Two Years with Vitess:
How JD runs the biggest Vitess cluster in the world

Speaker: Haihua Xu and Jinke Xie
Why Vitess
Vitess is a database clustering system for horizontal scaling of MySQL.
Advantages of Vitess

- **MySQL based**
  - highly reliable

- **MySQL protocol support**
  - easy to migrate from MySQL

- **Stream query**
  - Stream data to big data platform

- **Resharding**
  - scale data as needed

- **Two Phase Commit**
  - atomic commits for distributed txn

- **Global Secondary Index**
  - avoid reading amplification
How we run Vitess at JD
JED Architecture
Key systems

- **JTransfer**: data migrating
- **Mole**: k8s scheduling and Vitess Integration
- **Orchestrator**: deadmaster switching
- **Backup System**: binlog based real time backup
How we migrate the app
How we do failover

- One master with two or three replications
- One or two semi-synchronous replications
- One asynchronous replication
How we apply Mysql Instances

1. **Apply**
   R&D colleagues apply to create a new keyspace, given shard number, resources volume

2. **Approval**
   PM approval, Passing or reject

3. **Approval**
   DBA approval, check whether the applyment is legal

4. **Create**
   Create a new database online (Mole will schedule through k8s and create it)
How we Apply a MySQL table

1. Apply
   R&D colleagues apply to create a new table, submit a statement in the system

2. Approval
   PM approval, Passing or reject

3. Approval
   DBA approval, check whether the statement is in compliance with the specification

4. Create
   Create a new table online (may be occur an Syntax error)
Issues may arise

- Syntax Error
- No primary key
- Table already exists
- Primary key not auto Inc
- Deprecated character
- Deprecated storage engine
Start new app vtcheck

Vtcheck will check if the table conforms to the database specification

```go
package main

import {
    "context"
    "fmt"
    "vitess.io/vitess/go/mysql"
    "vitess.io/vitess/go/vt/sqlparser"
}

func main() {
    stmt, err := sqlparser.Parse(sql: "create table t(id int(20), name varchar(20))")
    if err != nil {
        return
    }
    ddl := stmt.(*sqlparser.DDL)
    name := ddl.Table
    cols := ddl.TableSpec.Columns
    idxs := ddl.TableSpec.Indexes

    conn, _ := mysql.Connect(context.Background(), params: nil)

    checkTable(cols, idxs, t)
}
```
Start new app DSLFormatter

Formatter convert a ast to ElasticSearch DSL

```go
// Formatter translate sql to dsl.
func Formatter(buf *sqlparser.TrackedBuffer, node sqlparser.SQLNode) {
    if buf.HasTrackError() {
        return
    }

    switch node := node.(type) {
    case *sqlparser.ComparisonExpr:
        buf.InCmpExpr = true
        defer func() {
            buf.InCmpExpr = false
        }()
        switch node.Operator {
        case sqlparser.EqualStr:
            buf.Myprintf(fmt.Sprintf(`{%s : %s}`, `term`, `value`), node.Left, node.Right)
        case sqlparser.NotEqualStr:
            buf.Myprintf(fmt.Sprintf(`{%s : %s}`, `must_not`, `term`), node.Left, node.Right)
        case sqlparser.GreaterThanStr:
            buf.Myprintf(fmt.Sprintf(`{%s : %s}`, `term`, `value`), node.Left, node.Right)
        case sqlparser.GreaterEqualStr:
            buf.Myprintf(fmt.Sprintf(`{%s : %s}`, `term`, `value`), node.Left, node.Right)
        case sqlparser.LessThanStr:
            buf.Myprintf(fmt.Sprintf(`{%s : %s}`, `term`, `value`), node.Left, node.Right)
        case sqlparser.LessEqualStr:
            buf.Myprintf(fmt.Sprintf(`{%s : %s}`, `term`, `value`), node.Left, node.Right)
        case sqlparser.InStr:
            buf.Myprintf(fmt.Sprintf(`{%s : %s}`, `terms`, `values`), node.Left, node.Right)
        case sqlparser.NotInStr:
            buf.Myprintf(fmt.Sprintf(`{%s : %s}`, `term`, `value`), node.Left, node.Right)
        case sqlparser.LikeStr:
            buf.Myprintf(fmt.Sprintf(`{%s : %s}`, `term`, `value`), node.Left, node.Right)
        case sqlparser.NotLikeStr:
            buf.Myprintf(fmt.Sprintf(`{%s : %s}`, `term`, `value`), node.Left, node.Right)
        default:
            buf.SetTrackError(fmt.Errorf(`unsupported compare operator %s`, node.Operator))
        }
    return
    case sqlparser.BoolVal:
```
Start new app DSLFormatter

