CS 5450

Chubby and Bigtable

Slides from Cristina Nita-Rotaru

Bigtable

- Distributed storage system for managing structured data such as:
  - URLs: contents, crawl metadata, links, pagerank …
  - Per-user data: preferences, recent search results …
  - Geolocations: physical entities (shops, restaurants, etc.), roads, satellite image data, user annotations, …
- Used for many Google applications
  - Web indexing, personalized Search, Google Earth, Google Analytics, Google Finance, … and more

HBase is the open-source version

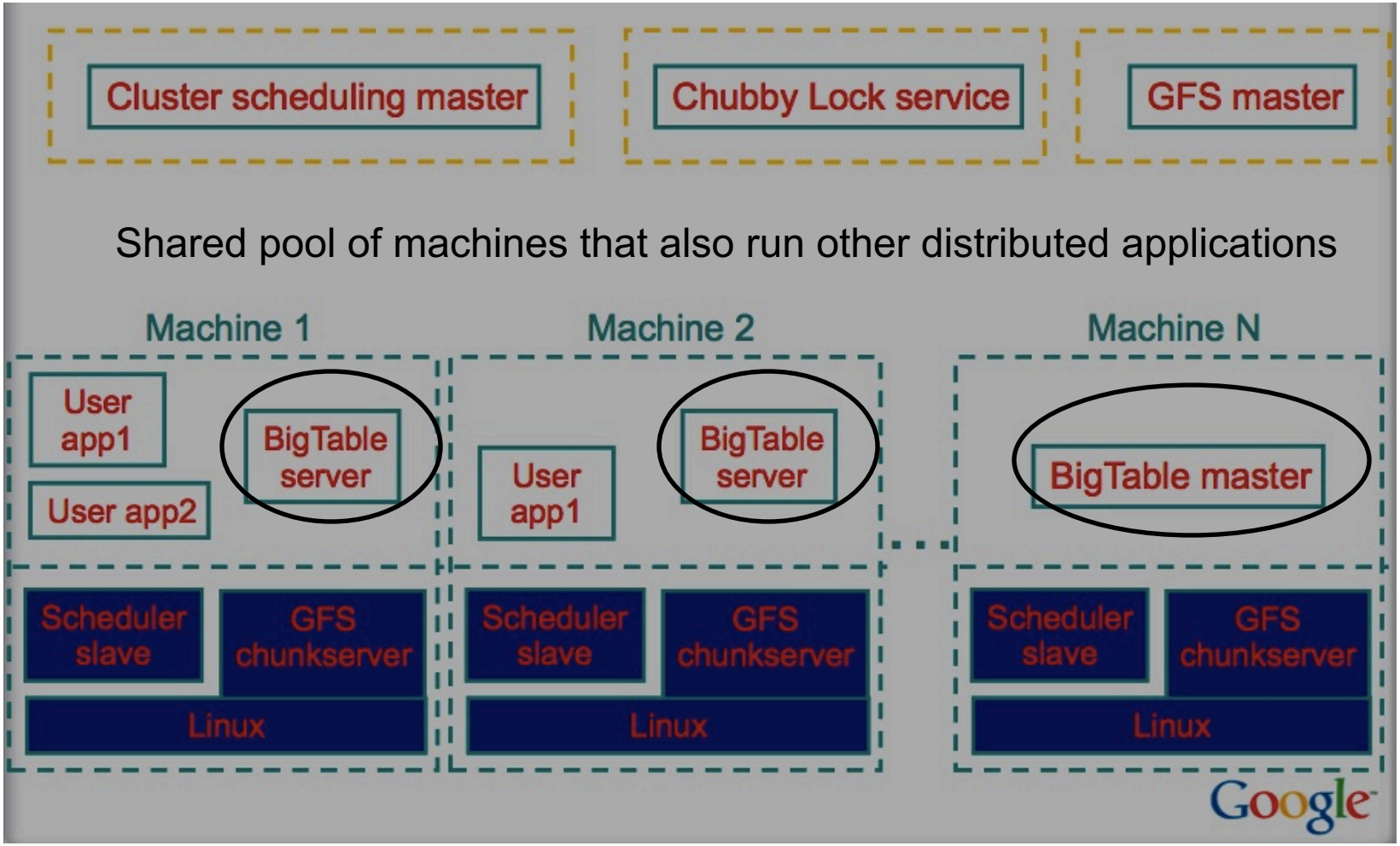
Scalability Requirements

- Petabytes of data distributed across thousands of servers
  - Hundreds of millions of users
  - Billions of URLs, many versions/page
  - Thousands or queries/sec
  - 100TB+ of satellite image data

Goals

- Asynchronous processes continuously update different pieces of data
- Examine data changes over time: eg, contents of a web page over multiple crawls
- Requirements
  - Simpler model that supports dynamic control over data and layout format
  - Very high read/write rates (millions ops per second)
  - Efficient scans over all or subsets of data
  - Efficient joins of large one-to-one and one-to-many datasets

