天善智能
TIANSHAN SOFT

开发性能调优

讲师：watermelon

目录

- 数据倾斜调优
- Hive合并小文件
- 使用Spark缓存
- 开发中间表

数据倾斜调优

数据倾斜原因

常见表现：在hive中 map阶段早就跑完了，reduce阶段一直卡在99%。很大情况是发生了数据倾斜，整个任务在等某个节点跑完。

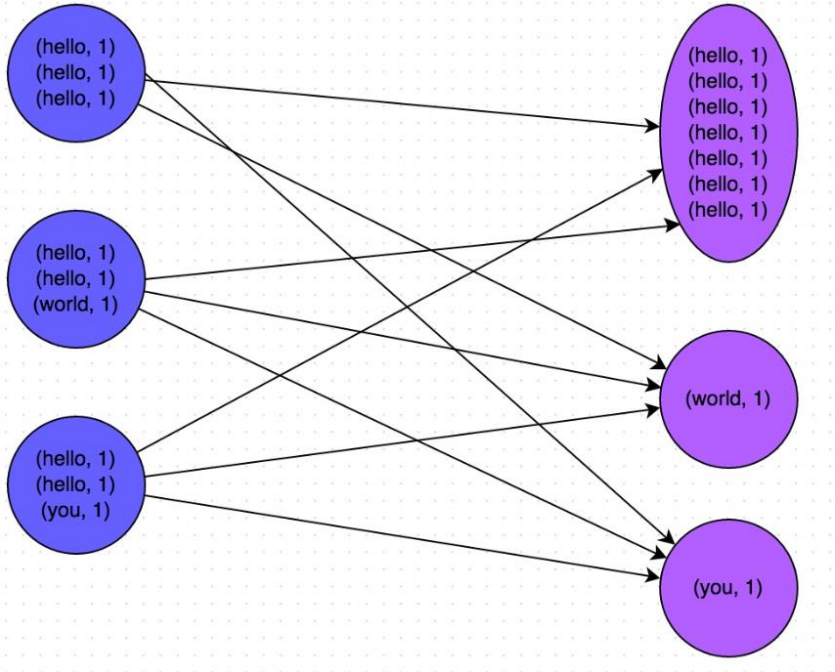
在spark中大部分的task执行的特别快，剩下的一些task执行的特别慢，要几分钟或几十分钟才执行完一个task

Hive中大表join的时候，容易产生数据倾斜问题，spark中产生shuffle类算子的操作，groupbykey、reducebykey、join等操作会引起数据倾斜。

通过stage去定位

数据倾斜原因：

在进行shuffle的时候，*必须将各个节点上相同的key拉取到某个节点上的一个task来进行处理*，比如按照key进行聚合或join等操作。此时如果某个key对应的数据量特别大的话，就会发生数据倾斜。比如大部分key对应10条数据，但是个别key却对应了100万条数据，那么大部分task可能就只会分配到10条数据，然后1秒钟就运行完了；但是个别task可能分配到了100万数据，要运行一两个小时



数据倾斜解决方案

解决方法1：直接过滤掉那些引起倾斜的key

例如
```
select key1,count(*) as num_1
  from dw.table_a
group by key1
order by num_1 desc limit 20
```

```
select key2,count(*) as num_2
  from dw.table_b
group by key2
order by num_2 desc limit 20
```

比如说，总共有100万个key。只有2 个key，是数据量达到10 万的。其他所有的key，对应的数量都是几十，这样join后会引起倾斜。这个时候，自己可以去取舍，如果业务和需求可以理解和接受的话，在从hive 表查询源数据的时候，直接在sql 中***用where 条件，过滤掉某几个key***。那么这几个原先有大量数据，会导致数据倾斜的key，被过滤掉之后，那么在的spark作业中，自然就不会发生数据倾斜了。

解决方法2：Hive ETL做处理

导致数据倾斜的是Hive表。如果该Hive表中的数据本身很不均匀（比如某个key对应了100万数据，其他key才对应了10条数据），而且应用中需要频繁使用Spark对Hive表执行分析操作时，可以使用Hive ETL去做一个预处理

