WOTA

51CTO

World Of Tech 2017

全球架构与运维技术峰会

2017年4月14日-15日　北京富力万丽酒店

ARCHITECTURE

WOTA

51CTO

出品人及主持人：

吕　毅　链家网架构师
大数据平台团队负责人

大数据系统架构

Streaming ETL系统@Airbnb

WOTA

51CTO

丁辰

Airbnb

软件工程师

分享主题：

Airbnb的Streaming ETL

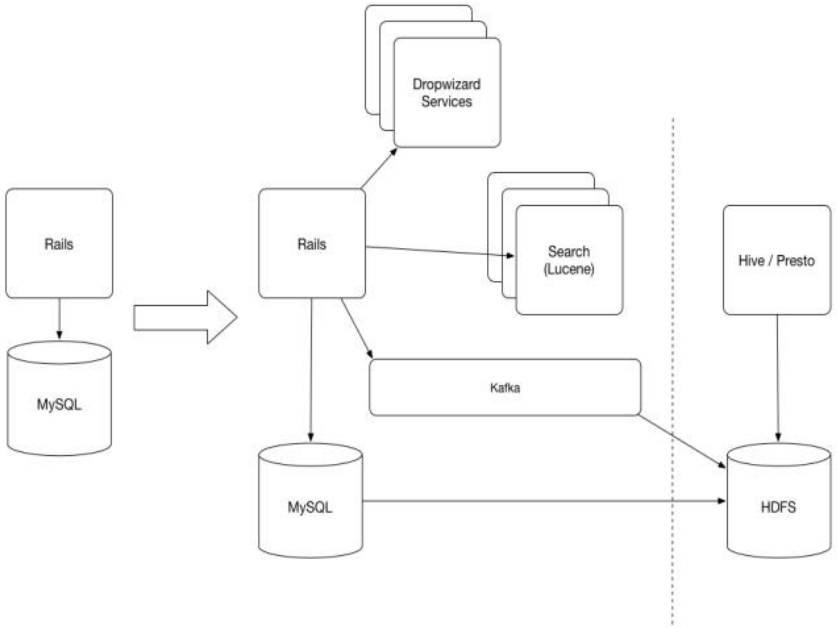
概要

- 数据库变化捕获（Change Data Capture）
- 基于streaming的数据库导出

数据库变化捕获Change Data Capture

Airbnb系统架构的进化

- 从最初单一Rails应用
- 到许多Rails+Java应用



新的挑战

- 业务逻辑跨越了许多不同的服务器
- 原始数据储存在许多不同的数据源

设计原则

- 为生产环境服务
- 考虑将来的增长
- 优先使用成熟的解决方案
- 服务拥有自己的数据，而不是共享它们的存储
- 数据变化应通过标准化事件传播

数据变化捕获（CDC）

- 需要保证时序
- 接近实时
- 保证系统一致

方案一：数据源应用在写数据库的同时记录变化

- 数据模型（简单）
  - 数据源应用生成数据表结构
- 开发（简单）
  - 将数据变化直接写入队列
- 数据一致性（困难）
  - 2PC，一致性协议

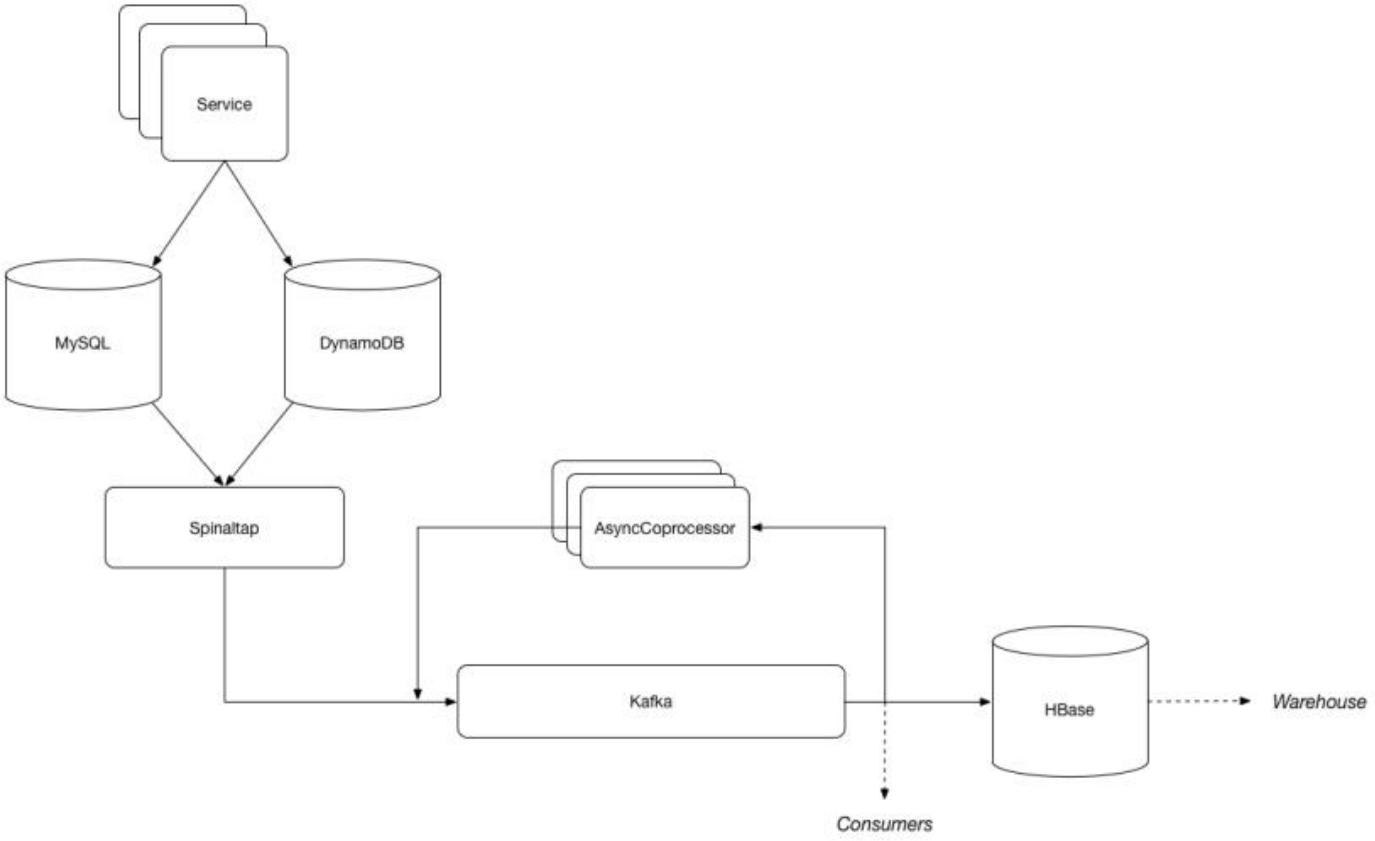
方案二：数据库日志挖掘

- 数据一致性（简单）
  - 依赖数据库提交日志
- 数据模型（困难）
  - 了解数据库表的结构
  - 分析数据库提交日志
  - 处理各种不同的数据源

我们使用了方案二

- 分析提交日志比一致性协议更容易实现
- 不需要改动服务，对服务不造成性能影响
- 提交日志给出了时序上的保证

ETL系统架构

