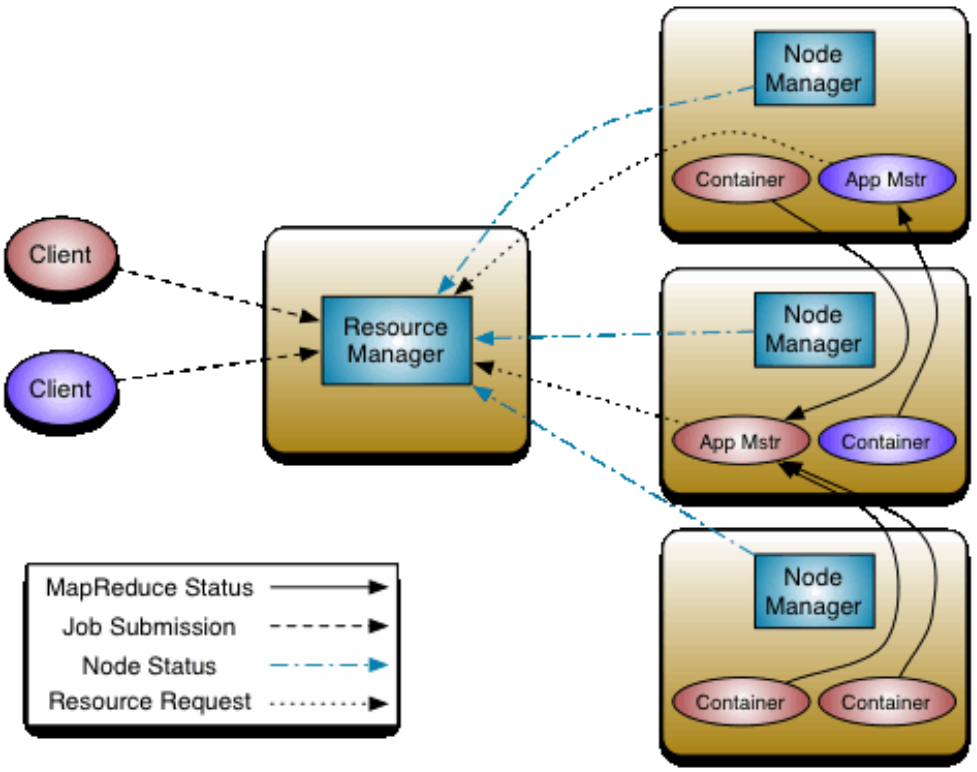
2018-03-20 15:13:31 星期二

架构

Yarn相比于hadoop1的最根本变化：分离**资源管理** 和 ***Job*监控调度**到不同的进程中。
资源管理：通过一个全局的ResourceManager服务实现和每个机器上的NodeManager实现；
***Job*监控调度**：每个任务分配一个ApplicationMaster进程实现；



上面图中，有以下几点：

1. RM和若干NM构成yarn的整个资源管理框架，NM可以认为是整个RM在各个node的Agent，负责监控以及上报containers的资源使用率；
2. ApplicationMaster是一组特定框架的Library，向RM请求特定的资源，并且启动指定的Task。
3. ResourceManager包括下面两个主要部分：
    - Scheduler:可插拔的资源调度线程，负责响应ApplicationMaster的请求，为Container分配资源
    - ApplicationsManager：负责接受job-submissions，之后在指定节点启动一个特定的ApplicationMaster进程，单启动失败时负责重启。

调度器

1. yarn的调度信息通过 `ResourceManager_IP:8088/cluster/scheduler` 可以在web页面看到。
2. 通过 `mapreduce.job.queuename` 可以指定应用提交的队列
3. Cloudera页面下有这几种调度器（默认是FairScheduler）：

