COMP5338 – Advanced Data Models

Week 5: Column Store and Google Bigtable

Dr. Ying Zhou
School of Information Technologies

THE UNIVERSITY OF SYDNEY

Administrative

- There will be a quiz on Week 6
  - Covers week 1- week 5 content
  - Paper based
  - It is running on Tuesday evening 8-9pm
  - All Tuesday classes please go to your allocated tutorial rooms
  - All Wednesday e theatre for the quiz
  - There is no regu

Outline

- Overview
  - Row Store vs. Column Store
  - Bigtable motivation
- Bigtable Data model
- Bigtable Architecture

Organization of Disk Based Storage System

MongoDB file structure

https://blog.mlab.com/2014/01/how-big-is-your-mongodb/

Column Store From RDBMS Perspective

- Row store is eas but might read in unnecessary dat columns
  - Good for OLTP type of application
- Column store is good for read and analysis relevant data but requires multiple accesses to update a row
  - Good for OLAP (data warehouse type of application)
- The only fundamental difference is storage layout!

From Stavros Harizopoulos, Daniel Abadi, Peter Boncz VLDB 2009 Tutorial

Row Store vs. Column Store

- Row store or NSM (N-ary Storage Model) is used in most database management systems
  - Many relational database systems
  - Considered in general as write optimized
  - MongoDB is a “row store”
    - All data in a document is placed contiguously in storage
    - Schema less fe hallenging as document may grow or shrink i
  - Compression is less efficient as row c data
- Column store or DSM (Decomposed l)
  - The idea is proposed in 1985, the real practical modern implementation is C-Store from MIT by Stonebraker et. al in 2005
  - Google’s BigTable is influence by DSM principle
    - With distinct key-value features
    - So does HBase

Bigtable Motivation

- Some of Google’s daily business
  - Query
    - A whole copy of the web
    - Links between pages
  - Personalized Search
    - User’s query
  - Google Analytic
    - Traffic data (who visits what at w w long)
  - Google Earth
    - Satellite images, geo information
  - And so on..

How are Data Accessed

- Web pages
  - Scanned to build inverted index (word -> page)
    - Unstructured, sequential read
- Page meta data, links between pages
  - Used to rank pages, to compute PageRank algorithm
    - Structured, ra ies
- Query history, c
  - Used to build profile and run pers gorithm
    - Structured, random access, point or range query
- Traffic data
  - Used to build summary statistics
    - Structured, random access, point or range query

Google Storage Systems

- Typical Data/Access Features
  - Massive scale data set of structured or unstructured data
  - Sequential or simple random access, majority of the data updates are “append”
- Storage systems to cater for such data storage/access
  - Google File System (SOSP’03 paper)
    - Unstructured da
  - BigTable (OSDI’0
    - Structured data, random access
- More recent storage system to cater f s’ desire to use SQL
  - MegaStore (CIDR’11 paper)
    - Build on top of Bigtable, an effort to combine the scalability of NoSQL and the convenience of a RDBMS
  - Spanner (OSDI’12 paper)
    - A successor to BigTable with more relational features and better performance than MegaStore
    - There is a recent SIGMOD’17 paper focusing on how SQL is implemented

Outline

- Overview
- **Bigtable Data model**
- Bigtable Archit

Data Model

- “A Bigtable is a **sparse**, distributed, persistent **multidimensional sorted map**”
- Basic concepts: table, row, column family, column, timestamp
  - (rowkey:string, c e:int64) -> value: string

- Example bigta es
  - Stores the data about “home n website

    - The URL is “www.cnn.com”
    - The language is “EN”
    - The content is “<html> …</html>”
    - It is referenced by two other pages
      - Sports Illustrated (**cnnsi.com**) , using an anchor text “CNN”
      - My-Look (**my.look.ca**), using an anchor text “CNN.com”

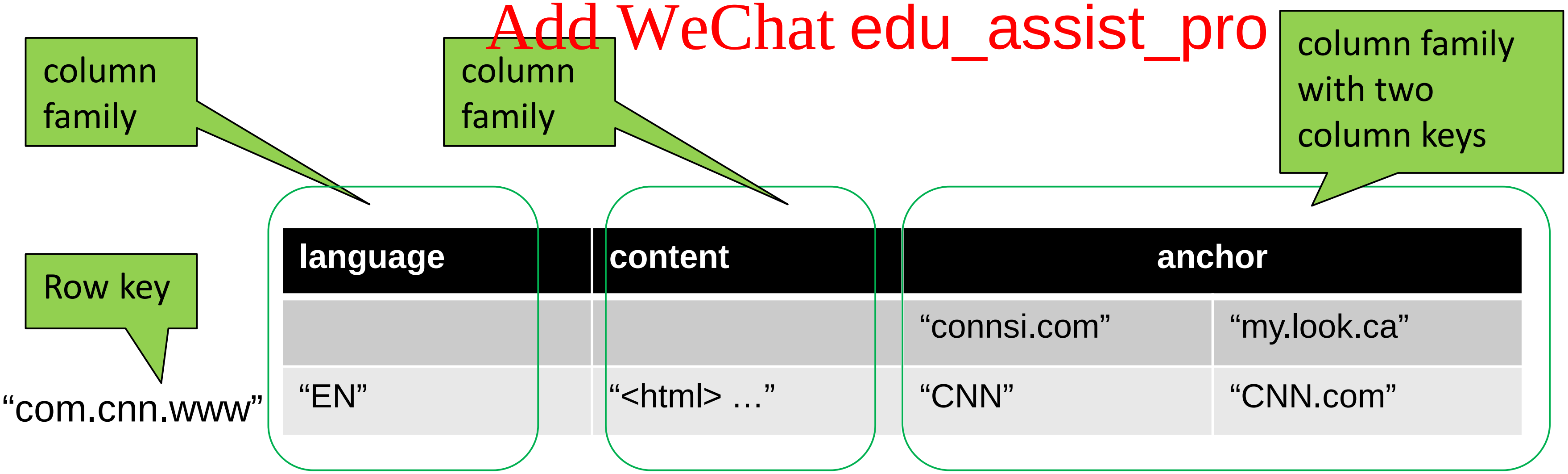
Relational Data Model vs Bigtable Model

web table

url	language	content
“www.cnn.com”	“EN”	“<html> … </html>”

