# Bigtable – Review

Yixing Chen, yc3094

The paper *Bigtable: A Distributed Storage System for Structured Data* introduced 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. It helps to store data for many projects in Google, it has successfully provided a flexible, high-performance solution and gives clients dynamic control over data layout and format.

Bigtable is designed to reliably scale to petabytes of data and thousands of machines. It has achieved several goals: wide applicability, scalability, high performance, and high availability. It resembles a database, but it has many different features. For example, Bigtable provides clients with a simple data model rather than a full relational data model, treats data as uninterpreted strings, and Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

A Bigtable is a sparse, distributed, persistent multidimensional sorted map which is indexed by a row key, column key, and a timestamp. Row keys are arbitrary strings and help the data to be maintained in lexicographic order. And cells in Bigtable contain multiple version of the same data, so we need timestamp to index these data.

The Bigtable API provides many kinds of functions – creating, deleting, changing tables, clusters and column families. Bigtable uses GFS to store log and data files so it can operates in a shared pool of machines that run a wide variety of other distributed applications. Chubby is a highly-available and persistent distributed lock service, which helps Bigtable achieve many tasks. For example, it helps to ensure whether there is at most one active master, store the bootstrap location, schema information, access control lists, and discover tablet servers and finalize tablet server deaths.

The Bigtable implementation has three major components: a library that is linked into every client, one master server, and many tablet servers. Master should assign tablets to tablet servers, detect the addition and expiration of tablet servers, balance tablet-server load, and collect garbage. And tablet server should handle read and write requests to the tablet. Three level: Root tablet → METADATA tablets → User Table.

To achieve the high performance, availability and reliability, Bigtable implementation uses many kinds of refinements such as locality groups, compression, caching, bloom filters, commit-log implementation and so on to increase data access efficiency.

The performance evaluation shows that random and sequential writes perform better than random reads, and the performances of random writes and sequential writes are nearly of the same efficiency.

At the last of this paper, writers tell us many lessons which they learn from the process of designing and implementing Bigtable. First, large distributed systems are vulnerable to many types of failures, standard network partitions and fail-stop failures assumed in many distributed protocols. Second, delay adding new features until it is clear how the new features will be used. Third, proper system-level monitoring is very important. And the most important lesson is the value of simple designs.

In my opinion, it's a great paper to introduce the Bigtable to us. It illustrate many features about Bigtable: data model, API, building blocks, implementation, refinement, performance evaluation and so on. We not only learn the features of Bigtable, but also learn many mechanism, possible problem and many important lessons in designing a distributed storage system, which may provide valuable experience for our future studying and working.