Motivation

- Lots of (semi-)structured data, e.g.,
  - Web pages: URL, contents, anchor, metadata, page rank
  - Geography: country, city, satellite imagery
- Wide applicability: Google Analytics, Google Finance...
- Most commercial databases cannot handle this large scale
Motivation: a database?

BigTable resembles a database, however it has more flexibility.

- Control data layout and format dynamically
- Control locality of data
- May declare data to be in-memory
Data Model

A Bigtable is simply a big dictionary

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(row: string, column: string, timestamp: int64)</td>
<td>an uninterpreted array of bytes</td>
</tr>
</tbody>
</table>
Example

In the above example,

"com.cnn.www" - row key

"anchor" - column family

"anchor:cnnsi.com" and "anchor:my.look.ca" - column keys

t3, t5, t6, t8, t9 - timestamps
API

- Metadata & Configuration
  - Creating and deleting tables
  - Changing BigTable nodes for cluster with workload
  - Changing column family metadata, such as access control

- Read & Write
- Lookup
- Iterate over a subset of the data in a table
- More advanced operations
  - Atomic read-modify-write on single row
  - Client-supplied scripts
  - Working with MapReduce
Implementation: Architecture

Master

Chubby distributed lock service

Tablet server

Tablets

a tablet = a row range

GFS: stable storage for log and data files (SSTable file format)

SSTable

persistent, ordered, immutable map

SSTable

(key, value)

Log

(key, value)
More on SSTable

- persistent, ordered, immutable map
- data blocks + block index
  - Index is loaded into memory when SSTable is opened

• Operations:
  - Lookup: single seek, memory index -> block -> disk (GFS)
  - iteration
More on Chubby

- Consists of directories or files used as a lock
- A client’s session expires = loses the lock
- Tasks in Bigtable:
  - Ensure at most one master at a time
  - Store location of tablets
  - Discover live tablet servers
  - Store Bigtable schema and metadata like access control
- Paxos
Implementation: Architecture

- A single master:
  - Keep track of live tablet servers
  - Garbage collect GFS files
  - Assign tablets
  - Handle schema changes
- Many tablet servers:
  - Split an existing large tablet
  - Handle read/write requests
- Client send read/write requests to tablet server directly
Implementation: Tablet Location

Key: table ID+last row in range
Value: sstables, log, server ID
Implementation: Tablet Location

- $2^{34}$ tablets ($2^{61}$ bytes in 128 MB tablets)
- Efficient: cache & prefetch
- If the location is not in memory / is stale, it will several more round trips
Implementation: Tablet Assignment

- Master:
  - Keep track of live tablet servers
  - Keep track of the current assignment of tablets
  - Assign tablets by sending a load request
Implementation: Tablet Assignment

- A tablet is assigned to one tablet server
- Server file represents the liveness of a tablet server
Common case: Assign t5

Chubby
Servers directory: server files of all alive servers
- File for server 1
- File for server 2
- File for server 3
...

Unassigned list:
t5

Current Assignment:
- Server 1: t0 t1
- Server 2: t2 t3
- Server 3: t4
Chubby
Servers directory: server files of all alive servers
   File for server 1
   File for server 2
   File for server 3
   ...

Current Assignment:
   Server 1: t0 t1
   Server 2: t2 t3
   Server 3: t4

I should assign t5 to server 3 after load balancing algorithm
Chubby
Servers directory: server files of all alive servers
  File 1
  File 2
  File 3
  ...

Current Assignment:
  Server 1: t0 t1
  Server 2: t2 t3
  Server 3: t4

Unassigned list:
t5

Load request

Tablet server1
Tablet server2
Tablet server3
Master

Chubby
Servers directory: server files of all alive servers
  File 1
  File 2
  File 3
  ...

Tablet server1
Tablet server2
Tablet server3

Unassigned list:
  Current Assignment:
  Server 1: t0 t1
  Server 2: t2 t3
  Server 3: t4 t5
What if a tablet server is partitioned?

For example, server1 loses its Chubby session, and loses its exclusive lock.
Master periodically asks all tablet servers: What’s your status of lock?

