|  |  |  |
| --- | --- | --- |
| 作者：曾昭正 | 最后修改日期：2016/03/15 |  |

MR：

1、作业失败了怎么回事，怎么看日志？什么情况下应该看什么组件的日志？

2、作业失败了，应该保留什么有用现场信息

3、作业倾斜了？怎么判断倾斜,可以怎么调整。

4、作业跑的很慢，怎么判断是资源不足还是作业本身问题？我的作业占使用了多少资源在跑？

5、为什么作业提交到一些资源队列有时候需要等待一段时间才可以获得资源运行？这个资源等待时间是否可以变短？

6、自己写UDF应该注意什么问题？

7、自己配置的参数是否生效，运行时参数值是多少？

8、mapjoin使用问题

SPARK:

1、yarn-client模式和yarn-cluster模式有什么区别，什么场景用什么模式？

2、我的Spark应用失败了，日志应该怎么看？

3、我的Spark应用一直在运行，它正在运行到了我们自己写的哪段代码？

4、我自己配置的参数在运行时是否生效？

4、怎么找到spark应用对应的task对应的线程在界面哪里可以找到，并且怎么看它是否运行健康

5、我的Spark作业出现了OOM，什么组件出现OOM，哪个阶段出现了OOM，一般情况下什么阶段容易出现OOM，应该怎么调整？

6、我的Spark程序在Drvier端大量cache数据行不行？

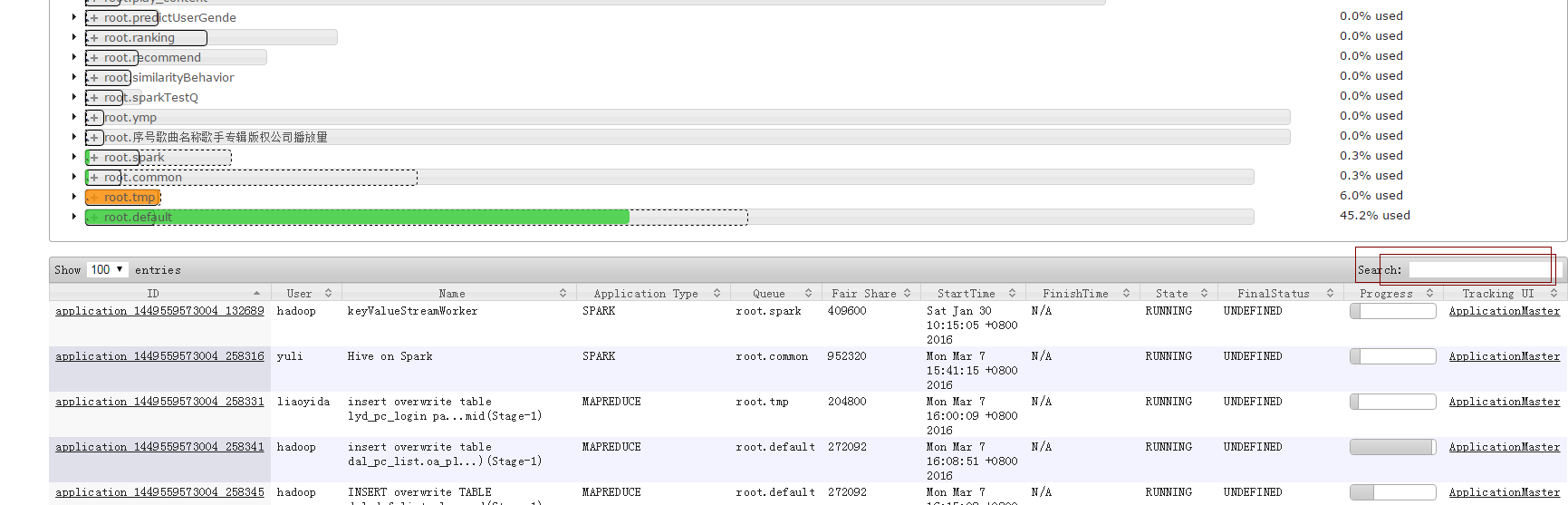
7、我spark程序中间结果是该存储在客户端单机还是HDFS中，为什么？

