WELCOME !

Google Bigtable

TOMCY THANKACHAN

Presentation Overview

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

Why not a DBMS?

- Scale is too large for most commercial Databases
- Cost would be very high
- Low-level storage optimizations help performance significantly
- Hard to map semi-structured data to relational database
- Non-uniform fields makes it difficult to insert/query data

What is bigtable ?

- BigTable is a distributed storage system for managing structured data
- Bigtable does not support a full relational data model
- Scalable
- Self managing

- Used by more than 60 Google products
  - Google Analytics
  - Google Finance
  - Personalized Search
  - Google Documents
  - Google Earth
  - Google Fusion Tables
  - …
- Used for variety of demanding workloads
  - Throughput oriented batch processing
  - Latency sensitive data serving

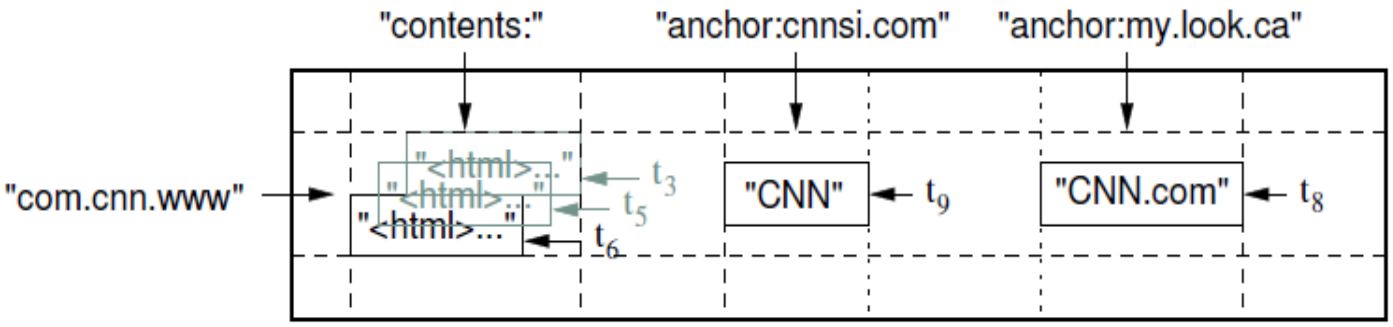
- Goals
  - Wide applicability
  - Scalability
  - High performance
  - High availability
- Simple data model that supports dynamic control over data layout and format

Data Model

- A Bigtable is a sparse, distributed, persistent multidimensional sorted map.
- The map is indexed by a row key, column key, and a timestamp.

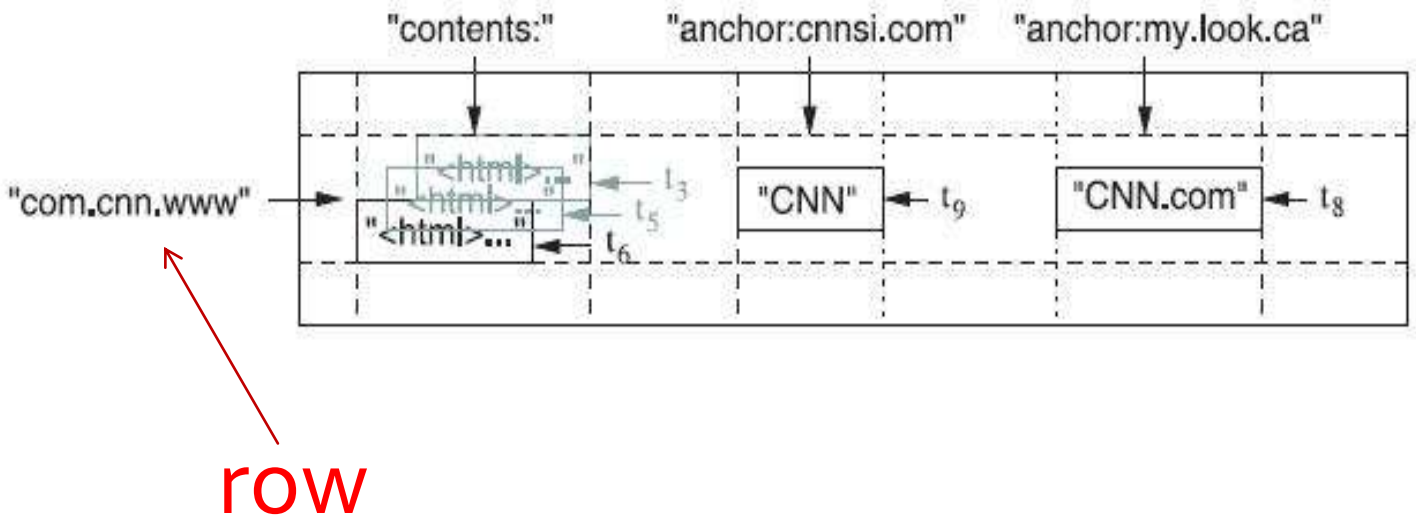
(row:string, column:string, time:int64) → string

