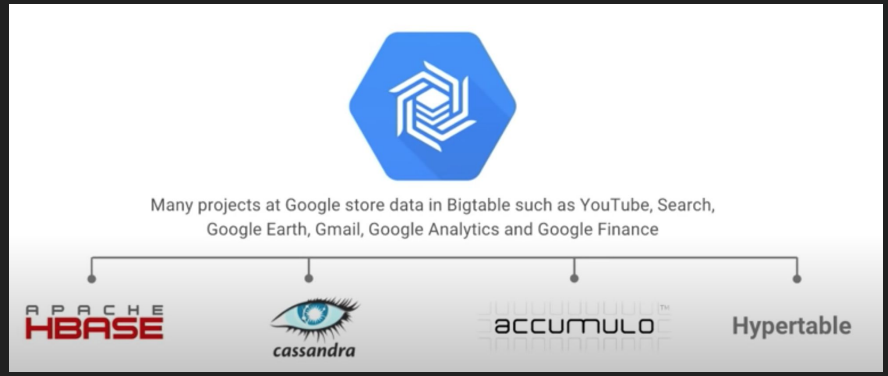
Google Bigtable

A Distributed Storage System for Structured Data

Paper: Chang et. al
Slides: Ryan W. West
ryan@ryanwwest.com
September 2021

What is Bigtable?

- Bigtable is a distributed storage system optimized for **size** and **price**:
  - Can scale to **petabytes** of data with fast speeds
  - Can run on cheap commodity hardware (1GB RAM servers in 2006)
- Widely used in 2021
  - Used by Google Search, Maps, Analytics, Finance, and more
  - Closed-source but available for public use through Google Cloud Platform (GCP)
    - Used by Twitter, PayPal, UPS…
- NoSQL Wide-column DB family



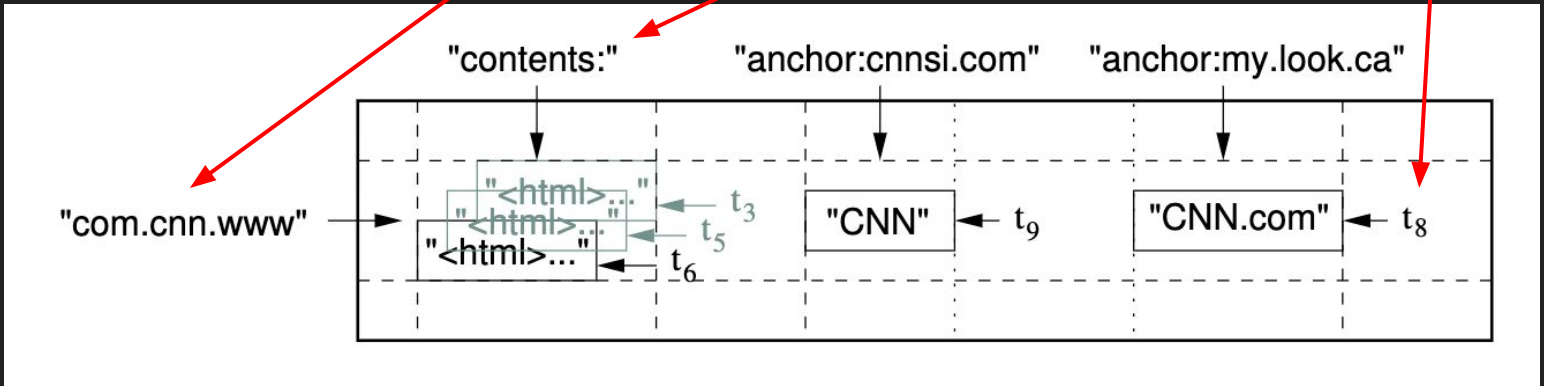
Main Features

- Simple Data Model — NoSQL-like relational querying
- Dynamic control over data layout and format
- Can be configured for high throughput or low latency
- Live Cluster resizing, automatic replication
- Data is automatically compressed, can configure the algorithm

Data Model

“A Bigtable is a sparse, distributed, persistent multidimensional sorted map.”

