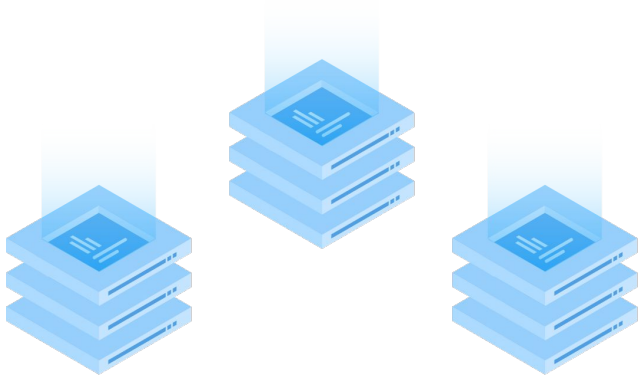
TiDB Change Data Capture

Presented by Yang Fei

PingCAP TiDB



目录

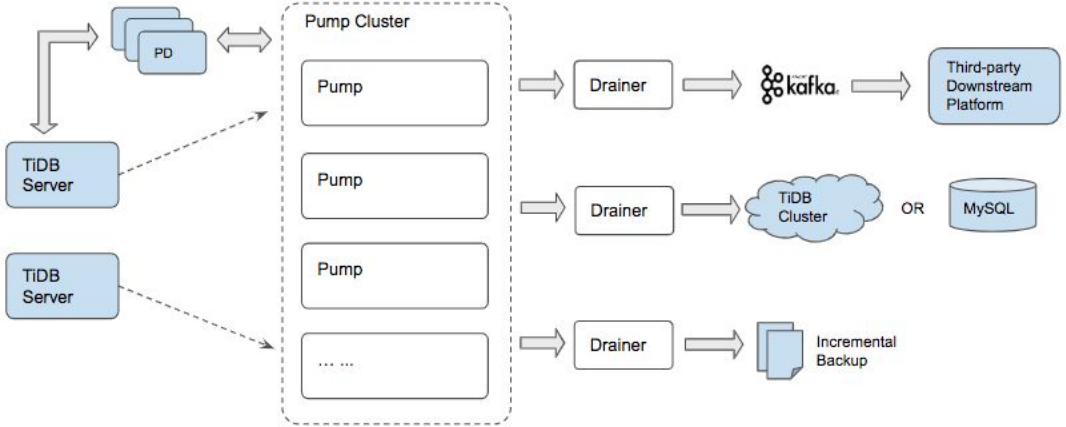
- Part 1
  - CDC 的使用场景
  - TiCDC 的设计实现
  - TiCDC 的特性
- Part 2
  - TiCDC 生态和开发指南
  - 社区嘉宾分享
    - **张亿皓**, 小红书存储与中间件研发工程师
    - **姚翔**, 海尔优家智能科技(北京)有限公司, 技术经理

CDC 的使用场景

- 定义：**C**hange **D**ata **C**apture
- 常见使用场景
  - 数据库主备
  - 环形复制与多数据中心多活
  - 数据变更订阅
- 其他数据库
  - MySQL: binlog
  - PostgreSQL: WAL plugin, streaming replication
  - Oracle: redo log based CDC
  - CockroachDB: rangefeed based CDC

