Flink入坑指南 第四章：SQL中的经典操作 Group By+Agg

简介

Group By + Agg这个最经典的SQL使用方式。Group By是SQL中最基础的分组操作，agg的全称是aggregation(聚合操作)，是一类SQL算子的统称，Flink中最常用的Agg操作有COUNT/SUM/AVG等，详情参见Flink支持的聚合操作列表。在实际使用中，Group By+Agg绝大部分场景下都会一起出现。作为最常用的SQL模式，学习好这种模式的最优写法，也就非常重要了。本章从两个需求开始，进一步了解一下Group By + Agg模式的最优写法，及实时计算产品/Alibaba Flink版本中的部分优化.

需求

上一章中，小明已经把第一个需求完成了，同时也了解了持续查询，state等流计算中的基础概念。熟悉了Flink/实时计算的基础用法之后，小明开始着手开发其他的需求：

1. 从0点开始，每个类目的成交额
2. 从0点开始，每个店铺的uv/pv
3. 从0点开始，每个用户点击了多少商品，多少店铺

Group By + Agg

先看需求1：从0点开始，每个类目的成交额。进入Flink的原始数据结构如下：

ctime	category_id	shop_id
2018-12-04 15:44:54	cat_01	shop_01
2018-12-04 15:45:46	cat_02	shop_02
2018-12-04 15:46:11	cat_01	shop_03

FlinkSQL代码如下，看上去与传统数据库/批处理的SQL相同：

```
SELECT
    date_format(ctime, '%Y%m%d') as cdate, -- 将数据从时间戳格式（2018-12-04 15:
       category_id,
    sum(price) as category_gmv
```

```
FROM src
GROUP BY date_format(ctime, '%Y%m%d'), category_id; --按照天做聚合
```

