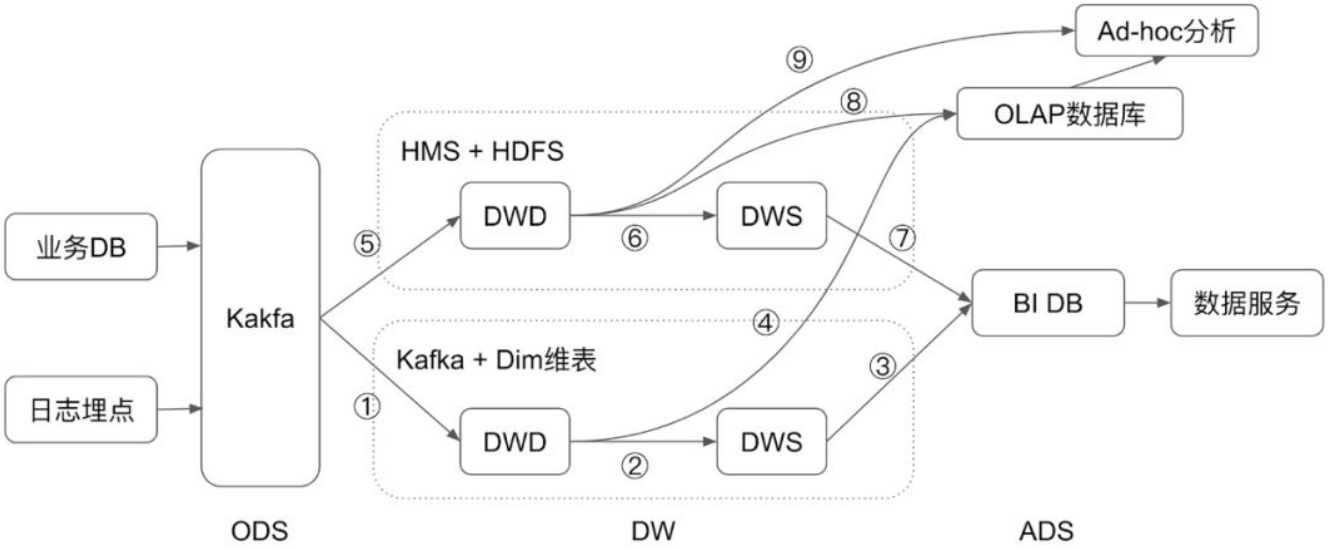
一套 SQL 搞定数据仓库？Flink有了新尝试

数据仓库是公司数据发展到一定规模后必然需要提供的一种基础服务，也是“数据智能”建设的基础环节。迅速获取数据反馈不仅有利于改善产品及用户体验，更有利于公司的科学决策，因此获取数据的实时性尤为重要。

目前企业的数仓建设大多是离线一套，实时一套。业务要求低延时的使用实时数仓；业务复杂的使用离线数仓。架构十分复杂，需要使用很多系统和计算框架，这就要求企业储备多方面的人才，导致人才成本较高，且出了问题难以排查，终端用户也需要熟悉多种语法。本文分析目前的数仓架构，探索离线和实时数仓是否能放在一起考虑，探索Flink的统一架构是否能解决大部分问题。

文末有福利，可下载电子书。

数仓架构



数据仓库可以分为三层：ODS（原始数据层）、DW（数据仓库层）、ADS（应用数据层）。

1. ODS (Operation Data Store) 层

从日志或者业务DB传输过来的原始数据，传统的离线数仓做法也有直接用CDC (Change Data Capture) 工具周期同步到数仓里面。用一套统一的Kafka来承接这个角色，可以让数据更实时的落入数仓，也可以在这一层统一实时和离线的。

2. DW (Data warehouse) 层

DW层一般也分为DWD层和DWS层：

- DWD (Data warehouse detail) 层：明细数据层，这一层的数据应该是经过清洗的，干净的、准确的数据，

它包含的信息和ODS层相同，但是它遵循数仓和数据库的标准Schema定义。

- DWS (Data warehouse service) 层：汇总数据层，这一层可能经过了轻度的聚合，可能是星型或雪花模型的结构数据，这一层已经做了一些业务层的计算，用户可以基于这一层，计算出数据服务所需数据。

