Flink精准去重

概述

- 为啥需要去重
  - 在某些情况下，由于上游的作业不是端到端的exactly-once，在上游出现问题自动failover的时候，该任务的sink端很大可能出现重复数据；这些重复数据又会影响下游的聚合作业（如SUM，COUNT）。所以，我们的作业需要去重完再进行计算
- 去重方法
  - TopN（Flink官网推荐的方式）
  - Group By
  - UDTF（维表去重）
- 各自优缺点
  - 前两者纯内存去重，效率高，速度快，使用简单；但是一旦任务KILL再启动就会有状态丢失，那么下游计算过的数据又会计算一次；同样，因为使用到STATE，那么就要配置合理的STATE TTL，不然STATE会越来越大，对checkPoint会有影响
  - 第三个的问题在于需要用户手写UDTF，难度较高（写UDTF可以参考Flink UDF）；其次状态保存在存储中间件中，查询和插入都是同步操作，效率较低，受网络和中间件的影响比较大；最后，如果上游发生撤回流，无法撤回中间件中的数据，会导致数据被错误过滤；也有优点：任务重启之后，不会影响下游数据的准确性。
- 下面我们开始依次看看怎么用的

TopN

- 语法：

```
SELECT [column_list]
FROM (
   SELECT [column_list],
     ROW_NUMBER() OVER ([PARTITION BY col1[, col2...]]
       ORDER BY time_attr [asc|desc]) AS rownum
   FROM table_name)
WHERE rownum = 1
```

- 目前`time_attr`只支持`PROCTIME`，按`ASC`意味着取第一条，其他的而丢掉；`DESC`则是最新一条，如果一条数据重复来多次，将会发出多次撤回流
- `PARTITION BY col1`可以简单理解为主键
- 会产生撤回流，如果将去重的结果直接插入结果表，且结果表支持`UPSERT`。那么，会把`PARTITION BY`之后的字段视为主键
- 如果插入的结果表不包含`PARTITION BY`字段，将无法识别主键，会抛出`UpsertStreamTableSink requires that Table has a full primary keys if it is updated.`的异常，请使用`GROUP BY`强行把流变成撤回流
- 下面是简单实例

```
String topN_lastRow = "insert into t2 " +
"select user_id ,item_id ,behavior ,category_id ,ts from " +
" (select *,ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY p DESC ) AS rn fro
m t1) " +
" where rn = 1";

tEnv.sqlUpdate(topN_lastRow);

String topN_firstRow = "insert into t2 " +
"select user_id ,item_id ,behavior ,category_id ,ts from " +
" (select *,ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY p ASC ) AS rn from
 t1) " +
" where rn = 1";

tEnv.sqlUpdate(topN_firstRow);
```

- `FIRST_ROW`的效率会高一点，因为STATE中只存储了KEY数据，所以性能较优
- `LAST_ROW`的效率弱一点，但会略优于下面说的`LAST_VALUE`函数

Group By

- 语法：

```
SELECT LAST_VALUE|FIRST_VALUE(col1),LAST_VALUE|FIRST_VALUE(col2)...
FROM table_name
GROUP BY key1,key2...
```

- 相对于`TopN`看上去更简单易懂些
- `FIRST_VALUE()`表示取第一次出现的
- `LAST_VALUE`表示取最后一次出现的
- 性能比`TopN`较弱
- 同样支持`UPSERT`

```
String groupBy_lastRow = "insert into t3 \n" +
"select \n" +
" user_id \n" +
",last_value(item_id) as item_id \n" +
",last_value(behavior) as behavior \n" +
",last_value(category_id) as category_id \n" +
",last_value(ts) as ts \n" +
"from t1 group by user_id";

tEnv.sqlUpdate(groupBy_lastRow);

String groupBy_firstRow = "insert into t3 \n" +
"select \n" +
" user_id \n" +
",first_value(item_id) as item_id \n" +
",first_value(behavior) as behavior \n" +
",first_value(category_id) as category_id \n" +
",first_value(ts) as ts \n" +
"from t1 group by user_id";

tEnv.sqlUpdate(groupBy_firstRow);
```