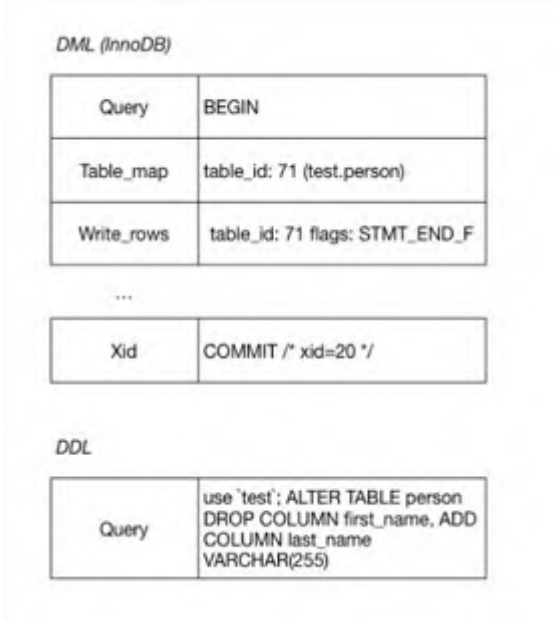

Service
MySQL
DynamoDB
Spinaltap
AsyncCoprocessor
Kafka
HBase
Warehouse
Consumers

系统的需求

- 保证时序一致性，保证生成数据变化至少一次
- 易于添加新的数据源，同时易于扩展（加机器）
- 支持低延迟高吞吐量的用例
- 高可用性，自动故障恢复
- 异构数据源（MySQL，DynamoDB，以及其他)

MySQL提交日志

- 数据变化事件
  - Write_rows, Update_rows, Delete_rows
- 使用提交日志的文件名 / 偏移量 / GTID生成逻辑时序
- 使用XidEvent决定事务的边界



DML (InnoDB)

Query	BEGIN
Table_map	table_id: 71 (test.person)
Write_rows	table_id: 71 flags: STMT_END_F

...

Xid	COMMIT /* xid=20 */

DDL

Query	use `test`; ALTER TABLE person DROP COLUMN first_name, ADD COLUMN last_name VARCHAR(255)

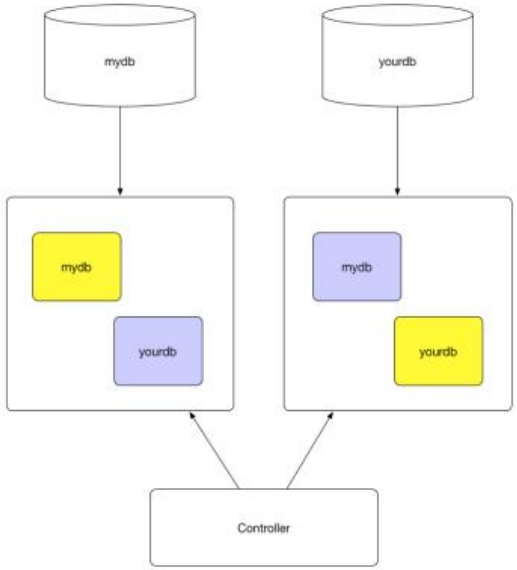
数据变化事件

- 提供逻辑时序
- 数据源特定的元数据（比如MySQL的文件名 / 偏移量 / GTID）
- 变化前的数据
- 变化后的数据
- 将变化事件写入队列（Kafka）

```
{
  id: Long,
  opCode: [
    INSERT,
    UPDATE,
    DELETE
  ],
  metadata: Map<String, String>,
  beforeImage: Record,
  afterImage: Record
}
```

