Virginia Tech
Invent the Future

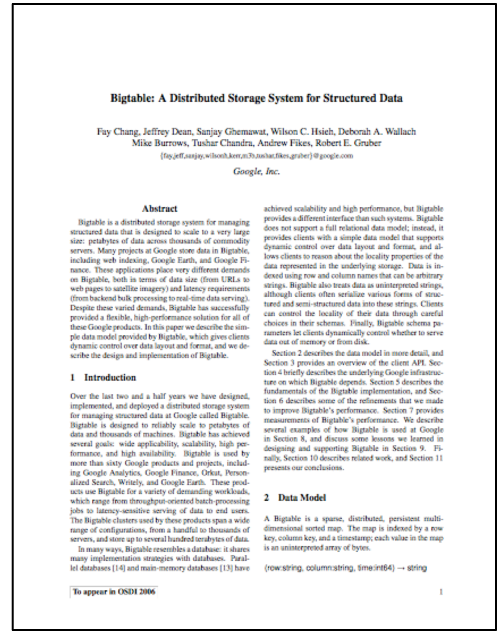
Google's BigTable

6 November 2012

Presenter: Jeffrey Kendall
jdk34@vt.edu

BigTable Introduction

- Development began in 2004 at Google (published 2006)
- A need to store/handle large amounts of (semi)-structured data



Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

Google, Inc.

Abstract

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. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

1 Introduction

Over the last two and a half years we have designed, implemented, and deployed a distributed storage system for managing structured data at Google 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, Writely, 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.

In many ways, Bigtable resembles a database: it shares many implementation strategies with databases. Parallel databases [14] and main-memory databases [13] have achieved scalability and high performance, but Bigtable provides a different interface than such systems. 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.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Section 4 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 5 describes the fundamentals of the Bigtable implementation, and Section 6 describes some of the refinements that we made to improve Bigtable's performance. Section 7 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 8, and discuss some lessons we learned in designing and supporting Bigtable in Section 9. Finally, Section 10 describes related work, and Section 11 presents our conclusions.

2 Data Model

A Bigtable is a sparse, distributed, persistent multi-dimensional 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

To appear in OSDI 2006 1

- Many Google projects store data in BigTable

Goals of BigTable

- Asynchronous processing across continuously evolving data
  - Petabytes in size
- High volume of concurrent reading/writing spanning many CPUs
- Need ability to conduct analysis across many subsets of data
  - Temporal analysis (e.g. how to anchors or content change over time?)
- Can work well with many clients, but not too specific to clients' needs

BigTable in a Nutshell

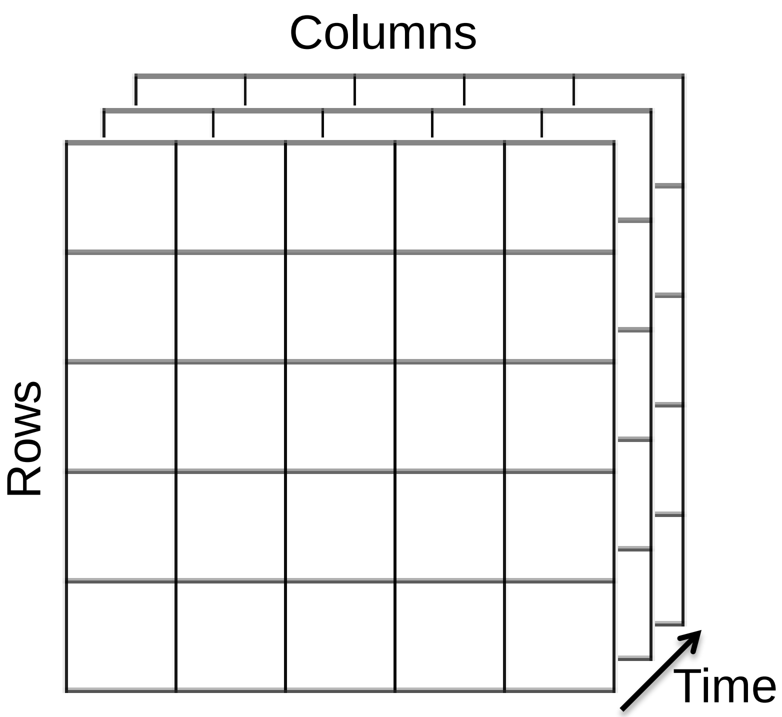
- Distributed multi-level map
- Fault-tolerant
- Scalable
  - Thousands of servers
  - Terabytes of memory-based data
  - Petabytes of disk-based data
  - Millions of reads/writes per second
- Self-managing
  - Dynamic server management

