ACMUG 中国MySQL用户组 | IT大咖说 知识共享平台

When TiDB Meets Spark

=> TiSpark

Who am I

- 马晓宇@PingCAP
- Architect@TiDB team
- Working on OLAP related products and features
- Previously lead of Big Data infra team@Netease
- Mainly working on SQL on Hadoop and BigData related stuff

Agenda

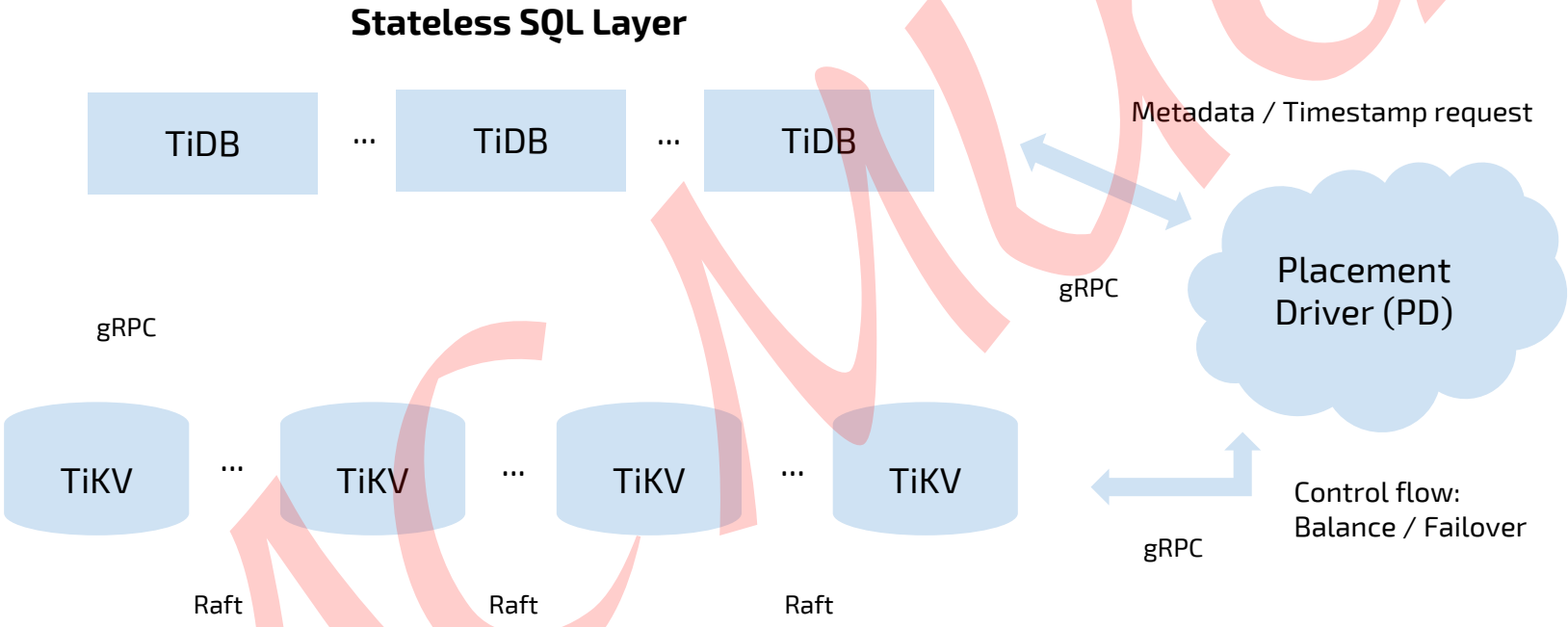
- A little bit about TiDB / TiKV
- What is TiSpark
- Architecture
- Features beyond raw Spark
- Use case
- Current Status

What's TiDB

- Horizontal Scalability
- ACID Transaction
- High Availability
- Auto-Failover
- SQL at scale
- TiKV is its storage engine

[What's Really New with NewSQL?]

