Colum Stores & BigTable

Prof. Yanlei Diao

Some slides taken from the talks by Jeff Dean

Traditional Row Stores

- Review of the logical data model
  - A database has a collection of tables
  - Each table has a collection of columns, defined in the schema
  - Each tuple has a value or NULL for each column
- Review of the storage model: A Row Store
  - File: a table is implemented as a file
  - Pages: a file contains a chain of pages
  - Records: multiple records are packed in a page

Why row stores?

- Natural: storage model similar to the logical model
- Good performance for writes (inserts, updates, deletes)

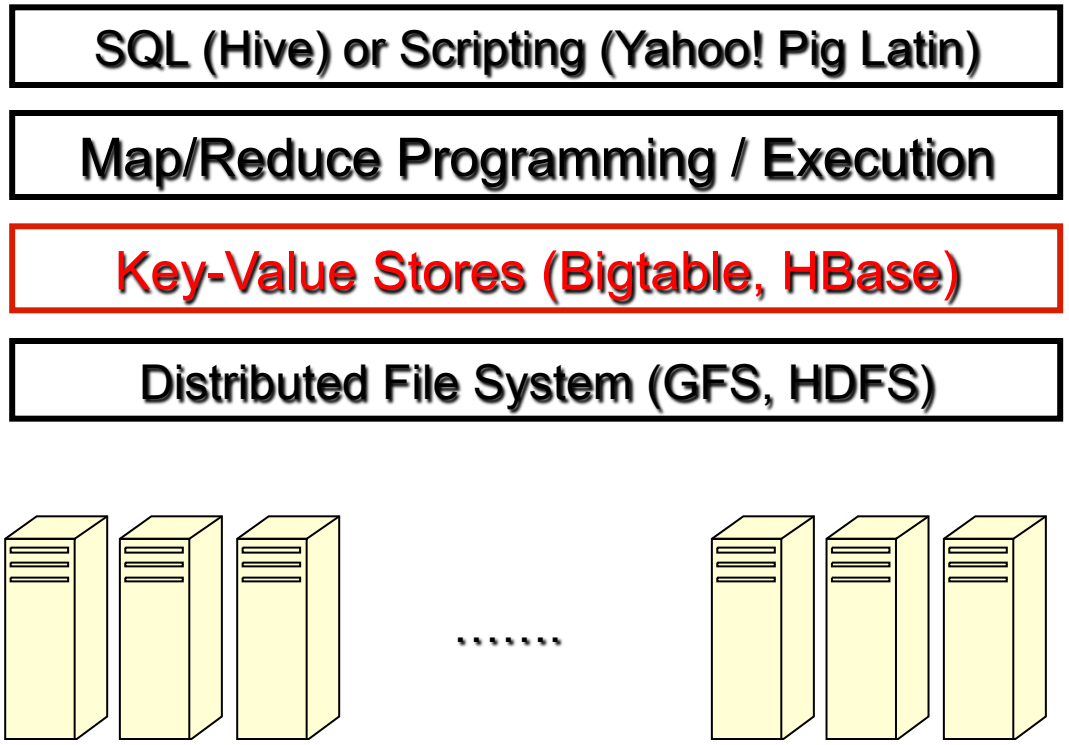
Column Stores

- Change the storage model w.o. changing the logical model
  - Store a table in a column-based fashion
- Why column stores?

  - IO performance: save I/O if queries only access a few columns
  - Storage efficiency: compression is easier.
  - Flexible for schema expansion: adding a new column is easy
  - Great for sparse tables or denomalized schema
  - Easy replication of a column with diff. ordering properties
- Potential problems?
  - Expensive updates: suitable for read-most workloads
  - Potentially many (expensive) joins

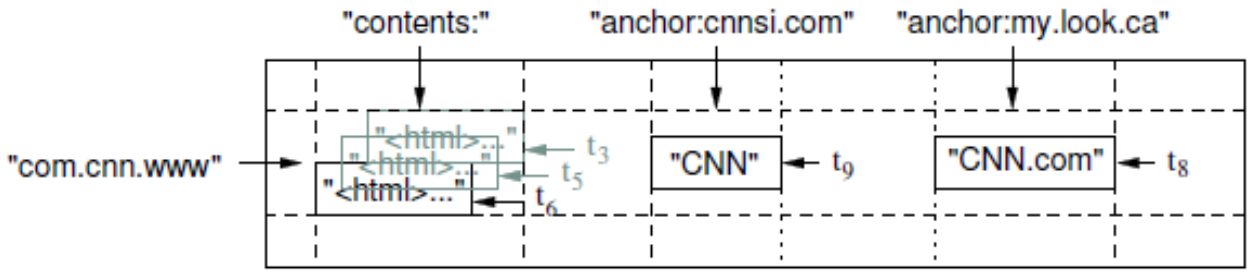
Overall Architecture

- Shared-nothing architecture of thousands of nodes!
  - A node is an off-the-shelf, commodity PC.

