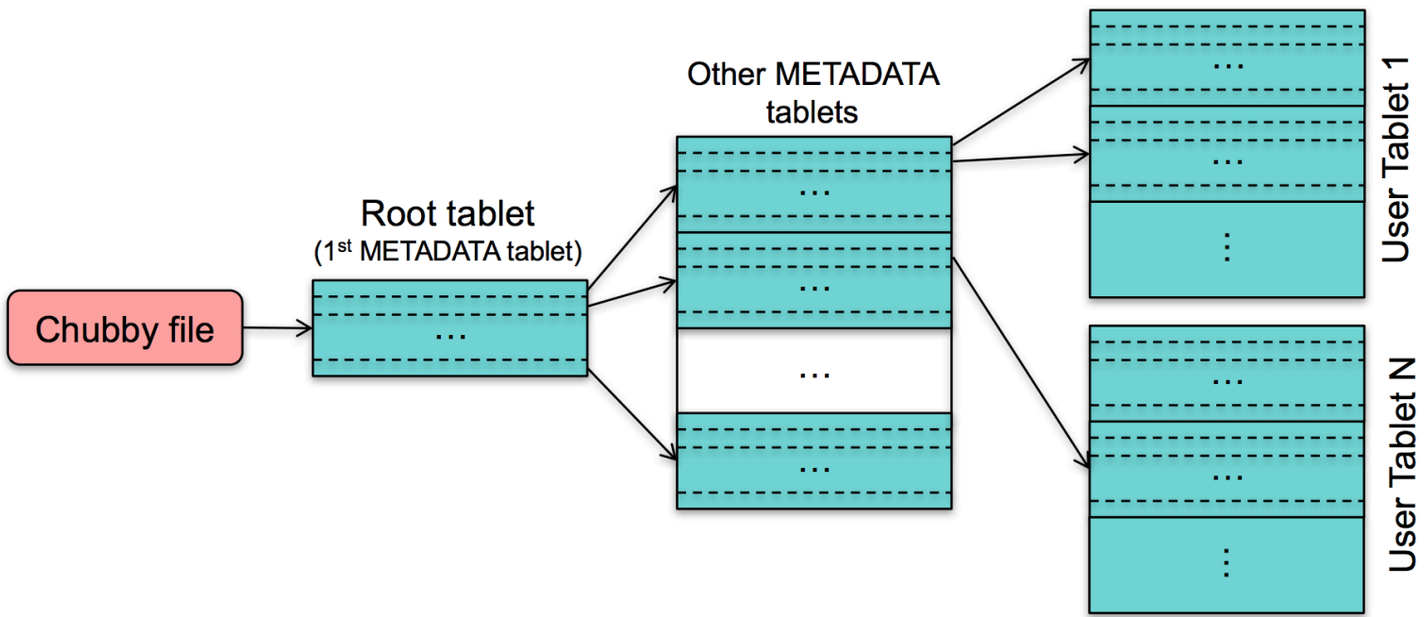
- GFS (Google File System)
  - for storing data files
  - Google published a research paper 2003
- Cluster Management System
  - for job scheduling, failure handling, system health monitoring
- Google **SSTable** (Sorted String Table)
  - internal file format for storing key/values
  - optimized for I/O operations
  - a persistent, immutable, ordered map of key/values
  - memory or disk based, but indexes are loaded into memory

Implementation: Supporting Services

- Chubby
  - lock-service for coordination in distributed system
  - one common use of this service is for electing the master
    - the first one getting the lock becomes the master
  - holds the name space of directories and files
    - the client create a session with the tablet server containing the data during the read/write operations
      - if the client didn’t manage to renew the session before the lease expiration time
      - then the client looses the lock
- used by BigTable
  - ensure that there is one active master
  - store BigTable Schema
  - discover tablet servers
- Zookeeper is an open-source implementation of Chubby

Implementation: Tablet Location

- Three-level hierarchy
  - level1: Chubby file contains location of the root tablet
  - level2: root tablet contains location of the METADATA tablets
  - level3: METADATA tablet contains locations of user tablets
    - (key - > location, where the key is the (tableId, rowKey)



Implementation: Tablet Assignment

- Tablet assigned to one tablet server at a time
- Master keeps track of
  - live tablet servers using chubby service
  - the current assignment of tablets to tablet servers
  - the current unassigned tablets
    - when the tablet is unassigned
      - the server assigned it to an available tablet server by sending tablet load request to that server

Tablet Serving

- updates are committed to a commit log
- Recent commits are stored in memory - MEMtable
- Old commits are stored on disk - SSTable

Tablet Serving

- Write operation

    1. server checks if the request is well-formed
    2. server checks if the sender is authorized to write
    3. valid operation is written into commit log which also store redo records
    4. after the commit, the data is inserted into MEMtable

Tablet Serving

- Read operations
  - server checks if the request is well-formed
  - server check if the sender is authorized
  - valid operation is executed on a merged view of MEMtable & SSTable

Tablet Serving

- Tablet recovery
  - tablet server reads its data from the METADATA table which contains a list of all SSTables and pointers into any commit log that might contain data from that tablet

Compaction

- In order to control size of MEMtable , SSTable, and commit log, compaction is needed
    - minor compaction
        - move data from MEMtable to SSTable
    - merging compaction
        - merging multiple SSTables and MEMtables into one SSTable
    - major compaction
        - rewrite all SSTables into one SSTable

Compaction

- Minor compaction
    - when MEMtable size reaches a threshold
    - MEMtable is frozen
    - a new MEMtable is created
    - the frozen MEMtable is converted into SSTable and written to GFS

Compaction

- Merging compaction
  - problem with minor compaction is that every minor compaction result into new SSTable (arbitrary number of SSTables)

  - solution: periodic compaction of SSTables & the MEMtable

Compaction

- Major compaction
  - rewrite all SSTables into one SSTable
    - remove all log pointers

Sample Application

- Google Analytics
  - Raw Click Table (~200 TB)
    - row for each user session
      - row key: website name + time of the session
        - all sessions related to the same website will be stored next to each other
- Summary Table
  - information about each crawled website (included in Google index)
  - this information is generated from the Raw Click Table using batch MapReduce jobs

Sample Application

- Personalized Search
  - one row per unique user
  - column family per each type of actions
    - for example, search queries (search history)
    - clicked/viewed URLs
    - liked, rated URLs
- timestamp is explicitly identified based on the action time
  - for example, the time when the user issued the query
- show result personalized based on the past search history

References

- Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, _Bigtable: A Distributed Storage System for Structured Data_, Google, Inc. OSDI 2006

- Robin Harris, _Google's Bigtable Distributed Storage System_, StorageMojo.com

- _Understanding HBase and Bigtable_, Jumoojw.com