Distributed Systems [Fall 2012]

Lec 19: Back to the Real World: Chubby and Bigtable

Slide acks: Shimin Chen, Mohsen Taheriyan

(http://www.cs.cmu.edu/~chensm/Big_Data_reading_group/slides/shimin-chubby.ppt, http://www-scf.usc.edu/~csci572/2011Spring/presentations/Taheriyan.pptx)

Paxos (Reminder)

- Consensus algorithm for multiple nodes to agree on a value despite failures or delays
  - Ensures correctness, provides good liveness
- Paxos can be used for lots of things, e.g.:
  - Distributed lock service
  - Choose master or primary in a master/slave or primary/secondary system
  - Choose which operation to perform next
  - Turn a set of unreliable connected machines into a reliable virtual machine (can be used to solve issues in 2PC and implement ACID transactions)
- Today: Chubby, Google's distributed lock service
  - Then we'll look at Bigtable, a distributed storage system at Google that uses Chubby

The Chubby lock service for loosely-coupled distributed systems

Mike Burrows

OSDI 2006

Slide acks to: Shimin Chen

(http://www.cs.cmu.edu/~chensm/Big_Data_reading_group/ slides/shimin-chubby.ppt)

Introduction

- What is Chubby?
  - Lock service in a loosely-coupled distributed system
  - Client interface similar to:
    - Whole-file advisory locks (think .lck files)
    - Notification of various events (e.g., file modifications, think inotify)
  - Primary goals: reliability, availability, easy-to-understand semantics

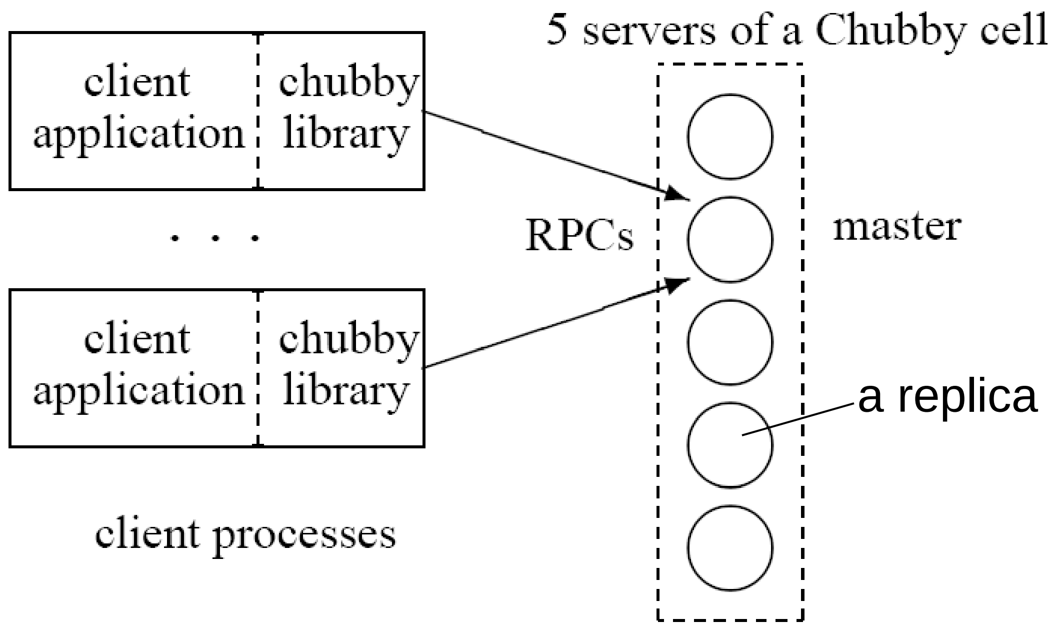
- How is it used?
  - Used in Google: GFS, Bigtable, etc.
  - Purposes: elect masters; store small amount of metadata, such as the root of the distributed data structures
  - Open-source version of Bigtable, Hbase, uses an open-source lock service, called Zookeeper

What Was the Chubby Paper About?

“Building Chubby was an engineering effort … it was not research. We claim no new algorithms or techniques. The purpose of this paper is to describe what we did and why, rather than to advocate it.”

