Large-Scale YARN Cluster

Hadoop是Apache 开源的分布式系统基础架构，使用集群进行分布式计算和存储，集群规模可以从单一服务器扩展上千服务器。Hadoop的核心包括：HDFS分布式文件系统，存储超过规模数据集；YARN, 资源管理器，负责对整个集群的资源(cpu，内存等)进行分配和调度，分配以资源container的形式分发到各个应用程序中，应用程序与资源所在节点的NodeManager协作启动Container来完成具体的任务。

# 引言

## 1.1 YARN基本概念

YARN在硬件和计算框架之间提供一个抽象层，其核心功能包括：资源抽象、资源管理（包括调度、使用、监控和隔离等）。几个基本概念如下：

1）资源：通常是指硬件资源，包括CPU、内存、磁盘、IO及网络流量等。

2）Container：RM进行资源分配的基本单位，Container包含特定数量的CPU和内存资源。用户程序运行在Container中。

3） 资源调度器与队列：资源调度器是可插拔组件，常见的有FIFO,CapacityScheduler及FairScheduler。将所有的资源分成一个或者多个队列，每个队列包含一定量的资源。用户程序会唯一的分配到一个队列中执行。

4）事件驱动，YARN实现基于状态机的事件驱动机制，对象内部有预先定义好的有限状态机，相应事件会触发状态转换。事件驱动的角色：Dispatcher,Event及Handler。

YARN资源管理器实际上是一个事件处理器，其处理来自外部的SchedulerEvent类型事件，并根据事件的具体含义进行相应处理，如下图所示：



图1:YARN资源管理器

NodeManager通过心跳机制向RM汇报本节点的资源情况，触发NODE\_UPDATE事件，当NM上的Container运行结束会释放资源，YARN RM会将这部分可用资源分配给应用程序。该事件触发了资源调度器最核心的模块资源分配，资源分配由ResourceScheduler来完成。

## 1.2.资源调度器（ResourceScheduler）

资源调度器的两个主要作用：划分队列和分配资源。在YARN上调度器是可插拔组件，常见的有FIFO,CapacityScheduler和FairScheduler，可以通过配置文件选择不同的调度器。FIFO 是YARN默认的调度器，只有一个对了，所有用户共享，一般不能用于生产环境；Capacity Scheduler基本思想是每个用户都可以使用特定量的资源，但是集群空闲时，也可以使用整个集群的资源，其在FIFO基础上，增加了多用户。FairScheduler的设计目标是为所有的应用分配公平（通过参数来设置）的资源，核心思想是按照used\_memory/minShare大小调度，按照该调度算法决定调度顺序

在RM端，根据不同的调度器将所有的资源分成一个或者多个队列，每个队列包含一定量的资源。用户的每个应用，会被唯一的分配到一个队列中去执行，队列决定了用户的使用资源上限。

在YARN中，用户以队列的形式组织资源，每个队列被划分一定比例或者固定大小的资源，每个用户可以属于一个或者多个队列，且只能向这些队列中提交应用。队列的组织是资源管理和分配的基础，当前YARN使用层次队列组织方式，如下图：



图2：层级队列

1. 队列可以嵌套，每个队列均可以包含子队列，用户只能将作业提交到最底层队列，即叶子队列
2. 每个子队列均有一个“最少容量比”属性，表示可以使用父队列容量的百分比；调度器优先选择当前资源使用率最低的队列，并为之分配资源；最少容量不是总会保证的最低容量，例如一个队列最少容量是20，但是该队列仅使用5，那么剩下的15资源可能会分配给其他需要的队列
3. 最大容量，为了防止队列超过使用资源，可以为队列设置一个最大容量，这是一个资源使用上限；默认情况下最大容量无限大，例如当一个队列只分配了20%的资源，所有其他队列没有作业时，该队列可以使用100%的资源，当其他队列有作业提交时，再逐步归还。

## 1.3.资源分配过程

YARN的资源调度过程是异步的，资源调度器将资源分配给一个应用后，不会立刻push给对应的ApplicationMaster，而是暂时放到一个缓冲区中，等待AM通过周期性的RPC函数主动获取，即采用pull-based模型，过程如下所示：

1. 当应用程序被提交到ResourceManager上，RM会向资源调度器发送一个APP\_ADDED事件，资源调度器收到该事件后，会为应用程序创建一个FSAppAttempt对象跟踪和维护该应用程序进行一系列合法性检查，然后应用等待资源调度器的资源调度。
2. RM启动后，就不停的调用ResourceScheduler进行资源的分配，过程如下图：