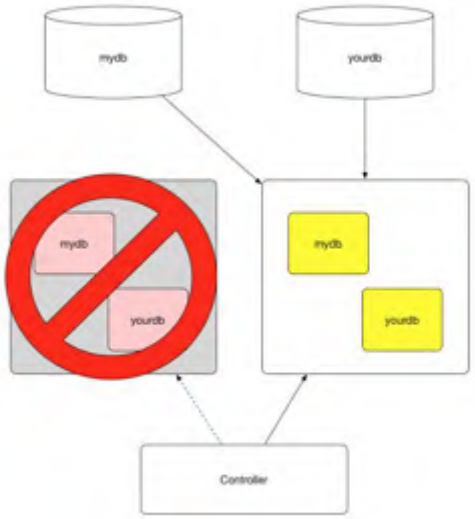
集群配置

- Leader / Standby模型
- 每台机器作为一部分数据源的Leader进行处理
- 使用Apache Helix（基于ZooKeeper）进行集群管理
- 动态配置改变
- 使用Helix group tags

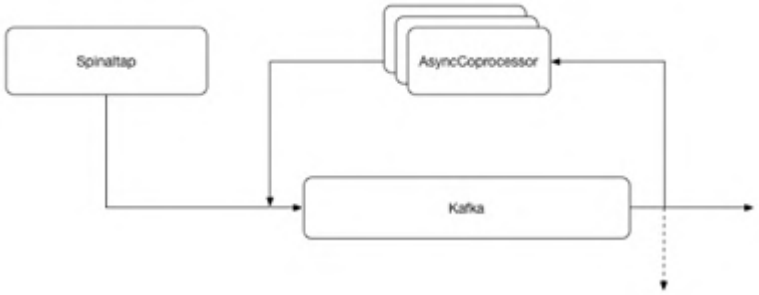


容错

- Controller负责为故障的数据源选举新的Leader
- 成为Leader的同时在zookeeper记录Leader_epoch
- 将时间戳放在ID的前面（Leader_epoch + binlog_file + binlog_pos）



- 配置Kafka
- 保存MySQL原始提交日志
  - 用于出错后从新处理



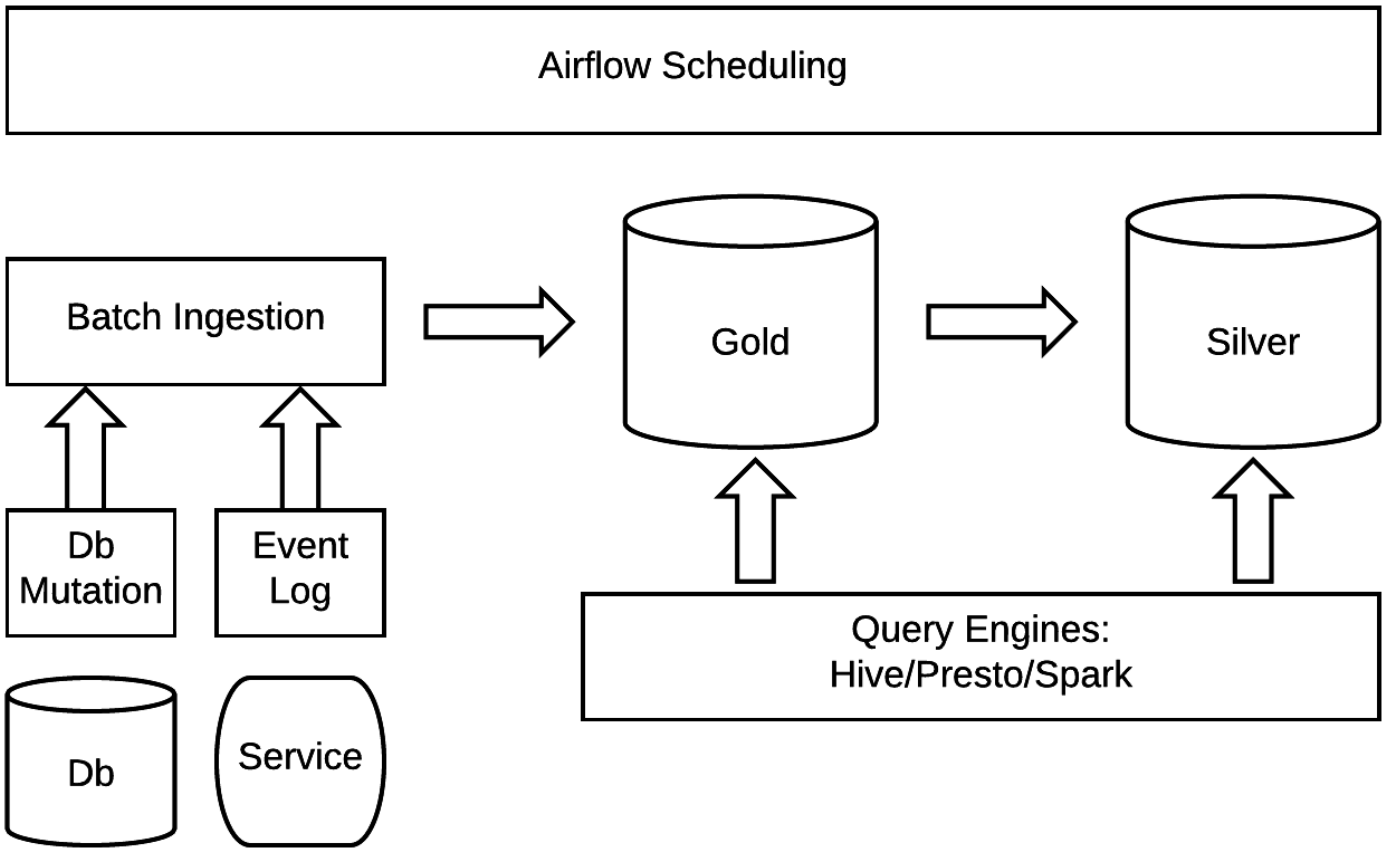

```
*
request.required.acks=-1 # all
default.replication.factor=3
min.insync.replicas=2
```

