##### Spark 调度策略:<https://yq.aliyun.com/articles/6041>,FIFO and FIAR ,

## Spark的调度策略详解

摘要： Spark的调度策略 Spark目前有两种调度策略，一种是FIFO即先来先得，另一种是FAIR即公平策略。所谓的调度策略就是对待调度的对象进行排序，按照优先级来进行调度。调度的排序接口如下所示，就是对两个可调度的对象进行比较。 private[spark] trait SchedulingAlgo...

# Spark的调度策略

Spark目前有两种调度策略，一种是FIFO即先来先得，另一种是FAIR即公平策略。所谓的调度策略就是对待调度的对象进行排序，按照优先级来进行调度。调度的排序接口如下所示，就是对两个可调度的对象进行比较。

private[spark] trait SchedulingAlgorithm {

def comparator(s1: Schedulable, s2: Schedulable): Boolean

}

其实现类为FIFOSchedulingAlgorithm、FairSchedulingAlgorithm

/\*\*

\* FIFO排序的实现，主要因素是优先级、其次是对应的Stage

\* 优先级高的在前面，优先级相同，则靠前的stage优先

\*/

private[spark] class FIFOSchedulingAlgorithm extends *SchedulingAlgorithm* {

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

//一般来说优先级越小优先级越高

val priority1 = s1.priority

val priority2 = s2.priority

var res = math.signum(priority1 - priority2)

if (res == 0) {

//如果优先级相同，那么Stage靠前的优先

val stageId1 = s1.stageId

val stageId2 = s2.stageId

res = math.signum(stageId1 - stageId2)

}

if (res < 0) {

true

} else {

false

}

}

}

**注：**  
可以根据自己对优先级的定义重写这个比较方法，但有一点注意，就是如果优先级和Stage都相同，那么默认后来居上

private[spark] class FairSchedulingAlgorithm extends *SchedulingAlgorithm* {

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

//最小共享,可以理解为执行需要的最小资源即CPU核数，其他相同时，所需最小核数小的优先调度

val minShare1 = s1.minShare

val minShare2 = s2.minShare

//运行的任务的数量

val runningTasks1 = s1.runningTasks

val runningTasks2 = s2.runningTasks

//是否有处于挨饿状态的任务，看可分配的核数是否少于任务数，如果资源不够用，那么处于挨饿状态

val s1Needy = runningTasks1 < minShare1

val s2Needy = runningTasks2 < minShare2

//最小资源占用比例，这里可以理解为偏向任务较轻的

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

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

//权重，任务数相同，权重高的优先

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

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

var compare: Int = 0

//挨饿的优先

if (s1Needy && !s2Needy) {

return true

} else if (!s1Needy && s2Needy) {

return false

} else if (s1Needy && s2Needy) {

//都处于挨饿状态则，需要资源占用比小 的优先

compare = minShareRatio1.compareTo(minShareRatio2)

} else {

//都不挨饿，则比较权重比，比例低的优先

compare = taskToWeightRatio1.compareTo(taskToWeightRatio2)

}

if (compare < 0) {

true

} else if (compare > 0) {

false

} else {

//如果都一样，那么比较名字,按照字母顺序比较，不考虑长度，所以名字比较重要

s1.name < s2.name

}

}

}

**注：**

1. 公平原则本着的原则就是谁最需要就给谁，所以挨饿者优先；
2. 资源占用比这块有点费解，如果把他理解成一个贪心问题就容易理解了。对于都是出于挨饿状态的任务可以这样理解，负载大的即时给你资源你也不一定能有效缓解，莫不如给负载小的，让其快速使用，完成后可以释放更多的资源，这是一种贪心策略。如JobA和JobB的Task数量相同都是10，A的minShare是2，B的是5，那占用比为5和2，显然B的占用比更小，贪心的策略应该给B先调度处理；
3. 对于都处于满足状态的，当然谁的权重有着更好的决定性，权重比低得优先（偏向权利大的）；
4. 如果所有上述的比较都相同，那么名字字典排序靠前的优先（哈哈，名字很重要哦）；名字aaa要比abc优先，所以这里在给Pool或者TaskSetManager起名字的时候要考虑这一点。

这两种调度的排序算法针对的可比较对象都是Schedule的具体对象，其(trait可理解成java中接口)定义如下：

private[spark] trait Schedulable {

//指明父对象，即这个Pool或TaskSetManager所属的调度对象，调度是层级的，是树状的

var parent: Pool

// 他拥有的调度对象，即负责管理的调度对象

def schedulableQueue: ConcurrentLinkedQueue[Schedulable]