TiDB 的 CDC 方案

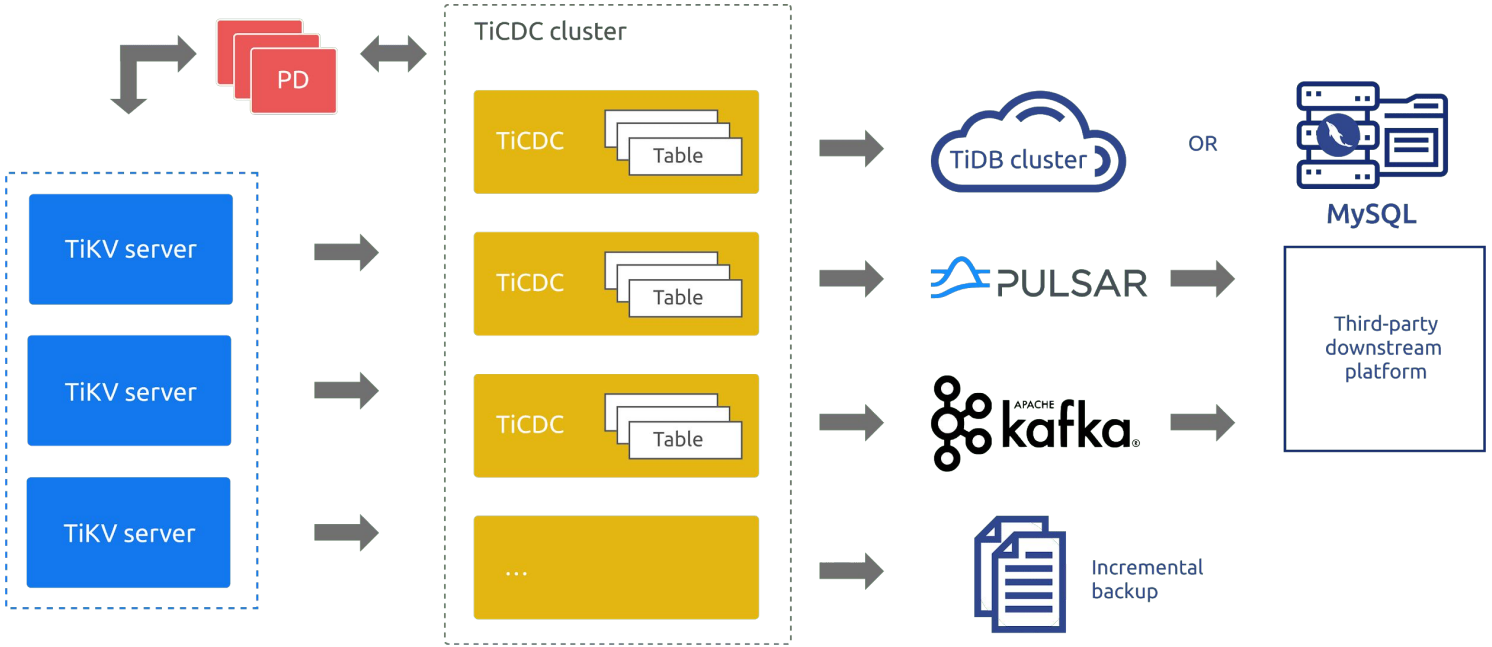
- TiDB-Binlog ?
  - ✔️ 主从复制
  - ❓ 低同步延迟
  - ❓ 多活支持
  - ❓ 数据订阅
  - ❓ 现有方案的集成
  - ❌ 高可用
  - ❌ 行级别变更订阅
  - ❌ TiDB 集群的水平扩展



TiCDC requirements from sketch

- 实时的变更数据流
  - INSERT INTO users (1, “Alice”), (2, “Bob”);
  - Get {“id”: 1, “name”: “Alice”}, {“id”: 2, “name”: “Bob”}
- 行变更的顺序
  - 单表
  - 跨表
- 事务一致性
  - 单表事务
  - 跨表事务
- 支持水平扩展

