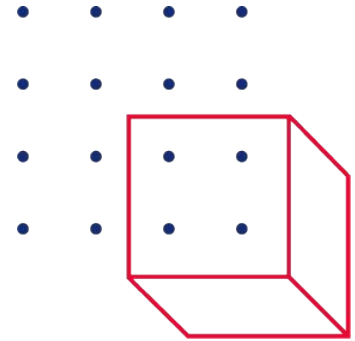
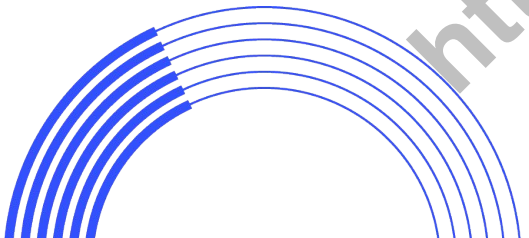


CDC: Why, How and What's next?

黄东旭



关于我

- 黄东旭
- PingCAP 联合创始人，CTO
- 分布式系统工程师
- TiDB 创始人
- huang@pingcap.com

CDC？

- 定义：**C**hange **D**ata **C**apture
- 数据存储方案越来越碎片化
- 业务需求越来越实时
- 「企业消息总线」的模式越来越流行

CDC?

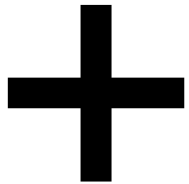
- 常见使用场景
  - 数据库主备，PITR（Point in Time Recovery）
  - 环形复制与多数据中心多活
  - 数据变更订阅到第三方系统
- 现代数据库标配
  - TiDB：TiDB-Binlog、TiCDC
  - MySQL: Binlog
  - PostgreSQL: WAL plugin, streaming replication
  - Oracle: GoldenGate (Redo Log based)
  - CockroachDB: RangeFeed

CDC 技术的几个挑战

- 实时的变更数据流（对接第三方管道，如 Kafka 等）
  - INSERT INTO users (1, “Alice”), (2, “Bob”);
  - Get {“id”: 1, “name”: “Alice”}, {“id”: 2, “name”: “Bob”}
- 保持行变更的顺序
  - 单表
  - 跨表
- 事务一致性
  - 单表事务
  - 跨表事务

