
Spark Core面试篇01

随着Spark技术在企业中应用越来越广泛，Spark成为大数据开发必须掌握的技能。前期分享了很多关于Spark的学习视频和文章，为了进一步巩固和掌握Spark，在原有spark专刊基础上，新增《Spark面试2000题》专刊，题集包含基础概念、原理、编码开发、性能调优、运维、源代码以及Spark周边生态系统等。部分题集来源于互联网，由梅峰谷志愿者收集和整理，部分题集由梅峰谷志愿者结合生产实际碰到的问题设计出来，希望能给大家带来帮助。

一、简答题

1.Spark master使用zookeeper进行HA的，有哪些元数据保存在Zookeeper?

答：spark通过这个参数spark.deploy.zookeeper.dir指定master元数据在zookeeper中保存的位置，包括Worker，Driver和Application以及Executors。standby节点要从zk中，获得元数据信息，恢复集群运行状态，才能对外继续提供服务，作业提交资源申请等，在恢复前是不能接受请求的。另外，Master切换需要注意2点

1）在Master切换的过程中，所有的已经在运行的程序皆正常运行！因为Spark Application在运行前就已经通过Cluster Manager获得了计算资源，所以在运行时Job本身的调度和处理和Master是没有任何关系的！

2） 在Master的切换过程中唯一的影响是不能提交新的Job：一方面不能够提交新的应用程序给集群，因为只有Active Master才能接受新的程序的提交请求；另外一方面，已经运行的程序中也不能够因为Action操作触发新的Job的提交请求；

2.Spark master HA 主从切换过程不会影响集群已有的作业运行，为什么?

答：因为程序在运行之前，已经申请过资源了，driver和**Executors**通讯，不需要和master进行通讯的。

3.Spark on Mesos中，什么是的粗粒度分配，什么是细粒度分配，各自的优点和缺点是什么?

答：1）粗粒度：启动时就分配好资源， 程序启动，后续具体使用就使用分配好的资源，不需要再分配资源；好处：作业特别多时，资源复用率高，适合粗粒度；不好：容易资源浪费，假如一个job有1000个task，完成了999个，还有一个没完成，那么使用粗粒度，999个资源就会闲置在那里，资源浪费。2）细粒度分配：用资源的时候分配，用完了就立即回收资源，启动会麻烦一点，启动一次分配一次，会比较麻烦。

4.如何配置spark master的HA?

1)配置zookeeper

2)修改spark_env.sh文件,spark的master参数不在指定，添加如下代码到各个master节点

export SPARK_DAEMON_JAVA_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER -Dspark.deploy.zookeeper.url=zk01:2181,zk02:2181,zk03:2181 -Dspark.deploy.zookeeper.dir=/spark"

3) 将spark_env.sh分发到各个节点

4)找到一个master节点，执行./start-all.sh，会在这里启动主master,其他的master备节点，启动master命令: ./sbin/start-master.sh

5)提交程序的时候指定master的时候要指定三台master，例如

./spark-shell --master spark://master01:7077,master02:7077,master03:7077

5.Apache Spark有哪些常见的稳定版本，Spark1.6.0的数字分别代表什么意思?

答：常见的大的稳定版本有Spark 1.3,Spark1.6, Spark 2.0 ，**Spark1.6.0的数字含义**

1）第一个数字：1

major version : 代表大版本更新，一般都会有一些 api 的变化，以及大的优化或是一些结构的改变；

2）第二个数字：6

minor version : 代表小版本更新，一般会新加 api，或者是对当前的 api 就行优化，或者是其他内容的更新，比如说 WEB UI 的更新等等；

3）第三个数字：0

patch version ， 代表修复当前小版本存在的一些 bug，基本不会有任何 api 的改变和功能更新；记得有一个大神曾经说过，如果要切换 spark 版本的话，最好选 patch version 非 0 的版本，因为一般类似于 1.2.0, … 1.6.0 这样的版本是属于大更新的，有可能会有一些隐藏的 bug 或是不稳定性存在，所以最好选择 1.2.1, … 1.6.1 这样的版本。

通过版本号的解释说明，可以很容易了解到，spark2.1.1的发布时是针对大版本2.1做的一些bug修改，不会新增功能，也不会新增API，会比2.1.0版本更加稳定。

6.driver的功能是什么?

答： 1）一个Spark作业运行时包括一个Driver进程，也是作业的主进程，具有main函数，并且有SparkContext的实例，是程序的人口点；2）功能：负责向集群申请资源，向master注册信息，负责了作业的调度，，负责作业的解析、生成Stage并调度Task到Executor上。包括DAGScheduler，TaskScheduler。

7.spark的有几种部署模式，每种模式特点?

1）本地模式

Spark不一定非要跑在hadoop集群，可以在本地，起多个线程的方式来指定。将Spark应用以多线程的方式直接运行在本地，一般都是为了方便调试，本地模式分三

类

· local：只启动一个executor

· local[k]:启动k个executor

· local

- ：启动跟cpu数目相同的 executor

2)standalone模式

分布式部署集群， 自带完整的服务，资源管理和任务监控是Spark自己监控，这个模式也是其他模式的基础，

3)Spark on yarn模式

分布式部署集群，资源和任务监控交给yarn管理，但是目前仅支持粗粒度资源分配方式，包含cluster和client运行模式，cluster适合生产，driver运行在集群子节点，具有容错功能，client适合调试，dirver运行在客户端

4）Spark On Mesos模式。官方推荐这种模式（当然，原因之一是血缘关系）。正是由于Spark开发之初就考虑到支持Mesos，因此，目前而言，Spark运行在Mesos上会比运行在YARN上更加灵活，更加自然。用户可选择两种调度模式之一运行自己的应用程序：

1） 粗粒度模式（Coarse-grained Mode）：每个应用程序的运行环境由一个Dirver和若干个Executor组成，其中，每个Executor占用若干资源，内部可运行多个Task（对应多少个“slot”）。应用程序的各个任务正式运行之前，需要将运行环境中的资源全部申请好，且运行过程中要一直占用这些资源，即使不用，最后程序运行结束后，回收这些资源。

2） 细粒度模式（Fine-grained Mode）：鉴于粗粒度模式会造成大量资源浪费，Spark On Mesos还提供了另外一种调度模式：细粒度模式，这种模式类似于现在的云计算，思想是按需分配。

8.Spark技术栈有哪些组件，每个组件都有什么功能，适合什么应用场景？

答：可以画一个这样的技术栈图先，然后分别解释下每个组件的功能和场景

file:///E:/%E5%AE%89%E8%A3%85%E8%BD%AF%E4%BB%B6/%E6%9C%89%E9%81%93%E7%AC%94%E8%AE%B0%E6%96%87%E4%BB%B6/qq19B99AF2399E52F466CC3CF7E3B24ED5/dc318cd93346448487e9f423ce499b4b/d1d97571615f01111094fdcae4bed078.jpg

1）Spark core：是其它组件的基础，spark的内核，主要包含：有向循环图、RDD、Lingage、Cache、broadcast等，并封装了底层通讯框架，是Spark的基础。

2）SparkStreaming是一个对实时数据流进行高通量、容错处理的流式处理系统，可以对多种数据源（如Kdfka、Flume、Twitter、Zero和TCP 套接字）进行类似Map、Reduce和Join等复杂操作，将流式计算分解成一系列短小的批处理作业。

3）Spark sql：Shark是SparkSQL的前身，Spark SQL的一个重要特点是其能够统一处理关系表和RDD，使得开发人员可以轻松地使用SQL命令进行外部查询，

同时进行更复杂的数据分析

4）BlinkDB ： 是一个用于在海量数据上运行交互式 SQL 查询的大规模并行查询引擎，它允许用户通过权衡数据精度来提升查询响应时间，其数据的精度被控制在允许的误差范围内。

5）MLBase是Spark生态圈的一部分专注于机器学习，让机器学习的门槛更低，让一些可能并不了解机器学习的用户也能方便地使用MLbase。MLBase分为四部分：MLlib、MLI、ML Optimizer和MLRuntime。

6）GraphX是Spark中用于图和图并行计算

9.Spark中Work的主要工作是什么?

答：主要功能：管理当前节点内存，CPU的使用状况，接收master分配过来的资源指令，通过ExecutorRunner启动程序分配任务，worker就类似于包工头，管理分配新进程，做计算的服务，相当于process服务。需要注意的是：1）worker会不会汇报当前信息给master，worker心跳给master主要只有workid，它不会发送资源信息以心跳的方式给mater，master分配的时候就知道work，只有出现故障的时候才会发送资源。2）worker不会运行代码，具体运行的是Executor是可以运行具体appliaction写的业务逻辑代码，操作代码的节点，它不会运行程序的代码的。

10.Spark为什么比mapreduce快?

答：1）基于内存计算，减少低效的磁盘交互；2）高效的调度算法，基于DAG；3)容错机制Linage，精华部分就是DAG和Lingae

11.简单说一下hadoop和spark的shuffle相同和差异?

答：1）从 high-level 的角度来看，两者并没有大的差别。 都是将 mapper（Spark 里是 ShuffleMapTask）的输出进行 partition，不同的 partition 送到不同的 reducer（Spark 里 reducer 可能是下一个 stage 里的 ShuffleMapTask，也可能是 ResultTask）。Reducer 以内存作缓冲区，边 shuffle 边 aggregate 数据，等到数据 aggregate 好以后进行 reduce() （Spark 里可能是后续的一系列操作）。

2）从 low-level 的角度来看，两者差别不小。 Hadoop MapReduce 是 sort-based，进入 combine() 和 reduce() 的 records 必须先 sort。这样的好处在于 combine/reduce() 可以处理大规模的数据，因为其输入数据可以通过外排得到（mapper 对每段数据先做排序，reducer 的 shuffle 对排好序的每段数据做归并）。目前的 Spark 默认选择的是 hash-based，通常使用 HashMap 来对 shuffle 来的数据进行 aggregate，不会对数据进行提前排序。如果用户需要经过排序的数据，那么需要自己调用类似 sortByKey() 的操作；如果你是Spark 1.1的用户，可以将spark.shuffle.manager设置为sort，则会对数据进行排序。在Spark 1.2中，sort将作为默认的Shuffle实现。

3）从实现角度来看，两者也有不少差别。 Hadoop MapReduce 将处理流程划分出明显的几个阶段：map(), spill, merge, shuffle, sort, reduce() 等。每个阶段各司其职，可以按照过程式的编程思想来逐一实现每个阶段的功能。在 Spark 中，没有这样功能明确的阶段，只有不同的 stage 和一系列的 transformation()，所以 spill, merge, aggregate 等操作需要蕴含在 transformation() 中。

如果我们将 map 端划分数据、持久化数据的过程称为 shuffle write，而将 reducer 读入数据、aggregate 数据的过程称为 shuffle read。那么在 Spark 中，问题就变为怎么在 job 的逻辑或者物理执行图中加入 shuffle write 和 shuffle read 的处理逻辑？以及两个处理逻辑应该怎么高效实现？

Shuffle write由于不要求数据有序，shuffle write 的任务很简单：将数据 partition 好，并持久化。之所以要持久化，一方面是要减少内存存储空间压力，另一方面也是为了 fault-tolerance。

12.Mapreduce和Spark的都是并行计算，那么他们有什么相同和区别

答：两者都是用mr模型来进行并行计算:

1)hadoop的一个作业称为job，job里面分为map task和reduce task，每个task都是在自己的进程中运行的，当task结束时，进程也会结束。

2)spark用户提交的任务成为application，一个application对应一个sparkcontext，app中存在多个job，每触发一次action操作就会产生一个job。这些job可以并行或串行执行，每个job中有多个stage，stage是shuffle过程中DAGSchaduler通过RDD之间的依赖关系划分job而来的，每个stage里面有多个task，组成taskset有TaskSchaduler分发到各个executor中执行，executor的生命周期是和app一样的，即使没有job运行也是存在的，所以task可以快速启动读取内存进行计算。

3)hadoop的job只有map和reduce操作，表达能力比较欠缺而且在mr过程中会重复的读写hdfs，造成大量的io操作，多个job需要自己管理关系。

spark的迭代计算都是在内存中进行的，API中提供了大量的RDD操作如join，groupby等，而且通过DAG图可以实现良好的容错。

13.RDD机制?

答：rdd分布式弹性数据集，简单的理解成一种数据结构，是spark框架上的通用货币。

所有算子都是基于rdd来执行的，不同的场景会有不同的rdd实现类，但是都可以进行互相转换。

rdd执行过程中会形成dag图，然后形成lineage保证容错性等。 从物理的角度来看rdd存储的是block和node之间的映射。

14、spark有哪些组件?

答：主要有如下组件：

1）master：管理集群和节点，不参与计算。

2）worker：计算节点，进程本身不参与计算，和master汇报。

3）Driver：运行程序的main方法，创建spark context对象。

4）spark context：控制整个application的生命周期，包括dagsheduler和task scheduler等组件。

5）client：用户提交程序的入口。

15、spark工作机制?

答：用户在client端提交作业后，会由Driver运行main方法并创建spark context上下文。

执行add算子，形成dag图输入dagscheduler，按照add之间的依赖关系划分stage输入task scheduler。 task scheduler会将stage划分为task set分发到各个节点的executor中执行。

16、spark的优化怎么做?

答： spark调优比较复杂，但是大体可以分为三个方面来进行，1）平台层面的调优：防止不必要的jar包分发，提高数据的本地性，选择高效的存储格式如parquet，2）应用程序层面的调优：过滤操作符的优化降低过多小任务，降低单条记录的资源开销，处理数据倾斜，复用RDD进行缓存，作业并行化执行等等，3）JVM层面的调优：设置合适的资源量，设置合理的JVM，启用高效的序列化方法如kyro，增大off head内存等等

17.简要描述Spark分布式集群搭建的步骤

1）准备linux环境，设置集群搭建账号和用户组，设置ssh，关闭防火墙，关闭seLinux，配置host，hostname

2）配置jdk到环境变量

3）搭建hadoop集群，如果要做master ha，需要搭建zookeeper集群

修改hdfs-site.xml,hadoop_env.sh,yarn-site.xml,slaves等配置文件

4）启动hadoop集群，启动前要格式化namenode

5）配置spark集群，修改spark-env.xml，slaves等配置文件，拷贝hadoop相关配置到spark conf目录下

6)启动spark集群。

18.什么是RDD宽依赖和窄依赖?

