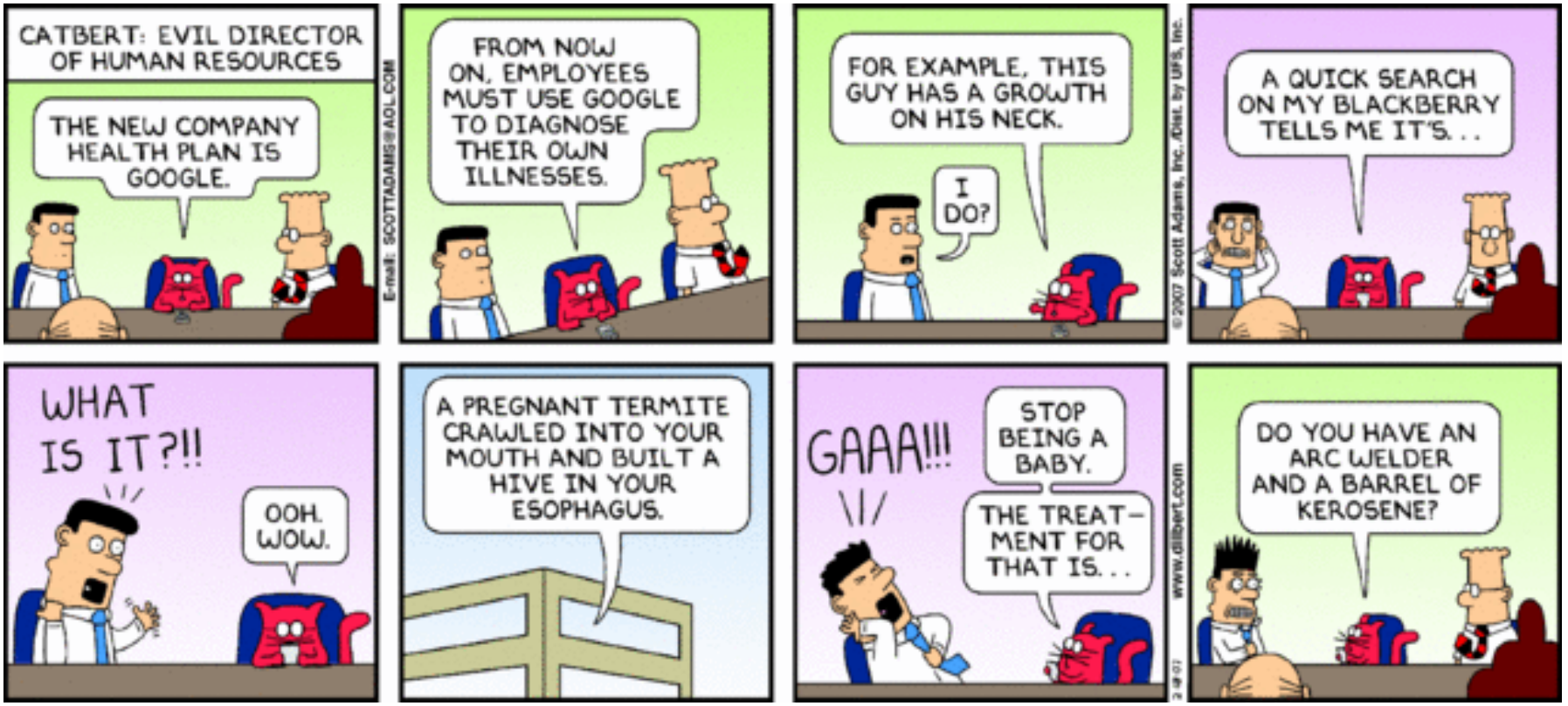
Google BigTable

CS4230

Jay Urbain, Ph.D.