CDC 技术的几个挑战

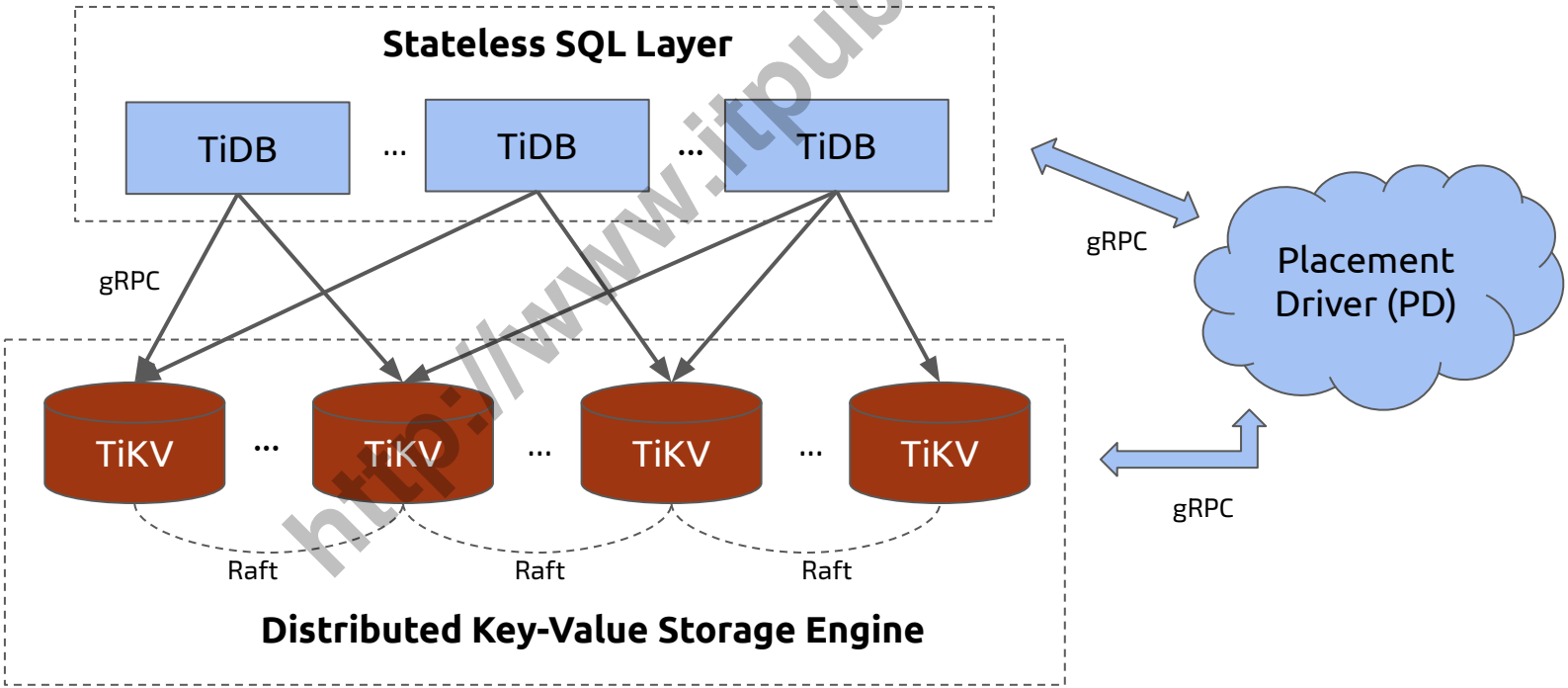
- 实时的变更数据流（对接第三方管道，如 Kafka 等）
  - INSERT INTO users (1, “Alice”), (2, “Bob”);
  - Get {“id”: 1, “name”: “Alice”}, {“id”: 2, “name”: “Bob”}
- 保持行变更的顺序
  - 单表
  - 跨表
- 事务一致性
  - 单表事务
  - 跨表事务

