Cairo University

Faculty of Engineering

Computer Engineering Dept.

Project

Google BigTable Concepts

1. Introduction:

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance.

2. Data Model:

A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

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

As one concrete example, suppose we want to keep a copy of a large collection of web pages and related information that could be used by many different projects; let us call this particular table the Webtable. In Webtable, we would use URLs as row keys, various aspects of web pages as column names. For example: store the contents of the web pages in the contents column under the timestamps when they were fetched, as illustrated in Figure1.

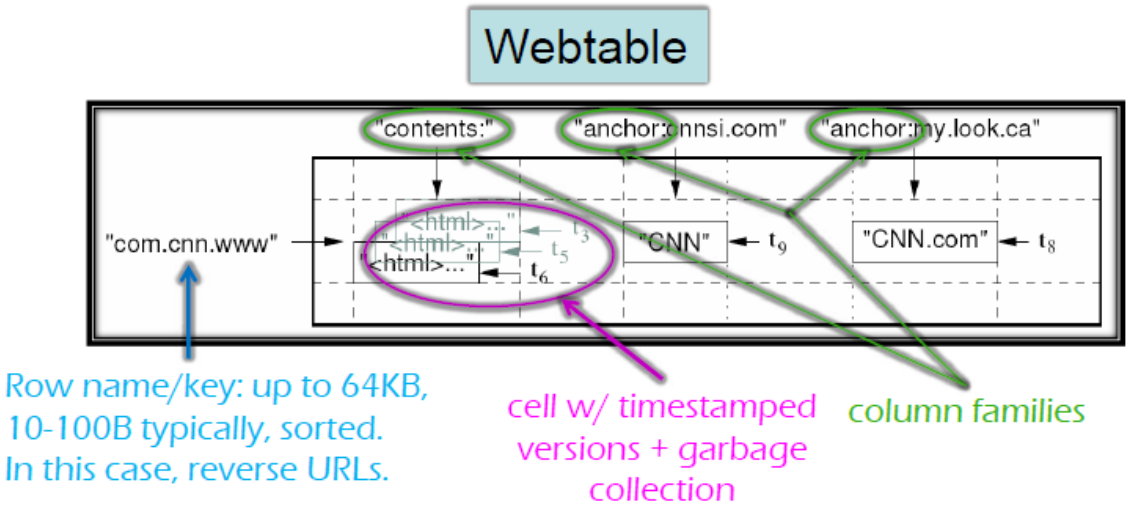

Figure 1: A slice of an example table that stores Web pages. The row name is a reversed URL. The contents column family contains the page contents, and the anchor column family contains the text of any anchors that reference the page. CNN's home page is referenced by both the Sports Illustrated and the MY-look home pages, so the row contains columns named anchor:cnnsi.com and anchor:my.look.ca. Each anchor cell has one version; the contents column has three versions, at timestamps t_3, t_5, and t_6.

2.1. Rows:

The row keys in a table are arbitrary strings (currently up to 64KB in size, although 10-100 bytes is a typical size for most of our users). 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), a design decision that makes it easier for clients to reason about the system's behavior in the presence of concurrent updates to the same row. Bigtable maintains data in lexicographic order by row key. The row range for a table is dynamically partitioned.

Each row range is called a tablet, which is the unit of distribution and load balancing. As a result, reads of short row ranges are efficient and typically require communication with only a small number of machines. Clients can exploit this property by selecting their row keys so that they get good locality for their data accesses.

2.2. Columns Families:

Column keys are grouped into sets called column families, which form the basic unit of access control. A column key is named using the following syntax: *family:qualifier*. Column family names must be present while qualifiers are optional. An example column family for the Webtable is language, which stores the language in which a web page was written. We use only one column key in the language family, and it stores each web page's language ID. Another useful column family for this table is anchor; each column key in this family represents a single anchor, as shown in Figure1. The qualifier is the name of the referring site; the cell contents is the link text.

2.3. Timestamps:

Each cell in a Bigtable can contain multiple versions of the same data; these versions are indexed by timestamp.

3. API:

The Bigtable API provides functions for creating and deleting tables and column families. Client applications can write or delete values in Bigtable, look up values from individual rows, or iterate over a subset of the data in a table. Figure 2 shows C++ code that uses a *RowMutation* abstraction to perform a series of updates. (Irrelevant details were elided to keep the example short.) The call to Apply performs an atomic mutation to the Webtable: it adds one anchor to row: *www.cnn.com* and deletes a different anchor.

```
// 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);
```

Figure 2: Writing to Bigtable.

4. Building Blocks

Bigtable is built on several other pieces of Google infrastructure. Bigtable uses the distributed Google File System (GFS) to store log and data files. A Bigtable cluster typically operates in a shared pool of machines that run a wide variety of other distributed applications, and Bigtable processes often share the same machines with processes from other applications. Bigtable depends on a cluster management system for scheduling jobs, managing resources on shared machines, dealing with machine failures, and monitoring machine status.

5. Implementation:

The Bigtable implementation has three major components:

- A library that is linked into every client,
- One master server,
- Many tablet servers.

