7680: Distributed Systems

BigTable. Hbase. Spanner.
1: BigTable
Acknowledgement

- Slides based on material from course at UMichigan, U Washington, and the authors of BigTable and Spanner.
**REQUIRED READING**

- Spanner, Google’s globally distributed database. OSDI 2012.
BigTable

- Distributed storage system for managing structured data such as:
  - URLs: contents, crawl metadata, links, anchors, pagerank
  - Per-user data: user preference settings, recent queries/search results
  - Geographic locations: physical entities (shops, restaurants, etc.), roads, satellite image data, user annotations, …
- Designed to scale to a very large size: petabytes of data distributed across thousands of servers
- Used for many Google applications
  - Web indexing, Personalized Search, Google Earth, Google Analytics, Google Finance, … and more
Why BigTable?

- Scalability requirements not met by existent commercial systems:
  - Millions of machines
  - Hundreds of millions of users
  - Billions of URLs, many versions/page
  - Thousands or queries/sec
  - 100TB+ of satellite image data

- Low-level storage optimization helps performance significantly
Goals

- Simpler model that supports dynamic control over data and layout format
- Want asynchronous processes to be continuously updating different pieces of data: access to most current data at any time
- Examine data changes over time: e.g. contents of a web page over multiple crawls
- Support for:
  - Very high read/write rates (millions ops per second)
  - Efficient scans over all or subsets of data
  - Efficient joins of large one-to-one and one-to-many datasets
Design Overview

- Distributed multi-level map
- Fault-tolerant, persistent
- Scalable
  - Thousands of servers
  - Terabytes of in-memory data
  - Petabyte of disk-based data
  - Millions of reads/writes per second, efficient scans
- Self-managing
  - Servers can be added/removed dynamically
  - Servers adjust to load imbalance
Typical Google Cluster

Shared pool of machines that also run other distributed applications

Machine 1
- User app1
- User app2
- Scheduler slave
- GFS chunkserver
- Linux

Machine 2
- User app1
- Scheduler slave
- GFS chunkserver
- Linux

Machine N
- BigTable master
- Scheduler slave
- GFS chunkserver
- Linux
Building Blocks

- **Google File System (GFS)**
  - Stores persistent data (SSTable file format)

- **Scheduler**
  - Schedules jobs onto machines

- **Chubby**
  - Lock service: distributed lock manager, master election, location bootstrapping

- **MapReduce (optional)**
  - Data processing
  - Read/write BigTable data
Chubby

- \{lock/file/name\} service
- Coarse-grained locks
  - Provides a namespace that consists of directories and small files.
  - Each of the directories or files can be used as a lock.
- Each client has a session with Chubby
  - The session expires if it is unable to renew its session lease within the lease expiration time.
- 5 replicas Paxos, need a majority vote to be active
Data Model

- A sparse, distributed persistent multi-dimensional sorted map
- Rows, column are arbitrary strings

- (row, column, timestamp) -> cell contents
Data Model: Rows

- Arbitrary string
- Access to data in a row is atomic
  - Row creation is implicit upon storing data
  - Ordered lexicographically
Rows (cont.)

- Rows close together lexicographically usually on one or a small number of machines
- Reads of short row ranges are efficient and typically require communication with a small number of machines
- Can exploit lexicographic order by selecting row keys so they get good locality for data access

Example:
- math.gatech.edu, math.uga.edu, phys.gatech.edu, phys.uga.edu
- VS
- edu.gatech.math, edu.gatech.phys, edu.uga.math, edu.uga.phys
Data Model: Columns

- Two-level name structure: family: qualifier
- Column family:
  - Is the unit of access control
  - Has associated type information
- Qualifier gives unbounded columns
  - Additional levels of indexing, if desired
Data Model: Timestamps (64bit integers)

Store different versions of data in a cell:
  - New writes default to current time, but timestamps for writes can also be set explicitly by clients

Lookup options
  - Return most recent K values
  - Return all values

Column families can be marked with attributes:
  - Retain most recent K values in a cell
  - Keep values until they are older than K seconds
Data Model: Tablet

