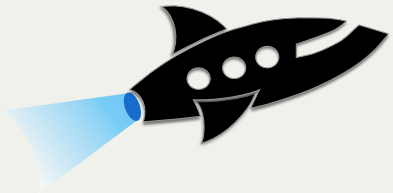
NewSQL Database and its application to Massively Distributed OpenStack

Ronan-Alexandre Cherrueau

Inria, Discovery Initiative

Inria
INVENTEURS DU MONDE NUMÉRIQUE



What is a NewSQL Database?

Globally Distributed SQL Database

- Scalable – Table data are *distributed* & all nodes can accept query
- Survivable – Loose one node but not its data thanks to *replication*
- Consistent – Preserve integrity by supporting *transactions*

What is CockroachDB?

- OpenSource NewSQL Database
- Speaks ”PostgreSQLtongue”
- Made by Google Spanner engineers
- Developed at Cockroach Labs

Interest for Massively Distributed OpenStack

OpenStack services use MariaDB database to store their state

- Replace MariaDB by CockroachDB
- One CockroachDB node per OpenStack instance

$\Rightarrow$ All OpenStack instances shared the Database

$\Rightarrow$ Make all my OpenStack instances collaborative *for free*

  - I do not have to re-implement the global view as with multi-region deployment

Shall we use NewSQL (e. g. CockroachDB) instead of MariaDB for a Massively Distributed OpenStack?

RDBMS (OpenStack Single Cell)

Relational Database

- i. e. MariaDB, Oracle, PostregSQL
- Implementation supports Transaction

Problems

- Designed to run on a *single* machine
- How to scale to millions of millions of requests?
  - Scaling by clusturing is not an option for a Massively Distributed OpenStack
  - Active/Active rep. doesn't work in WAN
  - Active/Passive rep. doesn't help to scale

Custom Sharding (OpenStack Multi-Cell V2)

Scale by spreading the load on different database servers

- Partition the database horizontally and put each partition on a separate database instance.
- Application do the glue between each partition

Problems

- More complex query to handle sharding logic: Nova cells V2 instance list operation not sort and paginate results properly and the performance is considerably slower.
- Writing global *transaction* is a nightmare: Nova cells v2 removes VM migration and quota features[1].
- Managing *distribution* is difficult: Re-balance shards, change database schema, ...
- Managing *replication* is tough: one replication system per shard?

[1] Caveats of a Multi-Cell deployment: `https://docs.openstack.org/nova/pike/user/cellsv2_layout.html#caveats-of-a-multi-cell-deployment`

NoSQL Database (OpenStack + Rome)

Assumption

Scaling and *Survivability* are the important properties (not the *Consistency*).

Key Value Store based

- *Distribution* for free with a hashing function
  - Deterministically map hashed keys to server
  - Inefficient range scans
- Simple *Replication* with active/passive model
  - Don't care about consistency
  - Keep copies in sync with "last write wins"

Problem: sacrificed a lot to get there

- No relational model (custom APIs instead of SQL)
- No transaction

$\Rightarrow$ Pushes the complexity back to the developers side

NewSQL Database

- Scalable
- Survivable
- Consistent

NewSQL Database

- Scalable – How it *Distributes* data?
- Survivable – How it *Replicats* data?
- Consistent – How it implements *Transaction*?

Good Candidate for Massively Distributed OpenStack? Next, a focus on CockroachDB Implementation

Scalability: Data Distribution in CockroachDB

Relies on Key Value Store with *order-preserving* distribution function

- Divided sorted key space up into ranges of nearly equal size
- Distributed resulting key ranges across the server
- Pro: efficient scans
- Cons: require additional indexing

The `Fruit` relation example

- `Fruit(name, color)`
- Each range contains a contiguous segment of the key space: $[\emptyset, \text{lemon}[$; $[\text{lemon}, \text{orange}[$; $[\text{orange}, \infty[$
- Indexing structure locate ranges: $ip(range1)$ is $[\emptyset, \text{lemon}[$, $ip(range2)$ is $[\text{lemon}, \text{orange}[$, ...
- Scans `fruit.name >= "cherry" AND <= "mango"` are efficient

Scalability: Data Distribution in CockroachDB

Relies on Key Value Store with *order-preserving* distribution function

- Divided sorted key space up into ranges of nearly equal size
- Distributed resulting key ranges across the server
- Pro: efficient scans
- Cons: require additional indexing

The `Fruit` relation example

- `Fruit(name, color)`
- Each range contains a contiguous segment of the key space: $[\emptyset, \text{lemon}[$; $[\text{lemon}, \text{orange}[$; $[\text{orange}, \infty[$
- Indexing structure locate ranges: $ip(range1)$ is $[\emptyset, \text{lemon}[$, $ip(range2)$ is $[\text{lemon}, \text{orange}[$, ...
- Scans `fruit.name >= "cherry" AND <= "mango"` are efficient

CockroachDB offers scalability and keeps efficient scans

Survivability: Data Replication in CockroachDB

Use a distributed consensus algorithm (Raft)

- Set of replicas that elects a leader (other are followers)
- Leader is responsible for log replication to the followers

In the context of CockroachDB

- Logs are database queries
- Raft instance replicates range to n replicas (often $n = 3$ or $n = 5$)
- Commit happens when a majority have written data to disk

Survivability: Data Replication in CockroachDB

Use a distributed consensus algorithm (Raft)

- Set of replicas that elects a leader (other are followers)
- Leader is responsible for log replication to the followers

In the context of CockroachDB

- Logs are database queries
- Raft instance replicates range to n replicas (often $n = 3$ or $n = 5$)
- Commit happens when a majority have written data to disk
    - **Good for WAN deployment**

Consistency: Transaction in CockroachDB

Implements ACID properties over the Raft algorithm

- *Atomicity*:
  - If one part of the transaction fails, then the entire transactions fails.
  - Implemented with two phase commit (2PC) that leverage consensus algorithm: "A `COMMIT` log write to consensus system marks the transaction as committed"
- *Isolation*:
  - Concurrent transactions don't interfer with each other.
  - Implemented with Multi-Version Concurrency Control that snapshots tuples.
- *Consistency* and *Durability* are orthogonal to massively distributed systems.

OpenStack with CockroachDB

- Replace MariaDB by CockroachDB
  - Get many OpenStack instances that collaborate as a unique one *for free*
- PoC: Keystone runs over CockroachDB
  - Few modifications of OpenStack `oslo.db` (only 5 lines!)
  - Deployment of other services is not supported (database migration)
- Global database is not always the *desiderata*
  - e. g. Doesn't make sens for Keystone Service Provider

Thank you
Questions?