//负责管理的对象间的排序模型，目前只有FIFO和FAIR两种算法

def schedulingMode: SchedulingMode

//权重，指的是和同级的相比的权重，权重越大获得的资源越多

def weight: Int

//最小共享值，指的是可运行需要的最小资源数，即CPU数量

def minShare: Int

def runningTasks: Int

//优先级，指的是在同级别中的优先级，优先级高的优先调度

def priority: Int

//这个stageId是对TaskSetManager而言，因为一个Stage的Tasks，实际以一个TaskSet提交

def stageId: Int

def name: String

def addSchedulable(schedulable: Schedulable): Unit

def removeSchedulable(schedulable: Schedulable): Unit

def getSchedulableByName(name: String): Schedulable

def executorLost(executorId: String, host: String): Unit

def checkSpeculatableTasks(): Boolean

def getSortedTaskSetQueue: ArrayBuffer[TaskSetManager]

}

目前Spark中有两种可调度的实体，Pool和TaskSetManager。Pool是一个调度池，Pool里面还可以有子Pool，Spark中的rootPool即根节点默认是一个无名的Pool。

/*\*\*\**TaskSchedulerImpl的初始化方法\*/

def initialize(backend: SchedulerBackend) {

this.backend = backend

// temporarily set rootPool name to empty

rootPool = new Pool("", schedulingMode, 0, 0)

schedulableBuilder = {

schedulingMode match {

case SchedulingMode.FIFO =>

new FIFOSchedulableBuilder(rootPool)

case SchedulingMode.FAIR =>

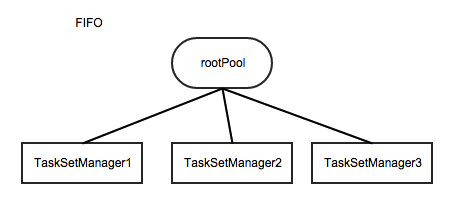
new FairSchedulableBuilder(rootPool, conf)

}

}

schedulableBuilder.buildPools()

}

其实对于FIFO模式的调度，rootPool管理的直接就是TaskSetManager,没有子Pool这个概念，就只有两层，rootPool和叶子节点TaskSetManager，实现如下所示。  
[](http://img3.tbcdn.cn/L1/461/1/68df446720adb44acc48411d61a2183a3035dbd7)

/\*\*\*FIFO模式下的Pools的构建\*/

private[spark] class FIFOSchedulableBuilder(val rootPool: Pool)

extends SchedulableBuilder with Logging {

override def buildPools() {

// 实际什么都不做

}

//添加下级调度实体的时候，直接添加到rootPool

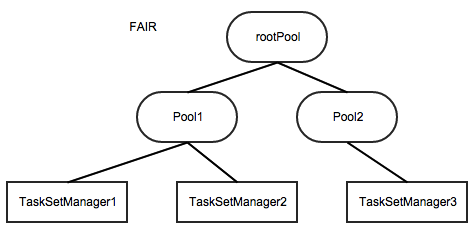
override def addTaskSetManager(manager: Schedulable, properties: Properties) {

rootPool.addSchedulable(manager)

}

}

但对于FAIR这种模式来说，是三层的，根节点是rootPool,为无名Pool,下一层为用户定义的Pool（不指定名称默认名称为default），再下一层才是TaskSetManager,即根调度池管理一组调度池，每个调度池管理自己的TaskSetManager，其实现如下所示。

[](http://img3.tbcdn.cn/L1/461/1/4c1ffd18add463205756c803871e918a6ef871e2)

/\*\*FAIR模式下的Pools的构建\*/

private[spark] class FairSchedulableBuilder(val rootPool: Pool, conf: SparkConf)

extends SchedulableBuilder with Logging {

....省略代码...

override def buildPools() {

...省略...

buildDefaultPool()

}

private def buildDefaultPool() {

if (rootPool.getSchedulableByName(DEFAULT\_POOL\_NAME) == null) {

val pool = new Pool(DEFAULT\_POOL\_NAME, DEFAULT\_SCHEDULING\_MODE,

DEFAULT\_MINIMUM\_SHARE, DEFAULT\_WEIGHT)

rootPool.addSchedulable(pool)

......

}

}

这里的调度顺序是指在一个SparkContext之内的调度，一般情况下我们自行使用是不太会需要Pool这个概念的，因为不存在Pool之间的竞争，但如果我们提供一个Spark应用，大家都可以提交任务，服务端有一个常驻的任务，对应一个SparkContext，每个用户提交的任务都由其代理执行，那么针对每个用户提交的任务可以按照用户等级和任务优先级设置一个Pool，这样不同的用户的Pool之间就存在竞争关系了，可以用Pool的优先级来区分任务和用户的优先级了，\*\*但要再强调一点名字很重要，因为FAIR机制中，如果其他比较无法判断，那么会按照名字来进行字典排序的\*\*。