RDD和它依赖的parent RDD(s)的关系有两种不同的类型，即窄依赖（narrow dependency）和宽依赖（wide dependency）。

1）窄依赖指的是每一个parent RDD的Partition最多被子RDD的一个Partition使用

2）宽依赖指的是多个子RDD的Partition会依赖同一个parent RDD的Partition

19.spark-submit的时候如何引入外部jar包

方法一：spark-submit –jars

根据spark官网，在提交任务的时候指定–jars，用逗号分开。这样做的缺点是每次都要指定jar包，如果jar包少的话可以这么做，但是如果多的话会很麻烦。

命令：spark-submit --master yarn-client --jars ***.jar,***.jar

方法二：extraClassPath

提交时在spark-default中设定参数，将所有需要的jar包考到一个文件里，然后在参数中指定该目录就可以了，较上一个方便很多：

spark.executor.extraClassPath=/home/hadoop/wzq_workspace/lib/* spark.driver.extraClassPath=/home/hadoop/wzq_workspace/lib/*

需要注意的是,你要在所有可能运行spark任务的机器上保证该目录存在，并且将jar包考到所有机器上。这样做的好处是提交代码的时候不用再写一长串jar了，缺点是要把所有的jar包都拷一遍。

20.cache和pesist的区别

答：1）cache和persist都是用于将一个RDD进行缓存的，这样在之后使用的过程中就不需要重新计算了，可以大大节省程序运行时间；2） cache只有一个默认的缓存级别MEMORY_ONLY ，cache调用了persist，而persist可以根据情况设置其它的缓存级别；3）executor执行的时候，默认60%做cache，40%做task操作，persist最根本的函数，最底层的函数

二、选择题

1. Spark 的四大组件下面哪个不是 (D)

A.Spark Streaming　B. Mlib

C Graphx　D.Spark R

2.下面哪个端口不是 spark 自带服务的端口 (C)

A.8080 B.4040 C.8090 D.18080

备注：8080：spark集群web ui端口，4040：sparkjob监控端口，18080：jobhistory端口

3.spark 1.4 版本的最大变化 (B)

A spark sql Release 版本　B .引入 Spark R

C DataFrame D.支持动态资源分配

4. Spark Job 默认的调度模式 (A)

A FIFO　B FAIR

C 无　D 运行时指定

5.哪个不是本地模式运行的个条件 (D)

A spark.localExecution.enabled=true

B 显式指定本地运行

C finalStage 无父 Stage

D partition默认值

6.下面哪个不是 RDD 的特点 (C)

A. 可分区　B 可序列化　C 可修改　D 可持久化

7. 关于广播变量，下面哪个是错误的 (D)

A 任何函数调用　B 是只读的

C 存储在各个节点　D 存储在磁盘或 HDFS

8. 关于累加器，下面哪个是错误的 (D)

A 支持加法 B 支持数值类型

C 可并行 D 不支持自定义类型

9.Spark 支持的分布式部署方式中哪个是错误的 (D)

A standalone B spark on mesos

C spark on YARN D Spark on local

10.Stage 的 Task 的数量由什么决定 (A)

A Partition B Job C Stage D TaskScheduler

11.下面哪个操作是窄依赖 (B)

A join B filter

C group D sort

12.下面哪个操作肯定是宽依赖 (C)

A map B flatMap

C reduceByKey D sample

13.spark 的 master 和 worker 通过什么方式进行通信的？ (D)

A http B nio C netty D Akka

14 默认的存储级别 (A)

A MEMORY_ONLY B MEMORY_ONLY_SER

C MEMORY_AND_DISK D MEMORY_AND_DISK_SER

15 spark.deploy.recoveryMode 不支持那种 (D)

A.ZooKeeper B. FileSystem

D NONE D Hadoop

16.下列哪个不是 RDD 的缓存方法 (C)

A persist() B Cache()

C Memory()

17.Task 运行在下来哪里个选项中 Executor 上的工作单元 (C)

A Driver program B. spark master

C.worker node D Cluster manager

18.hive 的元数据存储在 derby 和 MySQL 中有什么区别 (B)

A.没区别 B.多会话

C.支持网络环境 D数据库的区别

19.DataFrame 和 RDD 最大的区别 (B)

A.科学统计支持 B.多了 schema

C.存储方式不一样 D.外部数据源支持

20.Master 的 ElectedLeader 事件后做了哪些操作 (D)

A. 通知 driver B.通知 worker

C.注册 application D.直接 ALIVE

【Spark面试2000题41-70】Spark core面试篇02

这批Spark面试题由志愿者Taffry（某高校研究生）提供，非常感谢志愿者的优质题集，大家如果有好的面试题可以私信给群主（可加入志愿者群QQ群：233864572）。为确保题集质量，志愿者贡献出来的题集，群主及各位梅峰谷平台组成员会审核，个别地方会略加修改，还请志愿者理解。

一、面试30题

1.cache后面能不能接其他算子,它是不是action操作?

答：cache可以接其他算子，但是接了算子之后，起不到缓存应有的效果，因为会重新触发cache。

cache不是action操作

2.reduceByKey是不是action?

答：不是，很多人都会以为是action，reduce rdd是action

3.数据本地性是在哪个环节确定的?

具体的task运行在那他机器上，dag划分stage的时候确定的

4.RDD的弹性表现在哪几点?

1）自动的进行内存和磁盘的存储切换;

2）基于Lingage的高效容错;

3）task如果失败会自动进行特定次数的重试;

4）stage如果失败会自动进行特定次数的重试，而且只会计算失败的分片;

5）checkpoint和persist，数据计算之后持久化缓存

6）数据调度弹性，DAG TASK调度和资源无关

7）数据分片的高度弹性，a.分片很多碎片可以合并成大的，b.par

5.常规的容错方式有哪几种类型?

1）.数据检查点,会发生拷贝，浪费资源

2）.记录数据的更新，每次更新都会记录下来，比较复杂且比较消耗性能

6.RDD通过Linage（记录数据更新）的方式为何很高效?

1）lazy记录了数据的来源，RDD是不可变的，且是lazy级别的，且rDD

之间构成了链条，lazy是弹性的基石。由于RDD不可变，所以每次操作就

产生新的rdd，不存在全局修改的问题，控制难度下降，所有有计算链条

将复杂计算链条存储下来，计算的时候从后往前回溯

900步是上一个stage的结束，要么就checkpoint

2）记录原数据，是每次修改都记录，代价很大

如果修改一个集合，代价就很小，官方说rdd是

粗粒度的操作，是为了效率，为了简化，每次都是

操作数据集合，写或者修改操作，都是基于集合的

rdd的写操作是粗粒度的，rdd的读操作既可以是粗粒度的

也可以是细粒度，读可以读其中的一条条的记录。

3）简化复杂度，是高效率的一方面，写的粗粒度限制了使用场景

如网络爬虫，现实世界中，大多数写是粗粒度的场景

7.RDD有哪些缺陷?

1）不支持细粒度的写和更新操作（如网络爬虫），spark写数据是粗粒度的

所谓粗粒度，就是批量写入数据，为了提高效率。但是读数据是细粒度的也就是

说可以一条条的读

2）不支持增量迭代计算，Flink支持

8.说一说Spark程序编写的一般步骤?

答：初始化，资源，数据源，并行化，rdd转化，action算子打印输出结果或者也可以存至相应的数据存储介质，具体的可看下图：

file:///E:/%E5%AE%89%E8%A3%85%E8%BD%AF%E4%BB%B6/%E6%9C%89%E9%81%93%E7%AC%94%E8%AE%B0%E6%96%87%E4%BB%B6/qq19B99AF2399E52F466CC3CF7E3B24ED5/069fa7b471f54e038440faf63233acce/640.webp

9. Spark有哪两种算子?

答：Transformation（转化）算子和Action（执行）算子。

10. Spark提交你的jar包时所用的命令是什么?

答：spark-submit。

11. Spark有哪些聚合类的算子,我们应该尽量避免什么类型的算子?

答：在我们的开发过程中，能避免则尽可能避免使用reduceByKey、join、distinct、repartition等会进行shuffle的算子，尽量使用map类的非shuffle算子。这样的话，没有shuffle操作或者仅有较少shuffle操作的Spark作业，可以大大减少性能开销。

12. 你所理解的Spark的shuffle过程?

答：从下面三点去展开

1）shuffle过程的划分

2）shuffle的中间结果如何存储

3）shuffle的数据如何拉取过来

可以参考这篇博文：http://www.cnblogs.com/jxhd1/p/6528540.html

13. 你如何从Kafka中获取数据?

1)基于Receiver的方式

这种方式使用Receiver来获取数据。Receiver是使用Kafka的高层次Consumer API来实现的。receiver从Kafka中获取的数据都是存储在Spark Executor的内存中的，然后Spark Streaming启动的job会去处理那些数据。

2)基于Direct的方式

这种新的不基于Receiver的直接方式，是在Spark 1.3中引入的，从而能够确保更加健壮的机制。替代掉使用Receiver来接收数据后，这种方式会周期性地查询Kafka，来获得每个topic+partition的最新的offset，从而定义每个batch的offset的范围。当处理数据的job启动时，就会使用Kafka的简单consumer api来获取Kafka指定offset范围的数据

14. 对于Spark中的数据倾斜问题你有什么好的方案?

1）前提是定位数据倾斜，是OOM了，还是任务执行缓慢，看日志，看WebUI

2)解决方法，有多个方面

· 避免不必要的shuffle，如使用广播小表的方式，将reduce-side-join提升为map-side-join

·分拆发生数据倾斜的记录，分成几个部分进行，然后合并join后的结果

·改变并行度，可能并行度太少了，导致个别task数据压力大

·两阶段聚合，先局部聚合，再全局聚合

·自定义paritioner，分散key的分布，使其更加均匀

详细解决方案参考博文《Spark数据倾斜优化方法》

15.RDD创建有哪几种方式?

1).使用程序中的集合创建rdd

2).使用本地文件系统创建rdd

3).使用hdfs创建rdd，

4).基于数据库db创建rdd

5).基于Nosql创建rdd，如hbase

6).基于s3创建rdd，

7).基于数据流，如socket创建rdd

如果只回答了前面三种，是不够的，只能说明你的水平还是入门级的，实践过程中有很多种创建方式。

16.Spark并行度怎么设置比较合适

答：spark并行度，每个core承载2~4个partition,如，32个core，那么64~128之间的并行度，也就是

设置64~128个partion，并行读和数据规模无关，只和内存使用量和cpu使用

时间有关

17.Spark中数据的位置是被谁管理的?

答：每个数据分片都对应具体物理位置，数据的位置是被blockManager，无论

数据是在磁盘，内存还是tacyan，都是由blockManager管理

18.Spark的数据本地性有哪几种?

答：Spark中的数据本地性有三种：

a.PROCESS_LOCAL是指读取缓存在本地节点的数据

b.NODE_LOCAL是指读取本地节点硬盘数据

c.ANY是指读取非本地节点数据

通常读取数据PROCESS_LOCAL>NODE_LOCAL>ANY，尽量使数据以PROCESS_LOCAL或NODE_LOCAL方式读取。其中PROCESS_LOCAL还和cache有关，如果RDD经常用的话将该RDD cache到内存中，注意，由于cache是lazy的，所以必须通过一个action的触发，才能真正的将该RDD cache到内存中。

19.rdd有几种操作类型?

1）transformation，rdd由一种转为另一种rdd

2）action，

3）cronroller，crontroller是控制算子,cache,persist，对性能和效率的有很好的支持

三种类型，不要回答只有2中操作

19.rdd有几种操作类型?

1）transformation，rdd由一种转为另一种rdd

2）action，

3）cronroller，crontroller是控制算子,cache,persist，对性能和效率的有很好的支持

三种类型，不要回答只有2中操作

20.Spark如何处理不能被序列化的对象?

将不能序列化的内容封装成object

21.collect功能是什么，其底层是怎么实现的?

答：driver通过collect把集群中各个节点的内容收集过来汇总成结果，collect返回结果是Array类型的，collect把各个节点上的数据抓过来，抓过来数据是Array型，collect对Array抓过来的结果进行合并，合并后Array中只有一个元素，是tuple类型（KV类型的）的。

22.Spaek程序执行，有时候默认为什么会产生很多task，怎么修改默认task执行个数?

答：1）因为输入数据有很多task，尤其是有很多小文件的时候，有多少个输入

block就会有多少个task启动；2）spark中有partition的概念，每个partition都会对应一个task，task越多，在处理大规模数据的时候，就会越有效率。不过task并不是越多越好，如果平时测试，或者数据量没有那么大，则没有必要task数量太多。3）参数可以通过spark_home/conf/spark-default.conf配置文件设置:

spark.sql.shuffle.partitions 50 spark.default.parallelism 10

第一个是针对spark sql的task数量

第二个是非spark sql程序设置生效

23.为什么Spark Application在没有获得足够的资源，job就开始执行了，可能会导致什么什么问题发生?

答：会导致执行该job时候集群资源不足，导致执行job结束也没有分配足够的资源，分配了部分Executor，该job就开始执行task，应该是task的调度线程和Executor资源申请是异步的；如果想等待申请完所有的资源再执行job的：需要将spark.scheduler.maxRegisteredResourcesWaitingTime设置的很大；

spark.scheduler.minRegisteredResourcesRatio 设置为1，但是应该结合实际考虑

否则很容易出现长时间分配不到资源，job一直不能运行的情况。

24.map与flatMap的区别

map：对RDD每个元素转换，文件中的每一行数据返回一个数组对象

flatMap：对RDD每个元素转换，然后再扁平化

将所有的对象合并为一个对象，文件中的所有行数据仅返回一个数组

对象，会抛弃值为null的值

25.列举你常用的action?

collect，reduce,take,count,saveAsTextFile等

26.Spark为什么要持久化，一般什么场景下要进行persist操作?

为什么要进行持久化?

spark所有复杂一点的算法都会有persist身影,spark默认数据放在内存，spark很多内容都是放在内存的，非常适合高速迭代，1000个步骤

只有第一个输入数据，中间不产生临时数据，但分布式系统风险很高，所以容易出错，就要容错，rdd出错或者分片可以根据血统算出来，如果没有对父rdd进行persist 或者cache的化，就需要重头做。

以下场景会使用persist

1）某个步骤计算非常耗时，需要进行persist持久化

2）计算链条非常长，重新恢复要算很多步骤，很好使，persist

3）checkpoint所在的rdd要持久化persist，

lazy级别，框架发现有checnkpoint，checkpoint时单独触发一个job，需要重算一遍，checkpoint前

要持久化，写个rdd.cache或者rdd.persist，将结果保存起来，再写checkpoint操作，这样执行起来会非常快，不需要重新计算rdd链条了。checkpoint之前一定会进行persist。

4）shuffle之后为什么要persist，shuffle要进性网络传输，风险很大，数据丢失重来，恢复代价很大

5）shuffle之前进行persist，框架默认将数据持久化到磁盘，这个是框架自动做的。

27.为什么要进行序列化

序列化可以减少数据的体积，减少存储空间，高效存储和传输数据，不好的是使用的时候要反序列化，非常消耗CPU

28.介绍一下join操作优化经验?

答：join其实常见的就分为两类： map-side join 和 reduce-side join。当大表和小表join时，用map-side join能显著提高效率。将多份数据进行关联是数据处理过程中非常普遍的用法，不过在分布式计算系统中，这个问题往往会变的非常麻烦，因为框架提供的 join 操作一般会将所有数据根据 key 发送到所有的 reduce 分区中去，也就是 shuffle 的过程。造成大量的网络以及磁盘IO消耗，运行效率极其低下，这个过程一般被称为 reduce-side-join。如果其中有张表较小的话，我们则可以自己实现在 map 端实现数据关联，跳过大量数据进行 shuffle 的过程，运行时间得到大量缩短，根据不同数据可能会有几倍到数十倍的性能提升。

备注：这个题目面试中非常非常大概率见到，务必搜索相关资料掌握，这里抛砖引玉。

29.介绍一下cogroup rdd实现原理，你在什么场景下用过这个rdd?