Webtable



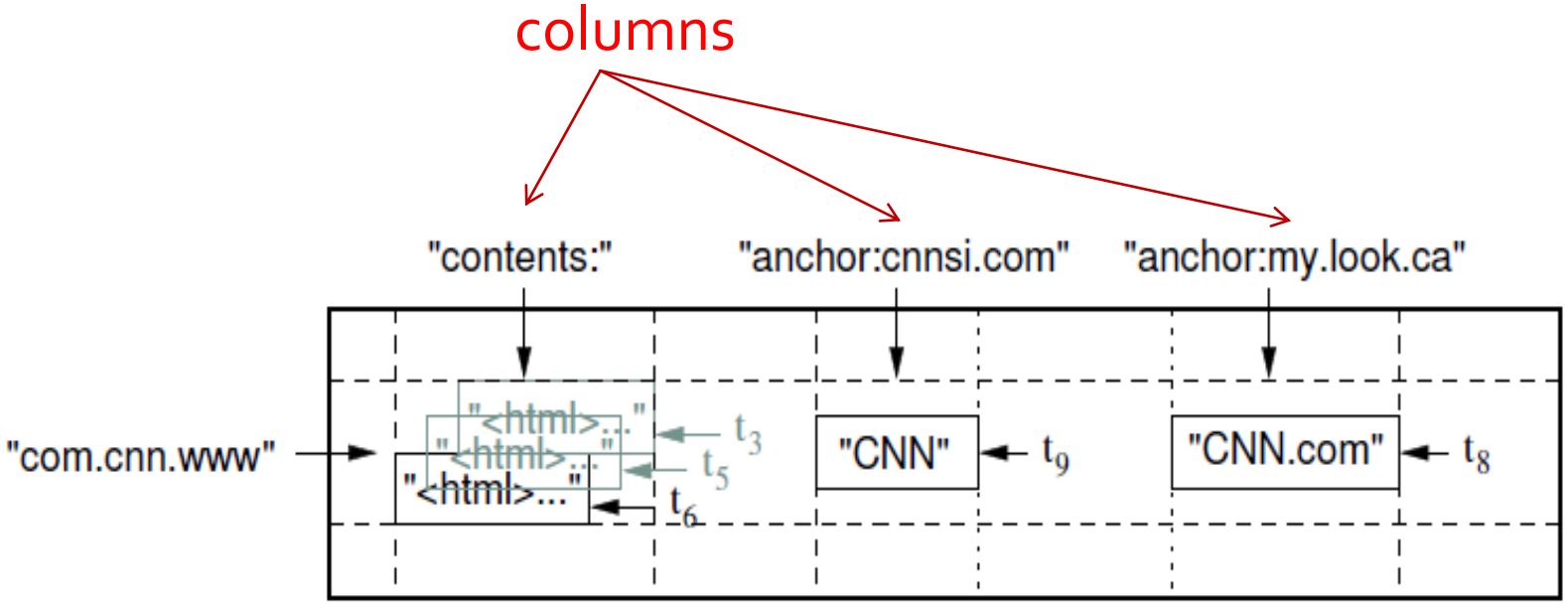
Datamodel-ROW

- The row keys in a table are arbitrary strings.
- Data is maintained in lexicographic order by row key
- Each row range is called a tablet, which is the unit of distribution and load balancing.



