FAQs

- Term project presentations
  - 4/30 (6 teams), 5/2 (6 teams), 5/4 (4 teams)
  - Please attend at least 2 presentation sessions and ask questions or provide comments
  - Participation score (attendance + question)

Today's topics

- No SQL storage

This material is built based on,


Sloppy quorum

- All read and write operations are performed on the first N healthy nodes from the preference list
  - May not always be the first N nodes on the hashing ring

- Hinted handoff
  - If a node is temporarily unavailable, data is propagated to the next node in the ring
  - Metadata contains information about the originally intended node
  - Stored in a separate local database and scanned periodically

- Upon detecting that the original node is recovered,
  - A data delivery attempt will be made
  - Once the transfer succeeds, the data at the temporary node will be removed
Example: Updated

The data will be sent to the node D if C is temporarily down. After the recovery, D will send data to C, then it will remove the data.

What if $W$ is 1?

- Applications that need the highest level of availability can set $W$ as 1
  - Under Amazon's model
    - A write is accepted as long as a single node in the system has durably written the key to its local store
    - A write request is rejected, only if all nodes in the system are unavailable

Anti-entropy protocol

- Replica synchronization protocol
  - Hinted replica can be lost before they can be returned to the original replica node
  - Detect inconsistencies between the replicas faster
  - Minimize the amount of transferred data
- Dynamo uses Merkle tree

Merkle tree

- Hash tree where leaves are hashes of the values of individual keys
  - Parent nodes are hashes of their respective children
- Each branch of the tree can be checked independently
  - Without requiring nodes to download the entire tree or dataset
- If the hash values of the root of two trees are equal
  - The values of the leaf nodes in the tree are equal
  - No synchronization needed

Uses of Merkle tree

- Merkle trees can be used to verify any kind of data stored, handled and transferred
  - Used in a peer-to-peer network
- Trusted computing systems
  - Sun's ZFS (Zeta File System)
- Google's Wave protocol
- Git
- Cassandra and Dynamo
- Bittorrent protocol
How Merkle tree works

- Each node maintains a separate Merkle tree for each key range
- Two nodes exchange the root of the Merkle tree corresponding to the key ranges that they host in common
- Node performs tree traversal and determines if there is any difference
- Perform the appropriate synchronization action
- Disadvantage
  - When a new node joins or leaves
    - Tree needs to be recalculated

How Dynamo uses Merkle tree

- Partitioning
- High Availability for writes
- Handling temporary failures
- Recovering from permanent failures
- Membership and failure detection

Identifier “Ring” Membership

- A node outage should not result in re-balancing of the partition assignment or repair of the unreachable replicas
- A node outage is mostly temporary

- Gossip-based protocol
  - Propagates membership changes
  - Maintains an eventually consistent view of membership

- Each node contacts a peer every second
  - Random selection
  - Two nodes reconcile their persisted membership change history

Logical partitioning

- Almost concurrent addition of two new nodes
  - Node A joins the ring
  - Node B joins the ring

- A and B consider themselves members of the ring
  - Yet neither would be immediately aware of each other
  - A does not know the existence of B
  - Logical partitioning
External Discovery

- Addresses the logical partitioning
- Seeds
  - Discovered via an external mechanism
  - Known to all nodes
  - Statically configured (or from a configuration service)
- Seed nodes will eventually reconcile their membership with all of the nodes

Failure Detection

- Attempts to
  - Avoid communication with unreachable peers during a get or put operation
  - Transfer partitions and hinted replicas
- Detecting communication failures
  - When there is no response to an initiated communication
- Responding to communication failures
  - Sender will try alternate nodes that map to failed node's partitions
  - Periodically retry failed node for recovery

Part 2. Large scale data storage system

NoSQL Storage: 2. Column Family Stores

Google’s Big Table

This material is built based on,


Column-family storage

- Optimized for the data
  - Sparse columns and no schema
- Aggregate-oriented storage
  - Most data interaction is done with the same aggregate
  - Aggregate
    - A collection of data that we interact with as a unit
- Stores groups of columns (column families) together
Storing data in a column-family store

- The stores organize their columns into column families
- Each column may be part of a single column family
- The column acts as unit for access
- The assumption is that data for a particular column family will be usually accessed together

BigTable

- Google’s first answer to the question
  - “How do you store semi-structured data at scale?”