图3：资源分配过程

步骤1）NodeManager通过周期性心跳汇报节点信息

步骤2）RM为NM返回一个心跳应答，包括释放Container列表等

步骤3）RM接收来自NM的信息后，触发NODE\_UPDATE事件

步骤4）ResourceScheduler接收到NODE\_UPDATE事件后，按照一定的策略将该节点的资源分配给各个应用程序，并将分配结果放到一个内存数据结构中。具体执行过程如下图：



图4：ResourceScheduler执行过程

* RM中的资源调度器，遍历各子队列（叶子队列），计算队列的资源使用率，根据优先级选择一个有资源请求的队列，
* 选择一个队列后，对提交到队列中的应用进行排序，选择对资源最紧急的应用程序
* 将分配的资源分配分配给该应用，ApplicationMaster使用这些资源启动任务

步骤5）ApplicationMaster向RM发送周期性心跳，以领取最新分配的Container

步骤6）RM接收到AM的心跳后，将为它分配的container以心跳的形式返回给AM

步骤7）AM收到新分配的Container列表后，进一步分配给它内部的Task

# YARN集群规模

## 2.1 YARN集群规模的影响因素

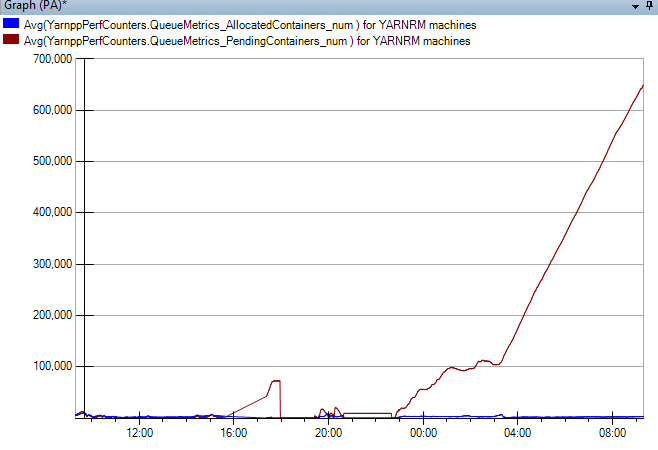
在第一部分图3和图4中分析了YARN资源管理的过程，影响YARN资源分配（YARN的主要功能）主要有以下过程:

1. NodeManager与ResourceManager之间的心跳，NM通过心跳向RM更新节点的资源使用情况，并通过NODE\_UPDATE事件触发RM的资源分配过程
2. 应用的ApplicationMaster与ResourceManager之间的心跳，AM通过心跳与RM交互，通过pull的方式来获取RM分配给本应用的资源
3. ResourceScheduler执行资源分配过程，RM根据NM汇总的资源进行资源的分配，其需要一定的计算：选择队列（队列的排序），选择队列中的应用（应用的排序）等过程。

RM处理心跳的能力及ResourceScheduler执行资源分配的能力是影响YARN集群规模的两个主要因素。当YARN集群规模扩大的挑战是Hadoop架构的单点问题，当集群规模增大后，NM及集群中运行的应用相应增加，这些进行均会与RM之间进行RPC交互，其请求数量会猛增，但是单节点的RM其最大处理请求数（QPS）是一定的，极易造成消息阻塞，从而造成请求处理严重超时，即RM处理心跳的能力。另外YARN集群的考虑因素是应用获取资源的响应速度，会影响集群Job的吞吐量，即ResourceScheduler的资源分配能力。

## 2.2 Microsoft集群规模测试结果及各厂商最大集群规模

在YARN集群中一般要求RM的QPS平均达到3000/秒，应用的Container的分配时间小于5秒，当集群规模达到一定成都后，container分配的延迟过高会造成Job失败，从而造成Job的大量积压。根据当前Apache 社区提供的信息，YARN集群的最大规模可以到达4000个节点，下图来自微软的测试结果，Hadoop版本为2.7,集群规模为3000节点，使用CapacityScheduler调度器：