答：cogroup的函数实现:这个实现根据两个要进行合并的两个RDD操作,生成一个CoGroupedRDD的实例,这个RDD的返回结果是把相同的key中两个RDD分别进行合并操作,最后返回的RDD的value是一个Pair的实例,这个实例包含两个Iterable的值,第一个值表示的是RDD1中相同KEY的值,第二个值表示的是RDD2中相同key的值.由于做cogroup的操作,需要通过partitioner进行重新分区的操作,因此,执行这个流程时,需要执行一次shuffle的操作(如果要进行合并的两个RDD的都已经是shuffle后的rdd,同时他们对应的partitioner相同时,就不需要执行shuffle,)，

场景：表关联查询

30 下面这段代码输出结果是什么?

```
--------------------------
def joinRdd(sc:SparkContext) {
val name= Array(
```

```
Tuple2(1,"spark"),
Tuple2(2,"tachyon"),
Tuple2(3,"hadoop")
)
val score= Array(
Tuple2(1,100),
Tuple2(2,90),
Tuple2(3,80)
)
val namerdd=sc.parallelize(name);
val scorerdd=sc.parallelize(score);
val result = namerdd.join(scorerdd);
result .collect.foreach(println);
}
```

答案:

```
(1,(Spark,100))
(2,(tachyon,90))
(3,(hadoop,80))
```

【Spark面试2000题71-100】Spark core面试篇03

Spark Core是Spark的基石，有很多知识点，面试题集的知识点比较跳跃和分散，建议系统学习了Spark知识再看面试题集。今天继续放送最新整理和设计的《Spark面试2000题》题集，仅供参考学习。本篇博文属于梅峰谷原创，转载请注明出处，如果您觉得对您有帮助，请不要吝啬点赞，你的赞，是志愿者们坚持的动力，是早日做出2000道高质量Spark面试题的动力，如有不准确的地方，请留言说明。

一、面试30题(第71-100题)

1.Spark使用parquet文件存储格式能带来哪些好处?

1) 如果说HDFS 是大数据时代分布式文件系统首选标准，那么parquet则是整个大数据时代文件存储格式实时首选标准

2) 速度更快：从使用spark sql操作普通文件CSV和parquet文件速度对比上看，绝大多数情况

会比使用csv等普通文件速度提升10倍左右，在一些普通文件系统无法在spark上成功运行的情况

下，使用parquet很多时候可以成功运行

3) parquet的压缩技术非常稳定出色，在spark sql中对压缩技术的处理可能无法正常的完成工作

（例如会导致lost task，lost executor）但是此时如果使用parquet就可以正常的完成

4) 极大的减少磁盘I/o,通常情况下能够减少75%的存储空间，由此可以极大的减少spark sql处理

数据的时候的数据输入内容，尤其是在spark1.6x中有个下推过滤器在一些情况下可以极大的

减少磁盘的IO和内存的占用，（下推过滤器）

5) spark 1.6x parquet方式极大的提升了扫描的吞吐量，极大提高了数据的查找速度spark1.6和spark1.5x相比而言，提升了大约1倍的速度，在spark1.6X中，操作parquet时候cpu也进行了极大的优化，有效的降低了cpu

6) 采用parquet可以极大的优化spark的调度和执行。我们测试spark如果用parquet可以有效的减少stage的执行消耗，同时可以优化执行路径

2.Executor之间如何共享数据?

答：基于hdfs或者基于tachyon

3.Spark累加器有哪些特点?

1）累加器在全局唯一的，只增不减，记录全局集群的唯一状态

2）在exe中修改它，在driver读取

3）executor级别共享的，广播变量是task级别的共享

两个application不可以共享累加器，但是同一个app不同的job可以共享

4.如何在一个不确定的数据规模的范围内进行排序?

为了提高效率，要划分划分，划分的范围并且是有序的

要么有序，要么降序?

水塘抽样：目的是从一个集合中选取，集合非常答，适合内存

无法容纳数据的时候使用

从N中抽取出K个，N是随机数

5.spark hashParitioner的弊端是什么?

答:HashPartitioner分区的原理很简单，对于给定的key，计算其hashCode，并除于分区的个数取余，如果余数小于0，则用余数+分区的个数，最后返回的值就是这个key所属的分区ID；弊端是数据不均匀，容易导致数据倾斜，极端情况下某几个分区会拥有rdd的所有数据

6.RangePartitioner分区的原理?

答:RangePartitioner分区则尽量保证每个分区中数据量的均匀，而且分区与分区之间是有序的，也就是说一个分区中的元素肯定都是比另一个分区内的元素小或者大；但是分区内的元素是不能保证顺序的。简单的说就是将一定范围内的数映射到某一个分区内。其原理是水塘抽样。可以参考这篇博文

https://www.iteblog.com/archives/1522.html

7.介绍parition和block有什么关联关系?

答：1）hdfs中的block是分布式存储的最小单元，等分，可设置冗余，这样设计有一部分磁盘空间的浪费，但是整齐的block大小，便于快速找到、读取对应的内容；2）Spark中的partion是弹性分布式数据集RDD的最小单元，RDD是由分布在各个节点上的partion组成的。partion是指的spark在计算过程中，生成的数据在计算空间内最小单元，同一份数据（RDD）的partion大小不一，数量不定，是根据application里的算子和最初读入的数据分块数量决定；3）block位于存储空间、partion位于计算空间，block的大小是固定的、partion大小是不固定的，是从2个不同的角度去看数据。

8.Spark应用程序的执行过程是什么?

1)构建Spark Application的运行环境（启动SparkContext），SparkContext向资源管理器（可以是Standalone、Mesos或YARN）注册并申请运行Executor资源；

2).资源管理器分配Executor资源并启动StandaloneExecutorBackend，Executor运行情况将随着心跳发送到资源管理器上；

3).SparkContext构建成DAG图，将DAG图分解成Stage，并把Taskset发送给Task Scheduler。Executor向SparkContext申请Task，Task Scheduler将Task发放给Executor运行同时SparkContext将应用程序代码发放给Executor。

4).Task在Executor上运行，运行完毕释放所有资源。

9.hbase预分区个数和spark过程中的reduce个数相同么

答：和spark的map个数相同，reduce个数如果没有设置和reduce前的map数相同。

10.如何理解Standalone模式下，Spark资源分配是粗粒度的?

答：spark默认情况下资源分配是粗粒度的，也就是说程序在提交时就分配好资源，后面执行的时候

使用分配好的资源，除非资源出现了故障才会重新分配。比如Spark shell启动，已提交，一注册，哪怕没有任务，worker都会分配资源给executor。

11.Spark如何自定义partitioner分区器?

答：1）spark默认实现了HashPartitioner和RangePartitioner两种分区策略，我们也可以自己扩展分区策略，自定义分区器的时候继承org.apache.spark.Partitioner类，实现类中的三个方法

def numPartitions: Int：这个方法需要返回你想要创建分区的个数；

def getPartition(key: Any): Int：这个函数需要对输入的key做计算，然后返回该key的分区ID，范围一定是0到numPartitions-1；

equals()：这个是Java标准的判断相等的函数，之所以要求用户实现这个函数是因为Spark内部会比较两个RDD的分区是否一样。

2）使用，调用parttionBy方法中传入自定义分区对象

参考：http://blog.csdn.net/high2011/article/details/68491115

12.spark中task有几种类型?

答：2种类型：1）result task类型，最后一个task，2是shuffleMapTask类型，除了最后一个task都是

13.union操作是产生宽依赖还是窄依赖?

答：窄依赖

14.rangePartioner分区器特点?

答：rangePartioner尽量保证每个分区中数据量的均匀，而且分区与分区之间是有序的，一个分区中的元素肯定都是比另一个分区内的元素小或者大；但是分区内的元素是不能保证顺序的。

简单的说就是将一定范围内的数映射到某一个分区内。RangePartitioner作用：将一定范围内的数映射到某一个分区内，在实现中，分界的算法尤为重要。算法对应的函数是rangeBounds

15.什么是二次排序，你是如何用spark实现二次排序的？（互联网公司常面）

答：就是考虑2个维度的排序，key相同的情况下如何排序，参考博文：http://blog.csdn.net/sundujing/article/details/51399606

16.如何使用Spark解决TopN问题？（互联网公司常面）

答：常见的面试题,参考博文：http://www.cnblogs.com/yurunmiao/p/4898672.html

17.如何使用Spark解决分组排序问题？（互联网公司常面）

组织数据形式：

aa 11

bb 11

cc 34

aa 22

bb 67

cc 29

aa 36

bb 33

cc 30

aa 42

bb 44

cc 49

需求：

1、对上述数据按key值进行分组

2、对分组后的值进行排序

3、截取分组后值得top 3位以key-value形式返回结果

答案：如下

```
----------------------
val groupTopNRdd = sc.textFile("hdfs://db02:8020/user/hadoop/groupsorttop/groupsorttop.data")
groupTopNRdd.map(_.split(" ")).map(x => (x(0),x(1))).groupByKey().map(
x => {
val xx = x._1
val yy = x._2
(xx,yy.toList.sorted.reverse.take(3))
}
).collect
----------------------
```

18.窄依赖父RDD的partition和子RDD的parition是不是都是一对一的关系？

答：不一定，除了一对一的窄依赖，还包含一对固定个数的窄依赖（就是对父RDD的依赖的Partition的数量不会随着RDD数量规模的改变而改变），比如join操作的每个partiion仅仅和已知的partition进行join，这个join操作是窄依赖，依赖固定数量的父rdd，因为是确定的partition关系

19.Hadoop中，Mapreduce操作的mapper和reducer阶段相当于spark中的哪几个算子？

答：相当于spark中的map算子和reduceByKey算子，当然还是有点区别的,MR会自动进行排序的，spark要看你用的是什么partitioner

20.什么是shuffle，以及为什么需要shuffle？

shuffle中文翻译为洗牌，需要shuffle的原因是：某种具有共同特征的数据汇聚到一个计算节点上进行计算

21.不需要排序的hash shuffle是否一定比需要排序的sort shuffle速度快？

答：不一定！！当数据规模小，Hash shuffle快于Sorted Shuffle数据规模大的时候；当数据量大，sorted Shuffle会比Hash shuffle快很多，因为数量大的有很多小文件，不均匀，甚至出现数据倾斜，消耗内存大，1.x之前spark使用hash，适合处理中小规模，1.x之后，增加了Sorted shuffle，Spark更能胜任大规模处理了。

22.Spark中的HashShufle的有哪些不足？

答：1）shuffle产生海量的小文件在磁盘上，此时会产生大量耗时的、低效的IO操作；2）.容易导致内存不够用，由于内存需要保存海量的文件操作句柄和临时缓存信息，如果数据处理规模比较大的化，容易出现OOM；3）容易出现数据倾斜，导致OOM

23.conslidate是如何优化Hash shuffle时在map端产生的小文件？

答：1）conslidate为了解决Hash Shuffle同时打开过多文件导致Writer handler内存使用过大以及产生过多文件导致大量的随机读写带来的低效磁盘IO；2）conslidate根据CPU的个数来决定

每个task shuffle map端产生多少个文件，假设原来有10个task，100个reduce，每个CPU有10个CPU

那么使用hash shuffle会产生10*100=1000个文件，conslidate产生10*10=100个文件

备注：conslidate部分减少了文件和文件句柄，并行读很高的情况下（task很多时）还是会很多文件

24.Sort-basesd shuffle产生多少个临时文件

答：2*Map阶段所有的task数量，Mapper阶段中并行的Partition的总数量，其实就是Mapper端task

25.Sort-based shuffle的缺陷?

1) 如果mapper中task的数量过大，依旧会产生很多小文件，此时在shuffle传递数据的过程中reducer段，reduce会需要同时大量的记录进行反序列化，导致大量的内存消耗和GC的巨大负担，造成系统缓慢甚至崩溃

2）如果需要在分片内也进行排序，此时需要进行mapper段和reducer段的两次排序

26.Spark shell启动时会启动derby?

答： spark shell启动会启动spark sql，spark sql默认使用derby保存元数据，但是尽量不要用derby，它是单实例，不利于开发。会在本地生成一个文件metastore_db,如果启动报错，就把那个文件给删了 ，derby数据库是单实例，不能支持多个用户同时操作，尽量避免使用

27.spark.default.parallelism这个参数有什么意义，实际生产中如何设置?

答：1）参数用于设置每个stage的默认task数量。这个参数极为重要，如果不设置可能会直接影响你的Spark作业性能；2）很多人都不会设置这个参数，会使得集群非常低效，你的cpu，内存再多，如果task始终为1，那也是浪费，spark官网建议task个数为CPU的核数*executor的个数的2~3倍。

28.spark.storage.memoryFraction参数的含义,实际生产中如何调优?

答：1）用于设置RDD持久化数据在Executor内存中能占的比例，默认是0.6,，默认Executor 60%的内存，可以用来保存持久化的RDD数据。根据你选择的不同的持久化策略，如果内存不够时，可能数据就不会持久化，或者数据会写入磁盘。2）如果持久化操作比较多，可以提高spark.storage.memoryFraction参数，使得更多的持久化数据保存在内存中，提高数据的读取性能，如果shuffle的操作比较多，有很多的数据读写操作到JVM中，那么应该调小一点，节约出更多的内存给JVM，避免过多的JVM gc发生。在web ui中观察如果发现gc时间很长，可以设置spark.storage.memoryFraction更小一点。

29.spark.shuffle.memoryFraction参数的含义，以及优化经验?

答：1）spark.shuffle.memoryFraction是shuffle调优中 重要参数，shuffle从上一个task拉去数据过来，要在Executor进行聚合操作，聚合操作时使用Executor内存的比例由该参数决定，默认是20%

如果聚合时数据超过了该大小，那么就会spill到磁盘，极大降低性能；2）如果Spark作业中的RDD持久化操作较少，shuffle操作较多时，建议降低持久化操作的内存占比，提高shuffle操作的内存占比比例，避免shuffle过程中数据过多时内存不够用，必须溢写到磁盘上，降低了性能。此外，如果发现作业由于频繁的gc导致运行缓慢，意味着task执行用户代码的内存不够用，那么同样建议调低这个参数的值

30.介绍一下你对Unified Memory Management内存管理模型的理解?

答：Spark中的内存使用分为两部分：执行（execution）与存储（storage）。执行内存主要用于shuffles、joins、sorts和aggregations，存储内存则用于缓存或者跨节点的内部数据传输。

1.6之前，对于一个Executor,内存都有哪些部分构成：

1）ExecutionMemory。这片内存区域是为了解决 shuffles,joins, sorts and aggregations 过程中为了避免频繁IO需要的buffer。 通过spark.shuffle.memoryFraction(默认 0.2) 配置。

2）StorageMemory。这片内存区域是为了解决 block cache(就是你显示调用dd.cache, rdd.persist等方法), 还有就是broadcasts,以及task results的存储。可以通过参数 spark.storage.memoryFraction(默认0.6)。设置

3）OtherMemory。给系统预留的，因为程序本身运行也是需要内存的。 (默认为0.2).

