Google

Spark调度（一）：Task调度算法，FIFO还是FAIR

by 伊布

July 11, 2016 in Tech

Spark调度分几个层次?

两层。

第一层，Spark应用间：Spark提交作业到YARN上，由YARN来调度各作业间的关系，可以配置YARN的调度策略为FAIR或者FIFO。

这一层可以再分两层，第一小层是YARN的队列，第二小层是队列内的调度。Spark作业提交到不同的队列，通过设置不同队列的minishare、weight等，来实现不同作业调度的优先级，这一点Spark应用跟其他跑在YARN上的应用并无二致，统一由YARN公平调度。比较好的做法是每个用户单独一个队列，这种配置FAIR调度就是针对用户的了，可以防止恶意用户提交大量作业导致拖垮所有人的问题。这个配置在hadoop的yarn-site.xml里。

```
    <property>
        <name>yarn.resourcemanager.scheduler.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>
    </property>
```

第二层，Spark应用内（同一个sparkContext），可以配置一个应用内的多个TaskSetManager间调度为FIFO还是FAIR。以Spark的Thrift Server为例，考虑一个问题，用户a的作业很大，需要处理 TOP 数据，且SQL也非常复杂，而用户b的作业很简单，可能只是Select查看前面几条数据而已。由于用户a、b都在

同一个SparkContext里，所以其调度完全由Spark决定；如果按先入先出的原则，可能用户b要等好一会，才能从用户a的牙缝里扣出一点计算资源完成自己的这个作业，这样对用户b就不是那么友好了。

比较好的做法是配置Spark应用内各个TaskSetManager的调度算法为Fair，不需要等待用户a的资源，用户b的作业可以尽快得到执行。这里需要注意，FIFO并不是说用户b只能等待用户a所有Task执行完毕才能执行，而只是执行的很迟，并且不可预料。从实测情况来看，配置为FIFO，用户b完成时间不一定，一般是4-6s左右；而配置为FAIR，用户b完成时间几乎是不变的，几百毫秒。

应用内调度的配置项在{spark_base_dir}/conf/spark_default.conf：`spark.scheduler.mode FAIR`。

这一层也可以再分两层，第一小层是Pool（资源池）间的公平调度，第二小层是Pool内的。注意哦，**Pool内部调度默认是FIFO的**，需要设置{spark_base_dir}/conf/fairscheduler.xml，针对具体的Pool设置调度规则。所以前面说TaskSetManager间调度不准确，应该是先Pool间，再Pool内。

下面配置default队列为FAIR调度。

```
<allocations>
  <pool name="default">
    <schedulingMode>FAIR</schedulingMode>
    <weight>1</weight>
    <minShare>3</minShare>
  </pool>
</allocations>
```

其中：

- weight：控制资源池相对其他资源池，可以分配到资源的比例。默认所有资源池的weight都是1。如果你将某个资源池的weight设为2，那么该资源池中的资源将是其他池子的2倍。如果将weight设得很高，如1000，可以实现资源池之间的调度优先级 – 也就是说，weight=1000的资源池总能立即启动其对应的作业。这个算法在后面代码里有体现。
- minShare：除了整体weight之外，每个资源池还能指定一个最小资源分配值（CPU个数），管理员可能会需要这个设置。公平调度器总是会尝试优先满足所有活跃（active）资源池的最小资源分配值，然后再根据各个池子的weight来分配剩下的资源。因此，minShare属性能够确保每个资源池都能至少获得一定量的集群资源。minShare的默认值是0。

综上，如果你想要thriftserver达到SQL级别的公平调度，需要配置三个配置文件：yarn-site.xml、spark-defaults.conf、fairscheduler.xml。由于thriftserver的SQL没有按照不同用户区分多个Pool，所以其实还不算特别完整。

代码上怎么体现Task调度是FIFO还是FAIR?

TOP

应用内调度配置为FAIR后，Spark的TaskSchedule会进入FairSchedulableBuilder模式，之后创建的Pool的作业调度算法taskSetSchedulingAlgorithm为FairSchedulingAlgorithm。这个算法会影响YARN/Mesos分配作业给Excutor时调用的makeOffer的任务队列排序，从而影响Pool之间、Pool内部调度排序。