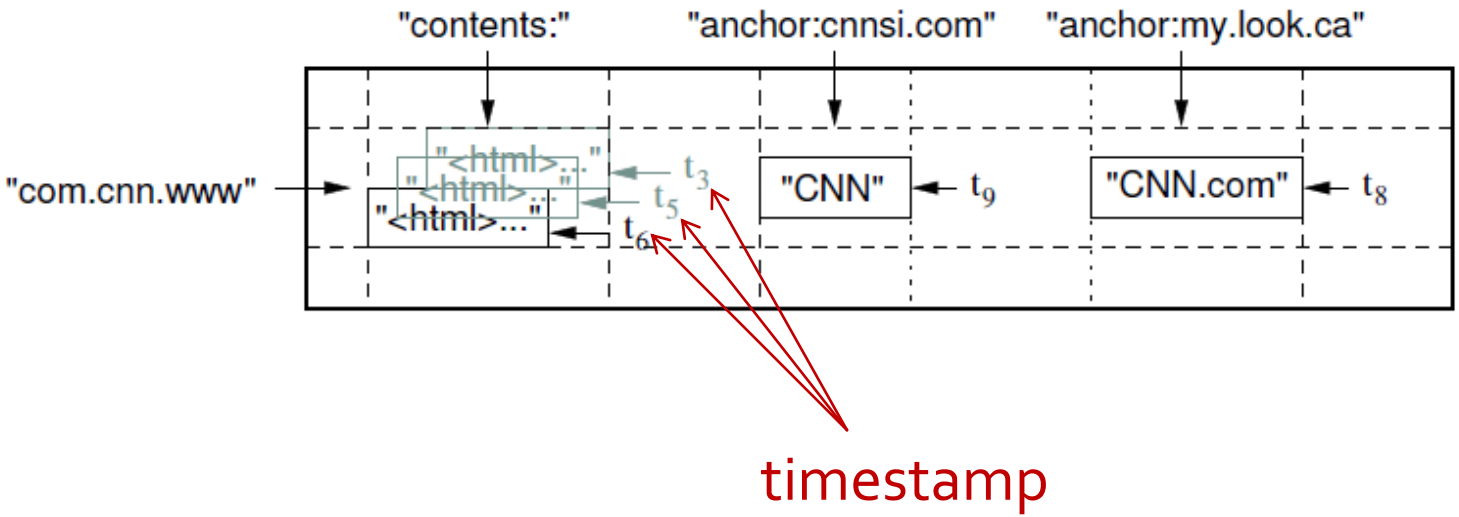
Datamodel-column

- Column keys are grouped into sets called *column families*.
- Data stored in a column family is usually of the same type
- A column key is named using the syntax: *family* : *qualifier*.
- Column family names must be printable , but qualifiers may be arbitrary strings.



Datamodel-Timestamp

- Each cell in a Bigtable can contain multiple versions of the same data
- Versions are indexed by 64-bit integer timestamps
- Timestamps can be assigned:
  - automatically by Bigtable , or
  - explicitly by client applications

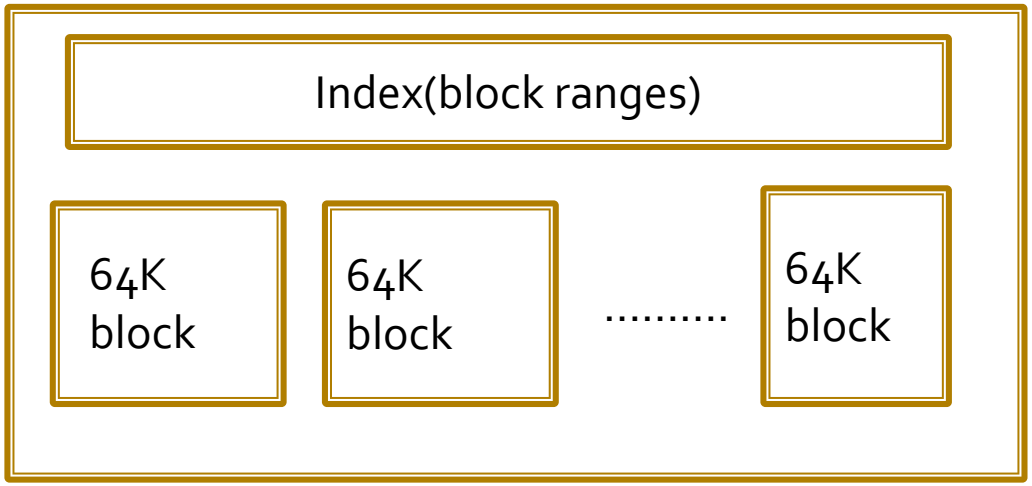


APIs

- The Bigtable API provides functions :
  - Creating and deleting tables and column families.
  - Changing cluster , table and column family metadata.
  - Support for single row transactions
  - Allows cells to be used as integer counters
  - Client supplied scripts can be executed in the address space of servers

BUILDING BLOCKS

- Bigtable is built on several other pieces of Google infrastructure.
- **Google File system(GFS)**
- **SSTable** : Data structure for storage



- **Chubby**: Distributed lock service.

