spark源码分析执行流程

2019年5月2日 14:43

当spark遇到action类算子，开始调起任务

1.Action类型的算子触发job的执行。源码中调用了SparkContext的runJob()方法，根进源码发现底层调用的是DAGScheduler的runJob()方法。

2.DAGScheduler会将我们的job按照宽窄依赖划分为一个个stage(每个stage根据RDD的Partition的个数决定task的个数)，每个stage中有一组并行计算的task，每一个task都可以看做是一个pipeline，这个管道里面数据是一条一条被计算的，每经过一个RDD会经过一次处理，RDD是一个抽象的概念里面存储的是一些计算的逻辑，每一条数据计算完成之后会在shuffle write过程中将数据落地写入我们的磁盘中。

将 Taskset 传给底层调度器

a）– spark-cluster TaskScheduler

b）– yarn-cluster YarnClusterScheduler

c）– yarn-client YarnClientClusterScheduler

3.stage划分完之后会以TaskSet的形式（实际上是task的list集合）提交给我们的TaskScheduler。TaskScheduler接收到TaskSet之后会进行遍历，将每个元素依次调用launchTask()方法，launchTask()根据数据本地化的算法发送task到指定的Executor中执行。task在发送到Executor之前首先进行序列化，Executor中有ThreadPool，ThreadPool中有很多线程，在这里面来具体执行我们的task。

4.TaskScheduler和Executor之间有通信（Executor有一个邮箱（消息循环体CoresExecutorGraintedBackend）），Executor接收到task后首先将task反序列化（得到task的list集合），反序列化后将这个task变为taskRunner（new taskRunner），Executor中的ThreadPool中启动相应的线程接收并且计算相应的task任务。

5.Executor接收到task任务先执行stage1中的task，计算结果会在shuffle write阶段数据落地，数据落地会根据我们的分区策略写入不同的磁盘小文件中，stage1中task全部执行完以后，会向Driver中的DAGScheduler对象里面的MapOutPutTracker发送每一个task的执行状态，以及生成的中间文件的地址。然后，stage2的task开始执行，stage2中task的输入数据就是stage1中task的输出数据，stage2中的task会先向Driver中MapOutPutTracker请求上一批中间文件的地址，拿到地址后stage2-task所在的Executor里面的BlockManager向stage1- task所在的Executor先建立连接，连接是由ConnectionManager负责的，然后由BlockTransformService去拉取数据，（如果使用到了广播变量，stage1-task或者stage2-task会先向它所在的Executor中的BlockManager要广播变量，没有的话，本地的BlockManager会去连接Driver中的BlockManagerMaster，连接完成之后由BlockTransformService将广播变量拉取过来）

6.程序按照上述流程执行直到最后一个stage执行完毕，最后一个stage输出的结果即程序最终输出结果。

spark节点调度过程

2019年5月2日 14:51

1.我们使用spark-submit提交一个Spark作业之后，这个作业就会启动一个对应的Driver进程。根据你使用的部署模式（deploy-mode）不同，Driver进程可能在本地启动，也可能在集群中某个工作节点上启动。Driver进程本身会根据我们设置的参数，占有一定数量的内存和CPU core。而Driver进程要做的第一件事情，就是向集群管理器（可以是Spark Standalone集群，也可以是其他的资源管理集群）申请运行Spark作业需要使用的资源，这里的资源指的就是Executor进程。YARN集群管理器会根据我们为Spark作业设置的资源参数，在各个工作节点上，启动一定数量的Executor进程，每个Executor进程都占有一定数量的内存和CPU core。

2.在申请到了作业执行所需的资源之后，Driver进程就会开始调度和执行我们编写的作业代码了。Driver进程会将我们编写的Spark作业代码分拆为多个stage，每个stage执行一部分代码片段，并为每个stage创建一批task，然后将这些task分配到各个Executor进程中执行。task是最小的计算单元，负责执行一模一样的计算逻辑（也就是我们自己编写的某个代码片段），只是每个task处理的数据不同而已。一个stage的所有task都执行完毕之后，会在各个节点本地的磁盘文件中写入计算中间结果，然后Driver就会调度运行下一个stage。下一个stage的task的输入数据就是上一个stage输出的中间结果。如此循环往复，直到将我们自己编写的代码逻辑全部执行完，并且计算完所有的数据，得到我们想要的结果为止。