- The row range for a table is dynamically partitioned.
- Each row range is called a tablet (typically 10-100 bytes).
- Tablet is the unit for distribution and load balancing.
Storage: SSTable

- Immutable, sorted file of key-value pairs
- Optionally, SSTable can be completely mapped into memory
- Chunks of data plus an index
  - Index is of block ranges, not values
  - Index is loaded into memory when SSTable is open
Tablet vs. SSTable

- Tablet is built out of multiple SSTables

<table>
<thead>
<tr>
<th>Tablet</th>
<th>Start:aardvark</th>
<th>End:apple</th>
</tr>
</thead>
<tbody>
<tr>
<td>64K block</td>
<td>64K block</td>
<td>64K block</td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSTable</th>
<th>64K block</th>
<th>64K block</th>
<th>64K block</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Index</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table vs. Tablet vs. SSTable

- Multiple tablets make up the table
- SSTables can be shared
- Tablets do not overlap, SSTables can overlap
Example: WebTable

- Want to keep copy of a large collection of web pages and related information
- Use URLs as row keys
- Various aspects of web page as column names
- Store contents of web pages in the contents: column under the timestamps when they were fetched.
Implementation

- Library linked into every client
- One master server responsible for:
  - Assigning tablets to tablet servers
  - Detecting addition and expiration of tablet servers
  - Balancing tablet-server load
  - Garbage collection
  - Handling schema changes such as table and column family creation
- Many tablet servers, each of them:
  - Handles read and write requests to its table
  - Splits tablets that have grown too large
- Clients communicate directly with tablet servers for reads and writes.
Deployment

BigTable Cell

Bigtable master
performs metadata ops + load balancing

Bigtable tablet server
serves data

Bigtable tablet server
serves data

Bigtable tablet server

Bigtable client

metadata ops

Bigtable client library

read/write

Open()

Cluster scheduling system
handles failover, monitoring

GFS
holds tablet data, logs

Lock service
holds metadata, handles master-election

BigTable. HBase. Spanner
More about Tablets

- Serving machine responsible for 10 - 1000
  - Usually about 100 tablets

- Fast recovery:
  - 100 machines each pick up 1 tablet for failed machine

- Fine-grained load balancing:
  - Migrate tablets away from overloaded machine
  - Master makes load-balancing decisions
Tablet Location

- Since tablets move around from server to server, given a row, how do clients find the right machine
  - Find tablet whose row range covers the target row
- METADATA: Key: table id + end row, Data: location
- Aggressive caching and prefetching at client side
Tablet Assignment

- Each tablet is assigned to one tablet server at a time.
- Master server
  - Keeps track of the set of live tablet servers and current assignments of tablets to servers.
  - Keeps track of unassigned tablets.
- When a tablet is unassigned, master assigns the tablet to a tablet server with sufficient room.
- It uses Chubby to monitor health of tablet servers, and restart/replace failed servers.
Tablet Assignment: Chubby

- Tablet server registers itself with Chubby by getting a lock in a specific directory of Chubby.
- Chubby gives “lease” on lock, must be renewed periodically.
- Server loses lock if it gets disconnected.
- Master monitors this directory to find which servers exist/are alive.
  - If server not contactable/has lost lock, master grabs lock and reassigns tablets.
  - GFS replicates data. Prefer to start tablet server on same machine that the data is already at.
API

- **Metadata operations**
  - Create/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 a bigtable
    - Each row read is atomic
    - Can restrict returned rows to a particular range
    - Can ask for just data from 1 row, all rows, etc.
    - Can ask for all columns, just certain column families, or specific columns

BigTable. HBase. Spanner
Refinements: Locality Groups

- Can group multiple column families into a locality group
  - Separate SSTable is created for each locality group in each tablet.