link table

url	referencingUrl	anchorText
“www.cnn.com”		CNN”
“www.cnn.com”		CNN.com”



	language	content	anchor	
			“connsi.com”	“my.look.ca”
“com.cnn.www”	“EN”	“<html> …”	“CNN”	“CNN.com”

Rows

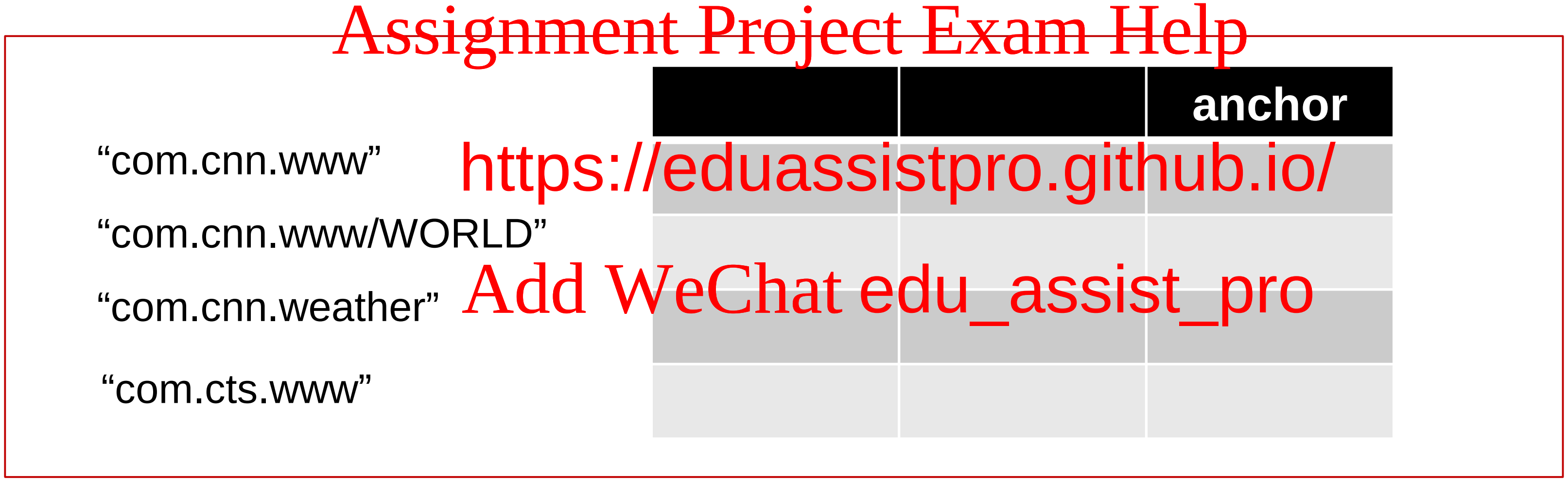
sorted

	language	content	anchor
“com.cnn.www”			
“com.cnn.www/WORLD”			
“com.cnn.weather”			
“com.cts.www”			

- Row keys are arbitrary strings
- Read/write of data under a single ro [illegible] ic
- Row keys are sorted in lexicographic order
- Large table is dynamically partitioned by row key ranges
  - Each partition is called a **tablet**
  - Nearby rows will usually be served by the same server
  - Accessing nearby rows requires communication with a small number of machines