3.Spark是根据shuffle类算子来进行stage的划分。如果我们的代码中执行了某个shuffle类算子（比如reduceByKey、join等），那么就会在该算子处，划分出一个stage界限来。可以大致理解为，shuffle算子执行之前的代码会被划分为一个stage，shuffle算子执行以及之后的代码会被划分为下一个stage。因此一个stage刚开始执行的时候，它的每个task可能都会从上一个stage的task所在的节点，去通过网络传输拉取需要自己处理的所有key，然后对拉取到的所有相同的key使用我们自己编写的算子函数执行聚合操作（比如reduceByKey()算子接收的函数）。这个过程就是shuffle。

4.当我们在代码中执行了cache/persist等持久化操作时，根据我们选择的持久化级别的不同，每个task计算出来的数据也会保存到Executor进程的内存或者所在节点的磁盘文件中。

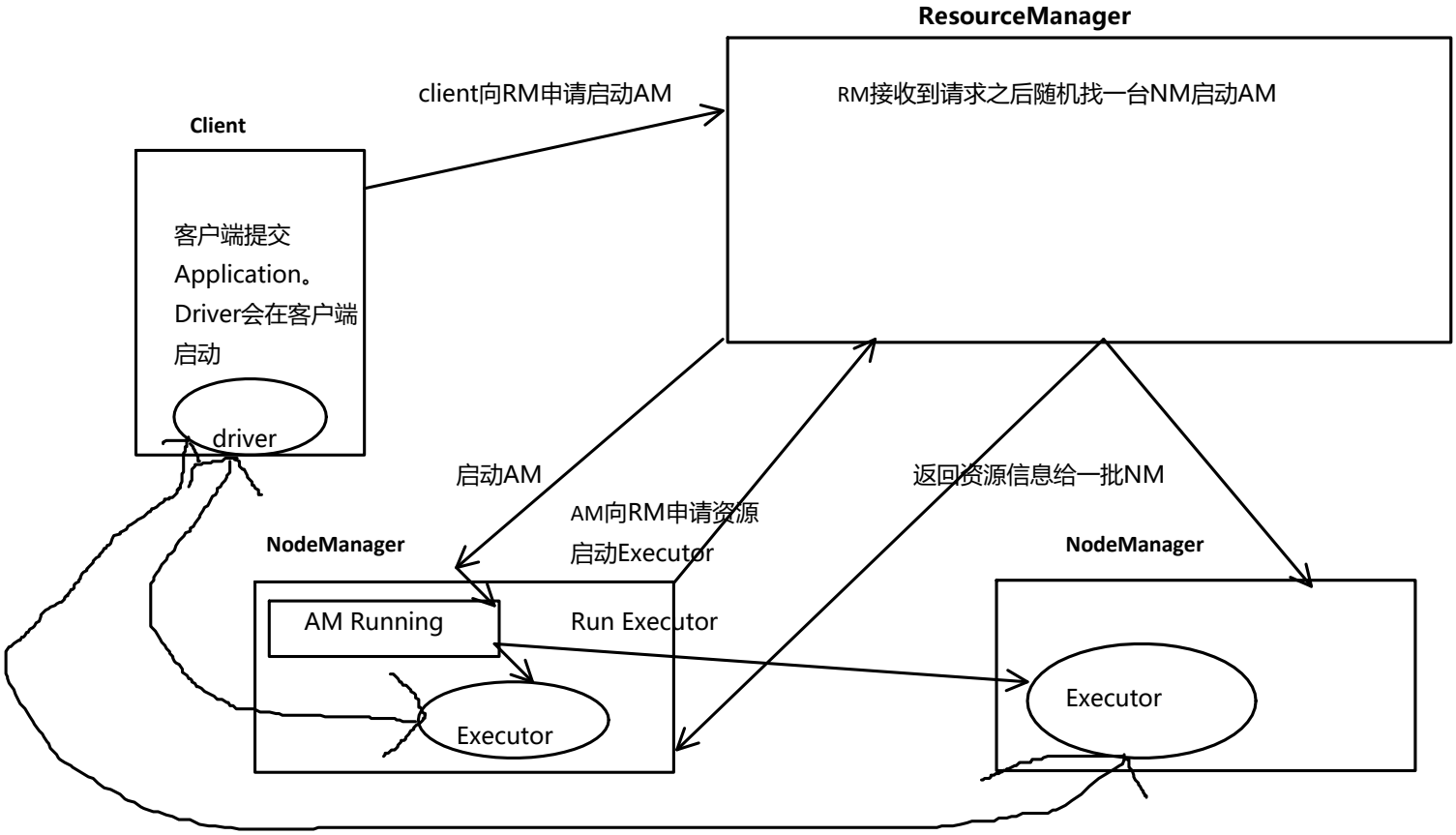
5.因此Executor的内存主要分为三块：第一块是让task执行我们自己编写的代码时使用，默认是占Executor总内存的20%；第二块是让task通过shuffle过程拉取了上一个stage的task的输出后，进行聚合等操作时使用，默认也是占Executor总内存的20%；第三块是让RDD持久化时使用，默认占Executor总内存的60%。

