Lecture 5
Wednesday, 16 October 2024 8:08 PM

Kubernetes Architecture.

It consists of multiple worker nodes; it could be physical or virtual. Each Physical node has its own disks (SSD or HDD), network, DRAM. Each node may host multiple pods & containers.

FoundationDB Deployment.

It consists of Hundreds of nodes per datacenter; with cross-Datacenter (DC) latency of around 20-30 milliseconds & inner-DC latency of 0.2 milliseconds. If one DC fails, DCs from other regions are active. Each DC can tolerate 2 storage node failures.

It is able to manage transactions with multi version concurrency control & optimistic concurrent control, layers can be built on top to support other data models;

It follows 'Divide-and-conquer' approach, so each component is like a model that handle a specific task.

It consists of:

1. Control plane
- Co-ordinators that form a PAXOS group to reach consensus on some decision.
- Singletons meaning that there is only one process across all of the MDB nodes.
2. Data Plane
- Transaction System that assigns read & commit versions to transactions.
- Log Systems that store transaction logs.
- Storage System: Read the transaction logs.

The system is designed to handle failure, whenever encountered with a failure, a reconfiguration process brings the transaction management system to a new configuration. It has a Sequencer process monitoring the health of proxies, Resolvers & logServers. If any one of monitored processes fails or the database configuration changes the sequencer process terminates.

FDB Transactions

The client sends a request to a ProxyServer asking for a read version, here the proxy server gets the read version from the sequencer. The client issues reads directly to the StorageServers. During write there is Buffer locally at the client, The client sends read & write sets to a proxy Server. The Proxy asks the sequencer for a commit version. The proxy will asks Resolver to check for read-write conflicts. The proxy sends committed mutations to log servers for persistence. The proxy reports the committed version to the sequencer before return success to client.

FDB Resolvers Conflict checking.

When a transaction commits, it includes read & write sets which track the keys it has read from & written to. The resolvers job is to detect conflicts between transactions be comparing their read & write sets. If 2 transactions attempt to modify overlapping keys, or if one reads a key that another transaction writes, the resolver flags a conflict. The conflicting transaction will be aborted to maintain serializability.

FDB logging Protocol

Once the conflict has been resolved, the next step is that the proxy would try to shift all of the mutation log to the designated log servers. The proxy uses an in-memory shard map to determine the responsible StorageServers for the key range & attack storageServers tags to the mutation. log message headers contain sequence numbers & metadata for tracking. After all replica logServers confirms durability, the proxy updates the Known committed Version (KCV) with the latest log sequence Number, guaranteeing that the data is consistent & durable across the systems.

FDB Storage Server.

- Maintains versions of data in the last 5 seconds in memory.
- An older version is stored persistently on kv engine (SQLite).

How FDB storage Server serves a Read Request?

FDB storage server handles a read request by checking the kcv & last applied lsn for the key requested. If the request version is less that the known commit version, the server returns the value. However, if the request version is greater then the transaction may not have fully committed yet. The server will attempt to serve the most recent committed version yet, it either waits for the logs to be applied or directs the client to try another server storing the key.

1. What does commit do in a read-only transaction (no write)?

A. It will skip the commit protocol because read can be serialized before any write that happen in concurrent.

2. Is it possible to have multiple resolvers resolve conflicts for a transaction?

A. When the transaction provide with a set of range, it partioned into different resolver to resolve conflict; ∴ Yes, it is possible.

3. What if a StorageServer is lagging behind on applying the redo logs & a client requests a version of a key pair it does not have?

A. It will wait or the client will try another storage server before giving up.

Data model built on FDB.

- Graph

It's elements include vertices (nodes) & edges. Each vertex/edge has one label & zero, one or more properties. A edge connects 2 vertices.

Janus Graph

It is designed to manage large numbers of nodes. It is open source & uses TinkerPop, a graph computing framework with Gremlin query language.

Graph operations.

- Add a vertex.

```
g.addV("airport")
  .property("city","los angeles")
  .property("code","LAX")
g.addV("airport")
  .property("city","Sydney")
  .property("code","SYD")
```

- Add an edge

```
g.addE("route")
 .property("dist","11235")
 .from(V(1)).to(V(2))
```

Add 2 keys; one represents source vertex & other destination vertex

- Get all outgoing edges of a vertex.

```
g.V(1).outE("route").valueMap()
```

- Delete a vertex.

```
g.V(1).drop()
```

key	
1	city
1	code
1	_label
3	city
3	code
3	_label

value
los Angeles
LAX
90
Sydney
SYD
90

Janus Graph Indices

Composite Index Example.

It constructs a key based on the label & the property & the value is the vertex id, so inorder to get vertex id for any vertex; it just needs one look up.

Vertex-Centric Index Example.

This is to deal with the super node issue, where we can have any vertex with thousands of vertice that connect to it.

Mixed indices.

Are external indices, to support global range queries & text search.

FoundationDB is a distributed key-value store that is horizontally scalable & strong consistency.