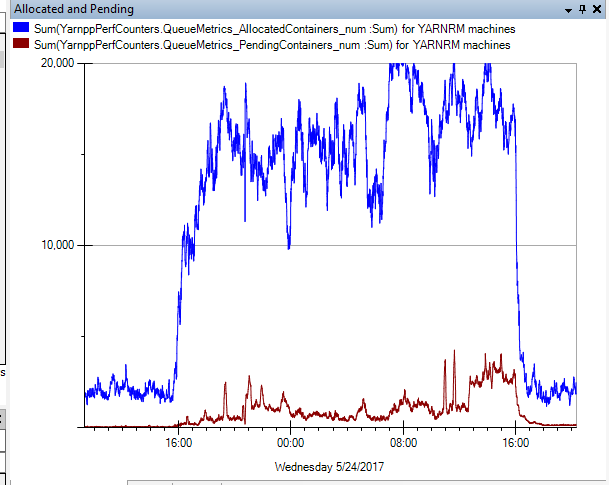


其中蓝线为allocated containers，红线为pending containers，从上图看当集群规模超过3000个节点，YARN RM就会瘫掉，从RM中获取Container的延迟变得无法接受。

在图3,4中，分析了AM获取资源的过程，在运行过程中AM向RM发送Container请求，RM收到后在pending queue中进行排队。当NM发送心跳中有资源释放，RM将资源分配给队列中满足要条件的请求。这个过程中，主要的问题是RM接受到每个心跳，都是遍历所有的outstanding request，即使这些请求不满足条件，这个Schedule循环很耗费时间；在心跳处理过程中，会不停的创建不可变的Resource对象，创建的过程也比较慢；同时Log4j是同步的；另外每次心跳仅分配一个Container。综上，资源调度器很多可以调优的地方：

* 每次node心跳时，调度仅考虑该node的分配请求，配置Scheduler Key来减少资源分配的时间。Scheduler key pruning
* 减少每次Node心跳的工作量，例如优化Resource对象的构建。同时Use time to decide if we should downgrade to rack or any where in the cluster。Time based decay for locality
* 使用async log4j appender，减少lock及IO连接

测试4000个集群节点，RM-NM和RM-AM的心跳设置1秒，使用log4j async logger，测试结果如下：



与优化前的对比结果，如下表：

|  |  |
| --- | --- |
| 阶段 | 结果 |
| 没有优化前，使用relaxLocality | * Allocation latency @ 95th%tile< **10s** * Promotion latency @ 95th%tile< **10s** * Node locality – <**10%** * ANY> locality – >**80%** * Sustained load < **500 QPS** |
| Scheduler key pruning | * Allocation latency @ 95th%tile< **4s** * Promotion latency @ 95th%tile< **4s** * Node locality – **99.51%** * Rack locality– **0.23%** * Sustained load < **2000 QPS** |
| Time based decay for locality | * Allocation latency @ 95th%tile< **3s** * Promotion latency @ 95th%tile< **3s** * Node locality – **99.84%** * Rack locality– **0.11%** * Sustained load < **3000 QPS** |

经过Scheduler Key Pruning及Time based decay for locality的优化以后，YARN集群规模可以达到4000个节点。

根据Hortonworks的经验，当前YARN集群的规模量级大概在4000~4500节点左右。经过调研，各厂商的最大YARN集群规模如下：

|  |  |
| --- | --- |
| 厂商 | YARN单集群最大规模 |
| Twitter | 4000节点 |
| 阿里 | 3000节点（2013年规模） |
| 腾讯 | 4400节点 |
| Hortonworks | 6000-8000节点 |
| Yahoo | 19个集群，一共4万个节点 |

