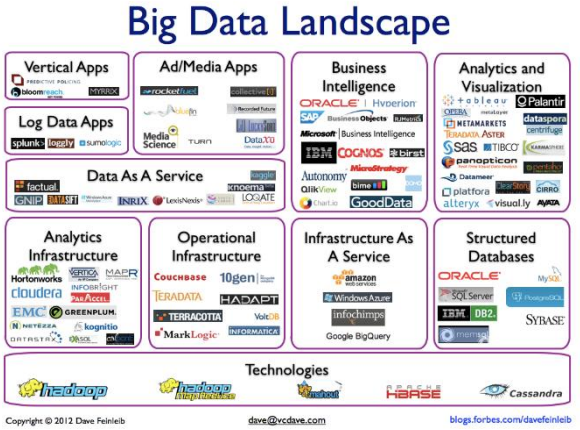


GFS, Mapreduce and Bigtable

Seminar on big data management

Lecturer: Jiaheng Lu

Spring 2016

Google big data techniques

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

- Google File System
- MapReduce model
- Bigtable data storage platform

The Google File System

The Google File System (GFS)

- A scalable distributed file system for large distributed data intensive applications
- Multiple GFS clusters are currently deployed.
- The largest ones (in 2003) have:
  - 1000+ storage nodes
  - 300+ TeraBytes of disk storage
  - heavily accessed by hundreds of clients on distinct machines

Introduction

- Shares many same goals as previous distributed file systems

  performance, scalability, reliability, etc

  GFS design has been driven by four key observation of Google application workloads and technological environment

Intro: Observations 1

- **1. Component failures are the norm**

  constant monitoring, error detection, fault tolerance and automatic recovery are integral to the system

- **2. Huge files (by traditional standards)**

  Multi GB files are common

  I/O operations and blocks sizes must be revisited

Intro: Observations 2

- **3. Most files are mutated by appending new data**

  This is the focus of performance optimization and atomicity guarantees

- **4. Co-designing the applications and APIs benefits overall system by increasing flexibility**

The Design

- *Cluster consists of a single* master *and multiple* chunkservers *and is accessed by multiple* clients

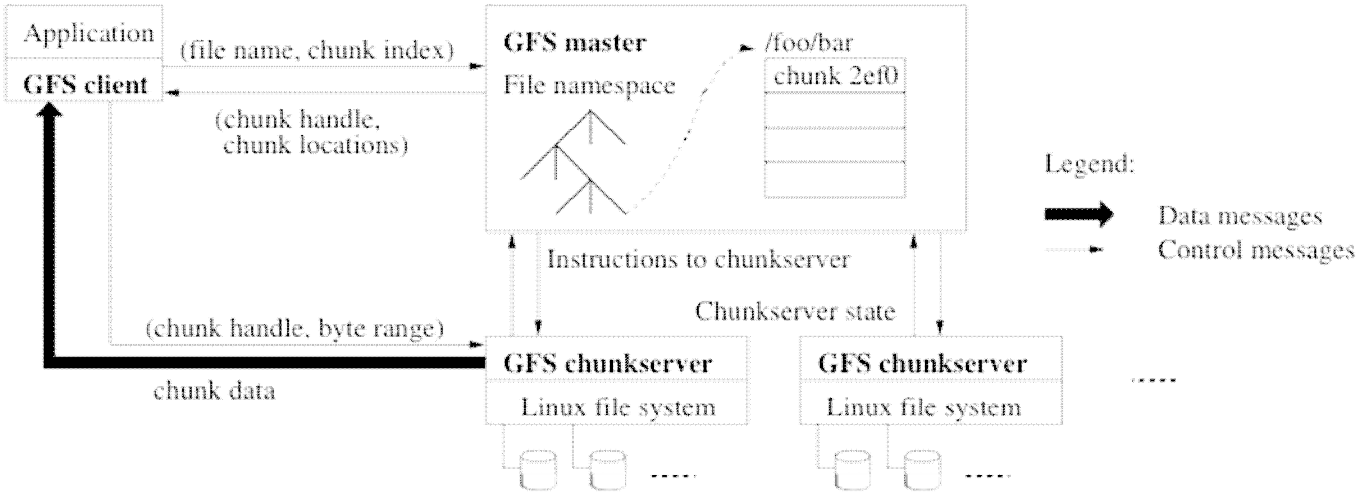
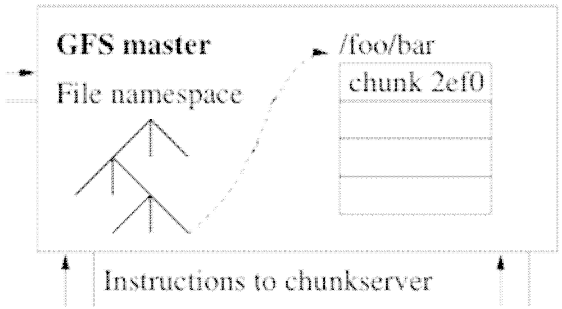


