Large-Scale Data and Computation

Yifu Huang

School of Computer Science, Fudan University
huangyifu@fudan.edu.cn

COMP620003 Advanced Computer Networks Report, 2013

Outline

Motivation

- Why we choose these papers?
  - Big data
  - Google
- Why we need GFS, MapReduce, Bigtable?
  - A scalable distributed file system for large distributed data-intensive applications
  - A programming model for processing and generating large datasets that is amenable to a broad variety of real-world tasks
  - A distributed storage system for managing structured data that is designed to scale to a very large size
- What can we learn from these papers?
  - The design and implementation ideas behind GFS, MapReduce, Bigtable
  - Get ready to enjoy open source equivalents HDFS, YARN, HBase

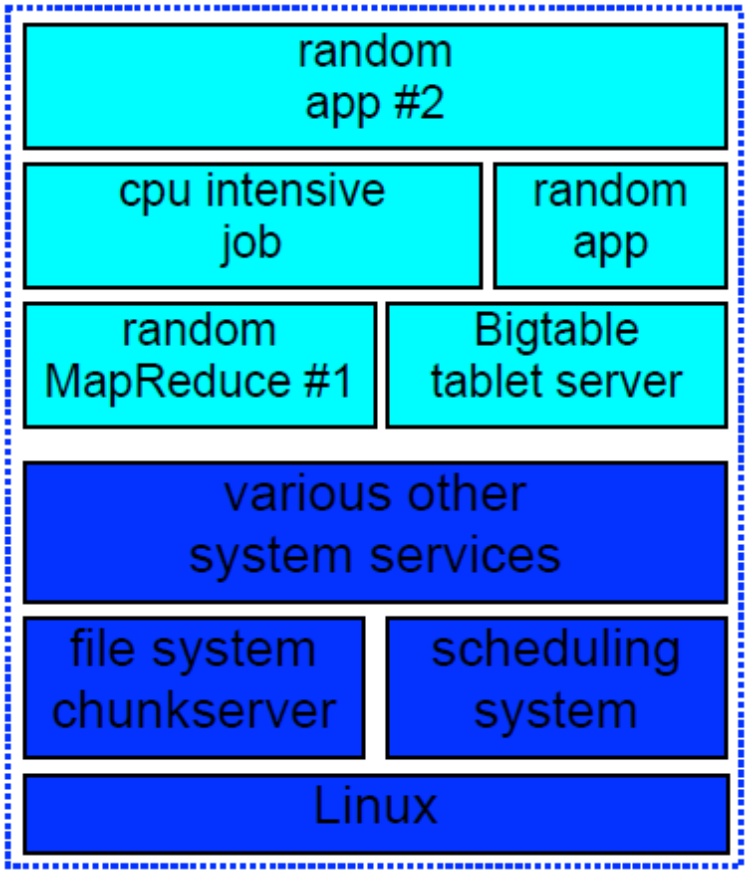
Motivation

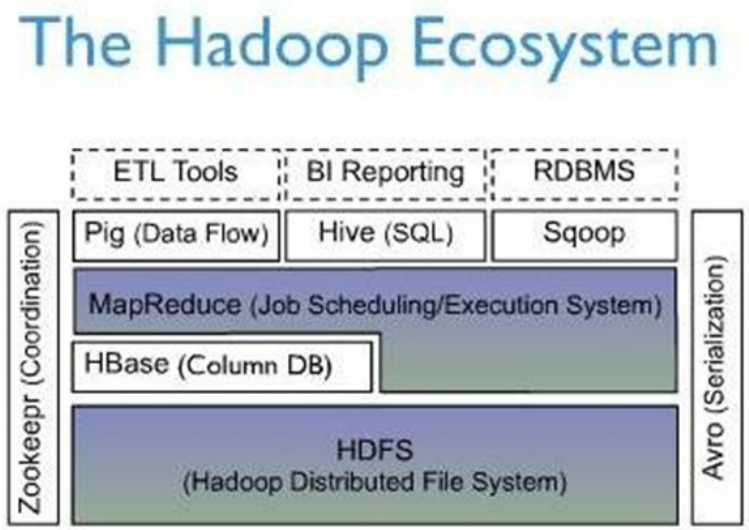
- Why we choose these papers?
  - Big data
  - Google
- Why we need GFS, MapReduce, Bigtable?
  - A scalable distributed file system for large distributed data-intensive applications
  - A programming model for processing and generating large datasets that is amenable to a broad variety of real-world tasks
  - A distributed storage system for managing structured data that is designed to scale to a very large size
- What can we learn from these papers?
  - The design and implementation ideas behind GFS, MapReduce, Bigtable
  - Get ready to enjoy open source equivalents HDFS, YARN, HBase

