Bigtable: A Distributed Storage System for Structured Data

Jia Liu
Outline

• Introduction
• Data Model
• API
• Building Blocks
• Implementation
• Refinements
• Performance Evaluation
• Applications
Introduction

• Def: Big table is a distributed storage system for managing structured data.
• scale: a very large size. PB data.
• Many projects in Bigtable:
  – Web indexing
  – Google Earth
  – Google Finance
• Difference demands
  – Data size(from URL to Web pages)
  – latency requirements(backend bulk processing to real time data serving)
Introduction

• Objective
  – wide applicability
  – scalability
  – high performance
  – high availability

• Clusters Configurations
  – handful
  – thousands of servers (Terabytes data)
Introduction

- **Bigtable** resembles a database
  - share implementation strategies with database.
- Not support full relational data model
- Provides simple data model (dynamic control over data layout and format)
- Treats data as uninterpreted string
- Schema parameters dynamically control
  - from memory
  - from disk
Data Model

• A bigtable is a sparse, distributed persistent multi-dimensional sorted map.
• row: string, column: string, time: int64-string
• value in map: uninterpreted array of bytes
• Ex.: URL, Various aspects of web pages, column under timestamps when fetched.
Data Model

Bigtable: A distributed Storage System for Structured Data
Data Model

• **Row**
  - Row key is an arbitrary string
    • access to data under row is atomic
  - Bigtable maintains data in lexicographic order by row key
  - **Row range** is dynamically partitioned
  - Tablet: unit of distribution and load balancing

• Ex. Group same domain pages together into rows by reversing the hostname components of URLs.
Data Model

• Columns Families
  - Column key are grouped into sets
  - Basic unit of access control
  - Must be created before store data

  – Syntax
    - Family: qualifier
      - Family name must be printable
      - Qualifier may be arbitrary string

  – Access control and both disk&memory accounting are performed at this level.
Data Model

• Timestamps
  – Versions of same data are indexed by timestamps
  – 64-bit integers
  – Can represent "real-time" in microseconds
  – Stored in decreasing timestamp (recent read 1st)
  – garbage-collect
API

• Bigtable API functions provide
  – creating/deleting tables and column families.
  – changing cluster, table, and column family metadata. Ex. access control rights

• Writes
  – Set()  //write cells in a row
  – DeleteCells()  //delete cells in a row
  – DeleteRow()  //delete all cells in a row
API

- // Open the table
- Table *T =
  OpenOrDie("/bigtable/web/webtable");
- // Write a new anchor and delete an old anchor
- RowMutation r1(T, "com.cnn.www");
- r1.Delete("anchor:www.abc.com");
- Operation op;
- Apply(&op, &r1);
API

• Reads
  – Scanner: read arbitrary cells in a bigtable
    • Can restrict returned rows to a particular range
    • Can ask for data from 1 row, 2 rowsetc.
    • Can ask for all columns, just certain column families, or specific columns.

• Ex. Restrict scan only produce anchor: *.cnn.com
API

- 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());
}
Building Blocks

- Bigtable uses the distributed Google Files System (GFS) to store log and data files.
- Use **SSTable** to store bigtable data.

- Persistent, ordered and immutable map
- Chunks of data (64KB) plus index
- Open SSTable:
  - Index → memory → finding block → disk
Building Blocks

• **Chubby**
  - highly-available and persistent distributed lock service.

• **Tasks:**
  - ensure at least one active master at any time
  - store the bootstrap location (Tablet location)
  - discover tablet servers and finalize tablet server deaths (Tablet assignment)
  - store Bigtable schema information (column family information)
  - store access control lists.
Implementation

- **Three major components:**
  - a library is linked into every client
  - one master server
  - many tablet servers.
- **Master server:**
  - assign tablets to tablet server
  - detecting the addition / expiration of tablet server
  - balancing tablet server load
  - garbage collection of files in GFS
  - handles schema changes.
Implementation

• **Tablet server:**
  - manage a set of tablets.
  - handles read and write request to the tablet
  - Initially each table consist just one tablet.
  - splits tables if grown too large(default 100-200MB)

• **Master is lightly loaded**
  - client data does not move through the master service. client communicate directly with tablet server for read and writes.
- A three level hierarchy
- Client caches tablet locations.

Figure 4: Tablet location hierarchy.

Bigtable: A distributed Storage System for Structured Data
Tablet Location

- Client library caches tablet locations.
  - uncorrect/unknown ----> move up the tablet location hierarchy
  - empty---->requires three network round-trips
  - stale----> take up to six round-trips
Tablet Assignment

• Each tablet is assigned to one tablet server at a time.
• master record
  – the set of live tablet servers
  – current assignment of tablets to tablet servers
  – unassigned tablets.
Tablet Assignment

• Steps at startup
  – Grabs a unique master lock in Chubby, prevents con-current master instantiations.
  – Scan the server directory in Chubby to find live servers
  – Master communicates with every live tablet server to discover what tablets are already assigned to each server
  – Master scans the METADATA table to learn the set of tablets.
Tablet Serving

Figure 5: Tablet Representation
Tablet Serving

• Write operation:
  – The server checks the well-formed
  – checks sender's operating authority
  – Write operation's content insert into memtable.

• Read operation:
  – check the authorization
  – well-formedness
  – valid read operation execute on a merged view of SSTable and memtable.
Compactions

• As write operations execute:
  – memtable size increase
  – frozen memtable when reaches threshold
  – Converted to SSTable & written to GFS (Minor compaction)

• Two goals:
  – shrink the memory usage of the tablet server
  – reduce read data from log during recovery.
Compactions

• Merging Compaction
  – Every Minor compaction create a new SSTable,
  – Read operations need to merge updates from SSTable
  – Discarded input SSTable and memtable when merging Compaction finished,
Refinements

• Locality groups
  – Clients can group multiple column families together into a locality group.
  – benefit: improve the reading efficiency
  – Ex. language& checksums (a group)
    contents (another group)

• Compression
  – Client can control SSTable for locality group
  – benefit: read small data do not need decompressing the entire file.
Refinements

• Caching for read performance
  – Tablet server use two levels of caching.
  – Scan Cache is a higher-level cache.
    • useful for applications tend to read same data repeatedly
  – Block Cache is the lower-level cache.
    • useful for applications tend to read the data close to recently read.

• Bloom Filters
  – allow us to ask SSTable contain specified row/column pair.
    • reduce read operations from disk.
Refinements

• Speeding up tablet recovery
  – source tablet server do minor compaction on that tablet.
  – benefit: reduce uncompacted state in commit log, so reduces recovery time.
## Performance

<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.*
Performance

- Single Tablet-server performance
  - Random reads are slower than all other operations.
  - Random reads from memory are much faster
  - Sequential reads perform better than random reads
  - Scan are faster.
Performance

Bigtable: A distributed Storage System for Structured Data
Real Applications

• Google Analytics
  – analytics.google.com is a service that helps webmaster analyze traffic patterns at their web sites. It provides aggregates statistics.
  – Row client table & Summary table

• Google Earth
  – Google operates a collection of services that provide user with access to high-resolution satellite imagery of the world's surface.

• Personalized Search
  – www.google.com/psearch is an opt-in service that records user queries and clicks across a variety of Google Properties.
## Real Applications

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

Bigtable: A distributed Storage System for Structured Data
Reference

- http://en.wikipedia.org/wiki/BigTable
- DeWitt, D., Katz, R. etc. Parallel database systems: The future of high performance database systems. CACM 35, 6(June 1992)
Thanks