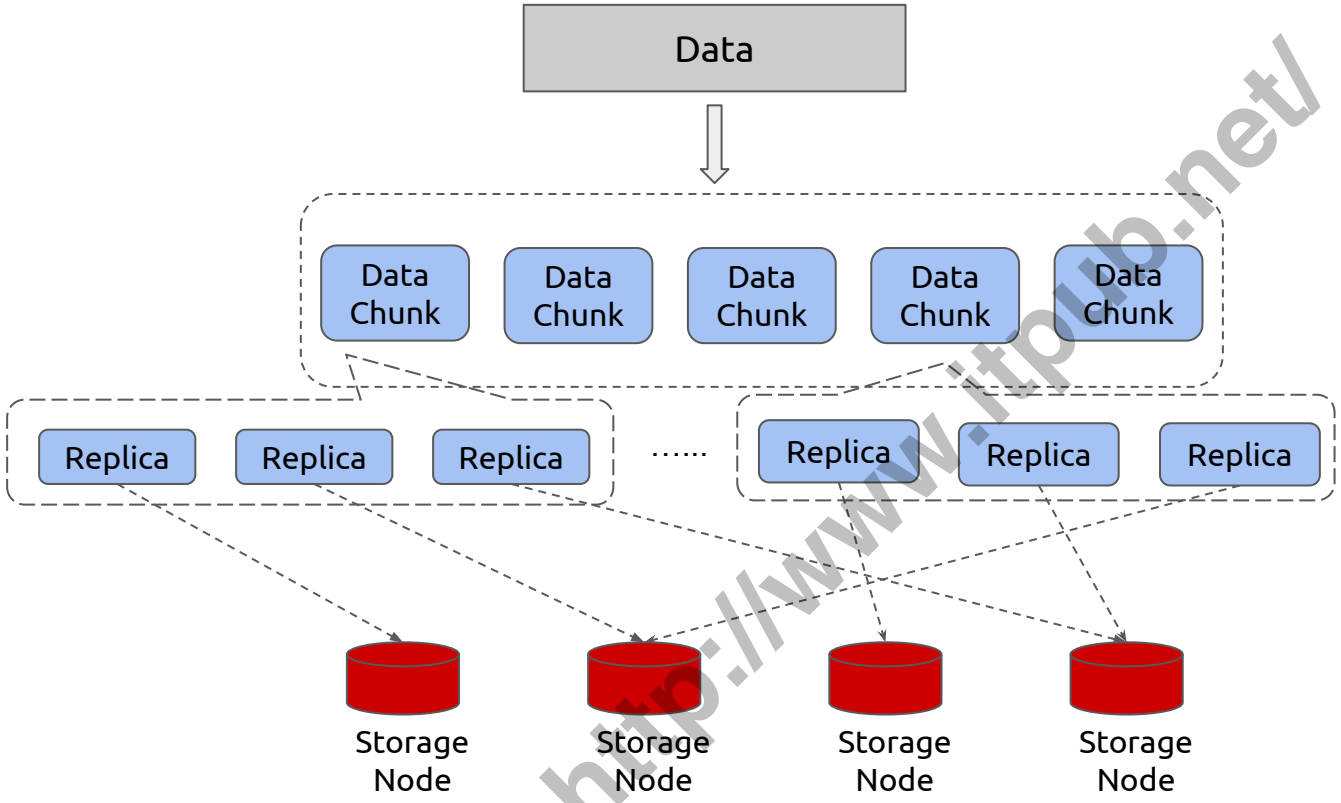


水平扩展

为什么为分布式数据库设计 CDC 是复杂的

- 以 TiDB 为例子





Split data into chunks

N-Replicas for each chunk

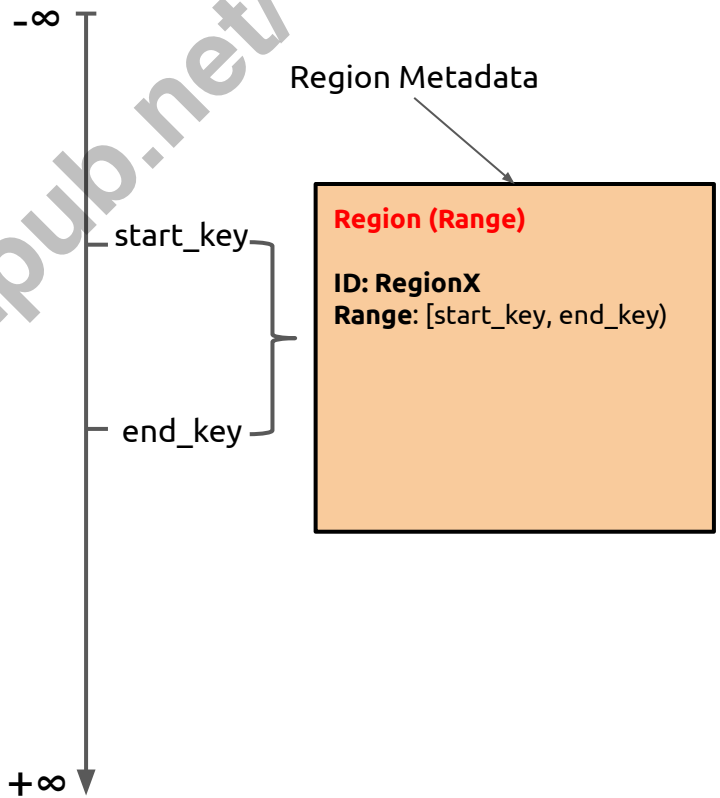
Distribute the replicas among the storage nodes