Motivation

- Why we choose these papers?
  - Big data
  - Google
- Why we need GFS, MapReduce, Bigtable?
  - A scalable distributed file system for large distributed data-intensive applications
  - A programming model for processing and generating large datasets that is amenable to a broad variety of real-world tasks
  - A distributed storage system for managing structured data that is designed to scale to a very large size
- What can we learn from these papers?
  - The design and implementation ideas behind GFS, MapReduce, Bigtable
  - Get ready to enjoy open source equivalents HDFS, YARN, HBase

Ecosystem


random
app #2
cpu intensive
job
random
app
random
MapReduce #1
Bigtable
tablet server
various other
system services
file system
chunkserver
scheduling
system
Linux


The Hadoop Ecosystem
ETL Tools
BI Reporting
RDBMS
Pig (Data Flow)
Hive (SQL)
Sqoop
MapReduce (Job Scheduling/Execution System)
HBase (Column DB)
HDFS
(Hadoop Distributed File System)
Zookeepr (Coordination)
Avro (Serialization)

The Google File System

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google, Inc.

Design Overview

- The system is built from many inexpensive commodity components that often fail
- The system stores a modest number of large files
- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads
- The workloads also have many large, sequential writes that append data to files
- The system must efficiently implement well-defined semantics or multiple clients that concurrently append to the same file
- High sustained bandwidth is more important than low latency

Design Overview

- The system is built from many inexpensive commodity components that often fail
- The system stores a modest number of large files
- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads
- The workloads also have many large, sequential writes that append data to files
- The system must efficiently implement well-defined semantics or multiple clients that concurrently append to the same file
- High sustained bandwidth is more important than low latency

Design Overview

- The system is built from many inexpensive commodity components that often fail
- The system stores a modest number of large files
- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads
- The workloads also have many large, sequential writes that append data to files
- The system must efficiently implement well-defined semantics or multiple clients that concurrently append to the same file
- High sustained bandwidth is more important than low latency

Design Overview

- The system is built from many inexpensive commodity components that often fail
- The system stores a modest number of large files
- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads
- The workloads also have many large, sequential writes that append data to files
- The system must efficiently implement well-defined semantics or multiple clients that concurrently append to the same file
- High sustained bandwidth is more important than low latency

Design Overview

- The system is built from many inexpensive commodity components that often fail
- The system stores a modest number of large files
- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads
- The workloads also have many large, sequential writes that append data to files
- The system must efficiently implement well-defined semantics or multiple clients that concurrently append to the same file
- High sustained bandwidth is more important than low latency

Design Overview

- The system is built from many inexpensive commodity components that often fail
- The system stores a modest number of large files
- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads
- The workloads also have many large, sequential writes that append data to files
- The system must efficiently implement well-defined semantics or multiple clients that concurrently append to the same file
- High sustained bandwidth is more important than low latency

Design Overview (cont.)