8、为spark程序申请多大的资源粒度才适合？应该起多少个task？

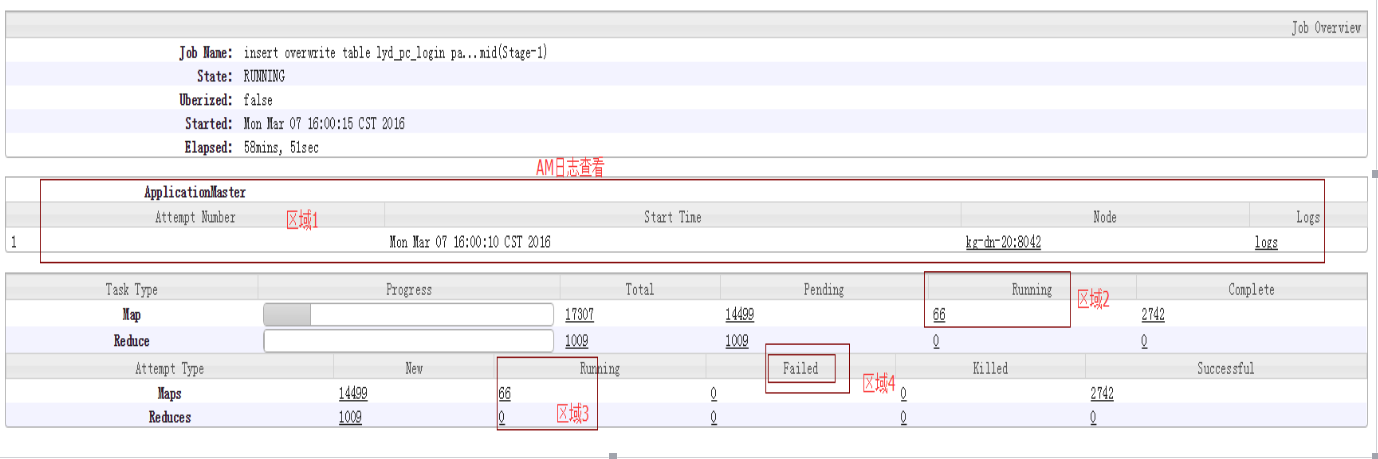
9、用于缓存的比例怎么设置才合理？

**这里只选了些涉及内容比较复杂的进行说明，一些简单的在分享过程中进行演示说明。**

1. **MR组件基本情况以及日志应该如何查看**
2. 在yarn中作业的唯一标识是ApplicationID，所以作业有问题先第一时间记下ApplicationID。
3. 在yarn管理界面所有对应ApplicationID对应的信息（需要自己配置好相应的代理信息）



1. yarn各个组件基本情况以及其日志分布



MR运行在yarn上面，组成一个作业且普通用户需要关注的基本组件有：ApplicationMaster（AM）、reduce task、map task。上面截图分为几个区域：

区域1：为AM相关信息，Attmpt Numer为AM尝试的次数，如果AM没有出错的话只会有一行AM信息，如果AM一旦有问题的话将会启动容错机制另外启动一个AM，这是这个区域会有2条AM的记录。具体出错信息可以点击界面上的logs查看。另外，node列相关信息表示为这个AM在kg-dn-20节点上运行其对应的端口为8042。

区域3：这个作业同时有多少个map和reduce数正在运行，这个也可以看出当前这个作业占用了多少资源在跑。占用资源：map个数\*Map(3G+1core)+reduce个数\*reduce(4G+1core)+AM(3G+2core)