在线验证

- 从MySQL下载已经写完的binlog
- 和数据流进行对比验证
- 当发生不一致的时候，回滚到之前的提交日志偏移量从新处理

基于流的数据导出

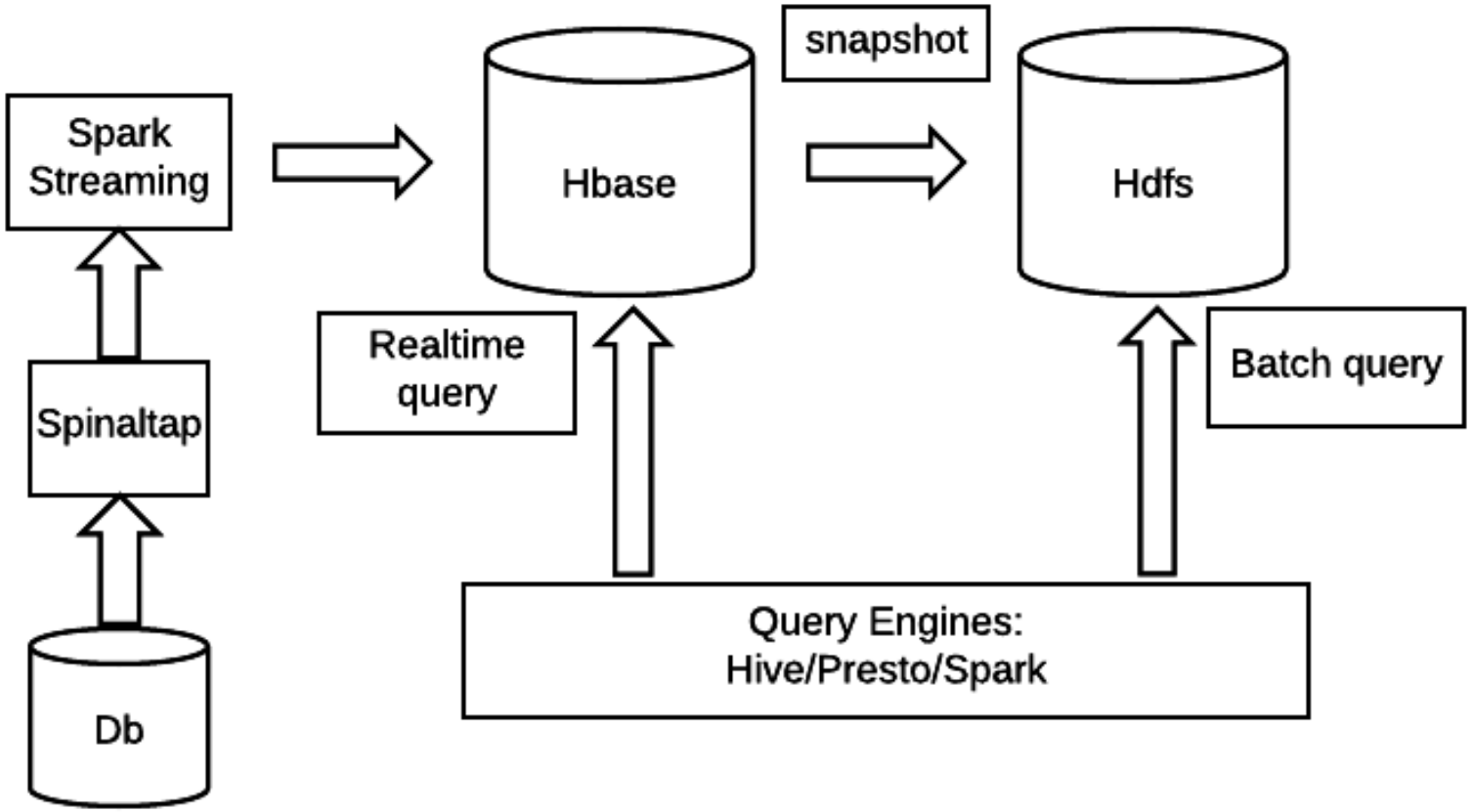
批处理


Airflow Scheduling
Batch Ingestion
Db
Mutation
Event
Log
Db
Service
Gold
Silver
Query Engines:
Hive/Presto/Spark