Credits:

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


CATBERT: EVIL DIRECTOR OF HUMAN RESOURCES
THE NEW COMPANY HEALTH PLAN IS GOOGLE.
E-mail: SCOTTADAMS@AOL.COM
FROM NOW ON, EMPLOYEES MUST USE GOOGLE TO DIAGNOSE THEIR OWN ILLNESSES.
FOR EXAMPLE, THIS GUY HAS A GROWTH ON HIS NECK.
I DO?
© 2007 Scott Adams, Inc./Dist. by UFS, Inc.
A QUICK SEARCH ON MY BLACKBERRY TELLS ME IT'S. . .
WHAT IS IT?!!
OOH. WOW.
A PREGNANT TERMITE CRAWLED INTO YOUR MOUTH AND BUILT A HIVE IN YOUR ESOPHAGUS.
GAAA!!!
STOP BEING A BABY.
THE TREAT-MENT FOR THAT IS. . .
www.dilbert.com
DO YOU HAVE AN ARC WELDER AND A BARREL OF KEROSENE?

BigTable

- **BigTable** - compressed, high performance, and proprietary data storage system built on Google File System, Chubby Lock Service, and SSTable.

- Not distributed outside Google, although access to it as part of its Google App Engine.

- Highly Scalable
  - *Petabytes* across *thousands* of commodity servers.
  - 1 PB = 1,000,000,000,000,000 B = 1000^5 B = 10^{15} B
  - 1M GB!

- Used by diverse applications with varied demands with respect to data size and latency:
  - Web indexing, search, Google Earth, Google Reader, Google Maps, Google Code Hosting, Google Finance, Google Analytics, etc.

BigTable Design Goals

- Wide applicability
  - Batch processing jobs
  - Latent sensitive serving of data to end users
- Scalability
  - From very large to mind boggling large
- High performance
  - Batch and real-time
- High availability
  - Assume failure will occur

BigTable versus RDBMS

- BigTable shares many implementation strategies with high performance database technology:
  - Parallel databases
  - Main-memory databases
  - Column oriented databases

- BigTable does **not** support a full *relational model*.
  - Simple data model
  - Supports dynamic control over data layout and format
  - Allows clients to reason about the locality properties

Similar Software

- Dynamo and SimpleDB – Amazon's BigTable equivalents
- Apache Accumulo - Cell-level access labels
- Apache Cassandra — From Facebook, some relational features
- HBase — BigTable-like support on Hadoop
- LevelDB — Google's embedded key/value store.

Big Table Data Model

- Sparse, distributed, persistent, multidimensional ***sorted map***.
- Associated, arbitrary byte arrays are indexed by ***row, column*** (string values),and ***timestamp*** string values.
- Treats data as ***uninterrupted strings***.
- Multiple versions are indexed by timestamp

(row:string, column:string, time:int64) -> string

Overview

- Timestamp facilitates versioning and garbage collection.
- Tables are optimized for Google File System (GFS) by being split into multiple *tablets*.
- Segments of the table are split along a row chosen such that the tablet will be ~200 MB in size.
- When sizes threaten to grow beyond a specified limit, the tablets are compressed using the **BMDiff** and the **Zippy** compression algorithms (Snappy).

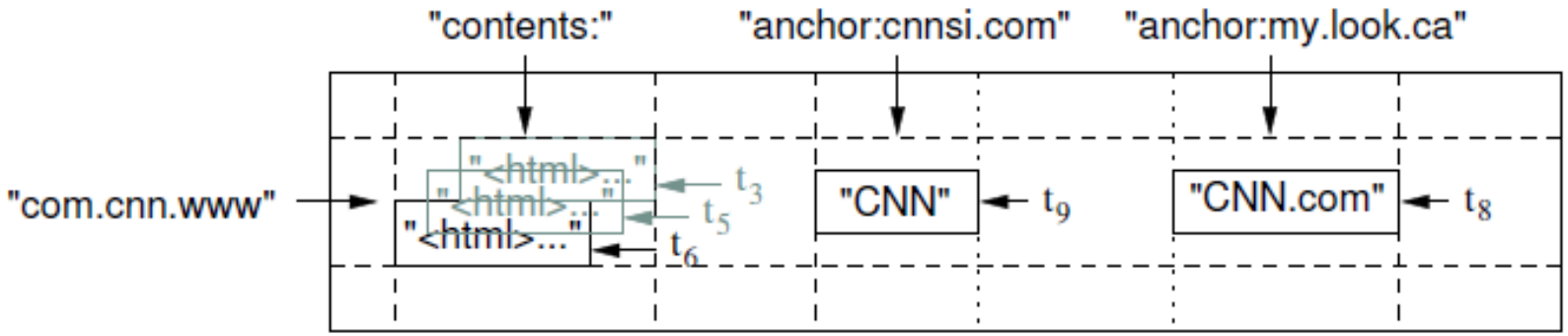
Overview (cont.)