实现方式

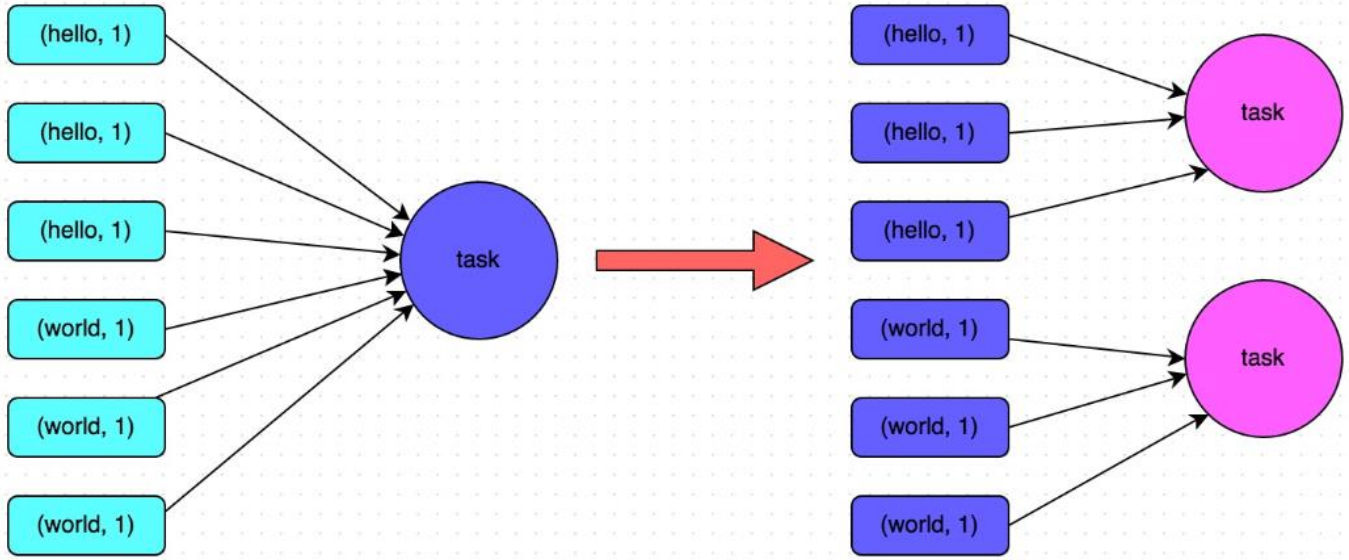
通过Hive ETL预先对数据按照key进行聚合，或者是预先和其他表进行join，然后在Spark作业中针对的数据源就不是原来的Hive表了，而是预处理后的Hive表。此时由于数据已经预先进行过聚合或join操作了，那么在Spark作业中也就不需要使用原先的shuffle类算子执行这类操作了。Hive ETL中进行group by或者join等shuffle操作时，还是会出现数据倾斜，导致Hive ETL的速度很慢。我们只是把数据倾斜的发生提前到了Hive ETL中

数据倾斜解决方案

解决方法3：提高shuffle操作并行度

在对RDD执行shuffle算子时，给shuffle算子传入一个参数，比如reduceByKey(1000)，该参数就设置了这个shuffle算子执行时shuffle read task的数量。对于Spark SQL中的shuffle类语句，比如group by、join等，需要设置一个参数，即spark.sql.shuffle.partitions，该参数代表了shuffle read task的并行度，该值默认是200，对于很多场景来说都有点过小

原理：增加shuffle read task的数量，可以***让原本分配给一个task的多个key分配给多个task，从而让每个task处理比原来更少的数据***。举例来说，如果原本有5个key，每个key对应10条数据，这5个key都是分配给一个task的，那么这个task就要处理50条数据。而增加了shuffle read task以后，每个task就分配到一个key，即每个task就处理10条数据，那么自然每个task的执行时间都会变短了



参考文章：美团技术团队博客

Hive合并小文件

合并小文件

- hadoop fs –ls /文件地址 ： 可以查看Hive表中每个数据文件的大小。小文件一般都几k、几十k的。
- HDFS用于存储大数据的文件，如果Hive中存在过多的小文件会给namenode带来较大的性能压力。同时小文件过多时会影响spark 中job的执行。为了提高namenode的使用效率，在向hdfs加载文件时需要提前对小文件进行合并；
- Spark将job转换成多个task，从hive中拉取数据，对于每个小文件也要分配一个task去处理，每个task只处理很少的数据，这样会起上万个task，非常影响性能；
- 处理大量小文件的速度远远小于处理同样大小的大文件速度，Task启动将耗费大量时间在启动task和释放task上。