Table Splitting

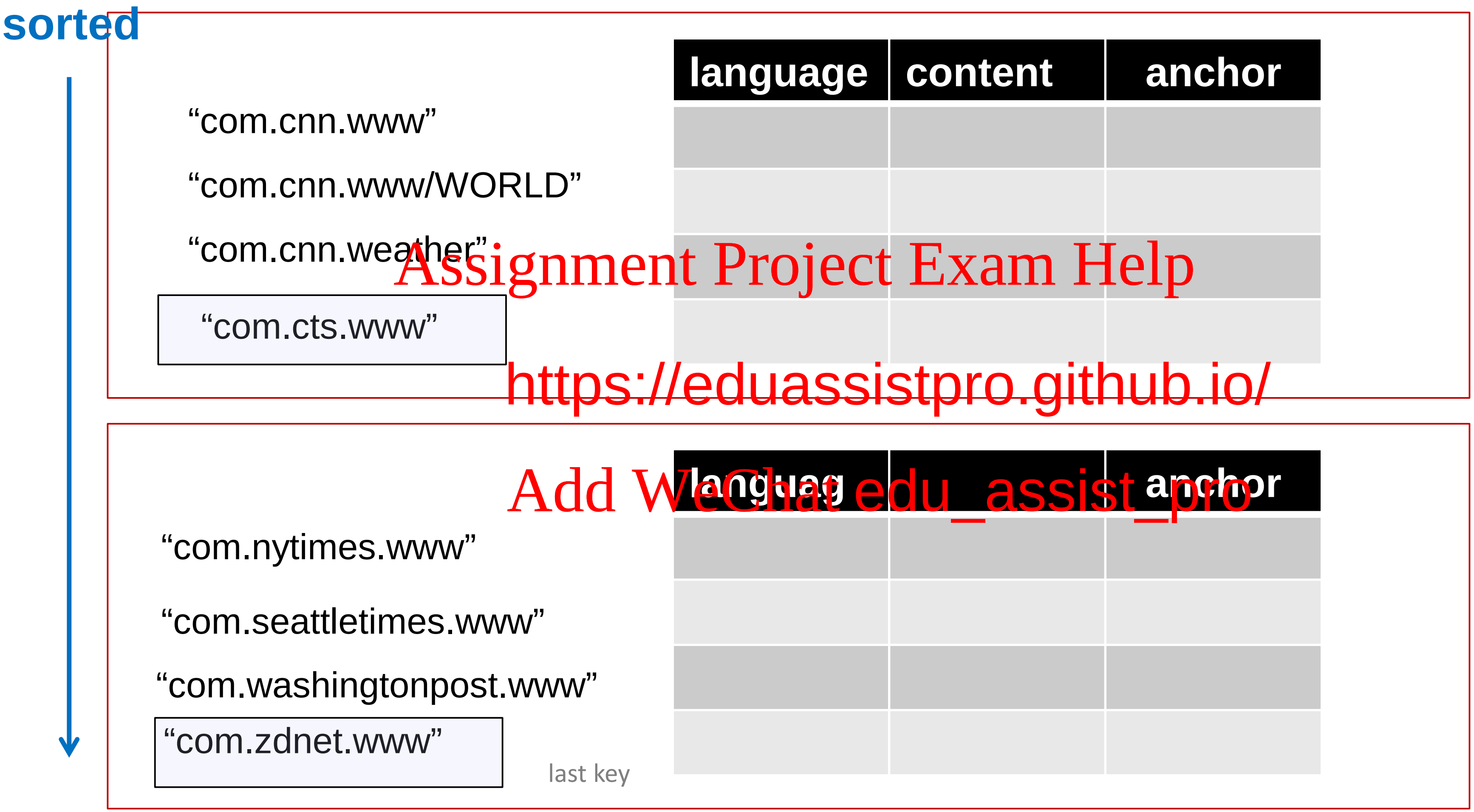
- A table starts as one tablet
- As it grows it splits into multiple tablets
  - Approximate size: 100-200 MB per tablet by default



			anchor
“com.cnn.www”			
“com.cnn.www/WORLD”			
“com.cnn.weather”			
“com.cts.www”			

One tablet

Table Splitting (cont’d)



sorted

	language	content	anchor
“com.cnn.www”			
“com.cnn.www/WORLD”			
“com.cnn.weather”			
“com.cts.www”			

	language	content	anchor
“com.nytimes.www”			
“com.seattletimes.www”			
“com.washingtonpost.www”			
“com.zdnet.www”			

last key

Columns and Column Families

- Relational model only has “row” and “column” concepts
- Bigtable has “row”, “column” and “column family” concepts
- Column family
  - Just a group of columns with a **printable name**
  - Each **column** inside a **column family** has a **column key**
    - Column key is [illegible]
- Column family c[illegible]nient “collection” type data at design
  - It also determines how table’s data are stored
- Column family is the basic unit of data access
- Data stored in a column family is usually of the same type

Columns and Column Families (cont'd)

- Column Family is part of the **schema definition**
  - When we create a table, we also create a few column families by specifying their names
  - The number of column families in a table is typically small and relatively stable
    - Less than hun
  - A column family ited number of columns
    - The row could be very "wide"
    - E.g. a popular web page in the w referenced by thousands, or even millions of other pages
    - *Implications*: we may have some tablet storing only one row!

Column Family Examples

- The web table example has three column families
  - “language” -- with only one column to store a web page’s language
    - Each web page can only have one language
    - Just like a normal column in relational table
    - Column key is “language”
  - “content” -- agai re the actual HTML text
    - Column key is
  - “anchor” -- with dynamic number
    - Each web page may be referenc number of other pages
    - E.g. *www.cnn.com* page has two referencing sites
    - Column key is “anchor:<referencing site url>”
      - Question: *Why can’t we use “anchor:<anchor text>” as column key?*

Timestamps

- Classic relational model can only store the “current” value of a particular row and its columns
  - Temporal DB may be able to store valid/transaction time
- Bigtable stores multiple versions of a column by design
- Version is index p
  - System time or
  - If system time is ansaction time
  - Client assigned time can have va gs
- Per-column-family settings for garbage collection
  - Keep only latest **n** versions
  - Or keep only versions written since time **t**
- Retrieve most recent version if no version specified
  - If specified, return version where timestamp ≤ requested time

Web Table with Timestamp

	language	content	anchor	
			“connsi.com”	“my.look.ca”
“com.cnn.www”	“EN” ← t1	“<html>…” ← t1 “<html>…” ← t2 “<html>…” ← t6	“CNN” ← t9	“CNN.com” ← t7

- The sorted map concept
  - (**rowkey:string**, **columnKey:string** **e:int64**) -> value: string
  - Examples:
    - (**“com.cnn.www”, “language:”, t1**) -> “EN”
    - (**“com.cnn.www”, “anchor:consi.com”, t9**) -> “CNN”

Typical APIs

- Data definition API
  - Create/delete table and column families
  - Update table/column family metadata
- Data Manipulation API
  - Write or delete value as specified by rowkey and some column qualifier
  - Look up specific
  - Scan a short range of rows
  - Support single row transaction

