

Distributed Systems

18. Bigtable

Paul Krzyzanowski
Rutgers University
Fall 2017

November 13, 2017 © 2014-2017 Paul Krzyzanowski 1

Bigtable

- Highly available distributed storage
- Built with semi-structured data in mind
 - URLs: content, metadata, links, anchors, page rank
 - User data: preferences, account info, recent queries
 - Geography: roads, satellite images, points of interest, annotations
- Large scale
 - Petabytes of data across thousands of servers
 - Billions of URLs with many versions per page
 - Hundreds of millions of users
 - Thousands of queries per second
 - 100TB+ satellite image data

November 13, 2017 © 2014-2017 Paul Krzyzanowski 2

Uses

At Google, used for:

- Google Analytics
- Google Finance
- Personalized search
- Blogger.com
- Google Code hosting
- YouTube
- Gmail
- Google Earth & Google Maps
- Dozens of others...

November 13, 2017 © 2014-2017 Paul Krzyzanowski 3

A big table

- Bigtable is NOT a relational database
- Bigtable appears as a large table
 - “A Bigtable is a sparse, distributed, persistent multidimensional sorted map”*

Web table example

*Bigtable: OSDI 2005

November 13, 2017 © 2014-2017 Paul Krzyzanowski 4

Table Model

- (row, column, timestamp) → cell contents
 - Contents are arbitrary strings (arrays of bytes)

Web table example

November 13, 2017 © 2014-2017 Paul Krzyzanowski 5

Tablets: Pieces of a Table

- Row operations are atomic
- Table partitioned dynamically by rows into tablets
- Tablet = range of contiguous rows
 - Unit of distribution and load balancing
 - Nearby rows will usually be served by the same server
 - Accessing nearby rows requires communication with a small # of machines
 - You need to select row keys to ensure good locality
 - E.g., reverse domain names:
com.cnn.www instead of www.cnn.com

November 13, 2017 © 2014-2017 Paul Krzyzanowski 6

Table splitting

- A table starts as one tablet
- As it grows, it is split into multiple tablets
 - Approximate size: 100-200 MB per tablet by default

	"language:"	"contents:"		
com.aaa	EN	<DOCTYPE html PUBLIC...		
com.cnn.www	EN	<DOCTYPE HTML PUBLIC...		
com.cnn.www/TECH	EN	<DOCTYPE HTML>...		
com.weather	EN	<DOCTYPE HTML>...		

tablet

November 13, 2017 © 2014-2017 Paul Krzyzanowski 7

Splitting a tablet

	"language:"	"contents:"		
com.aaa	EN	<DOCTYPE html PUBLIC...		
com.cnn.www	EN	<DOCTYPE HTML PUBLIC...		
com.cnn.www/TECH	EN	<DOCTYPE HTML>...		

Split

	"language:"	"contents:"		
com.weather	EN	<DOCTYPE HTML>...		
com.wikipedia	EN	<DOCTYPE HTML>...		
com.zcorp	EN	<DOCTYPE HTML>...		
com.zoom	EN	<DOCTYPE HTML>...		

November 13, 2017 © 2014-2017 Paul Krzyzanowski 8

Columns and Column Families

- Column Family**
 - Group of column keys
 - Column family is the basic unit of data access
 - Data in a column family is typically of the same type
 - Implementation compresses data in the same column family
- Operations**
 - (1) Create column family
 - (2) Store data in any key within the family
- Column families will typically be small
 - ≤ hundreds of keys; a table may have an unlimited # of column families
- Identified by **family:qualifier**

November 13, 2017 © 2014-2017 Paul Krzyzanowski 9

Column Families: example

Three column families

- "language:" – language for the web page
- "contents:" – contents of the web page
- "anchor:" – contains text of anchors that reference this page.
 - www.cnn.com is referenced by Sports Illustrated (cnnsi.com) and My-Look (mlook.ca)
 - The value of ("com.cnn.www", "anchor:cnnsi.com") is "CNN", the reference text from cnnsi.com.