- Architecture

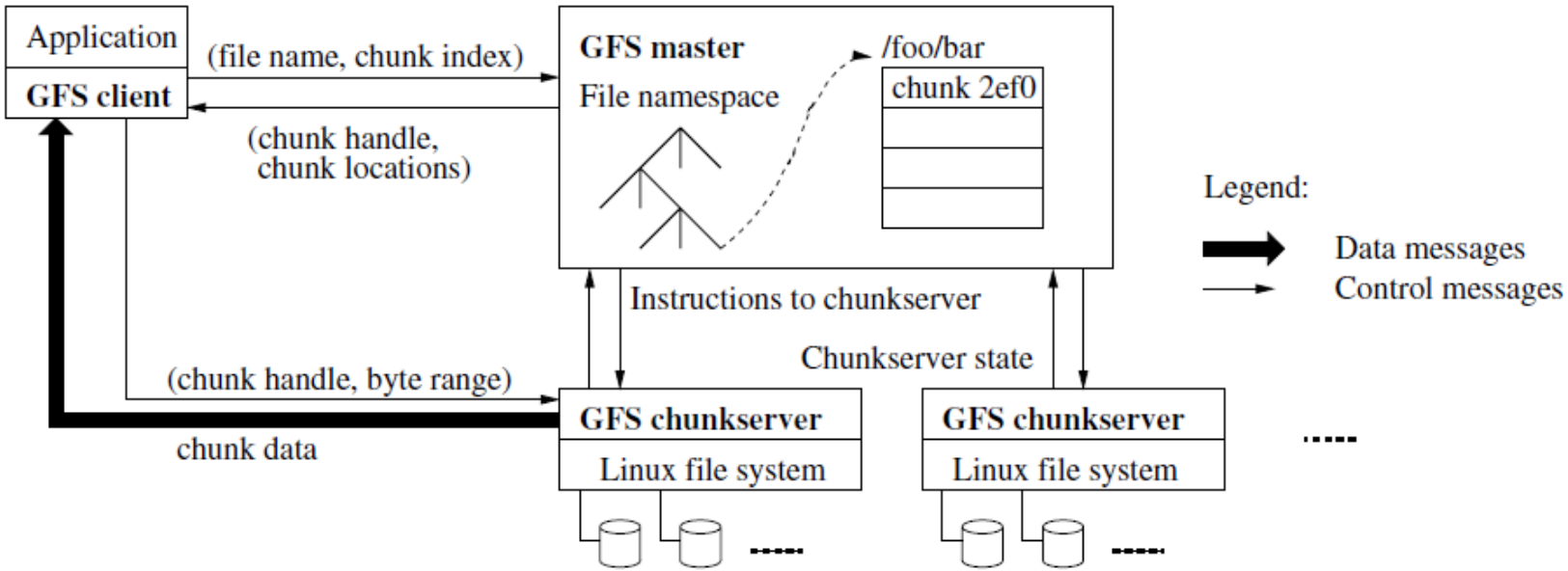


Figure 1: GFS Architecture

Design Overview (cont.)

- Chunk Size: 64 MB
- Metadata
  - The file and chunk namespaces
  - The mapping from files to chunks
  - The locations of each chunk's replicas
- Chunk Locations
- Operation Log
- Consistency Model
  - Guarantees by GFS
  - Implications for Applications

Design Overview (cont.)

- Chunk Size: 64 MB
- Metadata
  - The file and chunk namespaces
  - The mapping from files to chunks
  - The locations of each chunk's replicas
- Chunk Locations
- Operation Log
- Consistency Model
  - Guarantees by GFS
  - Implications for Applications

Design Overview (cont.)

- Chunk Size: 64 MB
- Metadata
  - The file and chunk namespaces
  - The mapping from files to chunks
  - The locations of each chunk's replicas
- Chunk Locations
- Operation Log
- Consistency Model
  - Guarantees by GFS
  - Implications for Applications

Design Overview (cont.)

- Chunk Size: 64 MB
- Metadata
  - The file and chunk namespaces
  - The mapping from files to chunks
  - The locations of each chunk's replicas
- Chunk Locations
- Operation Log
- Consistency Model
  - Guarantees by GFS
  - Implications for Applications

Design Overview (cont.)

- Chunk Size: 64 MB
- Metadata
  - The file and chunk namespaces
  - The mapping from files to chunks
  - The locations of each chunk's replicas
- Chunk Locations
- Operation Log
- Consistency Model
  - Guarantees by GFS
  - Implications for Applications

System Interactions

- Leases and Mutation Order

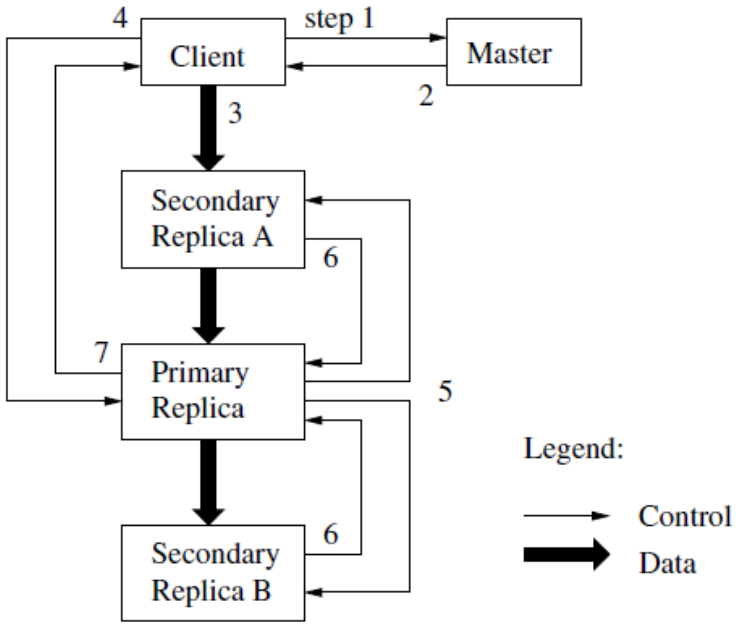


Figure 2: Write Control and Data Flow

System Interactions (cont.)

- Data Flow
  - We decouple the flow of data from the flow of control to use the network efficiently
  - Our goals are to fully utilize each machine's network bandwidth, avoid network bottlenecks and high-latency links, and minimize the latency to push through all the data
- Atomic Record Appends
  - GFS provides an atomic append operation called record append
- Snapshot
  - The snapshot operation makes a copy of a file or a directory tree almost instantaneously, while minimizing any interruptions of ongoing mutations

System Interactions (cont.)

- Data Flow
  - We decouple the flow of data from the flow of control to use the network efficiently
  - Our goals are to fully utilize each machine's network bandwidth, avoid network bottlenecks and high-latency links, and minimize the latency to push through all the data
- Atomic Record Appends
  - GFS provides an atomic append operation called record append
- Snapshot
  - The snapshot operation makes a copy of a file or a directory tree almost instantaneously, while minimizing any interruptions of ongoing mutations

System Interactions (cont.)