Outline

- Overview
- Bigtable Data model

- **Bigtable Archit**
  - **Immutable SSTabl**

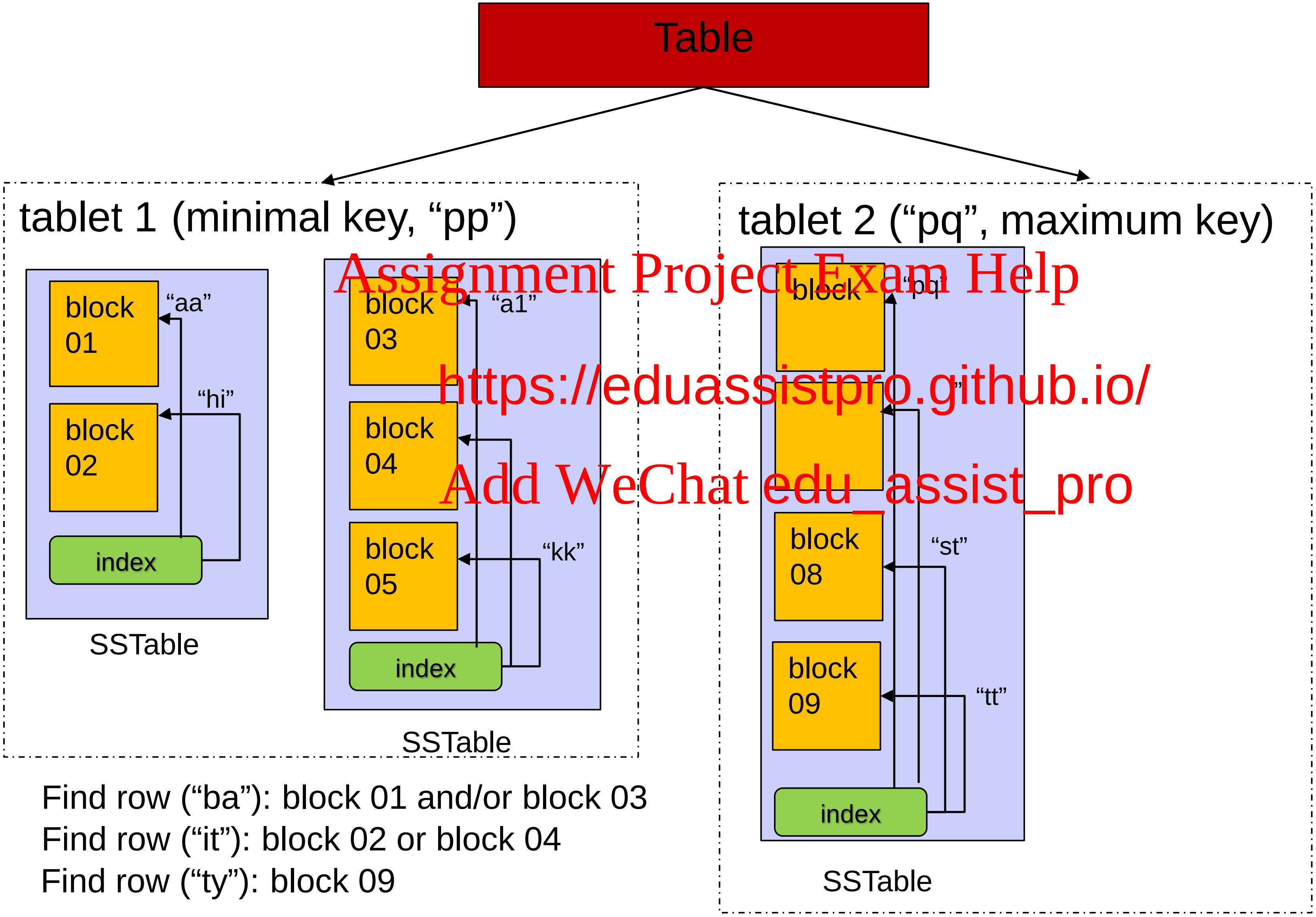
  - **Master-Tablet Server Architecture**

  - **Chubby Services**
  - **Read/Write Path**
  - **HBase**

Data Storage

- Google File System (GFS)
  - Is used to store actual Bigtable data (log and data files)
  - It provides replication/fault tolerance and other useful features in a cluster environment
- Google SSTa
  - Bigtable data SSTable format
  - Each SSTable consists of
    - Blocks (default 64KB size) to store **ordered** *immutable* map of key value pairs
    - Block index
- The SSTable is stored as GFS files and are replicated

Table-Tablet-SSTable



Find row (“ba”): block 01 and/or block 03
Find row (“it”): block 02 or block 04
Find row (“ty”): block 09

Architecture

- **Many** *tablet servers*
  - Can be added or removed dynamically
  - Each ***manages*** a set of tablets (typically 10-1,000 tablets/server)
  - Handles **read/write** requests to tablets
  - Splits tablets when too large

- **One** *master serv*
  - Assigns tablets t
  - Balances tablet server load
  - Garbage collection of unneeded files
  - Schema changes (table & column family creation)
  - It is **NOT** in the read/write path
- Client library

Tablet Location

- METADATA table contains the location of all tablets in the cluster
  - It might be very big and split into many tablets
- The location of METADATA tablets is kept in a root tablet
  - This can never be split
- Each tablet is assigned to **ONE** tablet server at a time.
- Both ROOT and M [illegible] ed by tablet servers as well

Chubby Services

- Chubby is distributed lock service consists of a small number of nodes (~5)
  - Each is a replica of one another
  - One is acting as the master
  - Paxos is used to ensure majority of the nodes have the latest data
- Chubby allows c /file and locks on them
  - Lock has short lease time and ne wed periodically
- Usage in Bigtable
  - Ensure there is only one master
  - Keep track of all tablet servers
  - Stores the root table location
  - If Chubby becomes unavailable for an extended period of time, Bigtable becomes unavailable.