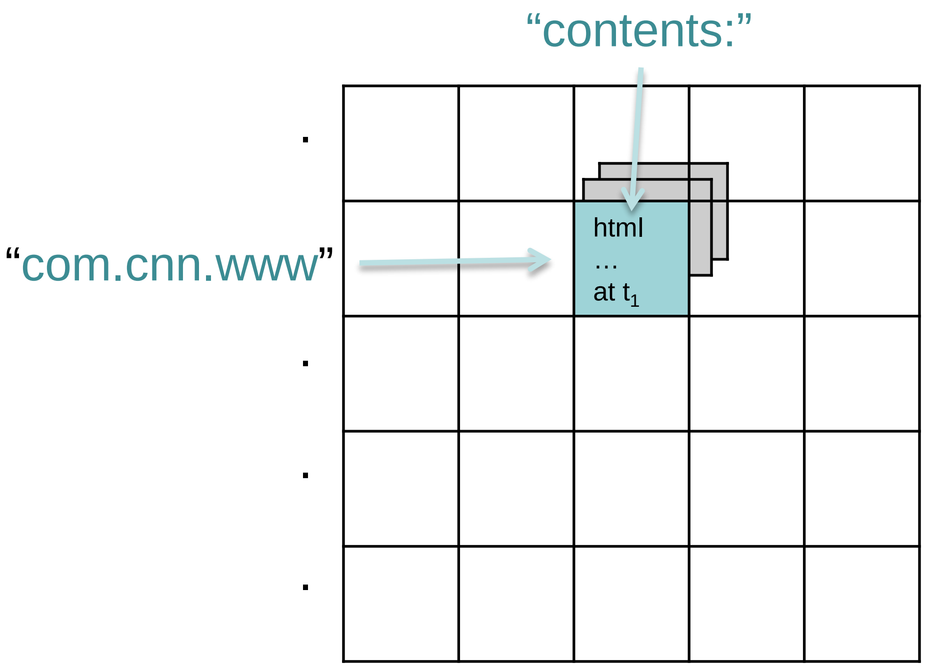
Building Blocks

- Google File System is used for BigTable's storage
- Scheduler assigns jobs across many CPUs and watches for failures
- Lock service distributed lock manager
- MapReduce is often used to read/write data to BigTable
  - BigTable can be an input or output

Data Model

- "Semi" Three Dimensional datacube
  - Input(row, column, timestamp) → Output(cell contents)