区域4：如果存在失败的task在这里可以显示，同样在界面上可以点击进去看相应情况。

1. **作业倾斜问题**

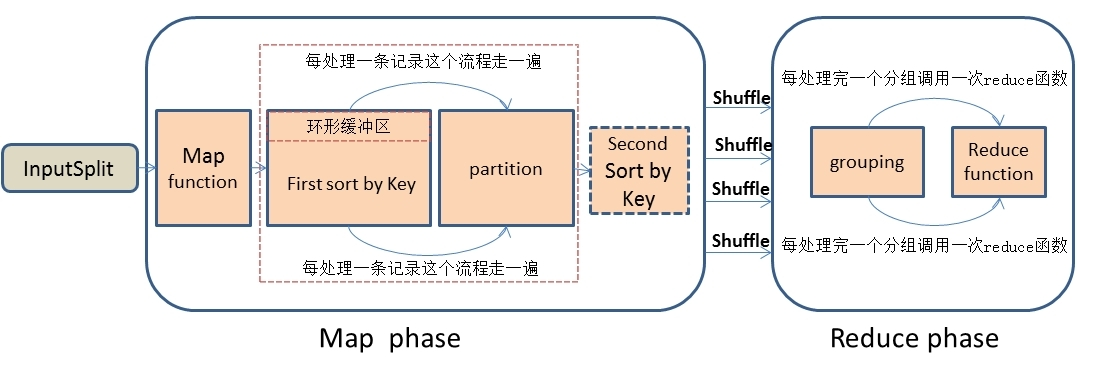
1、倾斜作业的症状

任务进度长时间维持在99%（或100%），查看任务监控页面，发现只有少量（1个或几个）reduce子任务未完成。因为其处理的数据量和其他reduce差异过大。

2、倾斜原因

倾斜的根本原因是查询的聚合维度组合结果数据量差距大。

首先看看MR处理的流程



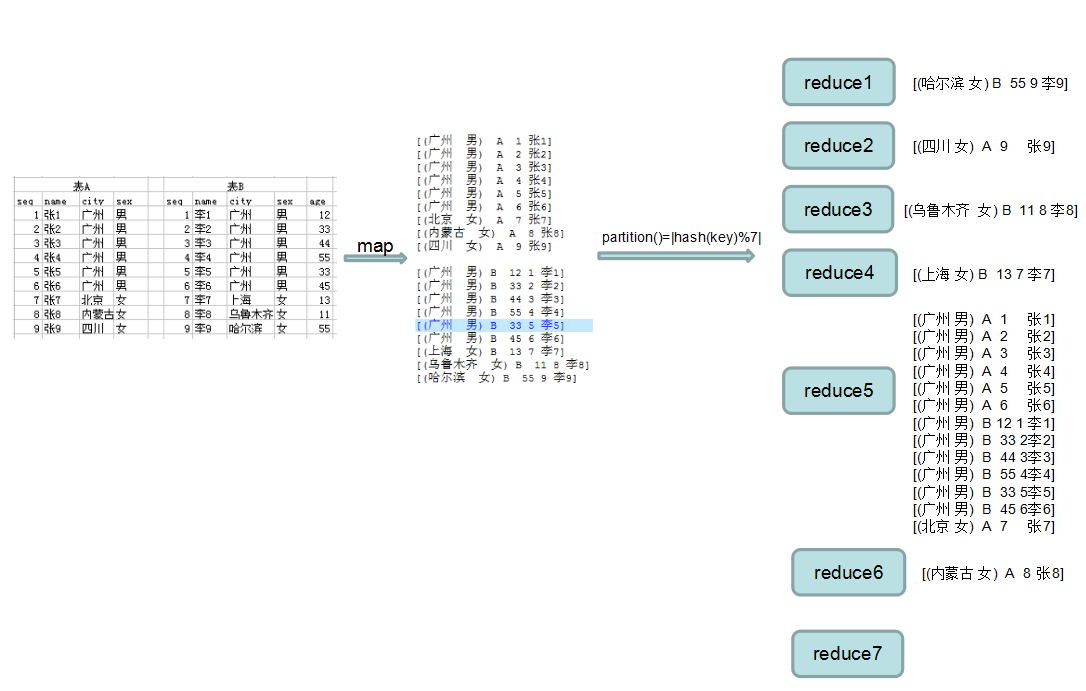
首先，MapReduce框架通过getSplit方法实现对原始文件的切片之后，每一个切片对应着一个map task，inputSplit输入到Map函数进行处理，中间结果经过环形缓冲区的排序（根据key值）,然后分区、自定义二次排序（如果有的话）和合并，再通过shuffle操作将数据传输到reduce task端，reduce端也存在着缓冲区，数据也会在缓冲区和磁盘中进行合并排序等操作，然后对数据按照Key值进行分组，然后每处理完一个分组之后就会去调用一次reduce函数，最终输出结果。