Implementation

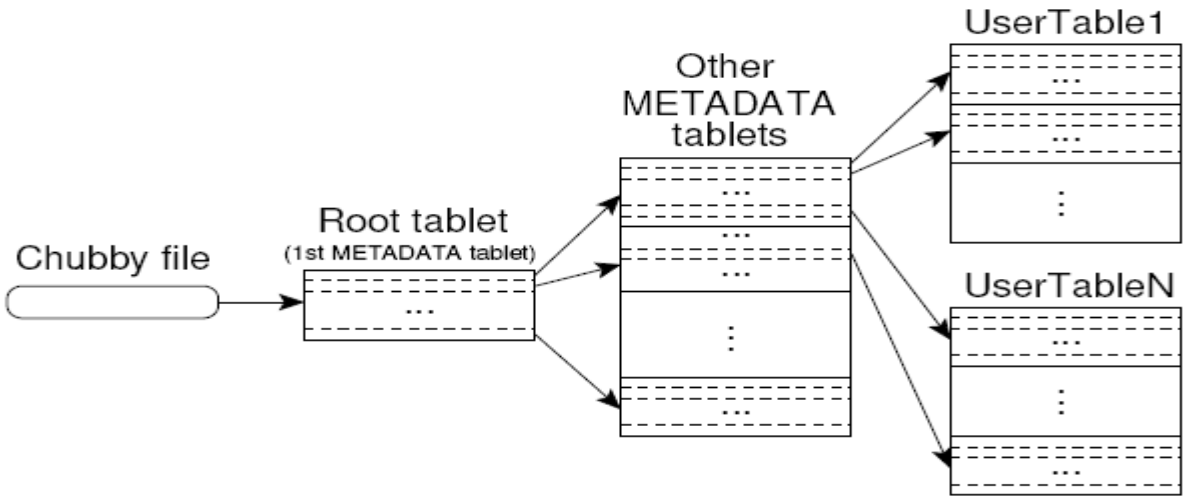
- Three major components
  - **Library linked into every client**
  - **Single master server**
    - Assigning tablets to tablet servers
    - Detecting addition and expiration of tablet servers
    - Balancing tablet-server load
    - Garbage collection files in GFS
  - **Many tablet servers**
    - Manages a set of tablets
    - Tablet servers handle read and write requests to its table
    - Splits tablets that have grown too large

Implementation (cont...)

- Clients communicates directly with tablet servers for read/write
  - Each table consists of a set of tablets
    - Initially, each table have just one tablet
    - Tablets are automatically split as the table grows
  - Row size can be arbitrary (hundreds of GB)

Locating Tablets

- Three level hierarchy
  - Level 2: Root tablet contains the location of METADATA tablets
  - Level 3: Each METADATA tablet contains the location of user tablets
  - Level 1: Chubby file containing location of the root tablet
    - Location of tablet is stored under a row key that encodes table identifier and its end row



Assigning Tablets

- Tablet server startup
  - It creates and acquires an exclusive lock on , a uniquely named file on Chubby.
  - Master monitors this directory to discover tablet servers.

- Tablet server stops serving tablets
  - If it loses its exclusive lock.
  - Tries to reacquire the lock on its file as long as the file still exists.
  - If file no longer exists, the tablet server will never be able to serve again.

