一:Stage划分算法

1. SparkApplication中可以因为不同的Action触发众多的Job，也就是说一个Application中可以有很多的Job，每个Job是由一个或者多个Stage构成的，后面的Stage依赖于前面的Stage，也就是说只有前面依赖的Stage计算完毕后，后面的Stage才会运行；
2. Stage划分的依据就是宽依赖，什么产生宽依赖呢？例如reduceByKey，groupByKey，等等；
3. 由Action（例如collect）导致了SparkContext.runJob的执行，最终导致了DAGScheduler中的submitJob的执行，其核心是通过发送一个case class JboSubmitted对象给eventProcessLoop，eventProcessLoop是DAGSchedulerEventProcessLoop的具体实例，而DAGSchedulerEventProcessLoop是EventLoop的子类，具体实现EventLoop的onReceive方法，onReveive方法转过来回调doOnReceive。
4. 自己给自己发送消息作用：第一：可以支持多个job的提交。都放在队列中慢慢处理。第二：无论自己发消息还是别人发消息，都采用一个消息循环线程去处理的话，这个时候大家处理的方式就是统一的，逻辑思路就是一致的。扩展性就非常好，代码也会非常干净，因为代码一般是分工协作。背后是线程然后是队列。
5. 在doOnReceive中通过模式匹配的方式把执行路由到handleJobSubmitted。
6. 在handleJobSubmitted中首先finalStage，创建finalStage时候会建立父Stage的依赖链条；

二：Task任务本地性算法实现：

1. 在submitMissingTasks中会通过调用以下代码来获得任务的本地性：

**val** taskIdToLocations: Map[Int, Seq[TaskLocation]] = **try** {  
 stage **match** {  
 **case** s: ShuffleMapStage =>  
 partitionsToCompute.map { id => (id, getPreferredLocs(stage.rdd, id))}.toMap  
 **case** s: ResultStage =>  
 **val** job = s.activeJob.get  
 partitionsToCompute.map { id =>  
 **val** p = s.partitions(id)  
 (id, getPreferredLocs(stage.rdd, p))  
 }.toMap  
 }  
}

1. 具体一个Partition中的数据本地性的算法实现为下述代码中

**private**[spark]  
**def** getPreferredLocs(rdd: RDD[\_], partition: Int): Seq[TaskLocation] = {  
 getPreferredLocsInternal(rdd, partition, **new** HashSet)  
}

在具体算法实现的时候首先查看DAGScheduler的内存数据结构中是否存在当前Partition的数据本地性的信息，如果有的话就直接返回；如果没有首先会调用rdd.getPreferedLocations

例如想让Spark运行咋Hbase上或者一种现在还没有直接支持的数据库上面，此时开发者需要自定义RDD，为了保证Task计算的数据本地性，最为关键的方式就是必须实现RDD的getPreferedLocations

1. DAGScheduler计算数据本地性的时候巧妙的借助了RDD自身的getPreferedLocations中的数据，最大化的优化了效率，因为getPreferedLocations中表明了每个Partition的数据本地性，虽然当前Partition可能被persist或者checkpoint，但是persist或者checkpoint默认情况下肯定是和getPreferedLocations中的Partition的数据本地性是一致的，所以这就极大的简化Task数据本地性算法的实现和效率的优化；