- Locations in the **GFS** of tablets are recorded as database entries in multiple special tablets, which are called "**META1**" tablets.
- **META1** tablets are found by querying the single "**META0**" tablet, which typically resides on a server of its own.
- Like **GFS**'s master server, the **META0** server is not generally a bottleneck since the processor time and bandwidth necessary to discover and transmit **META1** locations is minimal and clients aggressively cache locations to minimize queries.

Data Model

Example: **Webtable**

- Reverse URL as row key
- Aspects of web pages as column names
- Contents under the timestamp when they were fetched



Data Model - Rows

- Row keys are arbitrary strings
  - Up to 64K, 10-100 bytes typical
- Every read or write is atomic
- Maintains data in *lexicographic order* by row key
- Row range for a table is dynamically partitioned into *tablets*
- *Tablets* make up the unit of distribution for load balancing
  - Reads of short row ranges are efficient and require communication with a small number of machines

Data Model - Rows

- Exploit lexicographic ordering and tablet partitioning to control locality for data access.
- Storing pages from the same domain near each other can make some host and domain analysis more efficient.
- Example for Web table:
  - Pages in the same domain are grouped together into contiguous rows by reversing the hostname components of the URLs. I.e.,

    Use *com.google.maps/index.html* versus *maps.google.com/index.html*.

Data Model - Columns

Column families

- Column keys are grouped into sets called *column families.*
- Column families form the *basic unit of access control.*
- All data stored within a column family is usually the same type.
  - Data is compressed in the same column family together.
- Column family must be created before data can be stored under any column key in that family.
- After a *family* has been created, column keys within the family can be used.

Data Model – Column Family Design

Column family design intentions:

- The number of distinct column families in a table is small (hundreds at most)
  - Note: A table may have an unbounded number of columns
- Families rarely change during operation
- Column key naming: *family:qualifier*
- Example: *anchor:*
  - Each column key in the anchor family represents a single HTML anchor
  - Example: *anchor:cnnsi.com*
  - Qualifier is the name of the referring site.

Data Model – Timestamps

Each cell in Bigtable can contain multiple versions of the same data, indexed by timestamp.

- 64-bit integers
- Assigned by Bigtable as realtime values in microseconds or can be assigned by the client.
- Different versions are stored in decreasing timestamp order - so the most recent versions are read first.
- Support provided for per column-family settings for garbage collection. Example:
  - Keep only the 3 most recent versions, or versions within the past 7 days

API

API:

- Admin:
  - Create and delete tables and column families.
  - Change cluster, table, and column family metadata, e.g., access control.
- Client
  - Write or delete values.
  - Read values from individual rows, or iterate over a subset of the data in a table.

API – Writing to Bigtable

Use ***RowMutation*** to perform a series of updates.

- Performs an *atomic mutation* to the sample Webtable: adds one anchor to www.cnn.com and deletes a different anchor.

```
// 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.Set("anchor:www.c-span.org", "CNN");
r1.Delete("anchor:www.abc.com");
Operation op;
Apply(&op, &r1);
```

API – Reading from Bigtable

- Use Scanner to iterate over all anchors in a row.
- Can also iterate over multiple column families.
- Mechanisms for limiting the number of rows retrieved: anchor:*.cnn.com

```
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());
}
```

API – etc.