Chubby and Tablet Servers

- Tablet servers are able to join or leave a running cluster without interfering the normal cluster operation
- Chubby is used to keep track of tablet servers
  - Normal handling
- Each server creates & locks a unique file in Server *Directory* when it starts
- The lock has short l ed periodically
- If a tablet server is r, it will release its lock
  - Error handling
- A tablet server may lose the lock (e.g
  - It will stops serving the tablets
  - It will report to master that the lock is lost
  - It will attempt to reclaim the lock if the file still exists, otherwise it kills itself
- A tablet server may crash and its file become orphaned
  - Master will come to the rescue

Chubby and Master Operation

- Master also obtains an *exclusive master* lock from chubby to ensure there is only one master server
- Master monitors Chubby’s *server directory* to find the current list of tablet servers in the cluster
- Master detects t rs by periodically ask each server

- Error handling
  - If tablet server is alive but has no tablet server is unreachable
    - The master will contact Chubby to acquire a lock on the orphaned server file and delete it
    - The master also assigns all tablets to other servers
  - If a master cannot contact Chubby to renew its lock, it kills itself

Master Start Up

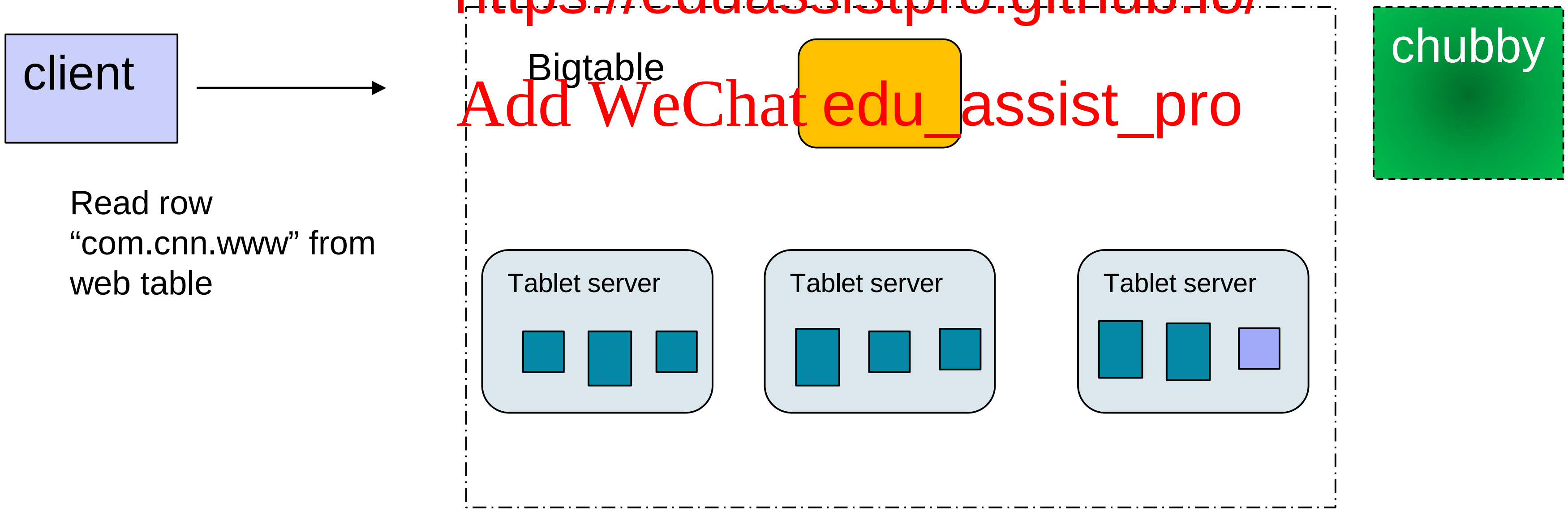
- When a master is started
  1. It grabs a unique master lock in Chubby
  2. Find out all live servers
  3. Communicate with all servers to find out what tablets they serve
  4. Scan the METADATA table to find the total set of tablets in the cluster
     - May discove d
  5. Assign tablets blet server
- Any cluster has a root tablet, in master may
  - Find the server that manages th **root** nd proceed with step 4
  - Find that the **root** tablet is not assigned to any server, the master will assign it to a server and proceed with step 4

Tablet Assignments

- Master knows the initial set of tablets during start up process
- Master assign tablets to servers to balance the load
- The set may change
  - When tables are
  - Two tablets are
  - An existing tabl s
- The master initiate the first two date tablet assignment accordingly
- The splitting is initiated by tablet server and the information of the new tablet will be updated in the **METADATA** table
- The tablet server also notifies the master of such change