Assigning Tablets

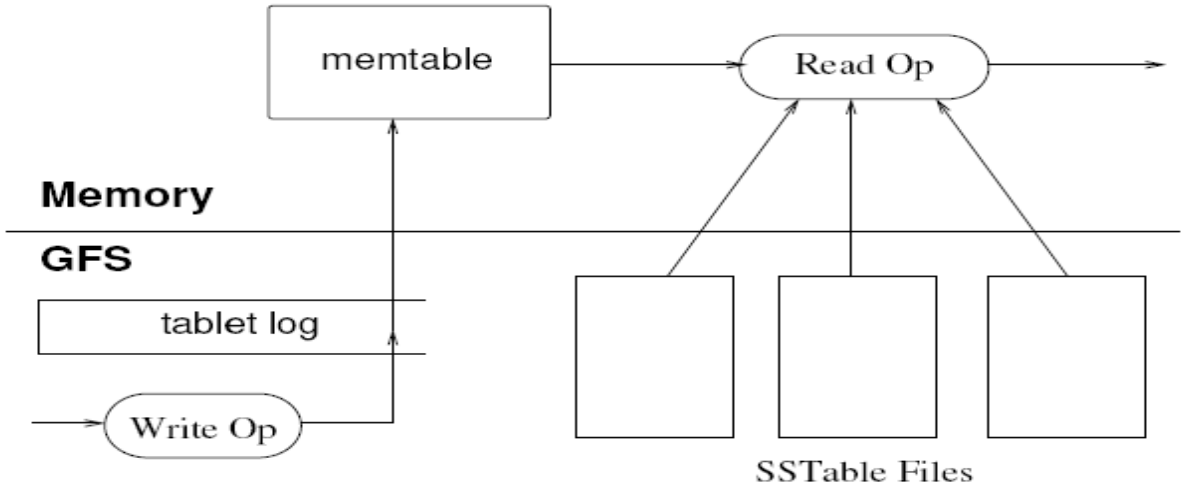
- Master server startup
  - Grabs unique master lock in Chubby.
  - Scans the tablet server directory in Chubby.
  - Communicates with every live tablet server
  - Scans METADATA table to learn set of tablets.

- Master is responsible for finding when tablet server is no longer serving its tablets and reassigning those tablets as soon as possible.
  - Periodically asks each tablet server for the status of its lock
  - If no reply, master tries to acquire the lock itself
  - If successful to acquire lock, then tablet server is either dead or having network trouble

Tablet Serving

- Updates committed to a commit log
- Recently committed updates are stored in memory –memtable
- Older updates are stored in a sequence of SSTables.

Tablet Serving



- Write operation
  - Server checks if it is well-formed
  - Checks if the sender is authorized
  - Write to commit log
  - After commit, contents are inserted into Memtable

- Read operation
  - Check well-formedness of request.
  - Check authorization in Chubby file
  - Merge memtable and SSTables to find data
  - Return data.

Compaction

In order to control size of memtable, tablet log, and SSTable files, "compaction" is used.

1. **Minor** Compaction.- Move data from memtable to SSTable.
2. **Merging** Compaction. - Merge multiple SSTables and memtable to a single SSTable.
3. **Major** Compaction. - that re-writes all SSTables into exactly one SSTable

Refinements

- **Locality groups**
  - Clients can group multiple column families together into a *locality group.*
- **Compression**
  - Compression applied to each SSTable block separately
  - Uses *Bentley and McIlroy's* scheme and *fast compression* algorithm
- **Bloom filters**
  - Reduce the number of disk accesses

Refinements

- **Caching**
  - Scan Cache: a high-level cache that caches key-value pairs returned by the SSTable interface
  - Block Cache: a lower-level cache that caches SSTable blocks read from file system
- **Commit-log implementation**
  - Suppose one log per tablet rather have one log per tablet server

Performance Evaluation

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

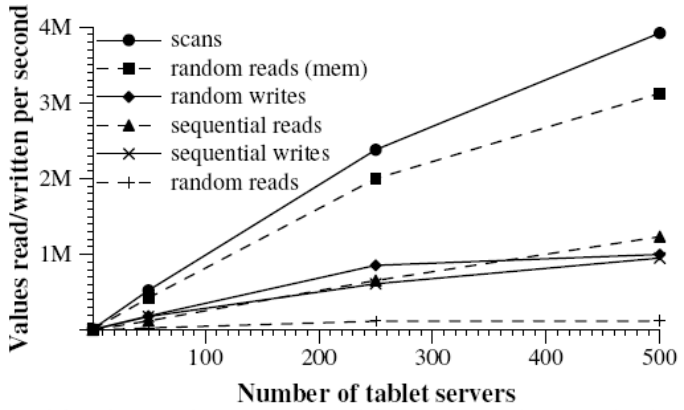


Figure 6: Number of 1000-byte values read/written per second. The table shows the rate per tablet server; the graph shows the aggregate rate.

Application at Google

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

Conclusion

- Satisfies goals of high-availability, high-performance, massively scalable data storage.