为了防止生成小文件，在hive ETL的时候可以通过配置参数在MapReduce过程中合并小文件。

一般在对ods层日志数据进行处理时，如果小文件过多，需要重新ETL合并小文件再重新写入

输出合并

合并输出小文件，以减少输出文件的大小，可通过如下参数设置：

```
set  hive.merge.mapfiles=true;      // map only job结束时合并小文件
set  hive.merge.mapredfiles=true;     // 合并reduce输出的小文件
set  hive.merge.size.per.task=64000000;      //合并之后的每个文件大小64M
```

使用Spark缓存(cache、persist、checkpoint)

三者的区别

Spark相比hadoop的优势之一在于可以不把中间计算的数据持久化到磁盘上，在内容中进行存储

Cache与persist区别

cache底层调用的是persist方法，存储等级为 memory only

Persisit与cache的主要区别是persist可以自定义存储级别StorageLevel。cache只使用memory only。

- MEMORY_ONLY：只存在内存中；
- DISK_ONLY：只存在磁盘中；
- MYMORY_AND_DISK：先存在内存中，内存不够的话存在磁盘中；
- OFF_HEAP：存在堆外内存中；

Persist与checkpoint区别

Persist虽然可以将partition持久化到磁盘上，但是该partition由blockmanager管理，一旦executor所在进程结束，blockmanager也会结束，被缓存到磁盘上的RDD也会被清空。而checkpoint将RDD持久化到HDFS中，如果不是手动删除的话，是一直存在的。