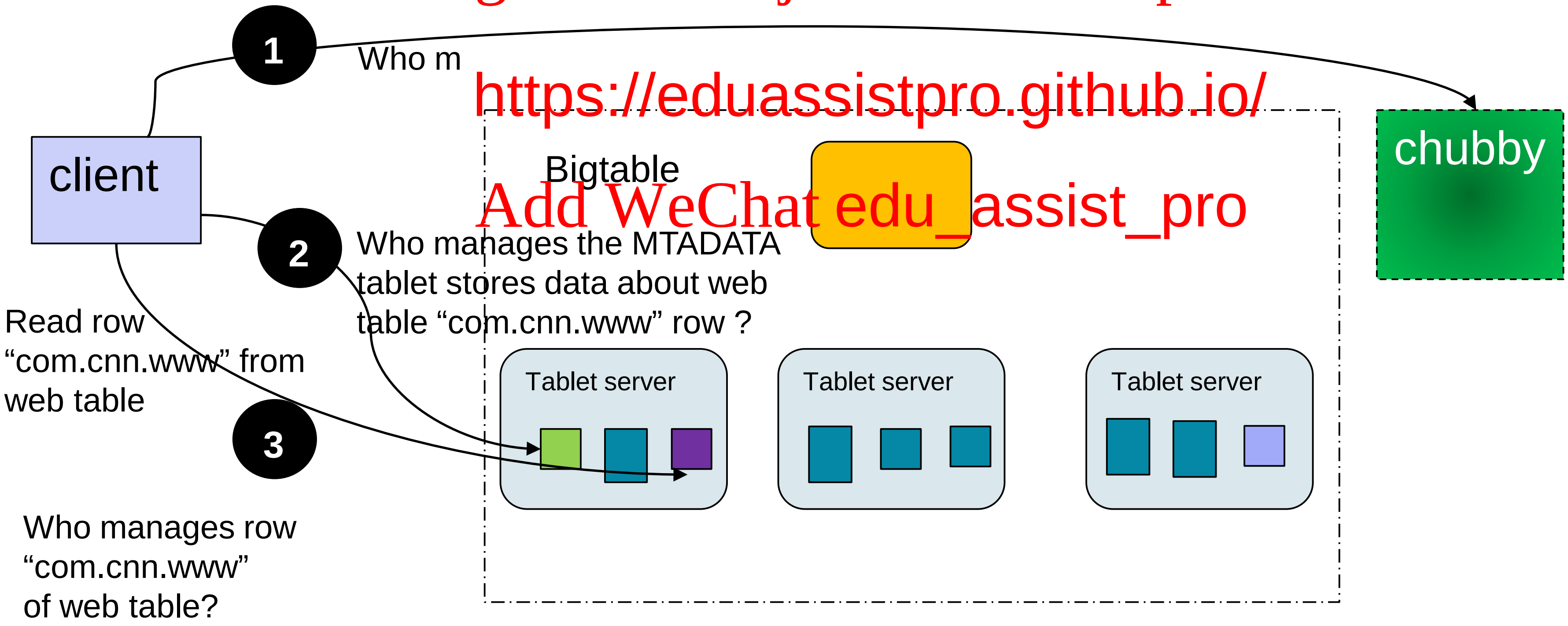
Tablet Serving

- Client read/write request
  - E.g. client wants to read the row corresponding to “com.cnn.www” from the web table
- Steps
  - Find the *tablet location*, the table server that serves the tablet
  - Contact the table writhe request



Find the tablet server

- If the client is requesting the data for first time
  - One round trip from chubby to find the root tablet's location
  - One round trip to the tablet server manages the root tablet
  - One round trip to the tablet server manages the METADATA tablet
- The client caches the tablet location for later use



client

1

Who m

chubby

Bigtable

2

Who manages the MTADATA tablet stores data about web table "com.cnn.www" row ?

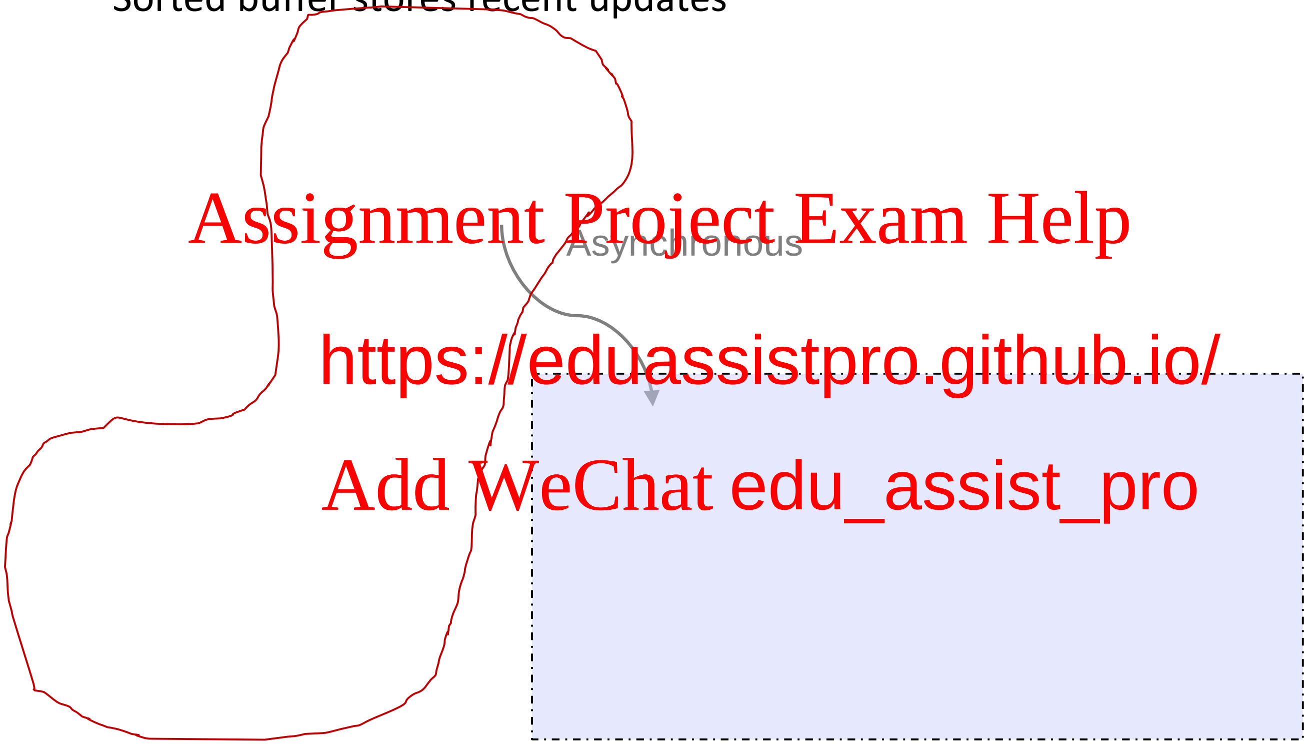
Read row "com.cnn.www" from web table

3

Who manages row "com.cnn.www" of web table?