然后，导致倾斜问题的根本原因是由于map阶段重新组合的key值（聚合维度组）过于集中，从而导致map阶段分区函数对reduce的选择上过于集中，这里有点抽象，举个实际查询例子：

表A（seq int,name string,city string,sex string) 表B（seq int , name string ,city string,sex string,age int）。

SQL: select A.name,B.age from A left outer join B on(A.city =B.city and A.sex=B.sex)

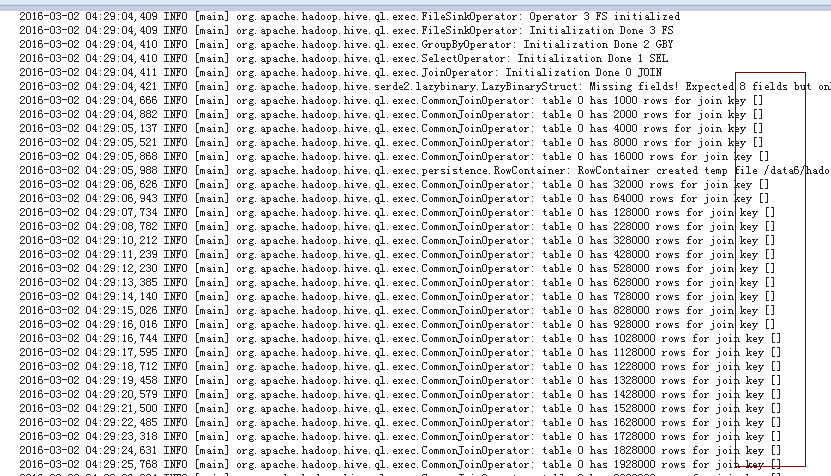
看看数据流程：



从数据流可以看出来，大部分数据被集中分到一个reduce上处理，这个就会导致作业倾斜。

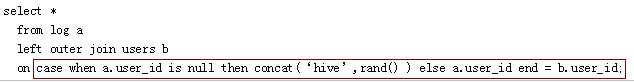
1. 缓解倾斜方法
2. 针对空值连接产生的倾斜情况（这种情况平时情况遇到的最多）

看下这种情况下产生的logs信息：



空值是不参与join的，所以可以把空值的 key 变成一个字符串加上随机数，就能把倾斜的数据分到不同的reduce上 ,解决数据倾斜问题。

举个例子：



### 不同数据类型关联产生数据倾斜（这种情况也是非常见）

**场景：**用户表中user\_id字段为int，log表中user\_id字段既有string类型也有int类 型。当按照user\_id进行两个表的Join操作时，默认的Hash操作会按int型的id来进行分配，

这样会导致所有string类型id的记录都分 配到一个Reducer中。

**解决方法：**把数字类型转换成字符串类型



3、对于相同key值不多，但是数据倾斜让然很严重的作业情况。需要重新编写partition分区函数（针对手动写MR情况），使其尽量均匀分布在不同的reduce上或者可以往key值前加随机数将key值打散。

4、对于用group by的sql，尽量维度将维度放大（实际上就是缓解了维度聚合结果数据量过大问题），这个可以有效的缓解数据倾斜问题。另外，采用sum() group by 方式来替换count(distinct)操作 （因为计算器的累加比字段文本的聚合IO低的多）。