- Support for single-row transactions.
  - Example: atomic read-modify-write sequences
- *Does not support general transactions across row keys*
- Execution of client-supplied scripts
  - Developed in a language called *Sawzall*
- Can be used with *MapReduce*

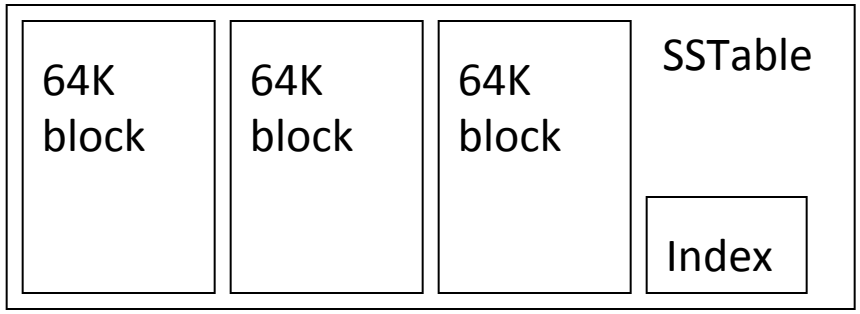
Building Blocks

- Google File System (GFS)
  - Store log and data files.
- A cluster management system
  - Scheduling jobs, managing resources on shared machines, dealing with machine failures, and monitoring machine status.
- Google SSTable file format to store Bigtable data
  - Provides a persistent ordered immutable map from keys to values (arbitrary strings)
- Chubby Lock service

Building Blocks - SSTable

SSTable

- Persistent, immutable, sorted file of key-value pairs
- Used to store Bigtable data within GFS
- Chunks of data plus an index
  - Index is of block ranges (typically 64k), not values
- Ops to look up data and iterate over keys



Building Blocks - Chubby

Chubby

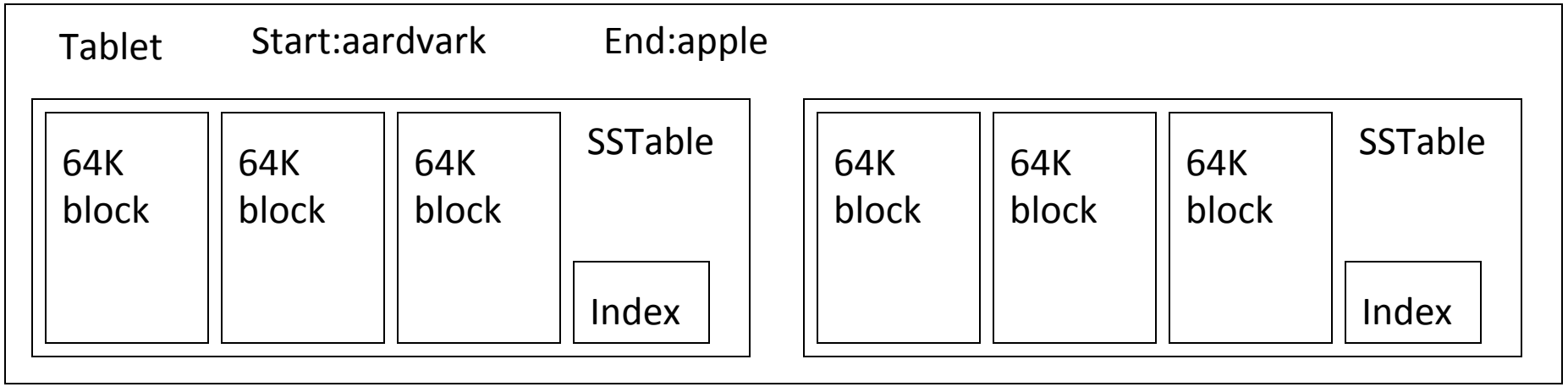
- Distributed {lock/file/name} service
- Maintains namespace of directories and small files
- Five active replicas, one is master, need a majority vote to be active
- Paxos algorithm to keep replicas consistent in the face of failure.
  - **Uses consensus** process of agreeing on one result among a group of participants.
- Coarse-grained locks, can store small amount of data in a lock
- Can lock directory or file
- Caching
- Session management