# 后续的扩展性优化

以上的结果均是在Hadoop 2.7版本之前进行的测试及实践结果，YARN在不断优化及改进，尤其是在扩展性方面进行的优化工作比较多，主要分成两个方面：

## 3.1 YARN调度器本身的优化

主要优化调度算法（例如排序方面）和代码优化方面，在hadoop 2.8以后进行的调度器优化包括一下方面：

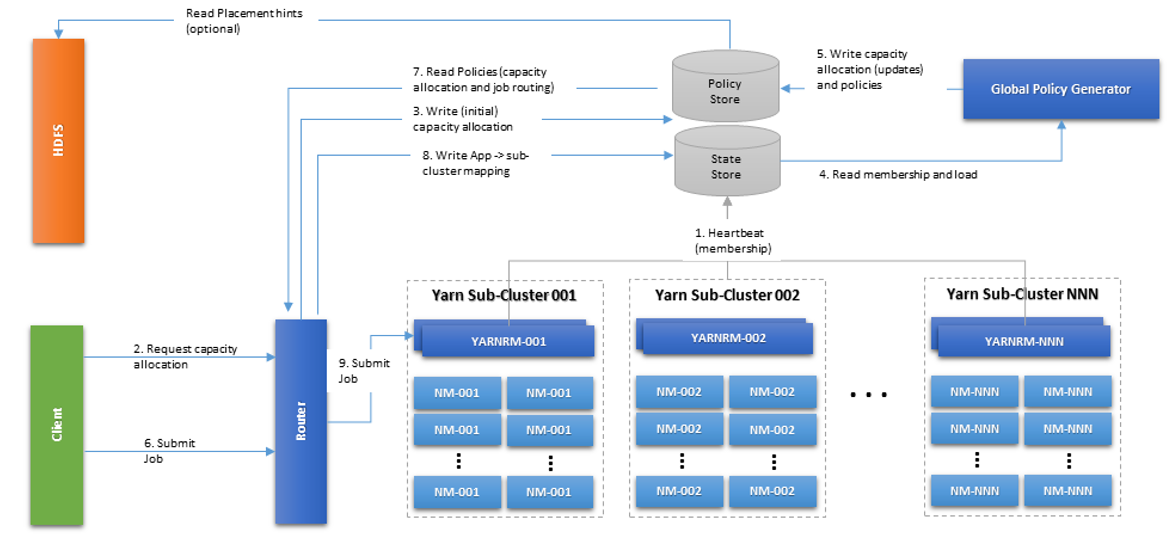
|  |  |  |
| --- | --- | --- |
| JIRA | 描述 | Fix Version |
| YARN-3547 | 没有资源请求的应用不参与调度，减少调度时app的处理数目 | 2.8.0，3.0 |
| YARN-6242 | Scheduler性能优化 | Umbrella |
| YARN-6775 | 在CS中，canAssignToUse|Queue等多次调用，可以先放到缓存中，减少调用次数 | 2.9.0,2.8.2，3.0 |
| YARN-4090 | 在FS中，Collection.sort消耗时间太长，通过缓存resource usage来提高效率 | Patch Avaiable |
| ARN-6361 | FSLeafQueue中的fetchAppsWithDemand的cpu使用率太高：该方法中会对应用进行排序，大部分时间花费在FairShareComparator#compare，该排序花费时间O(n\*log(n))。但是仅有该队列有10K以上的应用才会有显著影响。优化点在于降低排序时间O(n)：Noteupdate操作，提高了24倍，scheduling 吞吐量10倍 | Patch Avaiable |
| YARN-6242 | Resource Class的实例化花费时间太长，增加FinlResource对象 | 2.9.0, 3.0 |
| YARN-6680 | 对Label添加了读锁操作，其中hash操作比较消耗时间。因此无Label节点直接返回：在开启nodelabel3倍的slowdown | 2.9.0, 3.0 |
| YARN-6681 | 在canAssignToThisQueue中对child Queues进行了插入TreeSet ：占用时间由20% 变成 2% | 2.9.0, 3.0 |
| YARN-6682 | AssignmentInformation对象不合理，对GC不友好，例如有3个hashma中仅两个2emun-based keys | 2.9.0, 3.0 |
| YARN-6775 | 优化CS#assignContainers，避免不必要的canAssignToUser|Queue调用。优化后：  NodeUpdateScheduler操作由：13K/sec to 50ksec | 2.8.2,2.9.0, 3.0 |
| YARN-4198 | CS中的锁操作顺序有问题，对Queue集合改用并行数据结构，提供并行性 | Patch Available |

## 3.2 YARN系统架构方面的调整

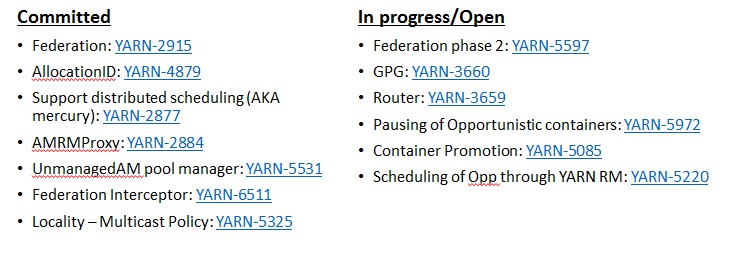
在YARN中RM是单点服务器，其YARN集群规模受RM的限制。YARN RM的运行和NM数目、当前正在运行的应用及心跳（NM和AM都通过心跳与RM交互），可以通过降低心跳来增加NM的扩展性，但是对于YARN的运行来说是不友好的。

### 3.2.1 YARN Federation

YARN提供一个基于federation的方法将集群扩展到上万节点，在这种模式下将YARN集群分成多个子集群，每个子集群都有自己的YARN RM和计算节点。Federation系统将各个子系统结合在一起，应用直接访问Federation System，对子系统是透明的，应用的Tasks可以运行在多个子系统中。系统架构图如下：