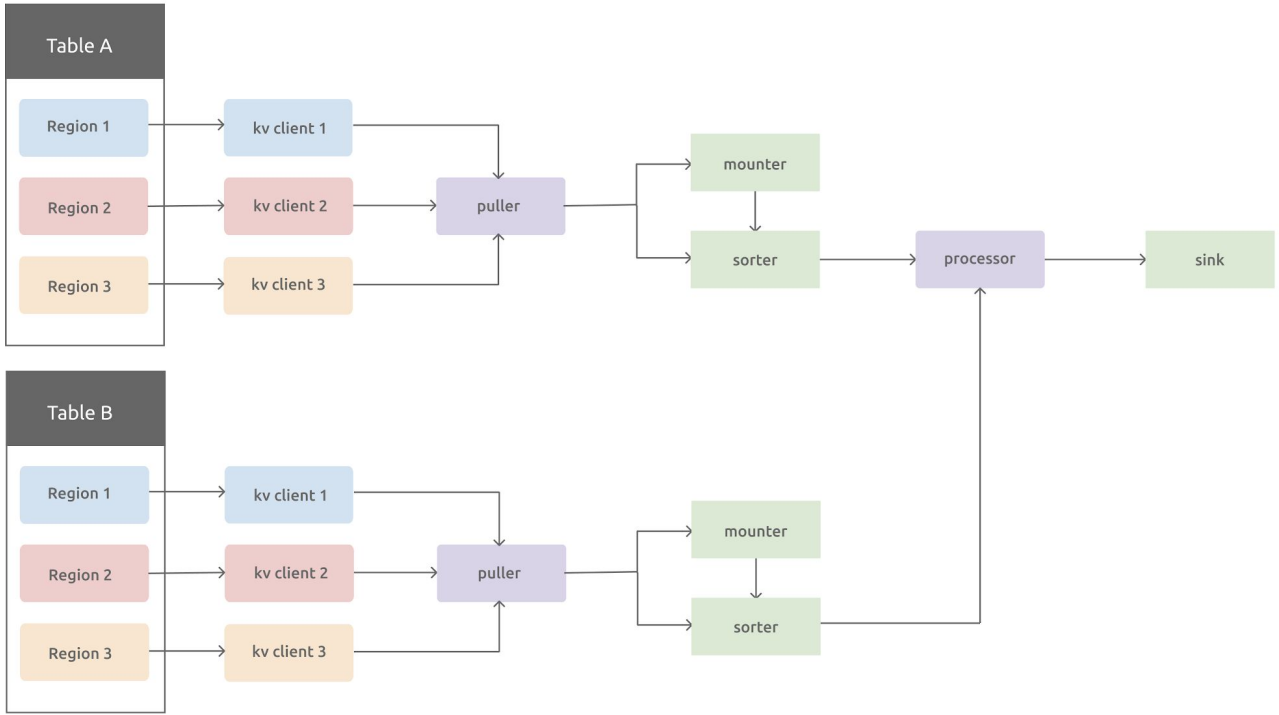
TiCDC 架构


PD
TiCDC cluster
TiKV server
TiKV server
TiKV server
TiCDC
Table
TiCDC
Table
TiCDC
Table
...
TiDB cluster
OR
MySQL
PULSAR
Third-party downstream platform
APACHE kafka
Incremental backup

TiCDC 实现原理

- TiKV row kv changed log
  - gRPC 流式 API: EventFeed
  - 在 Raft KV 层数据提交到 RocksDB 之前发送 CDC 数据
  - 每个流按照 TiKV Region 发送数据，region leader
- Capture
  - CDC 运行的进程，可以具有 owner 和 processor 两种角色
    - owner 负责集群调度和 DDL 的同步
    - processor 负责 DML kv changed log 的拉取，排序，还原和分发

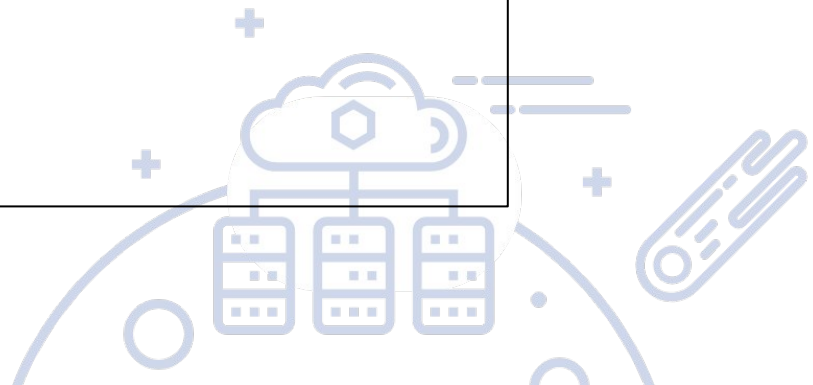
Data pipeline in TiCDC



TiCDC 数据排序

- 全局排序
  - Region 内 KV change log 按照提交顺序发送
  - TiKV 提供 ResolvedTs < ∀(unapplied CommitTs) 机制，在 region 没有数据写入时可以发送 ResolvedTs
  - CDC 内部机制推进全局的 ResolvedTs，每个 processor 内部可以将 Global ResolvedTs 前的数据分发到下游

```
Global ResolvedTs = min (
   min (resolvedTs of all table puller),
   resolvedTs of ddl puller,
   targetTs,
)
```