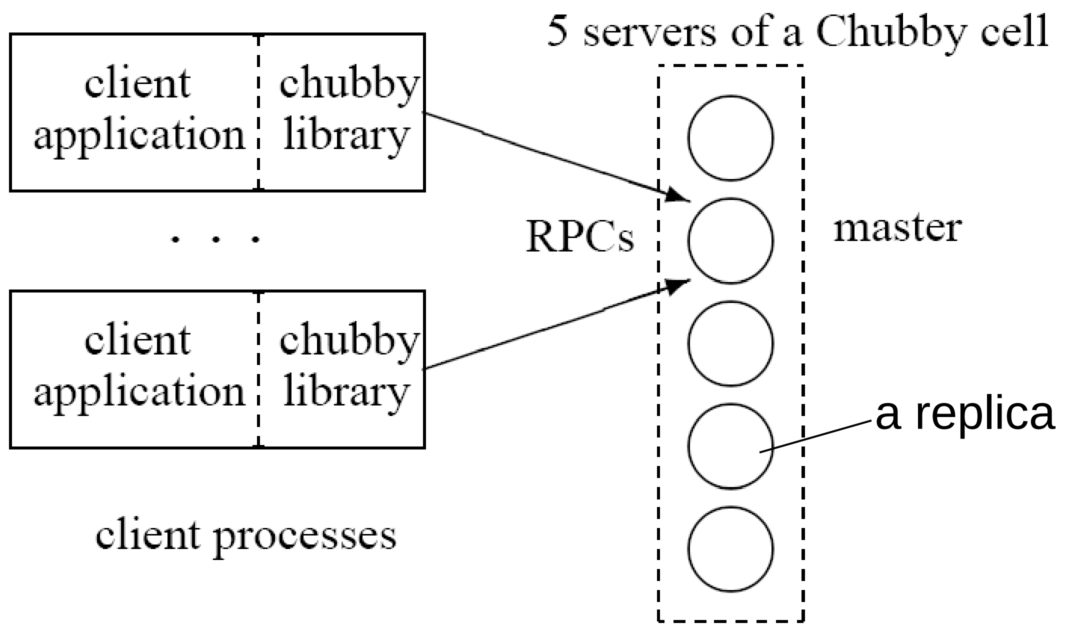
Chubby paper, OSDI 2006

System Architecture



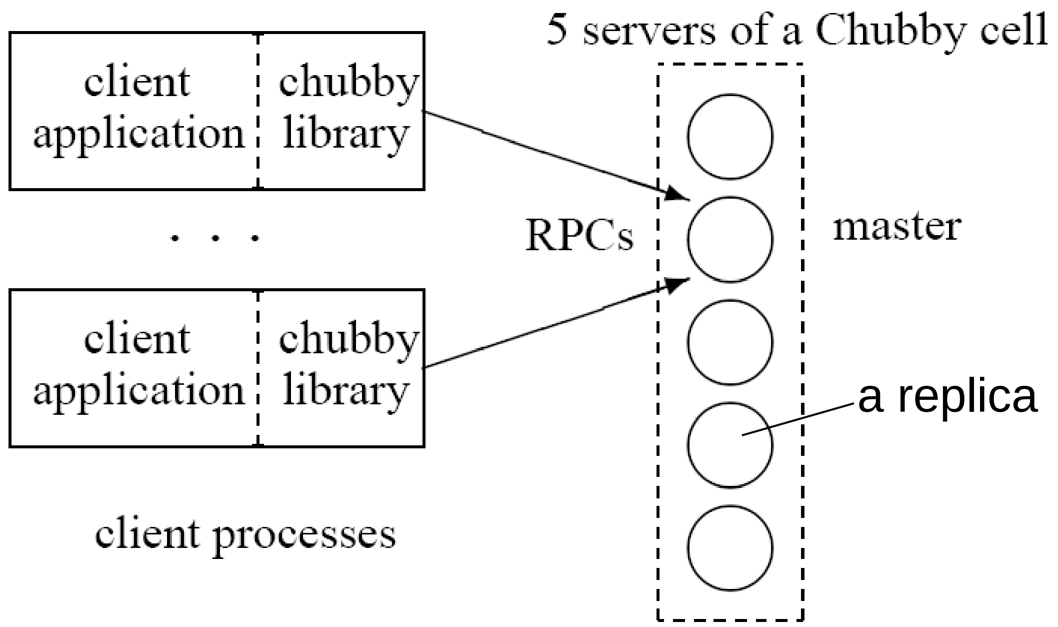
- A chubby cell consists of a small set of servers (replicas)
  - Placed in different racks, so as to minimize chance of correlated failures
- A master is elected from the replicas via Paxos
  - Master lease: several seconds
  - If master fails, a new one will be elected, but only after master leases expire
- Client talks to the master via the chubby library
  - All replicas are listed in DNS; clients discover master by talking to any replica

