CockroachDB

Scalable, survivable, strongly consistent, SQL

presented by Daniel Harrison / Member of Technical Staff
About Me

- Previously Google and Foursquare
- At Cockroach Labs for two and a half years
- We have several distributed consensus experts but I am not one of them
Agenda

- Motivation
- High-level architecture
- Some CockroachDB Features
- Q & A
- Interruptions are encouraged!
Motivation
Limitations of Existing Databases

**Relational**
- Hard to scale horizontally
  - **Scalability**: manual sharding results in high operational complexity and application rewrites
  - **Replication**: wasted resources (stand-by servers) or lost consistency (asynchronous replication)

**NoSQL**
- Scalability with strings attached
  - **Limited transactions**: developer burden due to complex data modeling
  - **Limited indexes**: lost flexibility with querying and analytics
  - **Eventual consistency**: correctness issues and higher risk of data corruption
CockroachDB: The Best of Both Worlds

- Single binary/symmetric nodes
- Applications see one logical DB, including cross-datacenter, global
- Self-healing/self-balancing
- Scale out is as simple as adding nodes
- SQL
High-Level Architecture
Abstraction Stack

- SQL
- Transactional KV
- Distribution
- Replication
- Storage
Transactional KV

- Monolithic sorted key-value map
- Automatically replicated and distributed
- Consistent
- Self-healing
Transactional KV: ACID

- **Atomicity.** All operations or no operations.
- **Consistency.** No violating constraints.
- **Isolation.** Exclusive database access.
- **Durability.** Committed data survives crashes.
SQL: Structured Data Model

- Tables
SQL: Structured Data Model

- Tables
- Rows

<table>
<thead>
<tr>
<th>Inventory</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SQL: Structured Data Model

- Tables
- Rows
- Columns

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Glove</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Ball</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Shirt</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Shorts</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>Bat</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>Shoes</td>
<td>4</td>
</tr>
</tbody>
</table>
## SQL: Structured Data Model

- **Tables**
- **Rows**
- **Columns**
- **Indexes**

### Tables and Rows

#### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Name</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ball</td>
<td>1</td>
<td>Glove</td>
<td>1</td>
</tr>
<tr>
<td>Bat</td>
<td>2</td>
<td>Ball</td>
<td>4</td>
</tr>
<tr>
<td>Glove</td>
<td>3</td>
<td>Shirt</td>
<td>2</td>
</tr>
<tr>
<td>Shirt</td>
<td>4</td>
<td>Shorts</td>
<td>12</td>
</tr>
<tr>
<td>Shoes</td>
<td>5</td>
<td>Bat</td>
<td>0</td>
</tr>
<tr>
<td>Shorts</td>
<td>6</td>
<td>Shoes</td>
<td>4</td>
</tr>
</tbody>
</table>

#### Inventory

```sql
CREATE INDEX Name_Idx ON Inventory (Name);
```
CREATE TABLE inventory ( 
    id INTEGER PRIMARY KEY, 
    name VARCHAR, 
    quantity INTEGER, 
    INDEX name_index (name) 
);

SQL

Transactional KV
Distribution
Replication
SQL: Key anatomy

```
INSERT INTO inventory VALUES (1, 'Apple', 12);
INSERT INTO inventory VALUES (2, 'Orange', 15);
```

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Apple</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>Orange</td>
<td>15</td>
</tr>
</tbody>
</table>

```
key /<table>/<index>/<key>/<column>    Value
/inventory/primary/1/name             Apple
/inventory/primary/1/quantity         12
/inventory/primary/2/name             Orange
/inventory/primary/2/quantity         15
```
Distribution: Sharding

The data is split into ~64MB ranges. Each holds a contiguous range of the key space.

Ø-lem
- apricot
- banana
- blueberry
- cherry
- grape

lem-pea
- lemon
- lime
- mango
- melon
- orange

pea-∞
- peach
- pear
- pineapple
- raspberry
- strawberry
Distribution: Index

An index maps from key to range ID

shard index
- Ø-lem
- lem-pea
- pea-∞

Ø-lem
- apricot
- banana
- blueberry
- cherry
- grape

lem-pea
- lemon
- lime
- mango
- melon
- orange

pea-∞
- peach
- pear
- pineapple
- raspberry
- strawberry
Distribution: Split

Split when a range is too large (or too hot, or...)
Replication: Survivability

- Each range is replicated to three or more nodes
- Consensus via Raft
- "Leaseholder" optimization to allow reads to be served without consensus
- Multi-Version Concurrency Control
Data Distribution: Placement

Each range is replicated to three or more nodes
Data Distribution: Rebalancing

Node 1

Node 2

Node 3

Node 4

Adding a new (empty) node
Data Distribution: Rebalancing

A new replica is allocated, data is copied.
Data Distribution: Rebalancing

Node 1
- Range 1
- Range 2
- Range 3

Node 2
- Range 1
- Range 2
- Range 3

Node 3
- Range 1
- Range 2
- Range 3

Node 4
- Range 3

The new replica is made live, replacing another.
Data Distribution: Rebalancing

Node 1
- Range 1
- Range 2
- Range 3

Node 2
- Range 1
- Range 2
- Range 3

Node 3
- Range 1
- Range 2
- Range 3

Node 4
- Range 3

The old (inactive) replica is deleted.
Data Distribution: Rebalancing

Process continues until nodes are balanced.
Data Distribution: Recovery

Losing a node causes recovery of its replicas.
A new replica gets created on an existing node.
Data Distribution: Recovery

Once at full replication, the old replicas are forgotten.
Some CockroachDB Features
Geographic Zone Configurations

- Control where your data is
- Nodes are tagged with attributes and hierarchical localities
- Rules target these
- Zero downtime data migrations
Geo-Partitioning

- Domicile data according to customer
  - Meet regulatory constraints
  - Low-latency reads / writes
- One *logical* database
  - Simplified app development
Distributed SQL

```sql
SELECT l_shipmode, AVG(l_extendedprice) FROM lineitem GROUP BY l_shipmode;
```
Online Schema Changes

- Based on Google's F1 Paper
- State machine, possibly with backfill
- Appears instantaneous to the client
- Zero downtime
Backup/Restore

- Distributed
- Consistent to a point in time
- Incremental
Other Topics

- (New) Query optimizer
- Graphical Admin UI
- Distributed Import
- (New) Change Data Capture
Questions?

jobs@cockroachlabs.com
github.com/cockroachdb
www.cockroachlabs.com