- Segregating columns families that are not typically accessed together enables more efficient reads.
  - In WebTable, page metadata can be in one group and contents of the page in another group.
Refinements: Compression

- Many opportunities for compression
  - Similar values in the same row/column at different timestamps
  - Similar values in different columns
  - Similar values across adjacent rows

- Two-pass custom compressions scheme
  - First pass: compress long common strings across a large window
  - Second pass: look for repetitions in small window

- Speed emphasized, but good space reduction (10-to-1)
Refinements: Bloom Filters

- Read operation has to read from disk when desired SSTable is not in memory
- Reduce number of accesses by specifying a Bloom filter:
  - Allows to ask if a SSTable might contain data for a specified row/column pair.
  - Small amount of memory for Bloom filters drastically reduces the number of disk seeks for read operations.
  - Results in most lookups for non-existent rows or columns not needing to touch disk.
### Real Applications

<table>
<thead>
<tr>
<th>Project name</th>
<th>Table size (TB)</th>
<th>Compression ratio</th>
<th># Cells (billions)</th>
<th># Column Families</th>
<th># Locality Groups</th>
<th>% in memory</th>
<th>Latency-sensitive?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crawl</td>
<td>800</td>
<td>11%</td>
<td>1000</td>
<td>16</td>
<td>8</td>
<td>0%</td>
<td>No</td>
</tr>
<tr>
<td>Crawl</td>
<td>50</td>
<td>33%</td>
<td>200</td>
<td>2</td>
<td>2</td>
<td>0%</td>
<td>No</td>
</tr>
<tr>
<td>Google Analytics</td>
<td>20</td>
<td>29%</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0%</td>
<td>Yes</td>
</tr>
<tr>
<td>Google Analytics</td>
<td>200</td>
<td>14%</td>
<td>80</td>
<td>1</td>
<td>1</td>
<td>0%</td>
<td>Yes</td>
</tr>
<tr>
<td>Google Base</td>
<td>2</td>
<td>31%</td>
<td>10</td>
<td>29</td>
<td>3</td>
<td>15%</td>
<td>Yes</td>
</tr>
<tr>
<td>Google Earth</td>
<td>0.5</td>
<td>64%</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>33%</td>
<td>Yes</td>
</tr>
<tr>
<td>Google Earth</td>
<td>70</td>
<td></td>
<td>9</td>
<td>8</td>
<td>3</td>
<td>0%</td>
<td>No</td>
</tr>
<tr>
<td>Orkut</td>
<td>9</td>
<td></td>
<td>0.9</td>
<td>8</td>
<td>5</td>
<td>1%</td>
<td>Yes</td>
</tr>
<tr>
<td>Personalized Search</td>
<td>4</td>
<td>47%</td>
<td>6</td>
<td>93</td>
<td>11</td>
<td>5%</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Limitations

- No transactions supported
- Does not support full relational data model
- Achieved throughput is limited by GFS
Lessons Learnt

- Large distributed systems vulnerable to many type of failures
  - Memory and network corruption
  - Large clock skew
  - Hung machines
  - Extended and asymmetric network partitions
  - Bugs in other systems
- Proper system-level monitoring critical
- Simple design better
- Do not add new features before they are needed
2: HBase
HBase

- Open-source, distributed, versioned, column-oriented data store, modeled after Google's Bigtable
- Random, real time read/write access to large data:
  - Billions of rows, millions of columns
  - Distributed across clusters of commodity hardware
History

- **2006.11**
  - Google releases paper on BigTable

- **2007.2**
  - Initial HBase prototype created as Hadoop contrib.

- **2007.10**
  - First useable HBase

- **2008.1**
  - Hadoop become Apache top-level project and HBase becomes subproject

- **Current stable release 0.98.x**
Tables have one primary index, the row key.
No join operators.
Scans and queries can select a subset of available columns.
There are three types of lookups:
  - Fast lookup using row key and optional timestamp.
  - Full table scan
  - Range scan from region start to end.
HBase Is Not ...(2)

- Limited atomicity and transaction support.
  - HBase supports multiple batched mutations of single rows only.
  - Data is unstructured and untyped.
- No accessed or manipulated via SQL.
  - Programmatic access via Java, REST, or Thrift APIs.
  - Scripting via JRuby.
3: Spanner
Limitations of BigTable

- Difficult to use for applications that
  - have complex, evolving schemas,
  - want strong consistency in the presence of wide-area replication
What is Spanner

- Scalable, multi-version, globally-distributed, and synchronously-replicated database
- Distribute data at global scale and support externally-consistent distributed transactions.

