CSE 544
Principles of Database Management Systems

Magdalena Balazinska

Winter 2009

Lecture 12 – Google Bigtable

References

- **Bigtable: A Distributed Storage System for Structured Data.** Fay Chang et. al. OSDI 2006.

Outline

- Motivation
  - Brief overview of information retrieval
  - Key features of Bigtable
- Bigtable API
- Bigtable architecture
- Bigtable performance and discussion

Different Types of Data

- **Structured data**
  - All data conforms to a schema. Ex: business data
- **Semistructured data**
  - Some structure in the data but implicit and irregular
  - Ex: resume, ads
- **Unstructured data**
  - No structure in data. Ex: text, sound, video, images

Information Retrieval (IR)

- **Goal: search collection of text documents**
- Field exists since the 1950's
- Field evolved independently of databases
- Renewed interest thanks to the Web
  - Traditional IR techniques geared toward relatively small data collections and expert users
  - Web has millions of documents and non-expert users

Document Search

- Two types of queries, called *searches*
- **Boolean query**: recipe AND (beef OR stew)
  - Result: all docs with word “recipe” and word “beef” or word “stew”
- **Ranked query**: “recipe beef stew”
  - Result**: list of docs ranked by relevance to query**

- Different from precise RDBMS queries

Success Criteria

- **Precision**
  - Percentage of retrieved docs that are relevant to query
- **Recall**
  - Percentage of relevant docs in the database that are returned in response to a query
- **Goal**: **high precision and high recall**

IR Framework

- Need a way to represent documents
- Need a way to compare documents

Vector Space Model

- Document vector

doc_id	$word_1$	$word_2$	...	$word_n$
1				
2				
3				

Nb occurrences of $word_n$ in doc_2

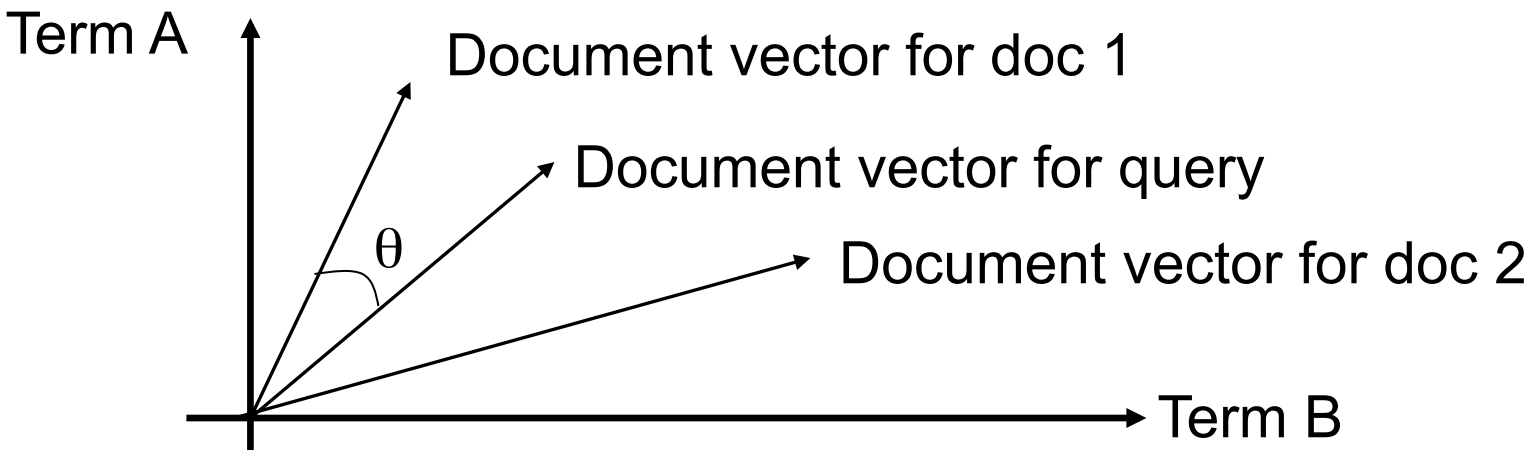
- **Term frequencies (TF)** term j document i: t_{ij}
  - Number of occurrences of term j in document i
  - Intuition: frequent terms are more important