传统内存管理的不足：

1).Shuffle占用内存0.2*0.8，内存分配这么少，可能会将数据spill到磁盘，频繁的磁盘IO是很大的负担，Storage内存占用0.6，主要是为了迭代处理。传统的Spark内存分配对操作人的要求非常高。（Shuffle分配内存：ShuffleMemoryManager, TaskMemoryManager,ExecutorMemoryManager）一个Task获得全部的Execution的Memory，其他Task过来就没有内存了，只能等待。

2).默认情况下，Task在线程中可能会占满整个内存，分片数据特别大的情况下就会出现这种情况，其他Task没有内存了，剩下的cores就空闲了，这是巨大的浪费。这也是人为操作的不当造成的。

3).MEMORY_AND_DISK_SER的storage方式，获得RDD的数据是一条条获取，iterator的方式。如果内存不够（spark.storage.unrollFraction），unroll的读取数据过程，就是看内存是否足够，如果足够，就下一条。unroll的space是从Storage的内存空间中获得的。unroll的方式失败，就会直接放磁盘。

4). 默认情况下，Task在spill到磁盘之前，会将部分数据存放到内存上，如果获取不到内存，就不会执行。永无止境的等待，消耗CPU和内存。

在此基础上，Spark提出了UnifiedMemoryManager，不再分ExecutionMemory和Storage Memory,实际上还是分的，只不过是Execution Memory访问Storage Memory，Storage Memory也可以访问Execution Memory，如果内存不够，就会去借。

【Spark面试2000题101-130】Spark on Yarn面试篇04

本篇题集主要是Spark on Yarn相关的面试题，主要涉及Spark on Yarn、Yarn、Mapreduce相关面试题。

一、面试题30题

1.MRV1有哪些不足?

1)可扩展性（对于变化的应付能力）

a) JobTracker内存中保存用户作业的信息

b) JobTracker使用的是粗粒度的锁

2)可靠性和可用性

a) JobTracker失效会多事集群中所有的运行作业，用户需手动重新提交和恢复工作流

3)对不同编程模型的支持

HadoopV1以MapReduce为中心的设计虽然能支持广泛的用例，但是并不适合所有大型计算,如storm，spark

2.描述Yarn执行一个任务的过程?

1）客户端client向ResouceManager提交Application，ResouceManager接受Application

并根据集群资源状况选取一个node来启动Application的任务调度器driver（ApplicationMaster）

2）ResouceManager找到那个node，命令其该node上的nodeManager来启动一个新的

JVM进程运行程序的driver（ApplicationMaster）部分，driver（ApplicationMaster）启动时会首先向ResourceManager注册，说明由自己来负责当前程序的运行

3）driver（ApplicationMaster）开始下载相关jar包等各种资源，基于下载的jar等信息决定向ResourceManager申请具体的资源内容。

4）ResouceManager接受到driver（ApplicationMaster）提出的申请后，会最大化的满足

资源分配请求，并发送资源的元数据信息给driver（ApplicationMaster）；

5）driver（ApplicationMaster）收到发过来的资源元数据信息后会根据元数据信息发指令给具体

机器上的NodeManager，让其启动具体的container。

6）NodeManager收到driver发来的指令，启动container，container启动后必须向driver（ApplicationMaster）注册。

7）driver（ApplicationMaster）收到container的注册，开始进行任务的调度和计算，直到

任务完成。

补充：如果ResourceManager第一次没有能够满足driver（ApplicationMaster）的资源请求

，后续发现有空闲的资源，会主动向driver（ApplicationMaster）发送可用资源的元数据信息

以提供更多的资源用于当前程序的运行。

3.Yarn中的container是由谁负责销毁的，在Hadoop Mapreduce中container可以复用么?

答：ApplicationMaster负责销毁，在Hadoop Mapreduce不可以复用，在spark on yarn程序container可以复用

4.提交任务时，如何指定Spark Application的运行模式?

1）cluster模式：./spark-submit --class xx.xx.xx --master yarn --deploy-mode cluster xx.jar

2) client模式:./spark-submit --class xx.xx.xx --master yarn --deploy-mode client xx.jar

5. 不启动Spark集群Master和work服务，可不可以运行Spark程序?

答：可以，只要资源管理器第三方管理就可以，如由yarn管理，spark集群不启动也可以使用spark；spark集群启动的是work和master，这个其实就是资源管理框架，yarn中的resourceManager相当于master，NodeManager相当于worker，做计算是Executor，和spark集群的work和manager可以没关系，归根接底还是JVM的运行，只要所在的JVM上安装了spark就可以。

6.Spark中的4040端口由什么功能?

答：收集Spark作业运行的信息

7.spark on yarn Cluster 模式下，ApplicationMaster和driver是在同一个进程么？

答：是,driver 位于ApplicationMaster进程中。该进程负责申请资源，还负责监控程序、资源的动态情况。

8.如何使用命令查看application运行的日志信息

答：yarn logs -applicationId <app ID>

9.Spark on Yarn 模式有哪些优点？

1)与其他计算框架共享集群资源（eg.Spark框架与MapReduce框架同时运行，如果不用Yarn进行资源分配，MapReduce分到的内存资源会很少，效率低下）；资源按需分配，进而提高集群资源利用等。

2)相较于Spark自带的Standalone模式，Yarn的资源分配更加细致

3)Application部署简化，例如Spark，Storm等多种框架的应用由客户端提交后，由Yarn负责资源的管理和调度，利用Container作为资源隔离的单位，以它为单位去使用内存,cpu等。

4)Yarn通过队列的方式，管理同时运行在Yarn集群中的多个服务，可根据不同类型的应用程序负载情况，调整对应的资源使用量，实现资源弹性管理。

10.谈谈你对container的理解？

1）Container作为资源分配和调度的基本单位，其中封装了的资源如内存，CPU，磁盘，网络带宽等。 目前yarn仅仅封装内存和CPU

2)Container由ApplicationMaster向ResourceManager申请的，由ResouceManager中的资源调度器异步分配给ApplicationMaster

3) Container的运行是由ApplicationMaster向资源所在的NodeManager发起的，Container运行时需提供内部执行的任务命令.

11.运行在yarn中Application有几种类型的container？

1） 运行ApplicationMaster的Container：这是由ResourceManager（向内部的资源调度器）申请和启动的，用户提交应用程序时，可指定唯一的ApplicationMaster所需的资源；

2） 运行各类任务的Container：这是由ApplicationMaster向ResourceManager申请的，并由ApplicationMaster与NodeManager通信以启动之。

12.Spark on Yarn架构是怎么样的？（要会画哦，这个图）

Yarn提到的App Master可以理解为Spark中Standalone模式中的driver。Container中运行着Executor,在Executor中以多线程并行的方式运行Task。运行过程和第二题相似。

13.Executor启动时，资源通过哪几个参数指定？

1)num-executors是executor的数量

2)executor-memory 是每个executor使用的内存

3)executor-cores 是每个executor分配的CPU

14.为什么会产生yarn，解决了什么问题，有什么优势？

1)为什么产生yarn，针对MRV1的各种缺陷提出来的资源管理框架

2)解决了什么问题，有什么优势，参考这篇博文：http://www.aboutyun.com/forum.php?mod=viewthread&tid=6785

15.Mapreduce的执行过程?

阶段1：input/map/partition/sort/spill

阶段2：mapper端merge

阶段3：reducer端merge/reduce/output

详细过程参考这个http://www.cnblogs.com/hipercomer/p/4516581.html

16.一个task的map数量由谁来决定?

一般情况下，在输入源是文件的时候，一个task的map数量由splitSize来决定的，那么splitSize是由以下几个来决定的

goalSize = totalSize / mapred.map.tasks

inSize = max {mapred.min.split.size, minSplitSize}

splitSize = max (minSize, min(goalSize, dfs.block.size))

一个task的reduce数量，由partition决定。

17.reduce后输出的数据量有多大?

并不是想知道确切的数据量有多大这个，而是想问你，MR的执行机制，开发完程序，有没有认真评估程序运行效率

1）用于处理redcue任务的资源情况，如果是MRV1的话，分了多少资源给map，多少个reduce

如果是MRV2的话，可以提一下，集群有分了多少内存、CPU给yarn做计算 。

2）结合实际应用场景回答，输入数据有多大，大约多少条记录，做了哪些逻辑操作，输出的时候有多少条记录，执行了多久，reduce执行时候的数据有没有倾斜等

3）再提一下，针对mapReduce做了哪几点优化，速度提升了多久，列举1,2个优化点就可以

18.你的项目提交到job的时候数据量有多大?

答：1）回答出数据是什么格式，有没有采用什么压缩，采用了压缩的话，压缩比大概是多少；2）文件大概多大：大概起了多少个map，起了多少个reduce，map阶段读取了多少数据，reduce阶段读取了多少数据，程序大约执行了多久，3）集群什么规模，集群有多少节点，多少内存，多少CPU核数等。把这些点回答进去，而不是给个数字了事。

19.你们提交的job任务大概有多少个？这些job执行完大概用多少时间?

还是考察你开发完程序有没有认真观察过程序的运行，有没有评估程序运行的效率

20.你们业务数据量多大？有多少行数据?

这个也是看你们有没有实际的经验,对于没有实战的同学，请把回答的侧重点放在MR的运行机制上面，

MR运行效率方面，以及如何优化MR程序（看别人的优化demo，然后在虚拟机上拿demo做一下测试）。

22.如何杀死一个正在运行的job

杀死一个job

MRV1：Hadoop job kill jobid

YARN: yarn application -kill applicationId

23.列出你所知道的调度器，说明其工作原理

a) Fifo schedular 默认的调度器 先进先出

b) Capacity schedular 计算能力调度器 选择占用内存小 优先级高的

c) Fair schedular 调肚脐 公平调度器 所有job 占用相同资源

24.YarnClient模式下，执行Spark SQL报这个错，Exception in thread "Thread-2" java.lang.OutOfMemoryError: PermGen space，但是在Yarn Cluster模式下正常运行，可能是什么原因?

1）原因查询过程中调用的是Hive的获取元数据信息、SQL解析，并且使用Cglib等进行序列化反序列化，中间可能产生较多的class文件，导致JVM中的持久代使用较多

Cluster模式的持久代默认大小是64M，Client模式的持久代默认大小是32M，而Driver端进行SQL处理时，其持久代的使用可能会达到90M，导致OOM溢出，任务失败。

yarn-cluster模式下出现，yarn-client模式运行时倒是正常的，原来在$SPARK_HOME/bin/spark-class文件中已经设置了持久代大小：

JAVA_OPTS="-XX:MaxPermSize=256m $OUR_JAVA_OPTS"

2）解决方法:在Spark的conf目录中的spark-defaults.conf里，增加对Driver的JVM配置，因为Driver才负责SQL的解析和元数据获取。配置如下：

spark.driver.extraJavaOptions -XX:PermSize=128M -XX:MaxPermSize=256M

25.spark.driver.extraJavaOptions这个参数是什么意思，你们生产环境配了多少?

传递给executors的JVM选项字符串。例如GC设置或者其它日志设置。注意，在这个选项中设置Spark属性或者堆大小是不合法的。Spark属性需要用SparkConf对象或者spark-submit脚本用到的spark-defaults.conf文件设置。堆内存可以通过spark.executor.memory设置

26.导致Executor产生FULL gc 的原因，可能导致什么问题?

答：可能导致Executor僵死问题，海量数据的shuffle和数据倾斜等都可能导致full gc。以shuffle为例，伴随着大量的Shuffle写操作，JVM的新生代不断GC，Eden Space写满了就往Survivor Space写，同时超过一定大小的数据会直接写到老生代，当新生代写满了之后，也会把老的数据搞到老生代，如果老生代空间不足了，就触发FULL GC，还是空间不够，那就OOM错误了，此时线程被Blocked，导致整个Executor处理数据的进程被卡住

27.Combiner 和partition的作用

combine分为map端和reduce端，作用是把同一个key的键值对合并在一起，可以自定义的。combine函数把一个map函数产生的<key,value>对（多个key,value）合并成一个新<key2,value2>.将新的<key2,value2>作为输入到reduce函数中这个value2亦可称之为values，因为有多个。这个合并的目的是为了减少网络传输。partition是分割map每个节点的结果，按照key分别映射给不同的reduce，也是可以自定义的。这里其实可以理解归类。我们对于错综复杂的数据归类。比如在动物园里有牛羊鸡鸭鹅，他们都是混在一起的，但是到了晚上他们就各自牛回牛棚，羊回羊圈，鸡回鸡窝。partition的作用就是把这些数据归类。只不过在写程序的时候，mapreduce使用哈希HashPartitioner帮我们归类了。这个我们也可以自定义。shuffle就是map和reduce之间的过程，包含了两端的combine和partition。Map的结果，会通过partition分发到Reducer上，Reducer做完Reduce操作后，通OutputFormat，进行输出shuffle阶段的主要函数是fetchOutputs(),这个函数的功能就是将map阶段的输出，copy到reduce 节点本地

28.Spark执行任务时出现java.lang.OutOfMemoryError: GC overhead limit exceeded和java.lang.OutOfMemoryError: java heap space原因和解决方法?

答：原因：加载了太多资源到内存，本地的性能也不好，gc时间消耗的较多

解决方法：

1）增加参数，-XX:-UseGCOverheadLimit，关闭这个特性，同时增加heap大小，-Xmx1024m

2）下面这个两个参数调大点

export SPARK_EXECUTOR_MEMORY=6000M

export SPARK_DRIVER_MEMORY=7000M

可以参考这个：http://www.cnblogs.com/hucn/p/3572384.html

29.请列出在你以前工作中所使用过的开发map /reduce的语言

答：java，Scala，Python，shell

30.你认为/etc/hosts配置错误，会对集群有什么影响?

答：1）直接导致域名没法解析，主节点与子节点，子节点与子节点没法正常通讯，2）间接导致配置错误的相关节点删的服务不正常，甚至没法启动，job执行失败等等

Spark Core面试篇05

原创 2017-06-12 梅峰谷 大数据梅峰谷

Spark RDD是Spark的编程基础，掌握RDD以及RDD编程技巧是企业实际开发的必备技能，本篇整理RDD常见的问题，汇编成题，以加深对RDD及RDD编程的理解。先把题目列举出来，各位感兴趣的自己去做一遍把，下一篇梅峰谷会通过网盘的方式，把答案公布出来，感兴趣的童鞋请及时关注。

1.scala中private 与 private[this] 修饰符的区别?

2.scala中内部类和java中的内部类区别

3.Spark中standalone模式特点，有哪些优点和缺点?

4.FIFO调度模式的基本原理、优点和缺点?

5.FAIR调度模式的优点和缺点?

6.CAPCACITY调度模式的优点和缺点?

7.列举你了解的序列化方法，并谈谈序列化有什么好处?

8.常见的数压缩方式，你们生产集群采用了什么压缩方式，提升了多少效率?

9.简要描述Spark写数据的流程?

10.Spark中Lineage的基本原理

11.使用shll和scala代码实现WordCount?

12.请列举你碰到的CPU密集型的应用场景，你有做哪些优化?

