

大家好，我是球球~

最近看到了各大厂裁员的消息，没办法，到了毕业季，大家总有一天会各奔东西，今天给大家分享一些面试题，希望大家能拿到更好的offer！

来源：大数据真好玩

1、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，driver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Driver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。

2、Spark为什么比mapreduce快？

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Lineage，精华部分就是DAG和Lineage

简单说一下**hadoop**和**spark**的**shuffle**相同和差异？

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。

4、spark 工作机制？

① 构建 Application 的运行环境，Driver 创建一个 SparkContext

② SparkContext 向资源管理器（Standalone、Mesos、Yarn）申请 Executor 资源，资源管理器启动 StandaloneExecutorBackend（Executor）

③ Executor 向 SparkContext 申请 Task

④ SparkContext 将应用程序分发给 Executor

⑤ SparkContext 就建成 DAG 图，DAGScheduler 将 DAG 图解析成 Stage，每个 Stage 有多个 task，形成 taskset 发送给 task Scheduler，由 task Scheduler 将 Task 发送给 Executor 运行

⑥ Task 在 Executor 上运行，运行完释放所有资源

5、spark 的优化怎么做？

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 heap内存等等

6、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

7、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Lineage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。

8、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

9、Spark的shuffle过程？

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>

10、Spark的数据本地性有哪几种？

Spark中的数据本地性有三种：

- 1) PROCESS_LOCAL是指读取缓存在本地节点的数据
- 2) NODE_LOCAL是指读取本地节点硬盘数据
- 3) 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。

 大数据真好玩

11、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。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。

 大数据真好玩

12、介绍一下join操作优化经验？

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。

13、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) ResourceManager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。

注意：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。

 大数据真好玩

14、Spark on Yarn 模式有哪些优点？

1) 与其他计算框架共享集群资源（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。

 大数据真好玩

15、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ResourceManager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

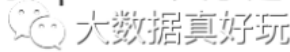
 大数据真好玩

16、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中有一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消耗。
- 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可以有效减少stage的执行消耗，同时可以优化执行路径。

 大数据真好玩

17、介绍partition和block有什么关联关系？

- 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。

18、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。

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

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。

20、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。

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

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。

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


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, 内存都由以下部分构成:

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rdd.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。|

 大数据真好玩

23、Spark有哪两种算子?

Transformation（转化）算子和Action（执行）算子。

24、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？

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。

25、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。

26、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

27、Spark并行度怎么设置比较合适？

spark并行度，每个core承载24个partition,如，32个core，那么64~128之间的并行度，也就是设置64~128个partition，并行度和数据规模无关，只和内存使用量和cpu使用时间有关。

28、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。

29、collect功能是什么，其底层是怎么实现的？

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。

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

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。

31、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。

32、Spark on Mesos中，什么是粗粒度分配，什么是细粒度分配，各自的优点和缺点是什么？

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；优点：作业特别多时，资源复用率高，适合粗粒度；缺点：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。

33、driver的功能是什么？

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，是程序的入口点；

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。

34、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？

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 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Kafka、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。

35、Spark中Worker的主要工作是什么？

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。

36、Mapreduce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:

- 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DAGScheduler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TaskScheduler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。
- 4) 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。

大数据真好玩

37、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。

38、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

39、cache和pesist的区别?

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 1) 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; 2) executor执行的时候, 默认60%做cache, 40%做task操作, persist是最根本的函数, 最底层的函数。

40、**cache**后面能不能接其他算子,它是不是**action**操作?

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。

41、**reduceByKey**是不是**action**?


不是,很多人都会以为是action, reduce rdd是action

42、**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。

 大数据真好玩

其他面试题推荐

- 1.rdd的属性
- 2.算子分为哪几类(RDD支持哪几种类型的操作)
- 3.创建rdd的几种方式
- 4.spark运行流程
- 5.Spark中coalesce与repartition的区别
- 6.sortBy 和 sortByKey的区别
- 7.map和mapPartitions的区别
- 8.数据存入Redis 优先使用map mapPartitions foreach foreachPartitions哪个

9.reduceByKey和groupByKey的区别

10.cache和checkPoint的比较

11.spark streaming流式统计单词数量代码

12.简述map和flatMap的区别和应用场景

13.计算曝光数和点击数

14.分别列出几个常用的transformation和action算子

15.按照需求使用spark编写以下程序，要求使用scala语言

16.spark应用程序的执行命令是什么？

17.Spark应用执行有哪些模式，其中哪几种是集群模式

18.请说明spark中广播变量的用途

19.以下代码会报错吗？如果会怎么解决 `val arr = new ArrayList[String]; arr.foreach(println)`

20.写出你用过的spark中的算子，其中哪些会产生shuffle过程

21.Spark中rdd与partition的区别

22.请写出创建Dataset的几种方式

23.描述一下RDD，DataFrame，DataSet的区别？

24.描述一下Spark中stage是如何划分的？描述一下shuffle的概念

25.Spark 在yarn上运行需要做哪些关键的配置工作？如何kill 一个Spark在yarn运行中Application

26.通常来说，Spark与MapReduce相比，Spark运行效率更高。请说明效率更高来源于Spark内置的哪些机制？请列举常见spark的运行模式？

27.RDD中的数据在哪？

28.如果对RDD进行cache操作后，数据在哪里？

29.Spark中Partition的数量由什么决定

30.Scala里面的函数和方法有什么区别

31.SparkStreaming怎么进行监控？

32.Spark判断Shuffle的依据？

33.Scala有没有多继承？可以实现多继承么？

34.Sparkstreaming和flink做实时处理的区别

35.Sparkcontext的作用

36. Sparkstreaming读取kafka数据为什么选择直连方式

37. 离线分析什么时候用sparkcore和sparksql

38. Sparkstreaming实时的数据不丢失的问题

39. 简述宽依赖和窄依赖概念, groupByKey, reduceByKey, map, filter, union五种操作哪些会导致宽依赖, 哪些会导致窄依赖

40. 数据倾斜可能会导致哪些问题, 如何监控和排查, 在设计之初, 要考虑哪些来避免

41. 有一千万条短信, 有重复, 以文本文件的形式保存, 一行一条数据, 请用五分钟时间, 找出重复出现最多的前10条

42. 现有一文件, 格式如下, 请用spark统计每个单词出现的次数

43. 共享变量和累加器

44. 当 Spark 涉及到数据库的操作时, 如何减少 Spark 运行中的数据库连接数?

45. 特别大的数据, 怎么发送到executor中?

46. spark调优都做过哪些方面?

47. spark任务为什么会被yarn kill掉?

48. Spark on Yarn作业执行流程? yarn-client和yarn-cluster有什么区别?

49. Flatmap底层编码实现?

50. spark_1.X与spark_2.X区别

51. 说说spark与flink

52. spark streaming如何保证7*24小时运行机制?

53. spark streaming是Exactly-Once吗?