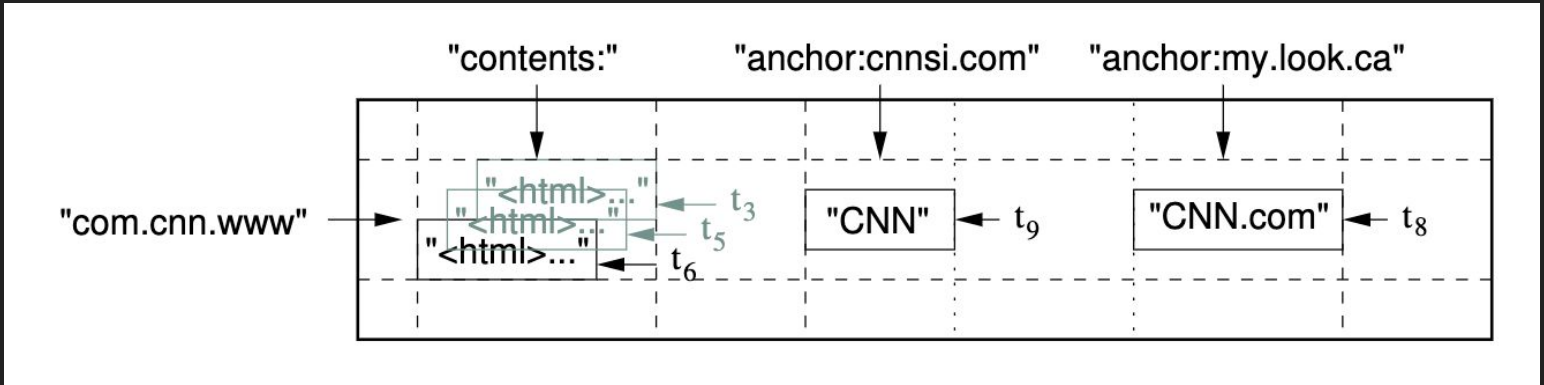
- Map data is indexed with **Row Keys**, **Column Keys**, and **Timestamps**



- Multiple similar columns can be grouped into a *column family* for faster compression/operations

Data Model

- Data is atomically read/written in full rows with the row key (row size <100MB)
- Rows are sorted lexicographically
  - Allows users to exploit locality for better read performance by carefully ordering their row-keys
  - Google's webpage table stores URLs in reverse so subdomains (money.cnn.com) are together



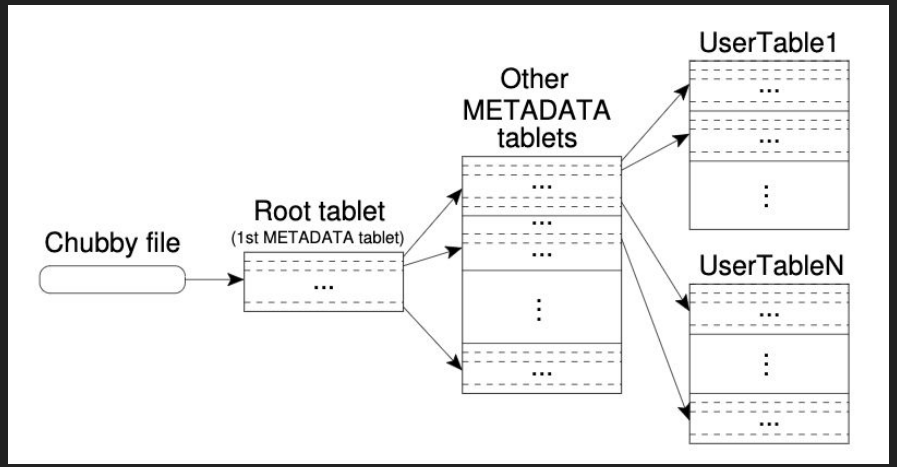
Architecture

- A range of rows is a *tablet* - the unit of distribution and load-balancing
  - A set of tablets (each 100-200 MB by default) makes a *table*
- Three major components make up a Bigtable cluster:

Master server	Tablet servers	Client libraries
● Assigns *tablets* to *tablet servers* ● Balances tablet-server load ● Garbage-collects old tablets ● Schema changes (to column-families) ● Only one, low load	● Manages a set of tablets (~10-1000) ● Handles read/write requests to its tablets ● Splits tablets that grow too large ● Dynamically added / removed from cluster	● An API is provided to manage the cluster, read/write data, etc. ● Done not through master but directly through tablet servers ● Some clients are open-sourced online

Architecture

- Bigtable relies on Chubby, a highly-available distributed lock service
- Uses GFS (Google File System) underneath to store & replicate data files
- Data addressing structure:
  - Chubby file contains root tablet's location
  - Root tablet contains locations of all other metadata tablets
  - Metadata tablets contain location of all other tablets (in tables)

