Notes 13: Distributed Column Stores
Previous Topics

Data
  Database design

Queries
  Query processing
  Localization
  Operators
  Optimization

Transactions
  Concurrency control
  Reliability
  Replication

Client-server architecture
Relational data
Good understanding of
  What the data is
  Where the data is
Previous Topics

Data
- Database design
- Query processing
  - Localization
  - Operators
  - Optimization
- Transactions
  - Concurrency control
  - Reliability
  - Replication

Client-server architecture

Relational data

Good understanding of

What the data is

Where the data is
Wide Column Stores

Bigtable

*Bigtable: A Distributed Storage System for Structured Data.* F. Chang et al., OSDI 2006

HBase

Cassandra

*Cassandra: A Decentralized Structured Storage System.* A. Lakshman and P. Malik, SIGOPS 2010
Basic Idea

Key-value store

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>$k_3$</td>
<td>$v_3$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$k_n$</td>
<td>$v_n$</td>
</tr>
</tbody>
</table>
Basic Idea

Key-value store

API

lookup(key) → value
scan(key range) → values
insert(key, value)
delete(key)
Fragmentation

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>$k_3$</td>
<td>$v_3$</td>
</tr>
<tr>
<td>$k_4$</td>
<td>$v_4$</td>
</tr>
<tr>
<td>$k_5$</td>
<td>$v_5$</td>
</tr>
<tr>
<td>$k_6$</td>
<td>$v_6$</td>
</tr>
<tr>
<td>$k_7$</td>
<td>$v_7$</td>
</tr>
<tr>
<td>$k_8$</td>
<td>$v_8$</td>
</tr>
<tr>
<td>$k_9$</td>
<td>$v_9$</td>
</tr>
</tbody>
</table>

Server 1

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>$k_3$</td>
<td>$v_3$</td>
</tr>
<tr>
<td>$k_4$</td>
<td>$v_4$</td>
</tr>
</tbody>
</table>

Server 2

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_5$</td>
<td>$v_5$</td>
</tr>
<tr>
<td>$k_6$</td>
<td>$v_6$</td>
</tr>
</tbody>
</table>

Server 3

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_7$</td>
<td>$v_7$</td>
</tr>
<tr>
<td>$k_8$</td>
<td>$v_8$</td>
</tr>
<tr>
<td>$k_9$</td>
<td>$v_9$</td>
</tr>
</tbody>
</table>
Fragmentation

Horizontal fragmentation
Partition vector
Auto-sharding in HBase
  Dynamic repartitioning
  Based on size

Partition = tablet
Server = tablet server
# Replication

<table>
<thead>
<tr>
<th>Primary</th>
<th>Backup</th>
<th>Backup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>key</strong></td>
<td><strong>key</strong></td>
<td><strong>key</strong></td>
</tr>
<tr>
<td>$k_1$</td>
<td>$k_1$</td>
<td>$k_1$</td>
</tr>
<tr>
<td>$v_1$</td>
<td>$v_1$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$k_2$</td>
<td>$k_2$</td>
</tr>
<tr>
<td>$v_2$</td>
<td>$v_2$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>$k_3$</td>
<td>$k_3$</td>
<td>$k_3$</td>
</tr>
<tr>
<td>$v_3$</td>
<td>$v_3$</td>
<td>$v_3$</td>
</tr>
<tr>
<td>$k_4$</td>
<td>$k_4$</td>
<td>$k_4$</td>
</tr>
<tr>
<td>$v_4$</td>
<td>$v_4$</td>
<td>$v_4$</td>
</tr>
</tbody>
</table>

**Cassandra**

- Replication factor (number of copies)
- Read/write levels: one, quorum, all
- Policy: simple vs. topology-based
Distributed Access

**Directory-based**
(table, key) → tablet server
Can be implemented as a special table

**Bigtable**
Relies on a distributed lock service
Used for other metadata as well
  E.g., schema, access control
Directory organized as a B+ tree
  With depth limit
# Tablet Internals

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_3$</td>
<td>$v_3$</td>
</tr>
<tr>
<td>$k_6$</td>
<td>$v_6$</td>
</tr>
<tr>
<td>$k_7$</td>
<td>✘</td>
</tr>
<tr>
<td>$k_9$</td>
<td>$v_9$</td>
</tr>
</tbody>
</table>

**Memory**

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>$v_3$</td>
</tr>
<tr>
<td>$k_3$</td>
<td>$v_6$</td>
</tr>
<tr>
<td>$k_7$</td>
<td>$v_7$</td>
</tr>
<tr>
<td>$k_9$</td>
<td>$v_9$</td>
</tr>
</tbody>
</table>

**Disk**

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_2$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>$k_6$</td>
<td>$v_6$</td>
</tr>
<tr>
<td>$k_8$</td>
<td>✘</td>
</tr>
</tbody>
</table>
Tablet Internals