Tablet Representation

Sorted buffer stores recent updates



Asynchronous

Write operation

SSTable files of one tablet, which persists earlier updates

Tablet Representation Implications

- A tablet server *manages* many tablets
  - Its memory contains latest updates of those tablets
  - BUT, the actual persisted data of those tablets might not be stored in this tablet server
    - Logs and SSTable Files are managed by the underlying file system GFS
    - GFS might re
- Bigtable system tual file replication and placement
- The separation of concern sim esign

Write Path

- A write operation may insert new data, update or delete existing data
- The client sends write operation directly to the tablet server
  - The operation is checked for syntax and authorization
  - The operation is written to the **commit log**
  - The actual muta **memtable**
    - Deleted data
- The only disk operation involve ath is to append update to commit log

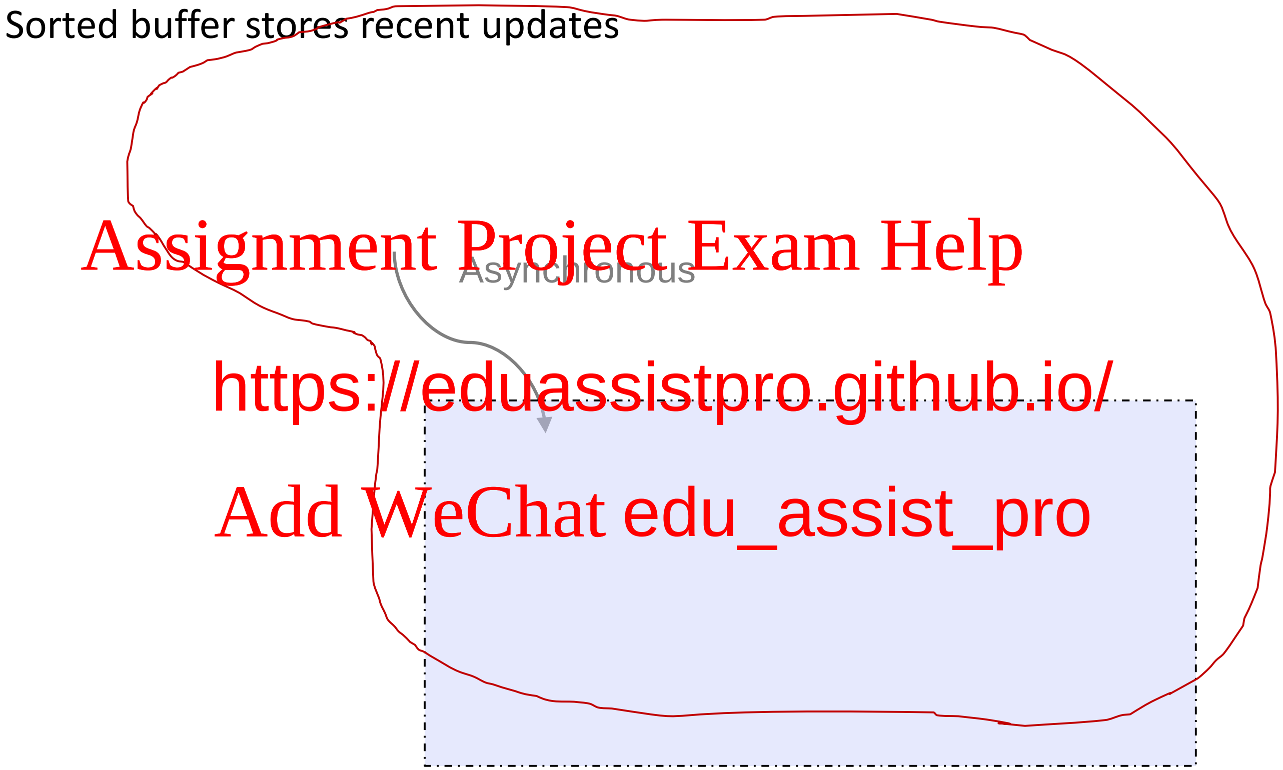
Compactions

- After many write operations, the size of memtable increases
- When memtable size reaches a threshold
  - The current one is frozen and converted to an SSTable and written to GFS
  - A new memtable is created to accept new updates
  - This is called **mi**
- Why minor com
  - Memory management of tablet se
  - Reduce the size of active log entries
    - Minor compaction persists the updates on disk
    - Log entries reflecting those updates are no longer required

Compactions (cont'd)

- Every **minor compaction** creates a new SSTable
  - A tablet may contain many SSTable with overlapping key ranges
- **Merging compaction** happens periodically to merge a few SSTables and the current memtable content into a new SSTable
- **Major compacti** ntents into a single SSTable. It will p deleted data.