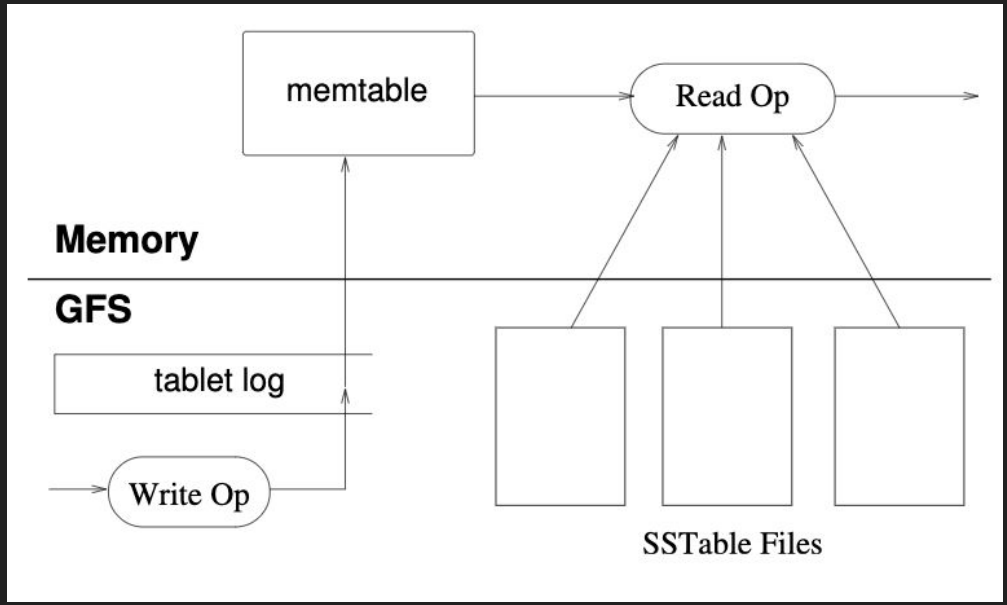


Tablet Assignment

- A tablet is assigned to one tablet server at a time through an exclusive Chubby lock
- Master server detects unassigned tablets and tablet servers that no longer serve their tablets, and quickly moves such tablets to new tablet servers
- Master continually communicates with tablet servers to ensure metadata (addressing) tablets are up-to-date

Tablet Serving

- Updates are stored in a tablet commit log on GFS (Google File System) then committed to an in-memory memtable
- If memtable grows too large, it is frozen, compacted, and converted to an SSTable on disk
- A new memtable is created
- SSTables allow for quick transfer and recovery of tablets



Refinements

- Locality groups - groups of column families, allows for more efficient reads (a user who wants webpage metadata need not fetch page contents)
- Compression - SSTables can be compressed for space or not for speed
- Caching of SSTable key-value pairs & entire SSTable blocks read from GFS
- Bloom filters - allows tablet server to ask if each SSTable *might* contain data for a certain row/column pair.
    - Allows certain read processing to skip scanning large amounts of SSTables of a tablet

Performance Benchmarks

Experiment	# of Tablet Servers			
	1	50	250	500
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843

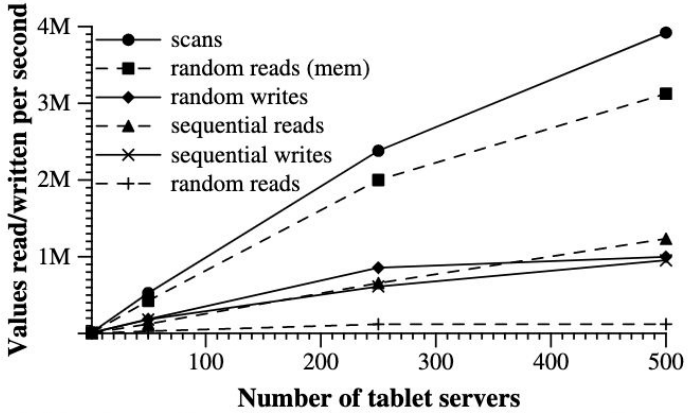


Figure 6: Number of 1000-byte values read/written per second. The table shows the rate per tablet server; the graph shows the aggregate rate.

Real Cluster Examples (in 2006)

Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
Crawl	800	11%	1000	16	8	0%	No
Crawl	50	33%	200	2	2	0%	No
Google Analytics	20	29%	10	1	1	0%	Yes
Google Analytics	200	14%	80	1	1	0%	Yes
Google Base	2	31%	10	29	3	15%	Yes
Google Earth	0.5	64%	8	7	2	33%	Yes
Google Earth	70	–	9	8	3	0%	No
Orkut	9	–	0.9	8	5	1%	Yes
Personalized Search	4	47%	6	93	11	5%	Yes

Table 2: Characteristics of a few tables in production use. *Table size* (measured before compression) and *# Cells* indicate approximate sizes. *Compression ratio* is not given for tables that have compression disabled.

My Thoughts

- Shows that many different data models work
  - A simple model can work well; no SQL querying can work well
- Few databases can scale to petabytes in a single cluster
- Wish it were open-sourced
  - Could have third-party security audits this way
  - But might not be possible with dependency on GFS and Chubby (also closed-source)