Figure 1: GFS Architecture

The Master

- *Maintains all file system metadata.*
  - *names space, access control info, file to chunk mappings, chunk (including replicas) location, etc.*
- *Periodically communicates with chunkservers in* HeartBeat *messages to give instructions and check state*



The Master

- *Helps make sophisticated chunk placement and replication decision, using global knowledge*
- *For reading and writing, client contacts Master to get chunk locations, then deals directly with chunkservers*

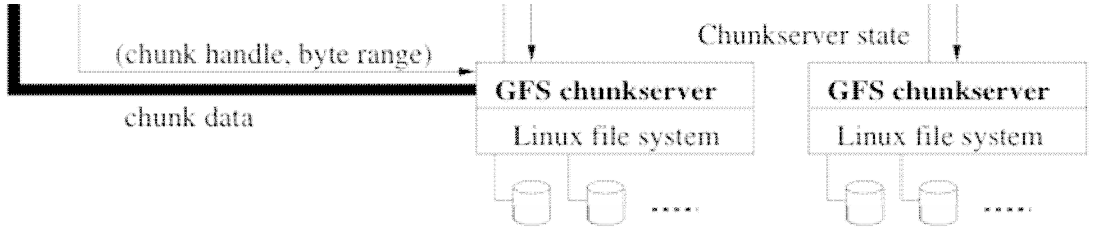
  Master is not a bottleneck for reads/writes

Chunkservers

- *Files are broken into* chunks*. Each chunk has a immutable globally unique 64-bit* chunk-handle.

  handle is assigned by the master at chunk creation

- ***Chunk size is 64 MB***
- *Each chunk is replicated on 3 (default) servers*



Clients

- *Linked to apps using the file system API.*
- *Communicates with master and chunkservers for reading and writing*
  - *Master interactions only for metadata*
  - *Chunkserver interactions for data*
- *Only caches metadata information*
  - *Data is too large to cache.*

Chunk Locations

- **Master does not keep a persistent record of locations of chunks and replicas.**
- **Polls** chunkservers at startup, and when new chunkservers join/leave for this.
- Stays up to date by controlling placement of new chunks and through ***HeartBeat*** messages (when monitoring chunkservers)

Introduction to MapReduce

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

MapReduce: Insight

- A Toy Problem:
- • We have 10 billion documents
- • Average documents size is 20KB

- • 10 Billion docs == 200 TB
- • We want build a language model of the Web:
- – Basically count how many times each word occur

MapReduce: Insight

- for each document d
- { for each word w in d {word_count[w]++; } }

- Approximately 1 month.

- Assumptions:
- 1. All disk reads are sequential
- 2. Dictionary fits into the memory

MapReduce Programming Model

- Inspired from map and reduce operations commonly used in functional programming languages like Lisp.

- Users implement interface of two primary methods:
  - 1. Map: (key1, val1) → (key2, val2)
  - 2. Reduce: (key2, [val2]) → [val3]

Map operation

- Map, a pure function, written by the user, takes an input key/value pair and produces a set of intermediate key/value pairs.
  - e.g. (doc—id, doc-content)

- Draw an analogy to SQL, map can be visualized as *group-by* clause of an aggregate query.

Reduce operation

- On completion of map phase, all the intermediate values for a given output key are combined together into a list and given to a reducer.

