## hadoop及其生态内技术部分

1. hadoop1.x中namenode保存的元数据有什么？

文件名、权限、大小、被分成哪些块，注意磁盘上保存的并没有块分布在哪些node上，而是在启动集群的时候datanode上报的，在内存中。

1. hadoop1.x中secondarynamenode是如何合并fsimage和edit log的？

①新建一个edits文件用来记录此时刻之后的用户所有操作日志记录

②把此时刻之前的edits文件和对用的fsimage文件拷贝到snn上

③snn进行合并

④把心的fsimage文件传给nn

⑤替换旧的fsimage

注意：内存中的元素局与fsimage中的同步问题，不要认为内存中的慢，因为在操作的时候记录edits的时候内存中就已经改变了，而fsimage就是一个持久化的机制而已，用于再次启动的时候加载。

合并时机：

①fs.checkpoint.period默认3600秒

②fs.checkpoint.size，默认edits文件64M

1. HDFS读写流程？

读流程：

①调用DistributedFileSystemAPI的open方法

②从NN中获取元数据——数据块的位置信息

③调用FSDataInputStreamAPI的read方法

④具体read读取数据，根据第二步返回的block信息，比如block1有三个副本，会找一个空闲的副本所在datanode所在机器读取，其他的Block可以同时进行。

⑤调用FSDataInputStreamAPI的close方法

写流程：

①调用DsitributedFileSystemAPI的create方法

②访问NN，告诉NN我要上传文件，并把文件名、权限、大小、用户给NN，此时，NN会根据文件的大小分blocks，以及blocks的第一个副本需要放在哪些机器上

③调用FSDataOutputStreamAPI的write方法

④根据第二步NN反馈的信息把block的第一个副本写入指定的DN上，写完第一个副本就结束

⑤调用FSDataOutputStreamAPI的close方法

⑥反馈给NN说success

注意：第四步client往DN上写完第一个副本就结束了，那么副本机制怎么实现？这里的第二个第三个副本的复制以及位置选择都由第一个副本所在的DN决定并完成！

1. 说一说你对MR的shuffle过程的理解？

从整理上来说shuffle过程主要分为三个机制的合作：分区、排序、合并、合并

①partition分区

把map的输出进行分区，默认分区是根据key的hash模 partition，可以自定义，目的是负载均衡

②sort排序

当缓冲区快满的时候对80M（总大小100M）的数据进行sort，默认的排序规则是字典排序，一定要注意字典排序中11在9个前面

③combiner合并（可有可无）

内存中有个环形缓冲区默认100M，在内存中进行partition和sort操作，当达到80%的时候将会发生溢写到磁盘，如果大量的小文件肯定不适合网络传输，所以在溢写之前进行combiner操作，将相同key的value加起来，减少溢写数据量。

④merge合并

小文件合并为大文件

用一个例子说一下比较明了

map读入的记录如下（两个reduce）：c: 2,a: 1,b: 1, b: 1,d: 1

partition操作：a和c分为一个区，b和d分为一个区

sort操作：环形缓冲区快满的时候，a和c位置调换，总的顺序是[a:1 c:1][b:1 b:1 d:1]

merge操作：这里合并多次溢写的临时文件{a,[1,1,1,1,1...],c,[1,1,1,1]}...

如果存在combiner的话：

sort之后溢写之前会进行combiner：[a:1 c:1][b:1 b:1 d:1]变成[a:1 c:1][b:2 d:1]

注意：从map端的merge可以看出为什么reduce接收到的value是一个迭代器了！

注意：combiner和reduce端的merge的区别。conbiner是我们可以操控的，合不合并，怎么合并，产生什么的keyvalue我们说了算，但是reduce端的merge和sort是不可控的，是同一个merge和sort，并且merge只是合并相同key，而不是可控的。’

那么map端的merge和reduce端的merge有什么不同呢？也是有区别的，因为map端的输出和copy是同步进行的，所以map端的merge并不是merge所有map输出结果，但是reduce端的merge必须是所有的map输出copy过来的数据的总merge，不然数据肯定会出错。因为reduce的输入是按照key+iterator的，所有不允许出现相同key的多个结果。

1. MR的split大小？

max(min.split, min(max.split, block))