在底层实现上，Federation System和子集群的RM协商，并向应用提供资源。进展情况如下：



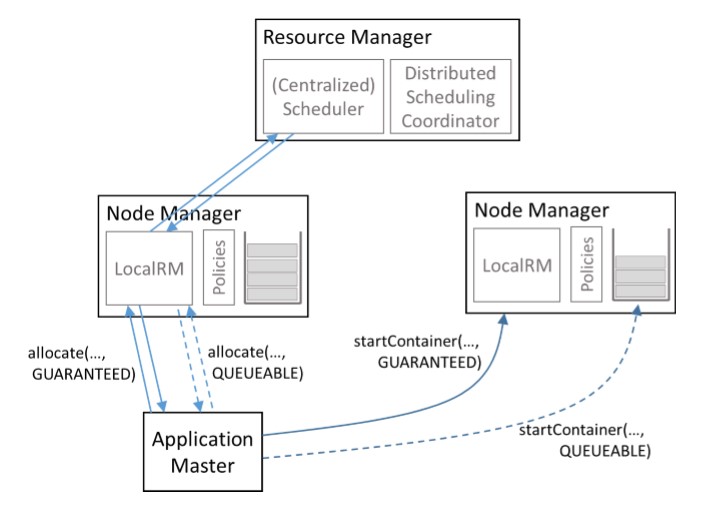
微软基于YARN Federation搭建了13个sub-clusters，一共40K+节点。

### 3.2.2 Distributed Scheduling

在第一章中已经分析了YARN集群规模的首先因素，其中一个因素是YARN RM的集中式ResourceScheduler在集群规模达到一定程度后分配延迟太高。在Hadoop 2.9.0版本后，增加了RS的Distributed Scheduling机制，将集中式资源分配变成分布式资源分配，提高分配运行效率，避免出现Job的获取container的时间过程造成失败。

分布式资源调度允许有多个调度器（设置每个job或者节点）来执行分配策略，在每个节点上进行tasks的排队。为了支持分布式调度，需要一个全局的集群负载状态，所有的scheduler运行相同的调度算法。本地的local scheduling策略不是全局最有的，在end node进行任务的排队可能会导致不可知的Job性能。

在本方案中，保留YARN central RM，在每个集群节点上添加一个Scheduler，节点上的scheduler进行分配策略。在启动应用时，根据task的需要选择scheduling type（centralized或者distribued），对于端作业来说会有很大提升。目标是支持多种不同的应用framework，因此低延迟的调度很重要，而且要支持不同的sharing invariants(capacity/ fairnewss)。系统框架如下所示：



总体上，扩展了YARN RM：

1. LocalRM，新增该组件，代表Local Resource Manager，运行在每个NodeManager上。
2. AM在请求资源时，与其运行的NM交互。根据reqeust type，LocalRM选择将请求转发给RM进行centralized fashion或者直接作为distributed scheduler。
3. 扩展NM，在每个节点进行请求的排队操作，因此多个NM做出的allocation decision可能会冲突。
4. 支持不同的Policy，例如在distributed scheduling中进行container request。Distribured Scheduling决策设置quota，每次进行的请求。NM队列的均衡。
5. 添加Distributed Scheduling Coordinator组件，该组件通过NM-RM心跳接收到NM queue的集合，然后将这些集合分发到其他的LocalRM上。

该功能模块在Hadoop 2.9版本开始支持。

# 总结

根据调研YARN集群规模的影响因素在于（YARN的主要功能）主要有以下过程:

1. NodeManager与ResourceManager之间的心跳，NM通过心跳向RM更新节点的资源使用情况，并通过NODE\_UPDATE事件触发RM的资源分配过程
2. 应用的ApplicationMaster与ResourceManager之间的心跳，AM通过心跳与RM交互，通过pull的方式来获取RM分配给本应用的资源
3. ResourceScheduler执行资源分配过程，RM根据NM汇总的资源进行资源的分配，其需要一定的计算：选择队列（队列的排序），选择队列中的应用（应用的排序）等过程。

RM处理心跳的能力及ResourceScheduler执行资源分配的能力是影响YARN集群规模的两个主要因素。目前对于大规模yarn集群规模的优化主要工作集中在yarn调度本身的算法优化上，这种优化并不能从根本上解决yarn大规模集群的支持上。为了支持yarn大规模集群，需要从yarn的系统架构上做出根本性的改变，主要工作在yarn federation实现及distributed scheduling的两个方面，其中这部分功能在Hadoop 2.9以后会支持。