TF/IDF Weighting of Terms

- **Term frequencies (TF)** term j document i: t_{ij}
  - But some terms are frequent in general (e.g., "the")
  - Some terms frequent is some collections
    - Example: "object" and "class" in Java tutorial docs
- **Inverse doc. frequency (IDF): $\log(N/n_j)$**
  - N = nb documents; n_j = nb docs that contain word j
- **w_{ij} = TF * IDF**
- Length normalization: $w_{ij}^* = w_{ij} / (\sqrt{\sum} w_{ik}^2)$
  - Because some documents longer than others

Document Similarity and Ranking

- Assume a t-dimensional space (t is nb terms)



- Similarity
  - Dot product of document vectors
  - **$sim(Q, D_i) = \sum_{terms} q_j^* \ w_{ij}^*$**

Cosine Similarity

- We defined
  - $w_{ij}^{*} = w_{ij} / (\sqrt{\sum} w_{ik}^{2})$
  - $sim(Q, D_i) = \sum_{terms} q_j^{*} w_{ij}^{*}$
- Hence, similarity is equal to
  - $sim(Q,D_i) = (\sum_{terms} q_j w_{ij}) / ((\sqrt{\sum} q_{ik}^{2}) (\sqrt{\sum} w_{ik}^{2}))$
  - $sim(Q,D_i) = Q \bullet D_i / |Q| |D_i|$
  - $\cos(\theta) = Q \bullet D_i / |Q| |D_i|$
- This metric is called **cosine similarity**

Indexing Text Documents

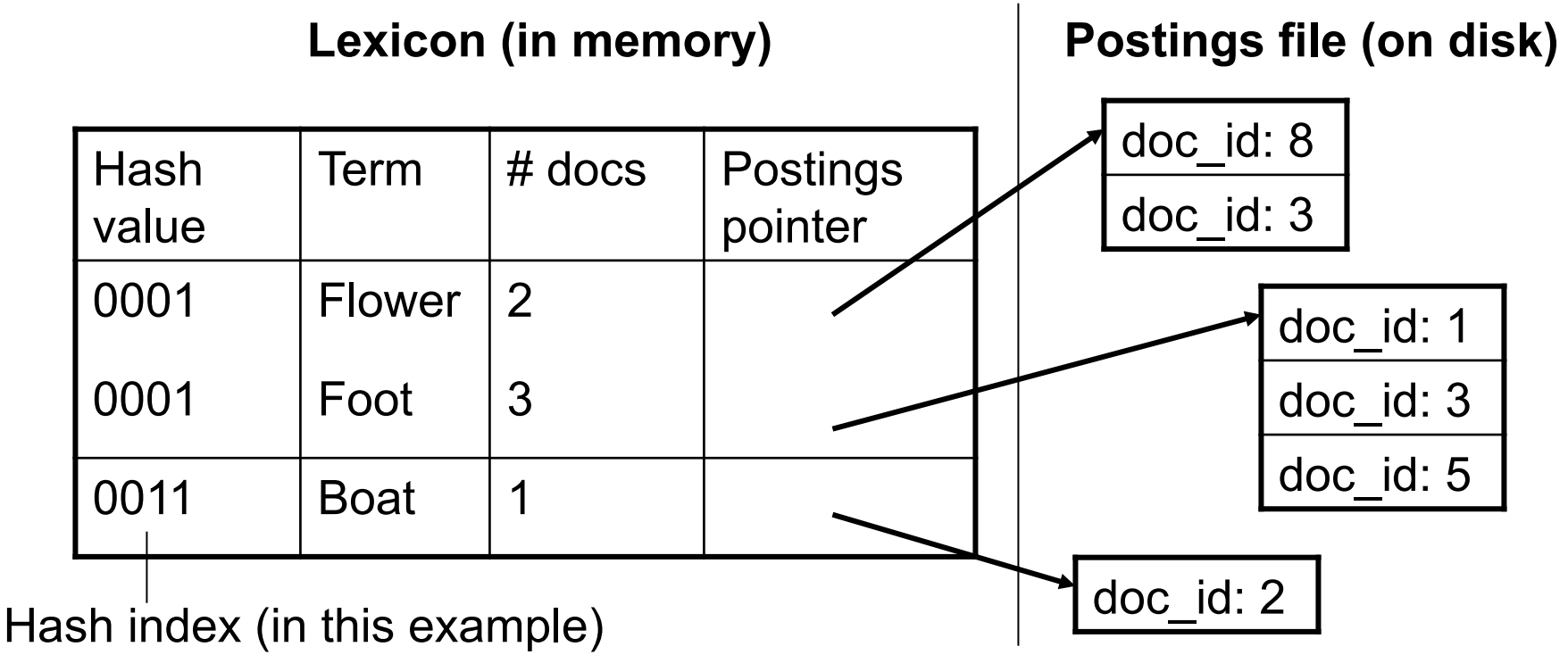
- Goal: efficient eval. of boolean and ranked queries