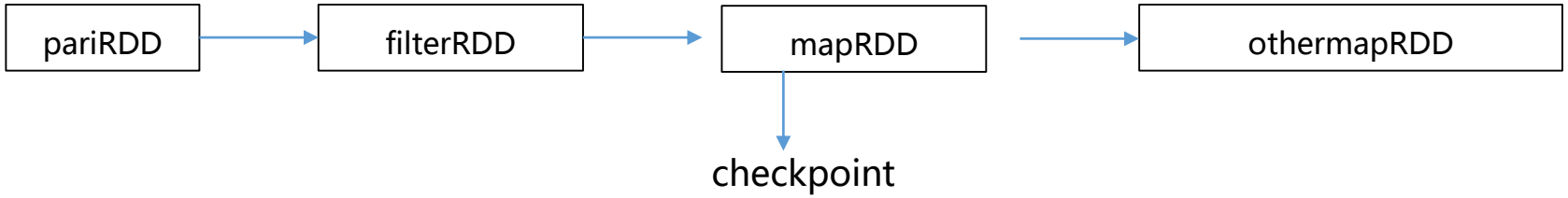
脚本中的应用-cache

```
def main():
    start_date = sys.argv[1]
    start_date_str = str(start_date)
    date_str = datetime.datetime.strftime(datetime.date.today()-datetime.timedelta(days=1),'%Y-%m-%d')
    format = "%Y%m%d"
    target_table = 'dw.profile_tag_user'


    # 用户RFM维度数据 (用户最后一次购买时间非空)
    user_rfm_info = "  select t1.user_id,                                                                                                  \
                              t2.country,                                                                                                  \
                              t1.last_1y_paid_orders,                                                                                      \
                              t1.last_1y_paid_order_amount,                                                                                \
                              concat(substr(t1.last_order_paid_time,1,4),'-',substr(t1.last_order_paid_time,5,2),'-',substr(t1.last_order_paid_time,7,2)
                         from dw.user_consume_info t1                                                                                      \
                        left join (                                                                                                        \
                                   select user_id,                                                                                         \
                                          country                                                                                          \
                                     from (                                                                                                \
                                           select user_id,                                                                                 \
                                                  country,                                                                                 \
                                                  row_number() over(partition by user_id order by last_date desc) as rank                  \
                                             from dim.dim_user_info                                                                        \
                                            where data_date ="+"'"+start_date_str+"'"+"                                                    \
                                              and site_id in (600,900)                                                                     \
                                              and country is not null                                                                      \
                                          ) t                                                                                              \
                                    where t.rank =1                                                                                        \
                                   )  t2                                                                                                   \
                           on t1.user_id = t2.user_id                                                                                      \
                        where t1.data_date = "+"'"+start_date_str+"'"+"                                                                    \
                          and t1.last_order_paid_time is not null                                                                          \
                          and t1.app_name = 'JC' "
```

```
    user_rfm = "   select user_id,                                                                                                  \
                          case when datediff("+"'"+date_str+"'"+",last_pay_date)<90 then '近'       \
                          else '远' end as date_diff,                                                                               \
                          case when last_1y_paid_orders <3 then '低频'                                                              \
                          else '高频' end as sum_orders,                                                                            \
                          case when last_1y_paid_order_amount <50 then '低额'                                                       \
                          else '高额' end as sum_amount                                                                             \
                     from user_rfm_info                                                                                             \
                    where country = 'ID'                                                                                            \
                union all                                                                                                           \
                    select user_id,                                                                                                 \
                          case when datediff("+"'"+date_str+"'"+",last_pay_date)<90 then '近'       \
                          else '远' end as date_diff,                                                                               \
                          case when last_1y_paid_orders <3 then '低频'                                                              \
                          else '高频' end as sum_orders,                                                                            \
                          case when last_1y_paid_order_amount <300 then '低额'                                                      \
                          else '高额' end as sum_amount                                                                             \
                     from user_rfm_info                                                                                             \
                    where country <> 'ID'    "


    insert_table = "insert overwrite table " + target_table + " partition(data_date="+"'"+start_date_str+"'"+",tagtype='rfm_model')     \
                           select case when date_diff = '近' and sum_orders = '高频' and sum_amount = '高额' then 'A111U008_001'     \
                                       when date_diff = '远' and sum_orders = '高频' and sum_amount = '高额' then 'A111U008_002'     \
                                       when date_diff = '近' and sum_orders = '低频' and sum_amount = '高额' then 'A111U008_003'     \
                                       when date_diff = '远' and sum_orders = '低频' and sum_amount = '高额' then 'A111U008_004'     \
                                       when date_diff = '近' and sum_orders = '高频' and sum_amount = '低额' then 'A111U008_005'     \
                                       when date_diff = '远' and sum_orders = '高频' and sum_amount = '低额' then 'A111U008_006'     \
                                       when date_diff = '近' and sum_orders = '低频' and sum_amount = '低额' then 'A111U008_007'     \
                                       else 'A111U008_008' end as tagid,                                                             \
                                  user_id as userid,           \
                                  '' as tagweight,             \
                                  '' as tagranking,            \
                                  '' as reserve,               \
                                  '' as reserve1               \
                             from user_rfm "


    spark = SparkSession.builder.appName("user_rfm_model").enableHiveSupport().getOrCreate()
    returned_df1 = spark.sql(user_rfm_info).cache()
    returned_df1.createTempView("user_rfm_info")
    returned_df2 = spark.sql(user_rfm).cache()
    returned_df2.createTempView("user_rfm")

    spark.sql(insert_table)

if __name__ == '__main__':
    main()
```

对于中间计算数据进行缓存，创建临时视图

脚本中的应用-checkpoint



切断这种依赖机制，把当前运行状态保存在hdfs上，可以用checkpoint来实现。

如果应用中使用了***updatestatebykey***或***reducebykeyandwindow***等函数，需要使用checkpoint来定时地。。

Checkpoint要接收一个文件目录hdfs，在streamingContext中设置一个目录，所有数据都会存放到目录中。每隔一段时间会自动checkpoint

