# Hadoop Yarn 总结

## **Yarn vs. Mesos**

由于先前已经读过 Mesos 的论文，在开始介绍 Yarn 前先简单讲解一下 Yarn 与 Mesos 的对比，以让大家更好地体会两个调度平台的差异。

总体而言，Yarn 在架构上有着和 Mesos 类似的组件角色，它们两者很多地方都是共通的，差别主要在于 Yarn 实现的是中心化的调度模型，而 Mesos 实现的是去中心化的分布式调度模型。由此带来的差别在于 Yarn 的 Master 角色（Resource Manager）在体量上会比 Mesos 的 Master 角色（Mesos Master）要更重，携带更多的集群信息，灾后恢复的过程将会更为复杂（在论文编写时，Yarn Resource Manager 仍无完整的灾后恢复机制）。而中心化调度的好处实际上在 Mesos 的论文中也有提到，就是 Master 更容易利用充足的信息做出最优的资源调度。除此以外，Yarn 和 Mesos 实际上大同小异。

在论文的行文上，Yarn 论文的篇幅比 Mesos 的论文要更长，花了 8 页纸的内容描述 Yarn 的背景、需求、基本组件及功能，而后者则只花了 5 页纸。实际上，Yarn 论文对 Yarn 各组件的介绍极为详细，且描述的多是每一个调度平台所应支持的功能，和 Yarn 本身关联性并不强，因此读者若有意自行实现一套新的集群资源调度平台，我很建议你去阅读 Yarn 论文的第 3 章，了解一个完整的资源调度平台除最核心的资源调度外应具备的辅助功能。可惜的是，Yarn 论文并未过多地描述 Yarn 实际可用的高可用架构，只提及了 Resource Manager 会利用一个“持久化存储”保存状态信息，而 Mesos 论文则详细地解释了 Mesos Master 如何利用 ZooKeeper 和 Mesos Slave 的信息进行有效恢复。

## **Yarn 的背景及需求**

在 Yarn 出现以前，Hadoop MapReduce 的资源管理是只为 Hadoop MapReduce 所设计的，设计之初只考虑了对常规 Hadoop MapReduce 作业的良好支持。然而，由于当时 Hadoop MapReduce 是少有的成熟分布式计算框架，许多开发者为了能够利用上大量的物理资源，不惜研究出各类奇技淫巧跳出 MapReduce 编程模型的限制，包括运行只包含 Map 任务的 MR 作业，甚至是利用 MR 作业部署长时间运行的服务。在这样的场景下，Hadoop MapReduce 的资源管理框架自然表现不佳，甚至出现了大量的论文以 Hadoop MapReduce 作为完全不相关测试环境的测试基准线。

总得来说，Hadoop 原有的资源管理模块有以下两个不足：

* 资源管理模块与 MapReduce 这个具体的编程模型之间存在紧密的耦合，导致开发者们不得不去滥用这个编程模型
* 对作业控制流和生命周期的中央式处理带来了各种各样的可扩展性问题

在这样的背景下， 通过将资源管理功能与具体编程模型分离，Yarn 得以将很多与作业调度相关的功能委托给了作业所提供的具体组件。在这样的语境下，MapReduce 实际上成为了运行在 Yarn 之上的一个应用。

除此以外，各种各样其他的公司内部需求延伸出了 Yahoo! 对 Yarn 的 10 个主要功能需求。这里只简要介绍这 10 个功能需求，关于它们产生的原因详见论文的第 2 章。

