Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

Google, Inc.

Presented by: Youhui Bai

OSDI ‘06: 7th USENIX Symposium on Operating Systems Design and Implementation

Outline

- Introduction
- Data model
- Implementation
- Refinements
- Performance Evaluation
- Real applications
- Conclusion

Motivation

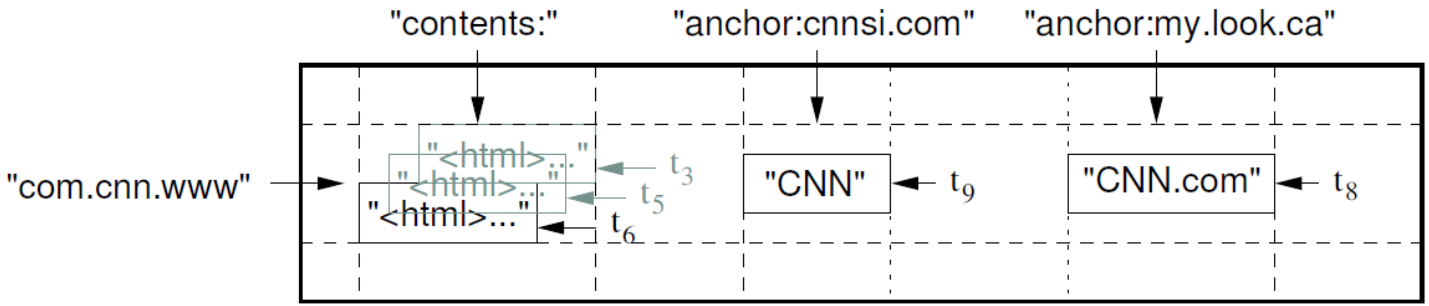
- Google scale:
  - Lots of requests
  - Need suitable system to back services like webpages, emails, maps...
- No commercial service big enough by then
- Well suitable for future scaling

Introduction

- Bigtable is a distributed Storage system for managing structured data at Google.
- Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability.
- Bigtable does not support full relational data model.Instead, it provides the clients with a simple data model.
- Data is indexed using row and column names that can be arbitrary strings.

Data Model

- Bigtable is a **sparse, distributed**, **persistent**, **multidimensional sorted** map.
- Map is indexed by a row key, column key, and a timestamp.
- (row:string, column:string, time:int64) -> string

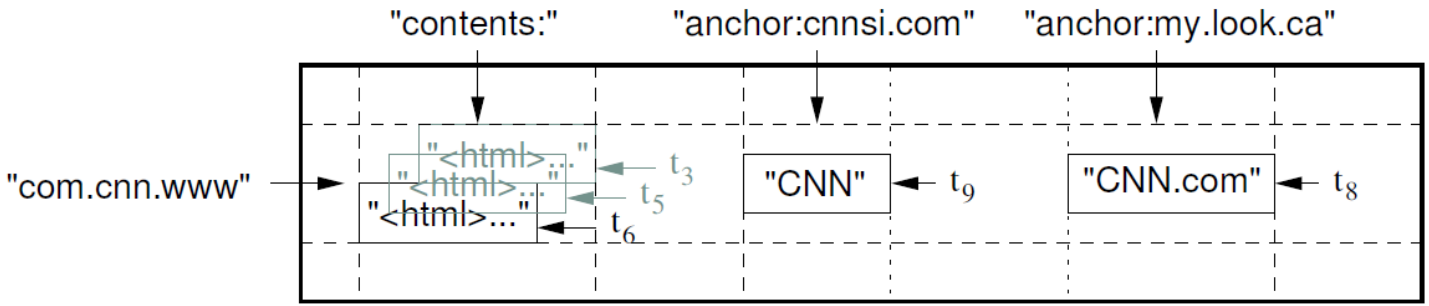


Data model : *Row Keys*

- Row keys are arbitrary strings.
- Every read and write of data under a single row key is atomic.
- Bigtable maintains data in lexicographic order by row key.
- Each row range is called a **tablet**.

Data Model: *Column Families*

- Column keys are grouped into sets called column families.
- Column key is named using the following syntax: *family: qualifier*
- Basic units of access control.



Data model: *Timestamps*

- Each cell in Bigtable can contain multiple versions of same data.
- Decreasing order: read the most recent version first.
- Bigtable garbage-collects cell versions automatically.