UDTF

- 这是今天的重头戏
- 开始之前说一下我们以前的去重做法
  - 先是用`TopN`去重
  - 再直接Join维表，看看数据在维表里面存不存在；存在就不下发，不存在再下发
  - 将数据KEY输出到HBASE
  - 再对数据进行聚合
- 优缺点也是有不少
  - 优点：`TopN`速度快，性能好；Join维表也是异步的；任务重启也不会有影响；如果上游发生撤回，那么HBASE中的数据也会被撤回
  - 缺点则是会有事务问题：如果KEY1对应的数据，插入HBASE维表的动作还没完成，此时，上面`TopN`的STATE也正好失效，而正好又从消息中间件中读到了一条KEY1对应的数据，那么，这条数据将没法被正确过滤。
- 考虑到事务的问题，我们决定使用UDTF来做去重
- UDTF中查数和写数的操作都是同步的，自然解决了事务的问题；当然，前提是你的KEY都得在同一个分区，不然还是会有事务的问题，这里建议大家可以先用`keyBy`算子，将数据打到同一个分区；纯FLINK SQL的话只能先用`TopN`或`Group By`先将数据打到同一个分区；最好的解决方法还是上游把数据写入消息中间件的时候，就根据KEY分区，保证相同KEY写入同一个分区，这样我们的下游就不用了多做处理了。

- 解决了事务的问题，带来的则是性能上的损耗。主要还是在于和存储中间件的通信，有得必有失；其次上游撤回了，我们没法撤回已经写入中间件的数据。解决方法就是上游直接对接消息，那么也不会有撤回的事件
- 当然，如果是发生了failover，也没法删除中间件的数据，这个解决方案还没想好，大家有兴趣可以一起讨论
- 下面贴一下完整的UDTF代码，老规矩，讲解都在注释里面了

```
package udf;

import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.TableFunction;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;

public class DeduplicationUDTF extends TableFunction<Tuple2<String, Integer>> {

    private static final Logger log = LoggerFactory.getLogger(DeduplicationUDTF
.class);

    private transient Connection hConnection;
    private transient Table table;
    private transient BufferedMutator mutator;

    private final String zkIp;
    private final String zkPort;

    private final String tableName;
    private final String cf;
    private final String col;

    public DeduplicationUDTF(String zkIp, String zkPort, String tableName, Stri
ng cf, String col) {
        this.zkIp = zkIp;
        this.zkPort = zkPort;
        this.tableName = tableName;
        this.cf = cf;
        this.col = col;
```

```
    }

    public void eval(String rowkey) {
        Get get = new Get(Bytes.toBytes(rowkey));



        try {
            Result result = table.get(get);

            //说明hbase中已有数据，这条可以过滤，所以给这条数据JOIN的字段is_duplicate赋
值1
            if (!result.isEmpty()) {

                collect(Tuple2.of(rowkey, 1));

            } else {

                //没有的话，就把这条数据放到hbase，然后再这条数据JOIN的字段is_duplicate
赋值1，最后下发
                //注意，必须先入hbase，而且hbase连接得是同步的，不然会有数据还没插入hbase
，又有新数据到了的情况
                Put put = new Put(Bytes.toBytes(rowkey));
                put.addColumn(Bytes.toBytes(cf), Bytes.toBytes(col), Bytes.toBy
tes("1"));

                mutator.mutate(put);
                mutator.flush();

                collect(Tuple2.of(rowkey, -1));

            }



        } catch (IOException e) {
            log.error("get from hbase error! ", e);
            e.printStackTrace();
        }

    }

    @Override
    public void open(FunctionContext context) throws Exception {
```

```
        super.open(context);
        //初始化hbase 连接
        Configuration config = HBaseConfiguration.create();
        config.set("hbase.zookeeper.quorum", zkIp);
        config.set("hbase.zookeeper.property.clientPort", zkPort);

        hConnection = ConnectionFactory.createConnection(config);

        table = hConnection.getTable(TableName.valueOf(tableName));
        BufferedMutatorParams params = new BufferedMutatorParams(TableName.valu
eOf(tableName))
                .writeBufferSize(-1);
        mutator = hConnection.getBufferedMutator(params);

    }

    @Override
    public void close() {
        //所有的流都关闭
        try {
            super.close();
        } catch (Exception e) {
            log.error("super class close error!", e);
            throw new RuntimeException(e);
        }

        if (table != null) {
            try {
                table.close();
            } catch (IOException e) {
                log.error("table close error!", e);
                throw new RuntimeException(e);
            }
        }

        if (mutator != null) {
            try {
                mutator.close();
            } catch (IOException e) {
                log.error("mutator close error!", e);
                throw new RuntimeException(e);
            }
        }
```

```
        if (hConnection != null) {
            try {
                hConnection.close();
            } catch (IOException e) {
                log.error("Connection close error!", e);
                throw new RuntimeException(e);
            }
        }

    }
}
```