System Architecture (2)



- Replicas maintain copies of a simple database
- Clients send read/write requests only to the master
- For a write:
  - The master propagates it to replicas via Paxos
  - Replies after the write reaches a majority of replicas
- For a read:
  - The master satisfies the read alone

System Architecture (3)



- If a replica fails and does not recover for a long time (a few hours)
  - A fresh machine is selected to be a new replica, replacing the failed one
  - It updates the DNS
  - Obtains a recent copy of the database
  - The current master polls DNS periodically to discover new replicas
  - Integrating the new replica into the group is another Paxos run

Paxos Use in Master Election

- At any point in time, there must be at most one master
  - No two nodes must think they are masters at same time
- Example:
  - Suppose A is master and it gets disconnected from B
  - B times out trying to talk to A, thinks A is dead, and proposes that it be the master
  - If other nodes agree and A doesn’t hear about the new master, then A will continue to act as master for a while, accepting read requests for what could be stale data, for example
  - How would you solve this?

Paxos Use in Master Election

- Chubby combines Paxos with a lease mechanism
  - When a master dies, a node proposes a master change through Paxos
  - When nodes receive the proposal, they will only accept it if the old master's lease has expired
  - A node becomes the master if a majority of nodes have given it the accept to become the master
  - Once a node becomes a master, it knows that it will remain so for at least the lease period
    - It can extend the lease by getting the accept from a majority of the nodes

Chubby Interface: UNIX File System

- Chubby supports a strict tree of files and directories
  - The way to think about these files is that they are locks with a little bit of contents (e.g., identity and location of a primary)
  - No symbolic links, no hard links
  - /ls/foo/wombat/pouch
    - 1st component (ls): lock service (common to all names)
    - 2nd component (foo): the chubby cell (used in DNS lookup to find the cell master)
    - The rest: name inside the cell
- Support most normal operations
  - Create, delete, open, write, …
- Support reader/writer lock on a node

Chubby Events

- Clients can subscribe to events
  - File contents modified: e.g., if the file contains the location of a service, this event can be used to track the service’s location
  - Master failed over
  - Child node added, removed, modified
  - Handle becomes invalid: probably communication problem
  - Lock acquired (rarely used)
  - Locks are conflicting (rarely used)

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()

Example: Primary Election

```
Open("/ls/foo/OurServicePrimary", "write mode");
if (successful) {
    // primary
    SetContents(primary_identity);
} else {
    // replica
    Open("/ls/foo/OurServicePrimary", "read mode",
            "file-modification event");
    when notified of file modification:
            primary = GetContentsAndStat();
}
```

Five Nodes?! Is that Enough?

- It seems so, but Chubby's users need to be very careful!
- Most typical use: as a name service
  - Bigtable, systems that need a more consistent DNS, …
- Abusive uses:
  - Publish/subscribe system on Chubby!
  - Repeated open/close calls for polling a file
  - Solution for the above: cache file handles
  - General solution: Review clients' code before they can use shared Chubby cells!
- How can Chubby scale??
  - E.g., to support more files

Chubby Summary

- Lock Service
- UNIX-like file system interface
- Reliability and availability

- Chubby uses Paxos for everything
  – Propagate writes to a file
  – Choosing a Master
  – Even for adding new Chubby servers to a Chubby cell

- Paxos transforms a multi-node service into something that looks very much like one fault-tolerant, albeit slower, server! → pretty close to distributed systems' core goal

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

OSDI 2006

Slide acks to: Mohsen Taheriyan

(http://www-scf.usc.edu/~csci572/2011Spring/presentations/Taheriyan.pptx)

Bigtable Description Outline

- Motivation and goals (today)
- Interfaces and semantics (today)
- Architecture and implementation (next time)
- Hbase open-source implementation (next time) – with code snippets ☺!