Google File System

- Large-scale distributed file system.
- Master: responsible for metadata.
- Chunk servers: responsible for reading and writing large chunks of data.
- Chunks replicated on 3 machines, master responsible for ensuring replicas exist.

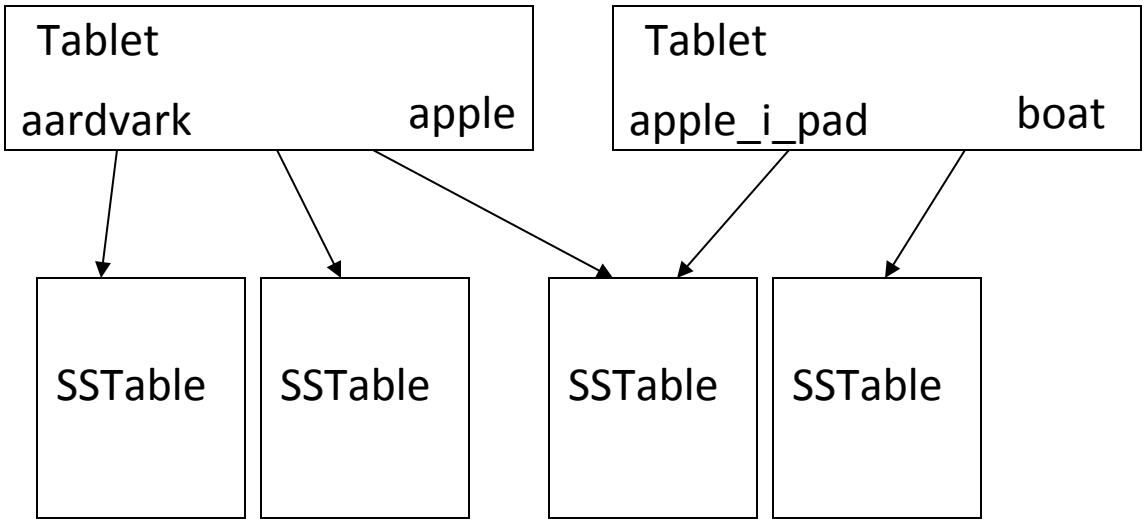
Tablet

- Contains some range of rows of the table
- Built out of multiple SSTables



Table

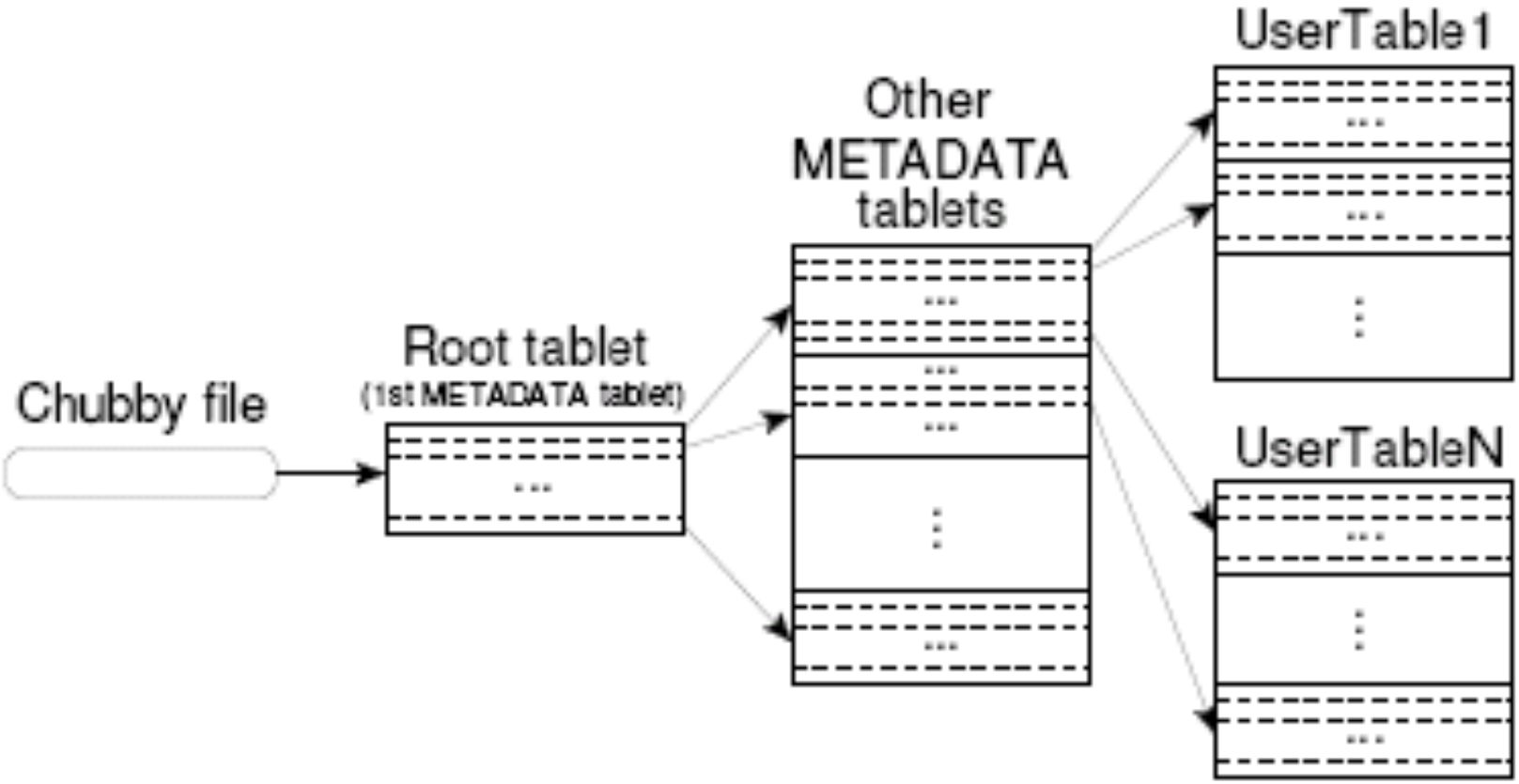
- Multiple tablets make up the table
- SSTables can be shared
- Tablets do not overlap, SSTables can overlap



Implementation - Major components

- Library is linked into every client
- One master server (META0)
  - Assigns tablets to tablet servers
  - Detects addition and expiration of tablet servers
  - Balances tablet-server load
  - Garbage collection of files in the GFS
  - Schema changes
- Many tablet servers (META1)
  - Manages a set up tablets
  - Read/write requests
  - Tablet splits when they become to large (100-200MB)
  - Clients communicate directly with tablet server not master
  - Can be added dynamically
- A Bigtable cluster stores a number of tables, each tablet contains all data associated with a row range.

