CS435 Introduction to Big Data - Spring 2017 Colorado State University

PART 3.
DATA STORAGE AND FLOW MANAGEMENT

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

Today's topics
- FAQs
- NoSQL Storage
- BigTable: Column-based storage system

FAQs
- Corrected midterm scores
  - Average: >82
- PA3 help session
  - April 7th, 3:00PM – 3:50PM CSB130

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

Data Storage and Flow Management
NoSQL Storage
Data Storage and Flow Management

NoSQL Storage

1. 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
    - 100TB 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

Example of data model with Webtable
- Webtable
  - A large collection of web pages and related information
  - URLs
  - Contents
  - Information

```
(row:string, column:string, time:int64) → 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
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
  - 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

System Structure

BigTable master
- Performs metadata ops + load balancing

BigTable server
- Serves data

BigTable client
- Library

GFS
- Holds tablet data, logs

Lock service
- Holds metadata, handles master-collection

Building blocks

- The Google SSTable (Sorted String Table) file format
  - Internally used to store BigTable data
  - Persistently ordered immutable map from key to values
  - Keys and values are arbitrary byte strings
- SSTable contains a sequence of blocks
  - 64KB, configurable
- Block index
  - Stored at the end of SSTable
  - Index is loaded into memory when the SSTable is opened

API

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

```java
// 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);
```
SSTable: Sorted String Table

Reading and writing data can dominate running time
Random reads and writes are critical features

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Key</th>
<th>Value</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
</table>

Index:

- Key
- Offset

Access to the block

- Lookup can be performed with a single disk seek
- Find the block by performing a binary search of the in-memory index
- Read the block from disk

Locating tablets

- Since tablets move around from server to server, given a row, how do clients find the right machine?
  - Need to find tablet whose row range covers the target row
- Using the BigTable master
  - Central server almost certainly would be bottleneck in large system
  - Instead: store special tables containing tablet location info in BigTable cell itself

Root tablet is never split

- To ensure that the tablet location hierarchy has no more than 3 levels

Metadata tablet

- Stores the location of a tablet under a row key
  - Tablet’s identifier and its end row
  - Each metadata row stores approximately 1KB of data in memory
  - Average limit of 128MB Metadata tablets
  - 2nd tablets are addressed

Caching the tablet locations

- Client library caches tablet locations
- Traverses up the tablet location hierarchy
  - If the client does not know the location of a tablet
  - If it discovers that the cached location information is incorrect
Caching the tablet locations

- If the client’s cache is empty?
  - One read from Chubby
  - One read from root tablet
  - One read from metadata tablet
  - Three network round-trips is required to locate the tablet

- If the client’s cache is stale?
  - With given information, client could not find the data
  - What is the maximum round-trips needed (if the root server has not changed)?

Prefetching tablet locations

- Client library reads the metadata for more than one tablet
  - Whenever it reads the metadata table
- No GFS accesses are required
  - Table locations are stored in memory

Tablet Assignment (1/2)

- Each tablet is assigned to one tablet server at a time
  - The master keeps track of:
    - The set of live tablet servers
    - Which tablets are assigned

- New tablet assignment
  - The master assigns the tablet by sending a tablet load request to the tablet server

Tablet Assignment (2/2)

- A tablet server starts
  - Chubby creates a uniquely-named file in a specific Chubby directory
  - Exclusive lock
  - Master monitors this directory to discover tablet servers

- A tablet server terminates
  - Release its lock
  - Master will reassign its tablets more quickly

Table serving

- The persistent state of a tablet is stored in GFS
Tablet Representation

SSTable: immutable on-disk ordered map from string → string
- String keys <row, column, timestamp>

Write buffer in memory (random-access) → MemTable

Append-only log on GFS → SSTable on GFS

Tablet server

write operation
- The tablet server checks,
  - If the data is well-formed
  - If the user is authorized to mutate data
- Operation is committed to a log file
- The contents are inserted into the MemTable

read operation
- Tablet server checks
  - If the request is well-formed
  - If the user is authorized to read data
- Merged view of MemTable (in memory) and SSTable (in disk)
- Read operation is performed