- Data Flow
  - We decouple the flow of data from the flow of control to use the network efficiently
  - Our goals are to fully utilize each machine's network bandwidth, avoid network bottlenecks and high-latency links, and minimize the latency to push through all the data
- Atomic Record Appends
  - GFS provides an atomic append operation called record append
- Snapshot
  - The snapshot operation makes a copy of a file or a directory tree almost instantaneously, while minimizing any interruptions of ongoing mutations

Fault Tolerance And Diagnosis

- High Availability
  - Fast Recovery
  - Chunk Replication
  - Master Replication
- Data Integrity
- Diagnostic Tools

Fault Tolerance And Diagnosis

- High Availability
  - Fast Recovery
  - Chunk Replication
  - Master Replication
- Data Integrity
- Diagnostic Tools

Fault Tolerance And Diagnosis

- High Availability
  - Fast Recovery
  - Chunk Replication
  - Master Replication
- Data Integrity
- Diagnostic Tools

MapReduce: Simplied Data Processing on Large Clusters

MapReduce: Simplied Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

Google, Inc.

Model

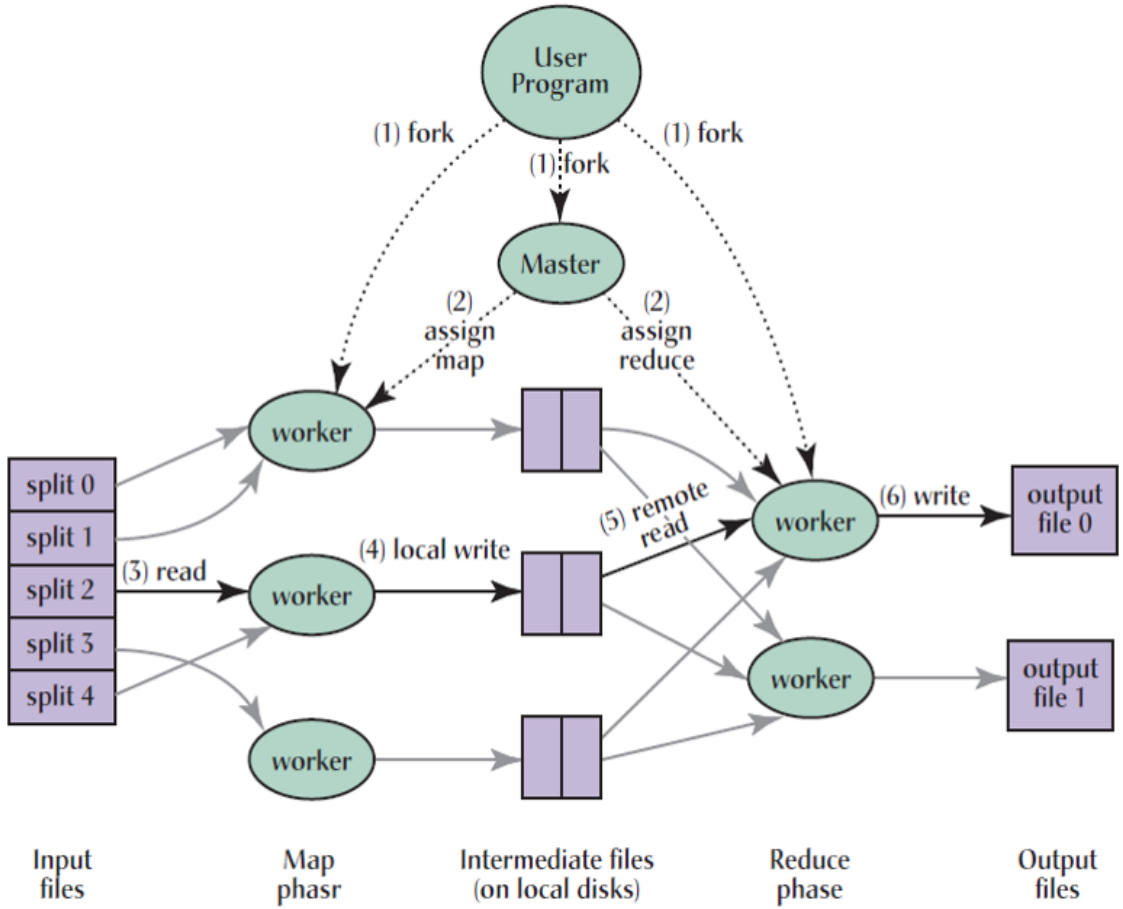
- Map
  - Takes an input pair and produces a set of intermediate key/value pairs
  - The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the reduce function
- Reduce
  - Accepts an intermediate key and a set of values for that key and merges these values together to form a possibly smaller set of values