Bigtable Description Outline

- Motivation and goals (today)
- Interfaces and semantics (today)
- Architecture and implementation (next time)
- Hbase open-source implementation (next time) – with code snippets ☺!

Motivation

- Lots of (semi-)structured data at Google
  - Web data:
    - Contents, crawl metadata, links, anchors, pagerank, …
  - Per-user data:
    - User preference settings, recent queries/search results, …
  - Map data:
    - Physical entities (shops, restaurants, etc.), roads, satellite image data, user annotations, …

- Scale is huge
  - Billions of Web pages, many versions/page (~20K/version)
  - Hundreds of millions of users, thousands of q/sec
  - 100TB+ of satellite image data
  - (Above numbers are as of 2006-7!)

Goals

- Want asynchronous processes to continuously update different pieces of data
  - Want access to most current data at any time
- Need to support:
  - Very high read/write rates (millions of ops per second)
  - Efficient retrieval of small subsets of the data
  - Efficient scans over entire or subsets of the data
- Often want to examine data changes over time
  - E.g. Contents of a web page over multiple crawls

Why Not Commercial DB?

- Scale is too large for most commercial databases
- Cost of licensing per machine would be very high
  - Building internally means system can be applied across many projects for low incremental cost
- Low-level storage optimizations help performance
  - Much harder when running on top of a database layer

"Also fun and challenging to build large-scale systems" ☺

Bigtable

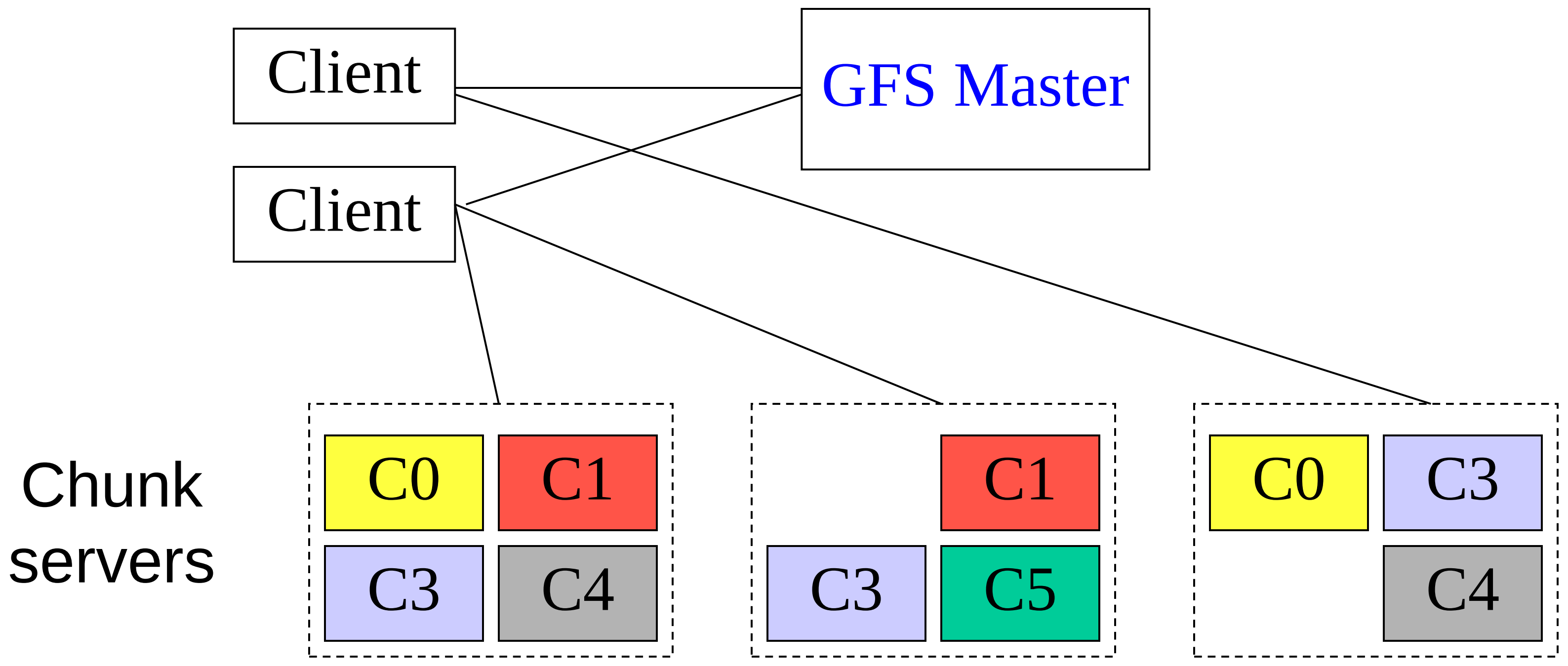
- A distributed storage system for (semi-)structured data
- Scalable
  - Thousands of servers
  - Terabytes of in-memory data
  - Petabyte of disk-based data
  - Millions of reads/writes per second, efficient scans
- Self-managing
  - Servers can be added/removed dynamically
  - Servers adjust to load imbalance
- Extremely popular at Google (as of 2008)
  - Web indexing, personalized search, Google Earth, Google Analytics, Google Finance, …

