Google BigTable

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. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. This describes the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and also describe the design and implementation of Bigtable.

Over the last two and a half years Google designed, implemented, and deployed a distributed storage system for managing structured data called Bigtable. Bigtable is designed to reliably scale to petabytes of data and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search and Google Earth. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users. The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data.

A Bigtable is really a sparse, distributed, persistent multidimensional sorted map. The map is listed in a row key, column key, along with a timestamp each value within the map is definitely an uninterpreted variety of bytes.

Google BigTable And NoSql

NoSQL is an umbrella term for all the databases that are different from 'the standard' SQL databases, such as MySQL, Microsoft SQL Server and PostgreSQL. These 'standard' SQL databases are all relational databases, feature the SQL query language and adhere to the ACID properties. These properties basically boil down to consistency. A NoSQL database is different because it doesn't support one or more of these key features of the so-called 'SQL databases':

* Consistency
* Relational data
* SQL language

Most of these features go hand in hand.

Consistency

Consistency is where most NoSQL databases differ from SQL databases. You can pull the plug from a SQL database and it will make sure your data is still consistent and uncorrupted. NoSQL databases tend to sacrifice this consistency for better scalability. Google's Bigtable also does this.

Relational data

SQL databases revolve around normalized, relational data. The database ensures that these relations stay valid and consistent, no matter what you throw at it.

NoSQL databases usually don't support relations, because they don't support the consistency to enforce these relations. Also, relational data is bad for performance when the data is distributed across several servers.

SQL language

The SQL language was designed especially for relational databases, the so-called 'SQL databases'. Since most NoSQL databases are very different from relational databases, they don't have the need for SQL. Also, some NoSQL databases have features that simply cannot be expressed in SQL, thus requiring a different query language.

The internal mechanics of Bigtable versus, say, MySQL are so dissimilar as to make comparison difficult, and the intended goals don't overlap much either. But you can think of Bigtable a bit like a single-table database. Imagine, for example, the difficulties you would run into if you tried to implement Google's entire web search system with a MySQL database -- Bigtable was built around solving those problems.

*Bigtable datasets can be queried from services like AppEngine using a language called GQL ("gee-kwal") which is a based on a subset of SQL.* Conspicuously missing from GQL is any sort of JOIN command. Because of the distributed nature of a Bigtable database, performing a join between two tables would be terribly inefficient. Instead, the programmer has to implement such logic in his application, or design his application so as to not need it.

NoSQL Emerged From a Need

*Data Storage:* The world's stored digital data is measured in exabytes. An exabyte is equal to one billion gigabytes (GB) of data.

According to Internet.com, the amount of stored data added in 2006 was 161 exabytes. Just 4 years later in 2010, the amount of data stored will be almost 1,000 ExaBytes which is an increase of over 500%. In other words, there is a lot of data being stored in the world and its just going to continue growing.

*Interconnected Data:* Data continues to become more connected. The creation of the web fostered in hyperlinks, blogs have pingbacks and every major social network system has tags that tie things together. Major systems are built to be interconnected.

*Complex Data Structure:* NoSQL can handle hierarchical nested data structures easily. To accomplish the same thing in SQL, you would need multiple relational tables with all kinds of keys. In addition, there is a relationship between performance and data complexity. Performance can degrade in a traditional RDBMS as we store the massive amounts of data required in social networking applications and the semantic web.

Googel BigTable Characteristics

BigTable is a distributed storage system that is structured as a large table: one that may be petabytes in size and distributed among tens of thousands of machines. It is designed for storing items such as billions of URLs, with many versions per page; over 100 TB of satellite image data; hundreds of millions of users; and performing thousands of queries a second. BigTable was developed at Google in has been in use since 2005 in dozens of Google services. An open source version, HBase, was created by the Apache project on top of the Hadoop core. Apache Cassandra, first developed at Facebook to power their search engine, is similar to BigTable with a tunable consistency model and no master (central server).

It is easy enough to picture a simple table. Let's look at a few characteristics of BigTable:

**Map**

A map is an associative array; a data structure that allows one to look up a value to a corresponding key quickly. BigTable is a collection of (key, value) pairs where the key identifies a row and the value is the set of columns.

**Persistant**

The data is stored peristantly on disk.