Typical Google Cluster



Building Blocks

- Google File System (GFS)
  - Stores persistent data (SSTable file format)
- Scheduler
  - Schedules jobs onto machines
- Chubby
  - Lock service: distributed lock manager, master election, location bootstrapping
- MapReduce (optional)
  - Data processing
  - Read/write Bigtable data

Chubby

- A coarse-grained lock service
  - Provides a means for distributed systems to synchronize access to shared resources
- Looks like a file system
  - Reads and writes are whole-file
  - Supports advisory reader/writer locks
  - Clients can register for notification of file update
- Known, available, highly reliable location to store small amount of metadata
  - "Root" for bootstrapping distributed data structures

Files as Locks

Files can have several attributes:

- The contents of the file is one (primary) attribute
- Owner of the file
- Permissions
- Date modified
- Whether the file is locked or not

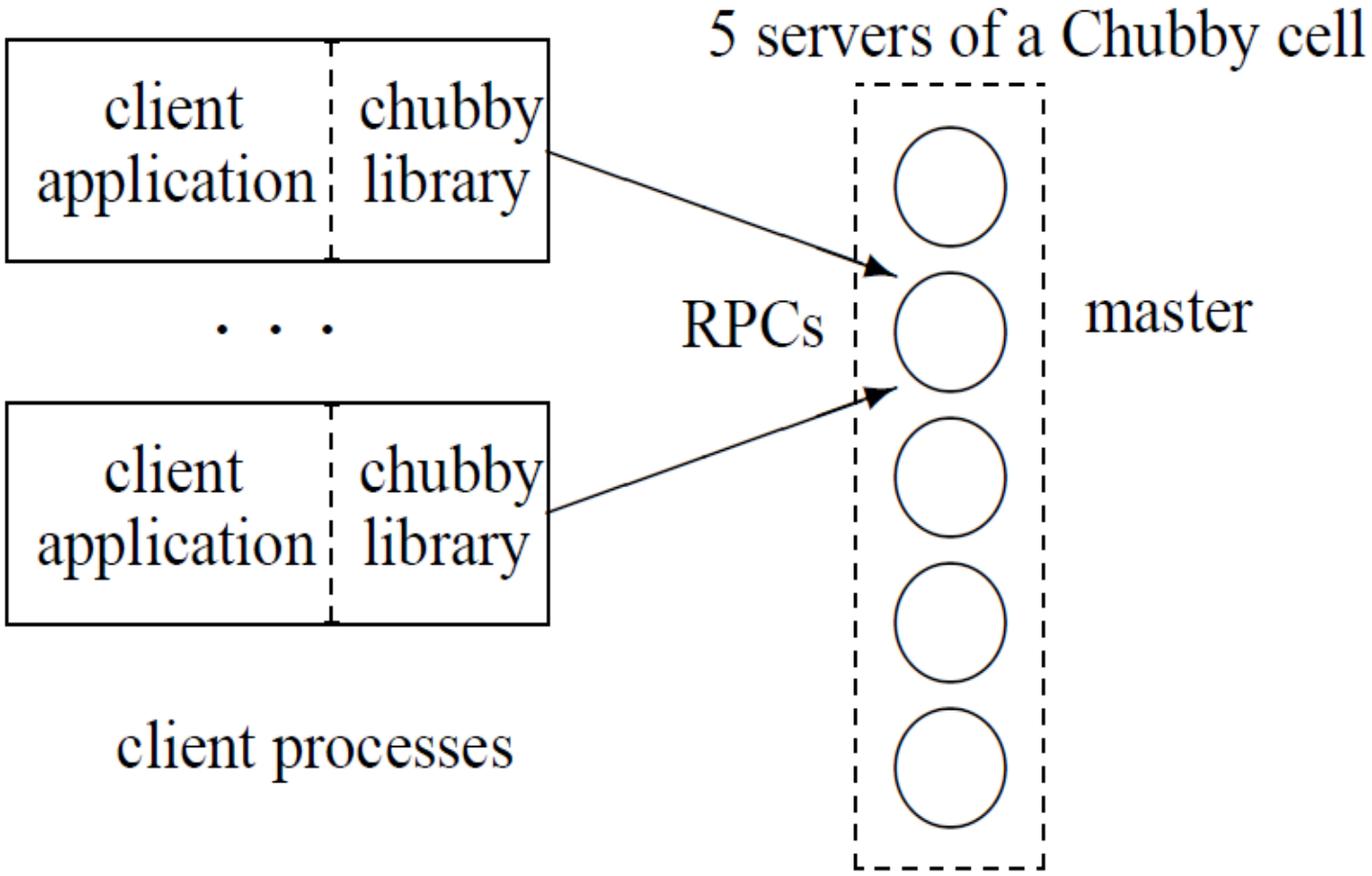
How Chubby Is Used at Google

- GFS: master election
- Bigtable: master election, client discovery, table service locking
- Partition workloads
- Name service because of its consistent client caching
  - Replacement for DNS inside Google