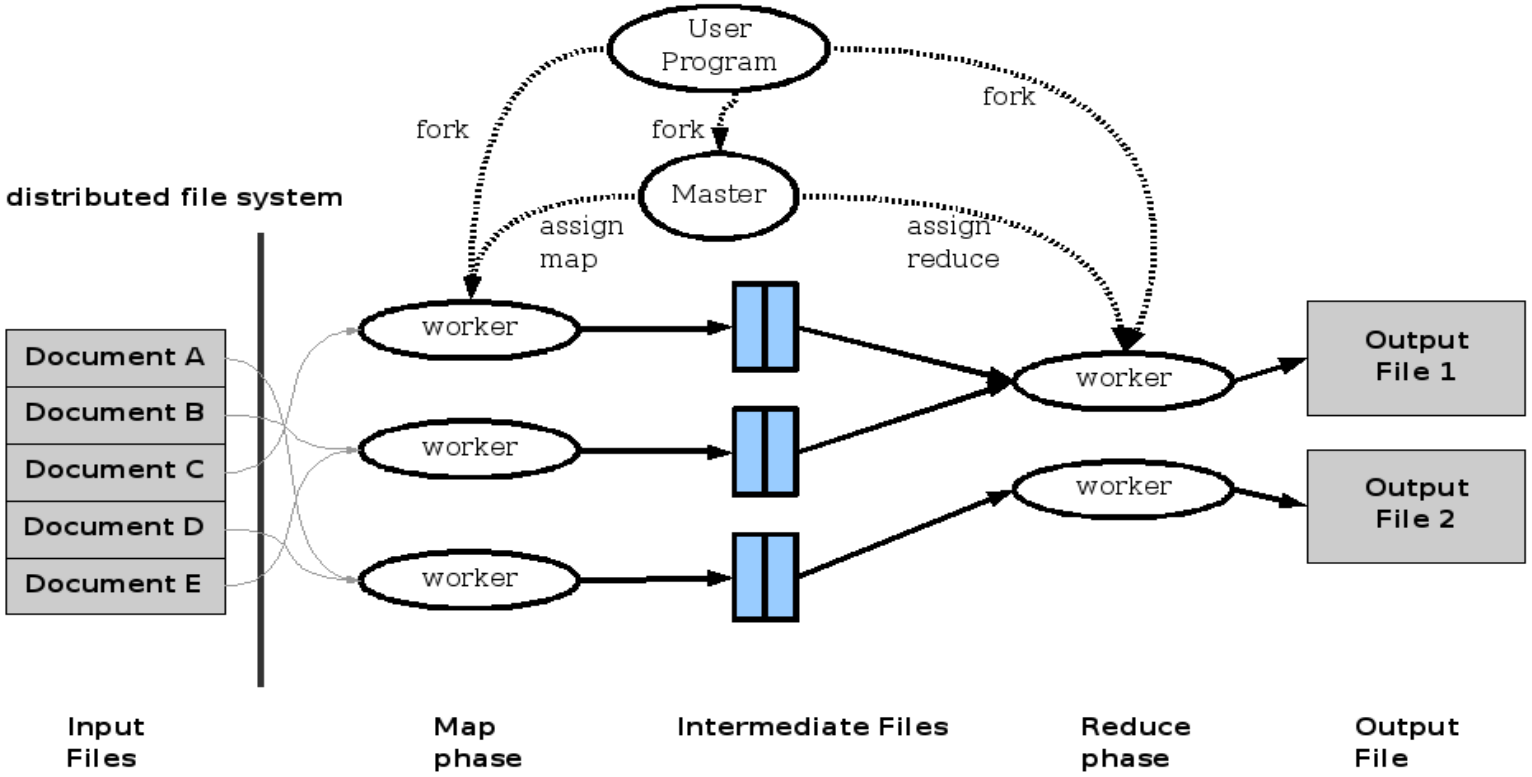
- Can be visualized as *aggregate* function (e.g., average) that is computed over all the rows with the same group-by attribute.

