

Flink x Zeppelin , Hive Streaming 实战解析

发布于: 2020 年 08 月 04 日

Flink 1.11 正式发布已经三周了，其中最吸引我的特性就是 Hive Streaming。正巧 Zeppelin-0.9-preview2 也在前不久发布了，所以就写了一篇 Zeppelin 上的 Flink Hive Streaming 的实战解析。本文主要从以下几部分跟大家分享：

Hive Streaming 的意义

Checkpoint & Dependency

写入 Kafka

Hive Streaming Sink

Hive Streaming Source

Hive Temporal Table

Hive Streaming 的意义

很多同学可能会好奇，为什么 Flink 1.11 中，Hive Streaming 的地位这么高？它的出现，到底能给我们带来什么？

其实在大数据领域，一直存在两种架构 **Lambda** 和 **Kappa**：

Lambda 架构——流批分离，静态数据通过定时调度同步到 Hive 数仓，实时数据既会同步到 Hive，也会被实时计算引擎消费，这里就引出了一点问题。

数据口径问题

离线计算产出延时太大

数据冗余存储

Kappa 架构——全部使用实时计算来产出数据，历史数据通过回溯消息的消费位点计算，同样也有很多的问题，毕竟没有一劳永逸的架构。

消息中间件无法保留全部历史数据，同样数据都是行式存储，占用空间太大

实时计算计算历史数据力不从心
无法进行 Ad-Hoc 的分析

为了解决这些问题，行业内推出了实时数仓，解决了大部分痛点，但是还是有些地方力不从心。比如涉及到历史数据的计算怎么办？我想做 Ad-Hoc 的分析又怎么玩？所以行业内现在都是实时数仓与离线数仓并行存在，而这又带来了更多的问题：模型需要多份、数据产出不一致、历史数据的计算等等。

而 Hive Streaming 的出现就可以解决这些问题！再也不用多套模型了；也不需要同一个指标因为涉及到历史数据，写一遍实时 SQL 再写一遍离线 SQL；Ad-Hoc 也能做了，怎么做？读 Hive Streaming 产出的表就行！

接下来，让我们从参数配置开始，接着流式的写入 Hive，再到流式的读取 Hive 表，最后再 Join 上 Hive 维表吧。这一整套流程都体验后，想必大家对 Hive Streaming 一定会有更深入的了解，更能够体会到它的作用。

Checkpoint & Dependency

因为只有在完成 Checkpoint 之后，文件才会从 In-progress 状态变成 Finish 状态，所以，我们需要合理的去配置 Checkpoint，在 Zeppelin 中配置 Checkpoint 很简单。

```
%flink.conf

# checkpoint 配置

pipeline.time-characteristic EventTime
execution.checkpointing.interval 120000
execution.checkpointing.min-pause 60000
execution.checkpointing.timeout 60000
execution.checkpointing.externalized-checkpoint-retention RETAIN_ON_CANCELLATION

# 依赖jar包配置

flink.execution.packages org.apache.flink:flink-connector-kafka_2.11:1.11.0,org.a
```

又因为我们需要从 Kafka 中读取数据，所以将 Kafka 的依赖也加入进去了。

写入Kafka

我们的数据来自于天池数据集，是以 CSV 的格式存在于本地磁盘，所以需要先将他们写入 Kafka。

先建一下 CSV Source 和 Kafka Sink 的表：

```
%flink.ssql
SET table.sql-dialect=default;
DROP TABLE IF EXISTS source_csv;
CREATE TABLE source_csv (
  user_id string,
  theme_id string,
  item_id string,
  leaf_cate_id string,
  cate_level1_id string,
  clk_cnt int,
  reach_time string
) WITH (
  'connector' = 'filesystem',
  'path' = 'file:///Users/dijie/Downloads/Cloud_Theme_Click/theme_click_log.csv',
  'format' = 'csv'

)
```

```
%flink.ssql
SET table.sql-dialect=default;
DROP TABLE IF EXISTS kafka_table;
CREATE TABLE kafka_table (
  user_id string,
  theme_id string,
  item_id string,
  leaf_cate_id string,
  cate_level1_id string,
  clk_cnt int,
  reach_time string,
  ts AS localtime,
  WATERMARK FOR ts AS ts - INTERVAL '5' SECOND
) WITH (
  'connector' = 'kafka',
  'topic' = 'theme_click_log',
  'properties.bootstrap.servers' = '10.70.98.1:9092',
```

```
'properties.group.id' = 'testGroup',  
'format' = 'json',  
'scan.startup.mode' = 'latest-offset'  
)
```