Background

- Building blocks
  - Google File System (GFS): Raw storage
  - Scheduler: Schedules jobs onto machines
  - Chubby: Lock service

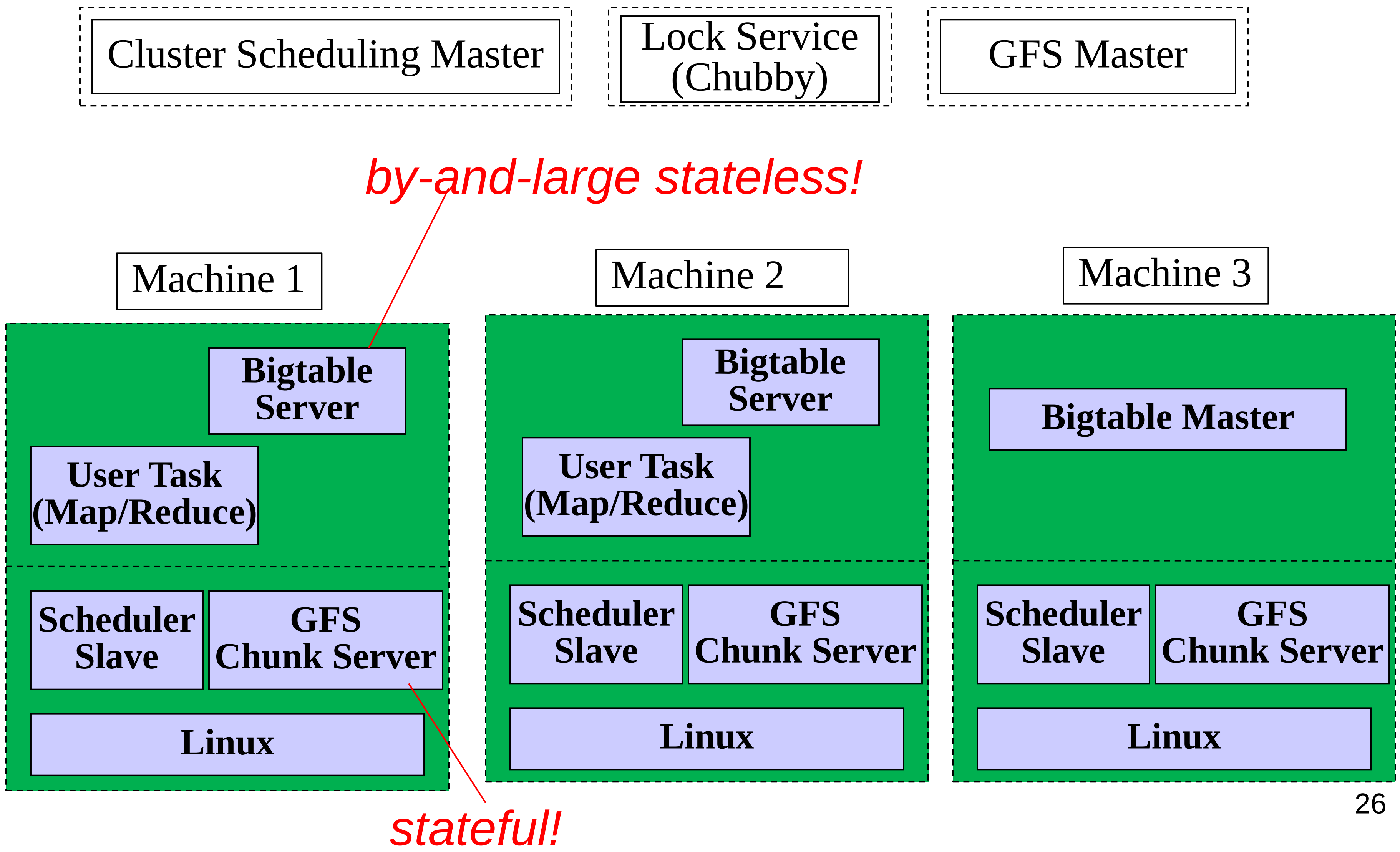
- BigTable uses of building blocks
  - GFS: stores all persistent state
  - Scheduler: schedules jobs involved in BigTable serving
  - Chubby: master election, location bootstrapping

