CS 744: BIG DATA SYSTEMS

Shivaram Venkataraman
Fall 2018
- Assignment 1
- Projects
- Piazza
MOTIVATION

Storing large amounts of *semi-structured* data
- Traditionally done using database systems

Varied processing needs
- low latency to bulk processing
- data size
- schema
BIGTABLE: HIGHLIGHTS

1. Scalability: Petabytes of data, thousands of machines

2. Wide applicability: Handles > 60 applications

3. Fault tolerant: High availability

4. High Performance
OUTLINE

- Data Model and API
- Architecture
- Master, Tabletserver functionality
- Optimizations
**WRITE API**

**Single** row at a time!

Set a number of columns or delete some

**Apply is atomic**

Support for **read-modify-write** transactions

```c
// Open the table
Table *T = OpenOrDie("/bigtable/web/webtable");

// Write a new anchor and delete an old anchor
RowMutation rl(T, "com.cnn.www");
rl.Set("anchor:www.c-span.org", "CNN");
rl.Delete("anchor:www.abc.com");
Operation op;
Apply(&op, &rl);
```
SCANNER API

Fetch any number of columns, column families

Filter rows by regex

Iterator pattern, rows arriving in sorted order

Scanner scanner(T);
ScanStream *stream;
stream = scanner.FetchColumnFamily("anchor");
stream->SetReturnAllVersions();
scanner.Lookup("com.cnn.www");
for (; !stream->Done(); stream->Next()) {
    printf("%s %s %lld %s\n",
            scanner.RowName(),
            stream->ColumnName(),
            stream->MicroTimestamp(),
            stream->Value());
}
<table>
<thead>
<tr>
<th>Language:</th>
<th>Contents:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;aaa.com&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;cnn.com&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;cnn.com/sports.html&quot;</td>
<td>&quot;&lt;html&gt;...&quot;</td>
</tr>
<tr>
<td>Tablets</td>
<td></td>
</tr>
<tr>
<td>&quot;website.com&quot;</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>&quot;yahoo.com/kids.html&quot;</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>&quot;yahoo.com/kids.html\0&quot;</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>&quot;zuppa.com/menu.html&quot;</td>
<td></td>
</tr>
</tbody>
</table>
SYSTEM ARCHITECTURE

BigTable Master: metadata ops, rebalancing

BigTable TabletServer
Serves data from tablets

BigTable TabletServer

BigTable TabletServer

GFS: Store tablets, replicate

Chubby: Leader election, store metadata
CHUBBY: A LOCK SERVICE

Leader election: Classic problem in distributed systems

Approach: Build a separate service to handle leader election

Properties:
- Uses Paxos algorithm
- Low write throughput
- Store small amounts of data
TABLET LOCATION

- Hierarchical metadata
- Root of metadata in Chubby
- Client library caches tablet locations
**MASTER FUNCTIONALITIES**

Tablet assignment

- Master tracks tablet → tablet server mapping
- **METADATA** has the complete list of tablets
- Each tabletserver has list of tablets that are being served

- Uses heartbeat + Chubby to detect tablet server failures
- On master failure, scan **METADATA** and list tablet servers
WORKER FUNCTIONALITY

Tablets stored in GFS

Writes
- Commit log
- Insert memtable

Read
- Merge SSTable and memtable
WORKER FUNCTIONALITY

Challenge: Memtable keeps growing over time

Minor Compaction
- Freeze memtable, write it as SSTable to disk
- But now need to merge more SSTables

Major Compaction
- Read memtable + all SSTables for this tablet
- Write out new SSTable. Handles garbage collection
NOTABLE OPTIMIZATIONS

Caching
- Scan Cache: key-value pairs returned by the SSTable
- Block Cache: SSTables blocks that were read from GFS.

Bloom filter
- Probabilistic data structure: Definitely not or maybe in it
- Use this to eliminate SSTables that need to be read
OTHER OPTIMIZATIONS

- Single commit log per tabletserver
- Sort commit log entries during recovery

- Tablet Splitting
  - Tablet server records changes in METADATA table
  - Child tablets share SSTables with parent
BigTable Replication (New Since OSDI’06)

• Configured on a per-table basis

• Typically used to replicate data to multiple bigtable clusters in different data centers

• \textit{Eventual consistency model}: writes to table in one cluster eventually appear in all configured replicas

• Nearly all user-facing production uses of BigTable use replication
BIGTABLE: DISCUSSION

Generality vs. Specificity

Simplicity, Layering

Scalability

User overheads
QUESTIONS / DISCUSSION ?