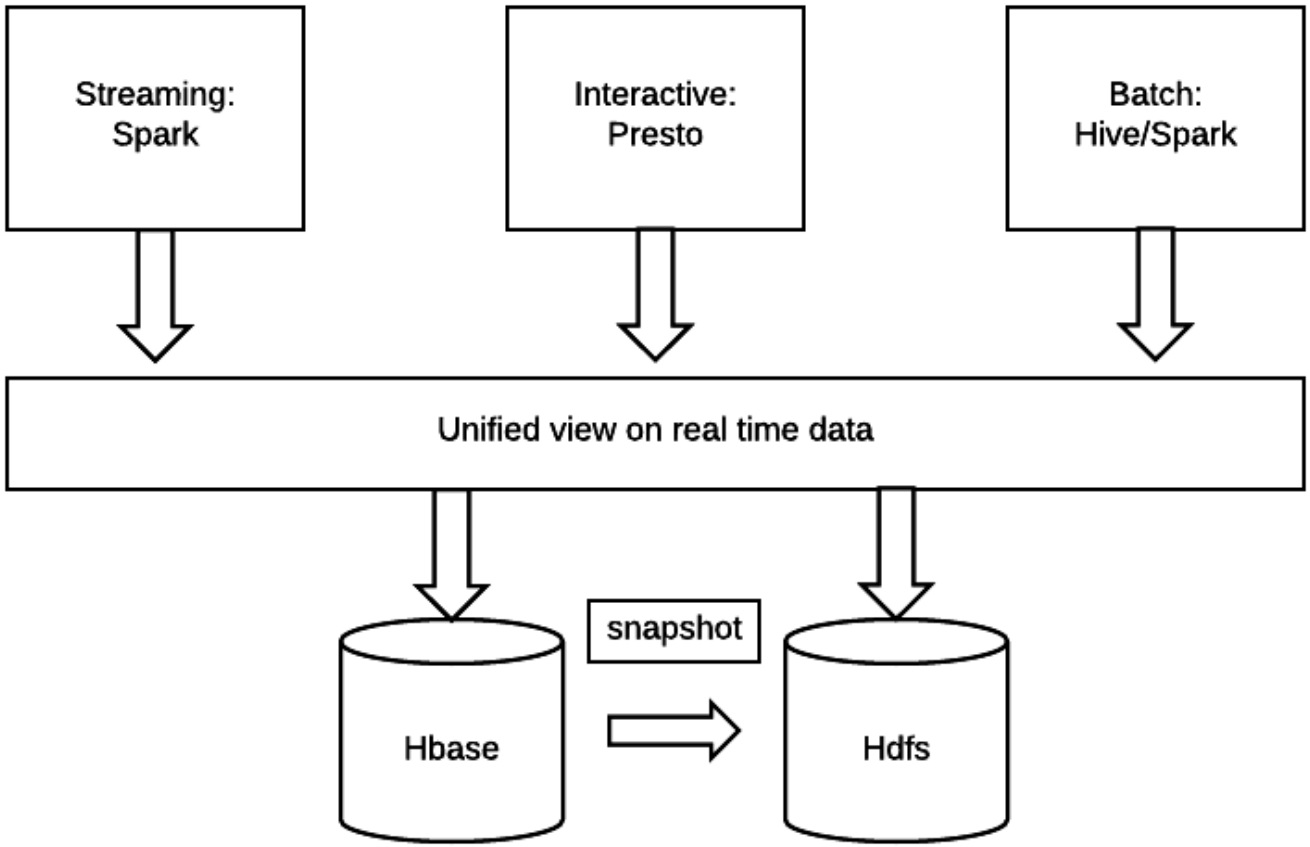
基于批处理的Point-in-Time Restore

- 优点
  - 简单
  - 一致
- 缺点
  - 生成时间没有保证
  - 没有接近实时的数据
  - 没有小时级的快照
  - 需要大量的存储空间

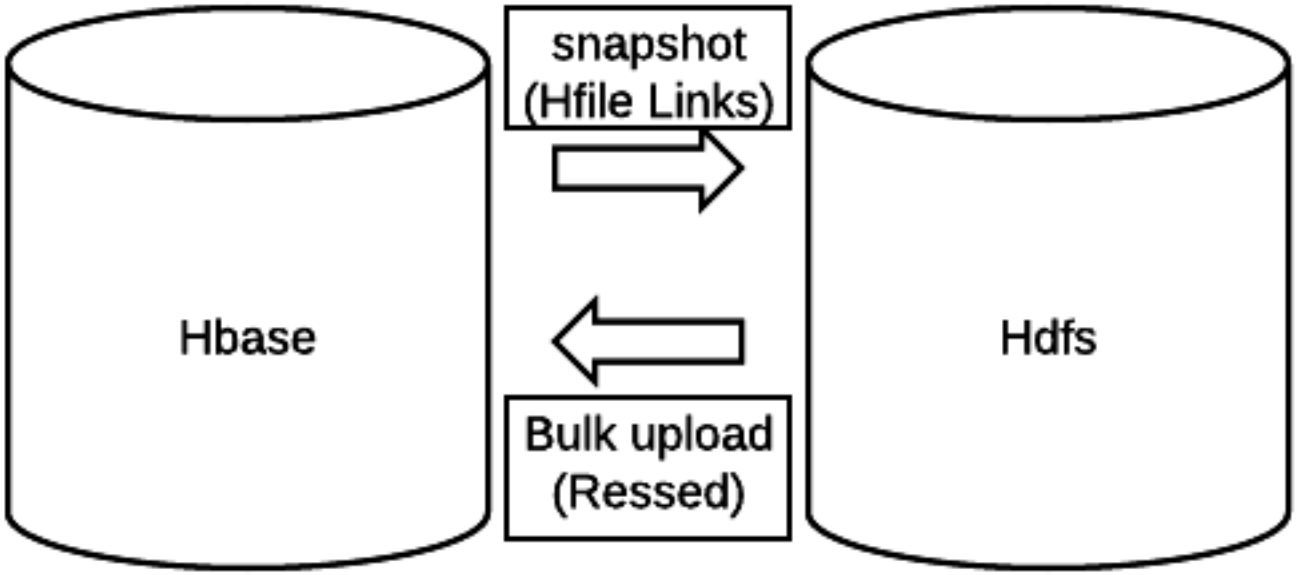
实时导入HBase


Spark Streaming
Spinaltap
Db
Hbase