Pseudo-code

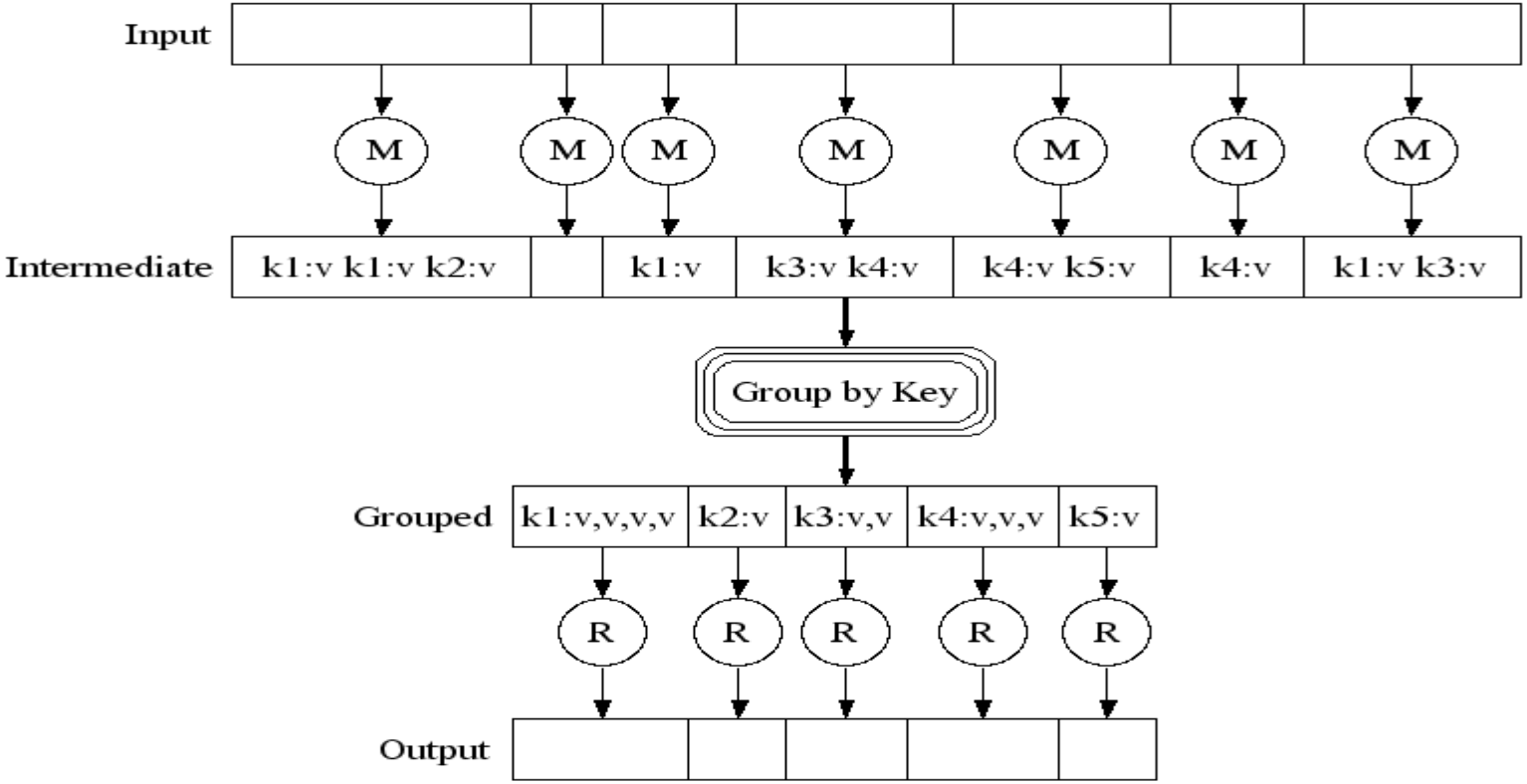
```
map(String input_key, String input_value):
// input_key: document name
// input_value: document contents
    for each word w in input_value:
      EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
// output_key: a word
// output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
      result += ParseInt(v);
    Emit(AsString(result));
```