TiCDC Resolved Ts

- 论文背景 : Naiad: A Timely Dataflow System

Realtime	Region 1 Stream	Region 2 Stream
		ts1: C -> 2
	ts2: A -> 6	ts1: Resolved
X	ts1: B -> 4	
Y	ts1: Resolved	
	ts2: B -> 3	
	ts2: Resolved	ts3: C -> 10
Z		ts3: Resolved

TiCDC 的集群调度和高可用

- Owner 选举
  - capture 节点启动时在 PD 自注册
  - 通过 PD 内置 etcd 的 Election 进行 owner 选举
- 集群调度策略
  - 读写 PD 内置 etcd 的声明式调度
- 高可用方案
  - 元数据存储在 PD 内置的 etcd，元数据具有高可用
  - 数据源来自 TiKV，也具有高可用的能力
  - 同步任务与进程无状态耦合，任意 capture 进程级别的故障可以通过 etcd 中的元数据恢复同步任务

What’s new in TiCDC 4.0GA

- **Old Value 特性**
- **环形同步**
- K8s support
- 更多的生态
  - Sink type
    - Kafka
    - Pulsar
  - Protocol
    - Canal
    - Avro
    - Maxwell

4.0GA 新特性: Old Value

- INSERT INTO users (1, “Alice”), (2, “Bob”);
- DELETE FROM users where id = 1;
- With old value

```
{
  "d":{
    "id":{
      "t":3,  "f":10,"v":1
    },
    "name":{
      "t":15,"f":0,"v":"Alice"
    }
  }
}
```

- Without old value

```
{
  "d":{
    "id":{
      "t":3,  "f":10,"v":1
    }
  }
}
```

4.0GA 新特性: Old Value

- INSERT INTO users (1, “Alice”), (2, “Bob”);
- UPDATE users SET name = “Carol” where id = 2;
- With old value

```
{
  "u":{
    "id":{"t":3,  "f":10,  "v":2},
    "name":{ "t":15,  "f":0, "v":"Carol"}
  },
  "p":{
    "id":{ "t":3, "f":10, "v":2},
    "name":{"t":15, "f":0, "v":"Bob"}
  }
}
```

- Without old value

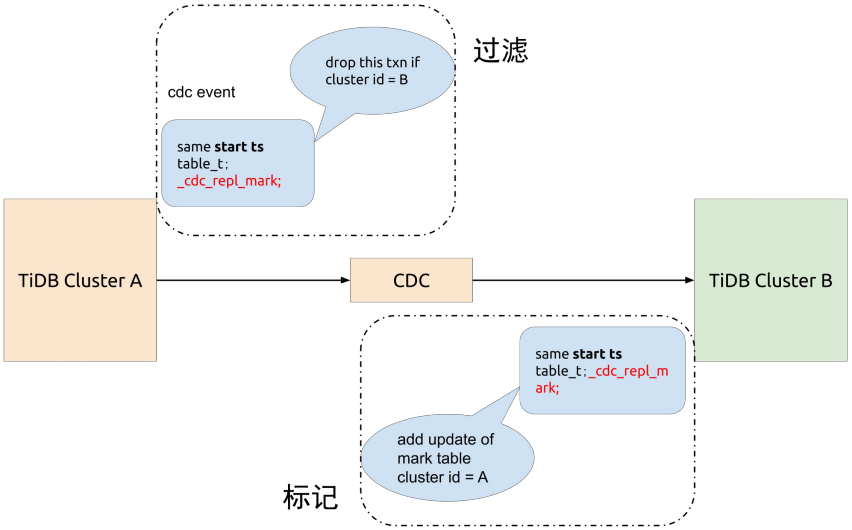
```
{
  "u":{
    "id":{"t":3,  "f":10,  "v":2},
    "name":{ "t":15,  "f":0, "v":"Carol"}
  }
}
```