A little bit about TiDB and TiKV



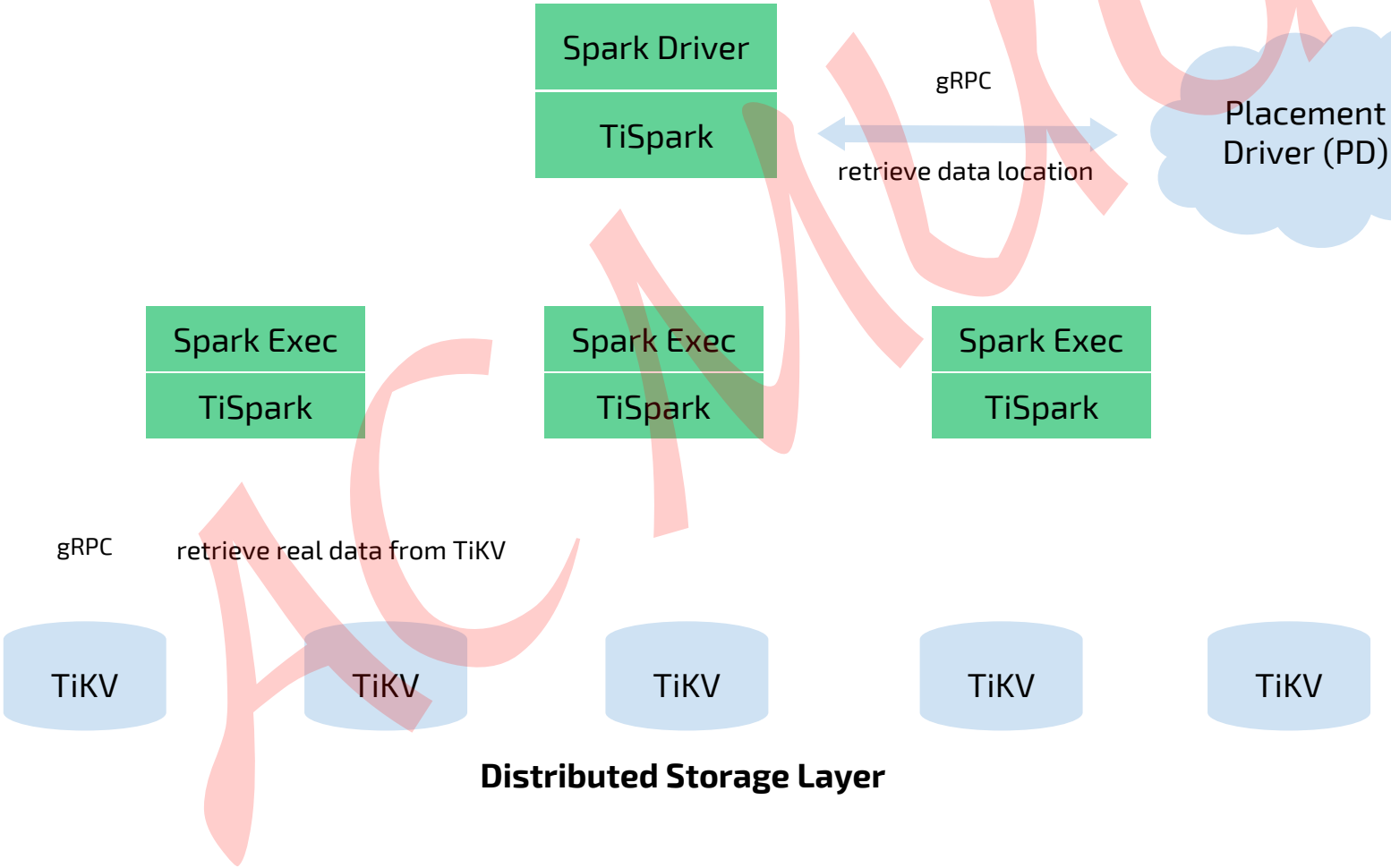
What is TiSpark

- TiSpark = Spark SQL on TiKV
  - Spark SQL directly on top of a distributed Database Storage
- Hybrid Transactional/Analytical Processing(HTAP) rocks
  - Provide strong OLAP capacity together with TiDB

What is TiSpark

- Complex Calculation Pushdown
- Key Range pruning
- Index support
  - Clustering index / Secondary index
  - Covering Index Support
- Cost Based Optimization
  - Histogram
  - Pick up right Access Path
- Batch write (2018 Q2)

Architecture



Architecture

- On Spark Driver
  - Translate metadata from TiDB into Spark meta info
  - Hijack Spark SQL logical plan, pick up elements to be leverage by storage (TiKV) and rewrite plan
  - Do extra optimization based on extra information (basically stats, maybe order)
  - Locate Data based on Region info from Placement Driver and split partitions;
- On Spark Executor
  - Encode Spark SQL plan into TiKV's coprocessor request
  - Decode TiKV / Coprocessor result and transform result into Spark SQL Rows

How everything made possible

```
public class ExperimentalMethods
extends Object
```

Holder for experimental methods for the bravest. We make NO guarantee about the stability regarding binary compatibility and source compatibility of methods here.

```
spark.experimental.extraStrategies += ...
```

Since:

1.3.0

Method Summary

Methods

Modifier and Type	Method and Description
`scala.collection.Seq<org.apache.spark.sql.catalyst.rules.Rule<org.apache.spark.sql.catalyst.plans.logical.LogicalPlan>>`	`extraOptimizations()`
`scala.collection.Seq<org.apache.spark.sql.execution.SparkStrategy>`	`extraStrategies()` Allows extra strategies to be injected into the query planner at runtime.