More on Rows and Columns

Rows

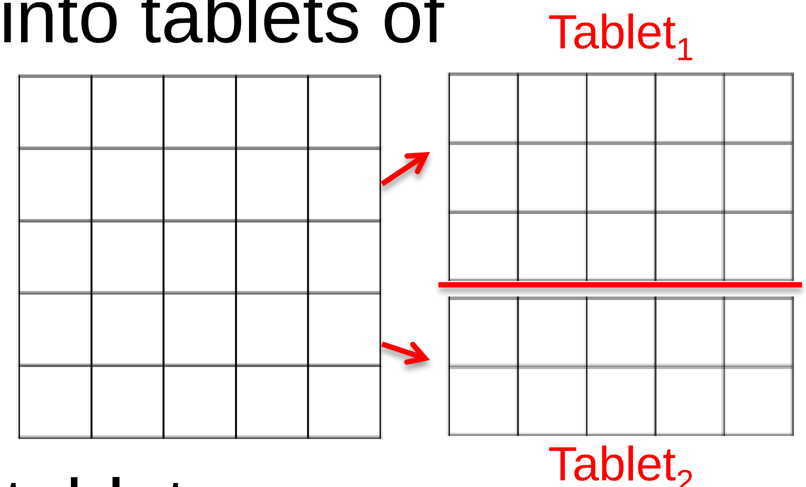
- Name is an arbitrary string
- Are created as needed when new data is written that has no preexisting row
- Ordered lexicographically so related data is stored on one or a small number of machines

Columns

- Columns have two-level name structure
  - family:optional_qualifier (e.g. anchor:cnnsi.com | anchor: stanford.edu)
- More flexibility in dimensions
  - Can be grouped by locality groups that are relevant to client

Tablets

- The entire BigTable is split into tablets of contiguous ranges of rows
  - Approximately 100MB to 200MB each



- One machine services 100 tablets
  - Fast recovery in event of tablet failure
  - Fine-grained load balancing
  - 100 tablets are assigned non-deterministically to avoid hot spots of data being located on one machine
- Tablets are split as their size grows

Implementation Structure

Client/API

-Lock service: Open()
-Tablets server: Read() and Write()
-Master: CreateTable() and DeleteTable()