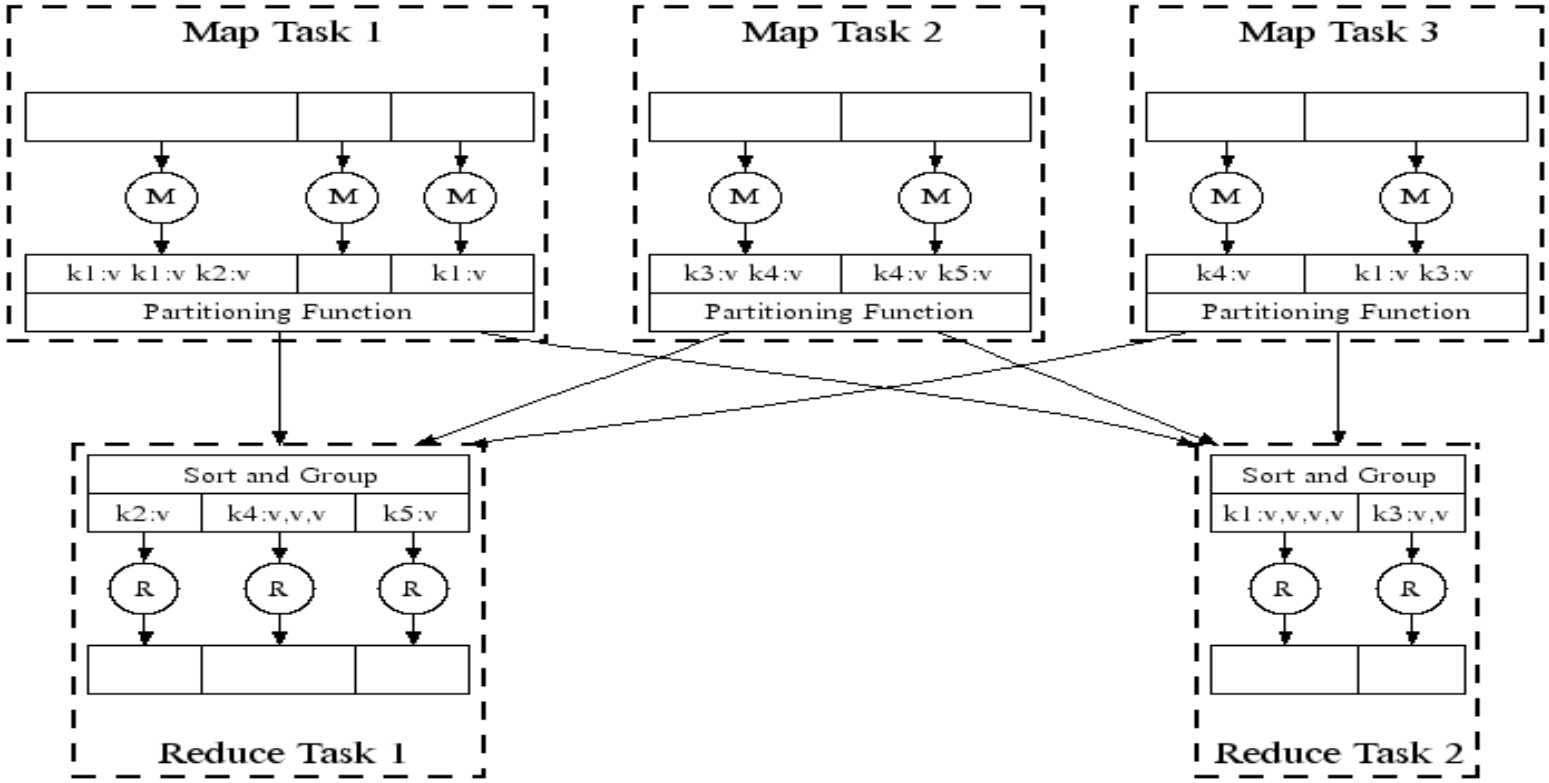
MapReduce: Execution overview



MapReduce: Example



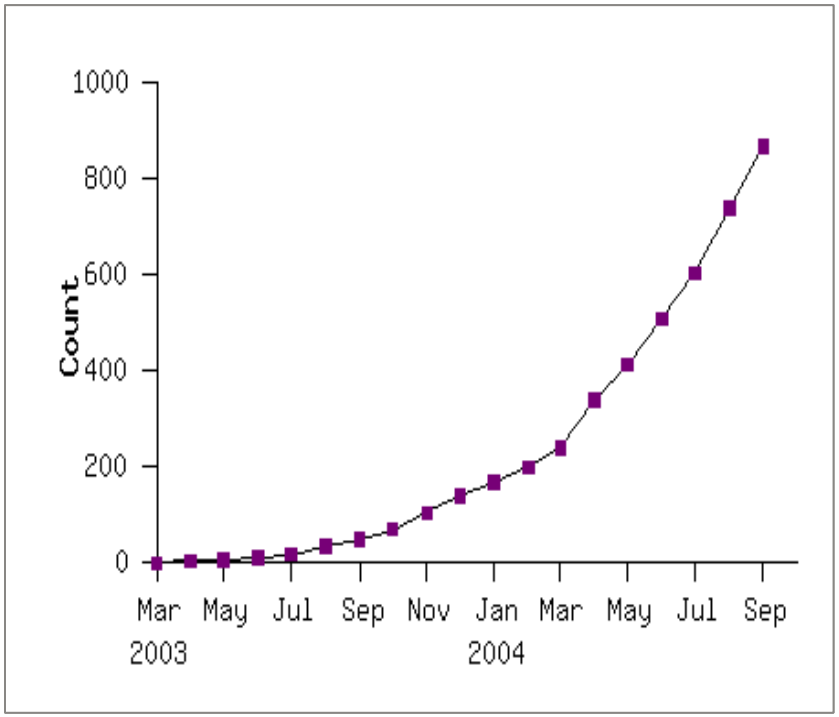
MapReduce in Parallel: Example



MapReduce: Some More Apps

- Distributed Grep.
- Count of URL Access Frequency.
- Clustering (K-means)
- Graph Algorithms.
- Indexing Systems

MapReduce Programs In Google Source Tree



- Google File System
- MapReduce model
- Bigtable data storage platform

BigTable: A Distributed Storage System for Structured Data

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Introduction

- BigTable is a distributed storage system for managing structured data.
- Designed to scale to a very large size
  - Petabytes of data across thousands of servers
- Used for many Google projects
  - Web indexing, Personalized Search, Google Earth, Google Analytics, Google Finance, …
- Flexible, high-performance solution for all of Google's products

Motivation

- Lots of (semi-)structured data at Google
  - URLs:
    - Contents, crawl metadata, links, anchors, pagerank, …
  - Per-user data:
    - User preference settings, recent queries/search results, …
  - Geographic locations:
    - Physical entities (shops, restaurants, etc.), roads, satellite image data, user annotations, …
- Scale is large
  - Billions of URLs, many versions/page (~20K/version)

Why not just use commercial DB?

- Scale is too large for most commercial databases
- Even if it weren’t, cost would be very high
  - Building internally means system can be applied across many projects for low incremental cost

Goals

- Want asynchronous processes to be continuously updating 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 scans over all or interesting subsets of data
  - Efficient joins of large one-to-one and one-to-many datasets
- Often want to examine data changes over time
  - E.g. Contents of a web page over multiple crawls

BigTable

- Distributed multi-level map
- Fault-tolerant, persistent
- 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

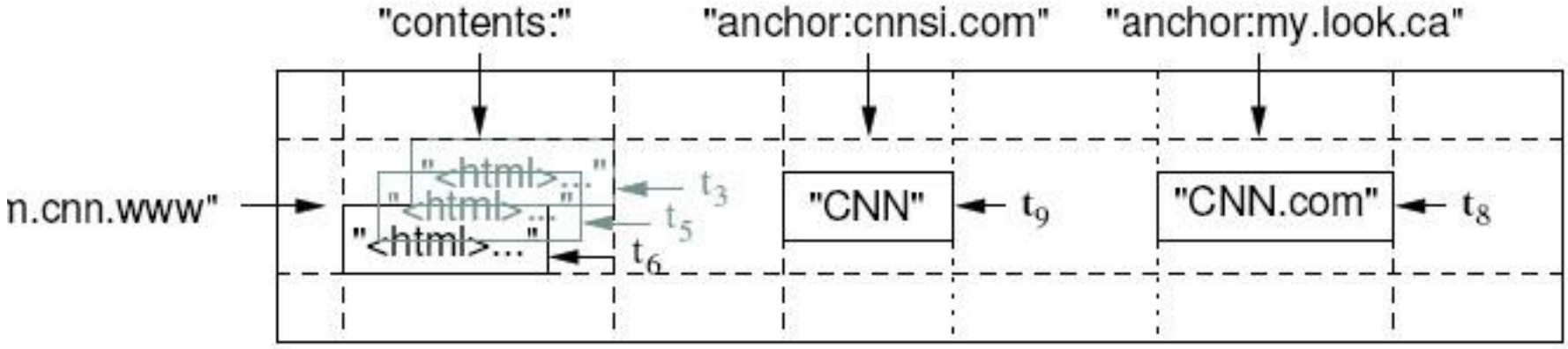
Building Blocks

- Building blocks:
  - Google File System (GFS): Raw storage
  - Scheduler: schedules jobs onto machines
  - Lock service: distributed lock manager
  - MapReduce: simplified large-scale data processing