snapshot
Hdfs
Realtime query
Batch query
Query Engines:
Hive/Presto/Spark

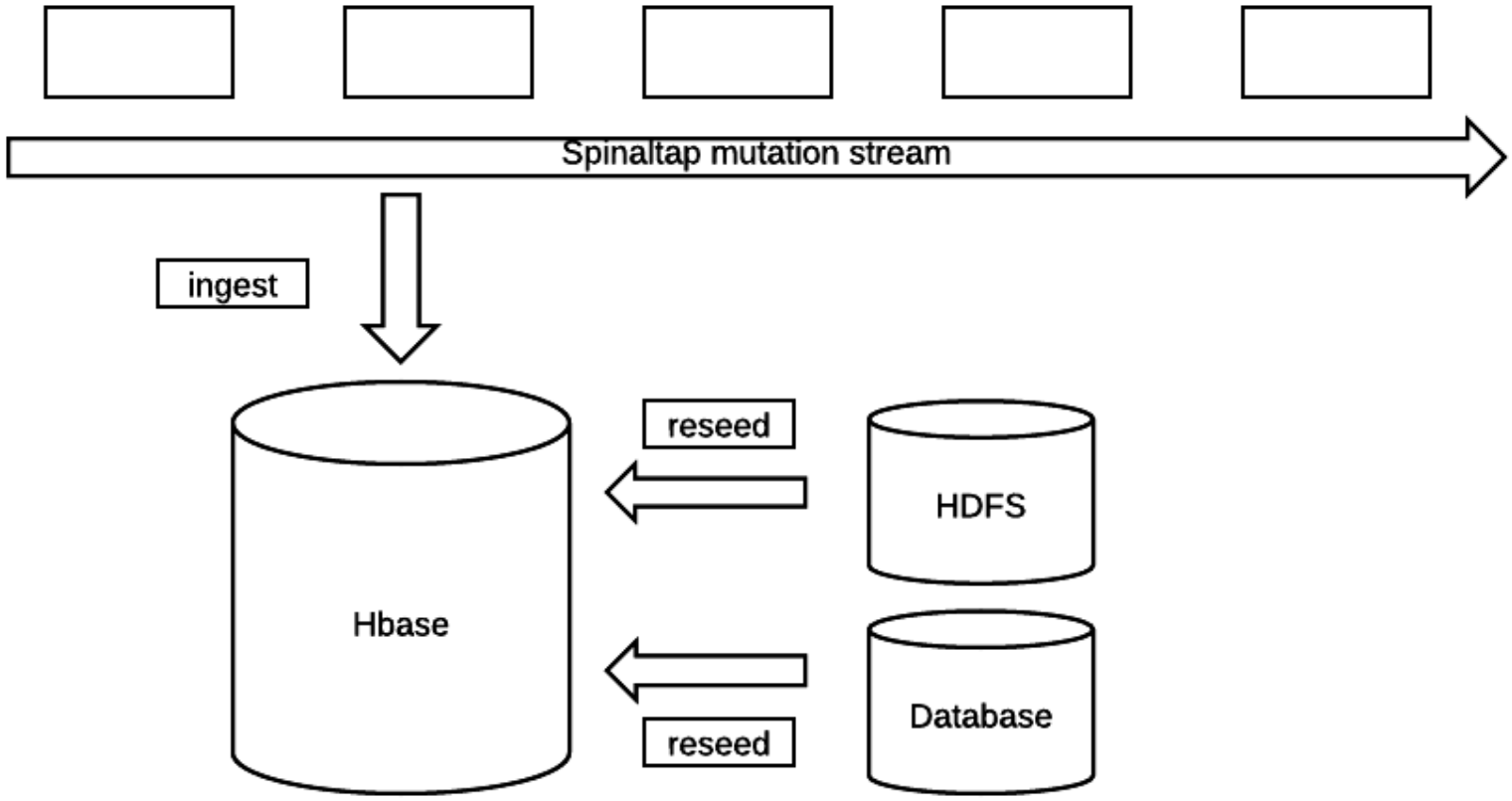
读取HBase


Streaming:
Spark
Interactive:
Presto
Batch:
Hive/Spark
Unified view on real time data
snapshot
Hbase
Hdfs