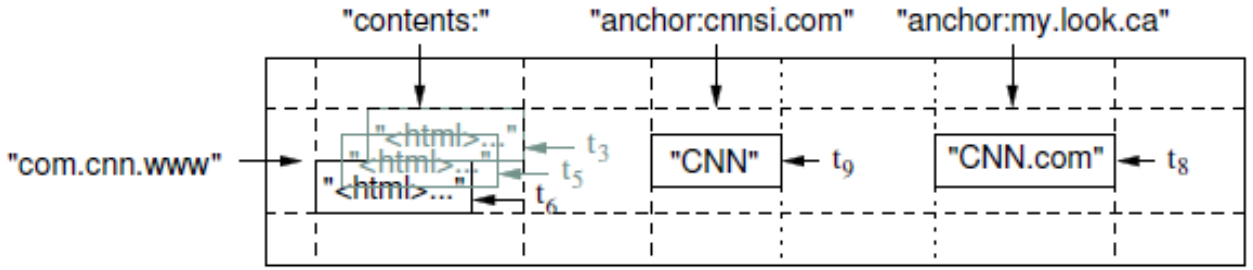


Google Bigtable

- A data model (a schema).
- A *sparse, distributed persistent multi-dimensional sorted map*.
- Data is partitioned across the nodes seamlessly.
- The map is indexed by a row key, column key, and a timestamp.
  - Output value in the map is an un-interpreted array of bytes.
  - (row: byte[ ], column: byte[ ], time: int64) → byte[ ]

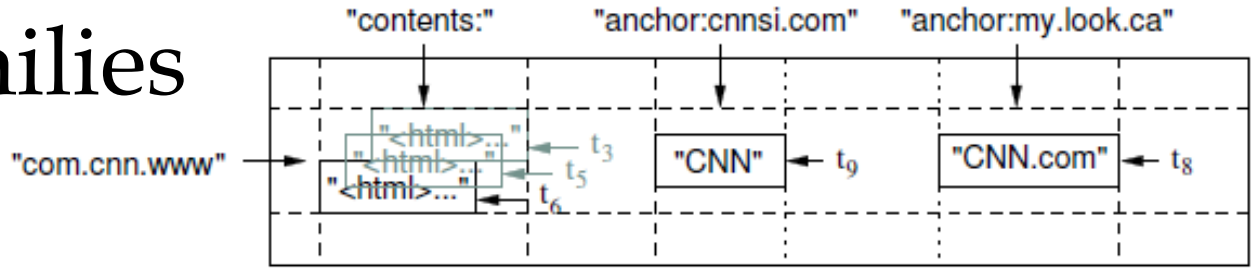


Rows



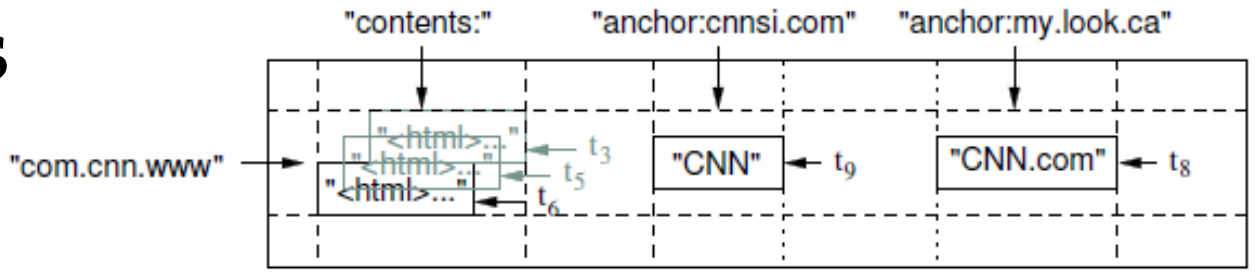
- A row key is an arbitrary string.
  - Typically 10-100 bytes in size, up to 64 KB.
  - Every read or write of data under a single row is atomic.
- Data is maintained in lexicographic order by row key.
- The row range for a table is dynamically partitioned.
  - Each partition (row range) is named a *tablet*.
    - Unit of distribution and load-balancing.
- Objective: make read operations single-sited!
  - E.g., In Webtable, pages in the same domain are grouped together by reversing the hostname components of the URLs: com.google.maps instead of maps.google.com.