Logical View of TiKV

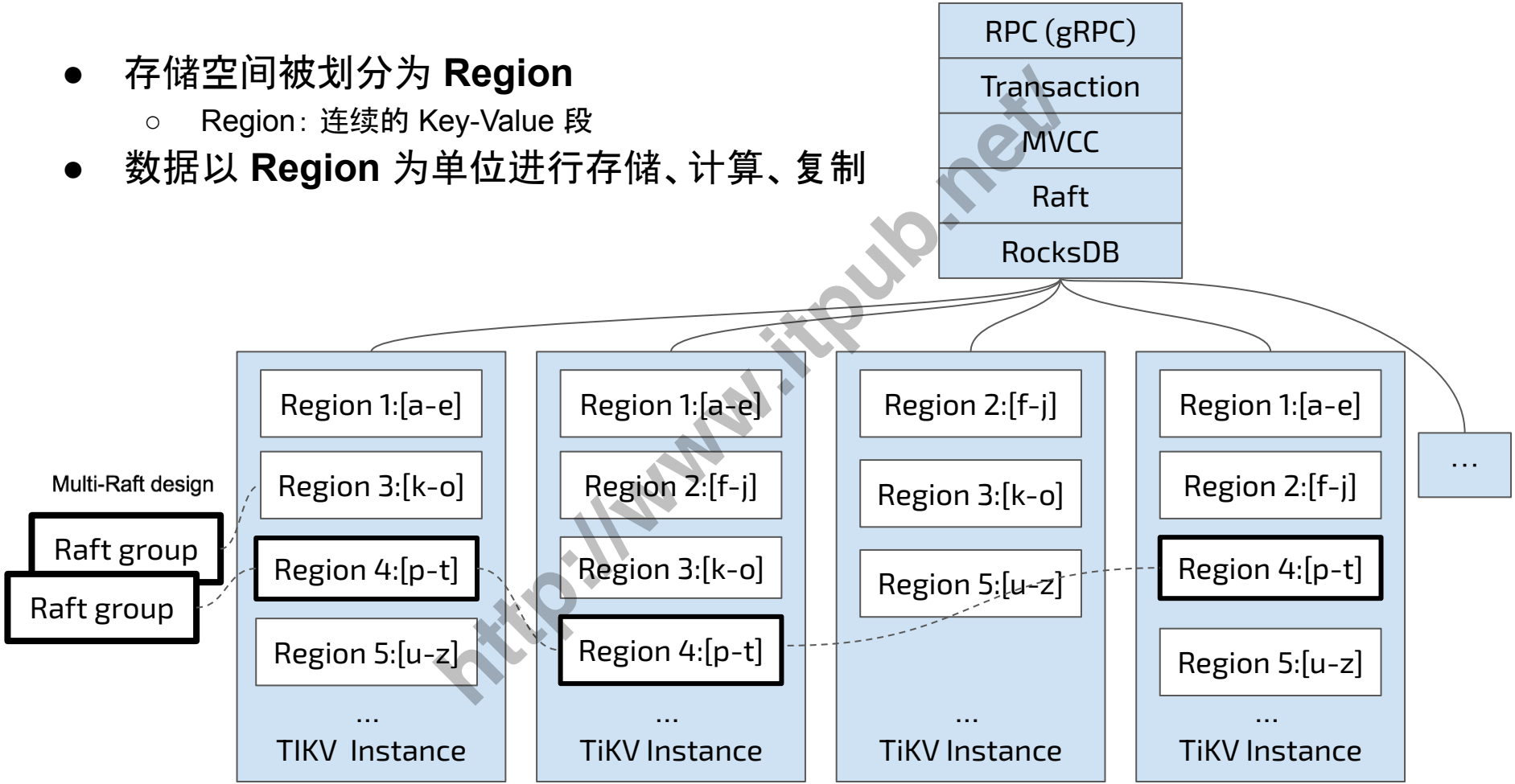
- A giant ***Map***
  - Sorted Key-Value Map
  - Both keys and values are byte arrarys
  - Keys are sorted by byte order
- Key space is divided into pieces
  - Data divided into chunks called "**Regions**"
  - Multiple regions within same node share 1 RocksDB

Terminology:

- **Region**



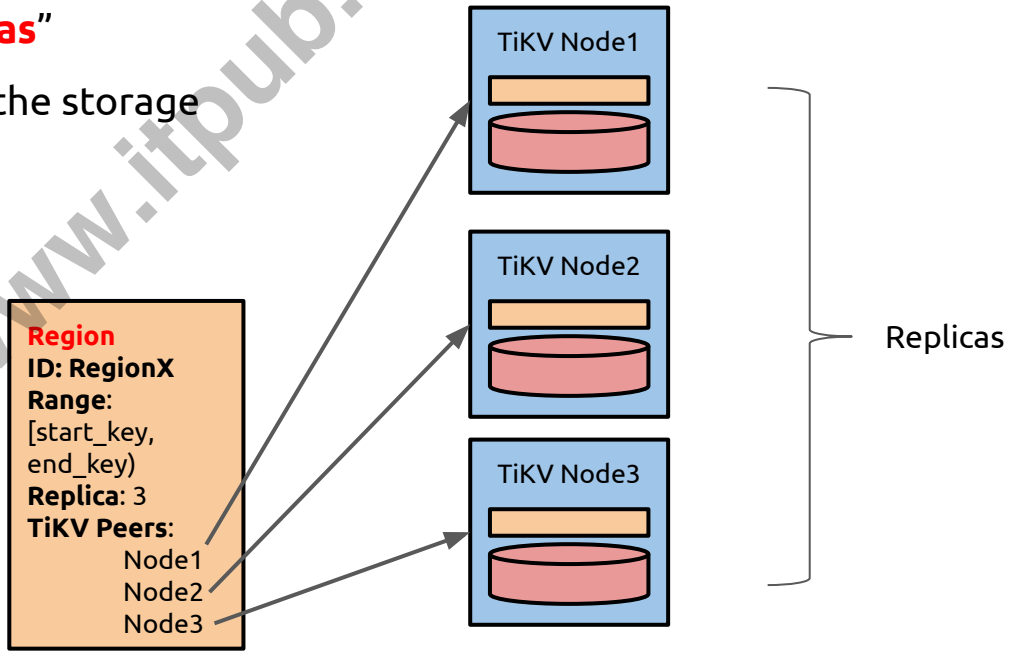
- 存储空间被划分为 **Region**
  - Region：连续的 Key-Value 段
- 数据以 **Region** 为单位进行存储、计算、复制



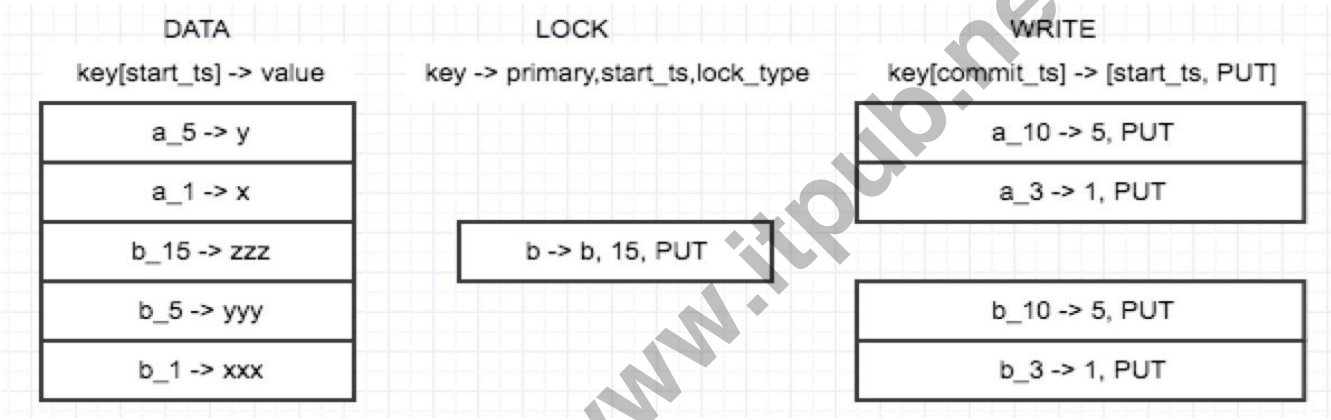
Physical View of TiKV

- A distributed storage engine
  - Each ***region*** has multiple "**replicas**"
  - Replicas are distributed among the storage instances via **Raft algorithm**

