Data Management in Large-Scale Distributed Systems

Google BigTable

Thomas Ropars

thomas.ropars@univ-grenoble-alpes.fr

http://tropars.github.io/

2021

References

- The lecture notes of Prof. E. Brewer
- Designing Data-Intensive Applications by Martin Kleppmann
  - Chapters 2 and 7

In this lecture

- The design of Google BigTable
- Column Family Databases
    - Scalability, locality
- Solutions to efficiently store and retrieve data
    - SSTable (Sorted-String Table)
    - LSM Tree

Agenda

Google BigTable

- Column family data store
  - Data storage system used by many Google services: Youtube, Google maps, Gmail, etc.
- Paper published by Google in 2006 (F. Chang et al)
  - Now available as a service on Google Cloud
- Many ideas reused in other NoSQL databases

Motivations

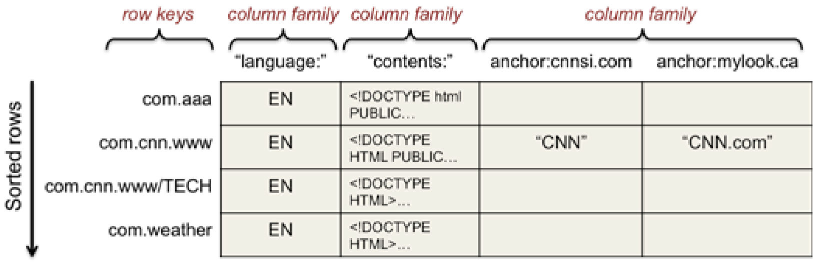
- A system that can store very large amount of data
  - TB or PB of data
  - A very large number of entries
  - Small entries (each entry is an array of bytes)
- A simple data model
  - Key-value pairs (A key identifies a row)
  - Sparse data
  - Multi-dimensional data: Data are associated with timestamps
- Works at very large scale
  - Thousands of machines
  - Millions of users

Agenda

About the data model

- Rows are identified by keys (arbitrary strings)
  - Modifications on one row are atomic
  - Rows are maintained in lexicographic order
- Columns are grouped in columns families
  - Columns can be sparse
  - Clients can ask to retrieve a column family for one row
- Each cell can contain multiple versions indexed by a timestamp
  - Assigned by BigTable or by the client
  - Most recent versions are accessed first
  - GC politics:
    - Keep last n versions
    - Keep all new-enough versions

About the data model: An example

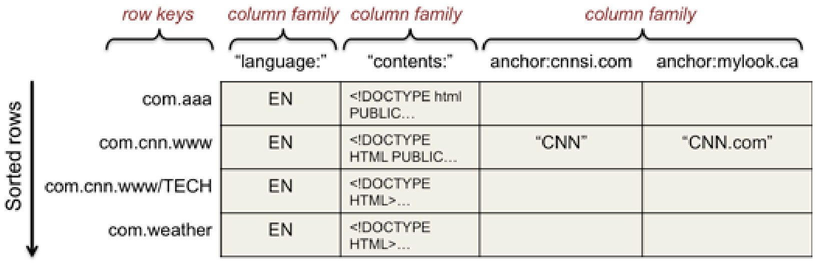


row keys	column family	column family	column family	
	"language:"	"contents:"	anchor:cnnsi.com	anchor:mylook.ca
com.aaa	EN	<!DOCTYPE html PUBLIC...		
com.cnn.www	EN	<!DOCTYPE HTML PUBLIC...	"CNN"	"CNN.com"
com.cnn.www/TECH	EN	<!DOCTYPE HTML>...		
com.weather	EN	<!DOCTYPE HTML>...		

Sorted rows

Figure: Example of Table: the *WebTable*

About the data model: An example


row keys
column family
column family
column family
Sorted rows

	"language:"	"contents:"	anchor:cnnsi.com	anchor:mylook.ca
com.aaa	EN	<!DOCTYPE html PUBLIC...		
com.cnn.www	EN	<!DOCTYPE HTML PUBLIC...	"CNN"	"CNN.com"
com.cnn.www/TECH	EN	<!DOCTYPE HTML>...		
com.weather	EN	<!DOCTYPE HTML>...		