52153514450

参考：http://www.mamicode.com/info-detail-1097801.html

调度器比较

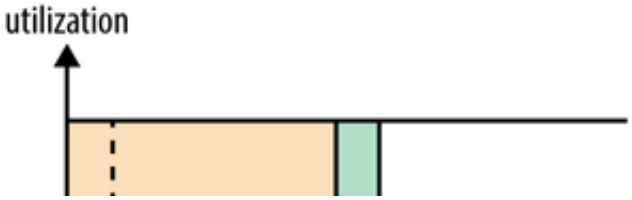
hadoop默认的资源调度器，根据提交顺序（FIFO）分配资源。

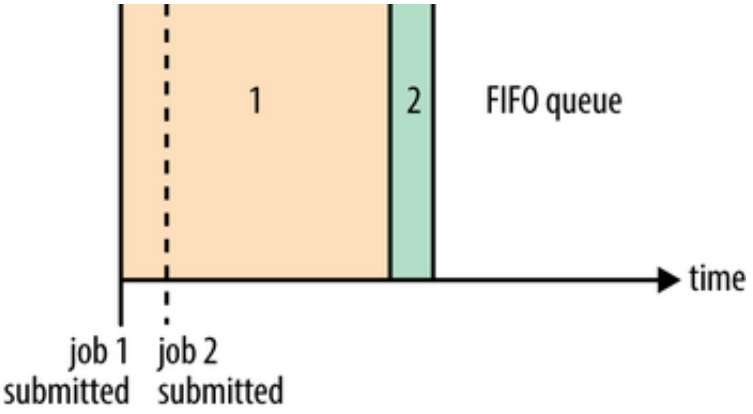
基于FIFO的单用户调度机制不能很好的利用集群资源（主要原因是：不同的作业类型对资源要求不一致）。大的应用可能会占用所有集群资源，这就导致其它应用被阻塞，导致小应用长时间饥饿。

下图是三个调度器的对比：

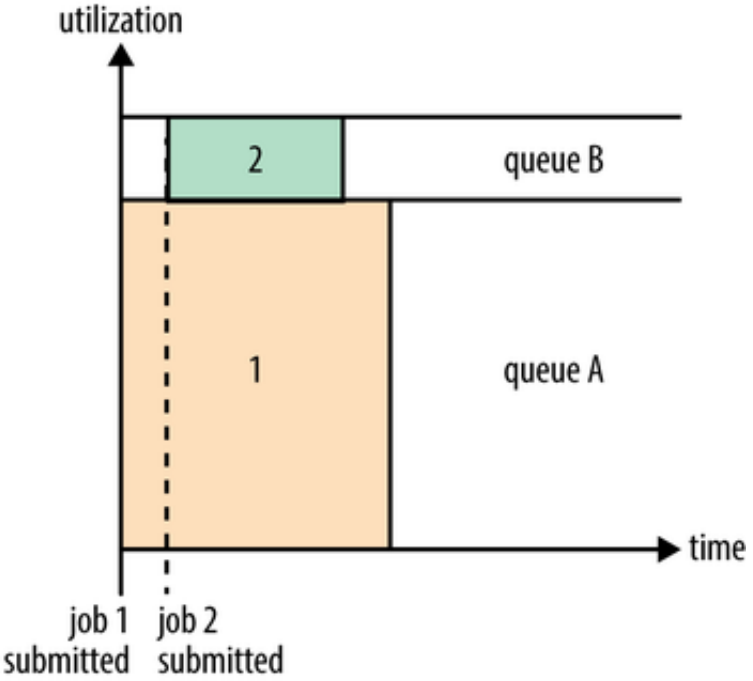
1. FIFO Scheduler中job1会完全占用资源，造成job2阻塞；
2. Capactiy Scheduler有一个专门的队列用来运行小任务，但是为小任务专门设置一个队列会预先占用一定的集群资源，这就导致大任务的执行时间会落后于使用FIFO调度器时的时间。
3. 在Fair调度器中，我们不需要预先占用一定的系统资源，Fair调度器会为所有运行的job动态的调整系统资源。如下图所示，当第一个大job提交时，只有这一个job在运行，此时它获得了所有集群资源；当第二个小任务提交后，Fair调度器会分配一半资源给这个小任务，让这两个任务公平的共享集群资源。需要注意的是，在下图Fair调度器中，从第二个任务提交到获得资源会有一定的延迟，因为它需要等待第一个任务释放占用的Container。小任务执行完成之后也会释放自己占用的资源，大任务又获得了全部的系统资源。最终的效果就是Fair调度器即得到了高的资源利用率又能保证小任务及时完成。