- Before indexing documents
  - Eliminate stop words
  - Stem all the words
    - Goal: reduce related terms to a canonical form
    - Example: “run”, “running”, “runner” all stem to “run”

Inverted Index

Maps each word to list of docs (**inverted list)** that contain it

Enables fast retrieval of all docs that contain given term



Inverted Index

- Can also add additional info with each document in the inverted list

Flower

Type	Position	...	DocID
title	5	...	8
header	10	...	8
text	23	...	3

Using Index to Answer Queries

- Boolean query: intersect/merge doc lists

- Ranked query:
  - Merge doc lists
  - For each document
  - Compute relevance with respect to query
  - Fetch and return docs in decreasing rank order

Web Search

- Goal: high quality results
- Challenges
  - Large number of documents
  - Large number of terms
  - Malicious content publishers
  - Users are only willing to look at a few results
- Observations
  - Additional info in hyperlink graph
  - Associate anchor text with destination page

Outline

- Motivation
  - Brief overview of information retrieval
  - Key features of Bigtable
- Bigtable API
- Bigtable architecture
- Bigtable performance and discussion

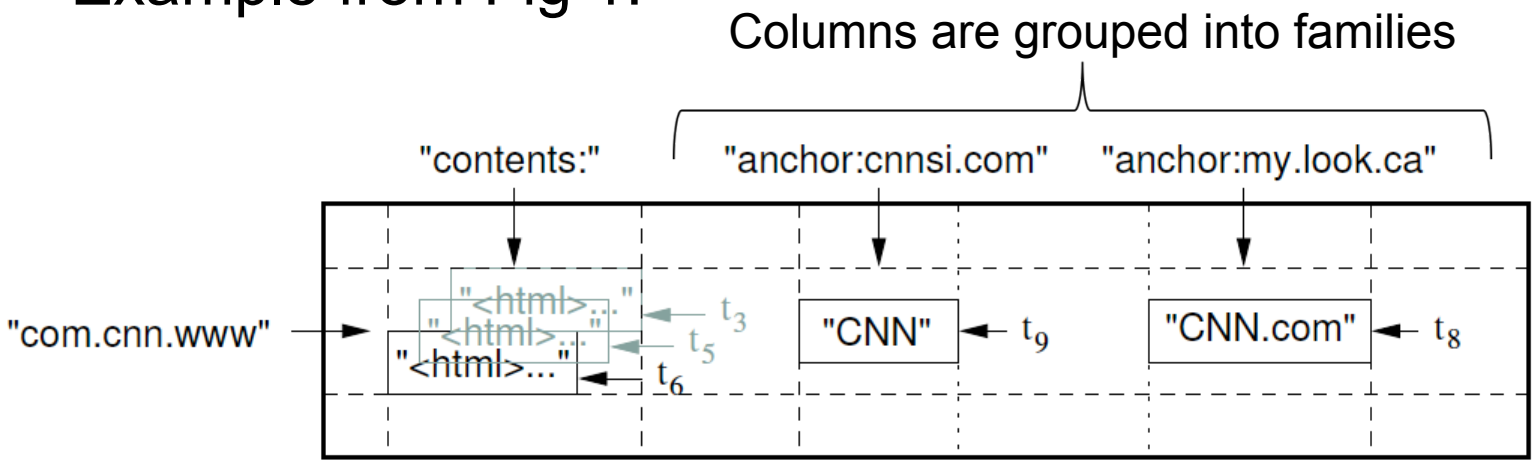
What is Bigtable?