Terminology:
- **Replica**



TiKV MVCC机制

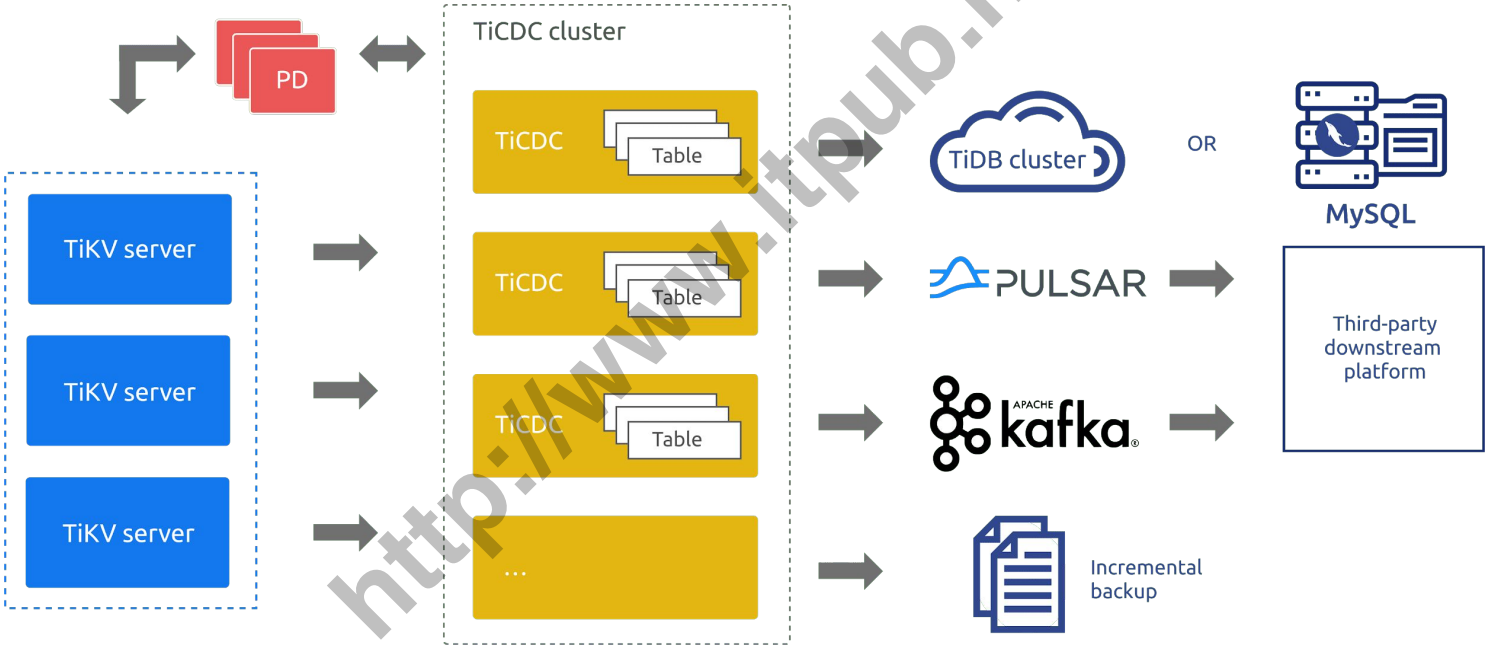


- read a ts==6 x
- read a ts==10 y
- read b ts==13 yyy
- read b ts==17 (wait)

为什么为分布式数据库设计 CDC 是复杂的

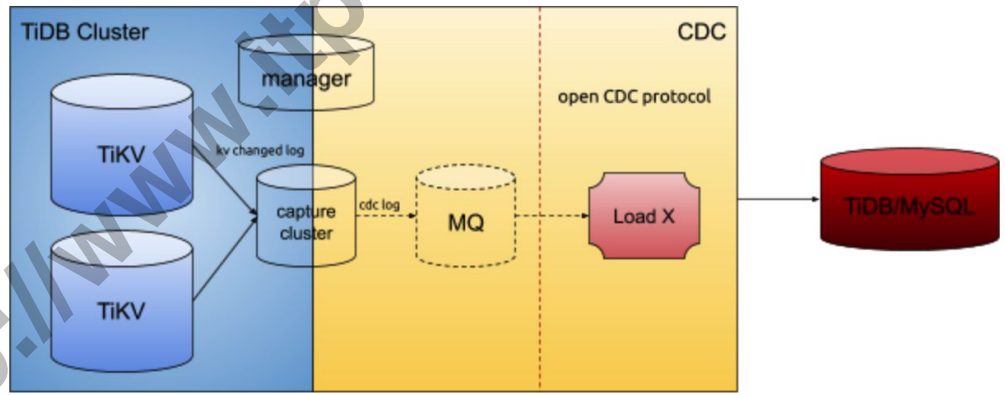
- 本质在于分布式数据库为了得到更大的吞吐(利用集群的能力)，逻辑上将数据进行分片，不同分片在不同的机器上
- 对于不同分片的并发写入，在全局上难以还原顺序
  - 不要忘记 CDC 是流式系统
  - 需要还原顺序必然要排序(单点？)
- 分布式系统的 CDC 本身也是一个分布式系统
  - 高可用？
  - 数据容灾？
  - 如何消除单点故障和瓶颈？
- 推还是拉？