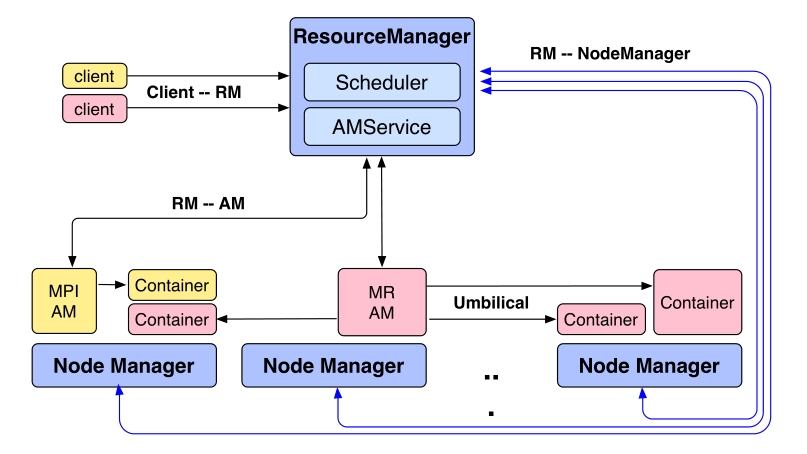
1. 可扩展性（Scalability）
2. 对多租户的支持（Multi-Tenancy）
3. 服务性（[Serviceability](https://en.wikipedia.org/wiki/Serviceability_(computer" \t "https://mr-dai.github.io/yarn_summary/_blank))）：源自 Hadoop on Demand 已有的一个功能特性，指将资源管理框架与上层应用框架各自的版本升级进行解耦的能力
4. 本地性感知（Locality Awareness）：HoD 无法做到的功能，指对 HDFS 上存储数据的位置的感知，将任务尽量分配到相近的结点上，提高数据本地性
5. 高集群利用率（High Cluster Utilization）
6. 可用性（Availability）
7. 安全且可审计的操作（Secure and Auditable Operation）：对多租户的支持所延伸出的安全需求
8. 支持不同的编程模型（Support for Programming Model Diversity）
9. 弹性的资源模型（Flexible Resource Model）
10. 前向兼容（Backward Compatability）

## **Yarn 资源分配流程**

Yarn 的语境包含两类角色：

* 平台（Platform）：负责集群的资源管理，如 Yarn
* 框架（Framework）：负责逻辑执行计划的协调，如 MR

为了实现集群资源管理，Yarn 采用的也是 Master - Slave 架构。Yarn 的架构示意图如下：

[](https://mr-dai.github.io/img/yarn_architecture.png)

Yarn 架构中主要包含以下 3 种角色：

* ****Resource Manager****：Yarn 集群的 Master，负责追踪集群的资源使用以及结点存活信息并进行资源调度
* ****Application Master****：负责为某个作业协调逻辑执行计划、向 Resource Manager 申请资源并生成物理执行计划，并在考虑错误可能发生的情况下协调物理计划的执行
* ****Node Manager****：运行在集群的各个结点上，负责管理结点的资源和健康信息，并与 Resource Manager 进行通信

要通过 Yarn 启动一次作业，大致需要经历如下几个步骤：

1. 客户端向 Resource Manager 提交作业，获取执行权限与资源
2. 当有足够的可用资源后，Resource Manager 会负责在集群的一个结点上启动一个容器并启动作业的 Application Master
3. Application Master 会开始向 Resource Manager 发起容器请求，包括每个容器的资源需求以及所需的容器个数，同时还包含数据本地性偏好等信息
4. Resource Manager 会根据自己的资源分配策略尽量满足每一个应用的需求。响应时，Resource Manager 会把可用的资源绑定到一个 Token 上并返回给 Application Master
5. 当 Application Master 获得 Token 后便会利用该 Token 向可用的 Node Manager 发出容器启动请求

## **Yarn 机制详解**

本节将基于上述 Yarn 资源分配的基本过程，详细讲解每一个 Yarn 集群资源的职责和功能。

### **Resource Manager**

作为 Yarn 集群的 Master 角色，Resource Manager 通常会作为守护进程运行在专属的机器上，利用其对全集群资源的使用情况的了解执行各种各样的调度策略。为了获知全集群的真实情况，Resource Manager 会基于心跳机制与 Node Manager 进行通信，通过 Node Manager 汇报的结点状态快照信息构建出全集群的状态。

当接收到来自客户端或 Application Master 的资源请求后，Resource Manager 会对资源进行调度，并在有足够的可用资源时以****容器****（Container）的形式返回资源。实际上，容器只是一个表示某个结点上指定量资源的逻辑结构。返回的内容中还包括了 Application Master 从 Node Manager 处使用这些资源时所需要的 Token。除此以外，Resource Manager 还会把在各个 Node Manager 上运行的应用任务的退出状态发送给 Application Master。

为了更有效地利用集群，Resource Manager 还可以向 Application Master 撤回已分配的资源，并在 Application Master 过长时间不作出响应时直接通知 Node Manager 杀死正在运行的任务。

值得注意的是，Resource Manager 作为独立于应用的 Yarn 组件是不会帮助应用做具体的处理的，这些逻辑将交由应用的 Application Master 或其他进程完成。类似的事情包括：