Basic Data Model

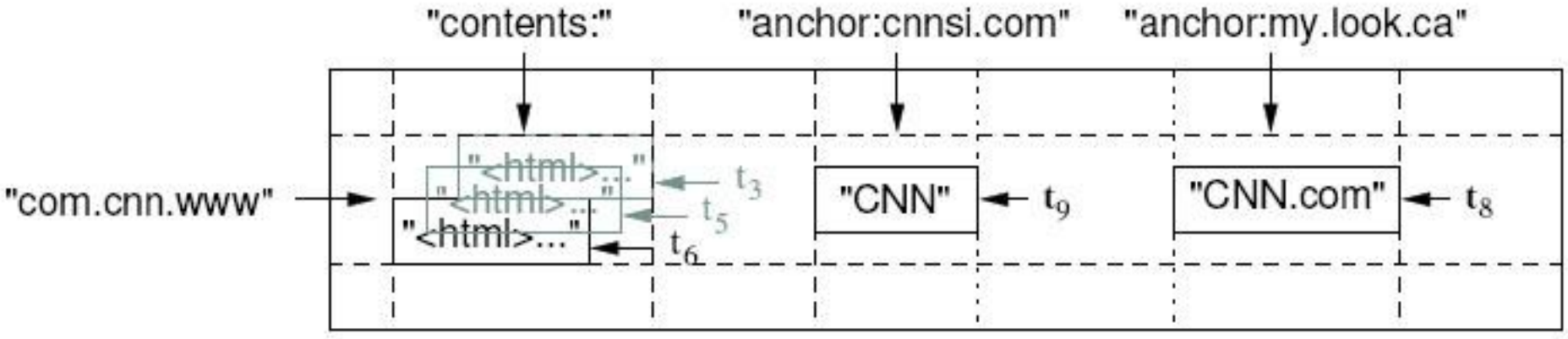
- A BigTable is a sparse, distributed persistent multi-dimensional sorted map

(row, column, timestamp) -> cell contents



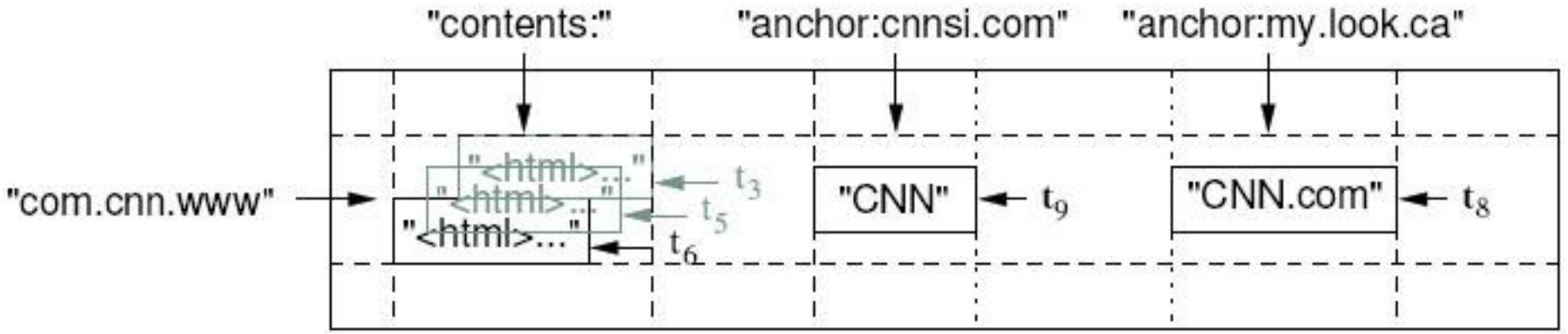
- Good match for most Google applications

WebTable Example



- 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.

Rows



- Name is an arbitrary string
  - Access to data in a row is atomic
  - Row creation is implicit upon storing data
- Rows ordered lexicographically
  - Rows close together lexicographically usually on one or a small number of machines

Rows (cont.)

Reads of short row ranges are efficient and typically require communication with a small number of machines.