**Features:**
- non-blocking reads in the past
- lock-free read-only transactions,
- atomic schema changes

**Scale up to**
- millions of machines
- hundreds of datacenters
- trillions of database rows
What is Spanner

- Applications can control replication configurations for data
- Applications can specify constraints
  - to control which datacenters contain which data, how far data is from its users (to control read latency)
  - how far replicas are from each other (to control write latency)
  - how many replicas are maintained (to control durability, availability, and read performance).
- Data can also be dynamically and transparently moved between datacenters by the system to balance resource usage across datacenters
Spanner – key idea

- Consistent reads and writes
- How:
  - use global commit timestamps to transactions, even though transactions may be distributed.
  - timestamps represent serialization order.
  - provide such guarantees at global scale
- How to get the global timestamps: TrueTime
- Relies on existing algorithms as Paxos and 2PC
Architecture

- **Instance** — it’s called universe; examples: test, deployment, production
  - Universe master
  - Placement master
    - handles automated movement of data across zones on the timescale of minutes
    - periodically communicates with the spanservers to find data that needs to be moved, either to meet updated replication constraints or to balance load.
  - Universe consists of zones
    - Denotes physical isolation
    - Several zones can be in a datacenter
Architecture

Zone 1
- zonemaster
- location proxy
- spanserver

Zone 2
- zonemaster
- location proxy
- spanserver

Zone N
- zonemaster
- location proxy
- spanserver

universe master
placement driver
Zones

- Zonemaster
  - assigns the data to span servers

- Spanservers
  - hundreds to thousands
  - store data
  - responsible for between 100 and 1000 instances of a data structure called a *tablet* (different from the BigTable tablet)
  - each data has a timestamp

- Location proxies
  - used by clients to locate the spanservers assigned to serve their data
Replication

- Other group's participant leader
- Participant leader
- Transaction manager
- Lock table
- Leader
- Replica
- Paxos
- Tablet
- Colossus
- Data Center X
- Data Center Y
- Data Center Z
More about replication

- Directory – analogous to bucket in BigTable
  - Smallest unit of data placement
  - Smallest unit to define replication properties

- 2PC and Paxos-based replication

- Back End: Colossus (successor to GFS)
  - Paxos State Machine on top of each tablet stores meta data and logs of the tablet.

- Leader among replicas in a Paxos group is chosen and all write requests for replicas in that group initiate at leader.

- Transaction Leader
  - Is Paxos Leader if transaction involves one Paxos group
TrueTime

- Leverages hardware features like GPS and Atomic Clocks
- Implemented via TrueTime API
  - Key method being now() which not only returns current system time but also another value ($\varepsilon$) which tells the maximum uncertainty in the time returned
- Set of time master server per datacenters and time slave daemon per machines
- Majority of time masters are GPS fitted and few others are atomic clock fitted (Armageddon masters)
- Daemon polls variety of masters and reaches a consensus about correct timestamp
TrueTime uses both GPS and Atomic clocks since they are different failure rates and scenarios.

Two other boolean methods in API are:
- After(t) – returns TRUE if t is definitely passed
- Before(t) – returns TRUE if t is definitely not arrived

TrueTime uses these methods in concurrency control and to serialize transactions.
After() is used for Paxos Leader Leases

- Uses after(Smax) to check if Smax is passed so that Paxos Leader can abdicate its slaves.

- Paxos Leaders can not assign timestamps(Si) greater than Smax for transactions(Ti) and clients can not see the data committed by transaction Ti till after(Si) is true.

- After(t) – returns TRUE if t is definitely passed
- Before(t) – returns TRUE if t is definitely not arrived

- Replicas maintain a timestamp tsafe which is the maximum timestamp at which that replica is up to date.
TrueTime

- **Read-Write** – requires lock.
- **Read-Only** – lock free.
  - Requires declaration before start of transaction.
  - Reads information that is up to date
- **Snapshot Read** – Read information from past by specifying a timestamp or bound
  - Use specifies specific timestamp from past or timestamp bound so that data till that point will be read.
Applications

- Google advertising backend application – F1
- Replicated across 5 datacenters spread across US