1. 上课须知

课程主题：Spark 分布式计算引擎 SparkCore 第二次课
上课时间：2020-09-25 20:00 - 23:00
课件休息：21:30 左右 休息10分钟
课前签到：如果能听见音乐，能看到画面，请在直播间扣 666

2. 上次课总结

上次课讲了的知识点：

```
1、MapReduce的缺点分析
2、Spark产生背景、发展历程、应用场景
3、Spark企业级高可用集群部署
4、SparkShell 和 SparkSubmit 使用详解
5、Spark编程模型详解和WordCount
6、Spark的核心功能，应用模块，基本架构
7、Spark的编程模型，运行机制，核心概念
8、RDD详解和SparkContext详解
```

未讲到的知识点：

```
1、RDD的算子实战
2、RDD的cache机制和Spark的共享变量
```

3. 课程预告

两个重要知识：

```
1、Spark 的高效代码怎么写？          开发调优
2、Spark 的数据倾斜怎么解决？         数据倾斜调优
```

关于Spark的调优：

```
1、Spark 的 shuffle 原理
2、Spark 的 资源参数 调优
```

spark的shuffle

spark的内存管理

spark on yarn

开发调优 资源调优 数据倾斜 shuffle调优

spark的源码：脚本启动分析，集群启动，sparkcontext

spark的源码：任务提交执行，stage切分，Shuffle的详细流程，

BlockManager ShuffleManager

4. 课程笔记

4.1. 方案一：使用Hive ETL预处理数据

原理：

假设现在有10个spark读取同一份数据(表)，都发生了数据倾斜！
大概率就是数据源本身分布式不均匀更导致的！

提前的ETL ，把解决数据倾斜的时机提前！

4.2. 方案二：调整shuffle操作的并行度

假设现在有一段代码
```
sparkcontext.textFile(...).flatMap(...).map().reduceByKey(func,
numTasks).collect()
```

默认的分区规则，Hash散列，既然由于分区数目的不同可能导致倾斜，那么调整分区的个数据，就有可能数据倾斜.

解决方案：通过碰运气的心理，去调整一下 task 的数量，还是有可能就解决当次数据倾斜

4.3. 方案三：过滤少数导致倾斜的key

假设现在我有个 join的需求实现代码，最终定位到导致倾斜的原来是由于参与链接的某一张表的链接字段的null比较多
所以通过shuffle之后，这些null记录一定偶都是被分配到同一个task
这个task所承担的计算压力要远远高于其他task

解决方案:

1、假设，这种带null的记录对于我们来说，是没有需要的结果。在进行join的时候提前进行null过滤
```
select a.*, b.* from a join b on a.id = b.id where a.id is not null;
select a.*, b.* from (select a.* from a where a.id is not null) a join b on a.id
= b.id;
case when a.id is null then concat(a.id, random()) else a.id 如果为空给一个随机数
```
解决：过滤掉

2、如果这些null记录，也是我们想要的！
    1、两阶段聚合，这个方案，只能用于聚合需求
    第一阶段：随机分区做预聚合，第二阶段做hash散列做最终聚合
    2、横向拆分
    把null拿出来单独成为一个job，把非null的单独拿出来成为一个job
    最后两个job结果在 unoin 起来

4.4. 方案五：采样倾斜 key 并分拆 join 操作

```
既然产生了倾斜，一定是由于某些数据量比较大，集中分布在一个节点，我需要去了解到底是那些个数据量比较大导致了倾斜。！
1、通过采样的方式，先了解数据分布的规律。
2、制定一个策略：把数据量比较大的数据，拆分出来单独处理
3、一个job拆分成两个job，并且这两个job都没有倾斜！
```

根据也无需求的SQL实现：

```
select ......from a group by a.age;
```

SQL 伪代码实现：

```
select ......from a group by a.age where a.age = 18 or a.age = 19
union
select ......from a group by a.age where a.age != 18 and a.age != 19;
```

rdd.filter("a.age = 18 or a.age = 19")

rdd.filter("a.age != 18 and a.age != 19;")