	"language:"	"contents:"	Column family <i>anchor</i>	
			anchor:cnnsi.com	anchor:mlook.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

November 13, 2017 © 2014-2017 Paul Krzyzanowski 10

Timestamps

- Each column family may contain multiple versions
- Version indexed by a 64-bit timestamp
 - Real time or assigned by client
- Per-column-family settings for garbage collection
 - Keep only latest *n* versions
 - Or keep only versions written since time *t*
- Retrieve most recent version if no version specified
 - If specified, return version where timestamp ≤ requested time

November 13, 2017 © 2014-2017 Paul Krzyzanowski 11

API: Operations on Bigtable

- Create/delete** tables & column families
- Change** cluster, table, and column family metadata (e.g., access control rights)
- Write** or **delete** values in cells
- Read** values from specific rows
- Iterate over a subset of data in a table**
 - All members of a column family
 - Multiple column families
 - E.g., regular expressions, such as anchor:*.cnn.com
 - Multiple timestamps
 - Multiple rows
- Atomic read-modify-write** row operations
- Allow clients to execute scripts (written in Sawzall) for processing data on the servers

November 13, 2017 © 2014-2017 Paul Krzyzanowski 12

Implementation: Supporting Services

- **GFS**
 - For storing log and data files
- **Cluster management system**
 - For scheduling jobs, monitoring health, dealing with failures
- **Google SSTable** (Sorted String Table)
 - Internal file format optimized for streaming I/O and storing <key,value> data
 - Provides a persistent, ordered, *immutable* map from keys to values
 - Append-only
 - Memory or disk based; indexes are cached in memory
 - If there are additions/deletions/changes to rows
 - New SSTables are written out with the deleted data removed
 - Periodic compaction merges SSTables and removes old retired ones

See <http://goo.gl/McD6ex> for a description of SSTable

November 13, 2017 © 2014-2017 Paul Krzyzanowski 13

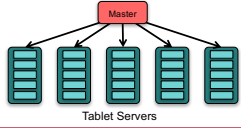
Implementation: Supporting Services

- **Chubby**
 - Highly-available & persistent distributed lock (lease) service & file system
 - Five active replicas; one elected as master to serve requests
 - Majority must be running
 - Paxos algorithm used to elect master & keep replicas consistent
 - Provides namespace of files & directories
 - Each file or directory can be used as a lock
- **In Bigtable, Chubby is used to:**
 - Ensure there is only one active master
 - Store bootstrap location of Bigtable data
 - Discover tablet servers
 - Store Bigtable schema information
 - Store access control lists

November 13, 2017 © 2014-2017 Paul Krzyzanowski 14

Implementation

1. **Many tablet servers – coordinate requests to tablets**
 - Can be added or removed dynamically
 - Each manages a set of tablets (typically 10-1,000 tablets/server)
 - Handles read/write requests to tablets
 - Splits tablets when too large
2. **One master server**
 - Assigns tablets to tablet server
 - Balances tablet server load
 - Garbage collection of unneeded files in GFS
 - Schema changes (table & column family creation)
3. **Client library**

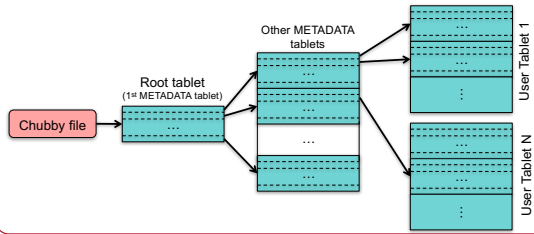


November 13, 2017 © 2014-2017 Paul Krzyzanowski 15

Implementation: METADATA table

Three-level hierarchy

- Balanced structure similar to a B+ tree
- Root tablet contains location of all tablets in a special METADATA table
- Row key of METADATA table contains location of each tablet $f(\text{table_ID, end_row}) \Rightarrow \text{location of tablet}$