因为注册的表既可以读又可以写，于是我在建表时将 Watermark 加上了；又因为源数据中的时间戳已经很老了，所以我这里采用当前时间减去5秒作为我的 Watermark。

大家可以看到，我在语句一开始指定了 SQL 方言为 Default，这是为啥呢？还有别的方言吗？别急，听我慢慢说。

其实在之前的版本，Flink 就已经可以和 Hive 打通，包括可以把表建在 Hive 上，但是很多语法和 Hive 不兼容，包括建的表在 Hive 中也无法查看，主要原因就是方言不兼容。所以，在 Flink 1.11 中，为了减少学习成本（语法不兼容），可以用 DDL 建 Hive 表并在 Hive 中查询，Flink 支持了方言，默认的就是 Default 了，就和之前一样，如果想建 Hive 表，并支持查询，请使用 Hive 方言，具体可以参考下方链接。

Hive 方言：

https://ci.apache.org/projects/flink/flink-docs-release-1.11/dev/table/hive/hive_catalog.html

再把数据从 CSV 中读取后写入 Kafka。

```
%flink.ssql(type=update)  
  
insert into kafka_table select * from source_csv ;
```

%!link.sql(type=update)
select * from kafka_table ;

FLINK JOB RUNNING 0%

user_id	theme_id	item_id	leaf_cate_id	cate_level1_id	clk_cnt	reach_time	ts
002d04c4ce1f46ae7a21 b6b68bce589d	33900000100010	b8a9d26a3f9f0a81aacd9 a33fc9e9780	32e702d06d3616e1a41a 9216d763be2b	a1ce652a0fd987c0411a3 7b3758aa6d1	2	20181207113901	2020-07-23 17:02:12.337
006be464cdd2fc6db0dc 9afdb655c9a8	33901101000001	0f15e1922e4477c29a495 3ecd63a0530	852a581f78e4f4d7ecf99 72a038e440c	933cb5e203c8e082cbf0 51856a7f26d4	1	20181209002502	2020-07-23 17:02:12.339
00daa5614fe2d9f14ee1d 16ea8f3b438	33901011100111	f04468846324c32fd8c5c d85197a05f8	41bf2bef4d2f0bbd0b0b7 76e40c49403	a981f2b708044d6fb4a71 a1463242520	1	20181212010205	2020-07-23 17:02:11.822
013446d12ebf40353d48 2835a169ea2e	33900000011110	d36cb06834ed85dba3e9 39b3a1bb1a35	7f0f1ffc9c19db4df52393 3010cd7a48	8e296a067a37563370de d05f5a3bf3ec	1	20181208221728	2020-07-23 17:02:12.261
0148391a217e2cd59519 6331f56c2b27	33901101000110	16b95f37abbc1afa488a7 4940b10f56e	126c091a8ff171f694733 a624089e68a	933cb5e203c8e082cbf0 51856a7f26d4	1	20181212015409	2020-07-23 17:02:12.345

Started a minute ago.

