Today's topics

• FAQs
• NoSQL Storage
• BigTable

FAQs

• TP1
  - Extended Deadline: March 28th (Monday) → March 29th (Tuesday)

• Help session for PA3
  - April 1st 11:00AM – noon CSB130
  - Please check the assignment page

NoSQL databases

• Basic Idea
  - Operates without a schema
  - Allows users to add fields without having to define any changes in structure first
  - Useful when dealing with nonuniform data and custom fields

• Stands for “Not Only SQL”

• Handles data access with size and performance that demand a cluster

• Improves the productivity of application development by using a more convenient data interaction style

Polyglot persistence

• Using different data stores in different circumstances
  - Without picking a particular database for all situations

• Most organizations have a mix of data storage technologies for different circumstances
Key-Value Store

- Simple hash table
  - All access to the storage is via primary key
    - Get the value for the key
    - Put a value for a key
    - Delete a key
    - Add a key

- “value” is stored as a blob
  - Without caring or knowing what’s inside
  - Application is responsible for understanding data

Suitable use cases

- Storing session information
- User profiles, preferences
- Shopping cart data

When Not to Use

- Relationships between data
- Multi-operation transactions
- Query by data
  - There is no way to inspect the value on the server side
  - Exceptions
    - Lucene, Solr, and Galileo

Document Storage Model

- Documents
  - Self-describing
  - Data structure
    - Maps, collections, tree, and scalar values
  - Stores documents in the value part of the key-value store

- MongoDB, CouchDB, OrientDB, RavenDB, etc.

- Users can query the data inside the document
  - without having to retrieve the whole document
Suitable Use Cases
- Event logging
- Content management system, blogging platforms
- Web analytics or real-time analytics

When Not to Use
- Complex transactions spanning different operations
- Queries against varying aggregate structure

Column-Family Stores
- Cassandra, Hbase, Hypertable, and Amazon SimpleDB
- Stores data in column family as rows
  - Have many columns associated with a row key
- Column families
  - Groups of related data that is often accessed together

This material is built based on,
Column-family storage

- Optimized for the data
  - Sparse columns and no schema
- Aggregate-oriented storage
  - Most data interaction is done with the same aggregate
  - Aggregate
    - A collection of data that we interact with as a unit
- Stores groups of columns (column families) together

Storing data in a column-family store

- The stores organize their columns into column families
- Each column may be part of a single column family
- The column acts as unit for access
  - The assumption is that data for a particular column family will be usually accessed together

Scalability and latency

- Scale in capacity
  - E.g., webtable
    - 100,000,000,000 pages * 10 versions per page * 20KB/version
    - 20PB of data (200 million gigabytes)
  - E.g., google maps
    - 10PB of satellite image data
- Scale in throughput
  - Hundreds of millions of users
  - Tens of thousands to millions of queries per second
- Low latency
  - A few dozen milliseconds of total budget inside Google
  - Probably have to involve several dozen internal services per request
  - Few milliseconds for lookup

BigTable

- Google’s first answer to the question
  - “How do you store semi-structured data at scale?”

BigTable is used by,

- Web indexing
- Google Reader
- Google Maps
- Google Book Search
- Google Earth
- Blogger.com
- Google Code
- YouTube
- Gmail
- …
BigTable [1/2]

- Provides a simple data model
- Dynamic control over the data layout and format
- Allows clients to reason about the locality properties of the data represented in the underlying storage
- Data is indexed using row and column names that can be arbitrary strings
- Data in BigTable
  - Uninterpreted strings
  - Clients often serialize various forms of structured and semi-structured data into these strings

BigTable [2/2]

- Clients can control locality of their data
- Clients can control whether to serve data out of memory or from disk

Topics in BigTable

1. Data model
2. Locating tablet
3. Data Compaction
4. Data Compression
5. Caching and prefetching

Data Model

- A BigTable is a sparse, distributed, persistent multi-dimensional sorted map
- The map is indexed by,
  - A row key
  - A column key
  - A timestamp
- Each value in the map is an uninterpreted array of bytes

\[
(row: string, column: string, time: int64) \rightarrow string
\]
Rows
- Row keys
  - Arbitrary strings
  - Every read or write of data under a single row key is atomic
- BigTable maintains data in lexicographic order by row key
- Row range for a table
  - Dynamically partitioned

Tablets
- Large tables are broken into **tablets** at row boundaries
  - A tablet holds a 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
    - Allows a 100 machines to each pick up 1 tablet from the failed machine
  - Fine-grained load balancing
    - Migrate tablets away from the overloaded machine
    - Master makes load-balancing decisions

Tablets
- Read of short row ranges are efficient
  - Require communication with only a small number of machines
  - Clients get good locality for their data access
- maps.google.com/index.html is stored using the key `com.google.maps/index.html`
  - Storing pages under the same domain near each other makes some host and domain analysis more efficient

Column Families
- Column keys are grouped into sets called column families
  - Basic unit of access control
  - All data stored in a column family is usually of the same type
    - BigTable compresses data in the same column family together
  - A column family must be created before data can be stored under any column key in that family
    - After a family has been created, any column key within the family can be used

Column Families
- Column key
  - `family:qualifier`
  - Family name must be printable
  - Qualifier may be an arbitrary string
  - Access control and disk/memory accounting
    - Performed at the column family level

**Webtable with multiple column-families**

- **row keys**
  - `com aaa`
  - `com cnn www`
  - `com cnn www/TECH`
  - `com weather`
- **column family**
  - `language`
  - `contents`
  - `anchor.comsi.com`
  - `anchor.mylook.ca`
  - `com aaa` EN `DOCTYPE html PUBLIC ~`
  - `com cnn www` EN `DOCTYPE html PUBLIC ~`
  - `com cnn www/TECH` EN `DOCTYPE html PUBLIC ~`
  - `com weather` EN `DOCTYPE html PUBLIC ~`
Timestamps

- Each cell in Bigtable can contain multiple versions of the same data
  - Indexed by timestamp

- BigTable timestamp
  - 64-bit integers
  - Assigned by BigTable
  - Real time in microseconds
  - Explicitly assigned by client application

- Application should generate unique timestamp to avoid collisions
- Different versions of a cell are stored in decreasing timestamp order
  - The most recent versions can be read first

Garbage collection

- Two per-column-family settings
  - Tell Bigtable to garbage-collect cell versions automatically
  - The last n versions are kept
  - i.e. only recent versions are kept

API

- Functions for creating and deleting tables and column families
- Changing cluster, table, and column-family metadata (access control rights)

```c++
// 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:www.abc.com");
Operation op;
Apply(&op, &r1);
```