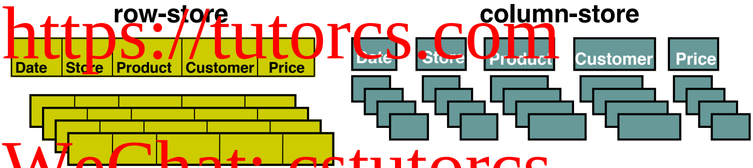
NoSQL Databases – Part 3

Column-oriented Data Stores

Column-oriented Data Stores[1]

- Inspired by Google's Bigtable.
- Store **data grouped by columns** (rather than rows) and may have a very large number of columns.



- Other column-oriented data stores
  - Hbase
  - Hypertable

[1] Figure source: S. Harizopoulos, D. Abadi and P. Boncz, Column-Oriented database systems, VLDB 2009

Google's Bigtable - Problem Analysis

- Used by over 60 projects at Google as of 2006, including Web indexing, Google Earth, Google Finance, Orkut, Google Docs, etc.
  - **Data types vary** from URLs to web pages to satellite imagery.
  - **Latency requirements vary** from backend bulk processing to real-time data processing.
  - **Infrastructures vary** from a handful to thousands of servers.

- **Need to scale to a very large size** such as petabytes of data across thousands of commodity servers.

- Most applications **require only single-row transactions**.

Google’s Bigtable - Problem Analysis

- Key questions:

1. How to represent data? (**expressiveness**)

   Key-value pairs are useful but limited

2. How to store data? (**scalability**)

   Data needs to be distributed across multiple servers

3. How to process data? (**efficiency**)

   Join on distributed tables needs to be avoided

Google's Bigtable - Problem Analysis

- Key questions:
  1. How to represent data? (**expressiveness**)

     Key-value pairs are useful but limited
  2. How to store data? (**scalability**)

     Data needs to be distributed across multiple servers
  3. How to process data? (**efficiency**)

     Join on distributed tables needs to be avoided

- Solution:

One big table

in which **both rows and columns can be split over multiple servers**, according to **their relatedness**.

Google’s Bigtable - Data Structure

- A (big) table is a **multi-dimensional sparse sorted map**.

$$(\underset{\text{string}}{\textbf{row key}},\ \underset{\text{string}}{\textbf{column key}},\ \underset{\text{int64}}{\textbf{timestamp}}) \rightarrow \underset{\text{string}}{\textbf{value}}$$

- The map is **indexed by** a row key, a column key, and a timestamp.

- Each value in the map is **an uninterpreted array of bytes**.

Google’s Bigtable - Data Structure

- A (big) table is a **multi-dimensional sparse sorted map**

  (**row key**, **column key**, **timestamp**) ↦ **value**

- **Example:** a (big) table that stores Web pages

Row Key	Column	Column	Column	…
	CONTENTS:	ANCHOR:CNNSI.COM	ANCHOR:MY.LOOK.CA	…
com.cnn.www	⟨html⟩⟨body⟩Home. . . ← t_1 404 Page not found ← t_2 ⟨html⟩⟨body⟩Inter. . . ← t_3	CNN ← t_9	CNN.com ← t_8	…
com.cnn.weather	…	…	…	…
com.cnn.live	…	…	…	…
…	…	…	…	…

- (“com.cnn.www”, “CONTENTS:”, t_1) ↦ “⟨html⟩⟨body⟩Home. . . ”
- (“com.cnn.www”, “ANCHOR:MY.LOOK.CA”, t_8) ↦ “CNN.com”

Google’s Bigtable - Data Structure (Row Key)

- Row keys are strings of up to 64KB size.
- Row keys are sorted in a lexicographical order.

ROW KEY	...
com.cnn.www	. . .
com.cnn.weather	. . .
com.cnn.live	. . .
. . .	. . .

- Every read or write of data under a single row key is atomic (regardless of the number of different columns being read or written in the row).

Google's Bigtable - Data Structure (Row Key)

- A table is **dynamically partitioned into tablets** (each approximately 100-200 MB in size by default). A tablet can be regarded as a horizontal partition in a table.
- Tablets are **the basic units of distribution and load balancing**, served by **tablet servers**.

	Row Key	...
tablet_1	com.cnn.www	. . .
	com.cnn.weather	. . .
	com.cnn.live	. . .
	. . .	. . .
tablet_2	nz.ac.otago.www	. . .
	nz.ac.otago.cs	. . .
	. . .	. . .
. . .	. . .	. . .

Google’s Bigtable - Data Structure (Row Key)

- **Question:** *Why are the reversed URLs used as row keys?*

	ROW KEY	...
tablet$_1$	com.cnn.www	. . .
	com.cnn.weather	. . .
	com.cnn.live	. . .
	. . .	. . .
tablet$_2$	nz.ac.otago.www	. . .
	nz.ac.otago.cs	. . .
	. . .	. . .
. . .	. . .	. . .

Google’s Bigtable - Data Structure (Row Key)

- Applications need to **wisely choose row keys**
  - The ordering of row-keys affects partitioning of rows into tablets.
  - Row ranges with smaller lexicographical distance are split into fewer tablets (good for reads).

	ROW KEY	. . .
tablet_1	com.cnn.www	. . .
	com.cnn.weather	. . .
	com.cnn.live	. . .
	. . .	. . .
tablet_2	nz.ac.otago.www	. . .
	nz.ac.otago.cs	. . .
	. . .	. . .
. . .	. . .	. . .