Hive Streaming Sink

```
%flink.sql
SET table.sql-dialect=hive;
DROP TABLE IF EXISTS hive_table;
CREATE TABLE hive_table (
  user_id string,
  theme_id string,
  item_id string,
  leaf_cate_id string,
  cate_level1_id string,
  clk_cnt int,
  reach_time string
) PARTITIONED BY (dt string, hr string, mi string) STORED AS parquet TBLPROPERTIES (
  'partition.time-extractor.timestamp-pattern'='$dt $hr:$mi:00',
  'sink.partition-commit.trigger'='partition-time',
  'sink.partition-commit.delay'='1 min',
  'sink.partition-commit.policy.kind'='metastore,success-file'
);
```

partition.time-extractor.timestamp-pattern: 分区时间抽取器, 与 DDL 中的分区字段保持一致;

sink.partition-commit.trigger: 分区触发器类型, 可选 process-time 或 partition-time。

process-time: 不需要上面的参数, 也不需要水印, 当当前时间大于分区创建时间

+sink.partition-commit.delay 中定义的时间, 提交分区; partition-time: 需要 Source 表中定义 watermark, 当 watermark > 提取到的分区时间 +sink.partition-commit.delay 中定义的时间, 提交分区;

sink.partition-commit.delay: 相当于延时时间;

sink.partition-commit.policy.kind: 怎么提交, 一般提交成功之后, 需要通知 metastore, 这样 Hive 才能读到你最新分区的数据; 如果需要合并小文件, 也可以自定义 Class, 通过实现 PartitionCommitPolicy 接口。

接下来让我们把数据插入刚刚创建的 Hive Table:

```
%flink.sql

insert into hive_table select  user_id,theme_id,item_id,leaf_cate_id,cate_level1_
```

让程序再跑一会儿~我们先去倒一杯 95 年的 Java☕。

然后再看看我们的 HDFS, 看看路径下的东西。

```
[data@data ~]# /home/data 21:55
$ hdfs dfs -ls /apps/hive/warehouse/hive_table
Found 1 items
drwxrwxr-x - data supergroup          0 2020-07-23 21:51 /apps/hive/warehouse/hive_table/dt=2020-07-23

[data@data ~]# /home/data 21:56
$ hdfs dfs -ls /apps/hive/warehouse/hive_table/dt=2020-07-23
Found 1 items
drwxrwxr-x - data supergroup          0 2020-07-23 21:56 /apps/hive/warehouse/hive_table/dt=2020-07-23/hr=21

[data@data ~]# /home/data 21:56
$ hdfs dfs -ls /apps/hive/warehouse/hive_table/dt=2020-07-23/hr=21
Found 6 items
drwxrwxr-x - data supergroup          0 2020-07-23 21:52 /apps/hive/warehouse/hive_table/dt=2020-07-23/hr=21/mi=51
drwxrwxr-x - data supergroup          0 2020-07-23 21:54 /apps/hive/warehouse/hive_table/dt=2020-07-23/hr=21/mi=52
drwxrwxr-x - data supergroup          0 2020-07-23 21:54 /apps/hive/warehouse/hive_table/dt=2020-07-23/hr=21/mi=53
drwxrwxr-x - data supergroup          0 2020-07-23 21:54 /apps/hive/warehouse/hive_table/dt=2020-07-23/hr=21/mi=54
drwxrwxr-x - data supergroup          0 2020-07-23 21:55 /apps/hive/warehouse/hive_table/dt=2020-07-23/hr=21/mi=55
drwxrwxr-x - data supergroup          0 2020-07-23 21:56 /apps/hive/warehouse/hive_table/dt=2020-07-23/hr=21/mi=56
```

大家也可以用 Hive 自行查询看看, 我呢就先卖个关子, 一会儿用 Hive Streaming 来读数据。

Hive Streaming Source

因为 Hive 表上面已经创建过了，所以这边读数据的时候直接拿来用就行了，不同的地方是需要使用 Table Hints 去覆盖参数。

Hive Streaming Source 最大的不足是，无法读取已经读取过的分区下新增的文件。简单来说就是，读过的分区，就不会再读了。看似很坑，不过仔细想想，这样才符合流的特性。

照旧给大家说一下参数的意思：

stream-source.enable: 显而易见，表示是否开启流模式。

stream-source.monitor-interval: 监控新文件/分区产生的间隔。

stream-source.consume-order: 可以选 create-time 或者 partition-time; create-time 指的并不是分区创建时间, 而是在 HDFS 中文件/文件夹的创建时间; partition-time 指的是分区的时间; 对于非分区表, 只能用 create-time。官网这边的介绍写的有点模糊, 会让人误以为可以查到已经读过的分区下新增的文件, 其实经过我的测试和翻看源码发现并不能。

stream-source.consume-start-offset: 表示从哪个分区开始读。

光说不干假把式，让我们捞一把数据看看~

[illegible]