- Distributed storage system
- Designed for structured data
- Designed to scale to thousands of servers
- Designed to store up to several hundred terabytes
- To scale, Bigtable has a limited set of features

Bigtable Data Model

- Sparse, multidimensional sorted map
- `(row:string, column:string, time:int64)` ➔ `string`

- Example from Fig 1:



- How could we use Bigtable for forward/inverted indexes?

Key Features

- Read/writes of data under single row key is atomic
  - Only single-row transactions!
- Data is stored in lexicographical order
  - Improves data access locality
- Column families are unit of access control
- Data is versioned (old versions are garbage collected)
  - Example: most recent three crawls of each page, with times

Outline

- Motivation
  - Brief overview of information retrieval
  - Key features of Bigtable
- Bigtable API
- Bigtable architecture
- Bigtable performance and discussion

API

- Data definition
  - Creating/deleting tables or column families
  - Changing access control rights
- Data manipulation
  - Writing or deleting values
  - Looking up values from individual rows
  - Iterate over subset of data in the table
- Bigtable can serve as input to or output from MapReduce

Outline

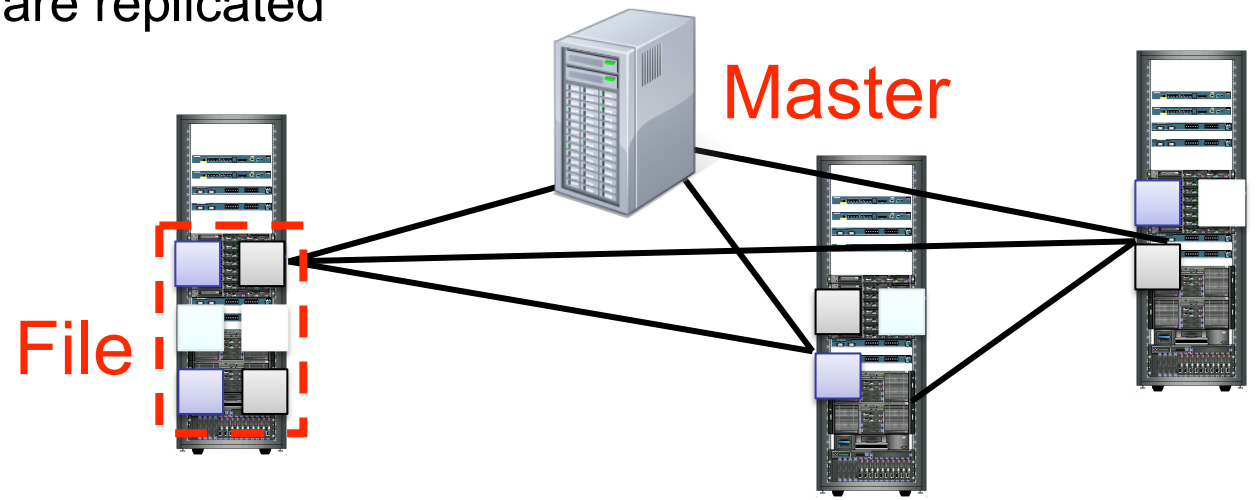
- Motivation
  - Brief overview of information retrieval
  - Key features of Bigtable
- Bigtable API
- Bigtable architecture
- Bigtable performance and discussion

Chubby Lock Service

- In a distributed system, agreement is a problem
  - Different failure scenarios are possible
  - Nodes can have inconsistent views of who is up and who is down
  - Messages can arrive out-of-order at different nodes
- But need agreement to make decisions
- Chubby
  - Provides black-box agreement service through lock abstraction
  - Uses the well-known Paxos algorithm

Google File System

- A file = A series of chunks
  - Size of a chunk ≥ 64MB
  - Append & read only
- Master node
  - Decides chunk placement
  - Decides replica placement
  - Tells clients where to find data
- Fault-tolerance
  - Chunks are distributed
  - Chunks are replicated