Finding a tablet



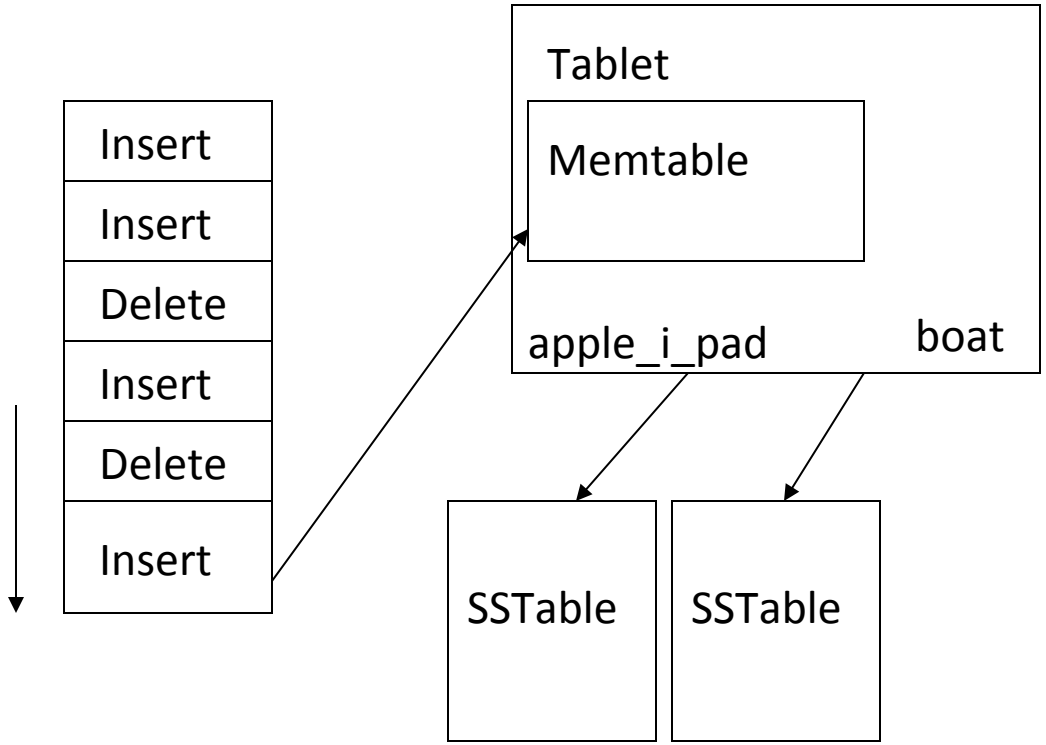
Tablet Serving

- Tablet servers manage tablets, multiple tablets per server. Each tablet is 100-200 MBs
  - Each tablet lives at only one server.
  - Tablet server splits tablets that get too big.

- Master responsible for load balancing and fault tolerance
  - Use Chubby to monitor health of tablet servers, restart failed servers.
  - GFS replicates data. Prefer to start tablet server on same machine that the data is already at

Editing a table

- Mutations are logged, then applied to an in-memory version
- Logfile stored in GFS



Recovering a tablet

1. Tablet server (META1) reads its metadata from the METADATA table.
2. Metadata contains the list of *SSTables* that comprise a table and a set of redo pointers into any commit logs.
3. The server reads the indices of the SSTables into memory, and reconstructs the *memtable* by applying all of the updates that have committed since the redo points.
4. When a write op arrives at a table server, the server checks that it is well-formed and that the sender is authorized to perform the mutation.
5. Authorization is checked with Chubby
6. After changes are committed, its contents are inserted into the memtable.

Compactions

- Minor compaction – convert the memtable into an SSTable
  - Reduce memory usage
  - Reduce log traffic on restart
- Merging compaction
  - Reduce number of SSTables
  - Good place to apply policy “keep only N versions”
- Major compaction
  - Merging compaction that results in only one SSTable
  - No deletion records, only live data