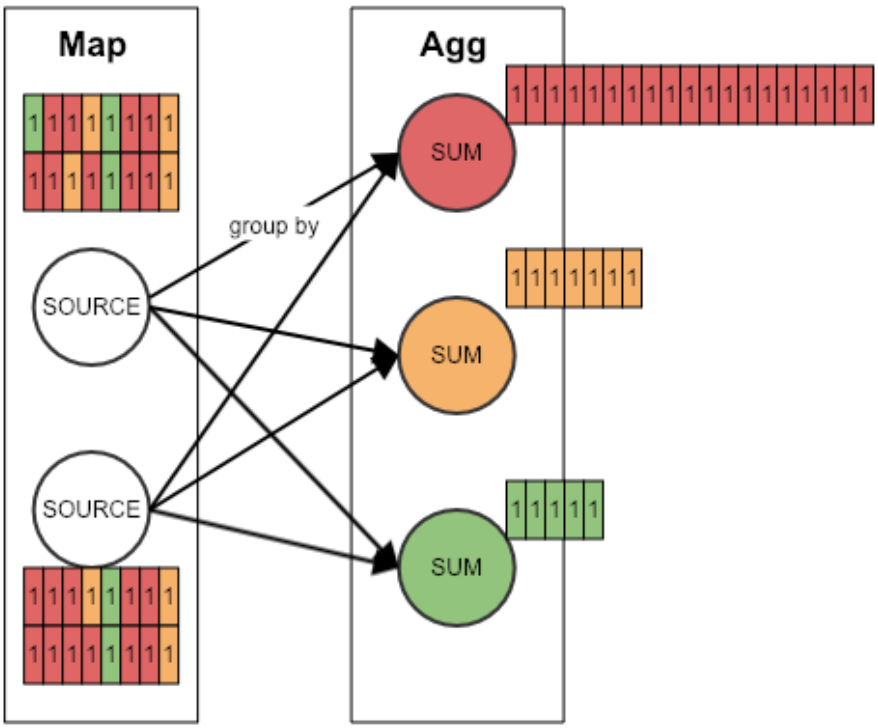
以这个例子入手，着重说明Group By+Agg通用模式的两个问题：

1. 计算特点
2. 常见问题及解法

Group by+Agg模式在底层的有一些特点：

1. Group by分组操作，会产生数据shuffle
2. **按Key的agg操作，最终都需要落到同一个物理进程上才能保证计算的正确性**

以这个最简单SQL为例，其数据流程图如下，不同颜色代表不同的category_id：



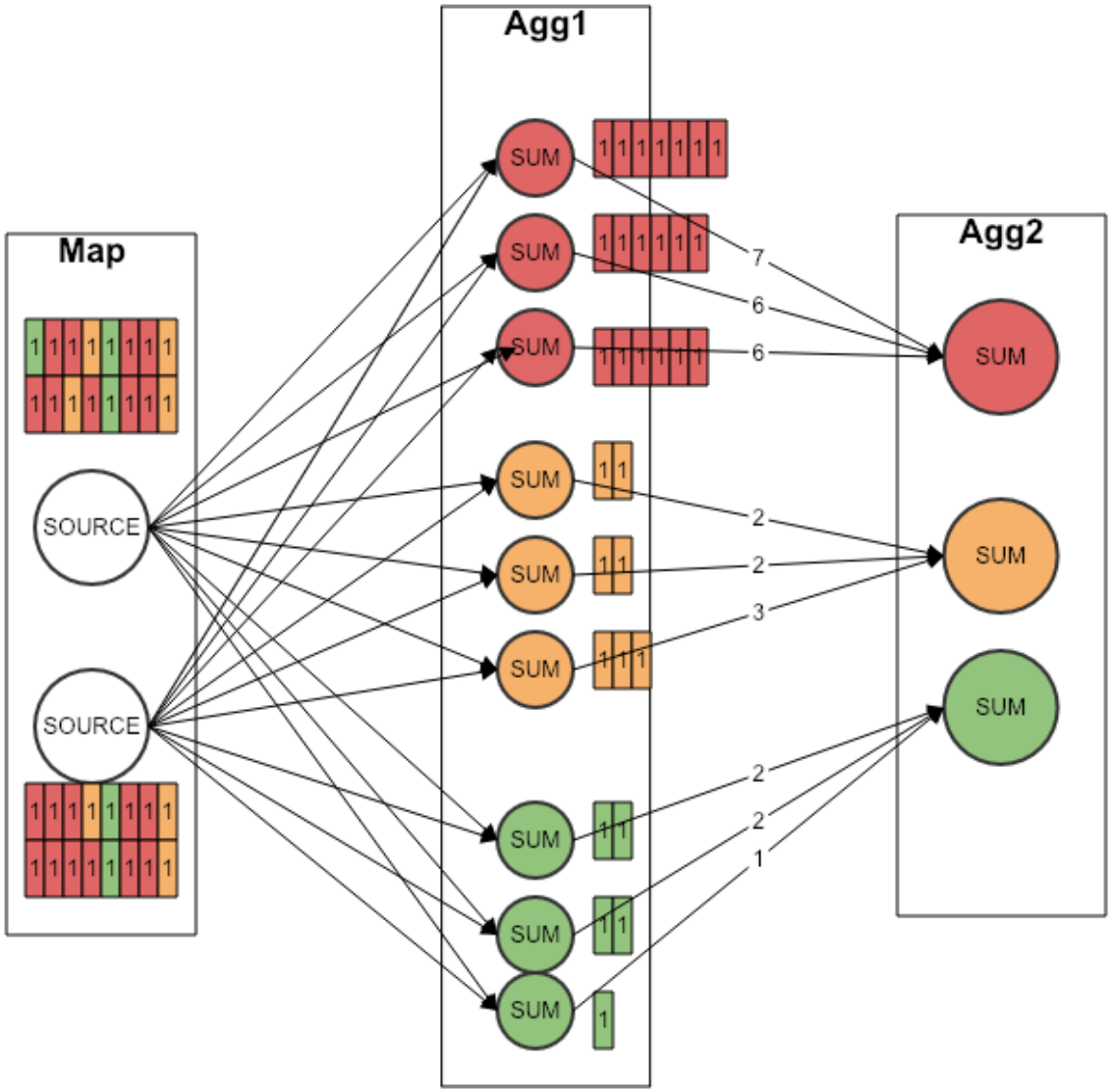
数据源进来的数据先经过group by进行分组，同一个key的数据被分到同一个worker上之后再进行聚合操作。特点2就决定了，Group By + Agg 模式中，SQL作业性能与数据分布非常相关，如果数据中存在__数据倾斜__，也就是某个key的数据异常的多，那么某个聚合节点就会成为瓶颈，作业就会有明显的反压及延时现象。