Example: Primary / Master Election

```
open("write mode");
If (successful) {
        // primary
   SetContents("identity");
}
Else {
   // replica
   open ("read mode", "file-modification event");
   when notified of file modification:
           primary= GetContentsAndStat();
}
```

Chubby Cell


5 servers of a Chubby cell
client application
chubby library
client application
chubby library
RPCs
master
client processes

Consensus in Chubby

- Chubby cell is usually 5 replicas
  - 3 replicas must be alive for cell to work
  - Tolerates 2 failures
- Replicas must agree on their own master and official lock values
  - Replicas promise not to try to elect new master for at least a few seconds (“master lease”)
- Uses Paxos for consensus
  - Memory for individual “facts” (variable-value bindings) in the entire distributed system

Client Updates to Chubby

- All replicas are listed in DNS
- Clients find master through DNS
  - Contacting replica causes redirect to master
- All client updates go through master
- Master updates official database; sends copy of update to replicas
  - Majority of replicas must acknowledge receipt of update before master writes its own value

Chubby APIs

- Open()
  - Mode: read/write/change ACL; Events; Lock-delay
  - Create new file or directory?
- Close()
- GetContentsAndStat(), GetStat(), ReadDir()
- SetContents(): set all contents; SetACL()
- Delete()
- Locks: Acquire(), TryAcquire(), Release()
- Sequencers: GetSequencer(), SetSequencer(), CheckSequencer()

Client Caching

- Clients cache all file content w/ strict consistency
  - Lease-based
  - Master invalidates cached copies upon a write request
- Client must respond to keep-alive message from server at frequent intervals
  - Keep-alive messages include invalidation requests
  - Responding to keep-alive implies acknowledgement of cache invalidation
- Modification only continues after all caches invalidated or keep-alive times out

Scalability

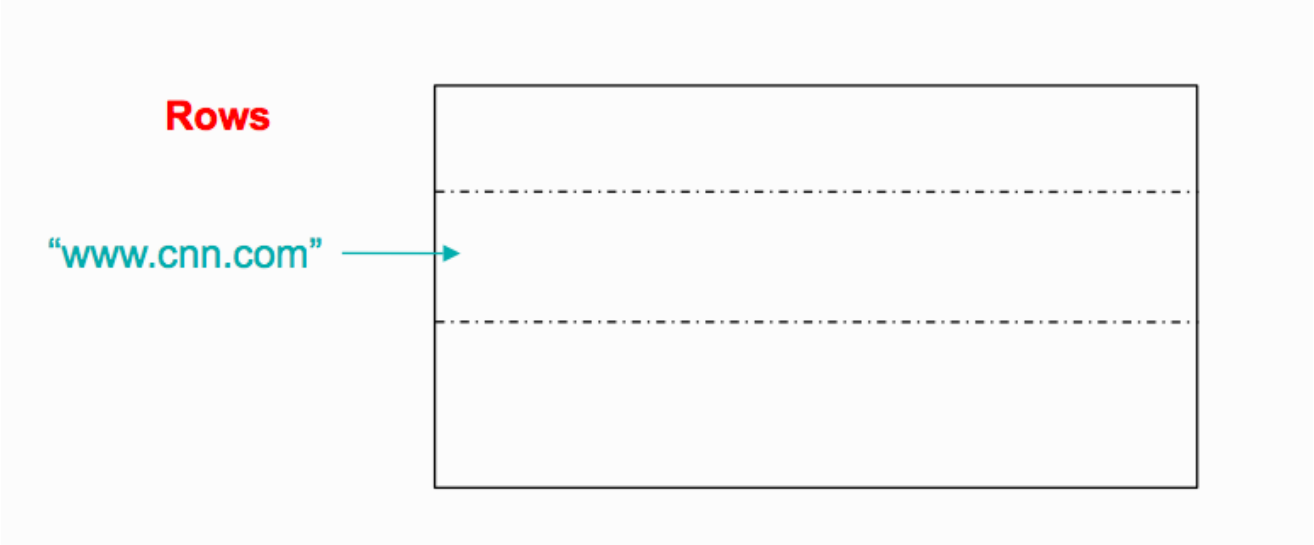
- 90K+ clients communicate with a single Chubby master (2 CPUs)
- System increases lease times from 12 sec up to 60 secs under heavy load
- Clients cache virtually everything
- Data is small – all held in RAM (as well as disk)

Bigtable Data Model

- A sparse, distributed, persistent multi-dimensional sorted map
- (row, column, timestamp) -> cell contents
  - Rows, column are arbitrary strings

Rows

- Arbitrary string
- Access to data in a row is atomic
  - Row creation is implicit upon storing data
  - Ordered lexicographically