i. FIFO Scheduler

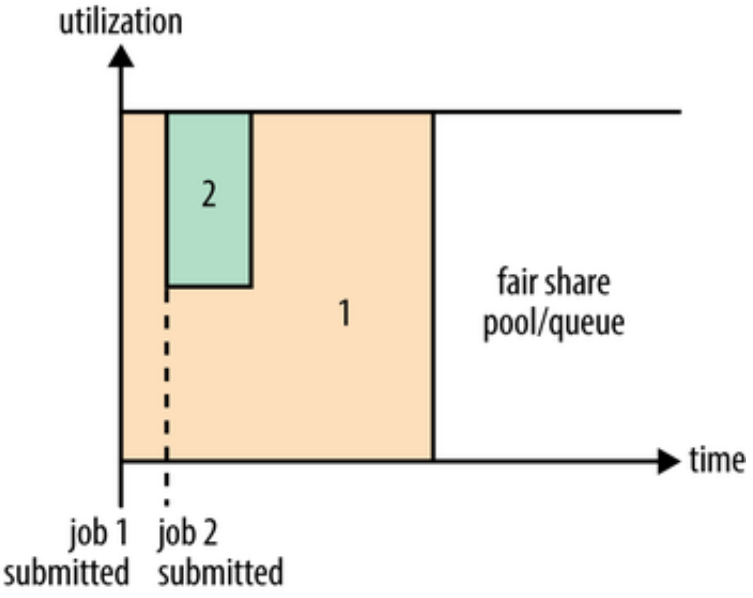




ii. Capacity Scheduler



iii. Fair Scheduler



CapacityScheduler

CapacityScheduler是一个基于**Hierarchical Queues**来管理集群的资源。CapacityScheduler会预定义*root*队列作为最底层的队列层级，系统中的所有队列都是根队列的子项。

通过为每个组织分配专门的队列，然后再为每个队列分配一定的集群资源，这样整个集群就可以通过设置多个队列的方式给多个组织提供服务了。除此之外，队列内部又可以垂直划分，这样一个组织内部的多个成员就可以共享这个队列资源了，在一个队列内部，资源的调度是采用的是先进先出(FIFO)策略。

CapacityScheduler允许配置**“弹性队列”(queue elasticity)**，即当队列内的资源不足时，获取其他队列释放的资源，分配给Container使用。在正常的操作中，Capacity调度器不会强制释放Container，当一个队列资源不够用时，这个队列只能获得其它队列释放后的Container资源。当然，我们可以为队列设置一个最大资源使用量，以免这个队列过多的占用空闲资源，导致其它队列无法使用这些空闲资源，这就是”弹性队列”需要权衡的地方。

原生Hadoop中通过etc/hadoop/capacity-scheduler.xml配置队列层级，CDH可以通过`容量调度程序配置高级配置代码段`来定义队列层级。通过`yarn rmadmin -refreshQueues`命令可以动态刷新queue的配置。

总结：

1. 管理员可以为每个队列分配资源的**软限制**、以及**可选的硬限制**（硬限制主要针对的是弹性队列）
2. 每个队列都有严格的的ACL控制列表
3. 允许“弹性队列”(queue elasticity)
4. 在单个队列内部遵循FIFO原则
5. 管理员可以在运行时停止队列，或添加新队列

在负载可预期的条件下，Capacity效果比较好。建议为每个队列分配的最小容量小于最高预期负载。

```
<!--
下面定义的队列层级，包括：
root
a----------b----------c
 a1--a2     b1--b2--b3
-->
```

```
<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>a,b,c</value>
  <description>The queues at the this level (root is the root queue).
  </description>
</property>

<property>
  <name>yarn.scheduler.capacity.root.a.queues</name>
  <value>a1,a2</value>
  <description>The queues at the this level (root is the root queue).
  </description>
</property>

<property>
  <name>yarn.scheduler.capacity.root.b.queues</name>
  <value>b1,b2,b3</value>
  <description>The queues at the this level (root is the root queue).
  </description>
</property>
```

1. queue名字通过来确定，比如上面的队列a1，可以表示为root.a.a1;
2. queue的一些属性
    - yarn.scheduler.capacity..capacity：队列的资源占比，同一个层级的queue，该值之和必须是100。由于capacity调度器允许资源弹性分配，因此队列中job占用资源的实际百分比可能大于这个值。
    - yarn.scheduler.capacity..maximum-capacity：队列的资源使用上限（百分比），这个值可以比上面的值大（超出部分是共享资源 ，这个配置是为弹性队列配置的）。
    - yarn.scheduler.capacity..minimum-user-limit-percent：一个队列中，每个用户最低资源保障。也就是说，无论当前有多少个用户在使用这个队列，他分配到资源都不会小于这个值（这个配置项的描述官方和网上的一些说法有出入）。若设为100，则不做任何用户限制
    - yarn.scheduler.capacity..user-limit-factor：用户可申请的队列资源比例，默认是1，表示不能超配申请
    - yarn.scheduler.capacity..maximum-allocation-mb：单个container 可以申请