GFS (Reminder)



- Master manages metadata
- Data transfers happen directly between clients/chunkservers
- Files broken into chunks (typically 64 MB)
- Chunks replicated across three machines for reliability

Typical Cluster
(We'll Return to This)



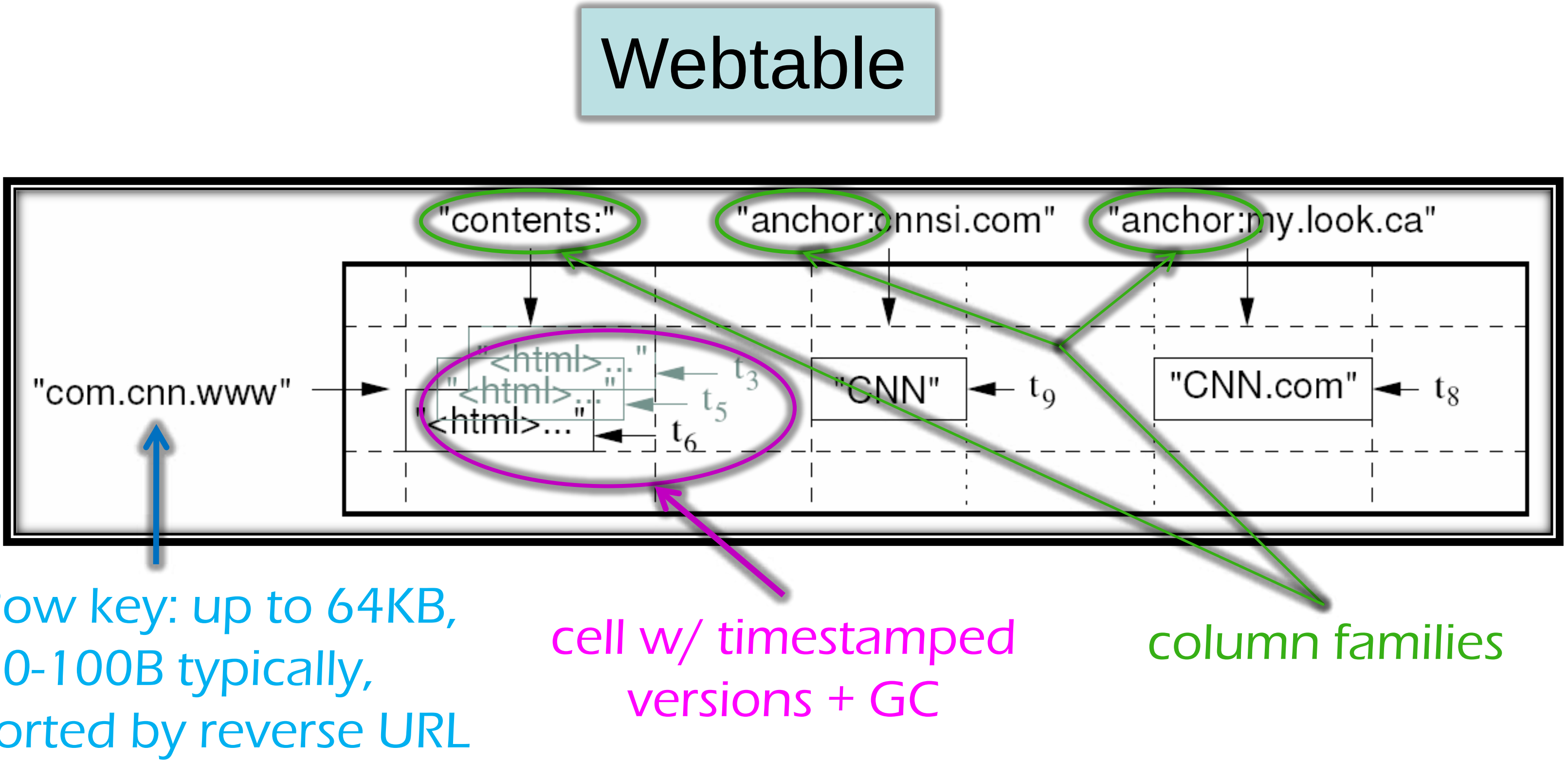
Bigtable Description Outline

- Motivation and goals (today)
- Interfaces and semantics (today)
- Architecture and implementation (next time)
- Hbase open-source implementation (next time) – with code snippets ☺!

Basic Data Model

- "A BigTable is a sparse, distributed, persistent multi-dimensional sorted map"