Bigtable Building Block: SSTable

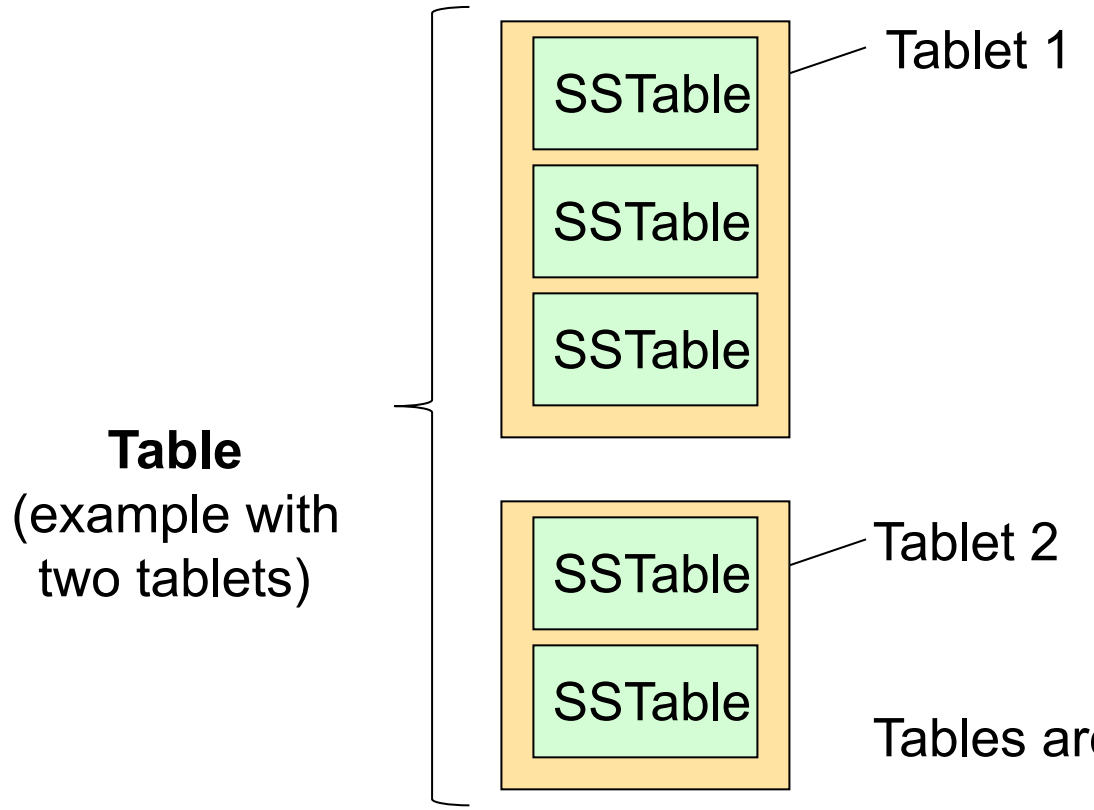
- Persistent map from keys to values
  - Ordered
  - Immutable
  - Keys and values are strings

- API
  - Look up value associated with a key
  - Iterate over all key/value pairs in given range

- Implementation
  - Sequence of blocks + one block index to locate other blocks

A Table in Bigtable: Basics

- A table consists of a set of tablets: Section 5.3
- Each tablet comprises one or more SSTables



- Tablets are stored in GFS
- Tablet servers load data into memory
- Reads are served from memory

Tables are range partitioned

Writing to Tablets

- Remember: SSTables are immutable

- When a write operation arrives at a tablet server, the latter
  - Writes the mutation to a commit log stored in GFS
  - Waits until done
  - Inserts the mutation into an in-memory buffer, the *memtable*
    - The memtable is sorted lexicographically

- To serve reads, the tablet server
  - Merges the SSTables and the memtable into a single view

Loading Tablets

- To load a tablet, a tablet server does the following
- Finds location of tablet through its METADATA (Fig. 4)
- Read SSTables index blocks into memory
  - Recall an SSTable consists of a set of blocks + 1 index block
- Read the commit log since redo point and reconstructs the memtable (the METADATA includes the redo point)

Compaction

- To keep memtables below a threshold
- Minor compaction: convert memtable into an SSTable
- Merging compaction:
  - Read a few SSTables and the memtable
  - Write out a new SSTable
- Major compaction:
  - Replace all SSTables and memtable with a new SSTable

Assigning Tablets to Tablet Servers

- Master
  - Assigns tablets to tablet servers
  - Manages tablet server churn and load imbalances
  - Processes schema changes

- Tablet server
  - Handles read/write to tablets that it has loaded
  - Splits large tablets