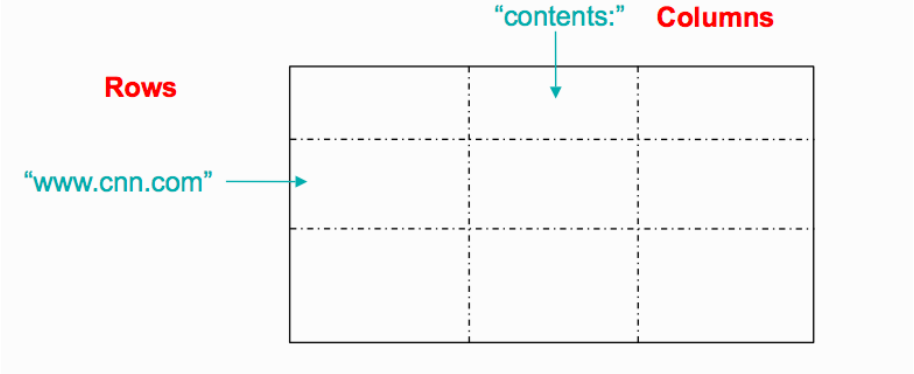


Rows

- Rows close together lexicographically usually on one or a small number of machines
- Reads of short row ranges are efficient and typically require communication with a small number of machines
- Can exploit lexicographic order by selecting row keys so they get good locality for data access
  - Example: edu.cornell.tech, edu.cornell.cs … instead of tech.cornell.edu, cs.cornell.edu

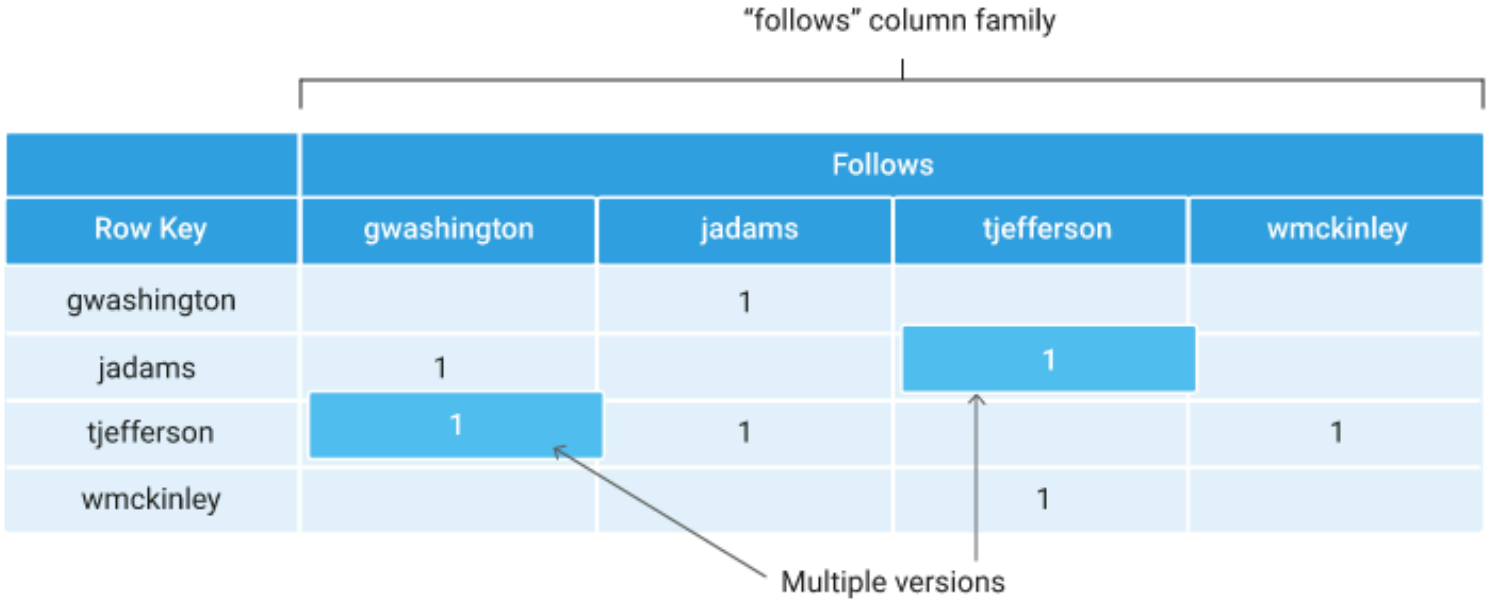
Columns

- Two-level name structure: family: qualifier
- Family is the unit of access control
  - Has associated type information
- Qualifier gives unbounded columns
  - Additional levels of indexing, if desired