Scalability and latency

- Scale in capacity
  - E.g., wiki table
    - 100,000,000,000 pages * 10 versions per page * 300KB/version
  - 20PB of data (200 million gigabytes)
  - E.g., google maps
    - 10TB of usable image data
- Scale in throughput
  - Hundreds of millions of users
  - Tens of thousands to millions of queries per second
- Low latency
  - A few dozen milliseconds of total budget “inside” Google
  - Probably have to involve several dozen internal services per request
  - Few milliseconds for lookup
  - Jake D. Brutlag and Hilary Hutchinson and Maria Stone, “User preference and search engine latency”, in Proc. ASA Joint Statistical Meetings, 2008

BigTable has been used by,

- Web indexing
- Google Reader
- Google Maps
- Google Book Search
- Google Earth
- Blogger.com
- Google Code
- YouTube
- Gmail
  - ...

BigTable

- Provides a simple data model
  - Dynamic control over the data layout and format
  - 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

- Data in BigTable
  - Uninterpreted strings
  - Clients can serialize various forms of structured and semi-structured data into these strings

- Clients can control locality of their data
- Clients can control whether to serve data out of memory or from disk
Topics in BigTable

1. Data model
2. Locating tablet
3. Data Compaction
4. Data Compression
5. Caching and prefetching

Data Model

- A BigTable is a sparse, distributed, persistent multi-dimensional sorted map
- The map is indexed by,
  - A row key
  - A column key
  - A timestamp
- Each value in the map is an uninterpreted array of bytes

(example)

\[
\text{row: string, column: string, time: int64} \rightarrow \text{string}
\]

Example of data model with Webtable

- Webtable
  - A large collection of web pages and related information
  - URLs
  - Contents
  - Information

Rows

- Row keys
  - Arbitrary strings
  - Every read or write of data under a single row key is atomic

- BigTable maintains data in lexicographic order by row key

- Row range for a table
  - Dynamically partitioned

Tablets

- Large tables are broken into tablets at row boundaries
  - A tablet holds a contiguous range of rows
  - Clients can often choose row keys to achieve locality
  - Aim for ~ 100KB to 200MB of data per tablet

- Serving machine responsible for ~100 tablets
  - Fast recovery
  - Allows a 100 machines to each pick up 1 tablet from the failed machine
  - Fine-grained load balancing
  - Migrate tablets away from the overloaded machine
  - Master makes load balancing decisions
### Tablets

- Read of short row ranges are efficient
  - Require communication with only a small number of machines
  - Clients get good locality for their data access
- `maps.google.com/index.html` is stored using the key `com.google.maps/index.html`
  - Storing pages under the same domain near each other makes some host and domain analysis more efficient

### Column Families

- Column keys are grouped into sets called column families
  - Basic unit of access control
- All data stored in a column family is usually of the same type
  - BigTable compresses 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

### Column Families

- Column key
  - `family:qualifier`
  - Family name must be printable
  - Qualifier may be an arbitrary string
- Access control and disk/memory accounting
  - Performed at the column family level

### Webtable with multiple column-families

<table>
<thead>
<tr>
<th>row keys</th>
<th>column family</th>
<th>column family</th>
</tr>
</thead>
<tbody>
<tr>
<td>comizzlies</td>
<td>anchor:www.c-pan.org</td>
<td>CNN</td>
</tr>
<tr>
<td>com.abc</td>
<td>anchor:www.abc.com</td>
<td>Tech</td>
</tr>
<tr>
<td>com.weather</td>
<td>anchor:www.weather.com</td>
<td>News</td>
</tr>
</tbody>
</table>

### Timestamps

- Each cell in Bigtable can contains multiple versions of the same data
  - Indexed by timestamp
- BigTable timestamp
  - 64-bit integers
  - Assigned by BigTable
  - Near-time in microseconds
  - Explicitly assigned by client application
- Application should generate unique timestamp to avoid collisions
  - Different versions of a cell are stored in decreasing timestamp order
  - The most recent versions can be read first

### API

- Functions for creating and deleting tables and column families
- Changing cluster, table, and column-family metadata (access control rights)
Garbage collection

- Two per-column-family settings
  - Tell Bigtable to garbage-collect cell versions automatically
  - The last n versions are kept
    - i.e., only recent versions are kept
Access to the block

- In-memory map of keys to (SSTables, memtable)
- Lookup can be performed with a single disk seek
  - Find the block by performing a binary search of the in-memory index
  - Read the block from disk

Locating tablets

- Since tablets move around from server to server, given a row, how do clients find the right machine?
  - Need to find tablet whose row range covers the target row
- Using the BigTable master
  - Central server almost certainly would be bottleneck in large system
  - Instead: store special tables containing tablet location info in BigTable cell itself