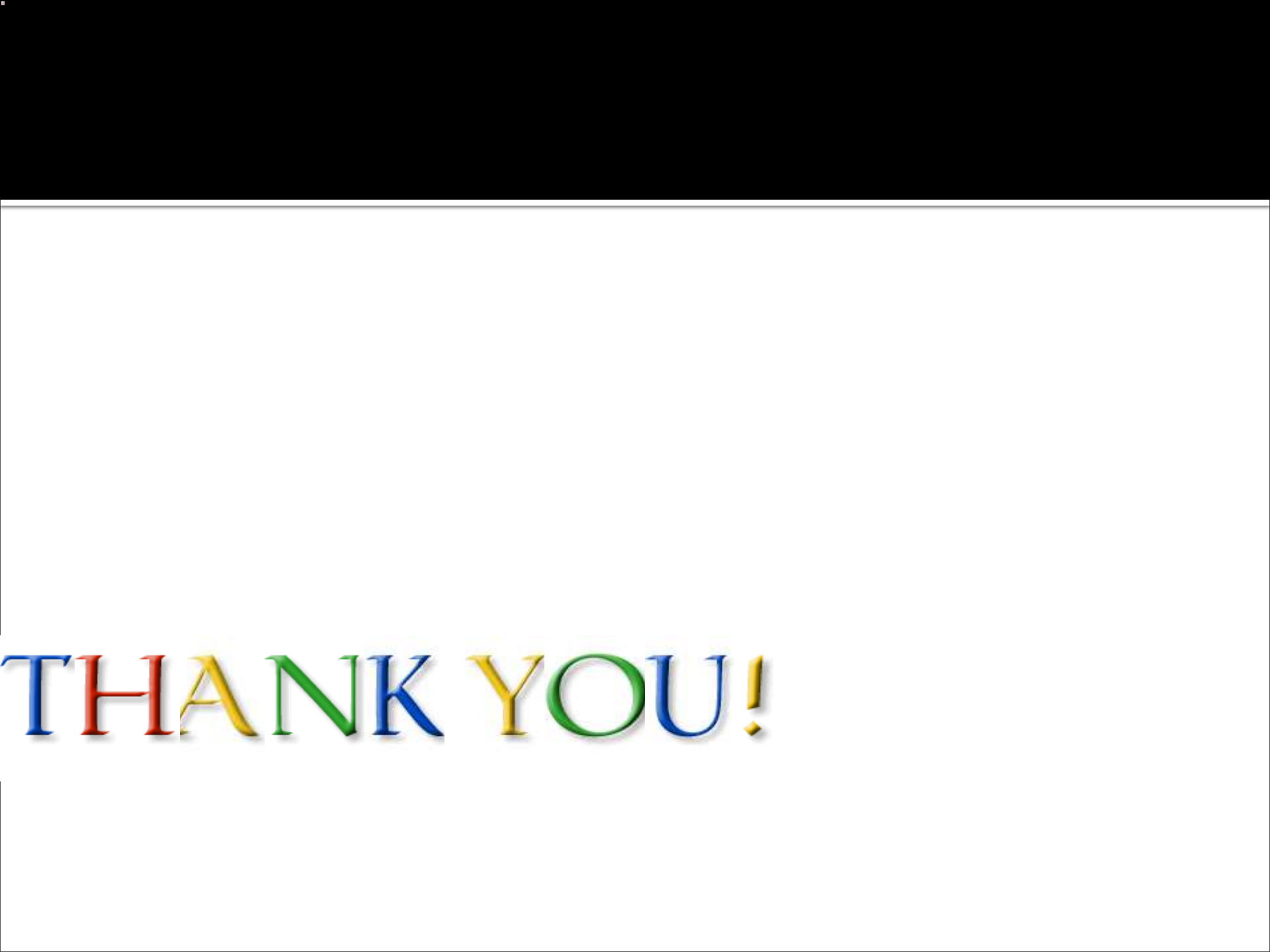
  -It has been successfully deployed in real apps (Personalized Search, Orkut, GoogleMaps, …)
- Successfully used by various Google products (>60).
- It's a distributed systems, designed to store enormous volumes of data.
- Thesesystems can be easily scaled to accommodate peta-bytes of data across thousands of nodes.

References

- *Bigtable: A Distributed Storage System for Structured Data by Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber*

- *http://glinden.blogspot.com/2006/08/google-bigtable-paper.html*

- Dean, J. 2005. BigTable: A Distributed Structured Storage System. University of Washington *CSE Colloquia (*Oct. 2005). http://www.uwtv.org/programs/displayevent.aspx?rID=4188


THANK YOU!

QUESTIONS ?