5、通过调整个别参数缓解（个人不推荐这种方法，因为会产生额外的计算资源开销并且效果不是太理想）

**（1）**开启combiner功能，减少IO开销。调整hive.map.aggr = true

**（2）**开启hive.groupby.skewindata=true

有数据倾斜的时候进行负载均衡，当选项设定为 true，生成的查询计划会有两个 MR Job。第一个 MR Job 中，Map 的输出结果集合会随机分布到 Reduce 中，每个 Reduce 做部分聚合操作，并输出结果，这样处理的结果是相同的 Group By Key 有可能被分发到不同的 Reduce 中，从而达到负载均衡的目的；第二个 MR Job 再根据预处理的数据结果按照 Group By Key 分布到 Reduce 中（这个过程可以保证相同的 Group By Key 被分布到同一个 Reduce 中），最后完成最终的聚合操作。

1. **自定义UDF内存使用问题**

自定义UDF要注意内存的使用，我们集群目前配的map内存为3G，在编写UDF中要注意控制缓存大小。

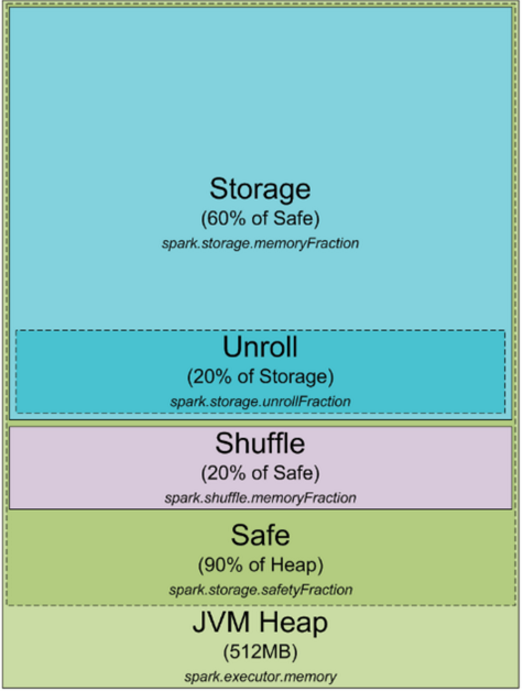
下面是个之前同事写的有问题的UDF（UDFGetIPProvince.jar）：



Evaluate函数，每进一条记录就会被调用一次，最终proviceCache（map对象）的大小变得不可控，导致map task频繁FULL GC，最终该map task落在的节点负载变得很高。

1. **spark的内存组成结构以及易发OOM点**

1、Executor内存结构



spark.storage.memoryFraction( 0.6)

spark.storage.safetyFraction(0.9)

=>用于cache RDD内存大小=JVM heap\*spark.storage.memoryFraction\*spark.storage.safetyFraction=JVM heap\*0.6\*0.9

spark.shuffle.memoryFraction(0.2)