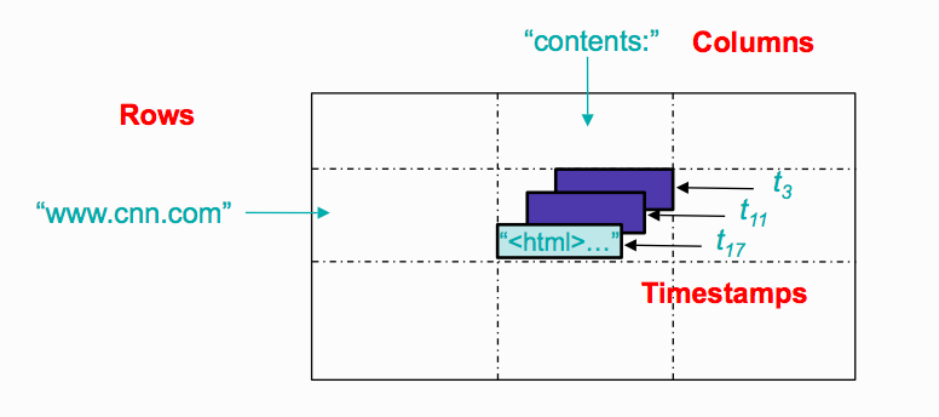
Example

https://cloud.google.com/bigtable/docs/overview



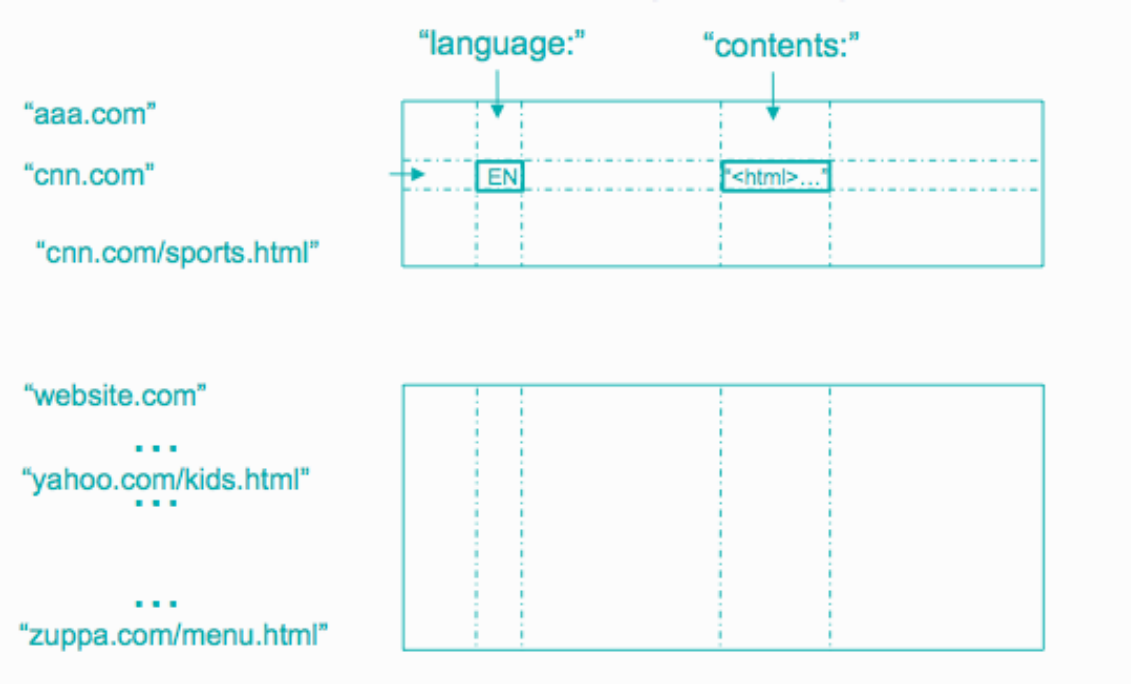
Timestamps

- Store different versions of data in a cell
  - New writes default to current time, but timestamps for writes can also be set explicitly by clients
- Lookup options
  - Return most recent K values
  - Return all values
- Column families can be marked w/ attributes
  - Retain most recent K values in a cell
  - Keep values until they are older than K seconds



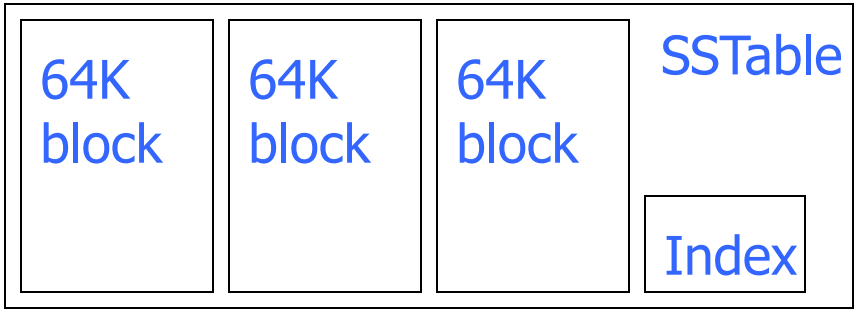
Tablet

- The row range for a table is dynamically partitioned, each range is called a tablet
- Tablet is the unit for distribution and load balancing



Storage: SSTable

- Immutable, sorted file of key-value pairs
- Can be completely mapped into memory (option)
- Chunks of data plus an index
  - Index is of block ranges, not values
  - Index is loaded into memory when SSTable is open