6.task的执行速度是跟每个Executor进程的CPU core数量有直接关系的。一个CPU core同一时间只能执行一个线程。而每个Executor进程上分配到的多个task，都是以每个task一条线程的方式，多线程并发运行的。如果CPU core数量比较充足，而且分配到的task数量比较合理，那么通常来说，可以比较快速和高效地执行完这些task线程。

YARN**调度过程**(YARN-Client)

2019年5月2日 14:56

YARN**调度过程**(YARN-Client)

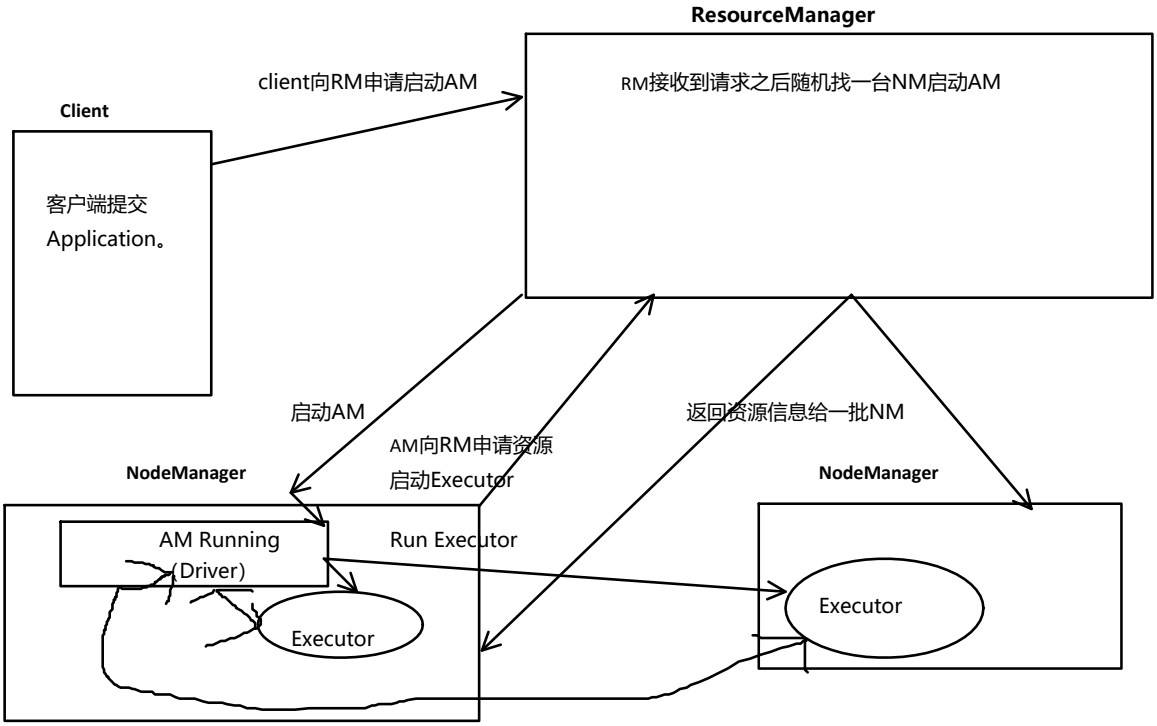


1.客户端提交一个Application，在客户端启动一个Driver进程。
2.Driver进程会向ResourceManager发送请求，启动ApplicationMaster的资源。
3.ResourceManager收到请求，随机选择一台NodeManager启动ApplicationMaster。这里的NodeManager相当于Standalone中的Worker节点。
4.ApplicationMaster启动后，会向ResourceManager请求一批container资源，用于启动Executor.
5.ResourceManager会找到一批NodeManager返回给ApplicationMaster,用于启动Executor。
6.ApplicationMaster会向NodeManager发送命令启动Executor。
7.Executor启动后，会反向注册给Driver，Driver发送task到Executor,执行情况和结果返回给Driver端