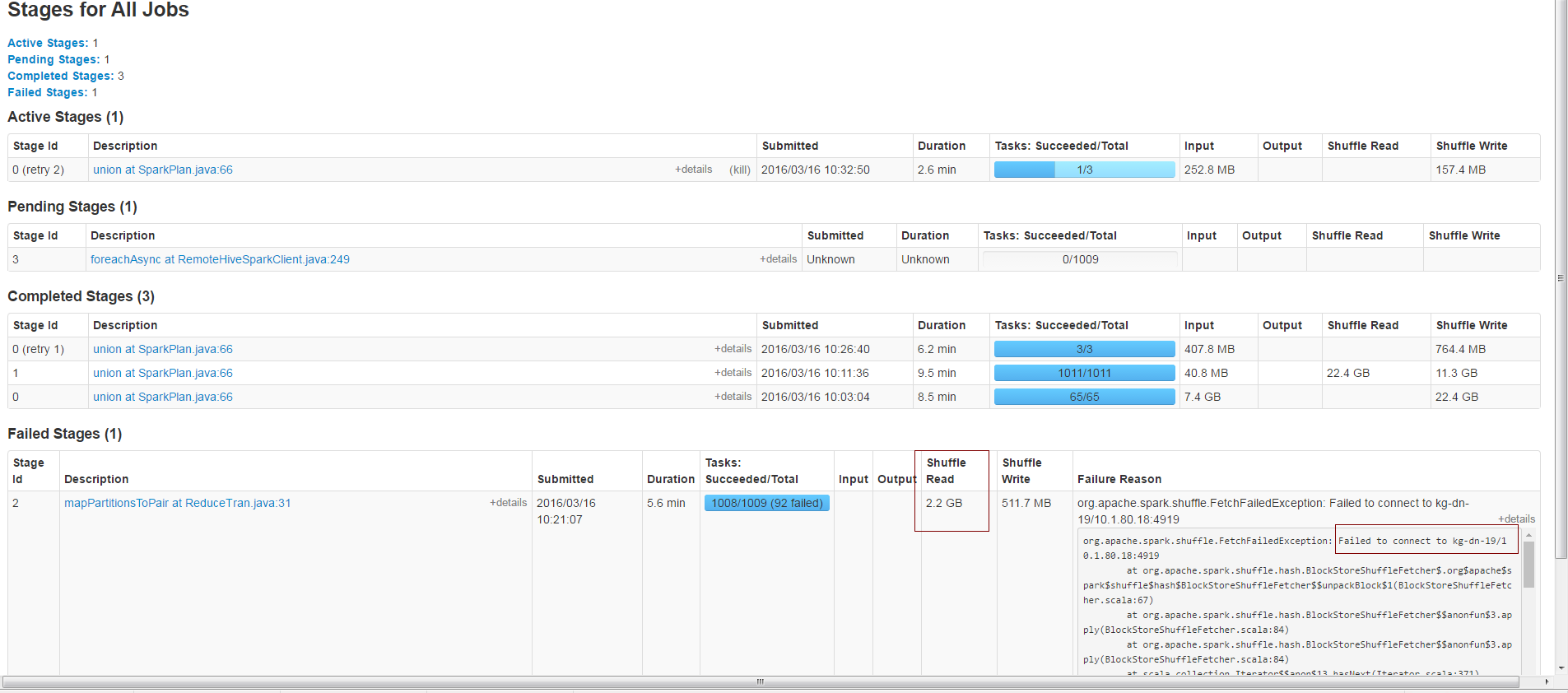
spark.shuffle.safetyFraction(0.8)

=>用于shuffle的内存JVM heap\*spark.shuffle.memoryFraction\*spark.shuffle.safetyFraction\*spark.storage.safetyFraction=JVM heap\*0.2\*0.8\*0.9（这一部分往往OOM频发区）

用于jvm运行时其他对象开销（这一部分往往OOM频发区）= JVM heap Size - 用于cache RDD内存大小 - 用于shuffle的内存

1. 易发OOM点
2. shuffle read阶段出现OOM

表现：shuffle read阶段，executor出现OOM进程直接死掉，出现failed to connect kg-dn-.....，最终触发容错机制重新开启另一个executor（还可能出现OOM死掉）重跑失败的task。



原因：在shuffle过程中估算的内存扩展量超过用于shuffle的内存阀值（想详细了解的看看ExternalAppendOnlyMap结构部分，附件资料第四篇）

解决：

A、调大jvm堆栈。（集群资源有限，这个尽量按照给予的资源分配量配置，自己不要动手配）

B、调低spark.shuffle.safetyFraction（默认0.8）

C、增大shuffleReduce Task并发，调大参数spark.default.parallelism（默认配置320）

D、降低每个excutor的task并发数，调低SPARK\_EXECUTOR\_CORES值。

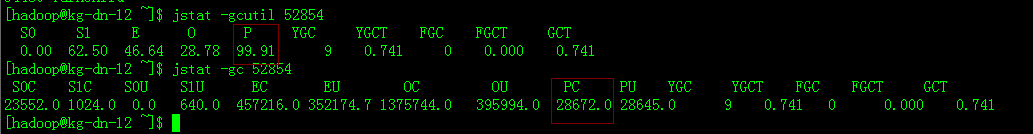
E、资源粒度调大，executor并发降低（确保使用的总资源量在分配的范围内）。