换成语言就是说，最大只能是max.split，最小只能是min.split，其他时候根据block大小决定，所以一般我们说split的大小是64M，不太准确

注意：可以优化，设置最小split和最大split的大小！

1. 请解释一下为什么hadoop2.x解决了hadoop1.x的NN的单点问题和内存受限问题？

单点问题：HA

内存受限：Federation

HA机制：

①两个NN，一个主一个备，DN要想所有的NN汇报块信息、心跳

②备用NN通过journalNode来同步元数据，所以当主宕机时，元数据还可以获取到

③FailoverController这个是主备切换的关键，用来检查NN的心跳和切换NN，具体操作就是Failovercontroller向zk汇报NN情况，如果主宕机了的话，zk就知道了，此时会通知备NN你们来竞争吧，竞争到锁的NN就切换为active。

注意：之前一直有一个疑问就是client是如何找到正在处于active的NN的IP的？纠结过一段时间，现在可以认为是zk的存在，因为zk可以做统一命名空间嘛，这个命名空间下有多个ip，这样client每次去找zk问我们的cluster现在活着的是哪个ip，所以....

Federation机制：

解决了NN可以水平扩展的问题，但是只有大公司才会用HA+Federation，直观上就是通过多个命名空间namespace将元数据的存储和管理分布在多态机器上。

注意：上面说的关于元数据高可用是有错误的！！！

关于两个NN和JN之间到底什么怎么实现元数据的高可用的？edits log和fsimage到底在哪合并的？

①客户端的操作请求发给Active NN，NN会在内存中修改，然后在edits log中追加一条操作记录，注意些edits log同样也会发生在JN上，所以他们是同步写入edits log的，所以并不是JN拉取的

②一把来说checkpoint合并操作并不会在Active NN上进行，所以会在StanBy NN上进行合并操作，具体就是StandBy根据dfs.namenode.checkpoint.preiod（默认3600秒）或者dfs.namenode.checkpoint.txns（默认100万次）时进行合并fsimage和edits，具体操作如下

③StandBy NN检查是否达到checkpoint条件（两个，上面说了）

④达到checkpoint条件后，将该namespace以fsimage.ckpt\_txid格式保存，并随机生成一个MD5文件，然后将fsimage.ckpt\_txid重命名为fsimage\_txid

⑤StandBy通过http与Active NN建立连接

⑥Active NN通过HTTP获取fsimage\_txid文件并保存为fsimage.ckpt\_txid文件，也生成一个MD5文件，ActiveNN与StandByNN上MD5文件比较，如果无误就将fsimage.xkpt\_txid重命名为fsimage\_txid。

所以关于第二个问题：到底在哪里进行的checkpoint，答案是StandBy NN上。

1. 简单说一说yarn，然后具体说一说任务申请执行的整体流程？

yarn是hadoop2.x引入，最重要的目的就是将资源分配和任务调度分离！！同时可以兼容第三方的计算框架。RM负责整个集群的资源分配，AM（applicationmaster）负责任务的分割、调度、监控、容错，RM不负责具体任务的容错，负责AM的容错，任务失败后AM向RM重新申请资源。

下面就说一下任务申请、调度、执行的流程：

step1：客户端向RM提交一个作业申请

step2：RM根据请求信息返回一些信息，包括application\_id、input路径对应的文件元数据、作业资源提交的hdfs路径

step3：客户端根据RM的返回信息生成资源文件（job.split、job.xml、app.jar）提交到指定hdfs路径

step4：向RM申请运行AM

step5：AM将用户的请求打包为task放置到任务队列中，等待RM的调度策略调度（从web上可以看出来，申请任务只有就会有AM出现但是是accepted状态而不是running状态）