的内存大小，覆盖yarn.scheduler.maximum-allocation-mb的值，必须小于集群总内存大小

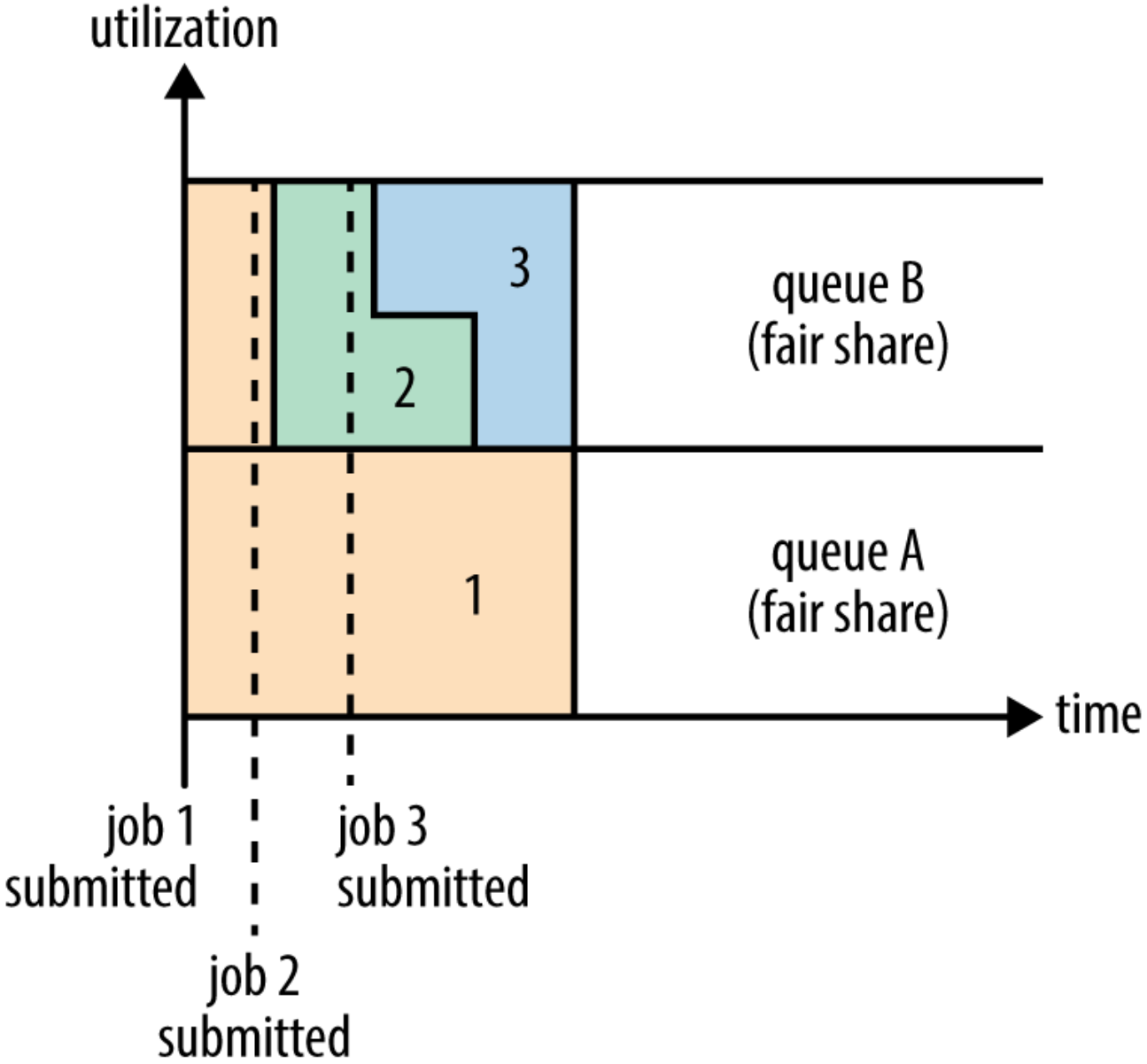
- yarn.scheduler.capacity..maximum-allocation-vcores：同上，覆盖yarn.scheduler.maximum-allocation-vcores
- yarn.scheduler.capacity.maximum-applications / yarn.scheduler.capacity..maximum-applications：队列里处于等待和运行的APP数目总和的上限，达到该上限是不能提交JOB。前者是全局配置，后者覆盖前者。
- yarn.scheduler.capacity.maximum-am-resource-percent / yarn.scheduler.capacity..maximum-am-resource-percent：集群中用于运行应用程序ApplicationMaster的资源比例上限。其实和上面差不多，一个从资源限制，一个从个数限制。
- yarn.scheduler.capacity..state：队列状态，STOPPED状态时，新的applications无法提交，运行中的applications可以正常执行完毕。将这个状态设定为STOPPED可以将queue退出。
- yarn.scheduler.capacity.root..acl_submit_applications：队列的ACL用户控制，可以遗传到子队列（例如：“user1, user2 group1,group2“）。
- yarn.scheduler.capacity.root..acl_administer_queue：队列超级用户设定，该管理员可控制该队列的所有应用程序，比如杀死任意一个应用程序等
- yarn.scheduler.capacity.queue-mappings/yarn.scheduler.capacity.queue-mappings
  -override.enable：queue的用户和用户组映射？？？没搞懂干什么用的。
- yarn.scheduler.capacity.resource-calculator：资源比较器设置：默认的是比较内存。Capacity Scheduler有两种比较器用以比较两个资源的大小，默认是DefaultResourceCalculator，它只考虑内存资源。另外一种是DominantResourceCalculator，它采用了DRF比较算法，同时考虑内存和CPU两种资源。
- yarn.scheduler.capacity.node-locality-delay：当调度次数小于本地延迟调度次数的时候不接受机架调度。（没搞懂有什么用）