13.Spark RDD 和 MR2的区别

14.Spark读取hdfs上的文件，然后count有多少行的操作，你可以说说过程吗。那这个count是在内存中，还是磁盘中计算的呢?

15.spark和Mapreduce快？ 为什么快呢？ 快在哪里呢?

16.spark sql又为什么比hive快呢？

17.RDD的数据结构是怎么样的？

18.RDD算子里操作一个外部map比如往里面put数据。然后算子外再遍历map。会有什么问题吗。

19.hadoop的生态呢。说说你的认识。

20.jvm怎么调优的，介绍你的Spark JVM调优经验？

21.jvm结构？堆里面几个区？

22.怎么用spark做数据清洗

23.spark怎么整合hive？

24.spark读取 数据，是几个Partition呢？

25.hbase region多大会分区，spark读取hbase数据是如何划分partition的？

26.画图，画Spark的工作模式，部署分布架构图

27.画图，画图讲解spark工作流程。以及在集群上和各个角色的对应关系。

28.java自带有哪几种线程池。

29.画图，讲讲shuffle的过程。那你怎么在编程的时候注意避免这些性能问题?

30.BlockManager怎么管理硬盘和内存的?

【Spark面试2000题161-190】Spark Core面试篇06

原创 2017-06-13 梅峰谷 大数据梅峰谷

继续发放《Spark面试2000题》第六期的题目，上一期的参考答案获取方式，发送消息给公众号，消息内容：第五期答案。第五期试题：链接

1.kafka收集数据的原理?

2.讲讲列式存储的 parquet文件底层格式?

3.dataset和dataframe?

4 scala中trait特征和用法?

5.redis和memcache的区别?

6.列举Spark中常见的端口，分别有什么功能?

7.Spark master如何通过Zookeeper做HA?

8.Spark官网中，你常用哪几个模块?

9.你有见过哪些原因导致的数据倾斜，怎么解决?

10.简要描述宽依赖窄依赖以及各自的特点?

11.yarn的原理?

12.BlockManager怎么管理硬盘和内存的?

13.哪些算子操作涉及到shuffle1

14.看过源码？ 你熟悉哪几个部分的源码?

15.集群上 nodemanager和ResourceManager的数量关系?

16.Spark如何处理结构化数据，Spark如何处理非结构话数据?

17.Spark性能优化主要有哪些手段?

18.简要描述Spark分布式集群搭建的步骤?

19.对于Spark你觉得他对于现有大数据的现状的优势和劣势在哪里?

20.对于算法是否进行过自主的研究设计?

21.简要描述你了解的一些数据挖掘算法与内容

22. 什么时候join不发生shuffle?

23.spark shuffle的具体过程，你知道几种shuffle方式

24.spark 如何防止内存溢出？

25.简述hadoop实现join的及各种方式?

26 rdd转为dataFrame两种方式?

27.列举你熟悉的内存系统，各自的优缺点?

28.Spark 中Master 实现HA有哪些方式？

29 函数式编程特点?

30.Sort-based shuffle的缺陷?

面试|大数据相关试题-面试篇07

面试系列重新继续发布，下面这个是从网上搜来的，题目都是好题目，答案作为参考是可以的，作为学习素材，仅供大家参考。

1、简答说一下hadoop的map-reduce编程模型

首先map task会从本地文件系统读取数据，转换成key-value形式的键值对集合

使用的是hadoop内置的数据类型，比如longwritable、text等

将键值对集合输入mapper进行业务处理过程，将其转换成需要的key-value在输出

之后会进行一个partition分区操作，默认使用的是hashpartitioner，可以通过重写hashpartitioner的getpartition方法来自定义分区规则

之后会对key进行进行sort排序，grouping分组操作将相同key的value合并分组输出，在这里可以使用自定义的数据类型，重写WritableComparator的Comparator方法来自定义排序规则，重写RawComparator的compara方法来自定义分组规则

之后进行一个combiner归约操作，其实就是一个本地段的reduce预处理，以减小后面shufle和reducer的工作量

reduce task会通过网络将各个数据收集进行reduce处理，最后将数据保存或者显示，结束整个job

2、hadoop的TextInputFormat作用是什么，如何自定义实现

InputFormat会在map操作之前对数据进行两方面的预处理

1是getSplits，返回的是InputSplit数组，对数据进行split分片，每片交给map操作一次

2是getRecordReader，返回的是RecordReader对象，对每个split分片进行转换为key-value键值对格式传递给map

常用的InputFormat是TextInputFormat，使用的是LineRecordReader对每个分片进行键值对的转换，以行偏移量作为键，行内容作为值

自定义类继承InputFormat接口，重写createRecordReader和isSplitable方法

在createRecordReader中可以自定义分隔符

3、hadoop和spark的都是并行计算，那么他们有什么相同和区别

两者都是用mr模型来进行并行计算，hadoop的一个作业称为job，job里面分为map task和reduce task，每个task都是在自己的进程中运行的，当task结束时，进程也会结束

spark用户提交的任务成为application，一个application对应一个sparkcontext，app中存在多个job，每触发一次action操作就会产生一个job

这些job可以并行或串行执行，每个job中有多个stage，stage是shuffle过程中DAGSchaduler通过RDD之间的依赖关系划分job而来的，每个stage里面有多个task，组成taskset有TaskSchaduler分发到各个executor中执行，executor的生命周期是和app一样的，即使没有job运行也是存在的，所以task可以快速启动读取内存进行计算

hadoop的job只有map和reduce操作，表达能力比较欠缺而且在mr过程中会重复的读写hdfs，造成大量的io操作，多个job需要自己管理关系

spark的迭代计算都是在内存中进行的，API中提供了大量的RDD操作如join，groupby等，而且通过DAG图可以实现良好的容错

4、为什么要用flume导入hdfs，hdfs的构架是怎样的

flume可以实时的导入数据到hdfs中，当hdfs上的文件达到一个指定大小的时候会形成一个文件，或者超过指定时间的话也形成一个文件

文件都是存储在datanode上面的，namenode记录着datanode的元数据信息，而namenode的元数据信息是存在内存中的，所以当文件切片很小或者很多的时候会卡死

5、map-reduce程序运行的时候会有什么比较常见的问题

比如说作业中大部分都完成了，但是总有几个reduce一直在运行

这是因为这几个reduce中的处理的数据要远远大于其他的reduce，可能是因为对键值对任务划分的不均匀造成的数据倾斜

解决的方法可以在分区的时候重新定义分区规则对于value数据很多的key可以进行拆分、均匀打散等处理，或者是在map端的combiner中进行数据预处理的操作

6、简单说一下hadoop和spark的shuffle过程

hadoop：map端保存分片数据，通过网络收集到reduce端

spark：spark的shuffle是在DAGSchedular划分Stage的时候产生的，TaskSchedule要分发Stage到各个worker的executor，减少shuffle可以提高性能

7、Hive中存放是什么?

表（数据+元数据）。 存的是和hdfs的映射关系，hive是逻辑上的数据仓库，实际操作的都是hdfs上的文件，HQL就是用sql语法来写的mr程序。

8、Hive与关系型数据库的关系?

没有关系，hive是数据仓库，不能和数据库一样进行实时的CURD操作。

是一次写入多次读取的操作，可以看成是ETL工具。

9、Flume工作机制是什么?

核心概念是agent，里面包括source、chanel和sink三个组件。

source运行在日志收集节点进行日志采集，之后临时存储在chanel中，sink负责将chanel中的数据发送到目的地。

只有成功发送之后chanel中的数据才会被删除。

首先书写flume配置文件，定义agent、source、chanel和sink然后将其组装，执行flume-ng命令。

10、Sqoop工作原理是什么?

hadoop生态圈上的数据传输工具。

可以将关系型数据库的数据导入非结构化的hdfs、hive或者bbase中，也可以将hdfs中的数据导出到关系型数据库或者文本文件中。

使用的是mr程序来执行任务，使用jdbc和关系型数据库进行交互。

import原理：通过指定的分隔符进行数据切分，将分片传入各个map中，在map任务中在每行数据进行写入处理没有reduce。

export原理：根据要操作的表名生成一个java类，并读取其元数据信息和分隔符对非结构化的数据进行匹配，多个map作业同时执行写入关系型数据库

11、Hbase行健列族的概念，物理模型，表的设计原则?

行健：是hbase表自带的，每个行健对应一条数据。

列族：是创建表时指定的，为列的集合，每个列族作为一个文件单独存储，存储的数据都是字节数组，其中的数据可以有很多，通过时间戳来区分。

物理模型：整个hbase表会拆分为多个region，每个region记录着行健的起始点保存在不同的节点上，查询时就是对各个节点的并行查询，当region很大时使用.META表存储各个region的起始点，-ROOT又可以存储.META的起始点。

rowkey的设计原则：各个列簇数据平衡，长度原则、相邻原则，创建表的时候设置表放入regionserver缓存中，避免自动增长和时间，使用字节数组代替string，最大长度64kb，最好16字节以内，按天分表，两个字节散列，四个字节存储时分毫秒。

列族的设计原则：尽可能少（按照列族进行存储，按照region进行读取，不必要的io操作），经常和不经常使用的两类数据放入不同列族中，列族名字尽可能短。

12、Spark Streaming和Storm有何区别?

一个实时毫秒一个准实时亚秒，不过storm的吞吐率比较低。

13、mllib支持的算法?

大体分为四大类，分类、聚类、回归、协同过滤。

14、简答说一下hadoop的map-reduce编程模型?

首先map task会从本地文件系统读取数据，转换成key-value形式的键值对集合。

将键值对集合输入mapper进行业务处理过程，将其转换成需要的key-value在输出。

之后会进行一个partition分区操作，默认使用的是hashpartitioner，可以通过重写hashpartitioner的getpartition方法来自定义分区规则。

之后会对key进行进行sort排序，grouping分组操作将相同key的value合并分组输出。

在这里可以使用自定义的数据类型，重写WritableComparator的Comparator方法来自定义排序规则，重写RawComparator的compara方法来自定义分组规则。

之后进行一个combiner归约操作，其实就是一个本地段的reduce预处理，以减小后面shufle和reducer的工作量。

reduce task会通过网络将各个数据收集进行reduce处理，最后将数据保存或者显示，结

束整个job。

15、Hadoop平台集群配置、环境变量设置?

zookeeper：修改zoo.cfg文件，配置dataDir，和各个zk节点的server地址端口，tickTime心跳时间默认是2000ms，其他超时的时间都是以这个为基础的整数倍，之后再dataDir对应目录下写入myid文件和zoo.cfg中的server相对应。

hadoop：修改

hadoop-env.sh配置java环境变量

core-site.xml配置zk地址，临时目录等

hdfs-site.xml配置nn信息，rpc和http通信地址，nn自动切换、zk连接超时时间等

yarn-site.xml配置resourcemanager地址

mapred-site.xml配置使用yarn

slaves配置节点信息

格式化nn和zk。

hbase：修改

hbase-env.sh配置java环境变量和是否使用自带的zk

hbase-site.xml配置hdfs上数据存放路径，zk地址和通讯超时时间、master节点

regionservers配置各个region节点

zoo.cfg拷贝到conf目录下

spark：

安装Scala

修改spark-env.sh配置环境变量和master和worker节点配置信息

环境变量的设置：直接在/etc/profile中配置安装的路径即可，或者在当前用户的宿主目录下，配置在.bashrc文件中，该文件不用source重新打开shell窗口即可，配置在.bash_profile的话只对当前用户有效。

16、Hadoop性能调优?

调优可以通过系统配置、程序编写和作业调度算法来进行。

hdfs的block.size可以调到128/256（网络很好的情况下，默认为64）

调优的大头：mapred.map.tasks、mapred.reduce.tasks设置mr任务数（默认都是1）

mapred.tasktracker.map.tasks.maximum每台机器上的最大map任务数

mapred.tasktracker.reduce.tasks.maximum每台机器上的最大reduce任务数

mapred.reduce.slowstart.completed.maps配置reduce任务在map任务完成到百分之几的时候开始进入

这个几个参数要看实际节点的情况进行配置，reduce任务是在33%的时候完成copy，要在这之前完成map任务，（map可以提前完成）

mapred.compress.map.output,mapred.output.compress配置压缩项，消耗cpu提升网络和磁盘io

合理利用combiner

注意重用writable对象

17、Hadoop高并发?

首先肯定要保证集群的高可靠性，在高并发的情况下不会挂掉，支撑不住可以通过横向扩展。

datanode挂掉了使用hadoop脚本重新启动。

18、hadoop的TextInputFormat作用是什么，如何自定义实现？

InputFormat会在map操作之前对数据进行两方面的预处理。

1是getSplits，返回的是InputSplit数组，对数据进行split分片，每片交给map操作一次 。

2是getRecordReader，返回的是RecordReader对象，对每个split分片进行转换为key-value键值对格式传递给map。

常用的InputFormat是TextInputFormat，使用的是LineRecordReader对每个分片进行键值对的转换，以行偏移量作为键，行内容作为值。

自定义类继承InputFormat接口，重写createRecordReader和isSplitable方法 。

在createRecordReader中可以自定义分隔符。

19、hadoop和spark的都是并行计算，那么他们有什么相同和区别？

两者都是用mr模型来进行并行计算，hadoop的一个作业称为job，job里面分为map task和reduce task，每个task都是在自己的进程中运行的，当task结束时，进程也会结束。

spark用户提交的任务成为application，一个application对应一个sparkcontext，app中存在多个job，每触发一次action操作就会产生一个job。

这些job可以并行或串行执行，每个job中有多个stage，stage是shuffle过程中DAGSchaduler通过RDD之间的依赖关系划分job而来的，每个stage里面有多个task，组成taskset有TaskSchaduler分发到各个executor中执行，executor的生命周期是和app一样的，即使没有job运行也是存在的，所以task可以快速启动读取内存进行计算。

hadoop的job只有map和reduce操作，表达能力比较欠缺而且在mr过程中会重复的读写hdfs，造成大量的io操作，多个job需要自己管理关系。

spark的迭代计算都是在内存中进行的，API中提供了大量的RDD操作如join，groupby等，而且通过DAG图可以实现良好的容错。

20、为什么要用flume导入hdfs，hdfs的构架是怎样的？

flume可以实时的导入数据到hdfs中，当hdfs上的文件达到一个指定大小的时候会形成一个文件，或者超过指定时间的话也形成一个文件。

文件都是存储在datanode上面的，namenode记录着datanode的元数据信息，而namenode的元数据信息是存在内存中的，所以当文件切片很小或者很多的时候会卡死。

21、map-reduce程序运行的时候会有什么比较常见的问题？

比如说作业中大部分都完成了，但是总有几个reduce一直在运行。

这是因为这几个reduce中的处理的数据要远远大于其他的reduce，可能是因为对键值对任务划分的不均匀造成的数据倾斜。

解决的方法可以在分区的时候重新定义分区规则对于value数据很多的key可以进行拆分、均匀打散等处理，或者是在map端的combiner中进行数据预处理的操作。

16、简单说一下hadoop和spark的shuffle过程？

hadoop：map端保存分片数据，通过网络收集到reduce端。

spark：spark的shuffle是在DAGSchedular划分Stage的时候产生的，TaskSchedule要分发Stage到各个worker的executor。

减少shuffle可以提高性能。

22、RDD机制?

rdd分布式弹性数据集，简单的理解成一种数据结构，是spark框架上的通用货币。

所有算子都是基于rdd来执行的，不同的场景会有不同的rdd实现类，但是都可以进行互相转换。

rdd执行过程中会形成dag图，然后形成lineage保证容错性等。

从物理的角度来看rdd存储的是block和node之间的映射。

18、spark有哪些组件?