为了解决这个问题，就需要将堵住的聚合节点进行拆分，优化后的SQL如下：

```
SELECT cdate,category_id,sum(category_gmv_p) as category_gmv
FROM(
    SELECT
```

```
        date_format(ctime, '%Y%m%d') as cdate, -- 将数据从时间戳格式（2018-12-04
        category_id,          sum(price) as category_gmv_p
    FROM src
    GROUP BY category_id, mod(hash_code(FLOOR(RAND(1)*1000), 256),date_format
)
GROUP BY cdate,category_id
```

SQL中做了将一个Group By+Agg拆称了两个，子查询里按照category_id和mod(hash_code(FLOOR(RAND(1)*1000), 256)分组，将同一个category_id上的数据打散成了256份，先做一层聚合。外层Group By+Agg，将子查询聚合后的结果再次做聚合。这样通过两层聚合的方式，即可大大缓解某聚合节点拥堵的现象。其数据流程图如下：



如果用户用的是开源Flink1.7版本，如果作业出现数据倾斜情况，就需要按以上方法对SQL进行改造，以提高作业吞吐，降低由于数据倾斜造成的业务延时。

相关函数用法，Floor, Rand, Hash_Code

在实时计算产品__使用 Flink版本，针对这种情况做了特殊优化，使用Local-Global Agg的方式完美解决了Group By+Agg模式中的数据倾斜问题，用户使用第一种（最简单）的SQL即可。__关于Local-Global Agg原理方面的介绍，后续会有专门文章，敬请期待。

GroupBy+单Distinct Agg

第二个需求：计算从0点开始，每个店铺的uv/pv
原始数据：

ctime	category_id	shop_id	item_id
2018-12-04 15:44:54	cat_01	shop_01	item_01
2018-12-04 15:45:46	cat_02	shop_02	item_02
2018-12-04 15:46:11	cat_01	shop_03	item_03

其中action有三种：

- 0: 浏览
- 1: 点击
- 2: 加购
- 3: 购买

经过这段时间的学习，小明三两下就写出SQL：

```
SELECT
    date_format(ctime, '%Y%m%d') as cdate, -- 将数据从时间戳格式（2018-12-04 15:
       shop_id,
    count(distinct uid) as shop_uv, -- shop uv
    count(uid) as shop_pv -- show pv
FROM src
GROUP BY date_format(ctime, '%Y%m%d'), shop_id; --按照天做聚合
```

同样，按照上节所述，如果这个作业出现了数据倾斜的现象，就需要将SQL优化为：

```
select
    cdate,
    shop_id,
    sum(shop_uv_partial) as shop_uv,
    sum(shop_pv_partial) as shop_pv
from (
    select
        date_format(ctime, '%Y%m%d') as cdate, -- 将数据从时间戳格式（2018-12-04
```

```
        shop_id,
        count(distinct uid) as shop_uv_partial,
        count(uid) as shop_pv_partial
    from src
    group by shop_id, mod(hash_code(uid), 256),date_format(ctime, '%Y%m%d')
)
group by cdate,shop_id
```

本例子中，将原始SQL中的一层查询，拆成了两层查询。内层子查询，按照shop_id和mod(hash_code(uid),256)做聚合，将同一个shop_id的数据打散到多个节点中。外层查询，将子查询聚合后的结果，再按shop_id聚合。通过两层聚合即可大大缓解数据倾斜情况下聚合节点的压力。

Group By+Agg场景与Group By+Distinct Agg场景的主要区别，在于state中存储的数据。上一章中提到过，Flink是增量计算，state中会保存增量数据，比如上次SUM的值等等，但是在DISTINCT计算过程中，就需要保留所有的distinct的key，在本例子中，就是uid。且在每一次计算过程中，都要查询当前state中是否有同一个uid，并计数。因此在大数据量情况下distinct节点往往成为Flink作业的瓶颈。需要通过扩并发等方式解决。