FairScheduler

公平原则：当有一个应用程序在运行时，应用程序可以请求整个群集资源。当提交其他应用程序时，释放资源分配给新应用程序，以便每个应用程序最终获得大致相同的

资源量。--- 向上面图中那样。

公平调度可以适用于队列内部，也可以用在队列之间。如下图：



总结：

1. 公平调度中优先级被用作**权重**来确定每个应用应该获得的总资源的比例
2. 支持资源强占（抢占就是允许调度器杀掉占用超过其应占份额资源队列的containers，这些containers资源便可**被分配到应该享有这些份额资源的队列中。**需要注意抢占会降低集群的执行效率，因为被终止的containers需要被重新执行。）
3. 公平调度可以让短应用程序在合理的时间内完成，而不会长时间得不到系统资
4. FairScheduler和CapacityScheduler一样，能指定最小资源余量。
5. 公平调度器在队列内部，可以用一下三种分配策略FifoPolicy，FairSharePolicy，DominantResourceFairnessPolicy

6. 通过yarn application -movetoqueue appID -queue targetQueueName，可以将app在队列间动态的移动

关于公平调度的配置：

1. FairScheduler的部分公有配置定义在yarn-site.xml，其他queue相关、队列相关的配置定义在yarn.scheduler.fair.allocation.file文件中（支持热生效，该配置文件10刷新一次）；
2. yarn-site.xml中的配置项
    - yarn.scheduler.fair.allocation.file：含义见上面
    - yarn.scheduler.fair.user-as-default-queue：将用户名作为默认的queue名，否则默认queue是default（当用户不指定queue时，分配到自己的队列中）
    - yarn.scheduler.fair.preemption：是否配置用户抢占
    - yarn.scheduler.fair.preemption.cluster-utilization-threshold：启动抢占后，资源利用率的最大比率。
    - yarn.scheduler.fair.sizebasedweight：平均分配的模式改为按权分配
    - yarn.scheduler.fair.assignmultiple\yarn.scheduler.fair.max.assign：即当一个节点出现了多个资源时，可以一次在这个节点分配多个Container。
    - yarn.scheduler.fair.allow-undeclared-pools：设置为true时，将使用默认设置创建在作业中指定但未明确配置的池。
    - yarn.scheduler.fair.update-interval-ms：默认值500ms，锁住调度器重新进行计算作业所需资源的间隔
    - yarn.scheduler.fair.locality.threshold.node/yarn.scheduler.fair.locality.threshold.rack：资源本地配置的选项
    - yarn.scheduler.increment-allocation-mb/yarn.scheduler.increment-allocation-vcores：资源化整单位
    - 调度器则默认采用如下规则：除非队列被准确的定义，否则会以用户名为队列名创建队列。
    - 设置`yarn.scheduler.fair.user-as-default-queue=false`，这样应用便会被放入default 队列，而不是各个用户名队列。
3. `yarn.scheduler.fair.allocation.file`文件配置

```
<?xml version="1.0"?>
```

```
<allocations>
  <queue name="sample_queue">
    <!--
    最少资源保证量。当一个队列的最少资源保证量未满足时，它将优先于其他同级队列获得
    对于不同的调度策略，最少资源保证量的含义不同：
    1. 对于fair策略，则只考虑内存资源；
    2. 对于drf策略，则考虑主资源使用的资源量；
    -->
    <minResources>10000 mb,0vcores</minResources>
    <!--
    最多可以使用的资源量，fair scheduler会保证每个队列使用的资源量不会超过该队列
    -->
    <maxResources>90000 mb,0vcores</maxResources>