SET 那一行得帶着，不然無法使用 Table Hints。

Hive Temporal Table

看完了 Streaming Source 和 Streaming Sink，让我们最后再试一下 Hive 作为维表吧。

其实用 Hive 维表很简单，只要是在 Hive 中存在的表，都可以当做维表使用，参数完全可以用 Table Hints 来覆盖。

lookup.join.cache.ttl：表示缓存时间；这里值得注意的是，因为 Hive 维表会把维表所有数据缓存在 TM 的内存中，如果维表量很大，那么很容易就 OOM；如果 ttl 时间太短，那么会频繁的加载数据，性能会有很大影响。

```
%flink.sql(type=update)
SET table.dynamic-table-options.enabled=TRUE;
select b.*,a.* from (select *, proctime() as p from hive_table /*+ OPTIONS('streaming-source.enable'='true', 'streaming-source.consume-start-offset'='2020-07-23')*/)
as a left join user_item_purchase_log for system_time as of a.p as b on a.user_id = b.user_id and a.item_id = b.item_id;
```

FLINK JOB RUNNING 0%

user_id	item_id	paytime	user_id0	theme_id	item_id0	leaf_cate_id	cate_level1_id	clk_cnt
null	null	null	4b874edb42a895cd83881212bcab7e6d	33901000110010	851e1d0f22ce528ea13890d67c93e308	664162d351ab59ea042546db288bb7c	ce72c3f83a25f6e6d9ad0536433b07fd	1
null	null	null	f9f5d4d4f3d63be269dcbdde50dcffb2	33900111001011	8497c063060547d7050f32b6c0eca89f	b81828adde9d67528d72fd901aa53d5	57cac5b0c46380bcd0e09d102b73014d	1
null	null	null	a88206c333a45c7ca22e2a6a24dd02a8	33900010111111	0f5ef958486740605d4408616056536d	7c45e9f20c9ee2019181d9e32a66f41e	6ea9ab1baa0efb9e19094440c317e21b	3
null	null	null	604a42e2b2e97d255ad93bca92babedb	33900000110011	9c323b61c472ae88783e7cf24d35cd27	7fe2c7b09f4369739de186e3b45b7e7b	9877c327c2b34bcfb9c9da8f05dd5cc15	1
null	null	null	4231ca76e86624f2330858f8497c8507	33900000110011	86ec2ffa2bcfcf8a087199746124641a	7fe2c7b09f4369739de186e3b45b7e7b	9877c327c2b34bcfb9c9da8f05dd5cc15	1
null	null	null	49404ac22b3c21b1e4	33900000110011	472c020b42510e4d72	b29fe0b750f6e4e1b80	0877c327c2b34bcfb9c	1

Output is truncated to 102400 bytes. Learn more about ZEPPELIN_INTERPRETER_OUTPUT_LIMIT

因为是 LEFT JOIN，所以维表中不存在的数据会以 NULL 补全。

再看一眼 DAG 图：



大家看一下画框的地方，能看到这边是使用的维表关联 LookupJoin。

如果大家 SQL 语句写错了，丢了 for system_time as of a.p，那么 DAG 图就会变成这样：



这种就不是维表 JOIN 其实更像是流和批在 JOIN。

写在最后

Hive Streaming 的完善意味着打通了流批一体的最后一道壁垒，既可以做到历史数据的 OLAP 分析，又可以实时吐出结果，这无疑是 ETL 开发者的福音，想必接下来的日子，会有更多的企业完成他们实时数仓的建设。

参考文档:

[1]<https://ci.apache.org/projects/flink/flink-docs-release-1.11/dev/table/hive/>

[2]<https://github.com/apache/zeppelin/blob/master/docs/interpreter/flink.md>