1. 持久代内存OOM

表现：任何阶段都存在发生可行性，报堆栈信息：java.lang.OutOfMemoryError:PermGen space

调整解决：

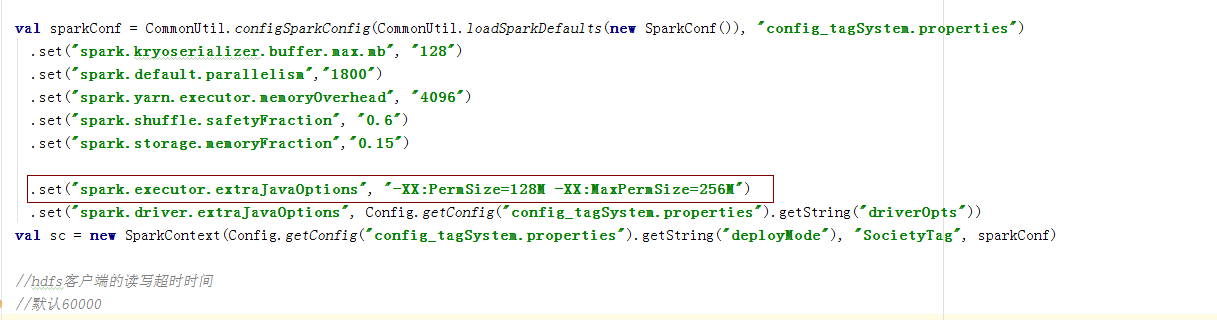
首先检查executor在运行的节点上的GC情况中的持久代信息：



Gcutil 中的P为持久代占用的百分比，gc最下面PC对应的是分配量，PU对应的是使用量。判断是否持久代不够，直接观察P的占比是不是一直都是99.99%而且不断FullGC且PD的百分比不下降。

调整参数：增大持久代的内存，调整spark.executor.extraJavaOptions。

Eg:



1. persist RDD时出现OOM

表现：报错信息java.lang.OutOfMemoryError: Java heap space

调整：选择适当的StorageLevel，如果存储的RDD很大，给予的内存总量不足以存放，那么请采用MEMORY\_AND\_DISK策略，另外一个重要的问题：如果要persist的RDD很大尽量将RDD压缩，将spark.rdd.compress设置为true，以节省内存。

1. 堆外内存OOM（想深入了解的话请自行研究下DirectBuffer ）

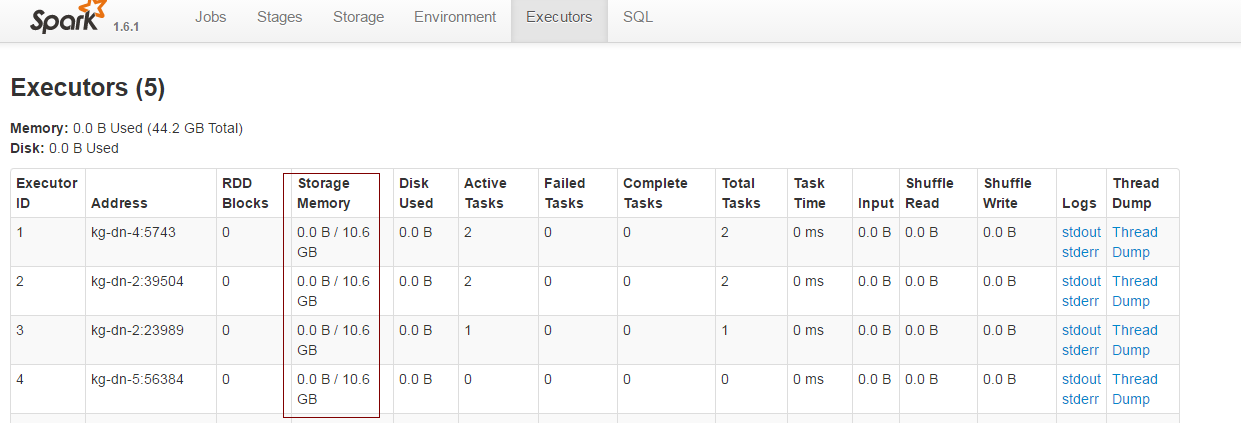
堆外内存OOM出现后的指引日志信息也是spark做的很不好的地方，没有相应的指引日志，起表面现象和第一点时一样的，当我们在调整堆内内存不起效果还是出错的话，我们可以调整堆外内存。