**Distributed**

BigTable's data is distributed among many independent machines. At Google, BigTable is built on top of GFS (Google File System). The Apache open source version of BigTable, HBase, is built on top of HDFS (Hadoop Distributed File System) or Amazon S3. The table is broken up among rows, with groups of adjacent rows managed by a server. *A row itself is never distributed.*

**Sparse**

The table is sparse, meaning that different rows in a table may use different columns, with many of the columns empty for a particular row.

**Sorted**

Most associative arrays are not sorted. A key is hashed to a position in a table. BigTable sorts its data by keys. This helps keep related data close together, usually on the same machine — assuming that one structures keys in such a way that sorting brings the data together. For example, if domain names are used as keys in a BigTable, it makes sense to store them in reverse order to ensure that related domains are close together. For example:

edu.rutgers.cs

edu.rutgers.nb

edu.rutgers.www

**Multidimensional**

A table is indexed by rows. Each row contains one or more named column families. Column families are defined when the table is first created. Within a column family, one may have one or more named columns. All data within a column family is usually of the same type. The implementation of BigTable usually compresses all the columns within a column family together. Columns within a column family can be created on the fly. Rows, column families and columns provide a three-level naming hierarchy in identifying data. For example:

edu.rutgers.cs" : { // row

"users" : { // column family

"watrous": "Donald", // column

"hedrick": "Charles", // column

"pxk" : "Paul" // column

}

"sysinfo" : { // another column family

"" : "SunOS 5.8" // column (null name)

}

}

To get data from BigTable, you need to provide a fully-qualified name in the form column-family:column. For example, users:pxk or sysinfo:. The latter shows an null column name.

**Time-based**

Time is another dimension in BigTable data. Every column family may keep multiple versions of column family data. If an application does not specify a timestamp, it will retrieve the latest version of the column family. Alternatively, it can specify a timestamp and get the latest version that is earlier than or equal to that timestamp.

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 that drove some of our design decisions, 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, and store the contents of the web pages in the contents: column under the timestamps when they were fetched, as illustrated in Figure 1

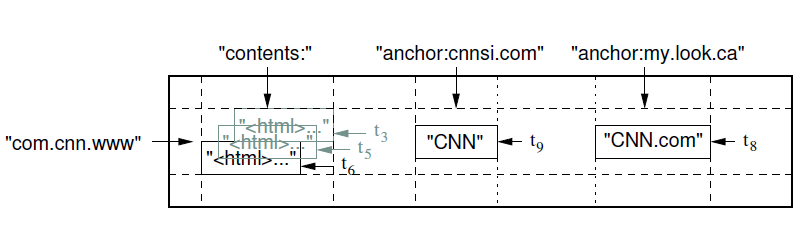
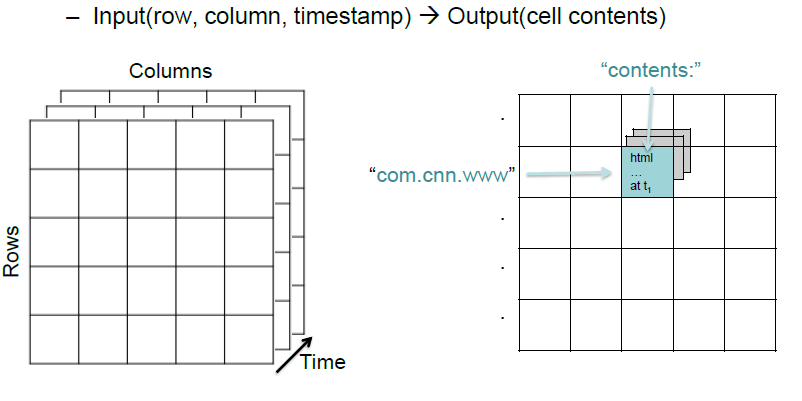


Figure 1

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 t3, t5, and t6.



***3 Dimensional Structure***

**Column Families**

Column keys are grouped into sets called column families, which form the basic unit of access control. All data stored in a column family is usually of the same type (we compress 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. It is our intent that the number of distinct column families in a table be small (in the hundreds at most), and that families rarely change during operation. In contrast, a table may have an unbounded number of columns.