Data Model

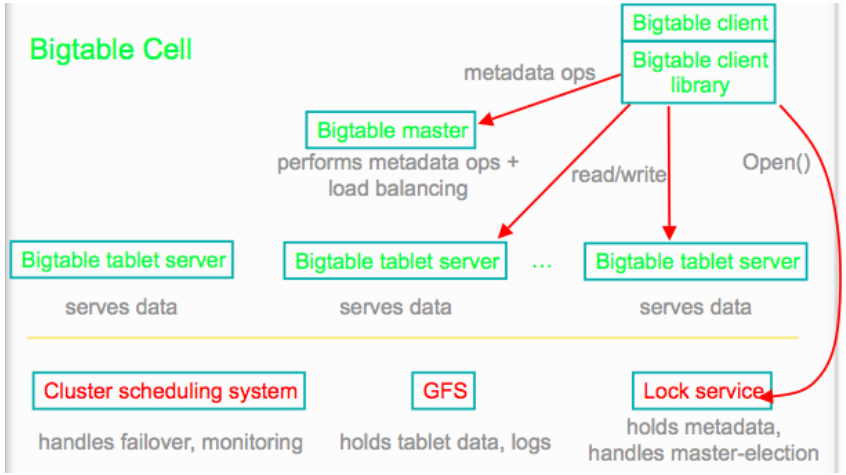
- API
  - Functions for creating and deleting tables and column families, changing clusters, tables and column family metadata (access control rights).
  - Client applications can write/delete values in the Bigtable, look up values from individual rows, columns, or iterate over a subset of the data from the table.

Data Model

- Building Blocks
  - *GFS:Google File System*-stores log and data files.
  - *Google SSTable File format*- provides a persistent, ordered immutable map from keys to values.
  - *Chubby*- highly available and persistent distributed lock service.

Data Model

- SSTable, Tablet and GFS
  - How bigtable is formatted /stored
- Chubby
  - How to access bigtable



Implementation

- A library that is linked into every client.
- A Master Server
- Tablet Servers

Implementation

Master server

- The master is responsible for assigning tablets to tablet servers.
- Detecting the addition or expiration of tablet servers.
- Balancing the tablet server load.
- Garbage collection.
- Handle schema changes.

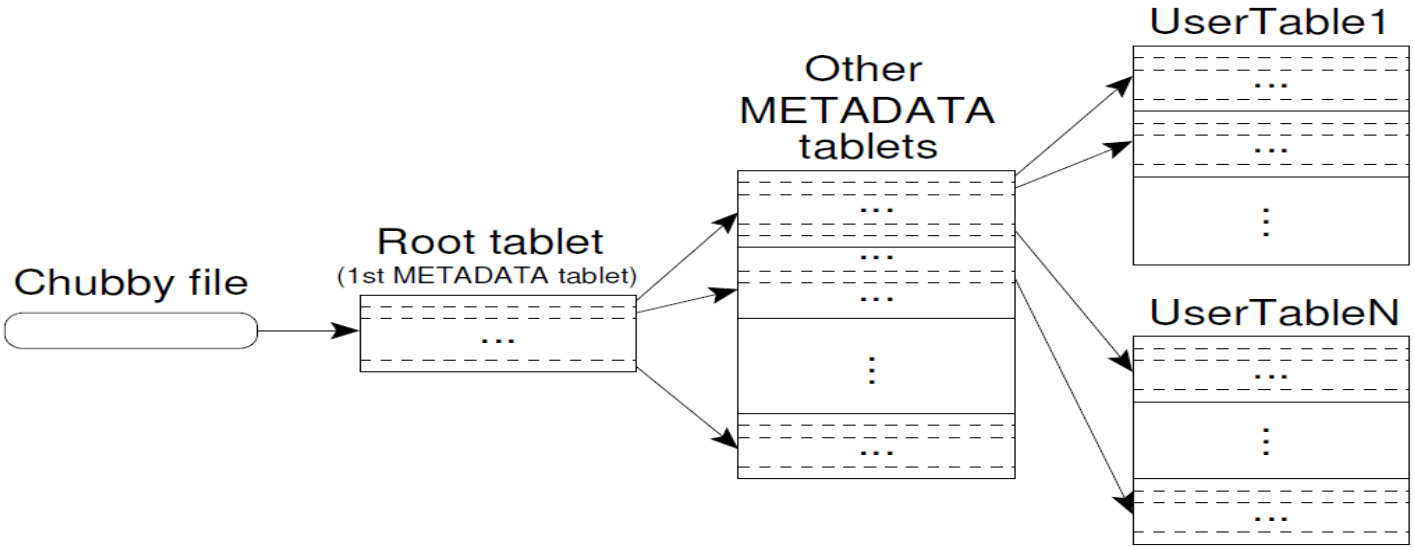
Implementation

Tablet server

- Tablet servers can be added and removed dynamically from a cluster to accommodate changes in the workload.
- Each tablet server manages a set of tablets.
- Tablet server handles read and write requests
- Also splits tablets that have grown too large.
- Clients communicate directly with the tablet server.