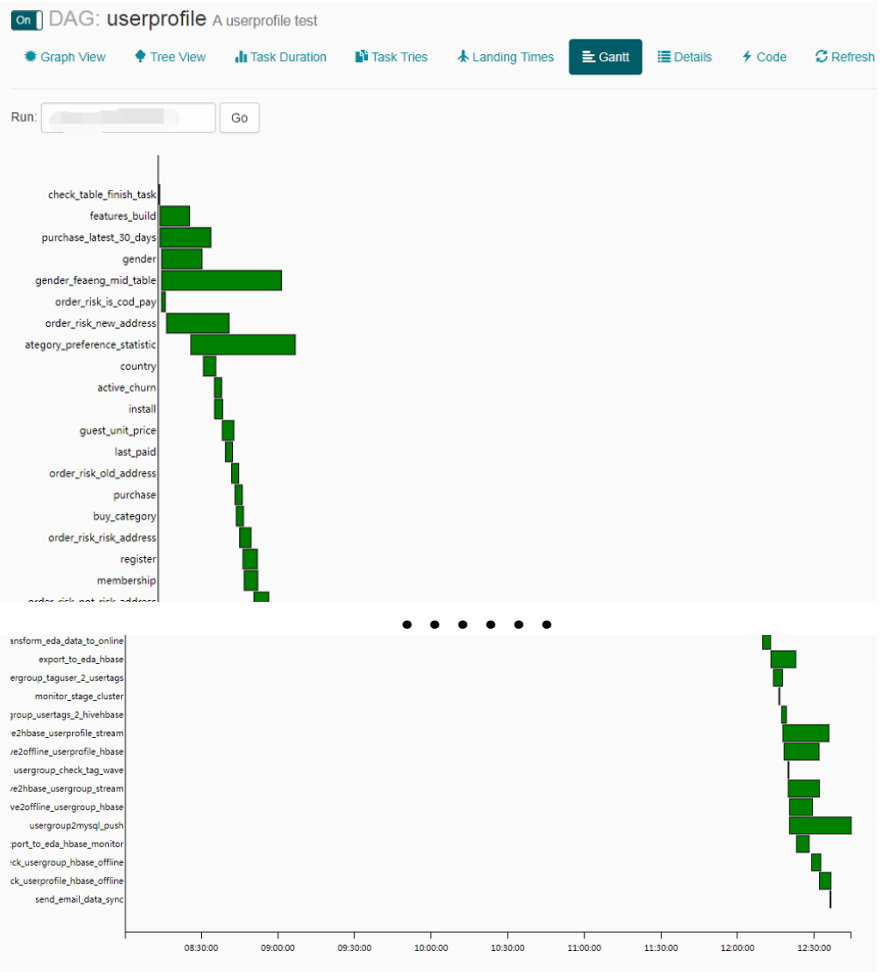
当然在Direct模式下如果自己存储offset信息到zk或mysql等地方的话，也可以不用checkpoint，中断重启后读取之前存储的offset即可

```
def run(interval: Int) {
  val sparkConf = new SparkConf()
    .setAppName("dataBrainStream")
    .set("spark.streaming.kafka.maxRatePerPartition", "10000")
    .set("spark.streaming.concurrentJobs", "5")

  // mysql 配置
  val mysqlProp = new java.util.Properties
  mysqlProp.setProperty("user", Property.MYSQL_USER)
  mysqlProp.setProperty("password", Property.MYSQL_PASSWORD)
  mysqlProp.setProperty("driver", Property.MYSQL_DRIVER)

  //kafka 配置
  val kafkaParams = Map[String, String](
    "metadata.broker.list" -> Property.KAFKA_BROKERS      // 包括master节点和 slave节点的kafka
    , "serializer.class" -> "kafka.serializer.StringEncoder"
    , "group.id" -> "dataBrainStream"
  )

  // Create the context with a $interval second batch size
  val ssc = new StreamingContext(sparkConf, Seconds(interval))
  ssc.checkpoint("/user/hunter/dataBrainStream")
```