Column Families



- Column keys are grouped into sets called column families.
- A column family must be created before data can be stored in a column key.
- Hundreds of static column families.
- Syntax is family:key
  - e.g., Language:English, Language:German, etc.

Timestamps



- Store different versions of data in a cell in decreasing timestamp order.
  - 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"*

Bigtable Usage

- Used in different applications supported by Google.

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

App 1: Google Analytics

- Enables webmasters to analyze traffic pattern at their web sites. Statistics such as:
  - number of unique visitors per day and the page views per URL per day,
  - percentage of users that made a purchase given that they earlier viewed a specific page.
- How?
  - A small JavaScript program that the webmaster embeds in their web pages.
  - Every time the page is visited, the program is executed.
  - Program records the following information about each request:
    - User identifier
    - The page being fetched…

App 1: Google Analytics (Cont…)

- Two of the Bigtables
  - Raw click table (~ 200 TB)
    - A row for each end-user session.
    - Row name includes the website's name and the time at which the session was created.
    - Clustering of sessions that visit the same web site. And a sorted chronological order.
    - Compression factor of 6-7.
  - Summary table (~ 20 TB)
    - Stores predefined summaries for each web site.
    - Generated from the raw click table by periodically scheduled MapReduce jobs.
    - Each MapReduce job extracts recent session data from the raw click table.
    - Row name includes the website's name and the column family is the aggregate summaries.
    - Compression factor is 2-3.

App 2: Google Earth & Maps

- Functionality: Pan, view, and annotate satellite imagery at different resolution levels.
- One Bigtable stores raw imagery (~ 70 TB):
  - Row name is a geographic segment. Names are chosen to ensure adjacent geographic segments are clustered together.
  - Column family maintains sources of data for each segment.

- There are different sets of tables for serving client data, e.g., index table.

App 3: Personalized Search

- Records user queries and clicks across Google properties.
- Users browse their search histories and request for personalized search results based on their historical usage patterns.
- One Bigtable:
  - Row name is userid
  - A column family is reserved for each action type, e.g., web queries, clicks.
  - User profiles are generated using MapReduce.
    - These profiles personalize live search results.
  - Replicated geographically to reduce latency and increase availability.

Bigtable API

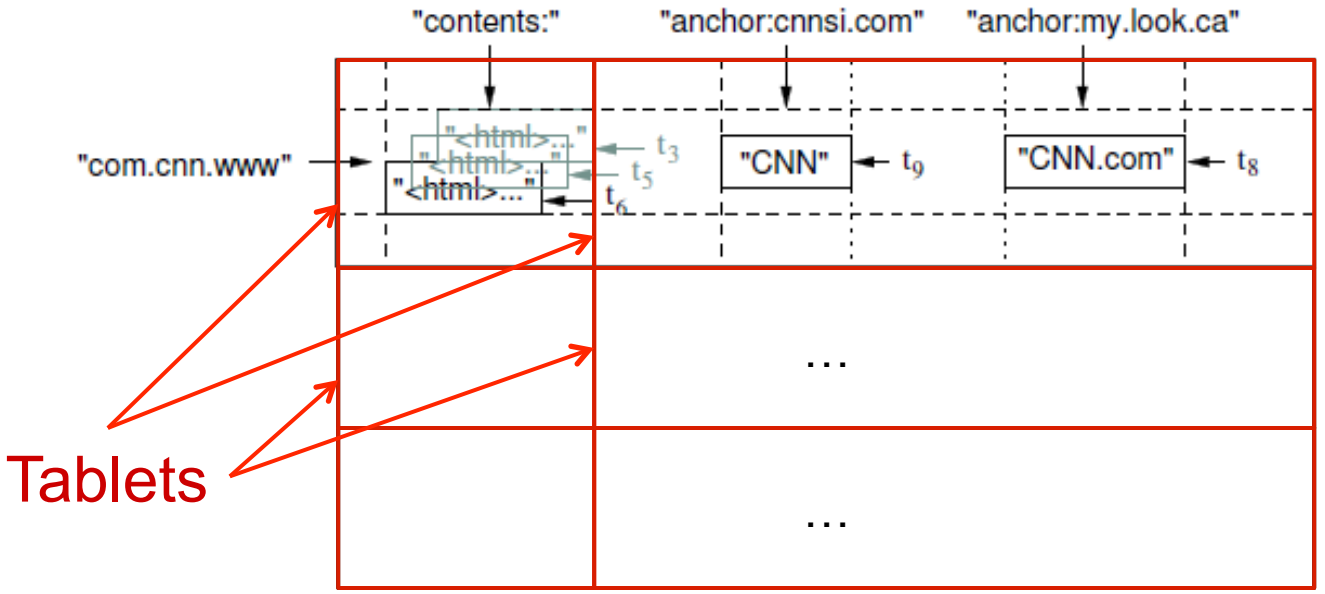
- Implements interfaces to
  - create and delete tables and column families,
  - modify cluster, table, and column family metadata such as access control rights,
  - Write or delete values in Bigtable,
  - Look up values from individual rows,
  - Iterate over a subset of the data in a table,
  - Atomic R-M-W sequences on data stored in a single row key (no support for Xacts across multiple rows).

