Spark调度算法

在上面的文档中，介绍了Spark Job如何构建Stage，拆分TaskSet，如下图所示：



在调度阶段，TaskScheduler收到发送过来的任务集TaskSet0和TaskSet1，然后通过submitTask方法提交任务：

*override def submitTasks(taskSet: TaskSet) {*

*val tasks = taskSet.tasks*

*this.synchronized {*

*//基于TaskSet构建TaskSetManager*

*val manager = createTaskSetManager(taskSet, maxTaskFailures)*

*…… //将TaskSetManager加入到SchedulableBuilder*

*schedulableBuilder.addTaskSetManager(manager, manager.taskSet.properties)*

*}*

代码中的SchedulableBuilder是调度的对象，目前支持的调度模式为FIFO和FAIR两种模式，这两种模式的配置参数为spark.scheduler.mode，默认FIFO。

schedulableBuilder对象在TaskSchedulerImpl类中的定义和实现如下所示：

*def initialize(backend: SchedulerBackend) {*

*this.backend = backend*

*schedulableBuilder = {*

*schedulingMode match {*

*case SchedulingMode.FIFO =>*

*new FIFOSchedulableBuilder(rootPool)*

*case SchedulingMode.FAIR =>*

*new FairSchedulableBuilder(rootPool, conf)*

*case \_ =>*

*throw new IllegalArgumentException(s"Unsupported $SCHEDULER\_MODE\_PROPERTY: " +*

*s"$schedulingMode")*

*}*

*}*

*schedulableBuilder.buildPools()*

*}*

根据用户的配置决定是生成FIFOSchedulableBuidler还是生成FairSchedulableBuilder类型的schedulableBuilder对象，类图如下所示：



在生成schedulableBuilder后，调用其buildPools方法生成调度池。rootPool中包含一组TaskSetManager，其成员变量如下：

*val schedulableQueue = new ConcurrentLinkedQueue[Schedulable]*

*val schedulableNameToSchedulable = new ConcurrentHashMap[String, Schedulable]*

*val weight = initWeight*

*val minShare = initMinShare*

*var runningTasks = 0*

*val priority = 0*

*var stageId = -1*

*val name = poolName*

*var parent: Pool = null*

*private val taskSetSchedulingAlgorithm: SchedulingAlgorithm*

Pool#addSchedulable方法:

*override def addSchedulable(schedulable: Schedulable) {*

*require(schedulable != null)*

*schedulableQueue.add(schedulable)*

*schedulableNameToSchedulable.put(schedulable.name, schedulable)*

*schedulable.parent = this*

*}*

将TaskSetManager加入调度队列schedulableQueue中，不同的schedulableBuilder的核心是SchedulingAlgorithm，如下所示：

*private val taskSetSchedulingAlgorithm: SchedulingAlgorithm = {*

*schedulingMode match {*

*case SchedulingMode.FAIR =>*

*new FairSchedulingAlgorithm()*

*case SchedulingMode.FIFO =>*

*new FIFOSchedulingAlgorithm()*

*case \_ =>*

*val msg = s"Unsupported scheduling mode: $schedulingMode. Use FAIR or FIFO instead."*

*throw new IllegalArgumentException(msg)*

*}}*

FIFO模式的算法类是FIFOSchedulingAlgorithm，FAIR模式的算法类是

FairSchedulingAlgorithm，核心方法是comparator。

FIFOSchedulingAlgorithm#comparator

*override def comparator(s1: Schedulable, s2: Schedulable): Boolean = {*

*val priority1 = s1.priority*

*val priority2 = s2.priority*

*var res = math.signum(priority1 - priority2)*

*if (res == 0) {*

*val stageId1 = s1.stageId*

*val stageId2 = s2.stageId*

*res = math.signum(stageId1 - stageId2)*

*}*

*res < 0*

*}*

两个Task任务，首先获得两个任务的priority，在FIFO的优先级实际上是Job ID，比较两个任务的Job ID，如果priority1比priority2小，那么返回true，表示s1的优先级比s2高。

FairSchedulingAlgorithm#comparator

*override def comparator(s1: Schedulable, s2: Schedulable): Boolean = {*

*val minShare1 = s1.minShare*

*val minShare2 = s2.minShare*

*val runningTasks1 = s1.runningTasks*

*val runningTasks2 = s2.runningTasks*

*val s1Needy = runningTasks1 < minShare1*

*val s2Needy = runningTasks2 < minShare2*

*val minShareRatio1 = runningTasks1.toDouble / math.max(minShare1, 1.0)*

*val minShareRatio2 = runningTasks2.toDouble / math.max(minShare2, 1.0)*

*val taskToWeightRatio1 = runningTasks1.toDouble / s1.weight.toDouble*

*val taskToWeightRatio2 = runningTasks2.toDouble / s2.weight.toDouble*

*var compare = 0*

*if (s1Needy && !s2Needy) {*

*return true*

*} else if (!s1Needy && s2Needy) {*

*return false*

*} else if (s1Needy && s2Needy) {*

*compare = minShareRatio1.compareTo(minShareRatio2)*

*} else {*

*compare = taskToWeightRatio1.compareTo(taskToWeightRatio2)*

*}*

*if (compare < 0) {*

*true*

*} else if (compare > 0) {*

*false*

*} else {*

*s1.name < s2.name*

*}*

*}*

minShare对应fairscheduler.xml配置文件中的minShare属性。

（1）如果s1所在Pool或者TaskSetManager中运行状态的task数量比minShare小，s2所在Pool或者TaskSetManager中运行状态的task数量比minShare大，那么s1会优先调度。反之，s2优先调度。

（2）如果s1和s2所在Pool或者TaskSetManager中运行状态的task数量都比各自minShare小，那么minShareRatio小的优先被调度。

minShareRatio是运行状态task数与minShare的比值，即相对来说minShare使用较少的先被调度。

（3）如果minShareRatio相同，那么最后比较各自Pool的名字。

其中TaskScheduler#resourceOffers方法获取资源后，

*def resourceOffers(offers: IndexedSeq[WorkerOffer]): Seq[Seq[TaskDescription]] = synchronized {*

*.....*

*val sortedTaskSets = rootPool.getSortedTaskSetQueue*

*for (taskSet <- sortedTaskSets) {*

*var launchedAnyTask = false*

*var launchedTaskAtCurrentMaxLocality = false*

*for (currentMaxLocality <- taskSet.myLocalityLevels) {*

*do {*

*launchedTaskAtCurrentMaxLocality = resourceOfferSingleTaskSet(*

*taskSet, currentMaxLocality, shuffledOffers, availableCpus, tasks)*

*launchedAnyTask |= launchedTaskAtCurrentMaxLocality*

*} while (launchedTaskAtCurrentMaxLocality)*

*}*

*if (!launchedAnyTask) {*

*taskSet.abortIfCompletelyBlacklisted(hostToExecutors) }*