4.5. 方案四：将reduce join转为map join

join 的需求分类：

```
1、小表 join 小表
2、大表 join 小表
    reducejoin mapjoin
    1、hive sql
        select /*+mapjoin(smalltable)*/
    2、spark
        rdd.mapPartition().withBroadCast(smallTable_rdd)
3、大表 join 大表
    hive 连续七天发朋友圈的用户有哪些？
    文华写的一篇公众号的文章
```

4.6. 方案六：两阶段聚合（局部聚合+全局聚合）

```
需求：
select word, count(*) as count from wc group by word;    10分钟
word分布不均匀：
解决方案：
1、随机数据分发，做第一次聚合      1分钟
2、hash散列，第二次聚合          2分钟
```

hive的调优：导致倾斜的三大应用需求场景：

```
group by
distinct
join
```

4.7. 方案七：使用随机前缀和扩容 RDD 进行 join

无条件join，笛卡尔积

```
select a.*, b.* from a, b;   #  最终只能使用一个task去实现需求
select a.*, b.* from a, b where 1 = 1;

解决方案：
链接条件必须包含多个不同的值！
随机前缀和扩容 RDD + 扩容
```

4.8. 方案八：任务横切，一分为二，单独处理

```
1、所有导致倾斜的数据拿出来处理，
2、所有不导致倾斜的数据拿出来处理，
```

4.9. 方案九：多种方案组合使用

多个方案组合使用

```
select a*, b.* from a join b on a.id = b.id;
存在两方面的情况：
1、id有部分为空
2、id不为空的数据中，也有部分key的分布特别多
```

4.10. 方案十：自定义 Partitioner

```
自己编写优秀的数据分发规则
保证两个事情：
1、最终的业务结果正确
2、保证你定义的数据分发规则能确保任何一个阶段的 shuffle 都没有倾斜！
```

在mapreduce：

```
1、默认数据分发规则：HashPartitioner
2、Partitioner

Partitioner 接口，  抽象类
```

在spark当中：

```
spark.repartition
spark.broadcast()
spark.coalesce()
```

在Flink当中，关于数据分区有五个核心的算子

在Storm当中，关于数据分发有8个招术

5. SQL Spark程序

```
Spark： ETL
开发： 落实到写SQL
```

```
流式：sparkstreaming  flink
```

三个方面的知识：

```
1、架构设计
2、企业最佳实战
3、源码调优
```

道 术

```
术：招术
    HDFS，元数据管理，
    Kafka：怎么保证吞吐，怎么保证数据的消费语义
道：核心思想
```

```
数据中台
OLAP体系
```

6. 课间休息

22：10 继续上课！

7. 开发调优

7.1. 避免创建重复的RDD + 尽可能复用同一个RDD + 对多次使用的RDD持久化

```
如果有一个RDD，需要在多个地方被使用，就请只使用这一个。，不要重复创建，既然这个RDD需要重复使用，最好进行持久化保存！
mapreduce: mapper shuffle reducer  mapper shuffle reducer
spark: stage1   stage2   stage3   stage4
       rdd1      rdd2     rdd3     rdd4

rdd的持久化：
1、memory
2、disk
BlockManager CacheManager

cache()  persist(STORAGE_LEVEL)
unpersist()

集成了 alluxio tachyon  基于内存的分布式文件
```

关于 Spark 的内存管理，

```
存储内存
执行内存
```

7.2. 尽量避免使用Shuffle类算子

两个大后果：

```
1、效率低
2、可能产生数据倾斜
```

能用则用的解方案：mapjoin

```
位图    位运算 没有shuffle
hashjoin  shuffle
```

Flink Doris Clickhouse

```
集成到了这些技术中，可以直接拿来使用！
```

Broadcast与map进行join代码示例

```
原来：多张表的全局数据分发  shuffle
现在：一张小表数据的全局广播  使用内存存储来替换shuffle操作
```