Figure: Example of Table: the *WebTable*

- The row name is a reverse URL
- Multiple column families:
  - The column family *contents* can contain multiple version with timestamps (corresponding to when the content was fetched)
  - The column family *anchor* is sparse
    - For each existing web page (column), store the text of the anchor that points to the considered web page (row)

Partitioning and performance

see `https://cloud.google.com/bigtable/docs/schema-design`

Partitioning

- Partitioning on the rows
- Each partition includes a range of keys

Recommendations about the schema for performance

- Accesses can be made based on key, key-prefix or key-range
  - Choose keys appropriately to access few partitions on read
  - Example: Reverse domain name, User_id prefix, etc.
  - To avoid:
    - Domain name, hash values (access to plenty of partitions)
    - timestamps (hotspot on write)
  - Take advantage of the concept of key prefix
- Group related columns in a column family
  - Avoids retrieving all data from a single row when not needed
- Creating plenty of tables is not a good pattern
  - Use column families instead

Agenda

About data storage

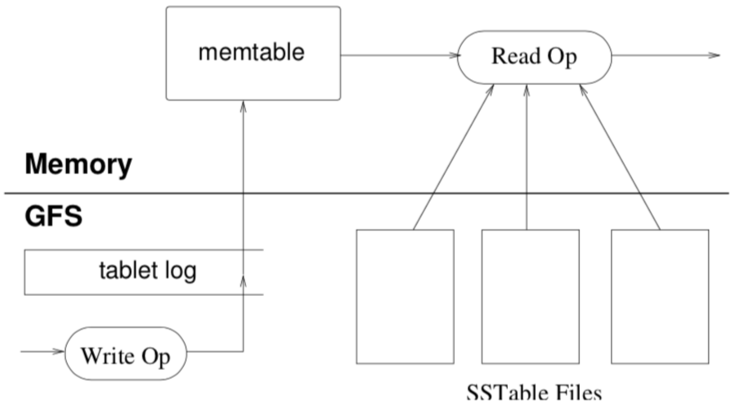
Main objectives and challenges

- Fast read and write operations
- Data persistence
- (Limit storage space consumption)

Tablets

- A partition of the table
  - Adjacent rows are stored in the same tablet
- Stored using SSTables
  - A persistent, ordered, immutable map
  - GFS is used as an underlying persistent file system
- Construction of an LSM (Log-Structured Merge) tree

Implementation of tablets



- Tablet log: Redo log of write operations
- Memtable: Sorted map of the most recent updates
- SSTables: Copy of previous memtables saved on Disk

SSTables

Sorted-String Table

Main principles

- File storing a sequence of key-value pairs (no padding)
  - Efficient sequential read/write operations

- An index can be created to know the offset of a key

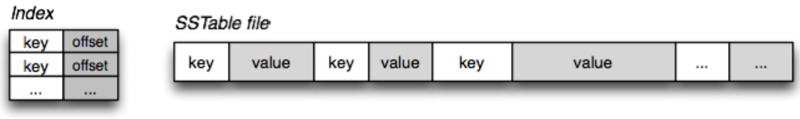


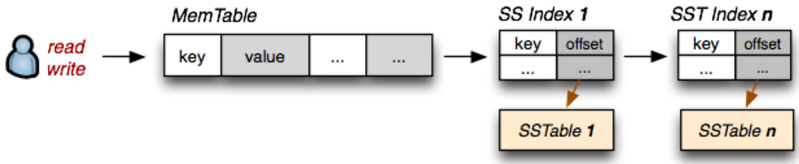
Figure: Figure by I. Grigorik

A Log-Structure Merge Tree

Figure by I. Grigorik

A multi-level data structure

- One fast level for new operations
  - In memory
  - The Memtable
- Other (slower) levels on disk
  - To save space in memory
  - Creation of a new level when the memory is full
    - Simple and efficient operation that flushes all content from memory to disk
    - Creation of an SSTable
    - Data on disk are immutable



Operation on Tablets

Write operations

- Data stored in memory (Memtable)
- Any update is written to a commit log on GFS for persistence
  - The log is shared between all hosted tablets

Periodic writes to disk

- When the Memtable becomes too big:
  - Copied as a new SSTable to GFS
    - Multiple SSTables are created if locality groups are defined (based on column families)
  - Reduces the memory footprint and reduces the amount of work to do during recovery
  - SSTables are immutable (no problem of concurrency control)
- Operation called minor compaction

Implementation of tablets

Read operations

- The state of the tablet = the Memtable + all SSTables
  - A merged view needs to be created
    - The Memtable is accessed first, then the SSTables from most recent to oldest
  - The Memtable and the SSTables may contain delete operations
- Locality groups help improving the performance of read operations

Major compaction

- When the number of SSTables becomes too big, merge them into a single SSTable
  - Allow reclaiming resources for deleted data
  - Improve the performance of read operations

Improving the performance of read operations

- During a read operation, potentially several SSTables need to be read
  - How to avoid reading all SSTables when not needed?