（1）master：管理集群和节点，不参与计算。

（2）worker：计算节点，进程本身不参与计算，和master汇报。

（3）Driver：运行程序的main方法，创建spark context对象。

（4）spark context：控制整个application的生命周期，包括dagsheduler和task scheduler等组件。

（5）client：用户提交程序的入口。

23、spark工作机制?

用户在client端提交作业后，会由Driver运行main方法并创建spark context上下文。

执行add算子，形成dag图输入dagscheduler，按照add之间的依赖关系划分stage输入task scheduler。

task scheduler会将stage划分为task set分发到各个节点的executor中执行。

24、spark的优化怎么做?

通过spark-env文件、程序中sparkconf和set property设置。

（1）计算量大，形成的lineage过大应该给已经缓存了的rdd添加checkpoint，以减少容错带来的开销。

（2）小分区合并，过小的分区造成过多的切换任务开销，使用repartition。

25、kafka工作原理?

producer向broker发送事件，consumer从broker消费事件。

事件由topic区分开，每个consumer都会属于一个group。

相同group中的consumer不能重复消费事件，而同一事件将会发送给每个不同group的consumer。

26、ALS算法原理?

答：对于user-product-rating数据，als会建立一个稀疏的评分矩阵，其目的就是通过一定的规则填满这个稀疏矩阵。

als会对稀疏矩阵进行分解，分为用户-特征值，产品-特征值，一个用户对一个产品的评分可以由这两个矩阵相乘得到。

通过固定一个未知的特征值，计算另外一个特征值，然后交替反复进行最小二乘法，直至差平方和最小，即可得想要的矩阵。

27、kmeans算法原理?

随机初始化中心点范围，计算各个类别的平均值得到新的中心点。

重新计算各个点到中心值的距离划分，再次计算平均值得到新的中心点，直至各个类别数据平均值无变化。

28、canopy算法原理?

根据两个阈值来划分数据，以随机的一个数据点作为canopy中心。

计算其他数据点到其的距离，划入t1、t2中，划入t2的从数据集中删除，划入t1的其他数据点继续计算，直至数据集中无数据。

29、朴素贝叶斯分类算法原理?

对于待分类的数据和分类项，根据待分类数据的各个特征属性，出现在各个分类项中的概率判断该数据是属于哪个类别的。

30、关联规则挖掘算法apriori原理?

一个频繁项集的子集也是频繁项集，针对数据得出每个产品的支持数列表，过滤支持数小于预设值的项，对剩下的项进行全排列，重新计算支持数，再次过滤，重复至全排列结束，可得到频繁项和对应的支持数。

面试的时候这样的题目给你，你会不会说，fuck~，怎么这么变态，经常写代码的人，看到这个应该会很面熟。究竟什么样的题目这么变态，解答如下，仅供参考学习

1、Operation category READ is not supported in state standby是什么原因导致的？org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.ipc.StandbyException): Operation category READ is not supported in state standby 答：此时请登录Hadoop的管理界面查看运行节点是否处于standby

如登录地址是：http://xx.xx.xx.xx:50070/dfshealth.html#tab-overview

如果是，则不可在处于StandBy机器运行spark计算，因为该台机器为备分机器

2、不配置spark.deploy.recoveryMode选项为ZOOKEEPER，会有什么不好的地方

如果不设置spark.deploy.recoveryMode的话，那么集群的所有运行数据在Master重启是都会丢失，可参考BlackHolePersistenceEngine的实现。

3、多Master如何配置

因为涉及到多个Master，所以对于应用程序的提交就有了一点变化，因为应用程序需要知道当前的Master的IP地址和端口。这种HA方案处理这种情况很简单，只需要在SparkContext指向一个Master列表就可以了，如spark://host1:port1,host2:port2,host3:port3，应用程序会轮询列表。

4、No Space Left on the device（Shuffle临时文件过多）

由于Spark在计算的时候会将中间结果存储到/tmp目录，而目前linux又都支持tmpfs，其实就是将/tmp目录挂载到内存当中。

那么这里就存在一个问题，中间结果过多导致/tmp目录写满而出现如下错误

No Space Left on the device

解决办法

第一种：修改配置文件spark-env.sh,把临时文件引入到一个自定义的目录中去即可

export SPARK_LOCAL_DIRS=/home/utoken/datadir/spark/tmp

第二种：偷懒方式，针对tmp目录不启用tmpfs,直接修改/etc/fstab

5、java.lang.OutOfMemory, unable to create new native thread

Caused by: java.lang.OutOfMemoryError: unable to create new native thread

at java.lang.Thread.start0(Native Method)

at java.lang.Thread.start(Thread.java:640)

上面这段错误提示的本质是Linux操作系统无法创建更多进程，导致出错，并不是系统的内存不足。因此要解决这个问题需要修改Linux允许创建更多的进程，就需要修改Linux最大进程数。

[utoken@nn1 ~]$ulimit -a

临时修改允许打开的最大进程数

[utoken@nn1 ~]$ulimit -u 65535

临时修改允许打开的文件句柄

[utoken@nn1 ~]$ulimit -n 65535

永久修改Linux最大进程数量

[utoken@nn1 ~]$ vim /etc/security/limits.d/90-nproc.conf

soft nproc 60000

root soft nproc unlimited

永久修改用户打开文件的最大句柄数，该值默认1024，一般都会不够，常见错误就是not open file

[utoken@nn1 ~]$ vim /etc/security/limits.conf

bdata soft nofile 65536

bdata hard nofile 65536

6、Worker节点中的work目录占用许多磁盘空间

目录地址：/home/utoken/software/spark-1.3.0-bin-hadoop2.4/work

这些是Driver上传到worker的文件，需要定时做手工清理，否则会占用许多磁盘空间

7、spark-shell提交Spark Application如何解决依赖库

spark-shell的话，利用–driver-class-path选项来指定所依赖的jar文件，注意的是–driver-class-path后如果需要跟着多个jar文件的话，jar文件之间使用冒号(:)来分割。

8、Spark在发布应用的时候，出现连接不上master问题，如下

15/11/19 11:35:50 INFO AppClient$ClientEndpoint: Connecting to master spark://s1:7077…

15/11/19 11:35:50 WARN ReliableDeliverySupervisor: Association with remote system [akka.tcp://sparkMaster@s1:7077] has failed, address is now gated for [5000] ms. Reason: [Disassociated]

解决方式

检查所有机器时间是否一致、hosts是否都配置了映射、客户端和服务器端的Scala版本是否一致、Scala版本是否和Spark兼容

检查是否兼容问题请参考官方网站介绍：

9、开发spark应用程序（和Flume-NG结合时）发布应用时可能出现org.jboss.netty.channel.ChannelException: Failed to bind to: /192.168.10.156:18800

15/11/27 10:33:44 ERROR ReceiverSupervisorImpl: Stopped receiver with error: org.jboss.netty.channel.ChannelException: Failed to bind to: /192.168.10.156:18800

15/11/27 10:33:44 ERROR Executor: Exception in task 0.0 in stage 2.0 (TID 70)

org.jboss.netty.channel.ChannelException: Failed to bind to: /192.168.10.156:18800

at org.jboss.netty.bootstrap.ServerBootstrap.bind(ServerBootstrap.java:272)

Caused by: java.net.BindException: Cannot assign requested address

由于spark通过Master发布的时候，会自动选取发送到某一台的worker节点上，所以这里绑定端口的时候，需要选择相应的worker服务器，但是由于我们无法事先了解到，spark发布到哪一台服务器的，所以这里启动报错，是因为在 192.168.10.156:18800的机器上面没有启动Driver程序，而是发布到了其他服务器去启动了，所以无法监听到该机器出现问题，所以我们需要设置spark分发包时，发布到所有worker节点机器，或者发布后，我们去寻找发布到了哪一台机器，重新修改绑定IP，重新发布，有一定几率发布成功。详情可见《印象笔记-战5渣系列——Spark Streaming启动问题 - 推酷》

10、spark-shell 找不到hadoop so问题解决

[main] WARN org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop library for your platform… using builtin-java classes where applicable

在Spark的conf目录下，修改spark-env.sh文件，加入LD_LIBRARY_PATH环境变量，值为HADOOP的native库路径即可.

11、ERROR XSDB6: Another instance of Derby may have already booted the database /home/bdata/data/metastore_db.

在使用Hive on Spark模式操作hive里面的数据时，报以上错误，原因是因为HIVE采用了derby这个内嵌数据库作为数据库，它不支持多用户同时访问,解决办法就是把derby数据库换成mysql数据库即可

变更方式

12、java.lang.IllegalArgumentException: java.net.UnknownHostException: dfscluster

解决办法：

找不到hdfs集群名字dfscluster,这个文件在HADOOP的etc/hadoop下面，有个文件hdfs-site.xml，复制到Spark的conf下，重启即可

如：执行脚本，分发到所有的Spark集群机器中，

[bdata@bdata4 hadoop]foriin34,35,36,37,38;doscphdfs−site.xml192.168.10.i:/u01/spark-1.5.1/conf/ ; done

13、Exception in thread “main” java.lang.Exception: When running with master ‘yarn-client’ either HADOOP_CONF_DIR or YARN_CONF_DIR must be set in the environment.

问题：在执行yarn集群或者客户端时，报以上错误，

[bdata@bdata4 bin]$ ./spark-sql –master yarn-client

Exception in thread “main” java.lang.Exception: When running with master ‘yarn-client’ either HADOOP_CONF_DIR or YARN_CONF_DIR must be set in the environment.

解决办法

根据提示，配置HADOOP_CONF_DIR or YARN_CONF_DIR的环境变量即可

export HADOOP_HOME=/u01/hadoop-2.6.1

export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop

PATH=PATH:HOME/.local/bin:HOME/bin:SQOOP_HOME/bin:HIVEHOME/bin:HADOOP_HOME/bin

14、Job aborted due to stage failure: Task 3 in stage 0.0 failed 4 times, most recent failure: Lost task 3.3 in

[Stage 0:> (0 + 4) / 42]2016-01-15 11:28:16,512 [org.apache.spark.scheduler.TaskSchedulerImpl]-[ERROR] Lost executor 0 on 192.168.10.38: remote Rpc client disassociated

[Stage 0:> (0 + 4) / 42]2016-01-15 11:28:23,188 [org.apache.spark.scheduler.TaskSchedulerImpl]-[ERROR] Lost executor 1 on 192.168.10.38: remote Rpc client disassociated

[Stage 0:> (0 + 4) / 42]2016-01-15 11:28:29,203 [org.apache.spark.scheduler.TaskSchedulerImpl]-[ERROR] Lost executor 2 on 192.168.10.38: remote Rpc client disassociated

[Stage 0:> (0 + 4) / 42]2016-01-15 11:28:36,319 [org.apache.spark.scheduler.TaskSchedulerImpl]-[ERROR] Lost executor 3 on 192.168.10.38: remote Rpc client disassociated

2016-01-15 11:28:36,321 [org.apache.spark.scheduler.TaskSetManager]-[ERROR] Task 3 in stage 0.0 failed 4 times; aborting job

Exception in thread “main” org.apache.spark.SparkException : Job aborted due to stage failure: Task 3 in stage 0.0 failed 4 times, most recent failure: Lost task 3.3 in stage 0.0 (TID 14, 192.168.10.38): ExecutorLostFailure (executor 3 lost)

Driver stacktrace:

at org.apache.spark.scheduler.DAGScheduler.org$apache$spark$scheduler$DAGScheduler$$failJobAndIndependentStages(DAGScheduler.scala:1283)

解决方案

这里遇到的问题主要是因为数据源数据量过大，而机器的内存无法满足需求，导致长时间执行超时断开的情况，数据无法有效进行交互计算，因此有必要增加内存

15、长时间等待无反应，并且看到服务器上面的web界面有内存和核心数，但是没有分配，如下图

[Stage 0:> (0 + 0) / 42]

或者日志信息显示：

16/01/15 14:18:56 WARN TaskSchedulerImpl: Initial job has not accepted any resources; check your cluster UI to ensure that workers are registered and have sufficient resources

解决方案

出现上面的问题主要原因是因为我们通过参数spark.executor.memory设置的内存过大，已经超过了实际机器拥有的内存，故无法执行，需要等待机器拥有足够的内存后，才能执行任务，可以减少任务执行内存，设置小一些即可

16、内存不足或数据倾斜导致Executor Lost（spark-submit提交）

TaskSetManager: Lost task 1.0 in stage 6.0 (TID 100, 192.168.10.37): java.lang.OutOfMemoryError: Java heap space

16/01/15 14:29:51 INFO BlockManagerInfo: Added broadcast_8_piece0 in memory on 192.168.10.37:57139 (size: 42.0 KB, free: 24.2 MB)

16/01/15 14:29:53 INFO BlockManagerInfo: Added broadcast_8_piece0 in memory on 192.168.10.38:53816 (size: 42.0 KB, free: 24.2 MB)

16/01/15 14:29:55 INFO TaskSetManager: Starting task 3.0 in stage 6.0 (TID 102, 192.168.10.37, ANY, 2152 bytes)

16/01/15 14:29:55 WARN TaskSetManager: Lost task 1.0 in stage 6.0 (TID 100, 192.168.10.37): java.lang.OutOfMemoryError: Java heap space

at java.io.BufferedOutputStream.(BufferedOutputStream.java:76)

at java.io.BufferedOutputStream.(BufferedOutputStream.java:59)

…….

org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stage 6.0 failed 4 times, most recent failure: Lost task 0.3 in stage 6.0 (TID 142, 192.168.10.36): ExecutorLostFailure (executor 4 lost)

……

WARN TaskSetManager: Lost task 4.1 in stage 6.0 (TID 137, 192.168.10.38): java.lang.OutOfMemoryError: GC overhead limit exceeded

解决办法：

由于我们在执行Spark任务是，读取所需要的原数据，数据量太大，导致在Worker上面分配的任务执行数据时所需要的内存不够，直接导致内存溢出了，所以我们有必要增加Worker上面的内存来满足程序运行需要。

在Spark Streaming或者其他spark任务中，会遇到在Spark中常见的问题，典型如Executor Lost 相关的问题(shuffle fetch 失败，Task失败重试等)。这就意味着发生了内存不足或者数据倾斜的问题。这个目前需要考虑如下几个点以获得解决方案：

A、相同资源下，增加partition数可以减少内存问题。 原因如下：通过增加partition数，每个task要处理的数据少了，同一时间内，所有正在运行的task要处理的数量少了很多，所有Executor占用的内存也变小了。这可以缓解数据倾斜以及内存不足的压力。

B、关注shuffle read 阶段的并行数。例如reduce,group 之类的函数，其实他们都有第二个参数，并行度(partition数)，只是大家一般都不设置。不过出了问题再设置一下，也不错。