7.3. 使用Map-Side预聚合的Shuffle操作

```
reduceByKey  groupByKey
```

7.4. 使用高性能的算子

```
1、使用reduceByKey/aggregateByKey替代groupByKey
    前两个算子在shuffle之前有预聚合
2、使用mapPartitions替代普通map
    rdd.map(element => {
        val connection = MySQL.getConnection();
        val user_data = connection.getTableData("user", id)
        element + user_data
    })
    rdd.mapMapParitions(rdd_partition => {
        val connection = MySQL.getConnection();
        val user_data = connection.getTableData("user", id)
        rdd_partition.map(element => {
            element + user_data
        })
    })  +  数据库的连接池!
3、使用foreachPartitions替代foreach
    同理上一条
4、使用filter之后进行coalesce操作
    rdd.filter(age > 18)   RDD 的数据量变少了。！ 做了filter之后，分区个数没有变，但是分区的数据量减少了。原来每个 分区1G数据，经过 filter之后 变成了 100M
    RDD的数据总量： 分区数 * 分区的平均数据量
    思路：在经过filter之后，进行coalesce操作：多个分区合并成一个分区
5、使用repartitionAndSortWithinPartitions替代repartition与sort类操作
    原来的分区不吻合我的需求：先reparation， 然后每个分区sort排序!
    rdd.repartition().sort(); < rdd.repartitionAndSortWithinPartitions
```

在mapredudceshuffle当中： 边shuffle边排序，就是为了排序

```
1、如果有reducer阶段就一定会进行shuffle
2、如果有shuffle，那么就一定会按照key排序
原因：
1、如果一个reduceTask执行计算的输入数据是无序的，则每个reduceTask进行分组聚合的时候，需要对这个输入文件尽心多次扫描。
2、reduceTask期望它的输入数据是有序的。按照顺序来扫描这个文件一次，就能做完所有分组操作
```

spark做了优化：

```
1、不用排序的shuffle
2、需要排序的shuffle
```

有两种优化技巧是能用则用：

```
1、combiner
2、mapjoin
```

7.5. 广播大变量

spark 的 mapjoin 的实现就依赖于 broadcast

```
提升效果明显：
1、如果待广播特别小，就没有广播的必要性了。
     driver中声明了一个小的全局变量，最后再算子的函数参数中，使用了
2、待广播的数据量大：如果使用广播变量的工作机制，则一个worker中启动的某个executor中的多个Task 就可以共用一份数据，这个广播数据就是存储在  存储内存中，这个内存有可能是堆内内存，也有可能是堆外内存
3、如果待广播的数据量特别大，不适合使用广播机制！
```

企业环境中：

```
yarn集群中的 containner 内存大小是：
1、默认是1G
2、4G, 2G, 8G
```

最终结论：在能广播的前提下，待广播的数据量越大，提升效率越明显。内存资源占用的减小很明显

7.6. 使用Kryo优化序列化性能

在海量数据处理中：

```
1、网络传输
2、数据的序列化
```

现在了解的关于做序列化的一些技术有哪些?

```
1、hive thrift
2、hadoop avro（class Student extends Writable）实现 readFilds() write()
3、java  serializable
     每个对象包含：对象头  对象体
                  类的信息，  对象的属性数值
           Spark默认就是使用 java自带的序列化机制
4、kryo 序列机制
     对象的对象头信息，只序列化一次。
     对象体，有多少个对象就序列化多少次
     1、声明使用这种优化机制
     2、注册使用这种优化机制的类
```

7.7. 优化数据结构

```
基本类型
字符串： String = char sequence
数组
对象
集合
```

```
使用字符串替代对象
使用原始类型（比如 Int、Long）替代字符串
使用数组替代集合类型
```

```
理论上来说有价值，实施起来很困难！
```

7.8. 融会贯通

可读性，可维护性，可扩展性

高性能，高吞吐，低延迟

8. 图数据库

```
titan  neo4j  ....
```

9. 疑问