同样，在实时计算产品使用 Flink版本，针对这种情况做了特殊优化，使用Partial-Final Agg的方式完美解决了Group By+Distinct Agg模式中的数据倾斜问题，用户使用第一种（最简单）的SQL即可。关于Partial-Final Agg原理方面的介绍，后续会有专门文章，敬请期待。

Group By+多Distinct Agg

第三个需求：从0点开始，每个用户点击了多少商品，多少店铺，以及该用户总点击item次数。原始数据如下：

ctime	category_id	shop_id	item_id
2018-12-04 15:44:54	cat_01	shop_01	item_01
2018-12-04 15:45:46	cat_02	shop_02	item_02
2018-12-04 15:46:11	cat_01	shop_03	item_03

经过一番思索，小明写出了如下SQL：

```
SELECT UDTF
    date_format(ctime, '%Y%m%d') as cdate, -- 将数据从时间戳格式（2018-12-04 15:
       uid,
    count(distinct shop_id) as shop_cnt,
    count(distinct item_id) as item_cnt,
    count(item_id) as click_cnt
```

```
FROM src
GROUP BY date_format(ctime, '%Y%m%d'), uid;
```

需求2相比，SQL中distinct个数变成了多个，这种情况下要优化SQL就更复杂了。有一种比较原始的做法：

1. 先使用UDTF，将原始数据一行拆成多行，每行添加n+1列，n为distinct的个数。n列分别对distinct的值做hash。具体例子如下：

ctime	category_id	shop_id	item_id	uid	action	hash_sho
2018-12-04 15:44:54	cat_01	shop_01	item_01	10001		hash(shop
2018-12-04 15:44:54	cat_01	shop_01	item_01	10001		null
2018-12-04 15:44:54	cat_01	shop_01	item_01	10001		null
2018-12-04 15:45:46	cat_02	shop_02	item_02	10001		hash(shop
2018-12-04 15:45:46	cat_02	shop_02	item_02	10001		null
2018-12-04 15:45:46	cat_02	shop_02	item_02	10001		null
2018-12-04 15:46:11	cat_01	shop_03	item_03	10002		hash(shop
2018-12-04 15:46:11	cat_01	shop_03	item_03	10002		null
2018-12-04 15:46:11	cat_01	shop_03	item_03	10002		null

1. 在SQL中，先在子查询中分别计算各指标的count值，在外层再做一层sum即可，SQL示例如下：

```
select
    cdate,
    uid,
    sum(shop_cnt_p) as shop_cnt,
    sum(item_id_p) as item_id_cnt,
    sum(item_cnt_p) as item_cnt
from (
    select
        date_format(ctime, '%Y%m%d') as cdate,
        uid,
        count(distinct shop_id) filter (where flag = flag0) as shop_cnt_p,
        count(distinct item_id) filter (where flag = flag1) as item_id_p ,
        sum(item_id) filter (where flag = flag2) as item_cnt_p
    from Expand_T
    group by uid, hash_user, hash_shop, date_format(ctime, '%Y%m%d')
    )
```

```
group by uid
```

这种问题可以解决多个distinct中的数据倾斜问题，但是会增加sql复杂度，并且计算过程中数量会膨胀，并且占用更多资源。

同样，在实时计算产品使用 Flink版本，针对这种情况做了特殊优化，使用Partial-Final Agg+Incremental Agg的方式完美解决了Group By+多个Distinct Agg模式中的数据倾斜问题，用户不需要在SQL上做拆分。关于Partial-Final Agg+Incremental Agg原理方面的介绍，后续会有专门文章，敬请期待。

数据倾斜相关配置

在使用实时计算产品时，如果遇到数据倾斜问题，可以增加以下配置，即可解决，不需要手动进行SQL优化。

```
# 开启5秒的microbatch
blink.microBatch.allowLatencyMs=5000
blink.miniBatch.allowLatencyMs=5000
blink.miniBatch.size=20000
# Local 优化，默认已经开启
# blink.localAgg.enabled=true
# 开启 Partial 优化，解决count distinct热点
blink.partialAgg.enabled=true
# Incremental 优化，默认已经开启
# blink.incrementalAgg.enabled=true
```