C、给一个Executor 核数设置的太多，也就意味着同一时刻，在该Executor 的内存压力会更大，GC也会更频繁。我一般会控制在3个左右。然后通过提高Executor数量来保持资源的总量不变。

17、 Spark Streaming 和kafka整合后读取消息报错：OffsetOutOfRangeException

解决方案：如果和kafka消息中间件结合使用，请检查消息体是否大于默认设置1m，如果大于，则需要设置fetch.message.max.bytes=1m，这里需要把值设置大些

18、java.io.IOException : Could not locate executable null\bin\winutils.exe in the Hadoop binaries.（spark sql on hive 任务引发HiveContext NullPointerException）

解决办法

在开发hive和Spark整合的时候，如果是Windows系统，并且没有配置HADOOP_HOME的环境变量，那么可能找不到winutils.exe这个工具，由于使用hive时，对该命令有依赖，所以不要忽视该错误，否则将无法创建HiveContext，一直报Exception in thread “main” java.lang.RuntimeException: java.lang.NullPointerException

因此，解决该办法有两个方式

A、把任务打包成jar，上传到服务器上面，服务器是配置过HADOOP_HOME环境变量的，并且不需要依赖winutils,所以只需要通过spark-submit方式提交即可，如：

[bdata@bdata4 app]$ spark-submit –class com.pride.hive.HiveOnSparkTest –master spark://bdata4:7077 spark-simple-1.0.jar

B、解决winutils.exe命令不可用问题，配置Windows上面HADOOP_HOME的环境变量，或者在程序最开始的地方设置HADOOP_HOME的属性配置,这里需要注意，由于最新版本已经没有winutils这些exe命令了，我们需要在其他地方下载该命令放入HADOOP的bin目录下，当然也可以直接配置下载项目的环境变量，变量名一定要是HADOOP_HOME才行

下载地址：https://github.com/srccodes/hadoop-common-2.2.0-bin/archive/master.zip （记得翻墙哦）

任何项目都生效，需要配置Windows的环境变量，如果只在程序中生效可在程序中配置即可，如

//用于解决Windows下找不到winutils.exe命令

System. setProperty(“hadoop.home.dir”, “E:\Software\hadoop-common-2.2.0-bin”);

19、The root scratch dir: /tmp/hive on HDFS should be writable. Current permissions are: rwx——

解决办法

1、程序中设置环境变量：System.setProperty(“HADOOP_USER_NAME”, “bdata”)

2、修改HDFS的目录权限

Update the permission of your /tmp/hive HDFS directory using the following command

hadoop dfs -chmod 777 /tmp/hive

此问题暂未解决，估计是17点解决winutils有问题，建议最好把任务程序发布到服务器上面解决

20、Exception in thread “main” org.apache.hadoop.security.AccessControlException : Permission denied: user=Administrator, access=WRITE, inode=”/data”:bdata:supergroup:drwxr-xr-x

解决办法

1、在系统的环境变量或java JVM变量里面添加HADOOP_USER_NAME，如程序中添加System.setProperty(“HADOOP_USER_NAME”, “bdata”);，这里的值就是以后会运行HADOOP上的Linux的用户名，如果是eclipse，则修改完重启eclipse，不然可能不生效

2、hdfs dfs -chmod 777 修改相应权限地址

21、运行Spark-SQL报错：org.apache.spark.sql.AnalysisException: unresolved operator ‘Project

解决办法：

在Spark-sql和hive结合时或者单独Spark-sql，运行某些sql语句时，偶尔出现上面错误，那么我们可以检查一下sql的问题，这里遇到的问题是嵌套语句太多，导致spark无法解析，所以需要修改sql或者改用其他方式处理；特别注意该语句可能在hive里面没有错误，spark才会出现的一种错误。

22.在$SPARK_HOME/conf/spark-env.sh中设置这些变量好像也只是在terminal中的shell环境中才有效JAVA_HOME is not set Exception: Java gateway process exited before sending the driver its port number

但是在命令行中是有的

pipi@pipicmp:~$ echo $JAVA_HOME

/home/pipi/ENV/jdk

解决方法1：在py代码中加入JAVA_HOME到os中

JAVA_HOME = /home/pipi/ENV/jdk

os.environ['JAVA_HOME'] = conf.get(SECTION, 'JAVA_HOME')