Read Path



Sorted buffer stores recent updates

Read Path

- The client sends read operation directly to the tablet server
  - The operation is check for syntax and authorization
  - Both memory and disk maybe involved to obtain the data
- What are kept in memory
  - Most recent updates in memtable (sorted by key)
  - Block indexes of S
- What are kept in dis
  - Earlier updates pe files
- How does tablet server find the data
  - Check if the memtable contains partial data, or special mark indicating certain data is deleted
  - Check the index to find the block(s) that may contain partial data
  - Load the block and extract the data if there is any
  - Combine the data from memtable and disk block to obtain the final result

Recover a Tablet

- Tablet may be re-assigned to a new tablet server as part of load balancing or recovery process
- The assignment is initiated by master sending a **load tablet** request to a tablet server.
- Upon receiving s rver performs the following:
  - Scan the META n about this tablet
    - List of SSTables
    - Log file
  - Read the block indexes in memory
  - Play the log file to reconstruct the memory with all updates are not yet persisted in SSTables

Refinements- Locality Group

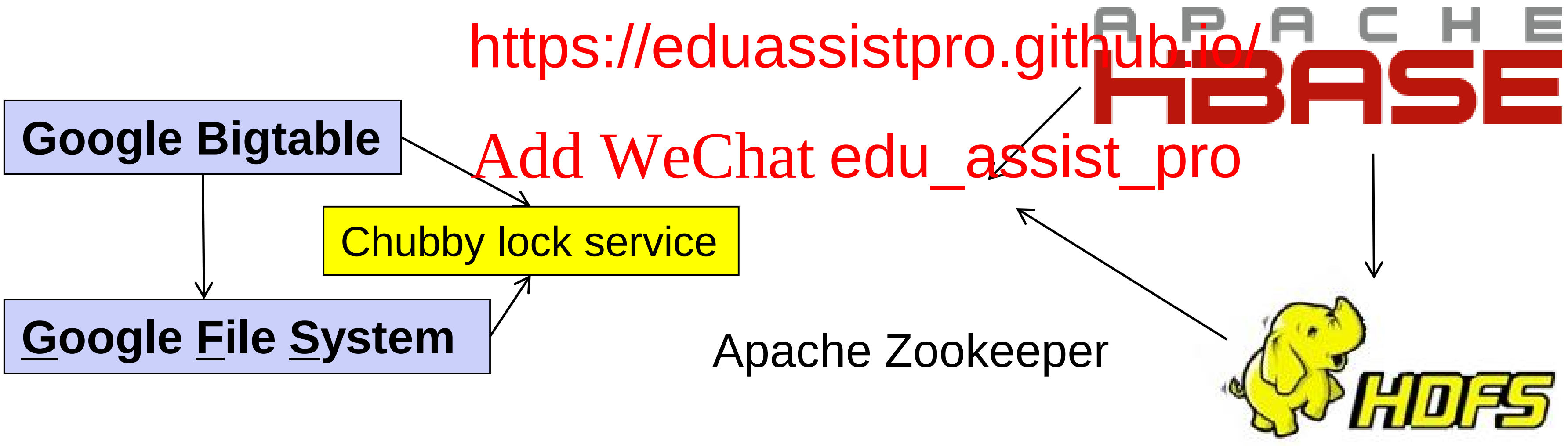
- Locality group consists of multiple column families specified by client
- There will be a separate SSTable for each locality group in each tablet.
  - “column based” storage
- Reasons
  - Bigtable support
  - Not all column families are requir [illegible] eration
  - Put column families that are typically access together in the same group enables more efficient read
    - E.g. web page’s metadata and actual content can be put in different groups

Sample Application – Google Analytics

- Raw Click Table (~200 TB)
  - Row for each end-user session
  - Row name: {website name and time of session}
  - Sessions that visit the same web site are sorted & contiguous
- Summary Table (~20 TB)
  - Contains variou e
  - Generated from dic MapReduce jobs

What is HBase?

- HBase is a column based NoSQL storage system based on Google’s Bigtable data model and architecture
- It is fully distributed
- It is not a general purpose storage system



Google Bigtable

Chubby lock service

Google File System

Apache Zookeeper

HBase and Bigtable Nomeclature

Bigtable	HBase
Tablet	Region
Tablet Server	Region Server
ROOT and METADATA tablet (two levels)	hbase:meta table (one level)
SSTable	
memtable	
Commit log	
Minor compaction	F
Merging compaction	Minor compaction
Major compaction	Major compaction
GFS	HDFS
Chubby	Zookeeper
Locality Group	By default, each column family is a locality group

References

- Google Storage Stake Reading List:

  - Sanjay Ghemawat, Howard Gobioff and Shun-Tak Leung, **The Google File System**, In *Proceedings of the 19th ACM Symposium on Operating Systems Principles* (SOSP'03), 2003
  - Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, **Bigtable: A Distributed Storage System for Structured Data, OSDI**'06: In Proceedings of the Seventh Symposium on Operating System Design and Implementation (OSDI'06), Seattle, WA, 2006
  - Jason Baker, Chris Bond, James C. Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, l. **Megastore: Providing Scalable, Highly Available Stor** IDR, 2011, pp. 223–234.
  - Corbett, James C; Dea w; Frost, Christopher; Furman, JJ; Ghemawat, Sanjay; Gubarev, Andrey; Heiser, C hild, Peter; Hsieh, Wilson; Kanthak, Sebastian; Kogan, Eugene; Li, Hongyi; Melnik, Sergey; Mwaura, David; Nagle, David; Quinlan, Sean; Rao, Rajes aito, Yasushi; Szymaniak, Michal; Taylor, Christopher; Wang, Ruth; Woodford, Dale, **Spanner: Google's Globally-Distributed Database** *Proceedings of OSDI, 2012*
  - Bacon, David F., et al. "Spanner: Becoming a SQL System." *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017.