Master can know how to reach out servers through files in servers directory.
Master tries to get File1’s lock, if succeed, delete file1, and move tablets assigned to server 1 into a set of unassigned tablets.
What if master is partitioned?

The master kills itself if its Chubby session expires
What will the new master do?

- Goals: discover the current tablet assignments
- How?
  - Grab master lock in Chubby
  - Scan the servers directory to find the live servers
  - Communicate with live servers to discover all assigned tablets
  - Scan the METADATA tablets to learn the set of all tablets
  - Add unassigned tablets to a set of unassigned tablets
Implementation: Tablet Serving

- Older data in SSTable, newer in memtable
- Memtable and SSTable are sorted by key
- Read:
  - Merge SSTable Files and memtable
- Write:
  - Check authorization
  - Write in log
  - Commit in memtable
- Recover:
  - Read metadata -> SSTables + log
  - Redo to construct memtable
Implementation: Compactions

- Minor compaction: Convert memtable to a new SSTable
  - To shrink memory usage
  - To reduce REDO in recovery
Implementation: Compactions

- Major/Merging compaction: SSTable + memtable -> one new SSTable periodically
  - To reclaim resources from deleted data
  - To make read less work on merging too many SSTables

<table>
<thead>
<tr>
<th>(a, 3)</th>
<th>delete a</th>
<th>(b, 6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b, 5)</td>
<td>update (b, 6)</td>
<td>(c, 7)</td>
</tr>
<tr>
<td>(c, 7)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Garbage collection
  - Update Metadata tablet
Refinement 1: Locality Group

Locality group = frequently-accessed-together column families

Clients can define locality groups

A separate SSTable for each locality group
Refinement 2: Caching for read performance

Two levels of caching used

The Scan Cache - caches the key-value pairs
- useful when same data read repeatedly

The Block Cache - caches SSTable blocks
- useful when reading data close to recently read ones
Refinement 3: Bloom Filters

Tells you whether a key exist in the SSTable

Needs little memory

Zero False Negative

Low False Positive rate

Looking for key 'b'
Bloom Filter

A 0-1 array
### Refinement 4: Commit-log implementation

| One log file per ______ | tablet | A lot of log files  
|:----------------------|--------|--------------------
|                      | tablet | A lot of concurrent disk seeks |
|                      | server | A refinement  
|                      |        | But complicates recovery - duplicate log reads |

Avoiding duplicate log reads - sort log entries by <table, row name, log sequence number>

Only sort the log during recovery
Refinement 5: Speeding up tablet movement

Need to move tablets when, e.g. changing cluster size

Source tablet server does minor compactions

Destination tablet server doesn't need to go through log
Refinement 6: Exploiting immutability

Benefits of having immutable SSTables:

1. No synchronization of SSTable accesses
2. During split, children reuse parent's SSTables

Dealing with mutable MemTable:

Copy-on-write allows concurrent reads and writes
Evaluation

<table>
<thead>
<tr>
<th>Experiment</th>
<th># of Tablet Servers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>random reads</td>
<td>1212</td>
</tr>
<tr>
<td>random reads (mem)</td>
<td>10811</td>
</tr>
<tr>
<td>random writes</td>
<td>8850</td>
</tr>
<tr>
<td>sequential reads</td>
<td>4425</td>
</tr>
<tr>
<td>sequential writes</td>
<td>8547</td>
</tr>
<tr>
<td>scans</td>
<td>15385</td>
</tr>
</tbody>
</table>

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.
Sequential write is not faster than random write because they are both appending to commit logs

<table>
<thead>
<tr>
<th>Faster operation</th>
<th>Slower operation</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>Read</td>
<td>Read involves accessing GFS for SSTables, but write only access GFS for log</td>
</tr>
<tr>
<td>Sequential read</td>
<td>Random read</td>
<td>caching</td>
</tr>
</tbody>
</table>

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.
Our Understanding

- **Pros:**
  - Scalability
  - Cluster resizing without downtime
  - High throughput

- **Cons:**
  - No multi-row transaction
  - No sql queries or joins
  - Not a good solution for less than 1 TB of data
Reference

https://www.youtube.com/watch?v=r1bh90_8dsg&t=630s