- Two extension points for Spark SQL Internal
- Extra Optimizer Rules allows us to do logical plan transform like Join Reorder
- Extra Strategies allow us to inject our own physical executor and that's what we leveraged for phase 1 in TiSpark
- Kept Spark Internal untouched to avoid compatibility issue

How everything made possible

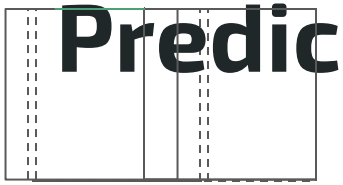
- A fat java client module, paying the price of bypassing TiDB
  - Parsing Schema, Type system, encoding / decoding, coprocessor
  - A full featured TiKV client (without write-support for now)
  - Predicates / Index - Key Range related logic
  - Aggregates pushdown related
  - Limit, Order, Stats related
- A thin layer inside Spark SQL
  - TiStrategy for Spark SQL plan transformation
  - And other utilities for mapping things from Spark SQL to TiKV client library
  - Thin enough for not bothering much of compatibility with Spark SQL
- Might be totally port to new DataSource API 2 (hopefully)
  - Spark 2.3's new API might save us from all possible incompatibilities

Too Abstract? Let's get concrete

```
select class, avg(score) from student
WHERE school = 'engineering'  and lottery(name) = 'picked'
and studentId >= 8000 and studentId < 10100
 group by class ;
```

- Above is a table on TiDB named student
- Clustered index on StudentId and a secondary index on School column
- Lottery is an Spark SQL UDF which pick up a name and output 'picked' if RNG decided so

Predicates Processing



WHERE school = 'engineering' and lottery(name) = 'picked'
and studentId >= 8000 and studentId < 10100

Predicates are converted into key ranges based on indexes

StudentId >= 8000

StudentId < 10100

School = 'engineering'

Lottery(name) = 'picked'

Key Range: [8000, 10100)

school = 'engineering'

Construct Tasks

1. Append remaining predicates if supported by coprocessor
2. Push back whatever needs to be computed by Spark SQL, e.g. UDFs, prefix index predicates
3. Cut them into tasks according to Region／Range
4. Encode into coprocessor request

Spark Task 1
Region2
[8000, 10000)
COP Request

Spark Task 2
Region3
[10000, 10100)
COP Request

gRPC via Spark worker

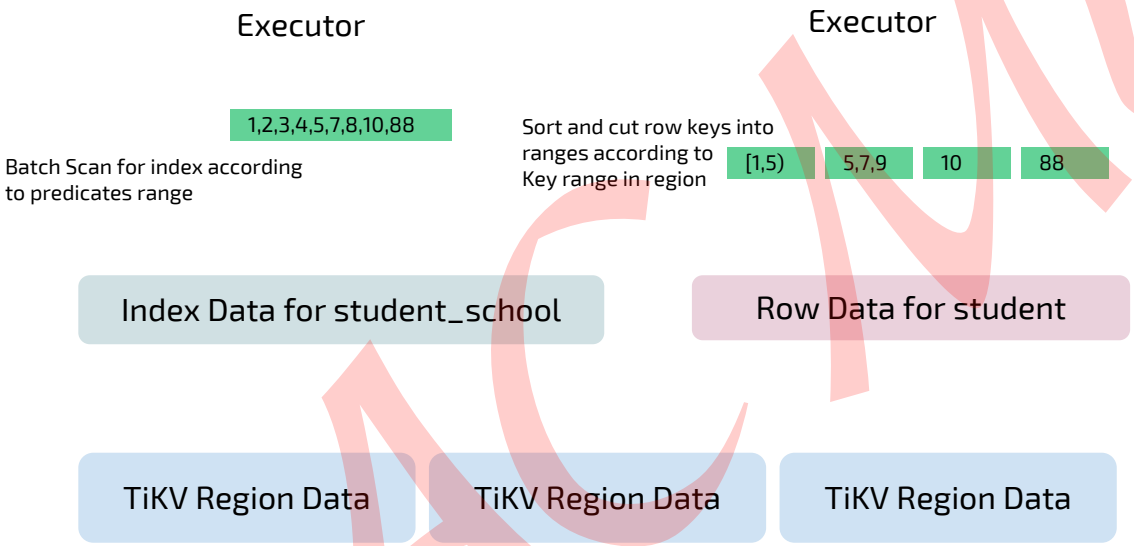
Region 1
StudentId
[0-5000)

Region 2
StudentId
[5000-10000)

Region 3
StudentId
[10000-15000)

Index Scan

WHERE school = 'engineering' and lottery(name) = 'picked'
and (studentId >= 8000 and studentId < 10100)



