Distributed Systems
19. Bigtable

Paul Krzyzanowski
Rutgers University
Spring 2020

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

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... over sixty products

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”*

Table Model

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

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 - this is an admin task done when table is created
  - (2) Store data in any key within the family - this can be done anytime

- There will typically be a small number of column families
- Hundreds of column families
- A table may have an unlimited # of columns: often sparsely populated

- Identified by family:qualifier
### 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)

<table>
<thead>
<tr>
<th>Column family</th>
<th>anchor:cnnsi.com</th>
<th>anchor:mlook.ca</th>
</tr>
</thead>
<tbody>
<tr>
<td>“language”</td>
<td>EN “HTML PUBLIC.”</td>
<td>EN “HTML PUBLIC.”</td>
</tr>
<tr>
<td>“contents”</td>
<td>“CNN”</td>
<td>“CNN”</td>
</tr>
<tr>
<td>“anchor”</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Tables & Tablets

- 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

### Table splitting

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

### Splitting a tablet

### 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

### 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 anchors: *.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
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

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

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 table
  \(table_ID, end_row\) \(\rightarrow\) location of table

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 get that server’s lock
      - If it succeeds, then the tablet server is dead or cannot reach Chubby
      - Master moves tables 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
**Bigtable Replication**

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

**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 website are sorted & contiguous
  - Summary Table (~20 TB)
    - Contains various summaries for each crawled website
    - Generated from the Raw Click table via periodic MapReduce jobs

- **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

- **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

**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

**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
The end