4.0GA 新特性: 环形同步

- 环形同步的定义
  - 2个或多个 TiDB 集群
  - 对相同的库、表在多个集群同时写入
  - 使用 CDC 在多个集群间同步，需要过滤成环的流量
  - 多个集群的最终一致性

4.0GA 新特性: 环形同步

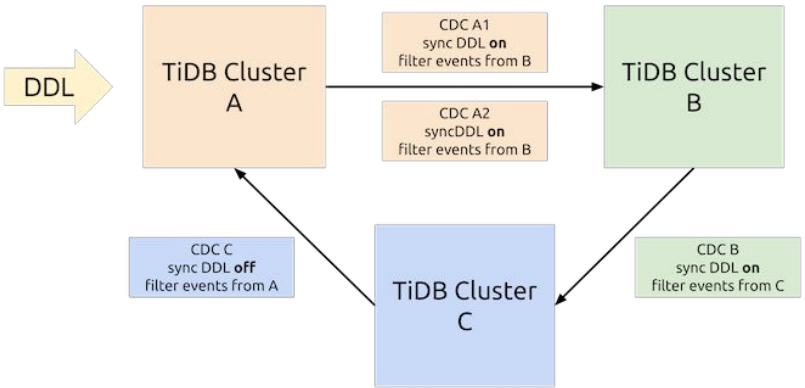
- DML 标记表和过滤



Same for B -> A

4.0GA 新特性: 环形同步

- DDL 的处理



TiCDC 展望

- MySQL/TiDB sink 的强一致性复制
- Point in Time Recovery
- Sink Plugin
- Opeartor and SaaS integration
- SQL management