Locality Groups

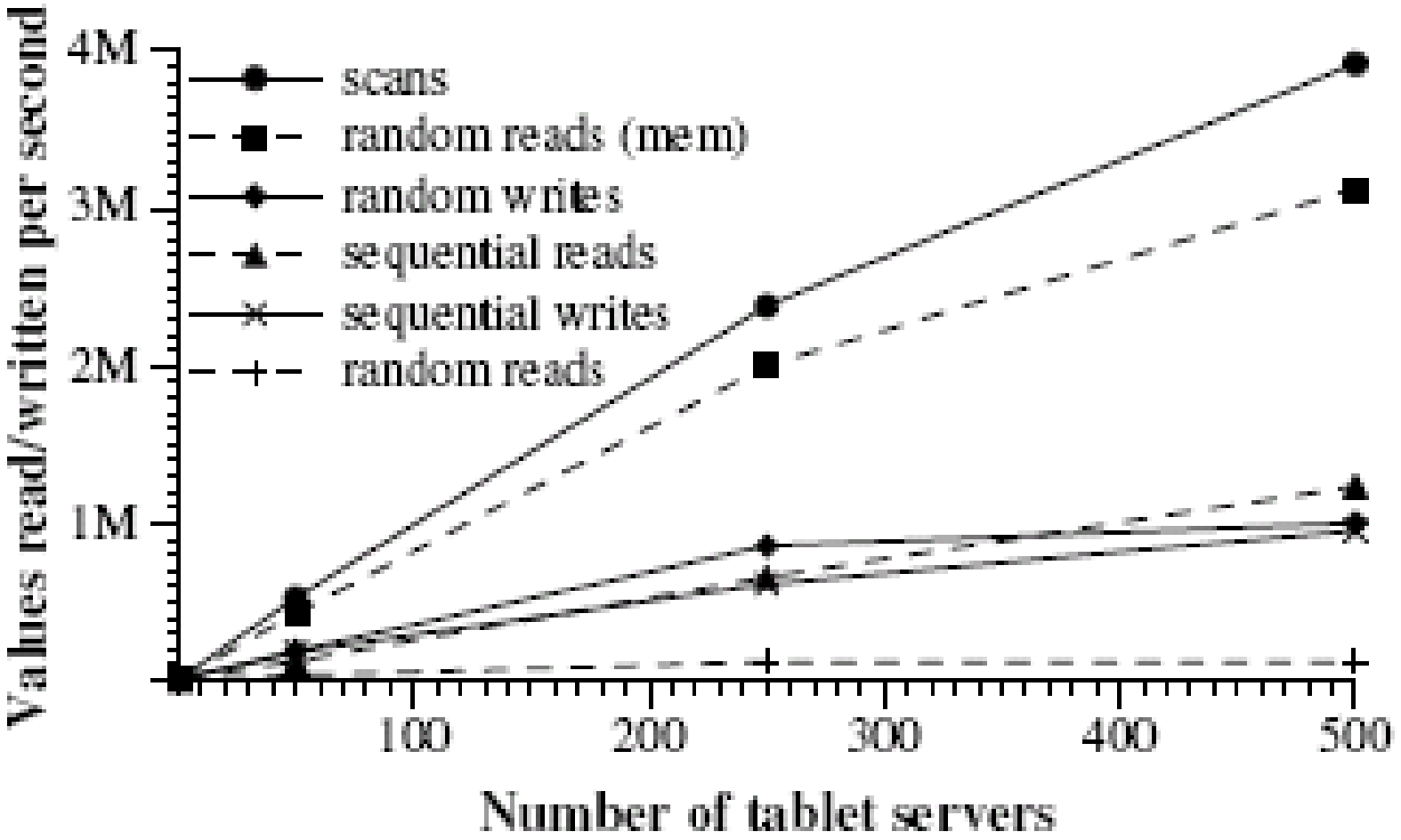
- Group column families together into an *SSTable*
  - Avoid mingling data, i.e., page contents and page metadata
  - Can keep some groups all in memory
- Can compress locality groups
- Bloom Filters on locality groups – avoid searching *SSTable*

Microbenchmarks

Number of 1000-byte values read/written per second.
Table shows the rate per tablet server.

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

Number of 1000-byte values read/written per second. 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

Lessons learned

- Interesting point- only implement some of the requirements, since the last is probably not needed
- Many types of failure possible
- Big systems need proper systems-level monitoring
- Value simple designs