- Clients cache tablets locations

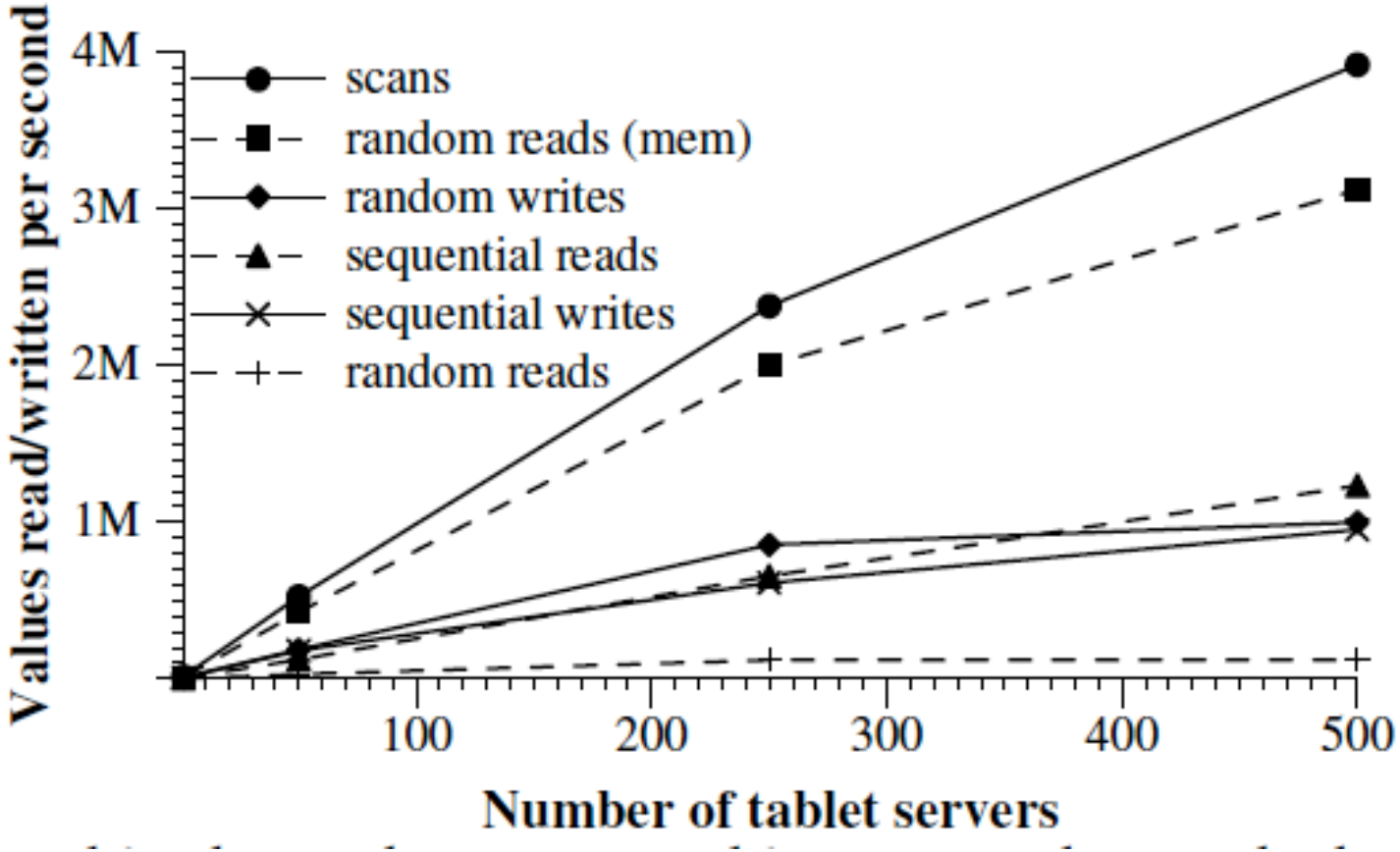
Optimizations

- Vertical partitioning: locality groups
- Compression
- Caching
- Additional indexing: bloom filters
- Commit log optimizations
- Tablet migration optimization

Outline

- Motivation
  - Brief overview of information retrieval
  - Key features of Bigtable
- Bigtable API
- Bigtable architecture
- Bigtable performance and discussion

Performance



Summary

- Bigtable is a distributed system for storing structured data
- Provides high performance and high availability
- Scales incrementally
- Restricted functionality
- Widely used by many applications at Google