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

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>k₁</td>
<td>v₁</td>
</tr>
<tr>
<td>k₂</td>
<td>v₂</td>
</tr>
<tr>
<td>k₃</td>
<td>v₃</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>kₙ</td>
<td>vₙ</td>
</tr>
</tbody>
</table>

**API**

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

**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>key</td>
<td>value</td>
<td>key</td>
</tr>
<tr>
<td>k₁</td>
<td>v₁</td>
<td>k₁</td>
</tr>
<tr>
<td>k₂</td>
<td>v₂</td>
<td>k₂</td>
</tr>
<tr>
<td>k₃</td>
<td>v₃</td>
<td>k₃</td>
</tr>
<tr>
<td>k₄</td>
<td>v₄</td>
<td>k₄</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**

**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
**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

**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**

**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**

- Tablet server
- Master
- Spare server

- Log
- Files
- GFS/HDFS

- Write-ahead logging

- 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

**Cassandra**

- No master, all nodes in a **cluster** are equal

  - Tablet server 1
  - Tablet server 2
  - Tablet server 3

- Access any table in cluster at any server

- 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