解决方法2：或者在hadoop中配置好JAVA_HOME

hadoop中配置JAVA_HOME

23.ValueError: Cannot run multiple SparkContexts at once

Welcome to

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.0.1
      /_/
```

Using Python version 3.5.2 (default, Sep 10 2016 08:21:44)

SparkSession available as 'spark'.

ValueError: Cannot run multiple SparkContexts at once; existing SparkContext(app=pyspark-shell, master=local) created by <module> at <frozen importlib._bootstrap>:222

原因是：from pyspark.shell import sqlContext

引入的包中也定义了一个sc = spark.sparkContext导致和本代码中定义重复了。

24.spark输出太多warning messages

调试log时候发现问题解决了

在简略Spark输出设置时[Spark安装和配置]修改过$SPARK_HOME/conf/log4j.properties.template文件只输出WARN信息，就算改成了ERROR，信息也还是会自动修改成WARN输出出来，

不过多了一条提示：

Setting default log level to "WARN". To adjust logging level use sc.setLogLevel(newLevel).

就在这时发现了一个解决方案：

根据提示在代码中加入sc.setLogLevel('ERROR')就可以解决了！

25.org.apache.spark.shuffle.FetchFailedException，一般发生在有大量shuffle操作的时候,task不断的failed,然后又重执行，一直循环下去，非常的耗时

一般遇到这种问题提高executor内存即可,同时增加每个executor的cpu,这样不会减少task并行度。

26.Executor&Task Lost因为网络或者gc的原因,worker或executor没有接收到executor或task的心跳反馈WARN TaskSetManager: Lost task 1.0 in stage 0.0 (TID 1, aa.local): ExecutorLostFailure (executor lost)

提高 spark.network.timeout 的值，根据情况改成300(5min)或更高。

默认为 120(120s),配置所有网络传输的延时，如果没有主动设置以下参数，默认覆盖其属性

spark.core.connection.ack.wait.timeout

spark.akka.timeout

spark.storage.blockManagerSlaveTimeoutMs

spark.shuffle.io.connectionTimeout

spark.rpc.askTimeout or spark.rpc.lookupTimeout

27. Master挂掉,standby重启也失效，如Master默认使用512M内存，当集群中运行的任务特别多时，就会挂掉，原因是master会读取每个task的event log日志去生成spark ui，内存不足自然会OOM，可以在master的运行日志中看到，通过HA启动的master自然也会因为这个原因失败。

1）.增加Master的内存占用，在Master节点spark-env.sh 中设置：

export SPARK_DAEMON_MEMORY 10g # 根据你的实际情况

2）.减少保存在Master内存中的作业信息

spark.ui.retainedJobs 500 # 默认都是1000 spark.ui.retainedStages 500

28. worker挂掉或假死有时候我们还会在web ui中看到worker节点消失或处于dead状态，在该节点运行的任务则会报各种 lost worker 的错误，引发原因和上述大体相同，worker内存中保存了大量的ui信息导致gc时失去和master之间的心跳。

解决

1）增加Master的内存占用，在Worker节点spark-env.sh 中设置：

export SPARK_DAEMON_MEMORY 2g # 根据你的实际情况

2）减少保存在Worker内存中的Driver,Executor信息

spark.worker.ui.retainedExecutors 200 # 默认都是1000 spark.worker.ui.retainedDrivers 200

29.报错：ERROR storage.DiskBlockObjectWriter: Uncaught exception while reverting partial writes to file /hadoop/application_1415632483774_448143/spark-local-20141127115224-9ca8/04/shuffle_1_1562_27

java.io.FileNotFoundException: /hadoop/application_1415632483774_448143/spark-local-20141127115224-9ca8/04/shuffle_1_1562_27 (No such file or directory)

表面上看是因为shuffle没有地方写了，如果后面的stack是local space 的问题，那么清一下磁盘就好了。上面这种问题，是因为一个excutor给分配的内存不够，此时，减少excutor-core的数量，加大excutor-memory的值应该就没有问题。

30.报错：ERROR executor.CoarseGrainedExecutorBackend: Driver Disassociated [akka.tcp://sparkExecutor@pc-jfqdfx31:48586] -> [akka.tcp://sparkDriver@pc-jfqdfx30:41656] disassociated! Shutting down.

15/07/23 10:50:56 ERROR executor.CoarseGrainedExecutorBackend: RECEIVED SIGNAL 15: SIGTERM

这个错误比较隐晦，从信息上看来不知道是什么问题，但是归根结底还是内存的问题，有两个方法可以解决这个错误，一是，如上面所说，加大excutor-memory的值，减少executor-cores的数量，问题可以解决。二是，加大executor.overhead的值，但是这样其实并没有解决掉根本的问题。所以如果集群的资源是支持的话，就用1的办法吧。

另外，这个错误也出现在partitionBy(new HashPartition(partiton-num))时，如果partiton-num太大或者太小的时候会报这种错误，说白了也是内存的原因，不过这个时候增加内存和overhead没有什么用，得去调整这个partiton-num的值。

为了早日成为上面BAT三个人中的一员，或者直接成为他们，坚持学习完后刷题把，最新整理和收集的题库。

1.给定a、b两个文件，各存放50亿个url，每个url各占64字节，内存限制是4G，让你找出a、b文件共同的url?

方案1：可以估计每个文件安的大小为5G×64=320G，远远大于内存限制的4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

遍历文件a，对每个url求取hash(url)%1000，然后根据所取得的值将url分别存储到1000个小文件(记为a0,a1,…,a999)中。这样每个小文件的大约为300M。

遍历文件b，采取和a相同的方式将url分别存储到1000小文件(记为b0,b1,…,b999)。这样处理后，所有可能相同的url都在对应的小文件(a0vsb0,a1vsb1,…,a999vsb999)中，不对应的小文件不可能有相同的url。然后我们只要求出1000对小文件中相同的url即可。

求每对小文件中相同的url时，可以把其中一个小文件的url存储到hash_set中。然后遍历另一个小文件的每个url，看其是否在刚才构建的hash_set中，如果是，那么就是共同的url，存到文件里面就可以了。

方案2：如果允许有一定的错误率，可以使用Bloomfilter，4G内存大概可以表示340亿bit。将其中一个文件中的url使用Bloomfilter映射为这340亿bit，然后挨个读取另外一个文件的url，检查是否与Bloomfilter，如果是，那么该url应该是共同的url(注意会有一定的错误率)。

Bloomfilter日后会在本BLOG内详细阐述。补充：另外一种思路，是将url通过算法转为数字类型，转换后的连接就是比较数值是否相等了。

2.有一个1G大小的一个文件，里面每一行是一个词，词的大小不超过16字节，内存限制大小是1M，要求返回频数最高的100个词。

Step1：顺序读文件中，对于每个词x，取hash(x)%5000，然后按照该值存到5000个小文件(记为f0,f1,...,f4999)中，这样每个文件大概是200k左右，如果其中的有的文件超过了1M大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过1M;

Step2：对每个小文件，统计每个文件中出现的词以及相应的频率(可以采用trie树/hash_map等)，并取出出现频率最大的100个词(可以用含100个结点的最小堆)，并把100词及相应的频率存入文件，这样又得到了5000个文件;

Step3：把这5000个文件进行归并(类似与归并排序);

草图如下(分割大问题，求解小问题，归并)：

3.现有海量日志数据保存在一个超级大的文件中，该文件无法直接读入内存，要求从中提取某天出访问百度次数最多的那个IP。

Step1：从这一天的日志数据中把访问百度的IP取出来，逐个写入到一个大文件中;

Step2：注意到IP是32位的，最多有2^32个IP。同样可以采用映射的方法，比如模1000，把整个大文件映射为1000个小文件;

Step3：找出每个小文中出现频率最大的IP(可以采用hash_map进行频率统计，然后再找出频率最大的几个)及相应的频率;

Step4：在这1000个最大的IP中，找出那个频率最大的IP，即为所求。

草图如下：

4.LVS和HAProxy相比，它的缺点是什么?

之前，的确是用LVS进行过MySQL集群的负载均衡，对HAProxy也有过了解，但是将这两者放在眼前进行比较，还真没试着了解过。面试中出现了这么一题，面试官给予的答案是LVS的配置相当繁琐，后来查找了相关资料，对这两种负载均衡方案有了更进一步的了解。LVS的负载均衡性能之强悍已经达到硬件负载均衡的F5的百分之60了，而HAproxy的负载均衡和Nginx负载均衡，均为硬件负载均衡的百分之十左右。由此可见，配置复杂，相应的效果也是显而易见的。在查找资料的过程中，试着将LVS的10种调度算法了解了一下，看似数量挺多的10种算法其实在不同的算法之间，有些只是有着一些细微的差别。在这10种调度算法中，静态调度算法有四种，动态调度算法有6种。

静态调度算法：

①RR轮询调度算法

这种调度算法不考虑服务器的状态，所以是无状态的，同时也不考虑每个服务器的性能，比如我有1-N台服务器，来N个请求了，第一个请求给第一台，第二个请求给第二台，，，第N个请求给第N台服务器，就酱紫。

②加权轮询

这种调度算法是考虑到服务器的性能的，你可以根据不同服务器的性能，加上权重进行分配相应的请求。

③基于目的地址的hash散列

这种调度算法和基于源地址的hash散列异曲同工，都是为了维持一个session，基于目的地址的hash散列，将记住同一请求的目的地址，将这类请求发往同一台目的服务器。简而言之，就是发往这个目的地址的请求都发往同一台服务器。而基于源地址的hash散列，就是来自同一源地址的请求都发往同一台服务器。

④基于源地址的hash散列

上述已讲，不再赘述。

动态调度

①最少连接调度算法

这种调度算法会记录响应请求的服务器上所建立的连接数，每接收到一个请求会相应的将该服务器的所建立连接数加1，同时将新来的请求分配到当前连接数最少的那台机器上。

②加权最少连接调度算法

这种调度算法在最少连接调度算法的基础上考虑到服务器的性能。当然，做这样子的考虑是有其合理性存在的，如果是同一规格的服务器，那么建立的连接数越多，必然越增加其负载，那

么仅仅根据最少连接数的调度算法，必然可以实现合理的负载均衡。但如果，服务器的性能不一样呢?比如我有一台服务器，最多只能处理10个连接，现在建立了3个，还有一台服务器最多能处理1000条连接，现在建立了5个，如果单纯地按照上述的最少连接调度算法，妥妥的前者嘛，但前者已经建立了百分之三十的连接了，而后者连百分之一的连接还没有建立，试问，这合理吗?显然不合理。所以加上权重，才算合理。相应的公式也相当简单：active*256/weight。

③最短期望调度算法

这种算法，是避免出现上述加权最少连接调度算法中的一种特殊情况，导致即使加上权重，调度器也无差别对待了，举个栗子：

假设有三台服务器ABC，其当前所建立的连接数相应地为1,2,3，而权重也是1,2,3。那么如果按照加权最少连接调度算法的话，算出来是这样子的：

A:1256/1=256

B:2256/2=256

C:3256/3=256

我们会发现，即便加上权重，A、B、C，经过计算还是一样的，这样子调度器会无差别的在A、B、C中任选一台，将请求发过去。

而最短期望将active256/weight的算法改进为(active+1)256/weight

那么还是之前的例子：

A:(1+1)256/1=2/1256=2256

B:(2+1)256/2=3/2256=1.5256

C:(3+1)256、3=4/3256≈1.3256

显然C

④永不排队算法

将请求发给当前连接数为0的服务器上。

⑤基于局部的最少连接调度算法

这种调度算法应用于Cache系统，维持一个请求到一台服务器的映射，其实我们仔细想想哈，之前做的一系列最少连接相关的调度算法。考虑到的是服务器的状态与性能，但是一次请求并不是单向的，就像有一个从未合作过的大牛，他很闲，你让他去解决一个之前碰到过的一个问题，未必有找一个之前已经跟你合作过哪怕现在不怎么闲的臭皮匠效果好哦~，所以基于局部的最少连接调度算法，维持的这种映射的作用是，如果来了一个请求，相对应的映射的那台服务器，没有超载，ok交给老伙伴完事吧，俺放心，如果那台服务器不存在，或者是超载的状态且有其他服务器工作在一半的负载状态，则按最少连接调度算法在集群其余的服务器中找一台将请求分配给它。

⑥基于复制的局部最少连接调度算法

这种调度算法同样应用于cache系统，但它维持的不是到一台服务器的映射而是到一组服务器的映射，当有新的请求到来，根据最小连接原则，从该映射的服务器组中选择一台服务器，如果它没有超载则交给它去处理这个请求，如果发现它超载，则从服务器组外的集群中，按最少连接原则拉一台机器加入服务器组，并且在服务器组有一段时间未修改后，将最忙的那台服务器从服务器组中剔除。

5.Sqoop用起来感觉怎样?

说实话，Sqoop在导入数据的速度上确实十分感人，通过进一步了解，发现Sqoop1和Sqoop2在架构上还是有明显不同的，无论是从数据类型上还是从安全权限，密码暴露方面，Sqoop2都

有了明显的改进，同时同一些其他的异构数据同步工具比较,如淘宝的DataX或者Kettle相比，Sqoop无论是从导入数据的效率上还是从支持插件的丰富程度上，Sqoop还是相当不错滴!!

6.ZooKeeper的角色以及相应的Zookepper工作原理?

果然，人的记忆力是有衰减曲线的，当面试官抛出这个问题后，前者角色，我只答出了两种(leader和follower)，后者原理压根就模糊至忘记了。所以恶补了一下，涉及到Zookeeper的角色大概有如下四种：leader、learner(follower)、observer、client。其中leader主要用来决策和调度，follower和observer的区别仅仅在于后者没有写的职能，但都有将client请求提交给leader的职能，而observer的出现是为了应对当投票压力过大这种情形的，client就是用来发起请求的。而Zookeeper所用的分布式一致性算法包括leader的选举其实和-原始部落的获得神器为酋长，或者得玉玺者为皇帝类似，谁id最小，谁为leader，会根据你所配置的相应的文件在相应的节点机下生成id，然后相应的节点会通过getchildren()这个函数获取之前设置的节点下生成的id，谁最小，谁是leader。并且如果万一这个leader挂掉了或者堕落了，则由次小的顶上。而且在配置相应的zookeeper文件的时候回有类似于如下字样的信息：Server.x=AAAA:BBBB:CCCC。其中的x即为你的节点号哈，AAAA对应你所部属zookeeper所在的ip地址，BBBB为接收client请求的端口，CCCC为重新选举leader端口。

7.HBase的Insert与Update的区别?

这个题目是就着最近的一次项目问的，当时实现的与hbase交互的三个方法分别为insert、delete、update。由于那个项目是对接的一个项目，对接的小伙伴和我协商了下，不将update合并为insert，如果合并的话，按那个项目本身，其实通过insert执行overwrite相当于间接地Update，本质上，或者说在展现上是没什么区别的包括所调用的put。但那仅仅是就着那个项目的程序而言，如果基于HBase shell层面。将同一rowkey的数据插入HBase，其实虽然展现一条，但是相应的timestamp是不一样的，而且最大的版本数可以通过配置文件进行相应地设置。

8.请简述大数据的结果展现方式。

1)报表形式

基于数据挖掘得出的数据报表，包括数据表格、矩阵、图形和自定义格式的报表等，使用方便、设计灵活。

2)图形化展现

提供曲线、饼图、堆积图、仪表盘、鱼骨分析图等图形形式宏观展现模型数据的分布情况，从而便于进行决策。

3)KPI展现

提供表格式绩效一览表并可自定义绩效查看方式，如数据表格或走势图，企业管理者可根据可度量的目标快速评估进度。

4)查询展现

按数据查询条件和查询内容，以数据表格来汇总查询结果，提供明细查询功能，并可在查询的数据表格基础上进行上钻、下钻、旋转等操作。

9.例举身边的大数据。

i.QQ，微博等社交软件产生的数据

ii.天猫，京东等电子商务产生的数据

iii.互联网上的各种数据

10.简述大数据的数据管理方式。

答：对于图像、视频、URL、地理位置等类型多样的数据，难以用传统的结构化方式描述，因此需要使用由多维表组成的面向列存储的数据管理系统来组织和管理数据。也就是说，将数据按行排序，按列存储，将相同字段的数据作为一个列族来聚合存储。不同的列族对应数据的不同属性，这些属性可以根据需求动态增加，通过这样的分布式实时列式数据库对数据统一进行结构化存储和管理，避免了传统数据存储方式下的关联查询。

11.什么是大数据?

答：大数据是指无法在容许的时间内用常规软件工具对其内容进行抓取、管理和处理的数据。

12.海量日志数据，提取出某日访问百度次数最多的那个IP。

首先是这一天，并且是访问百度的日志中的IP取出来，逐个写入到一个大文件中。注意到IP是32位的，最多有个2^32个IP。同样可以采用映射的方法，比如模1000，把整个大文件映射为1000个小文件，再找出每个小文中出现频率最大的IP(可以采用hash_map进行频率统计，然后再找出频率最大的几个)及相应的频率。然后再在这1000个最大的IP中，找出那个频率最大的IP，即为所求。

或者如下阐述(雪域之鹰)：

算法思想：分而治之+Hash

1)IP地址最多有2^32=4G种取值情况，所以不能完全加载到内存中处理;

2)可以考虑采用“分而治之”的思想，按照IP地址的Hash(IP)%1024值，把海量IP日志分别存储到1024个小文件中。这样，每个小文件最多包含4MB个IP地址;

3)对于每一个小文件，可以构建一个IP为key，出现次数为value的Hashmap，同时记录当前出现次数最多的那个IP地址;

4)可以得到1024个小文件中的出现次数最多的IP，再依据常规的排序算法得到总体上出现次数最多的IP;

13.搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为1-255字节。

假设目前有一千万个记录(这些查询串的重复度比较高，虽然总数是1千万，但如果除去重复后，不超过3百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门。)，请你统计最热门的10个查询串，要求使用的内存不能超过1G。

典型的TopK算法，还是在这篇文章里头有所阐述，详情请参见：十一、从头到尾彻底解析Hash表算法。

文中，给出的最终算法是：

第一步、先对这批海量数据预处理，在O(N)的时间内用Hash表完成统计(之前写成了排序，特此订正。July、2011.04.27);

第二步、借助堆这个数据结构，找出TopK，时间复杂度为N‘logK。

即，借助堆结构，我们可以在log量级的时间内查找和调整/移动。因此，维护一个K(该题目中是10)大小的小根堆，然后遍历300万的Query，分别和根元素进行对比所以，我们最终的时间复杂度是：O(N)+N’*O(logK)，(N为1000万，N’为300万)。ok，更多，详情，请参考原文。

或者：采用trie树，关键字域存该查询串出现的次数，没有出现为0。最后用10个元素的最小推来对出现频率进行排序。

14.有一个1G大小的一个文件，里面每一行是一个词，词的大小不超过16字节，内存限制大小是1M。返回频数最高的100个词。

方案：顺序读文件中，对于每个词x，取hash(x)%5000，然后按照该值存到5000个小文件(记为x0,x1,…x4999)中。这样每个文件大概是200k左右。

如果其中的有的文件超过了1M大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过1M。

对每个小文件，统计每个文件中出现的词以及相应的频率(可以采用trie树/hash_map等)，并取出出现频率最大的100个词(可以用含100个结点的最小堆)，并把100个词及相应的频率存入文件，这样又得到了5000个文件。下一步就是把这5000个文件进行归并(类似与归并排序)的过程了。

15.有10个文件，每个文件1G，每个文件的每一行存放的都是用户的query，每个文件的query都可能重复。要求你按照query的频度排序。

还是典型的TOPK算法，解决方案如下：

方案1：

顺序读取10个文件，按照hash(query)%10的结果将query写入到另外10个文件(记为)中。这样新生成的文件每个的大小大约也1G(假设hash函数是随机的)。

找一台内存在2G左右的机器，依次对用hash_map(query,query_count)来统计每个query出现的次数。利用快速/堆/归并排序按照出现次数进行排序。将排序好的query和对应的query_cout输出到文件中。这样得到了10个排好序的文件(记为)。

对这10个文件进行归并排序(内排序与外排序相结合)。

方案2：

一般query的总量是有限的，只是重复的次数比较多而已，可能对于所有的query，一次性就可以加入到内存了。这样，我们就可以采用trie树/hash_map等直接来统计每个query出现的次数，然后按出现次数做快速/堆/归并排序就可以了。

方案3：

与方案1类似，但在做完hash，分成多个文件后，可以交给多个文件来处理，采用分布式的架构来处理(比如MapReduce)，最后再进行合并。

16.JVM&垃圾回收机制

三个代：年轻代（Young Generation）、年老代（Old Generation）和持久代（Permanent Generation）

17.在2.5亿个整数中找出不重复的整数，注，内存不足以容纳这2.5亿个整数。

方案1：采用2-Bitmap(每个数分配2bit，00表示不存在，01表示出现一次，10表示多次，11无意义)进行，共需内存2^32*2bit=1GB内存，还可以接受。然后扫描这2.5亿个整数，查看Bitmap中相对应位，如果是00变01，01变10，10保持不变。所描完事后，查看bitmap，把对应位是01的整数输出即可。

方案2：也可采用与第1题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

18.腾讯面试题：给40亿个不重复的unsignedint的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那40亿个数当中?

第一反应时快速排序+二分查找。以下是其它更好的方法：

方案1：oo，申请512M的内存，一个bit位代表一个unsignedint值。读入40亿个数，设置相应的bit位，读入要查询的数，查看相应bit位是否为1，为1表示存在，为0表示不存在。

方案2：这个问题在《编程珠玑》里有很好的描述，大家可以参考下面的思路

19.怎么在海量数据中找出重复次数最多的一个?

方案1：先做hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求(具体参考前面的题)。

20.上千万或上亿数据(有重复)，统计其中出现次数最多的钱N个数据。

方案1：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用hash_map/搜索二叉树/红黑树等来进行统计次数。然后就是取出前N个出现次数最多的数据了，可以用第2题提到的堆机制完成。

21.一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前10个词，请给出思想，给出时间复杂度分析。

方案1：这题是考虑时间效率。用trie树统计每个词出现的次数，时间复杂度是O(n*le)(le表示单词的平准长度)。然后是找出出现最频繁的前10个词，可以用堆来实现，前面的题中已经讲到了，时间复杂度是O(n*lg10)。所以总的时间复杂度，是O(n*le)与O(n*lg10)中较大的哪一个。

22.WARN TaskSchedulerImpl: Initial job has not accepted any resources; check your cluster uito ensure that workers are registered and have sufficient memory

当前的集群的可用资源不能满足应用程序所请求的资源。

资源分2类： cores 和 ram

Core代表对执行可用的executor slots

Ram代表每个Worker上被需要的空闲内存来运行你的Application。

解决方法：

应用不要请求多余空闲可用资源的

关闭掉已经执行结束的Application

23.Application isn’t using all of the Cores: How to set the Cores used by a Spark App

设置每个App所能获得的core

解决方法：

spark-env.sh里设置spark.deploy.defaultCores 或spark.cores.max

24.Spark Executor OOM: How to set Memory Parameters on Spark

OOM是内存里堆的东西太多了

1）增加job的并行度，即增加job的partition数量，把大数据集切分成更小的数据，可以减少一次性load到内存中的数据量。InputFomart， getSplit来确定。

2）spark.storage.memoryFraction

管理executor中RDD和运行任务时的内存比例，如果shuffle比较小，只需要一点点shuffle memory，那么就调大这个比例。默认是0.6。不能比老年代还要大。大了就是浪费。

3）spark.executor.memory如果还是不行，那么就要加Executor的内存了，改完executor内存后，这个需要重启。

25.Shark Server/ Long Running Application Metadata Cleanup

Spark程序的元数据是会往内存中无限存储的。spark.cleaner.ttl来防止OOM，主要出现在Spark Steaming和Shark Server里。

export SPARK_JAVA_OPTS +="-Dspark.kryoserializer.buffer.mb=10 -Dspark.cleaner.ttl=43200"

26.Class Not Found: Classpath Issues

问题1、缺少jar，不在classpath里。3

问题2、jar包冲突，同一个jar不同版本。

解决1：

将所有依赖jar都打入到一个fatJar包里，然后手动设置依赖到指定每台机器的DIR。

val conf = new SparkConf().setAppName(appName).setJars(Seq(System.getProperty("user.dir") + "/target/scala-2.10/sparktest.jar"))

解决2：

把所需要的依赖jar包都放到default classpath里，分发到各个worker node上。

23.使用mr，spark,spark sql编写wordcount程序

这个网上都很多，

24.如何为一个hadoop任务设置mappers的数量

使用job.setNumMapTask(intn)手动分割，这是不靠谱的

官方文档：“Note:Thisisonlyahinttotheframework”说明这个方法只是提示作用，不起决定性作用

实际上要用公式计算：

Max(min.split，min(max.split，block))就设置分片的最大最下值computeSplitSize()设置

可以参考这篇文章：http://blog.csdn.net/strongerbit/article/details/7440111

25.有可能使hadoop任务输出到多个目录中么?如果可以，怎么做?

答案：在1.X版本后使用MultipleOutputs.java类实现

26.如何为一个hadoop任务设置要创建的reducer的数量

配置job.setNumReduceTask(intn)

或者调整hdfs-site.xml中的mapred.tasktracker.reduce.tasks.maximum默认参数值

27.Spark Streaming和Storm有何区别?

一个实时毫秒一个准实时亚秒，不过storm的吞吐率比较低。

28.如果公司叫你写hadoop平台设计方案，你会如何规划Hadoop生产集群?

这个题目比较考验全局观，站在架构师的层面去思考

29.hadoop集群监控，你会关注哪些监控点?

偏重集群的运维

30.实际生产中，传统关系型数据库如何迁移到hadoop平台，迁移过程中，你遇到了哪些问题?

希望持续更新

厉害楼主感谢