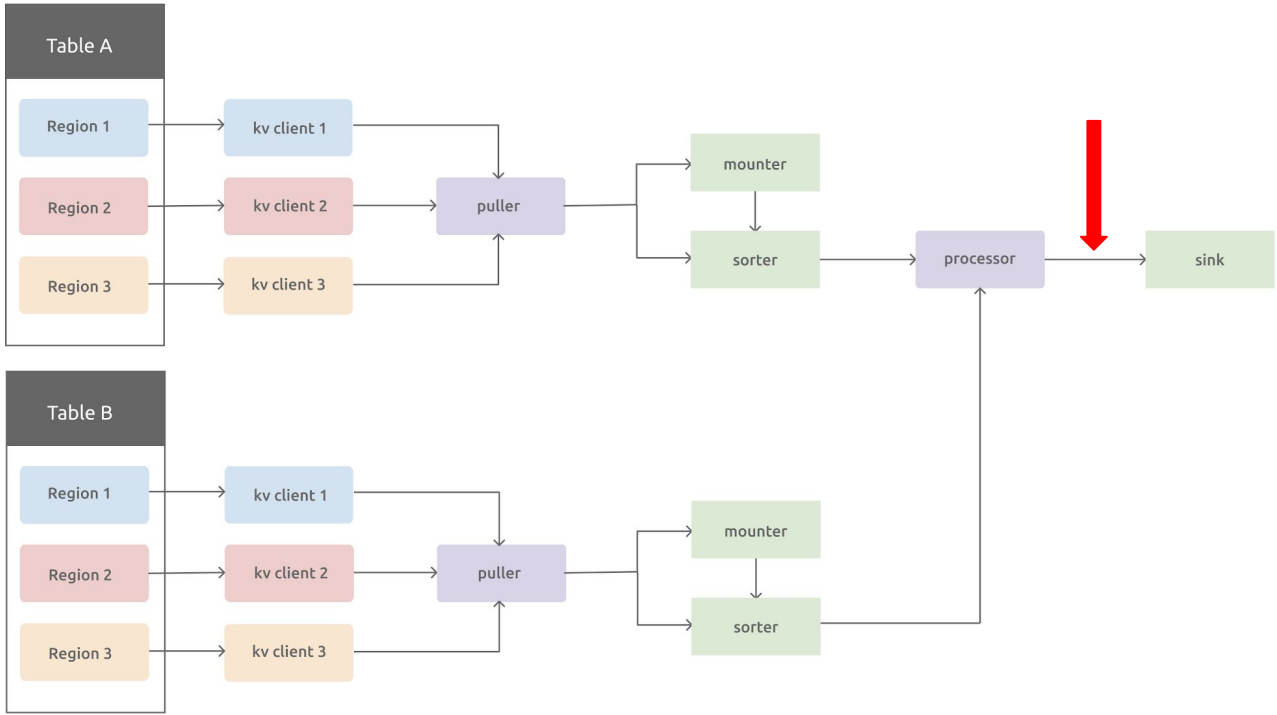
PingCAP

Part 2: TiCDC 生态

TiCDC 生态定义

- Sink 生态
  - 提供 sink interface，可定制的 sink 类型
    - MySQL sink
    - Kafka sink
    - Pulsar sink
    - PiTR S3 sink
- 协议生态
  - TiCDC Open Protocol
  - Avro
  - MySQL binlog 生态
    - Alibaba Canal
    - Maxwell

Data pipeline in TiCDC



TiCDC sink 扩展

- Sink interface

```
type Sink interface {
        Initialize(ctx context.Context, tableInfo []*model.SimpleTableInfo) error

        EmitRowChangedEvents(ctx context.Context, rows ...*model.RowChangedEvent) error

        EmitDDLEvent(ctx context.Context, ddl *model.DDLEvent) error

        FlushRowChangedEvents(ctx context.Context, resolvedTs uint64) (uint64, error)

        EmitCheckpointTs(ctx context.Context, ts uint64) error

        Close() error
}
```

TiCDC sink 扩展

```
type Sink interface {
        // EmitRowChangedEvents 向 Sink 传递 Row Changed Event
        // TiCDC 提供保证：输出到这个函数的 row change event 满足单表有序
        // * 如果不要求还原事务，直接写下游即可
        // * 如果要求还原事务，将 rows 缓存到内存中，等待 EmitResolvedEvent
        EmitRowChangedEvents(ctx context.Context, rows ...*model.RowChangedEvent) error

        // FlushRowChangedEvents 向下游刷新缓存中的 Row
        // 当这个函数被调用时，意味着所有 Ts 小于等于 resolvedTs 的 events 都已经通过
EmitRowChangedEvents 函数发送到 Sink
        // 返回值表示已经刷新到下游的最大 ts，该函数为非阻塞式
        FlushRowChangedEvents(ctx context.Context, resolvedTs uint64) (uint64, error)

        // EmitCheckpointTs 向 Sink 传递 CheckpointTs Event
        // 意味 TiCDC 集群的所有节点都已经成功执行了相同 ts 的 resolved event
        // 即确保整个 TiCDC 集群中 Ts 小于等于 Checkpoint Ts event 的 events 都成功刷新到下游
        EmitCheckpointTs(ctx context.Context, ts uint64) error
}
```

TiCDC sink 扩展—EventBatchEncoder

- Reuse MQ sink
- maxwell 的实现

```
type EventBatchEncoder interface {
    // 将 Checkpoint event 编码后，写入 buffer
    EncodeCheckpointEvent(ts uint64) (*MQMessage, error)

    // 将 DDL event 编码后，写入 buffer
    EncodeDDLEvent(e *model.DDLEvent) (*MQMessage, error)

    // 将 RowChangedEvent event 编码后，写入 buffer
    AppendRowChangedEvent(e *model.RowChangedEvent) (EncoderResult, error)

    // 将 ResolvedTs event 编码后，写入 buffer
    AppendResolvedEvent(ts uint64) (EncoderResult, error)

    // 读取 buffer 内 batch events 编码后的 bytes，编码后的 bytes 随后将被发送到 MQ
    Build() []*MQMessage
    Size() int
    Reset()
}
```

Avro and Kafka connector

- Apache Avro:
  - http://avro.apache.org/docs/current/
- Kafka Connector:
  - https://www.confluent.io/blog/no-more-silos-how-to-integrate-your-databases-with-apache-kafka-and-cdc/
  - Confluent 认证 sink connector 101. ref: https://docs.confluent.io/current/connect/managing/connectors.html

Next

- 嘉宾分享