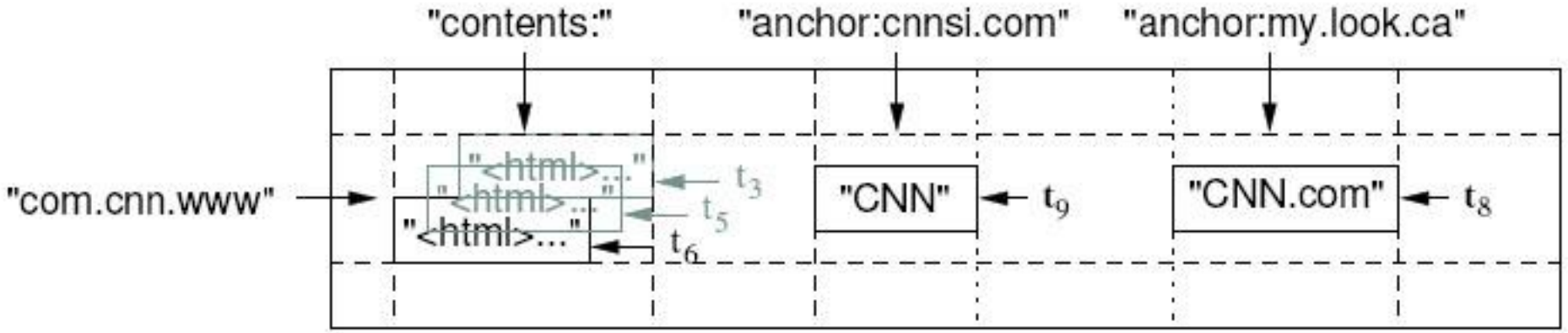
- Can exploit this property by selecting row keys so they get good locality for data access.
- Example:

```
math.gatech.edu, math.uga.edu, phys.gatech.edu,
phys.uga.edu
```

vs

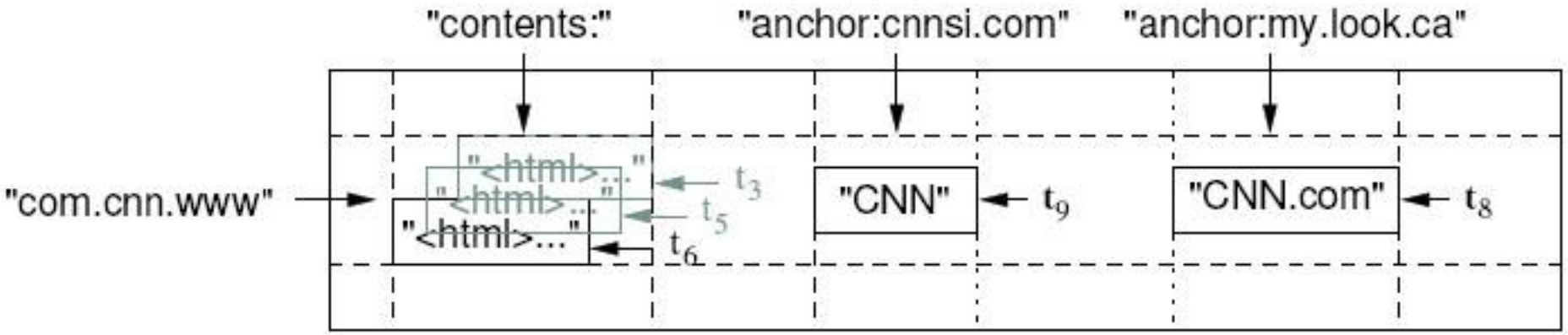
```
edu.gatech.math, edu.gatech.phys, edu.uga.math,
edu.uga.phys
```

Columns



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

Timestamps



- Used to 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 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"*

Implementation – Three Major Components

- Library linked into every client
- One master server
  - Responsible for:
    - Assigning tablets to tablet servers
    - Detecting addition and expiration of tablet servers
    - Balancing tablet-server load
    - Garbage collection
- Many tablet servers
  - Tablet servers handle read and write requests to its table
  - Splits tablets that have grown too large

Tablets

- Large tables broken into tablets at row boundaries
  - Tablet holds contiguous range of rows
    - Clients can often choose row keys to achieve locality
  - Aim for ~100MB to 200MB of data per tablet
- Serving machine responsible for ~100 tablets
  - Fast recovery:
    - 100 machines each pick up 1 tablet for failed machine
  - Fine-grained load balancing:
    - Migrate tablets away from overloaded machine
    - Master makes load-balancing decisions

Tablet Assignment

- Each tablet is 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. Also keeps track of unassigned tablets.
- When a tablet is unassigned, master assigns the tablet to an tablet server with sufficient room.

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

Summary

- GFS is developed by Google for big data challenge
- Hadoop is an open-source software of GFS
- Mapreduce is a distributed programming framework
- Bigtable is developed to store large-scale Web data