Master

-Metadata operations
-Load balancing

Tablet server

-Serves data

Tablet server

-Serves data

Tablet server

-Serves data

Cluster scheduling system

-Handles failover and monitoring

GFS

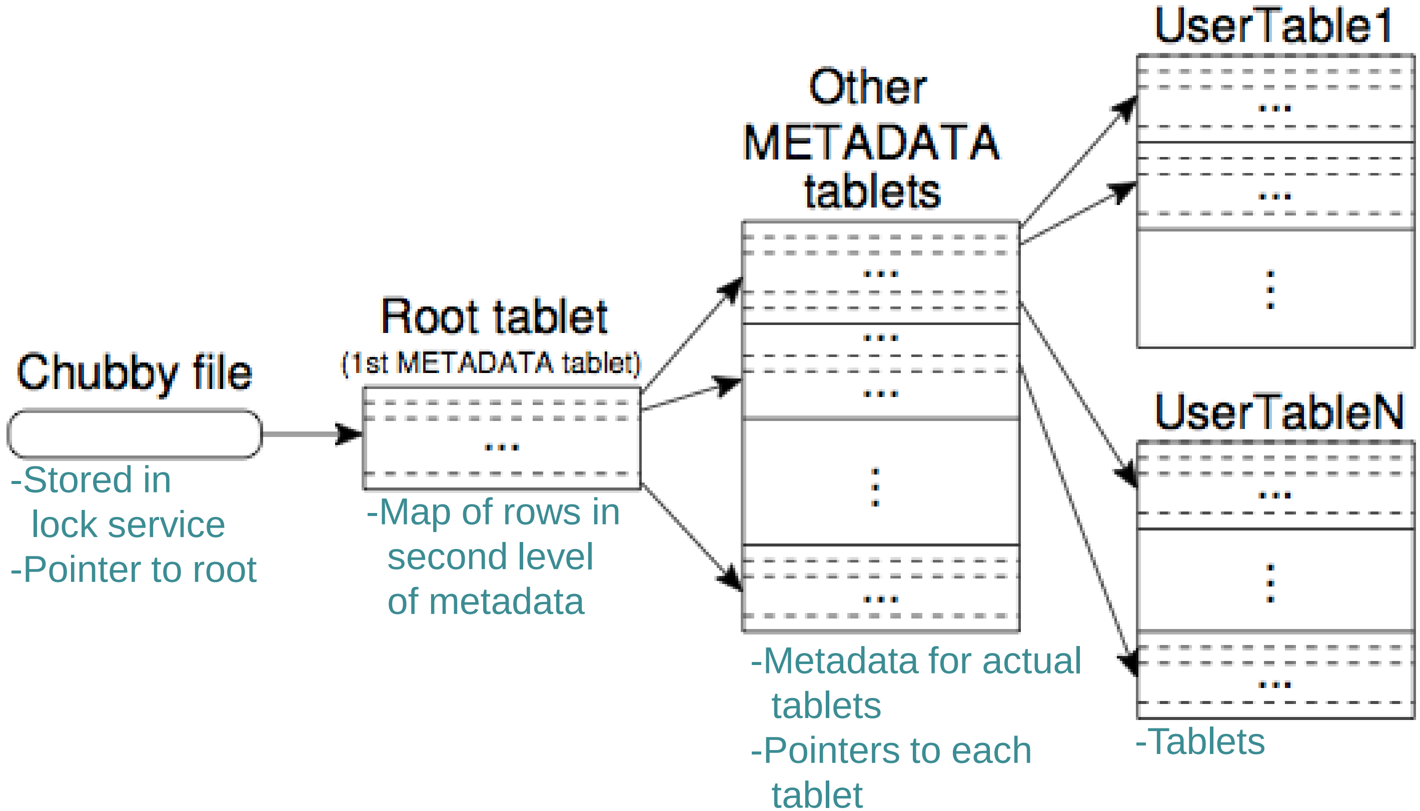
-Tablet data

Lock service

-Holds metadata
-Master election

Locating Tablets

- Metadata for tablet locations and start/end row are stored in a special Bigtable cell



Reading/Writing to Tablets

Write commands

- First write command gets put into a queue/log for commands on that tablet
- Data is written to GFS and when this write command is committed, queue is updated
  - Mirror this write on the tablet’s buffer memory

Read commands

- Must combine the buffered commands not yet committed with the data in GFS

API

- Metadata operations
  - Create and delete tables, column families, change metadata
- Writes (atomic)
  - Set(): write cells in a row
  - DeleteCells(): delete cells in a row
  - DeleteRow(): delete all cells in a row
- Reads
  - Scanner: read arbitrary cells in BigTable
    - Each row read is atomic
    - Can restrict returned rows to a particular range
    - Can ask for just data from one row, all rows, a subset of rows, etc.
    - Can ask for all columns, just certainly column families, or specific columns

Shared Logging

- Logs are kept on a per tablet level
  - Inefficient keep separate log files for each tablet tablet (100 tablets per server)
  - Logs are kept in 64MB chunks
- Problem: Recovery in machine failure becomes complicated because many new machines are all reading killed machine's logs
  - Many I/O operations
- Solved by master chunking killed machine's log file for each new machine

Compression

- Low CPU cost compression techniques are adopted
- Complete across each SSTable for a locality group
  - Used BMDiff and Zippy building blocks of compression
- Keys: sorted strings of (Row, Column, Timestamp)
- Values
  - Grouped by type/column family name
  - BMDiff across all values in one family
- Zippy as final pass over a whole block
  - Catches more localized repetitions
  - Also catches cross-column family repetition
- Compression at a factor of 10 from empirical results

Spanner: The New BigTable

- Is being replaced by Google's new database Spanner (OSDI 2012)
  - http://research.google.com/archive/spanner.html
- A more "true-time" focused API that can manage data across all of Google's datacenters
- Similar to a relational database but still relies on primary key
- Some features: non-blocking reads in the past, lock-free read-only transactions, and atomic schema changes.

Sources

- BigTable: A Distributed Storage System for Structured Data by Fay Chang, Jeffrey Dean, et all – published in 2004
  - http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/archive/bigtable-osdi06.pdf
- BigTable presentation by Google's Jeffrey Dean
  - http://video.google.com/videoplay?docid=7278544055668715642