Tablet vs. SSTable

Tablet is built out of multiple SSTables

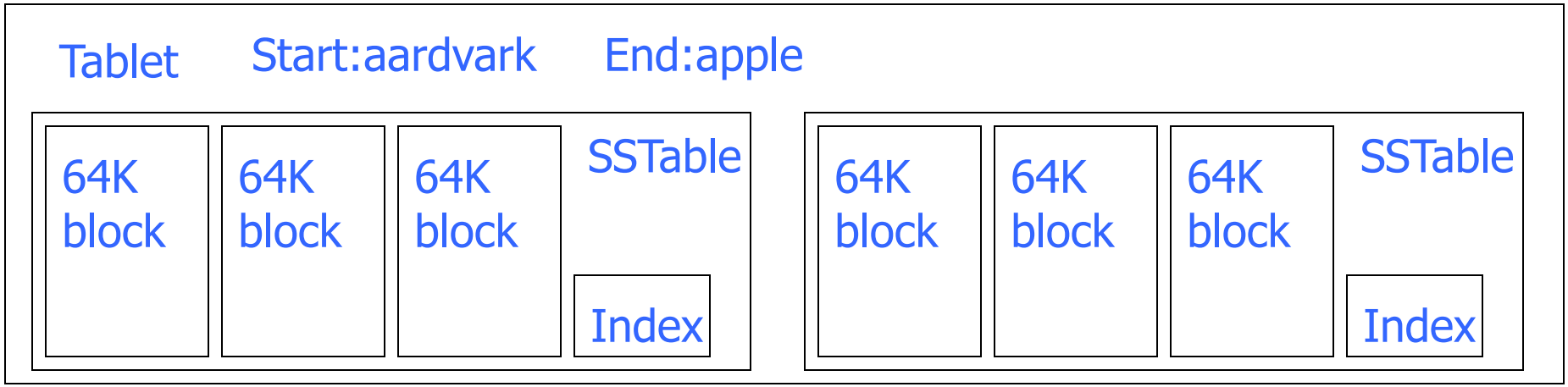
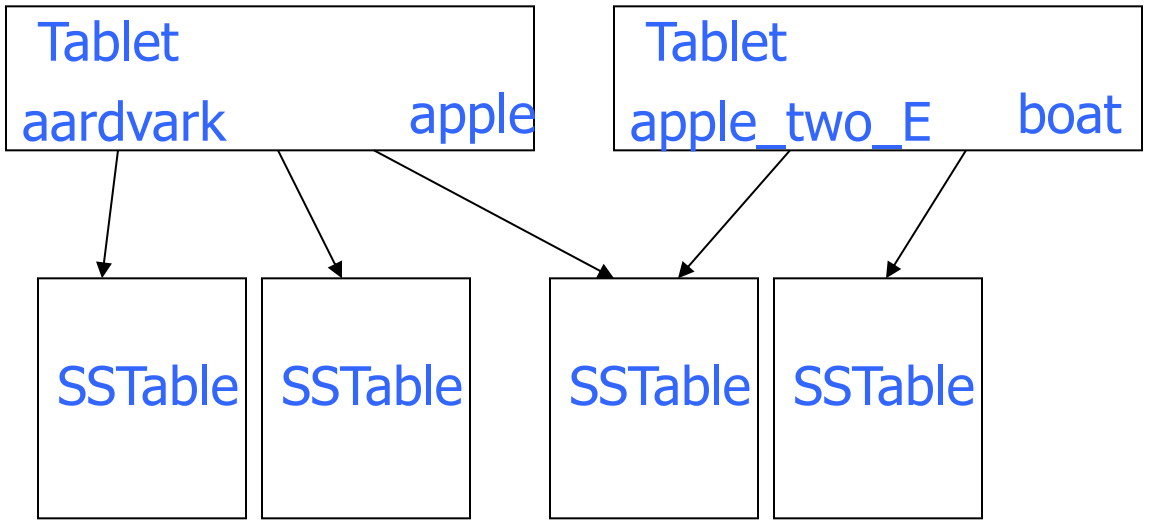
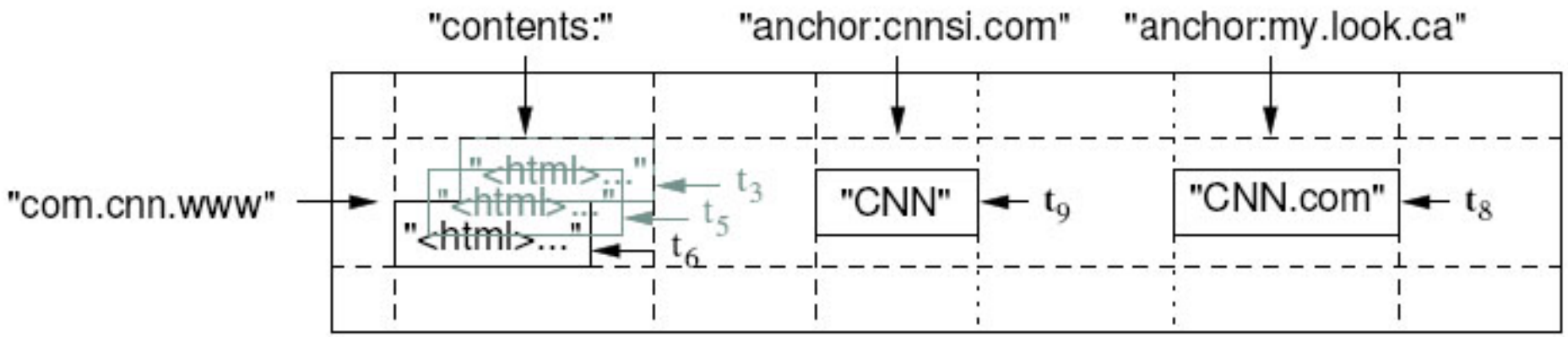