- 下面是主类的代码

```
package tutorial;

import org.apache.flink.table.api.Table;
import org.apache.flink.types.Row;
import udf.DeduplicationUDTF;

import static util.FlinkConstant.*;

public class FlinkSql08 {

    public static final String KAFKA_TABLE_SOURCE_DDL = "" +
            "CREATE TABLE t1 (\n" +
            "    user_id BIGINT,\n" +
            "    item_id BIGINT,\n" +
            "    category_id BIGINT,\n" +
            "    behavior STRING,\n" +
            "    ts BIGINT,\n" +
            "    p AS PROCTIME()" +
            ") WITH (\n" +
            "    'connector.type' = 'kafka',  -- 指定连接类型是kafka\n" +
            "    'connector.version' = '0.11',  -- 与我们之前Docker安装的kafka版本
要一致\n" +
            "    'connector.topic' = '08_test', -- 之前创建的topic \n" +
            "    'connector.properties.group.id' = '08_test', -- 消费者组，相关概
念可自行百度\n" +
            "    'connector.startup-mode' = 'earliest-offset',  --指定从最早消费\n
" +
```

```
            "    'connector.properties.zookeeper.connect' = 'localhost:2181',
-- zk地址\n" +
            "    'connector.properties.bootstrap.servers' = 'localhost:9092',
-- broker地址\n" +
            "    'format.type' = 'json'  -- json格式，和topic中的消息格式保持一致\n"
 +
            ")";

    public static final String MYSQL_TABLE_SINK_DDL = "" +
            "CREATE TABLE `t2` (\n" +
            "  `user_id` BIGINT  ,\n" +
            "  `item_id` BIGINT  ,\n" +
            "  `behavior` STRING  ,\n" +
            "  `category_id` BIGINT  ,\n" +
            "  `ts` BIGINT   \n" +
            ")WITH (\n" +
            "  'connector.type' = 'jdbc', -- 连接方式\n" +
            "  'connector.url' = 'jdbc:mysql://localhost:3306/test', -- jdbc的ur
l\n" +
            "  'connector.table' = 'user_behavior',  -- 表名\n" +
            "  'connector.driver' = 'com.mysql.jdbc.Driver', -- 驱动名字，可以不填
，会自动从上面的jdbc url解析 \n" +
            "  'connector.username' = 'root', -- 顾名思义 用户名\n" +
            "  'connector.password' = '123456' , -- 密码\n" +
            "  'connector.write.flush.max-rows' = '5000', -- 意思是攒满多少条才触发
写入 \n" +
            "  'connector.write.flush.interval' = '2s' -- 意思是攒满多少秒才触发写入
；这2个参数，无论数据满足哪个条件，就会触发写入\n" +
            ")";

    public static final String ES_TABLE_SINK_DDL = "" +
            "CREATE TABLE `t3` (\n" +
            "  `user_id` BIGINT  ,\n" +
            "  `item_id` BIGINT  ,\n" +
            "  `behavior` STRING  ,\n" +
            "  `category_id` BIGINT  ,\n" +
            "  `ts` BIGINT   \n" +
            ")WITH (\n" +
            "  'connector.type' = 'elasticsearch', -- required: specify this ta
ble type is elasticsearch\n" +
            "  'connector.version' = '6',          -- required: valid connector
 versions are \"6\"\n" +
            "  'connector.hosts' = 'http://127.0.0.1:9200',  -- required: one o
```

```
r more Elasticsearch hosts to connect to\n" +
            "  'connector.index' = 'user',       -- required: Elasticsearch ind
ex\n" +
            "  'connector.document-type' = 'user',  -- required: Elasticsearch
document type\n" +
            "  'update-mode' = 'upsert',            -- optional: update mode wh
en used as table sink.           \n" +
            "  'connector.flush-on-checkpoint' = 'false',   -- optional: disabl
es flushing on checkpoint (see notes below!)\n" +
            "  'connector.bulk-flush.max-actions' = '1',  -- optional: maximum