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

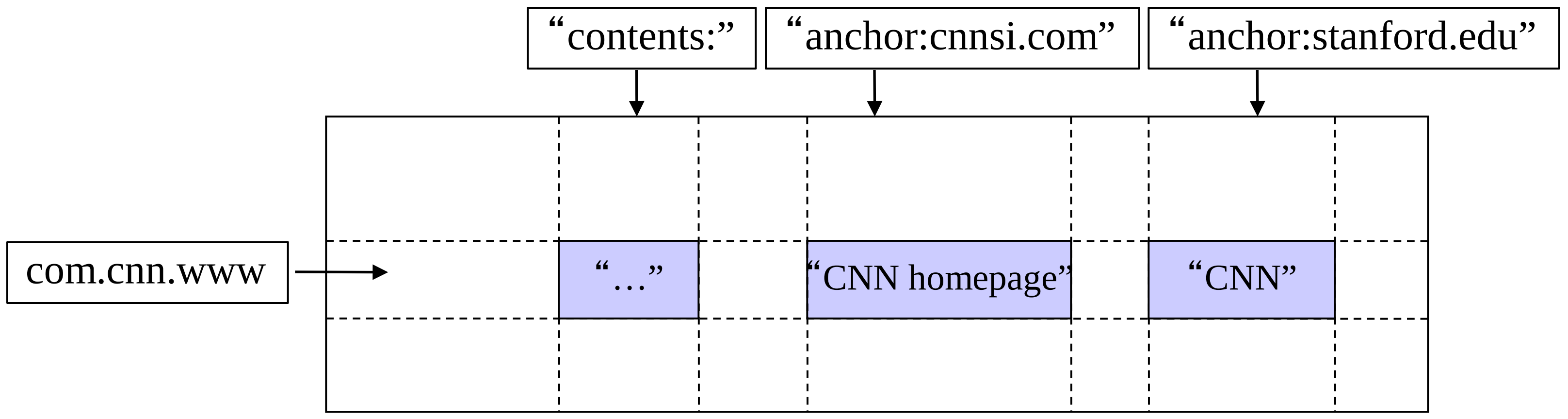


Rows

- Row name/key is an arbitrary string
- Rows are ordered lexicographically
  - Rows close together lexicographically usually on one or a small number of machines
  - Example: *com.cnn.www* vs. *www.cnn.com* – which one provides more locality for a site: query?
- Access to data in a row is atomic
  - Data row is the only unit of atomicity in Bigtable
- Does not support relational model
  - No table wide integrity constants
  - No multi-row transactions