堆外内存的默认值为 executor的堆内内存\*0.07。可以单独调整堆外内存值spark.yarn.executor.memoryOverhead，可以调整为512，这个堆外内存我建议不要设置超过1024。

1. **用于缓存的比例怎么设置才合理？**

缓存的比例取决于你想用于迭代的RDD量的大小和给予分配的executor的堆栈大小。

情况一：需求不需迭代使用RDD，那么RDD cache这部分必须设置为最低，设置为0.1。不能够出现配置了很大的RDD Cache内存但又是不用这种浪费资源的情况：



这里有10.6G的内存被配置为cache RDD用的，但是代码里边却没有使用，那么这10.6G的内存就浪费了。另外，用于shuffle的内存也相应调大spark.shuffle.memoryFraction（默认0.2）

情况二：需求需要重复迭代用RDD，需要cache RDD。

这种情况设置cache的比例为多少合适呢。

在给定资源不变的前提下，优先考虑shuffle部分内存使用量，开启StorageLevel的MEMORY\_AND\_DISK，并且开启cache的压缩机制。

调整步骤：先保留spark.storage.memoryFraction=0.6，然后在shuffle read阶段观察老年代的FGC频率与时间。如果FGC频繁且GC出来的量很少，那么spark.storage.memoryFraction这个值逐渐减少，spark.shuffle.memoryFraction这个值逐步调大，如果FGC频率降不小来那么需要逐步降低spark.shuffle.safetyFraction。

Rdd.persits(rdd)

Unpersits()

注意一点是，在代码里边当cache的RDD不需要被迭代了要立马手动unpersist，不要等到作业完成后然它自动释放。

App

一个作业：AM(3G+2core)+container(map(3G)、reduce(4G))

**附**

**平时使用平台时要注意的问题**

1. 对于在客户端使用spark-sql或者spark-shell时，作业跑完必须手动执行exit命令退出，如果不退出的话spark框架不会自动释放出申请到的资源，造成浪费。
2. hive客户端的使用，当不用的时候请执行exit命令退出hive 客户端，以免造成hive客户端升级打补丁后对于个别客户单不生效。
3. 避免运行大作业直接用打开hive客户端运行，直到结束。因为网络可能存在不稳定因素，如果你的大作业需要跑10个小时，但是跑到8个小时时，客户端网络突然不稳定断了下，那么你就白跑了8个小时需要重新跑，浪费集群资源。为了避免这种情况，跑大的作业必须使用 hive -f 挂后台跑。将脚本写到文件中a.sql中，然手执行 nohup hive -f a.sql &，日志可以直接看nohup文件。
4. 关于作业优先级的问题，临时提数、测试等作业只有权限提交到root.common队列，如果不是紧急特殊的作业我这边是不会为单个作业提高优先级的。
5. 关于集群管理界面的代理问题，我们集群部署在IDC机房，在我们办公网无法直接访问的。需要访问的话需要自己配置跳板机到本地的代理（具体的配置问题有需求的找显敬）。如果不配代理想临时查看想自己作业的情况那么可以访问这个地址http://10.12.0.23:8088/cluster（只对主备节点做了本地代理，如果你想看到每个节点的作业日志，那么你必须手动配置跳板机到本地的代理，并且配置好hosts映射）

6、关于代码中临时表的使用问题，之前发现有的同事使用的临时表非常多，有的一个作业用到十几个临时表。这里需要注意下，临时表尽量少用，并且用完后必须进行清除，大家可以看看temp.db库的情况。