Table vs. Tablet vs. SSTable

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



Example: WebTable

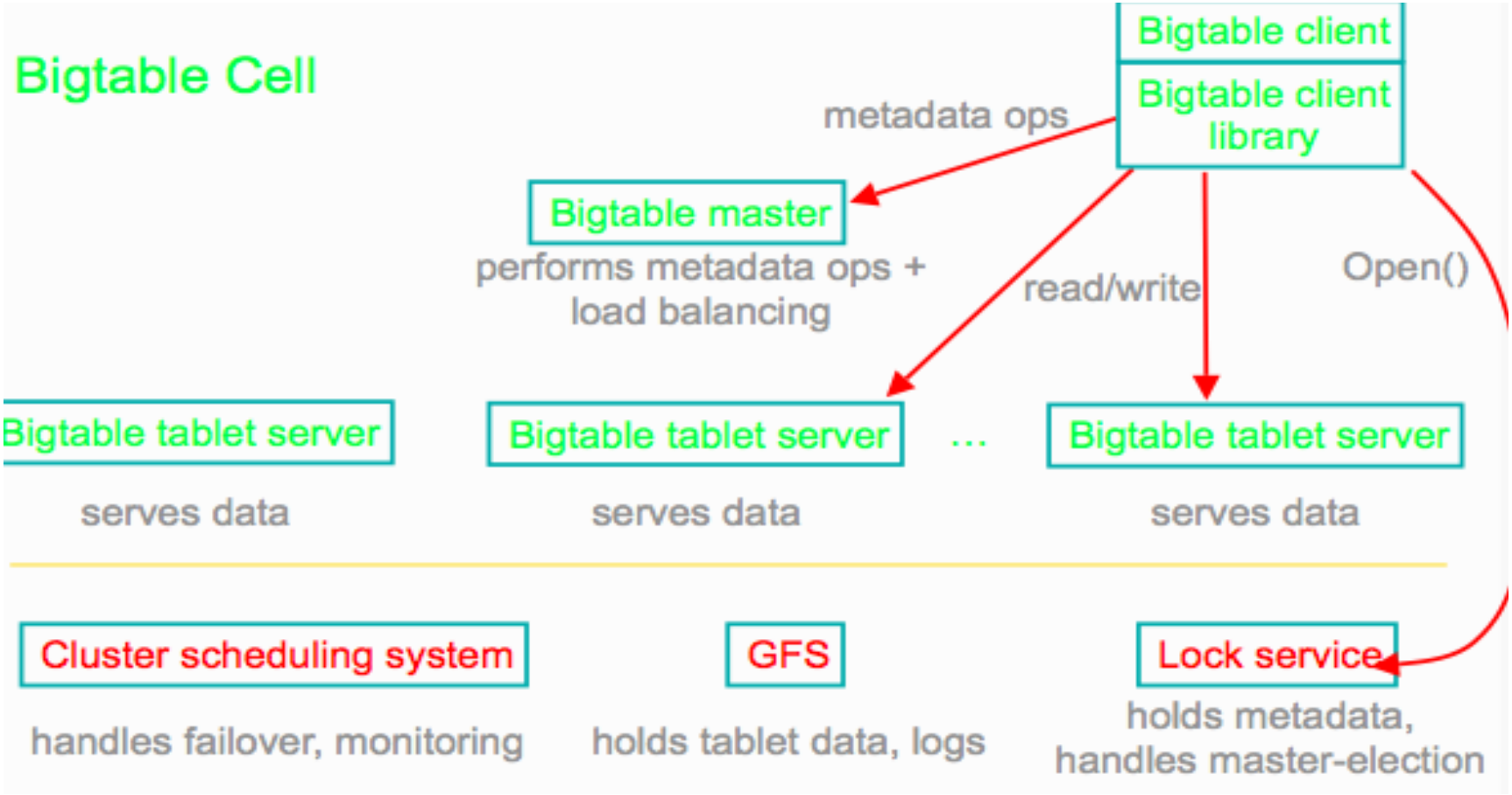


- Want to keep copy of a large collection of web pages and related information
- Use URLs as row keys
- Various aspects of web page as column names
- Store contents of web pages in the "contents:" column under the timestamps when they were fetched