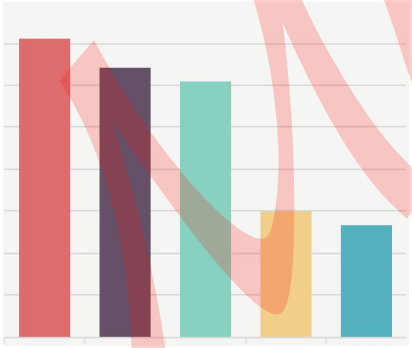
- Secondary Index is encode as key-value pair
  - Key is comparable bytes format of all index keys in defined order
  - Value is the row ID pointing to table row data
- Reading data via Secondary Index usually requires a double read.
  - First, read secondary index in range just like reading primary keys in previous slide.
  - Shuffle Row IDs according to region
  - Sort all row IDs retrieved and combine them into ranges if possible
  - Encoding row IDs into row keys for the table
  - Send those mini requests in batch concurrently
- Optimize away second read operation
  - If all required column covered by index itself already

Index Selection

WHERE school = ‘engineering’ and lottery(name) = ‘picked’ and (studentId >= 8000 and studentId < 10100) or studentId in (10323, 10327)

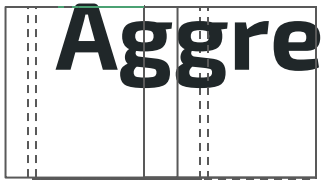
Clustered Index on StudentID + predicates related StudentId matched

Secondary Index on School + predicates related School matched



1K * Clustered Index Access Cost
<
800 * Secondary Index Access Cost

- If we the columns referred are all covered by index, then instead of retrieving actual rows, we apply index only query and cost function will be different
- If histogram not exists, TiSpark using pseudo selection logic.



Aggregates Processing

```
select class, avg(score) from student
.....
 group by class ;
```

Spark SQL plan received in TiStrategy

AVG(score) | Group BY class

AVG are rewritten into SUM and COUNT

SUM(score) / COUNT(score) | Group BY class

Reduce Task 1 | Reduce Task 2

Map Task 1 | Map Task 2

Construct Schema Transformation Rules
TiDB has totally different type system and infer rules

Spark Schema by its own type infer rules
[SUM, COUNT, class]

TiKV Schema to Spark Schema
[groupBy keys as bytes, SUM as Decimal, COUNT as BigInt]

gRPC via Spark worker

Region 1	Region 2	Region 3
StudentId	StudentId	StudentId
[5000-10000)	[5000-10000)	[10000-15000)

- After coprocessor preprocessing, TiSpark still rely on normal Spark aggregation strategy
- But if a specific key stays in a single region, we have a change to optimize aggregates away (planned)

TiSpark specific storage features

- Different Transaction Isolation Level
  - For OLTP: Snapshot Isolation
  - For OLAP: Read Committed to avoid lock

- Different resource schedule priority
  - Different Resource pool for different workload
  - User can specify different pool for OLAP workload to isolate resource usage

Features Beyond Raw Spark SQL

- What we have more than traditional SQL-On-Hadoop systems:
  - Index and global ordering
    - Point query to locate a single record in billions of rows
    - Flexible secondary Index
  - ACID transaction support
    - query in real time data without T+1 importing from other data sources
    - Unify OLAP and OLTP in one platform
    - Distributed batch query in a more consistent way

Use Case

- Analytical / Transactional support all on one platform
  - No need for ETL
  - Real-time query with Spark
  - Simplify your platform and reduce maintenance cost
- Embrace Spark eco-system
  - Support of complex transformation and analytics with Spark eco system (far more flexible than stored procedure)
  - Machine Learning Libraries
  - Spark Streaming

Currently In beta, RC towards 1.0

- Open sourced already and currently in beta
- RC1 for now and working towards v1.0
- Most of the important features discussed here done
- Evaluating and working on porting to Spark 2.3 with Data Source v2 API (if possible to cover all the futures)
- Important upcoming features are
  - Partition Table
  - Authentication
- Plan for GA release before the end of 2018 Q1
- Also, we are working on an slightly independent columnar OLAP product on top of Spark (again), theFlash

Some frequently asked questions

- Why not build on top of TiDB instead of TiKV
  - Full control of details like Range, Data locations and stats which are not exposed via SQL interface
  - no extra DB layer slowing things down
  - We need a Java client for TiKV as well for other use cases
- Why Spark SQL?
  - Spark SQL is not just SQL, it's the core of a whole eco-system
- Is UDF supported? Sure
- What Spark version supported?
  - For now it's built upon Spark 2.1 but will not be too hard to port to other versions since Spark layer is very relatively thin

Thanks!