Model

- Map
  - Takes an input pair and produces a set of intermediate key/value pairs
  - The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the reduce function
- Reduce
  - Accepts an intermediate key and a set of values for that key and merges these values together to form a possibly smaller set of values

Implementation

User Program
Master
Worker (Map/Reduce)
Fault tolerance
Locality
Task granularity
Backup tools



Performance

Grep

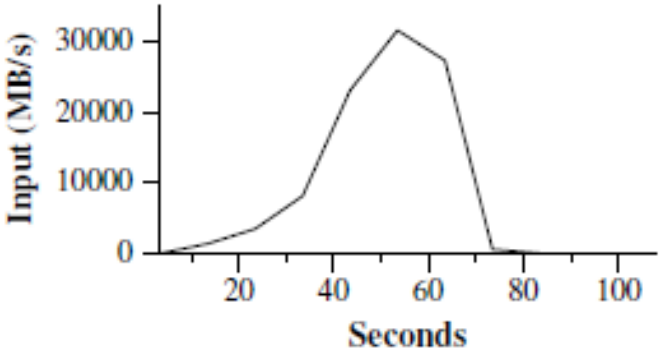
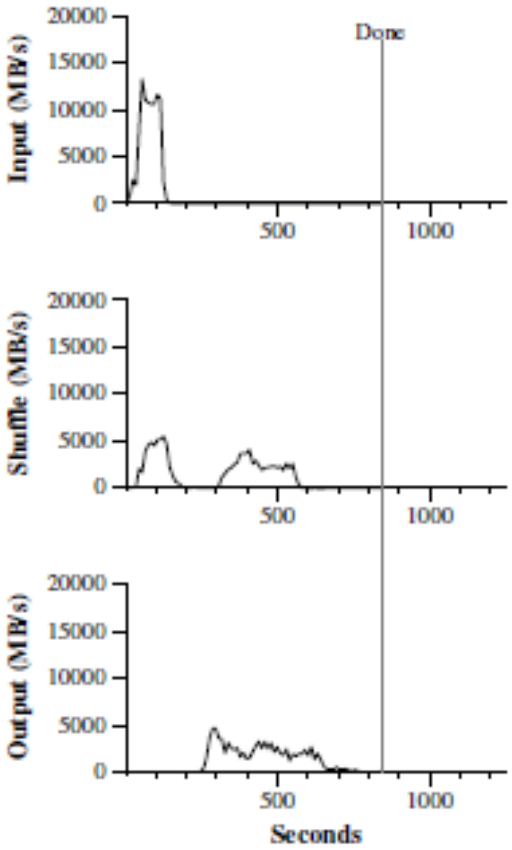


Figure 2: Data transfer rate over time

Performance (cont.)

Sort



(a) Normal execution

Bigtable: A Distributed Storage System for Structured Data

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

Google, Inc.

Model

- Rows
  - Atomic
  - Tablets
- Column Families
  - Family : qualifier
  - Access control
- Timestamps
  - Decreasing order
  - Garbage collect
- Cell
  - (row : string, column : string, time : int64) -> string

Model

- Rows
  - Atomic
  - Tablets
- Column Families
  - Family : qualifier
  - Access control
- Timestamps
  - Decreasing order
  - Garbage collect
- Cell
  - (row : string, column : string, time : int64) -> string

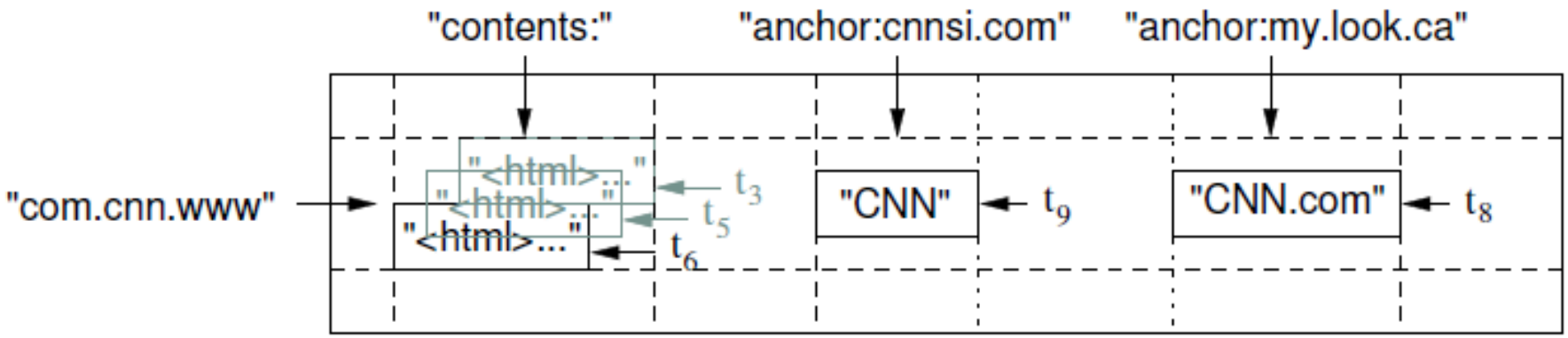
Model

- Rows
  - Atomic
  - Tablets
- Column Families
  - Family : qualifier
  - Access control
- Timestamps
  - Decreasing order
  - Garbage collect
- Cell
  - (row : string, column : string, time : int64) -> string