*Design philosophy: sequential disk I/O only*

In-memory table flushed to disk periodically
- Perform minor compaction
  - Each flush produces a *file layer* (i.e., a sharded file)
  - Files are immutable
- Writes are efficient
- Reads are efficient only when data is in memory
  - Can use Bloom filters to optimize lookups
  - Need to read all layers to reconstruct value
- Layers merged into single one periodically

Notes 13
Data Model Details

Sparse, distributed, persistent, multidimensional map
(row: string, column: string, timestamp: int64) → value: string

Rows
Row is the basic unit of fragmentation
Atomicity: read/write on row key

Columns
Grouped into families (column = family:qualifier)
Families must be declared in the schema, qualifiers are arbitrary
Family is the basic unit of locality and access control
Data Model Details

Timestamps
Microseconds or client-specified
Collision avoidance is the responsibility of the client
Support for multiple versions
  Last $n$ versions
  Recent versions (based on time)
  Periodic garbage collection deletes old versions
### Data Model Details

<table>
<thead>
<tr>
<th>key</th>
<th>a</th>
<th>b:x</th>
<th>b:y</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>$a_1$</td>
<td>$x_1$</td>
<td>$y_1$</td>
<td>$c_1$</td>
<td>$d_1$</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$a_2$</td>
<td>null</td>
<td>$y_2$</td>
<td>$c_2$</td>
<td>$d_2$</td>
</tr>
<tr>
<td>$k_3$</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>$c_3$</td>
<td>$d_3$</td>
</tr>
<tr>
<td>$k_4$</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>$d_4$</td>
</tr>
<tr>
<td>$k_5$</td>
<td>$a_5$</td>
<td>$x_5$</td>
<td>$y_5$</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Data Model Details

<table>
<thead>
<tr>
<th>key</th>
<th>a</th>
<th>b:x</th>
<th>b:y</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>k_1</td>
<td>a_1</td>
<td>x_1</td>
<td>y_1</td>
<td>c_1</td>
<td>d_1</td>
</tr>
<tr>
<td>k_2</td>
<td>a_2</td>
<td>null</td>
<td>y_2</td>
<td>c_2</td>
<td>d_2</td>
</tr>
<tr>
<td>k_3</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>c_3</td>
<td>d_3</td>
</tr>
<tr>
<td>k_4</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>d_4</td>
</tr>
<tr>
<td>k_5</td>
<td>a_5</td>
<td>x_5</td>
<td>y_5</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

Vertical fragmentation

- | key     | a    |
- | k_1     | a_1  |
- | k_2     | a_2  |
- | k_5     | a_5  |

- | key     | b:x  |
- | k_1     | x_1  |
- | k_2     | null |
- | k_5     | x_5  |

- | key     | b:y  |
- | k_1     | y_1  |
- | k_2     | y_2  |
- | k_5     | y_5  |

- | key     | c    |
- | k_1     | c_1  |
- | k_2     | c_2  |
- | k_3     | c_3  |

- | key     | d    |
- | k_1     | d_1  |
- | k_2     | d_2  |
- | k_3     | d_3  |
- | k_4     | d_4  |
Data Model Details

Column store
Good for sparse data
Good for column scans
Not so good for full row reads
Vertical partitioning done manually
   Need to know access patterns to optimize
Failure Recovery

Bigtable

One commit log stored per tablet server
Recovery procedure

Failed tablet server’s log sorted by row (sharded)
New tablet server gets scans to retrieve recent updates
Failure Recovery

Cassandra
No master, all nodes in a *cluster* are equal
Failure Recovery

Cassandra
No master, all nodes in a cluster are equal

Access any table in cluster at any server

Tablet server 1 → Tablet server 2 → Tablet server 3

Forward request as needed

Peer-to-peer setup
Transient failure model
- No automatic permanent removal of nodes from the cluster
- Recovering node receives updates from replicas
  - Manual repair may be necessary if failed node had unsent updates
Optimizations

Locality groups (Bigtable)
Column families grouped together
The columns in a locality group
  Are stored in the same file
  Share parameters
    E.g., can declare a group to be all in-memory (lazy load)

Multiple masters (HBase)
Support for hot masters kept on standby
Optimizations

Bloom filters
Probabilistic data structures for testing membership in a set
  Quick way to check whether a row + column is present (not null)
Client may configure one for a locality group / column

Read caches
Scan cache for rows
Block cache for files (shards)
Comparison

(+) Dynamic control over data layout and format
(+) Clients may reason about locality properties

(?) Data stored as uninterpretable strings
   Structured/semi-structured data is serialized

(−) Relational representation
(−) (Declarative) query language
(−) Multi-row transactions
(−) Automatic optimization
Summary

Wide column stores
Key-value databases tuned for the storage and simple retrieval of large numbers of dynamic columns with often sparse data per row

Data model
Implementation
Fragmentation
Replication
Failure recovery
Optimizations