3. ADS (Application Data Store) 层

和DWS不同的是，这一层直接面向用户的数据服务，不需要再次计算，已经是最终需要的数据。

主要分为两条链路：

1. 业务DB和日志 -> Kafka -> 实时数仓 (Kafka + Dim维表) -> BI DB -> 数据服务
2. 业务DB和日志 -> Kafka -> 离线数仓 (Hive metastore + HDFS) -> BI DB -> 数据服务

主流的数仓架构仍然是Lambda架构，Lambda架构虽然复杂，但是它能覆盖业务上需要的场景，对业务来说，是最灵活的方式。

Lambda架构分为两条链路：

- 传统离线数据具有稳定、计算复杂、灵活的优点，运行批计算，保证T+1的报表产生和灵活的Ad-hoc查询。
- 实时数仓提供低延时的数据服务，传统的离线数仓往往都是T+1的延时，这导致分析人员没法做一些实时化的决策，而实时数仓整条链路的延迟最低甚至可以做到秒级，这不但加快了分析和决策，而且也给更多的业务带来了可能，比如实时化的监控报警。Flink的强项是实时计算、流计算，而Kafka是实时数仓存储的核心。

上图标出了1-9条边，每条边代表数据的转换，就是大数据的计算，本文后续将分析这些边，探索Flink在其中可以发挥的作用。

Flink一栈式计算

元数据

先说下元数据的管理，离线数仓有Hive metastore来管理元数据，但是单纯的Kafka不具备元数据管理的能力，这里推荐两种做法：

1. Confluent schema registry

搭建起schema registry服务后，通过confluent的url即可获取到表的schema信息，对于上百个字段的表，它可以省编写Flink作业时的很多事，后续Flink也正在把它的schema推断功能结合Confluent schema registry。但是它仍然省不掉创建表的过程，用户也需要填写Confluent对应的URL。

2. Catalog

目前Flink内置已提供了HiveCatalog，Kafka的表可以直接集成到Hive metastore中，用户在SQL中可以直接使用这些表。但是Kafka的start-offset一些场景需要灵活的配置，为此，Flink也正在提供 LIKE [1] 和 Table Hints [2]

等手段来解决。

Flink中离线数仓和实时数仓都使用Hive Catalog：

```
use catalog my_hive;
-- build streaming database and tables;
create database stream_db;
use stream_db;
create table order_table (
    id long,
    amount double,
    user_id long,
    status string,
    ts timestamp,
    … -- 可能还有几十个字段
    ts_day string,
    ts_hour string
) with (
    'connector.type' = 'kafka',
    … -- Kafka table相关配置
);
-- build batch database and tables;
create database batch_db;
use batch_db;
create table order_table like stream_db.order_table (excluding options)
partitioned by (ts_day, ts_hour)
with (
    'connector.type' = 'hive',
    … -- Hive table相关配置
);
```

使用Catalog，后续的计算可以完全复用批和流，提供相同的体验。

数仓导入

计算①和⑤分别是实时数仓的导入和离线数仓的导入，近来，更加实时的离线数仓导入越来越成为数据仓库的常规做法，Flink的导入可以让离线数仓的数据更实时化。

以前主要通过DataStream + StreamingFileSink的方式进行导入，但是不支持ORC和无法更新HMS。

Flink streaming integrate Hive后，提供Hive的streaming sink [3]，用SQL的方式会更方便灵活，使用SQL的内置函数和UDF，而且流和批可以复用，运行两个流计算作业。

```
insert into [stream_db.|batch_db.]order_table select … from log_table;
```

数据处理

计算②和⑥分别是实时数仓和离线数仓的中间数据处理，这里面主要有三种计算：

1. **ETL**：和数据导入一样，批流没有区别。
2. **维表Join**：维表补字段是很常见的数仓操作，离线数仓中基本都是直接Join Hive表即可，但是Streaming作业却有些不同，下文将详细描述。
3. **Aggregation**：Streaming作业在这些有状态的计算中，产生的不是一次确定的值，而可能是不断变化的值。