Model

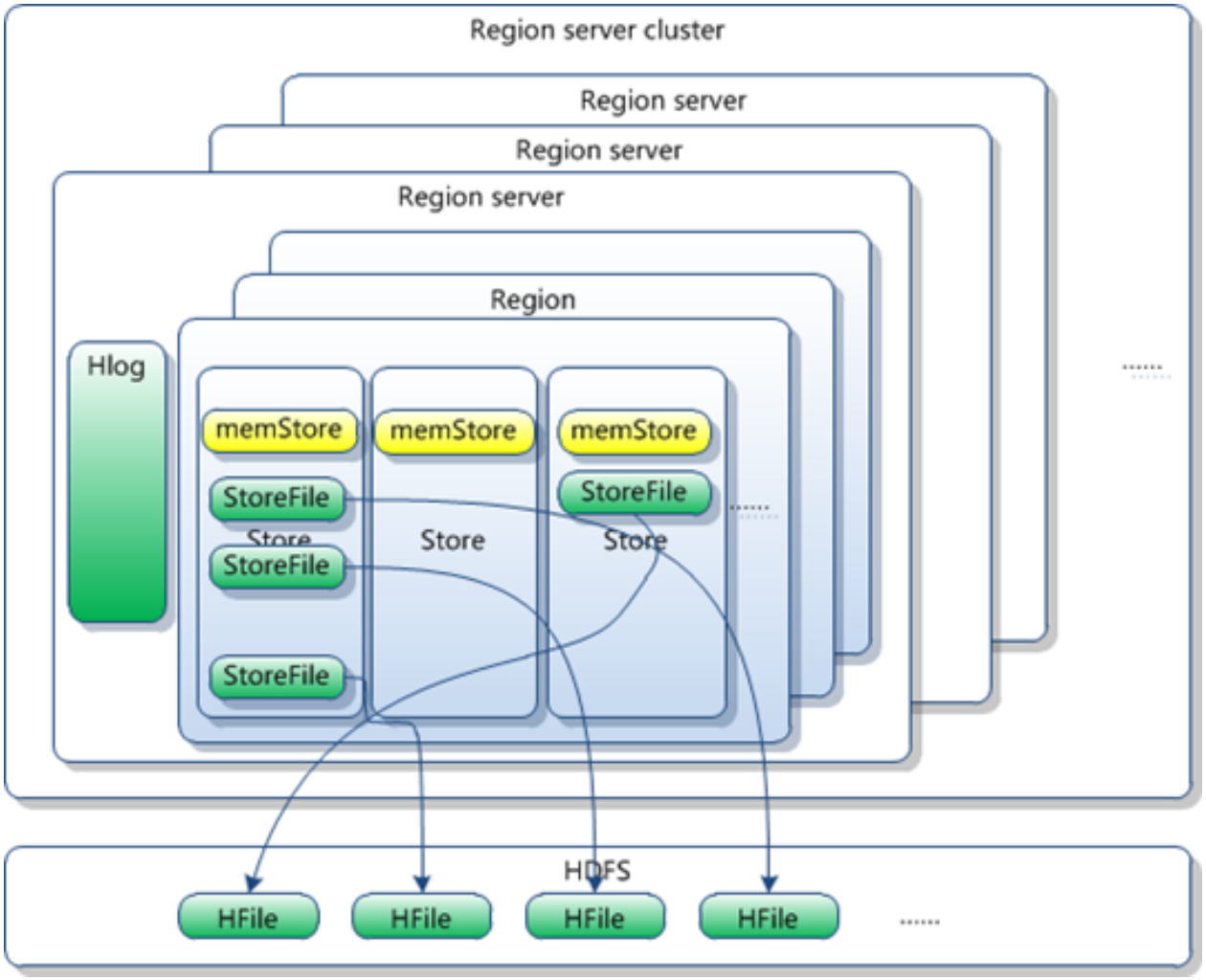
- Rows
  - Atomic
  - Tablets
- Column Families
  - Family : qualifier
  - Access control
- Timestamps
  - Decreasing order
  - Garbage collect
- Cell
  - (row : string, column : string, time : int64) -> string

Model (cont.)