    <!-- 可运行的APP数目 -->
    <maxRunningApps>50</maxRunningApps>
    <!--
    限制用于运行作业的共享资源，该属性只能配置在叶子队列，设置成1.0f，am在该队列
    -->
    <maxAMShare>0.1</maxAMShare>
    <!-- 该队列在同层级中的权重 -->
    <weight>2.0</weight>
    <!-- 队列内的调度策略，支持fifo、fair、drf -->
    <schedulingPolicy>fair</schedulingPolicy>
    <!-- 该队列的子队列 -->
    <queue name="sample_sub_queue">
      <!-- 只允许该用户提交 -->
      <aclSubmitApps>charlie</aclSubmitApps>
      <minResources>5000 mb,0vcores</minResources>
    </queue>

    <!-- 关于资源抢占的三个配置 -->
    <minSharePreemptionTimeout>当queue资源超过该时间，不能到达minResources定
    <fairSharePreemptionTimeout>当queue资源超过该时间，不能到达fairSharePree
    <fairSharePreemptionThreshold></fairSharePreemptionThreshold>

  </queue>
  <!-- 同maxAMShare配置，被其覆盖 -->
  <queueMaxAMShareDefault>0.5</queueMaxAMShareDefault>

  <!-- Queue 'secondary_group_queue' is a parent queue and may have user q
```

```
  <!-- 这个父队列下没有leaf队列，可能用户在提交job时会自动创建一些队列在该队列下
  <queue name="secondary_group_queue" type="parent">
    <weight>3.0</weight>
  </queue>
  <!-- user下只能配置maxRunningApps，限制用户运行的APP数目 -->
  <user name="sample_user">
    <maxRunningApps>30</maxRunningApps>
  </user>
  <userMaxAppsDefault>5</userMaxAppsDefault>
  <!-- queuePlacementPolicy包含一组规则元素，用于告知调度程序如何将传入应用程序
  <!--
  rule按照定义顺序加载
  -->
  <queuePlacementPolicy>
    <!-- 如果job指定了queuename，那么提交到指定的queue -->
    <!-- 如果指定的名称不存在，会直接被拒绝 -->
    <!-- 如果默认没有指定queuename，那么queuename被认为是default -->
    <rule name="specified" />
    <!-- 将job放入用户的username的queue中 -->
    <rule name="user" />
    <!-- 将job放入用户的primary group的queue中 -->
    <!-- create指示是否可以创建该queue，默认是可以创建的 -->
    <rule name="primaryGroup" create="false" />
    <!-- 和user参数类似，不同点：该配置在任何parent queue中都能创建队列，user配
    <!--  -->
    <rule name="nestedUserQueue">
        <!-- 将job放入用户的second group的queue中，有多个second group时选择第
        <rule name="secondaryGroupExistingQueue" create="false" />
    </rule>
    <!-- 放入默认队列中，这里默认队列是sample_queue，否者时root.default -->
    <rule name="default" queue="sample_queue"/>
  </queuePlacementPolicy>
</allocations>

```