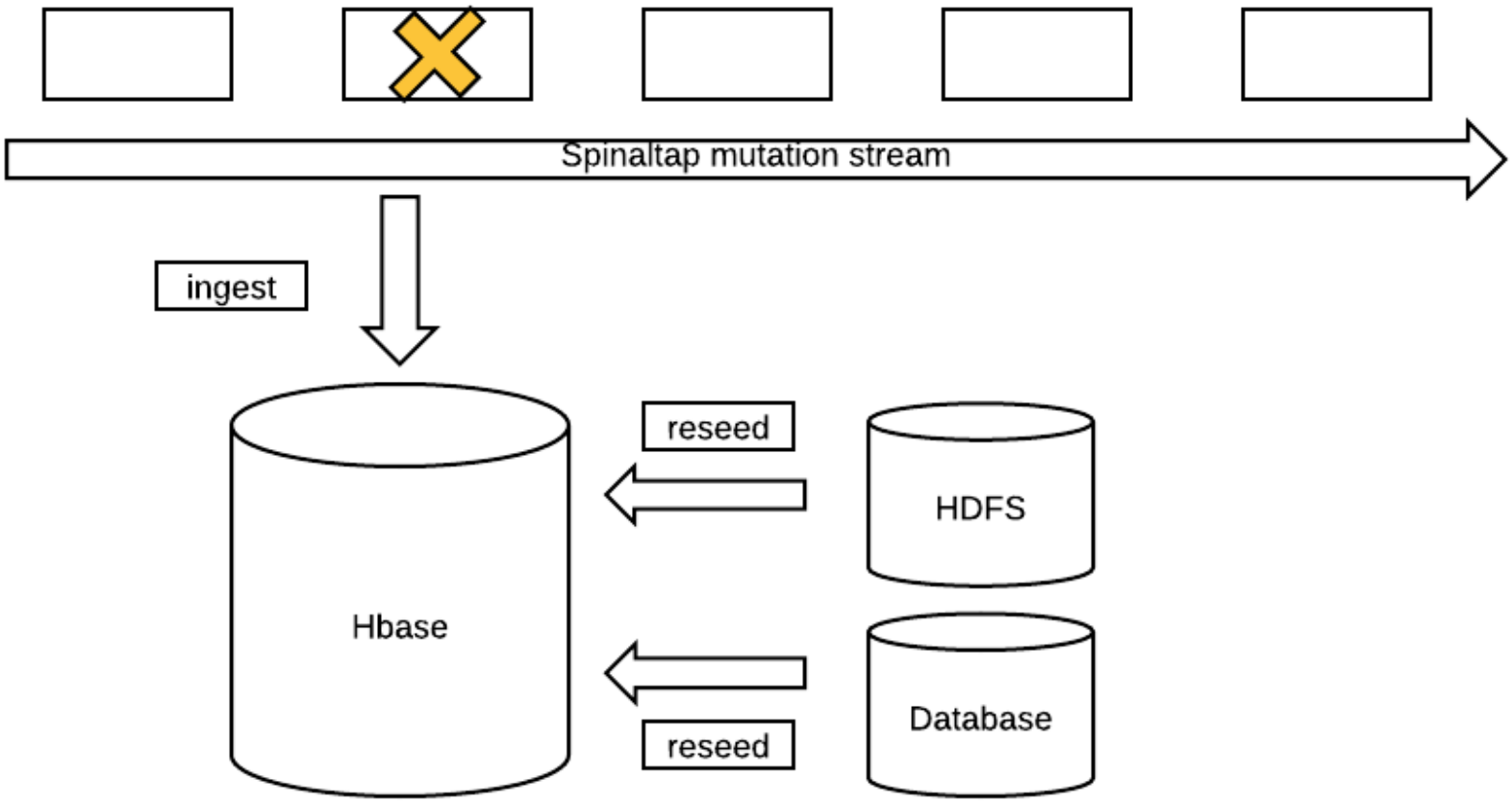
生成镜像 / 重新构建



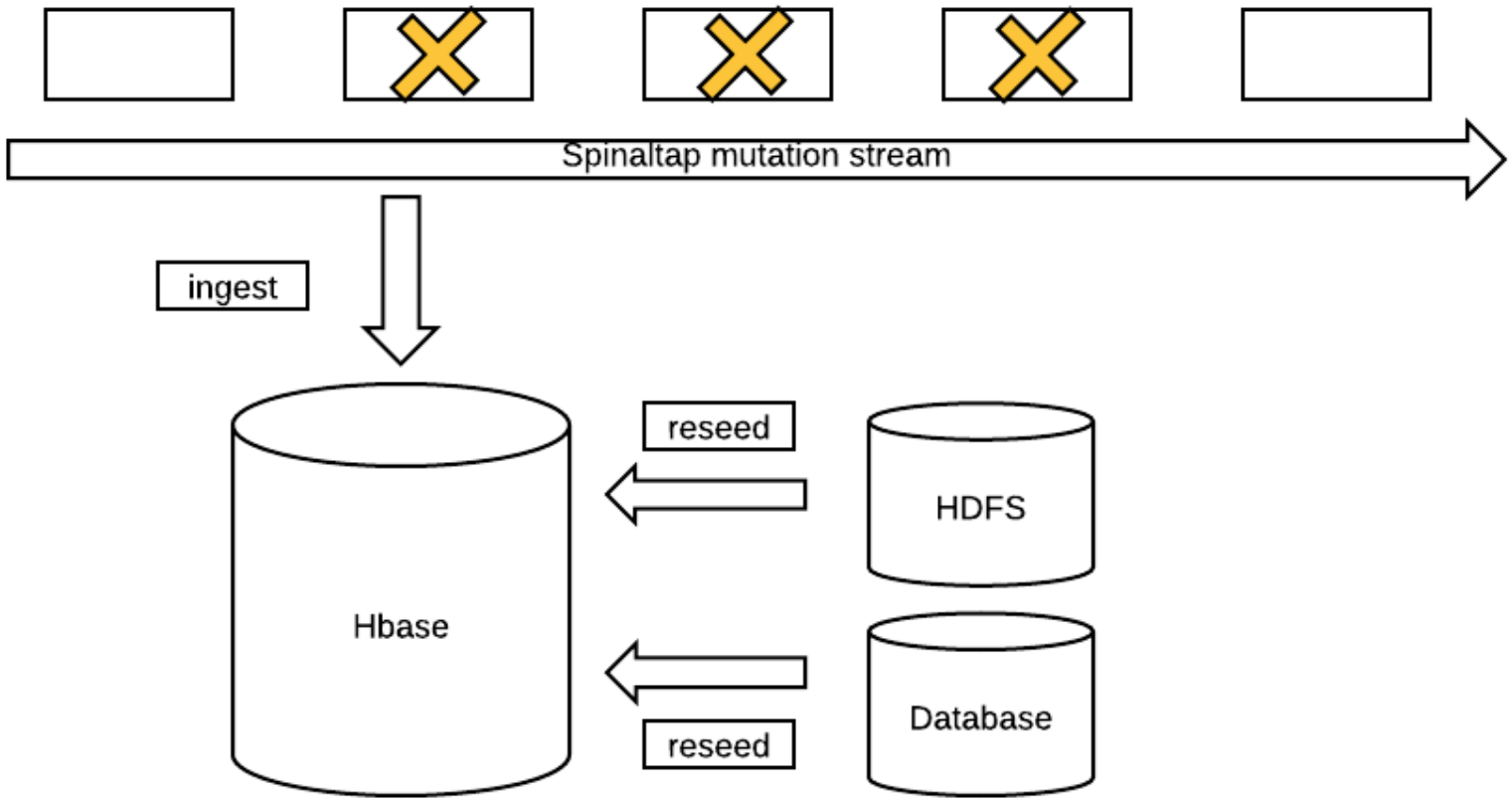
添加新的数据源


Spinaltap mutation stream
ingest
Hbase
reseed
HDFS
reseed
Database

灾难恢复 - 回滚


Spinaltap mutation stream
ingest
Hbase
reseed
HDFS
reseed
Database

灾难恢复 - 重新构建


Spinaltap mutation stream
ingest
Hbase
reseed
HDFS
reseed
Database

HBase schema

- 将所有数据库表存在一个HBase表中
- 快速主键 / 辅助索引查询
- 高效扫表
- 负载均衡

HBase Row Key - 主键

- Hash Key = md5(DB_TABLE, PK1=v1, PK2=v2)
- Row Key = Hash Key + DB_TABLE + PK1=v1 + PK2=v2
  - 快速主键查询
  - 高效扫表
  - 基于Hash Key的负载均衡

Hash	DB_TABLE	PK1=v1	PK2=v2

HBase Row Key - 辅助索引

- Hash Key = md5(DB_TABLE, Index1=v1)
- Row Key = Hash Key + DB_TABLE + Index1=v1 + PK1=v1
  - 基于前缀的索引遍历

Hash	DB_TABLE	Index1=v1	PK1=v1

HBase版本

Rows	CF:Columns	Version	Value
<ShardKey><DB_TABLE#1><PK_a=A>	id	Fri May 19 00:33:19 2016	101
<ShardKey><DB_TABLE#1><PK_a=A>	city	Fri May 19 00:33:19 2016	Beijing
<ShardKey><DB_TABLE#1><PK_a=A>	city	Fri May 20 00:33:19 2016	New York
<ShardKey><DB_TABLE#2><PK_a=A'>	id	Fri May 19 00:33:19 2016	1

基于数据流的数据库导出

- 优点
  - 数据一致
  - 保证了数据库快照生成时间
  - 接近实时的数据查询
  - 兼容Hive / Spark
  - 小时级的数据库快照
  - 节省了存储空间
- 缺点
  - schema变更


WOTA
51CTO
Thank you !