"contents:"
"anchor:cnnsi.com"
"anchor:my.look.ca"
"com.cnn.www"
"<html>..."
"<html>..."
"<html>..."
t3
t5
t6
"CNN"
t9
"CNN.com"
t8

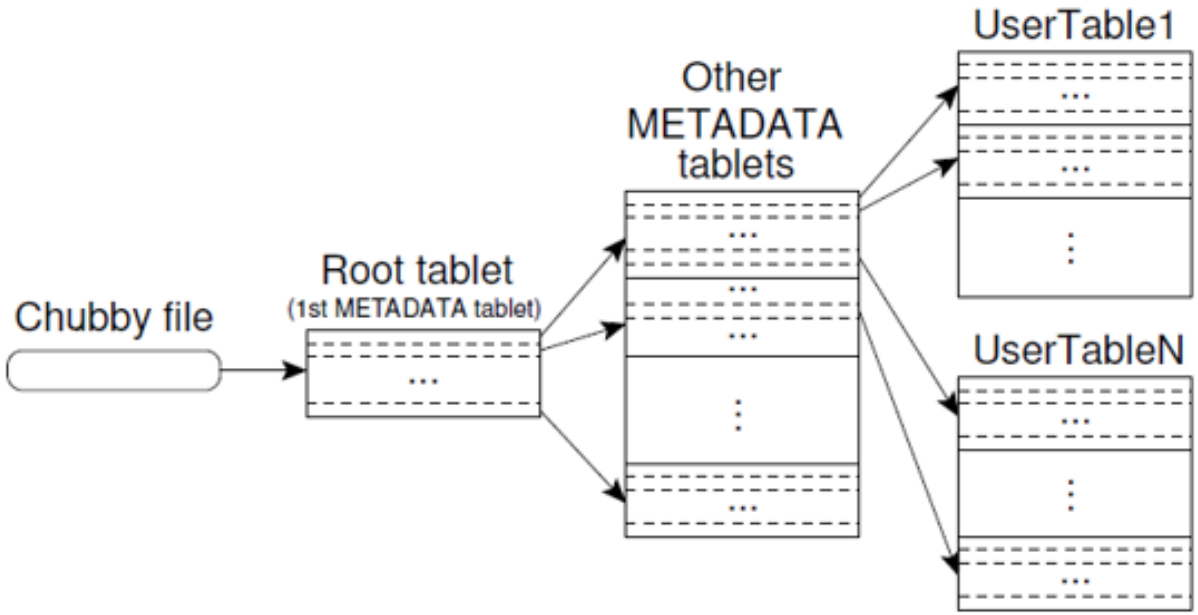
Storage

Region
–Store
—memStore
—StoreFile
File
–Blocks
Log
–Write ahead log



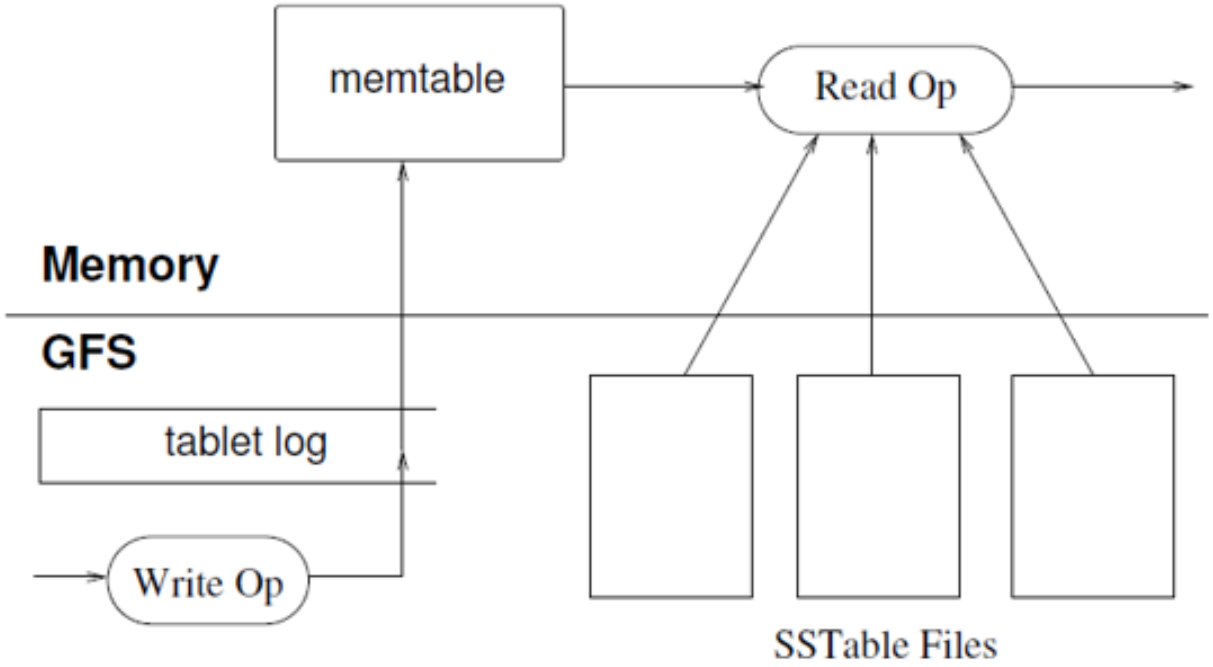
Implementation

- Tablet Location



Implementation (cont.)