Implementation

- Bigtable library linked into every client
- One master server responsible for:
  - Assigning tablets to tablet servers, balancing load
  - Detecting addition and expiration of tablet servers
  - Garbage collection
  - Handling schema changes, eg, table and column family creation
- Many tablet servers, each of them:
  - Handles read and write requests to its table
  - Splits tablets that have grown too large
- Clients communicate directly with tablet servers

Deployment


Bigtable Cell
Bigtable client
Bigtable client library
metadata ops
Bigtable master
performs metadata ops + load balancing
read/write
Open()
Bigtable tablet server
Bigtable tablet server
...
Bigtable tablet server
serves data
serves data
serves data
Cluster scheduling system
GFS
Lock service
handles failover, monitoring
holds tablet data, logs
holds metadata, handles master-election

Tablet Servers

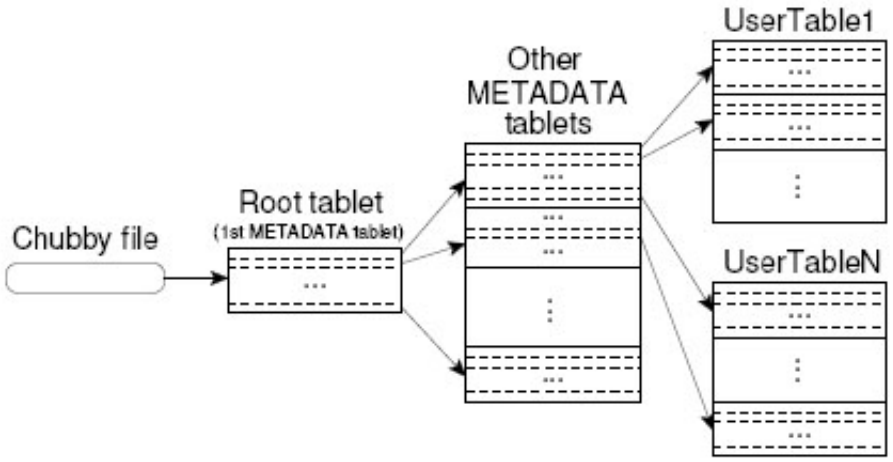
- Each tablet server responsible for 10 – 1000 tablets (usually about 100)
- Fast recovery
  - 100 machines each pick up 1 tablet for failed server
- Fine-grained load balancing
  - Migrate tablets away from overloaded machine
  - Master makes load-balancing decisions

Tablet Location

- How do clients find the right machine for a row?
  - Find tablet whose row range covers the target row
  - Tablets move around from server to server
- METADATA

  Key: table id + end row, Data: location
- Aggressive caching and prefetching

  at client side



Tablet Assignment

- Each tablet assigned to one tablet server at a time
- Master server
  - Keeps track of the set of live tablet servers and current assignments of tablets to servers
  - Keeps track of unassigned tablets
- When a tablet is unassigned, master assigns the tablet to a tablet server with sufficient room
- Master uses Chubby to monitor health of tablet servers and restart/replace failed servers

Tablet Assignment with Chubby

- Tablet server registers itself with Chubby by getting a lock in a specific directory of Chubby
- "Lease" on lock must be renewed periodically
- Master monitors this directory to find which servers exist/are alive
  - If server not contactable or has lost lock, master grabs lock and reassigns tablets
- Prefer to start tablet server on same machine that the data is already at
  - Data replicated by GFS

Bigtable API

- Metadata operations
  - Create/delete tables, column families, change metadata
- Writes (atomic)
  - Set(): write cells in a row
  - DeleteCells(), DeleteRow()
- Read
  - Scanner: read arbitrary cells in a bigtable
    - Each row read is atomic
    - Can restrict returned rows to a particular range
    - Can ask for just data from 1 row, all rows, etc.
    - Can ask for all columns, just certain families, or specific columns

Refinements: Locality Groups

- Can group multiple column families into a locality group
  - Separate SSTable is created for each locality group in each tablet
- Segregating columns families that are not typically accessed together enables more efficient reads
  - In WebTable, page metadata can be in one group and contents of the page in another group

Refinements: Compression

- Many opportunities for compression
  - Similar values in the same row/column at different timestamps
  - Similar values in different columns
  - Similar values across adjacent rows
- Two-pass custom compression scheme
  - First pass: compress long common strings across a large window
  - Second pass: look for repetitions in small window
- Speed emphasized, but good (10x) space reduction

Refinements: Bloom Filters

- Read operation has to read from disk when desired SSTable is not in memory
- Reduce number of accesses by specifying a Bloom filter
  - Allows to ask if a SSTable might contain data for a specified row/column pair
  - Small amount of memory for Bloom filters drastically reduces the number of disk seeks for read operations
  - Results in most lookups for non-existent rows or columns not needing to touch disk

Limitations

- No transactions
- Does not support full relational data model
- Achieved throughput is limited by GFS

- Difficult to use for applications that (1) have complex, evolving schemas; (2) want strong consistency in the presence of wide-area replication