step6：当RM调度到该task的时候会像一个NM分配该task，该NM从hdfs下载任务资源信息并运行AM

step7：！！AM是任务调度中心！AM根据任务资源信息想RM申请运行Map task的资源

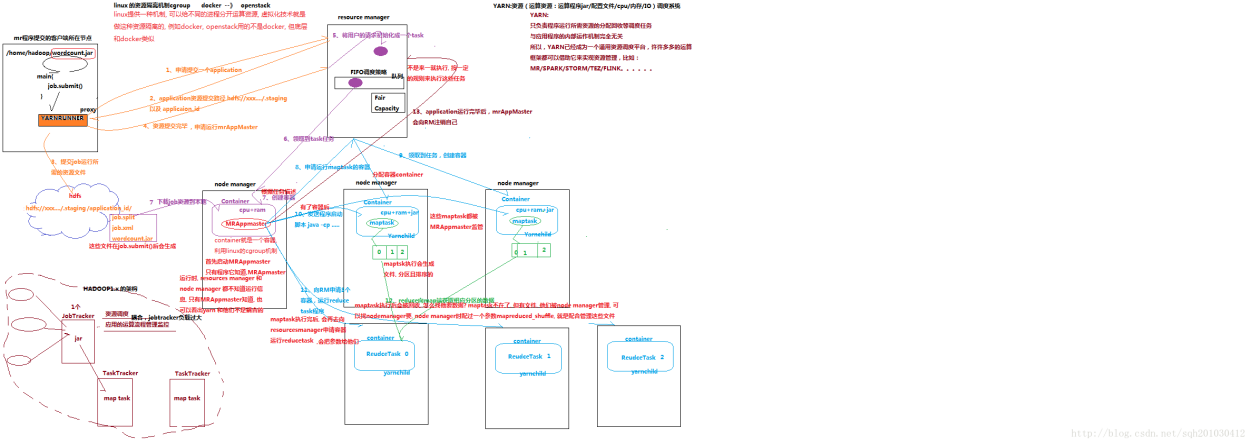
step8：注意：这里容易出错，AM申请的Map task仍然会放到RM的任务队列中等待，在map task运行之前AM显示的状态一直是accepted

step9：RM针对资源的调度是有自己的一套策略的，包括根据内存分配、资源阈值、动态分配等等，如果资源申请完毕，RM会指定哪些NM来运行containers，container下载task、资源文件信息

step10：AM启动NM上container的map task注意！！！container中的maptask不是RM启动的而是AM启动的！

step11：待运行到一定程度后会申请reduce task资源，重复上述过程

step12：AM向RM申请注销自己



注意：这里有一个优化就是container的重用，以及copy阶段的内存占用比例。

1. RM的三种调度策略对比以及适用条件？

①FIFO先进先出

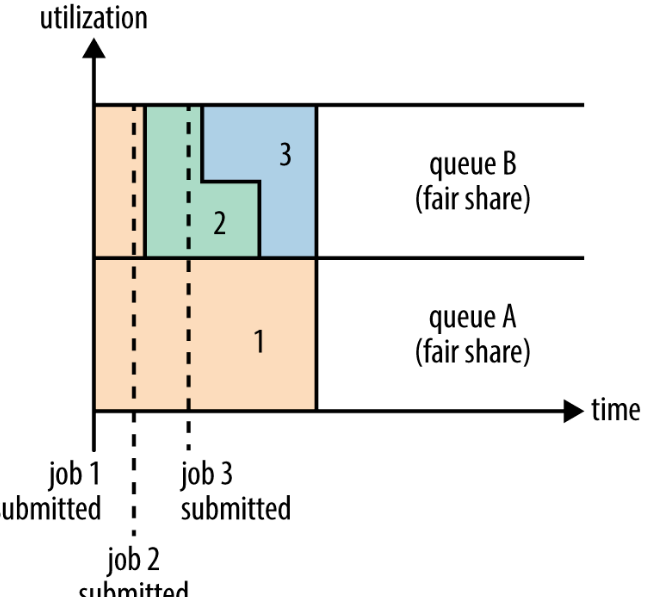
默认调度器，肯定不行

②capacity scheduler容器调度器

支持多队列，每个队列预先设定资源量，每个队列采用FIFO策略，存在资源抢占，但是仅限于配置中的最大设置

③fair schedule公平调度器

支持多队列，每个队列设置资源量，但是和capacity的区别在于队列中并不是固定的FIFO，而是可以设置的，默认是公平策略，说白了就是在分队列的情况下，内列内部的job是公平如下图



1. NN是如何用double-buffer双缓冲机制保证高并发访问的？

换句话问：每秒上千次请求的时候，NN多线程操作内存并进行edits log的磁盘写和网络传输给JN，多线程高并发存在安全隐患，也就是txid顺序的问题，那么为了保证安全就要加锁，那样的话，嘿嘿性能一落千丈，hadoop是如何解决的呢？