- Tablet Serving



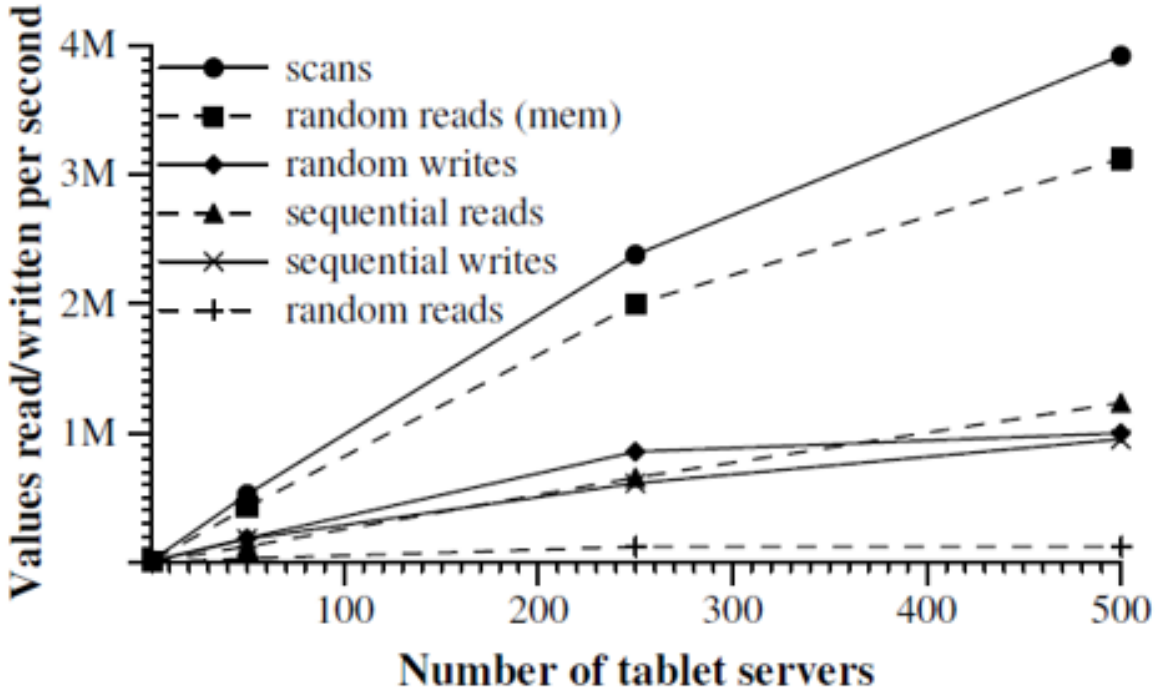
Evaluation

- Benchmarks

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

Evaluation (cont.)

- Scaling



Evaluation (cont.)

- Applications

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

Discussion

- Contributions of GFS, MapReduce, Bigtable
  - Supporting large-scale data processing workloads on commodity hardware
  - Easy to use, hides details of parallelization, fault tolerance, locality optimization, and load balancing
  - Scale well both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving)
- Drawbacks of GFS, MapReduce, Bigtable
  - Single master may be still a potential bottleneck
  - Disk I/O, time consuming
  - Do not support many database operations (multi-row transaction, secondary index . . .)

Discussion

- Contributions of GFS, MapReduce, Bigtable
  - Supporting large-scale data processing workloads on commodity hardware
  - Easy to use, hides details of parallelization, fault tolerance, locality optimization, and load balancing
  - Scale well both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving)
- Drawbacks of GFS, MapReduce, Bigtable
  - Single master may be still a potential bottleneck
  - Disk I/O, time consuming
  - Do not support many database operations (multi-row transaction, secondary index . . .)

References I

- [1] The Google File System. SOSP. 2003.
- [2] MapReduce: Simplied Data Processing on Large Clusters. OSDI. 2004.
- [3] Bigtable: A Distributed Storage System for Structured Data. OSDI. 2006.
- [4] The Hadoop Distributed File System. MSST. 2010.
- [5] Hadoop: the definitive guide. 2012.
- [6] HBase: the definitive guide. 2011.
- [7] Data-intensive text processing with MapReduce. 2010.
- [8] The Chubby lock service for loosely-coupled distributed systems. OSDI. 2006.