排序时，先pool之间排序，再递归pool内部排序(这里可能有错误)。

（Spark应用可以指定不存在的pool，spark调度会新建并加到pool链表上）

先来看TaskSchedulerImpl.scala，从rootPool开始，getSortedTaskSetQueue启动排序。

```
    val sortedTaskSets = rootPool.getSortedTaskSetQueue
    for (taskSet <- sortedTaskSets) {
      logDebug("parentName: %s, name: %s, runningTasks: %s".format(
        taskSet.parent.name, taskSet.name, taskSet.runningTasks))
      if (newExecAvail) {
        taskSet.executorAdded()
      }
    }
```

从上面代码可以明确看到，TaskScheduler的对象实际是TaskSet。继续，Pool.scala一层层递归排序，拉为一个sortedTaskSetQueue。注意先Pool这一层的FAIR调度，再Pool内的FAIR调度。

```
  override def getSortedTaskSetQueue: ArrayBuffer[TaskSetManager] = {
    var sortedTaskSetQueue = new ArrayBuffer[TaskSetManager]
    val sortedSchedulableQueue =
      schedulableQueue.asScala.toSeq.sortWith(taskSetSchedulingAlgorithm.comparator)
    for (schedulable <- sortedSchedulableQueue) {
      sortedTaskSetQueue ++= schedulable.getSortedTaskSetQueue
    }
    sortedTaskSetQueue
  }
```

等sortedTaskSets齐活，下面依次launch：

```
    for (taskSet <- sortedTaskSets; maxLocality <- taskSet.myLocalityLevels) {
      do {
        launchedTask = resourceOfferSingleTaskSet(
            taskSet, maxLocality, shuffledOffers, availableCpus, tasks)
      } while (launchedTask)
    }
```

resourceOfferSingleTaskSet会遍历随机重排的计算资源，将TaskSet里的多个Task分配到各个Host的Excutor上去。这段细节比较多，后面慢慢说。

TOP

Poll间、Pool内排序时的sortWith是怎么比较的?

简而言之，FIFOSchedulingAlgorithm只是比较job ID和stage ID，而FairSchedulingAlgorithm则复杂一些，会根据请求task数、minshare、weight来综合判断优先级。下面结合代码来看，比较简单其实。

FIFO算法。priority实际就是JOB ID，特别简单，先来后到。

```
private[spark] class FIFOSchedulingAlgorithm extends SchedulingAlgorithm {
  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)
    }
    if (res < 0) {
      true
    } else {
      false
    }
  }
}
```

FAIR算法。重点是“公平”。

```
private[spark] class FairSchedulingAlgorithm extends SchedulingAlgorithm {
  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).toDouble
    val minShareRatio2 = runningTasks2.toDouble / math.max(minShare2, 1.0).toDouble
    val taskToWeightRatio1 = runningTasks1.toDouble / s1.weight.toDouble
    val taskToWeightRatio2 = runningTasks2.toDouble / s2.weight.toDouble
    var compare: Int = 0
```

如果一个Pool的miniShare够用，另一个不够用，先分配给够用的。

```
    if (s1Needy && !s2Needy) {
      return true
    } else if (!s1Needy && s2Needy) {
      return false
```

TOP

两个poll都够用，谁占miniShare的少先分配给谁。例如两个Pool同样数量的runningTask，先分配给miniShare大的。

```
    } else if (s1Needy && s2Needy) {
      compare = minShareRatio1.compareTo(minShareRatio2)
```

然后比较权重，同样数量的runningTask，先分配给weight大的。

```
    } else {
      compare = taskToWeightRatio1.compareTo(taskToWeightRatio2)
    }
```

实在不行比较名字。

```
    if (compare < 0) {
      true
    } else if (compare > 0) {
      false
    } else {
      s1.name < s2.name
    }
  }
}
```

BTW：注意hive-site.xml里不要设置hive.server2.map.fair.scheduler.queue为true，否则会造成thrift server尝试加载FairSchedulerShim报ClassNotFoundException。Thrift server的公平调度是在Spark上做的，不需要hive上再加一层。

```
16/07/04 19:35:33 INFO ThriftCLIService: Client protocol version:
HIVE_CLI_SERVICE_PROTOCOL_V8
16/07/04 19:35:33 WARN ThriftCLIService: Error opening session:
java.lang.RuntimeException: Could not load shims in class
org.apache.hadoop.hive.schshim.FairSchedulerShim
        at org.apache.hadoop.hive.shims.ShimLoader.createShim(ShimLoader.java:149)
        at org.apache.hadoop.hive.shims.ShimLoader.getSchedulerShims(ShimLoader.java:133)
...
Caused by: java.lang.ClassNotFoundException:
org.apache.hadoop.hive.schshim.FairSchedulerShim
        at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
...
```

TOP

Related posts