An in-memory index

- Store information about the SSTables that contain information about a row
- Using a hash table
  - Constant time access
  - Potentially large memory footprint
- Using Bloom filters
  - Probabilistic data structure
  - Small memory footprint

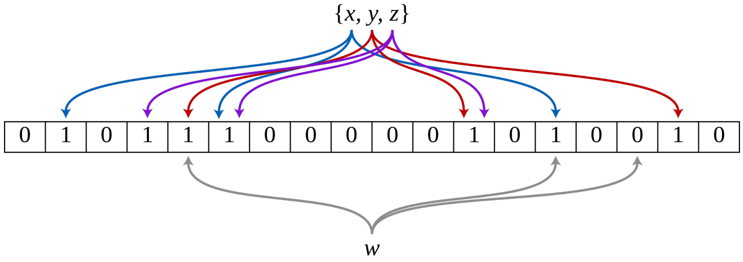
Improving the performance of read operations

Bloom filter

- Implements a membership function (is X in the set?)
- If the bloom filter answers no: it is guaranteed that X is not present
- If the bloom filter answers yes: the element is in the set with a high probability
- Good trade-off between accuracy and memory footprint

About bloom filters

- A vector of n bits and k hash functions
- On insert:
  - Compute the k hash values
  - Set the corresponding bits to 1 in the vector
- On lookup:
  - Compute the k hash values
  - Test whether all bits are set to 1



About the logs

On one node, a single commit log is created even if it hosts multiple tablets.

Advantages

Drawbacks

About the logs

On one node, a single commit log is created even if it hosts multiple tablets.

Advantages

- Write a single append-only file on disk
  - Improves performance by avoiding long seeks

Drawbacks

- Recovery is more complex since the log includes data associated with different tablets
- The tablets might be distributed over multiple nodes

Agenda

Building blocks of the BigTable infrastructure

A master

- Assign tablets to severs
- With the help of a locking service

Tablet servers

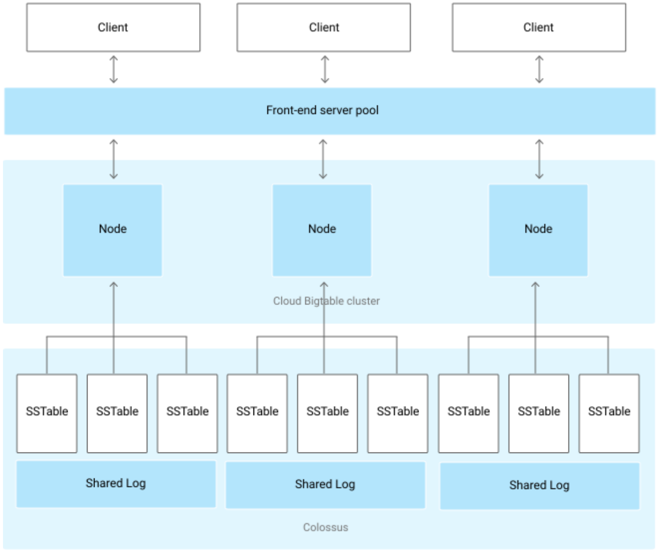
- Manage the tables (divided in tablets)
- Process client requests

Tablets

- Stores ranges of rows
- Stored as SSTables
- Stored in the Google File System for persistence

BigTable infrastructure

Image from `https://cloud.google.com/bigtable/docs/overview`



Additional references

Mandatory reading

- *Bigtable: A Distributed Storage System for Structured Data.*, F. Chang et al., OSDI, 2006.

Suggested reading

- `https://www.igvita.com/2012/02/06/sstable-and-log-structured-storage-leveldb/`, I. Grigorik, 2012.