**Bigtable vs. Pnuts**

1. **Bigtable**

**▪What were the key goals of the paper?**

This paper describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format and has successfully provided a flexible, high-performance solution for many Google products, and describe the design and implementation of Bigtable.

**▪A brief summary of main ideas.**

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. 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.

**▪Was this a good paper? Did it achieve what it set out to do?**

Yes, this is an excellent paper which presented a whole design of Bigtable. It mainly focus on two key point. First, it focus on the design of Bigtable. This paper demonstrate the detail of data model, design and implement of Bigtable. Second, it shows the efficiency on test and real products. This paper describe the efficiency of Bigtable on some test scenario and some deployment of Bigtable on some Google products, and get some useful advises from its and discuss some lessons that learned in designing and supporting Bigtable.

**▪What would you do differently?**

**▪What challenges you see ahead in the area.**

Bigtable does not currently support general transactions across row keys, although it provides an interface for batching writes across row keys at the clients.

**▪Key features**

1. **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

Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and 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. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

**1.1 Rows**

The row keys in a table are arbitrary strings. 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.

**1.2 Column Families**

Column keys are grouped into sets called column families, which form the basic unit of access control. 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. A column key is named using the following syntax: family:qualifier.

**1.3 Timestamps**

Each cell in a Bigtable can contain multiple versions of the same data; these versions are indexed by timestamp. Different versions of a cell are stored in decreasing timestamp order, so that the most recent versions can be read first. Support two per-column-family settings that tell Bigtable to garbage-collect cell versions automatically. The client can specify either that only the last n versions of a cell be kept, or that only new-enough versions be kept (e.g., only keep values that were written in the last seven days).

1. **API**

The Bigtable API provides functions for creating and deleting tables and column families. It also provides functions for changing cluster, table, and column family metadata, such as access control rights. Bigtable supports several other features that allow the user to manipulate data in more complex ways. First, Bigtable supports single-row transactions, which can be used to perform atomic read-modify-write sequences on data stored under a single row key. Second, Bigtable allows cells to be used as integer counters. Finally, Bigtable supports the execution of client-supplied scripts in the address spaces of the servers.

1. **Building Blocks**

Bigtable is built on several other pieces of Google infrastructure. Bigtable uses the distributed Google File System (GFS) [17] to store log and data files. Bigtable depends on a cluster management system for scheduling jobs, managing resources on shared machines, dealing with machine failures, and monitoring machine status. The Google SSTable file format is used internally to store Bigtable data. An SSTable provides a persistent, ordered immutable map from keys to values, where both keys and values are arbitrary byte strings.

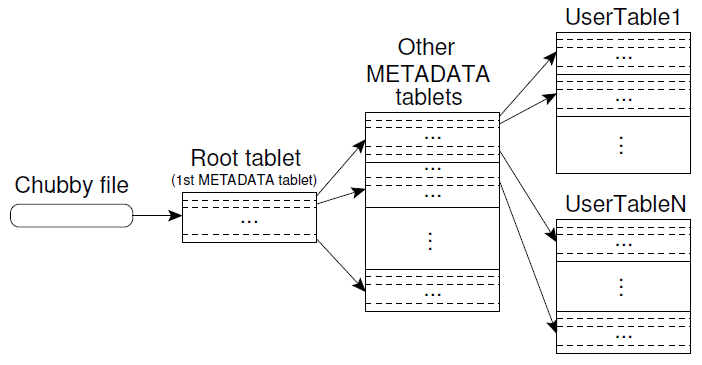
Bigtable relies on a highly-available and persistent distributed lock service called Chubby. Bigtable uses Chubby for a variety of tasks: to ensure that there is at most one active master at any time; to store the bootstrap location of Bigtable data; to discover tablet servers and finalize tablet server deaths; to store Bigtable schema information (the column family information for each table); and to store access control lists.

1. **Implementation**

The Bigtable implementation has three major components: a library that is linked into every client, one master server, and many tablet servers.