维表Join

与离线计算不同，离线计算只用关心某个时间点的维表数据，而Streaming的作业持续运行，所以它关注的不能只是静态数据，需要是动态的维表。

另外为了Join的效率，streaming作业往往是join一个数据库表，而不仅仅是Hive表。

例子：

```
-- stream 维表
use stream_db;
create table user_info (
    user_id long,
    age int,
    address,
    primary key(user_id)
) with (
    'connector.type' = 'jdbc',
    ...
);

-- 将离线数仓的维表导入实时数仓中
insert into user_info select * from batch_db.user_info;

-- 维表Join，SQL批流复用
insert into order_with_user_age select * from order_table join user_info for syst
```

这里有个非常麻烦的事情，那就是在实时数仓中，需要按时周期调度更新维表到实时维表数据库中，那能不能直接Join离线数仓的Hive维表呢？目前社区也正在开发Hive维表，它有哪些挑战：

1. Hive维表太大，放不进Cache中：

- 考虑Shuffle by key，分布式的维表Join，减少单并发Cache的数据量
- 考虑将维表数据放入State中

1. 维表更新问题：

- 简单的方案是TTL过期
- 复杂一些的方案是实现Hive streaming source，并结合Flink的watermark机制

有状态计算和数据导出

例子：

```
select age, avg(amount) from order_with_user_age group by age;
```

一句简单的聚合SQL，它在批计算和流计算的执行模式是完全不同的。

Streaming的聚合和离线计算的聚合最大的不同在于它是一个动态表[4]，它的输出是在持续变化的。动态表的概念简单来说，一个streaming的count，它的输出是由输入来驱动的，而不是像batch一样，获取全部输入后才会输出，所以，它的结果是动态变化的：

- 如果在SQL内部，Flink内部的retract机制会保证SQL 的结果的与批一样。
- 如果是外部的存储，这给sink带来了挑战。

有状态计算后的输出：

- 如果sink是一个可更新的数据库，比如HBase/Redis/JDBC，那这看起来不是问题，我们只需要不断的去更新就好了。
- 但是如果是不可更新的存储呢，我们没有办法去更新原本的数据。为此，Flink提出了Changelog的支持[5]，想内置支持这种sink，输出特定Schema的数据，让下游消费者也能很好的work起来。

例子：

```
-- batch：计算完成后，一次性输出到mysql中，同key只有一个数据
-- streaming：mysql里面的数据不断更新，不断变化
insert into mysql_table select age, avg(amount) from order_with_user_age group by
-- batch: 同key只有一个数据，append即可
insert into hive_table select age, avg(amount) from order_with_user_age group by
-- streaming: kafka里面的数据不断append，并且多出一列，来表示这是upsert的消息，后续的Flink
insert into kafka_table select age, avg(amount) from order_with_user_age group by
```

AD-HOC与OLAP

离线数仓可以进行计算⑨，对明细数据或者汇总数据都可以进行ad-hoc的查询，可以让数据分析师进行灵活的查询。

目前实时数仓一个比较大的缺点是不能Ad-hoc查询，因为它本身没有保存历史数据，Kafka可能可以保存3天以上的数据，但是一是存储成本高、二是查询效率也不好。

一个思路是提供OLAP数据库的批流统一Sink组件：

- Druid sink
- Doris sink
- Clickhouse sink
- HBase/Phoenix sink

总结

本文从目前的Lambda架构出发，分析了Flink一栈式数仓计算方案的能力，本文中一些Flink新功能还在快速迭代演进中，随着不断的探索和实践，希望朝着计算一体化的方向逐渐推进，将来的数仓架构希望能真正统一用户的离线和实时，提供统一的体验：

- 统一元数据
- 统一SQL开发
- 统一数据导入与导出
- 将来考虑统一存储

参考

[1]https://cwiki.apache.org/confluence/display/FLINK/FLIP-110%3A+Support+LIKE+clause+in+CREATE+TABLE

[2]https://cwiki.apache.org/confluence/display/FLINK/FLIP-113%3A+Supports+Table+Hints

[3]https://cwiki.apache.org/confluence/display/FLINK/FLIP-115%3A+Filesystem+connector+in+Table

[4]https://ci.apache.org/projects/flink/flink-docs-master/dev/table/streaming/dynamic_tables.html

[5]https://cwiki.apache.org/confluence/display/FLINK/FLIP-105%3A+Support+to+Interpret+and+Emit+Changelog+in+Flink+SQL