Building Blocks

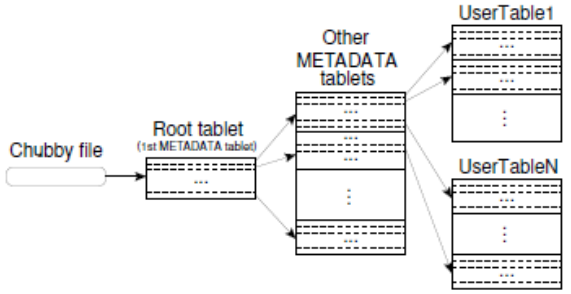
- Google File System:
  - for storing log and data files
  - high availability
- SSTable
  - a key/value database
- Chubby
  - name space

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



Locating Tablets (Ranges)



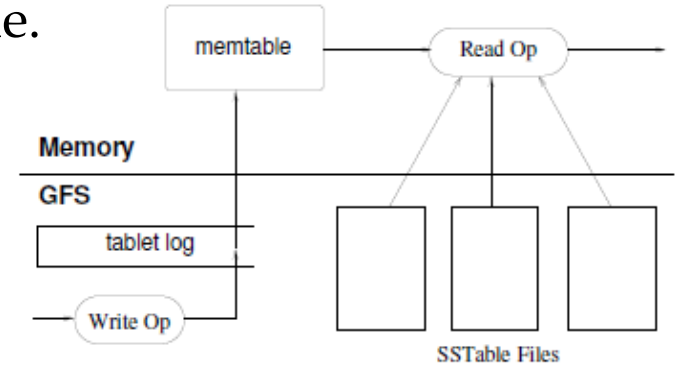
- A 3-level hierarchy:

- 1st Level: A file stored in chubby contains location of the root tablet, i.e., a directory of ranges (tablets) and associated meta-data.
  - The root tablet never splits.
- 2nd Level: Each meta-data tablet contains the location of a set of user tablets.
- 3rd Level: A set of SSTable identifiers for each tablet.

- Analysis:
  - Each meta-data row stores ~ 1KB of data,
  - With 128 MB tablets, the three level store addresses 2^{34} tablets (2^{61} bytes in 128 MB tablets).
    - Approaches a Zetabyte (million Petabytes).

Write & Read Operations

- Write operation arrives at a tablet server:
  - Server ensures the client has sufficient privileges for the write operation (Chubby),
  - A log record is generated to the commit log file,
  - Once the write commits, its contents are inserted into the memtable.
- Read operation arrives at a tablet server:
  - Server ensures client has sufficient privileges for the read operation (Chubby),
  - Read is performed on a merged view of (a) the SSTables that constitute the tablet, and (b) the memtable.



Write Operations

- As writes execute, size of memtable increases.
- Once memtable reaches a threshold:
  - Memtable is frozen,
  - A new memtable is created,
  - Frozen metable is converted to an SSTable and written to GFS.
- This *minor compaction* minimizes memory usage of tablet server, and reduces recovery time in the presence of crashes (checkpoints).
- *Merging compaction* (in the background) reads a few SSTables and memtable to produce one SSTable. (Input SSTables and memtable are discareded.)
- *Major compaction* rewrites all SSTables into exactly one SSTable (containing no deletion entries).

Questions