DSLFormatter convert a sql to ElasticSearch DSL

```java
{
    input: 'select * from t where id = 1',
    output: '{"query":{"term":{"id":1}},"from":0,"size":10000}',
},
{
    input: 'select * from t where id != 1',
    output: '{"query":{"bool":{"must_not": [{"term":{"id":1}}]}},"from":0,"size":10000}',
},
{
    input: 'select * from t where id > 1',
    output: '{"query":{"range":{"id":{"gt":1}}},"from":0,"size":10000}',
},
{
    input: 'select * from t where id >= 1',
    output: '{"query":{"range":{"id":{"gte":1}}},"from":0,"size":10000}',
},
{
    input: 'select * from t where id = 1',
    output: '{"query":{"range":{"id":{"lte":1}}},"from":0,"size":10000}',
},
{
    input: 'select * from t where id in (1,2,3)',
    output: '{"query":{"terms":{"id":[1,2,3]}},"from":0,"size":10000}',
},
{
    input: 'select * from t where name not in ("aa","bb","cc")',
    output: '{"query":{"bool":{"must_not": [{"terms":{"name": ["aa","bb","cc"]}}]}}},"from":0,"size":10000}',
},
{
    input: 'select * from t where id > 1 order by id desc',
    output: '{"query":{"range":{"id":{"gt":1}}},"sort":{"id":"desc"},"from":0,"size":10000}',
},
{
    input: 'select * from t where id > 1 order by desc',
    output: '{"query":{"range":{"id":{"gt":1}}},"sort":{"id":"desc"},"from":0,"size":10000}',
},
{
    input: 'select city, city, avg(age) from space3 group by city',
    output: '{"query":{"bool":{"must": [{"match_all": {}}]}},"aggs":{"city":{"terms":{"field":"city"}},"aggs":{"avg(age)":"s
```
Problems and Solutions
Challenges encountered

01 Vtgate cluster upgrading
   when we get a vtgate cluster with more than 1000 instances, how do we upgrade it?

02 Various demands
   Complex Aggregate statement
   Load statement
   Prepare protocol

03 Apps interaction
   Apps wants Read and write separation
   Apps wants to extract data from mysql to big data platform
   0 level app wants the highest priority

04 Etcdf problems
   Some cell local etcd OOM weird
   Local Vschema datasize growing as apps count increase
Solutions

ETCD

- Etdc client watcher leak fixed
- Migrate local etcd big value to Redis

Upgrade vtgate

- Gray-Release based on k8s scheduling

Apps interaction

- Different access role works in different instance
- Exclusive Extract data Vtgate cluster each cell
- Exclusive Cluster for 0 level app if necessary

Various demands

- Custom development base on Vitess
Future plan

Work plan for next year
Future Plan

Monitor everything

Convenient resharding

Intelligent scheduling

Migrating from 2.0 to 3.0
Future Plan

Monitor Everything
Including vgate vttablet mysql etc
Build our own monitoring system if necessary

Intelligent scheduling
Based on our Machine learning Algorithms
Dynamic expanse or reduce container resource

Convenient Reshard
Support a more interactive UI for DBA to reshard within minimum steps

Migrating from 2.0 to 3.0
We have been merged 2.0 to 3.0 and test it for some time.
New apps apply to 3.0 already.
It is time to make a plan to migrate apps from 2.0 to 3.0 gradually.
THANK YOU