TiCDC 整体架构

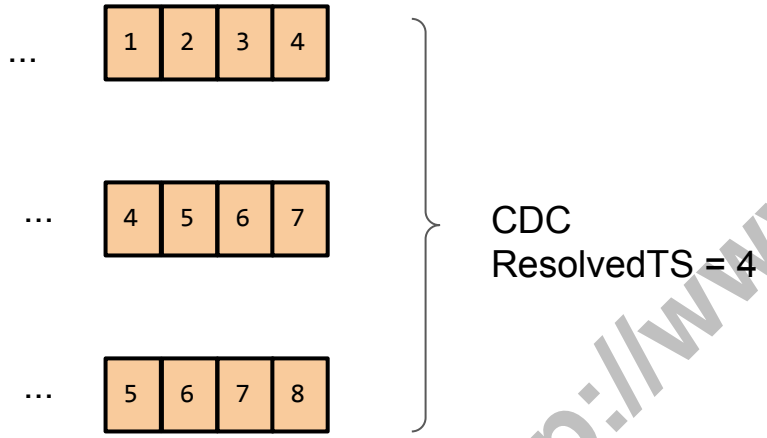


TiCDC 整体架构

- TiKV
- Capture Cluster（无状态）
- MQ (Kafka)
- Loader



TiCDC 整体架构



- 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 的集群调度和高可用

- Owner 选举
  - capture 节点启动时在 PD 自注册（负责推进 CDC）
  - 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