Columns

- Columns have two-level name structure:
  family:optional_qualifier
- Column family
  - Unit of access control
  - Has associated type information
- Qualifier gives unbounded columns in each row
  - Provides additional levels of indexing, if desired



Timestamps

- Used to store different versions of data in a cell
  - 64-bits integers
  - 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 in timestamp range (or all values)”*
- Column families can be marked w/ attributes:
  - *“Only retain most recent K values in a cell”*
  - *“Keep values until they are older than K seconds”*
- Example uses:
  - Keep versions of the Web
  - We’ve recently used timestamps to synchronize replicas in a project

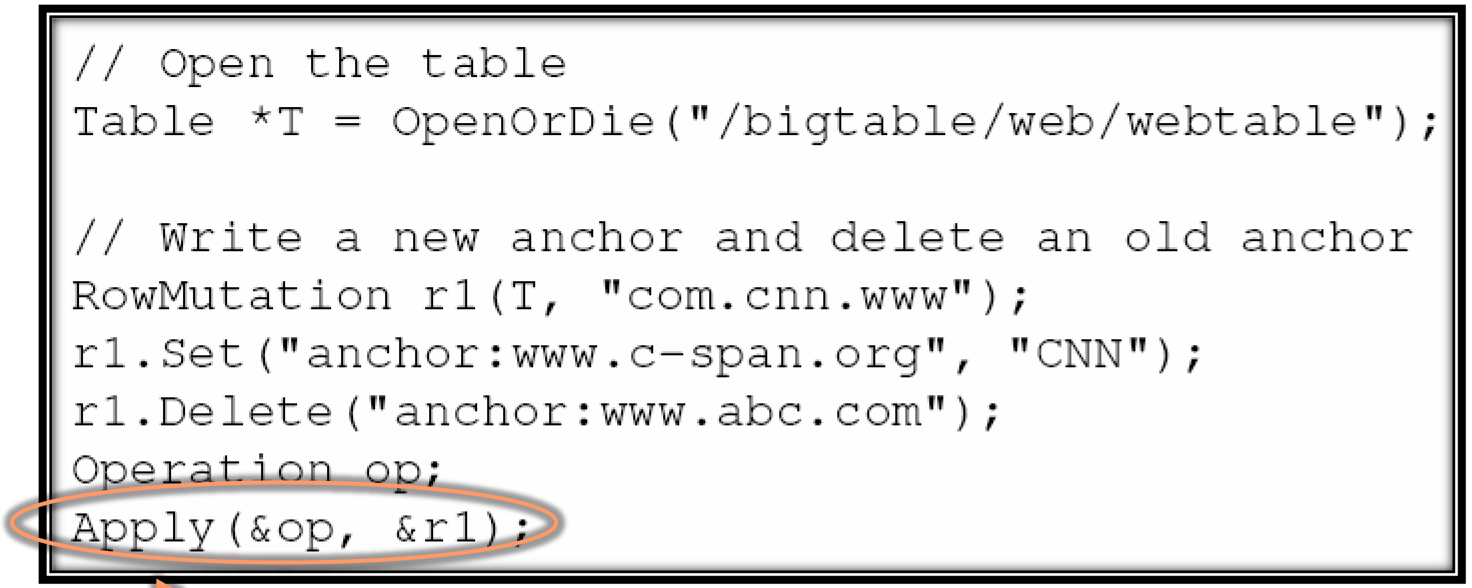
API

- Metadata operations
  - Create/delete tables, column families, change metadata

- Writes (atomic)
  - Set(): write cells in a row
  - DeleteCells(): delete cells in a row
  - DeleteRow(): delete all cells in a row

- Reads
  - 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 column families, or specific columns

API Examples: Write/Modify



```
// 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);
```

atomic row modification

No support for (RDBMS-style) multi-row transactions

Example Exercise: Define Bigtable Schema for (Simplified) Twitter

Bigtable Description Outline

- Motivation and goals (today)
- Interfaces and semantics (today)
- Architecture and implementation (next time)
- Hbase open-source implementation (next time) – with code snippets ☺!

Next Time: Continue Bigtable

- Motivation and goals (today)
- Interfaces and semantics (today)
- Architecture and implementation (next time)
- Hbase open-source implementation (next time)
  - We'll look at code snippets! ☺