The master is responsible for assigning tablets to tablet servers, detecting the addition and expiration of tablet servers, balancing tablet-server load, and garbage collection of files in GFS. In addition, it handles schema changes such as table and column family creations. Each tablet server manages a set of tablets (typically we have somewhere between ten to a thousand tablets per tablet server). The tablet server handles read and write requests to the tablets that it has loaded, and also splits tablets that have grown too large. As with many single-master distributed storage systems [17, 21], 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. A Bigtable cluster stores a number of tables. Each table consists of a set of tablets, and each tablet contains all data associated with a row range. Initially, each table consists of just one tablet. As a table grows, it is automatically split into multiple tablets, each approximately 100-200 MB in size by default.

**4.1 Tablet Location**



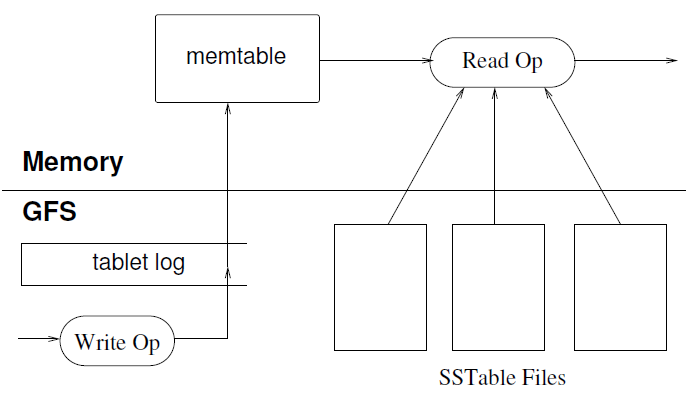
The first level is a file stored in Chubby that contains the location of the root tablet. The root tablet contains the location of all tablets in a special METADATA table. Each METADATA tablet contains the location of a set of user tablets. The root tablet is just the first tablet in the METADATA table, but is treated specially—it is never split–to ensure that the tablet location hierarchy has no more than three levels. The METADATA table stores the location of a tablet under a row key that is an encoding of the tablet's table identifier and its end row.

**4.2 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, including which tablets are unassigned. 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. Bigtable uses Chubby to keep track of tablet servers. When a tablet server starts, it creates, and acquires an exclusive lock on, a uniquely-named file in a specific Chubby directory. The master monitors this directory (the servers’ directory) to discover tablet servers.

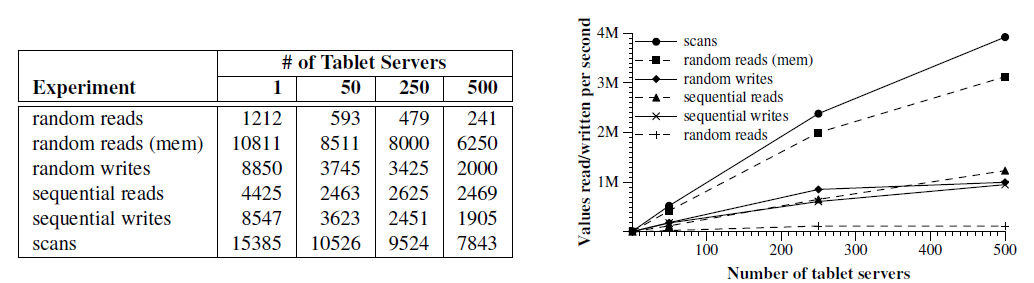
**4.3 Tablet Serving**

The persistent state of a tablet is stored in GFS, as illustrated in Figure 5. Updates are committed to a commit log that stores redo records. Of these updates, the recently committed ones are stored in memory in a sorted buffer called a *memtable*; the older updates are stored in a sequence of SSTables.



1. **Performance Evaluation**

We set up a Bigtable cluster with N tablet servers to measure the performance and scalability of Bigtable as N is varied. N client machines generated the Bigtable load used for these tests.



1. **Pnuts**

**▪What were the key goals of the paper?**

**▪A brief summary of main ideas.**

**▪Was this a good paper? Did it achieve what it set out to do?**

**▪What would you do differently?**

**▪What challenges you see ahead in the area.**