Double-Buffer：

hdfs采用一块缓冲内存，分成两部分，一部分专门用于edits log的写入，另一部分用于读出写入磁盘、通过网络写入journalnode

分段加锁+双缓冲机制：

第一阶段锁：各个操作线程依次获取锁，生成顺序的txid，把edits log写入第一块缓冲内存后，瞬间释放锁，这样操作元数据和写入edits log就比较快。可以感觉出并发的感觉。

第二阶段锁：之前的线程中的一个获取写磁盘和写网络的锁，如果当前没有线程在持久化edits，那么它就会把两块缓冲互换进行写磁盘和写网络操作，另一块可以进行并发写入，但是不读取。其他线程竞争这个锁发现有线程在写的话会阻塞（如果自己的txid已经被之前的线程写入磁盘的话就不阻塞，直接返回即可）

1. HDFS上传TB级大文件是如何优化上传速度的？

可以与hdfs写操作联系起来，当NN创建目录索引、返回第一个block副本需要上传到哪个DN上，之后client与DN打交道。这个过程无非就是本地输入输出流+拷贝流+网络传输+输入输出流的过程。但是如果基于简单的socket网络传输如用byte[]来作为缓冲效率肯定底下。

这就是为什么调用FSDataStreamInputAPI和FSDataStreamOuputAPI的write和close的原因。

Chunk缓冲+Packet缓冲+异步发送：

①chunk缓冲机制

inputstream读入的本地数据进入FSDataStreamoutput，数据会写入一个chunk缓冲数组，每个chunk有512字节大小，这样就可以不用因为等待网络传输而阻塞读入

②Packet数据包机制

chunk数组写满后就会进行切割，多个chunk数据段会写入Packet数据包，一个packet可以容纳127个chunk，默认64M，其实可以认为是另一层缓冲

③内存队列异步发送机制

当一个packet被塞满后，就会把packet放入队列，DataStreamer线程不断获取packet数据包并把它发送出去。

1. 谈一谈hadoop文件的续约检查机制是怎么实现的？

TreeSet：底层是TreeMap，TreeMap是排序的kv，底层是红黑树。

文件契约机制：

hadoop是不支持多客户端对同一文件的并发写入的，所以hadoop有一个文件契约机制保证同一时间只有一个客户端获取文件的契约。

但是有一个问题就是当长时间没有写入时，有一个时间阈值，hadoop认为放弃契约，解除契约好让其他客户端写入，但是低效的问题在这里，如果快速过滤过期且没有续约的契约。

Treeset可以按照续约时间来作为排序的key，这样每次只需要看最前头的契约是否过期即可，如果第一个都没过期其他的契约更不可能过期....

1. 你知道hadoop内部的数据限流吗？

①Balancer数据平衡数据流：如raplaceBlock限流，默认是1M。dfs.datanode.balance.bandwidthPerSec

②fdimage镜像文件的上传下载数据流传输，默认不开启dfs.image.transfer.bandwidthPerSec

③VolumeScanner磁盘扫描的数据读操作的数据传输，比如在探寻block是否损坏的时候，为了方式读取数据块影响业务的IO而设置的，默认1Mdfs.block.scanner.volume.bytes.per.second

注意：有一个优化点就是限流的period以及提高balance的带宽、balance的线程数等，因为限流的机制是period内流量是否用完，如果设置太小会有太多阻塞！

既然提到了数据平衡，那么就说一下数据平衡的过程吧？

1. 如果hdfs数据节点挂载多个磁盘大小不一致会出现什么情况？

会出现该节点小磁盘写满后该服务器上datanode和nodemanager挂掉，如何解决？

hadoop写磁盘策略：

①循环选择策略

很简单，循环往多个挂载的磁盘上写block，很容易造成小磁盘写满

②可用空间策略

剩余空间减去磁盘剩余阈值（配置）如果可以大于block大小，那么进行循环选择，如果小于block大小，那就找可以撑得下的磁盘写入

注意：可优化的点：第一：可以设置写入策略为可用空间策略和阈值；第二：设置datanode.dfs.reserved，这个设置目的是防止磁盘写满之后无法运行mapreduce任务；第三：设置datanode运行损坏的磁盘数，比如说挂载的4块，可以设置允许损坏1块。

## spark部分

1.