- As a result, reads of short row ranges are efficient and typically require communication with only a small number of machines.

Google's Bigtable - Data Structure (Column)

- Columns are **grouped into column families**, i.e., a column family contains columns of related data. A column is named as **family:qualifier**, e.g.,

COLUMN FAMILY 1	COLUMN FAMILY 2		
CONTENTS:	ANCHOR:CNNSI.COM	ANCHOR:MY.LOOK.CA	...
...	...	...	...

- **Question:** *Why are columns grouped into column families?*

Google's Bigtable - Data Structure (Column)

- Some properties
  - Column families form **the basic unit of access control**, discerning privileges to read, modify, create column-families, etc.
  - They can be vertically partitioned into different files.
  - Column families need to be defined in the schema (before data can be stored) but **columns within a family can be dynamically changed**.
  - The number of column families should be small (in the hundreds at most).

COLUMN FAMILY 1	COLUMN FAMILY 2		
CONTENTS:	ANCHOR:CNNSI.COM	ANCHOR:MY.LOOK.CA	...
...	...	...	...

Google’s Bigtable - Data Structure (Timestamp)

- Each cell can contain **multiple versions** of the same data, indexed by timestamp.

⟨html⟩⟨body⟩Home... ← t_1
404 Page not found ← t_2
⟨html⟩⟨body⟩Inter... ← t_3

- Each cell version is **a string**, i.e., a scalar value.
- Stored in decreasing timestamp order, and thus the most recent version can be **read first**.

Google’s Bigtable - Read Operations

```
Scanner scanner(T);
ScanStream *stream;
stream = scanner.FetchColumnFamily("anchor");
stream->SetReturnAllVersions();
scanner.Lookup("com.cnn.www");
for (; !stream->Done(); stream->Next()) {
  printf("%s %s %lld %s\n",
      scanner->RowName(),
      stream->ColumnName(),
      stream->MicroTimestamp(),
      stream->Value());}
```

Row Key	Column	Column	Column	...
	contents:	anchor:cnnsi.com	anchor:my.look.ca	...
com.cnn.www	⟨html⟩⟨body⟩Home. . . ← t_1 404 Page not found ← t_2 ⟨html⟩⟨body⟩Inter. . . ← t_3	CNN ← t_9	CNN.com ← t_8	...
com.cnn.weather	...	...	...	...
com.cnn.live	...	...	...	...
...	...	...	...	...

Google's Bigtable - Write Operations

```
# 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:my.look.ca");
Operation op;
Apply(&op, &r1);
```

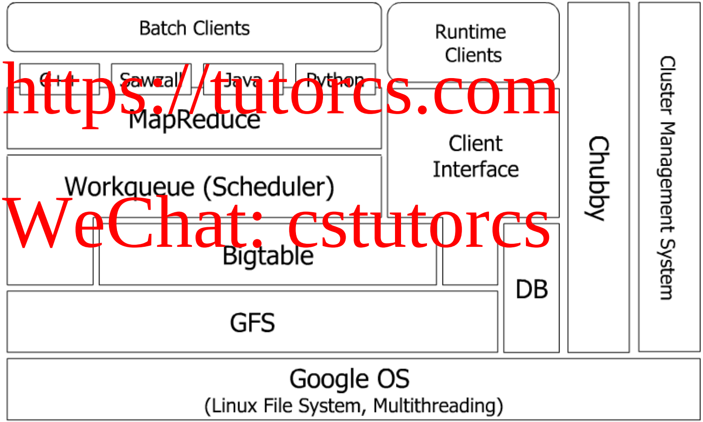
Row Key	Column	Column	Column	...
	contents:	anchor:cnnsi.com	anchor:my.look.ca	...
com.cnn.www	⟨html⟩⟨body⟩Home... ← t_1 404 Page not found ← t_2 ⟨html⟩⟨body⟩Inter... ← t_3	CNN ← t_9	CNN.com ← t_8	...
com.cnn.weather	...	...	...	...
com.cnn.live	...	...	...	...
...	...	...	...	...

Google's Bigtable - Infrastructure Dependencies

- Bigtable is built upon these components:
  - **Google file system** (GFS): a highly scalable distributed file system
    - e.g., store table data and log.
  - **Chubby lock service**: a highly-available and persistent distributed lock service
    - e.g., handles master-election, manage matadata, etc.
  - **MapReduce programming model**: a parallel computing model
    - Googles batch processing tool of choice
  - **Cluster scheduling system**: a cluster management system
    - e.g., handles failover, monitoring, etc.
  - . . .
- Similar components are being made available as Open Source by the Apache project Hadoop.

Google’s Overall Architecture

- Use **shared-nothing architecture**, consisting of thousands of commodity machines.



Batch Clients

Runtime Clients

C++

Sawzall

Java

Python

MapReduce

Client Interface

Workqueue (Scheduler)

Bigtable

DB

GFS

Chubby

Cluster Management System

Google OS

(Linux File System, Multithreading)

Google’s Bigtable - Summary

- Uses a **shared-nothing architecture** to provide scalability over massive data sets:
  - **Horizontal partitioning** by range of row keys.
  - **Vertical partitioning** by column families
- **Replication**: eventual-consistency replication across datacenters, between multiple BigTable serving setups (master/slave & multi-master)
- Supports **single-row transactions**.
- Supports **only simple queries**.
- Does **not support secondary indices**.