November 13, 2017 © 2014-2017 Paul Krzyzanowski 16

Implementation

- Tablet assigned to one tablet server at a time
- Chubby keeps track of tablet servers
 - When tablet server starts:
 - It creates & acquires an exclusive lock on a uniquely-named file in a Chubby servers directory
 - Master monitors this directory to discover tablet servers
- When master starts:
 - Grabs a **unique master lock** in Chubby (prevent multiple masters)
 - Scans the **servers** directory in Chubby to find live tablet servers
 - Contacts **each tablet server** to discover what tablets are assigned to that server
 - Scans the METADATA table to learn the full set of tablets
 - Build a list of tablets not assigned to servers
 - These will be assigned by choosing a tablet server & sending it a **tablet load** request

November 13, 2017 © 2014-2017 Paul Krzyzanowski 17

Implementation

- Tablet assigned to one tablet server at a time
- When master starts:
 - Grabs a **unique master lock** in Chubby (prevent multiple masters)
 - Scans the **servers** directory in Chubby to find live tablet servers
 - Contacts **each tablet server** to discover what tablets are assigned to that server
 - Scans the METADATA table to learn the full set of tablets
 - Build a list of tablets not assigned to servers
 - These will be assigned by choosing a tablet server & sending it a **tablet load** request

November 13, 2017 © 2014-2017 Paul Krzyzanowski 18

Fault Tolerance

- Fault tolerance is provided by GFS & Chubby
- Dead tablet server
 - Master is responsible for detecting when a tablet server is not working
 - Asks tablet server for status of its lock
 - If the tablet server cannot be reached or has lost its lock
 - Master attempts to grab that server's lock
 - If it succeeds, then the tablet server is dead or cannot reach Chubby
 - Master moves tablets that were assigned to that server into an unassigned state
- Dead master
 - Master kills itself when its Chubby lease expires
 - Cluster management system detects a non-responding master
- Chubby: designed for fault tolerance (5-way replication)
- GFS: stores underlying data – designed for n -way replication

November 13, 2017

© 2014-2017 Paul Krzyzanowski

19

Bigtable Replication

- Each table can be configured for replication to multiple Bigtable clusters in different data centers
- Eventual consistency model

November 13, 2017

© 2014-2017 Paul Krzyzanowski

20

Sample applications

- Google Analytics
 - Raw Click Table (~200 TB)
 - Row for each end-user session
 - Row name: {website name and time of session}
 - Sessions that visit the same web site are sorted & contiguous
 - Summary Table (~20 TB)
 - Contains various summaries for each crawled website
 - Generated from the Raw Click table via periodic MapReduce jobs

November 13, 2017

© 2014-2017 Paul Krzyzanowski

21

Sample applications

- Personalized Search
 - One Bigtable row per user (unique user ID)
 - Column family per type of action
 - E.g., column family for web queries (your entire search history!)
 - Bigtable timestamp for each element identifies when the event occurred
 - Uses MapReduce over Bigtable to personalize live search results

November 13, 2017

© 2014-2017 Paul Krzyzanowski

22

Sample applications

- Google Maps / Google Earth
 - Preprocessing
 - Table for raw imagery (~70 TB)
 - Each row corresponds to a single geographic segment
 - Rows are named to ensure that adjacent segments are near each other
 - Column family: keep track of sources of data per segment (this is a large # of columns – one for each raw data image – but sparse)
 - MapReduce used to preprocess data
 - Serving
 - Table to index data stored in GFS
 - Small (~500 GB) but serves tens of thousands of queries with low latency

November 13, 2017

© 2014-2017 Paul Krzyzanowski

23

Bigtable outside of Google

Apache HBase



- Built on the Bigtable design
- Small differences (may disappear)
 - access control not enforced per column family
 - Millisecond vs. microsecond timestamps
 - No client script execution to process stored data
 - Built to use HDFS or any other file system
 - No support for memory mapped tablets
 - Improved fault tolerance with multiple masters on standby

November 13, 2017

© 2014-2017 Paul Krzyzanowski

24

Bigtable vs. Amazon Dynamo

- Dynamo targets apps that only need key/value access with a primary focus on high availability
 - key-value store versus column-store (column families and columns within them)
 - Bigtable: distributed DB built on GFS
 - Dynamo: distributed hash table
 - Updates are not rejected even during network partitions or server failures

November 13, 2017

© 2014-2017 Paul Krzyzanowski

25

The end

November 13, 2017

© 2014-2017 Paul Krzyzanowski

26