开发中间表

统计各任务的调度时间,优化脚本

- 根据Airflow的调度管理工具，查看每个调度任务的执行时长，各标签计算的数据血缘，整理出一份文档。针对有公共血缘的标签，建立中间表



A	B	C	D	G	H	I	J	K	L
标签id	标签名称	标签汉语	标签主题	二级标签id	二级标签	来源表	标签类型(tag type)	维度	作业大致时间
A121H030_0_001	unregistered	未注册	用户属性			dw.dw_cookie_dau_visit dw.dw_cookie_user_relation	registed_state	cookieid	1分半
A121H030_0_002	registered	已注册	用户属性						
A121H031_0_001	purchased	购买过	用户属性			dw.dw_cookie_dau_visit dw.dw_cookie_user_relation dim.dim_user_info	purchase_state	cookieid	2分钟
A121H031_0_002	not purchased	未购买过	用户属性						
A220H029_0_001	installation date	安装距今天数	用户属性			dw.dw_cookie_dau_visit	install_days	cookieid	1分半
B121H030_0_001	high active morning	上午	用户行为			ods.ods_page_view_log	high_active_period	cookieid	4分钟
B121H030_0_002	high active noon	中午	用户行为						
B121H030_0_003	high active afternoon	下午	用户行为						
B121H030_0_004	high active evening	晚上	用户行为						
B121H030_0_005	high active dawn	凌晨	用户行为						
B121H034_001_001	push active morning	上午	用户行为	1	时间偏好	ods.ods_flume_apppush_log	push_active	cookieid	3分钟
B121H034_001_002	push active noon	中午	用户行为	1	时间偏好				
B121H034_001_003	push active afternoon	下午	用户行为	1	时间偏好				
B121H034_001_004	push active evening	晚上	用户行为	1	时间偏好				
B121H034_001_005	push active dawn	凌晨	用户行为	1	时间偏好				
B220H026_0_001	last order to now days	尾单距今天数	用户行为			dw.dw_cookie_dau_visit dw.dw_cookie_user_relation dim.dim_user_info	last_paid_days	cookieid	2分钟

甘特图查看每个task任务执行所需时间，再根据每个任务数据血缘，整理待开发中间表

案例：某用户特征中间表开发

脚本：附件中的 middletable_build.py

```
    start_date_str = str(start_date_in)
    end_date_str = str(end_date_in)

    start_date = datetime.datetime.strptime(start_date_str, "%Y%m%d")
    end_date = datetime.datetime.strptime(end_date_str, "%Y%m%d")

    date_timedelta = end_date - start_date
    epochs = date_timedelta.days+1
    #logger.info('time range: '+start_date_str+'to'+end_date_str+' cycle: '+str(epochs))

    # 循环处理
    spark = SparkSession.builder.appName('userprofile_features_build').enableHiveSupport().getOrCreate()
    partition_date_str = start_date_str
    partition_date = start_date
    for epoch in range(epochs):
        epoch_start_time = time.time()
        # 传入日期参数,返回待执行的str
        cookie_dau_goods_rela_sql, goods_detail_duration_sql, goods_paid_order_sql, goods_imp_sql, goods_click_gd_event_sql = buid_query_sql(partition_date_str)

        df_cookie_dau_goods_rela = spark.sql(cookie_dau_goods_rela_sql)
        df_goods_detail_duration = spark.sql(goods_detail_duration_sql)
        df_goods_paid_order_duration = spark.sql(goods_paid_order_sql)
        df_goods_imp = spark.sql(goods_imp_sql)
        df_goods_click_gd_event = spark.sql(goods_click_gd_event_sql)

        # 将用户各维度的行为特征union all起来
        df_cookie_goods_features = df_cookie_dau_goods_rela.unionAll(df_goods_detail_duration).unionAll(df_goods_paid_order_duration).unionAll(df_goods_imp).unionAll(df_goods_click_gd_event)
        df_cookie_goods_features_sum = df_cookie_goods_features.groupBy('cookieid', 'goodsid', 'siteid').sum(*df_cookie_goods_features.columns[3:])

        # 创建临时视图
        df_cookie_goods_features_sum.createOrReplaceTempView('tmp_view_df_cookie_goods_features_sum')
        # 插入目标分区表中
        insert_sql = "\
            insert overwrite table dw.dw_cookie_goods_log_day partition(data_date='"+ partition_date_str +"')\
            select * from tmp_view_df_cookie_goods_features_sum\
        "
        spark.sql(insert_sql)
        partition_elapse = exec_time_utils(epoch_start_time)
        #logger.info(partition_date_str+'daily data finished. '+partition_elapse)

        # 处理循环变量
        partition_date = datetime.datetime.strptime(partition_date_str, "%Y%m%d")+ONE_DAY
        partition_date_str = partition_date.strftime("%Y%m%d")
    all_elapse = exec_time_utils(start_time)

if __name__ == '__main__':
    main()
```