* Resource Manager 不会为正在运行的应用提供状态和性能状况等度量信息。这个实际上是 Application Master 的职责
* Resource Manager 不会为框架向外提供已完成作业的报告信息。这个实际上应由框架自己的某个守护进程完成

可用性方面，Resource Manager 作为集群的 Master 存在单点故障的问题。

Resource Manager 在恢复时首先会从一个持久化存储中恢复状态信息。恢复完毕后，Resource Manager 会杀死当前集群上的所有容器，包括 Application Master。然后它会再为每个 Application Master 进行重新启动。

这样的恢复无疑是十分粗糙的，极大地依赖了上层应用的可用性，因此 Yarn 也在不断改进，思考如何能在保留容器的情况下完成 Resource Manager 恢复。

### **Application Master**

Application Master 实际上类似于 Mesos 中的 Framework Scheduler，是有上层框架所提供的组件，负责为自己所负责的作业从 Resource Manager 处获取资源并运行各个子任务。Application Master 负责在集群中协调应用的执行，但其自身实际上也像其他容器一样运行在集群中。

Application Master 会周期地与 Resource Manager 通信以汇报自己的存活以及资源需求。Application Master 所能发出的资源请求（ResourceRequest）里的信息主要包括了其对每个容器的资源需求（CPU、内存）、容器的个数以及本地性偏好和优先级等。

来自 Resource Manager 的资源响应中包含了所提供的容器以及访问这些资源所需要的 Token。基于其从 Resource Manager 接收到的资源，Application Master 可以动态地调整自己的物理执行计划，并再向 Resource Manager 更新自己的资源需求。

由于 Application Master 本身也运行在某个 Node Manager 的一个容器里，Application Master 本身也需要具备高可用性，实现针对框架自身的灾后恢复。

当 Resource Manager 应集群资源紧缺发来资源撤回的请求时，Application Master 可以选择对部分容器进行主动关闭，并在关闭前完成数据保存的操作。相比于让 Resource Manager 通过 Node Manager 直接杀死容器，由 Application Master 主动退出更有利于容器后续的恢复。

Application Master 的失效实际上不是 Yarn 所关心的：Yarn 在 Application Master 失效后只会帮其进行重新启动，Application Master 的状态恢复需要由它自己完成。

### **Node Manager**

作为 Yarn 集群的 Worker 角色，Node Manager 作为守护进程运行在集群的每一个机器上。Node Manager 会验证来自 Application Master 的容器启动请求，并基于心跳信息与 Resource Manager 通信。Node Manager 主要负责追踪结点的资源使用情况和健康状况、管理容器的生命周期、向 Resource Manager 汇报错误等。

包括 Application Master 在内的所有容器都会采用一种叫做****容器启动上下文****（Container Launch Context，CLC）的数据结构所表述，其中包含启动一个容器所需的环境变量、对远程存储的依赖以及启动命令等信息。

在启动容器前，Node Manager 首先会对 Application Master 发来的 Token 进行验证。验证通过后，Node Manager 会先配置容器的启动环境，包括设置环境变量、初始化监控模块、拷贝依赖文件至本地存储等。拷贝到本地存储的依赖会被维持在本地，由不同的容器所共享。最终，Node Manager 会在某个依赖确定不再被使用时将其从本地存储上移除。

Node Manager 还会负责处理来自 Resource Manager 和 Application Master 的容器终止信号。发出这样的信号有可能是由于集群资源不足，也有可能是由于容器所属的应用已执行完毕，容器负责的任务已不再被需要。当应用结束后，其在所有结点上的容器所占用的资源都会被占用。总之，无论如何， Node Manager 在容器退出后都会负责清理容器的工作目录。

除此以外，Node Manager 也会为应用任务的执行提供一些便利服务。例如，Node Manager 会在任务完成后对任务的 stdout 和 stderr 输出进行日志聚合（Log Aggregation），上传到 HDFS 中进行持久化。除此以外，Node Manager 还提供了文件驻留的功能，允许任务把部分的输出文件进行保留而不被容器退出时的清理工作所删除。

由于 Node Manager 和 Resource Manager 中一直保持着心跳连接，Resource Manager 很容易通过超时发现某个 Node Manager 失效。此时 Resource Manager 就会认定该 Node Manager 上的所有容器已被杀死，并把执行失败的信息回报给 Application Master。Node Manager 在恢复时会重新与 Resource Manager 进行同步，并清理自己的本地状态。

# 参考

<https://mr-dai.github.io/yarn_summary/>