A Bigtable cluster stores a number of tables. Each Bigtable table is partitioned into many tablets based on row keys. Tablets are stored in a particular structure in GFS. Tablet servers can be dynamically added (or removed) from a cluster to accommodate changes in workloads. The master is responsible for assigning tablets to tablet servers, balancing tablet-server load..etc. Each tablet server manages a set of tablets. The tablet server handles read and write requests to the tablets that it has loaded. As with many single-master distributed storage systems, client data does not move through the master: clients communicate directly with tablet servers for reads and writes. Because Bigtable clients do not rely on the master for tablet location information, most clients never communicate with the master. As a result, the master is lightly loaded in practice.

5.1.Tablet Location:

The location of all tablets is stored in a special METADATA table.
The client library caches tablet locations.

5.2. Tablet Assignment:

Each tablet is assigned to one tablet server at a time. The master keeps track of the set of live tablet servers, and the current assignment of tablets to tablet servers. When a tablet is unassigned, and a tablet server with sufficient room for the tablet is available, the master assigns the tablet by sending a tablet load request to the tablet server. When a tablet server starts, it creates, and acquires an exclusive lock. The master is responsible for detecting when a tablet server is no longer serving its tablets, and for reassigning those tablets as soon as possible. To detect when a tablet server is no longer serving its tablets, the master periodically asks each tablet server for the status of its lock.

5.3. Tablet Serving:

The persistent state of a tablet is stored in GFS. Updates are committed to a commit log. Of these updates, the recently committed ones are stored in memory.

Reference

Chang, Fay, et al. "Bigtable: A distributed storage system for structured data." *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008): 4.

Requirement

You are required to simulate such distributed system. Choose any suitable application to apply the BigTable concepts. (e.g. A simplified schema for Twitter, A collection of Web pages, …etc)

System Architecture:

1-Global File System having original data tables, for simplicity we will assume that these files are stored on the master machine.

2-One Master: Machine1

- Responsible for dividing data tables into tablets.
  - May check locality requirements for better efficiency. (For example: a set of data is most probable to be accessed together so put together in a tablet to assign to a single tablet server).
- Responsible for assigning tablets to tablet servers
  - Ensuring load balance.
  - May check locality requirements for better efficiency. (More than one tablet may be related).
- Has Metadata table indicating the row key range (start key-end key) for each tablet server. (Note that data is sorted alphabetically on row key)

2-Two Clients: (Machine2, Machine3)

Each client caches Metadata table (tablet locations).

Requesting queries on data, these queries include:

- Set(): write cells in a row

    - Input: row key, [column family: (qualifier) and cell data]→ can be repeated n times in one query.
- DeleteCells(): delete certain cells in a row
    - Input: row key, [column family: (qualifier)]→ can be repeated n times in one query.
- DeleteRow(): delete a row
    - Input: [row key]→ can be repeated n times in one query.
- AddRow(): add a new row
    - Input: row key, [column family: (qualifier) and cell data]→ can be repeated n times in one query.
- ReadRows(): read a whole row
    - Input: [row key]→ can be repeated n times in one query.

3-Two Tablet Servers: (Machine4, Machine5)

Each tablet server is responsible for a group of data tablets (The exact number of tablets in each tablet server is according to the data size).

General Flow:

- At the beginning the master should divide the table into tablets and assign to tablet servers updating the Metadata table.
- When a client requests a certain query using the row key, the client sends its request to the designated tablet server as indicated in its cached Metadata table.
- If the query involves any write operations (set or delete), the tablet has to be locked (No other client can access this tablet) till these operations finish.
- Tablet server performs the operations requested by the client.
- Periodically each tablet server writes its tablets to the original data tables. (It is smart to only write updated data).
- If a client requested a locked tablet, it gets blocked. You can choose any reasonable unblocking mechanism.
- If a client requested a query that adds/deletes row(s) that will affect the tablet server row key range, the master gets notified by the tablet server. The master should update the Metadata table accordingly and check the load balance to determine if re-assignment of tablets should be performed.
- If the Metadata table is changed, then cached versions at the clients should be updated as well.

Notes:

- You will simulate real distribution i.e. each of the machines indicated in the system architecture should be a separate individual machine.
    - The communication mechanism between different machines is your choice.
- Due to the above requirements, each team should consist of 5 members.
- You don’t have to use a Database (can use any data structure).
- Your application should include only one BigTable table.

- The data should be divided to at least 3 tablets.
- You can use any programming language.
- The output to your program is a log (textual or graphical) showing all the steps performed to execute a given query →More efficient data allocation will lead to less overhead.

Deliverables:

- A proposal for the application chosen.
    1. When thinking about the idea of the proposal, at least the following points should be taken into consideration:
        - Description of the application chosen.
        - Why did you choose this application?
        - How will you get the data?
        - The design of the BigTable.
    2. Fill this link with your team names and emails in the sheet named "Teams" then fill the sheet named "Proposals" with the application chosen and how you will get the data.
    3. Responses will be received through the column named: "Response: Approve/Modify".
    4. At most two teams can choose the same application.
    5. Soft Deadline: 17 May 2021 at 11:59 p.m. (the earlier the better since you can start working in the project).
- Final Submission:
    1. All the code files you used for implementation.
    2. The log files for the output of your program based on test cases that you will prepare to test all the functionalities. (These testcases are subject to modification during the final discussion).
    3. Each team should submit one folder containing 1 and 2 via blackboard.
    4. Deadline: 7 June 2021 at 2:00 p.m.