number of actions to buffer \n" +
            "  'connector.bulk-flush.max-size' = '1 mb',  -- optional: maximum
size of buffered actions in bytes\n" +
            "  'connector.bulk-flush.interval' = '1000',  -- optional: bulk flu
sh interval (in milliseconds)\n" +
            "  'connector.bulk-flush.backoff.max-retries' = '3',  -- optional:
maximum number of retries\n" +
            "  'connector.bulk-flush.backoff.delay' = '1000',    -- optional: d
elay between each backoff attempt\n" +
            "  'format.type' = 'json'   -- required: Elasticsearch connector re
quires to specify a format,\n" +
            ")";

    public static void main(String[] args) throws Exception {

        tEnv.sqlUpdate(KAFKA_TABLE_SOURCE_DDL);

        tEnv.sqlUpdate(MYSQL_TABLE_SINK_DDL);

        tEnv.sqlUpdate(ES_TABLE_SINK_DDL);

//        String topN_lastRow = "insert into t2 " +
//                "select user_id ,item_id ,behavior ,category_id ,ts from " +
//                " (select *,ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY
p DESC ) AS rn from t1) " +
//                " where rn = 1";
//
//        tEnv.sqlUpdate(topN_lastRow);

//        String topN_firstRow = "insert into t2 " +
//                "select user_id ,item_id ,behavior ,category_id ,ts from " +
//                " (select *,ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY
p ASC ) AS rn from t1) " +
```

```
//                " where rn = 1";
//
//        tEnv.sqlUpdate(topN_firstRow);

//        String groupBy_lastRow = "insert into t3 \n" +
//                "select \n" +
//                " user_id \n" +
//                ",last_value(item_id) as item_id \n" +
//                ",last_value(behavior) as behavior \n" +
//                ",last_value(category_id) as category_id \n" +
//                ",last_value(ts) as ts \n" +
//                "from t1 group by user_id";
//
//        tEnv.sqlUpdate(groupBy_lastRow);

//        String groupBy_firstRow = "insert into t3 \n" +
//                "select \n" +
//                " user_id \n" +
//                ",first_value(item_id) as item_id \n" +
//                ",first_value(behavior) as behavior \n" +
//                ",first_value(category_id) as category_id \n" +
//                ",first_value(ts) as ts \n" +
//                "from t1 group by user_id";
//
//        tEnv.sqlUpdate(groupBy_firstRow);

        tEnv.registerFunction("deDuplication",new DeduplicationUDTF("127.0.0.1"
,"2182","test","cf","col"));

        //给每条数据打上标签，is_duplicate为1的表示为重复，-1表示没有重复，也就是第一条到
达的数据
        Table table = tEnv.sqlQuery("select a.* ,b.* from t1 a , \n" +
                "LATERAL TABLE(deDuplication(concat_ws('',cast(a.user_id as var
char)))) as b(rowkey,is_duplicate)");

        tEnv.toAppendStream(table,Row.class).print("没用where过滤").setParallelis
m(1);

        Table where = table.where("is_duplicate = -1");

        //这个应该只会输出10条数据，而且主键user_id都是唯一，否则就有误
        //大家多次测试的时候记得删除HBASE中的数据
        tEnv.toAppendStream(where,Row.class).print("用where过滤").setParallelism
```

```
(1);


        env.execute("FlinkSql08");

    }
}
```

- 大家看看UDTF打印输出的结果就一目了然了

后话

- ES和MYSQL的建表语句都在我代码的`resource`目录下，github拉一下我的代码吧，`pom.xml`也记得加一下依赖
- 同样，往KAFKA发消息的代码也在GIT里面，全路径是`util.KafkaProducerUtil`，这个类`啤酒鸭`提供的，哈哈哈我懒得写了
- 另外，还是希望大家能给我多多点赞+收藏+关注，要是能帮我宣传宣传就更好了
- 最后，下一章更新内容还不确定，大家想看什么可以交流。下期再见~