YARN调度过程(YARN-Cluster)

2019年5月2日 15:19

YARN调度过程(YARN-Cluster)



.Executor启动后，会反向注册给Driver，Driver发送task到Executor,执行情况和结果返回给Driver端

1.客户端提交一个Application
2.Driver进程会向ResourceManager发送请求，启动ApplicationMaster的资源。
3.ResourceManager收到请求，随机选择一台NodeManager启动ApplicationMaster。这里的NodeManager相当于Standalone中的Worker节点。
4.ApplicationMaster启动后，会向ResourceManager请求一批container资源，用于启动Executor.
5.ResourceManager会找到一批NodeManager返回给ApplicationMaster,用于启动Executor。
6.ApplicationMaster会向NodeManager发送命令启动Executor。
7.Executor启动后，会反向注册给Driver，Driver发送task到Executor,执行情况和结果返回给Driver端

Yarn-cluster和Yarn-client区别就是Yarn-cluster的driver端是在节点中随机选取启动，Yarn-client的driver端是在任务提交的节点启动。

RDD容错机制之checkpoint

2019年5月2日 17:10

checkpoint是什么：

（1）Spark 在生产环境下经常会面临transformation的RDD非常多（例如一个Job中包含1万个RDD）或者具体transformation的RDD本身计算特别复杂或者耗时（例如计算时长超过1个小时），这个时候就要考虑对计算结果数据的持久化；

（2）Spark是擅长多步骤迭代的，同时擅长基于Job的复用，这个时候如果能够对曾经计算的过程产生的数据进行复用，就可以极大的提升效率；

（3）如果采用persist把数据放在内存中，虽然是快速的，但是也是最不可靠的；如果把数据放在磁盘上，也不是完全可靠的！**例如磁盘会损坏，系统管理员可能清空磁盘。**

（4）Checkpoint的产生就是为了相对而言更加可靠的持久化数据，在Checkpoint的时候可以指定把数据放在本地，并且是多副本的方式，但是在生产环境下是放在HDFS上，这就天然的借助了HDFS高容错、高可靠的特征来完成了最大化的可靠的持久化数据的方式；

（5）Checkpoint是为了最大程度保证绝对可靠的复用RDD计算数据的Spark高级功能，通过checkpoint我们通常把数据持久化到HDFS来保证数据最大程度的安全性；

（6）Checkpoint就是针对整个RDD计算链条中特别需要数据持久化的环节（后面会反复使用当前环节的RDD）开始基于HDFS等的数据持久化复用策略，通过对RDD启动checkpoint机制来实现容错和高可用；由此当加入进行一个1万个步骤，在9000个步骤的时候persist，数据还是有可能丢失的，但是如果checkpoint，数据丢失的概率几乎为0。

checkpoint原理机制：

（1）当RDD使用cache机制从内存中读取数据，如果数据没有读到，会使用checkpoint机制读取数据。此时如果没有checkpoint机制，那么就需要找到父RDD重新计算数据了，因此checkpoint是个很重要的容错机制。checkpoint就是对于一个RDD chain（链），如果中间某些中间结果RDD，后面需要反复使用该数据，可能因为一些故障导致该中间数据丢失，那么就可以针对该RDD启动checkpoint机制，checkpoint，首先需要调用sparkContext的setCheckpoint方法，设置一个容错文件系统目录，比如hdfs，然后对RDD调用checkpoint方法。之后再RDD所处的job运行结束后，会启动一个单独的job，来将checkpoint过的数据写入之前设置的文件系统持久化，进行高可用。所以后面的计算在使用该RDD时，如果数据丢失了，但是还是可以从它的checkpoint中读取数据，不需要重新计算。

（2）persist或者cache与checkpoint的区别在于,前者持久化只是将数据保存在BlockManager中但是其lineage是不变的，但是后者checkpoint执行完后，rdd已经没有依赖RDD，只有一个checkpointRDD，checkpoint之后，RDD的lineage就改变了。而且，持久化的数据丢失的可能性更大，因为可能磁盘或内存被清理，但是checkpoint的数据通常保存到hdfs上，放在了高容错文件系统。