BIGTABLE: A DISTRIBUTED STORAGE SYSTEM FOR STRUCTURED DATA

Written by Fay Chang et al.
Presented by Allen George
Outline

- Introduction
- Data Model
  - Rows
  - Column Families
  - Timestamps
- API
- Building Blocks
- Implementation
  - Tablet Location
  - Tablet Serving
  - Compactions
- Conclusion
Introduction

• Bigtable is a distributed storage system for managing structured data.

• Goals of Bigtable
  • Wide applicability, scalability, high performance, high availability

• Serves many different products

• Data treated as uninterpreted strings
Data Model

- Distributed, persistent, multi-dimensional sorted map

From Bigtable Paper
“The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.” While a table is stored in the form of KV(Key-value) items, what is the key?

The key is a combination of the row key, column key, and the timestamp.

(row: string, column: string, time: int64) → string
Rows

- Row keys are arbitrary strings
  - Up to 64KB in size
  - Typically 10-100 bytes for most users
- Reads/writes for a single row key are atomic
Question 2

• “Clients can exploit this property by selecting their row keys so that they get good locality for their data accesses.”

How would clients select keys to get good locality? What possible advantages could a client obtain by having the locality?

• Bigtable maintains the row keys in lexicographic (alphabetic) order.

• Reading a short range of rows will be more efficient and require less machines to communicate to get the values.
Column Families

- Column keys grouped into sets called column families
- Named with syntax `family:qualifier`
- Family must be created before data can be stored.
Column Families

• Number of distinct column families is small
• Unbound number of columns
• Basic unit of access control
Timestamps

- 64-bit integers, either
  - Assigned by Bigtable ("real time" in microseconds)
  - Assigned by client applications
- Different versions in decreasing order
Timestamps

- Settings to control garbage collection
  - Last x number of versions
  - Versions made in a certain time period
API

- Creating/deleting tables/column families
- Read/write/delete values from rows
- Iterate over subset of table data
- Limit rows/columns/timestamps in a scan
- Batching writes across row keys
- Scripts (Google Sawzall)
- MapReduce compatibility
  - Input source and/or output target
Building Blocks

- Google File System (GFS) (logs/data)
- Google SSTable file format
  - Persistent, ordered, immutable map from keys to values

```
  64K block  64K block  64K block  SSTable
        |           |          |            |
        |           |          |            | Index
```

From Lecture 8
Question 3

• “Bigtable uses the distributed Google File System (GFS) to store log and data files.” To ensure high data reliability, does BigTable need to maintain multiple replicas for each of its data items?

• GFS already has inbuilt mechanisms to handle file replication.
The Google SSTable file format is used internally to store Bigtable data. An SSTable provides a persistent, ordered immutable map from keys to values, where both keys and values are arbitrary byte strings.

What does it mean by “immutable”? Why is this feature required?
Question 4

• What does it mean by “immutable”?

• “Immutable” means the SSTable cannot be modified once it is created.
Question 4

• Why is this feature required?

• Immutability is required because the cost of trying to modify SSTables as write requests come in is very high. Instead, it is faster to let the SSTables be immutable and store the changes in the memtable elsewhere.
Question 5

“A block index (stored at the end of the SSTable) is used to locate blocks; the index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: …”

Describe how a KV item is retrieved from an SSTable and why only one disk access is required for a lookup? [Hint: assume each block in an SSTable is 4KB, the disk access unit.]

• The system can do a binary search on the index in memory to locate the appropriate block in the SSTable.
Implementation

• Library for the client
• One master server
  • Assigning tablets, detecting added/removed servers, balancing load
• Many tablet servers
  • Each manages a set of tablets, handles read/write requests, splits large tablets
• Clients talk directly to tablet servers
Tablet Location

Chubby file

Root tablet
(1st METADATA tablet)

Never split

Other METADATA tablets

UserTable1

UserTableN

From Bigtable Paper

From Bigtable Paper
Tablet Location

- Each METADATA row is 1KB of memory
- Can address $2^{34}$ tablets
- Clients cache location
- Locations stored in server memory
- Clients prefetch location
Tablet Serving

- **GFS**
  - tablet log
  - Write Op

- **Memory**

- **memtable**

- **Read Op**
  - SSTable Files
    - From Bigtable Paper
Question 6

“Of these updates, the recently committed ones are stored in memory in a sorted buffer called a memtable; the older updates are stored in a sequence of SSTables.”

Why do older updates exist and possibly exist in a sequence of SSTables?

It’s not convenient or cost-effective to delete older versions as soon as they’re not needed; it would impact the performance. Instead, older versions are stored temporarily until the system finds a break to go back and delete them.
Compactions
Question 7

“A merging compaction that rewrites all SSTables into exactly one SSTable is called a major compaction.”

What is minor compaction, and what is major compaction?
Why is major compaction needed?
How is a KV item deleted?
Question 7

• What is minor compaction?

• Minor compaction is converting a memtable into an SSTable.

• Merging compaction reads the contents of a few SSTables and the memtable and produces a new SSTable.
Question 7

- What is major compaction?

- Major compaction rewrites all SSTables into one SSTable that contains no deleted data or deletion entries.
Question 7

• Why is major compaction needed?

• Major compaction is needed so that the level of SSTables can be reduced to a smaller amount. Without major compaction, the number of levels would continue to grow and make read requests take a long time to process.
Question 7

- How is a KV item deleted?
  - Delete operation sent to Bigtable
  - Stored in the memtable as a deletion entry
  - Deletion entry suppress KV item
  - Minor compaction turns the memtable into an SSTable.
  - Major compaction finally removes the deleted entry.
Conclusion

- Goals of Bigtable
  - Wide applicability
  - Scalability
  - High performance
  - High availability
References

• Bigtable: A Distributed Storage System for Structured Data
  http://research.google.com/archive/bigtable-osdi06.pdf

• ECE 7650 Lecture 8