Implementation: *Tablet location*

- Three-level hierarchy

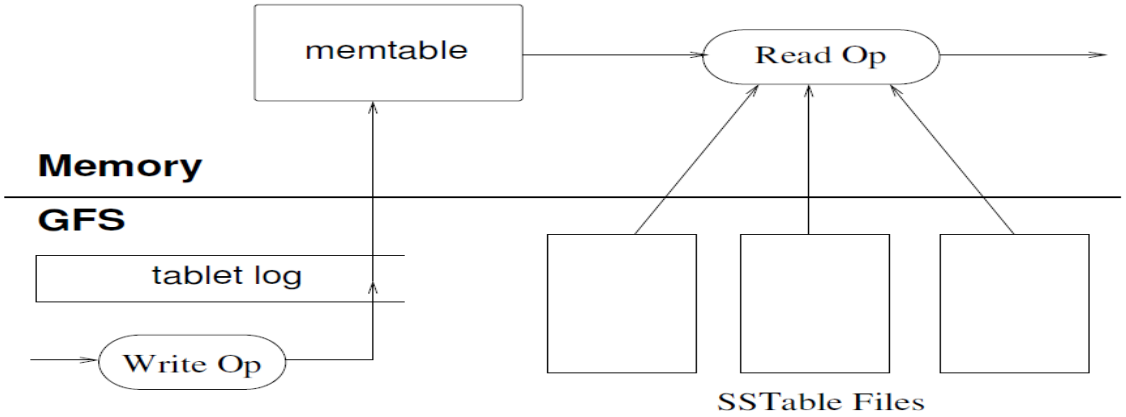


Implementation: *Tablet Assignment*

- Each tablet is assigned to one tablet server at a time.
- Bigtable uses chubby to keep track of tablet servers.
- When a tablet is unassigned, and a tablet server with sufficient room for the tablet is available, the master assigns the tablet by sending a tablet load request to the tablet server.
- Master periodically asks the tablet servers for the status of its lock.
    - A server losses its lock
    - The master can't reach a server

Implementation: *Tablet Serving*

- The persistent state of a tablet is stored in GFS as shown in the figure.
- Updates are committed to a commit log.
- The recently committed updates are stored in the memory in a sorted buffer called a memtable; older updates are stored in a sequence of SSTables.



Implementation: *Compaction*

- Minor Compaction
  - When the memtable size reaches a threshold, the memtable is frozen, a new memtable is created and the frozen memtable is converted to an SSTable and written to the GFS.
- Major Compaction
  - A merging compaction that rewrites all SSTables into exactly one SSTable.

Refinements

- *Locality groups*
  - Clients can group multiple column families together into a locality group.
- *Compression*
  - Clients can control whether or not SSTables for a locality group are compressed, and if so, which compression format is used.
- *Caching for read performance*
  - Two-level caching.
- *Bloom filters*
  - It allows to ask whether an SSTable might contain any data for a specified row/column pair.
- *Commit-log implementation*
  - Single commit log per tablet server.
- *Speeding up tablet recovery*
  - A minor compaction reduces recovery time by reducing the amount of uncompacted state in the tablet server's commit log.
- *Exploiting immutability*
  - SSTables generated are immutable and enables Bigtable to split tablets quickly.

Performance Evaluation

- **Experiment setups**
  - N tablet servers, N is varied
  - A tablet server uses 1 GB memory
  - GFS: 1786 machines with two 400 GB IDE hard drivers
  - N clients, ensure clients are never a bottleneck
  - Each machine
    - Two dual-core Opteron 2 GHz chips
    - Enough memory to hold the running processes
    - Single gigabit Ethernet link
  - Network architecture
    - Two-level tree-shaped switched network
    - 100-200Gbps of aggregate bandwidth available at the root

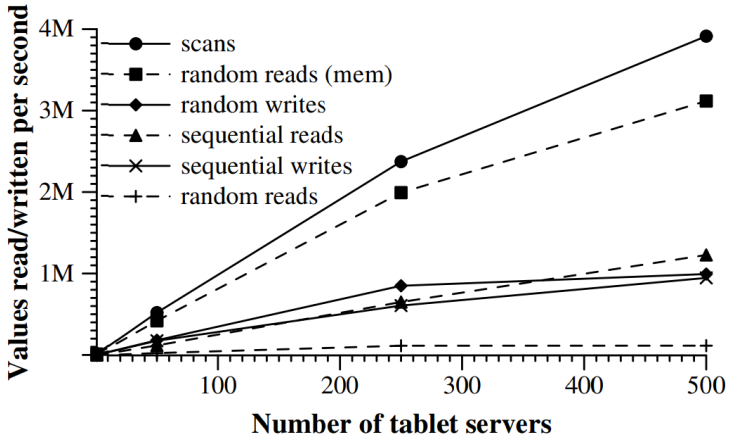
Performance Evaluation

- **Single tablet-server performance**
  - Random reads are slower than all other operations.
  - Random reads from the memory are much faster.
  - Random and sequential writes perform better than random reads.
  - Sequential reads perform better than random reads.
  - Scans are even faster.

Experiment	# of Tablet Servers			
	1	**50**	**250**	**500**
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843

Performance Evaluation

- **Scaling**
  - Aggregate throughput increases.
  - Performance does not increase linearly.
  - For most benchmarks, there is a significant drop in per server throughput.



Real Applications

- Google Analytics
- Google Earth
- Personalized Search
- Web Indexing
- Google Finance
- Orkut
- Writely

Conclusion

- Bigtable is a distributed system for storing structured data at Google.
- Significant advantages of building own storage system at Google.
- Users like the performance and high availability that is provided by Bigtable.