TiCDC 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
    }
  }
}
```

TiCDC 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

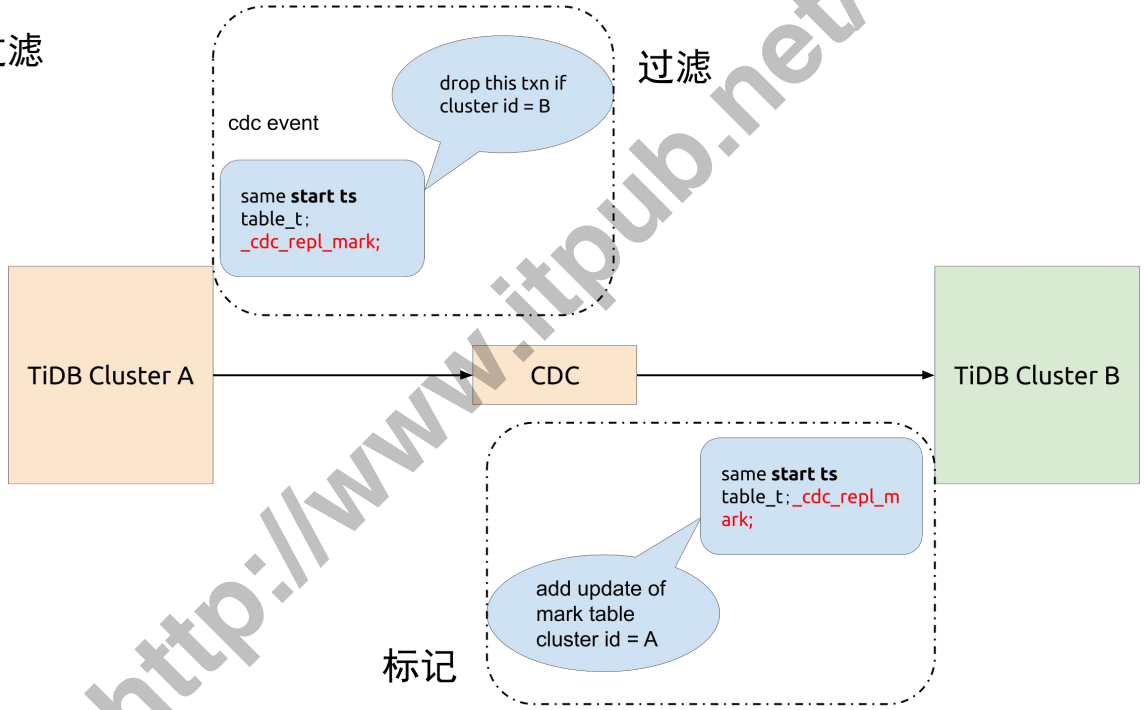
```
{
  "u":{
    "id":{"t":3,  "f":10,  "v":2},
    "name":{ "t":15,  "f":0, "v":"Carol"}
  }
}
```

TiCDC 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

TiCDC 特性: 环形同步

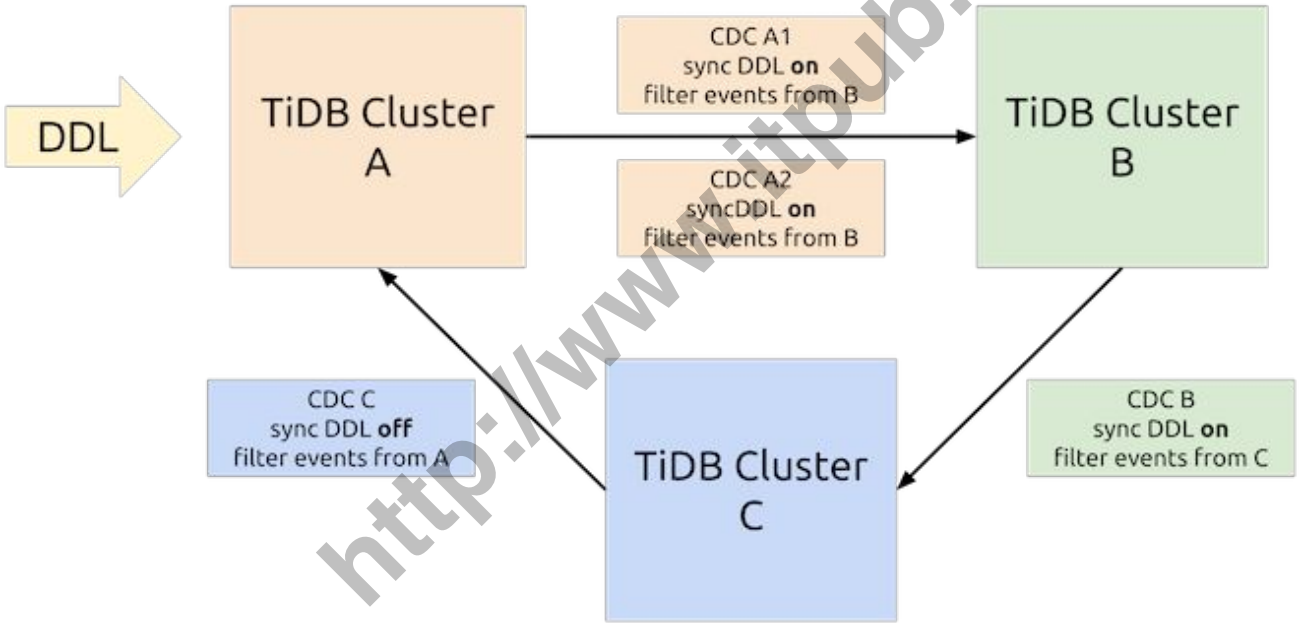
- DML 标记表和过滤



Same for B -> A

TiCDC 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 (PITR)
- K8s Opeartor and SaaS integration

谢谢

https://github.com/pingcap/tidb

https://github.com/pingcap/ticdc

微信公众号：PingCAP

我的邮箱：huang@pingcap.com