第八讲 资源管理系统Yarn

徐 辰
cxu@dase.ecnu.edu.cn
華東師範大學

大纲

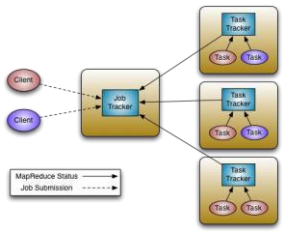
2

- 设计思想
  - 再论MapReduce的局限性
  - 平台与框架
- 体系架构
- 工作原理
- 容错机制
- 典型应用

MapReduce v1 Architecture

3

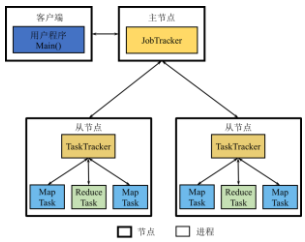
- JobTracker
  - Manage Cluster Resources & Job Scheduling
- TaskTracker
  - Per-node agent
  - Manage Tasks



MapReduce 1.0 JobTracker

4

- 作业管理：状态监控、信息汇总、任务调度等
- 资源管理：



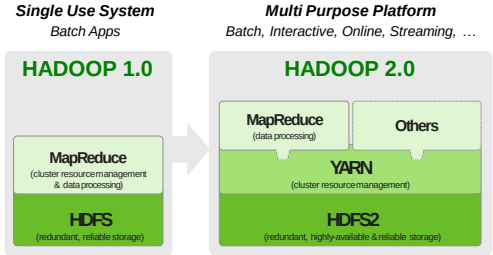
MapReduce 1.0的缺陷

5

- 资源管理与作业紧密耦合
  - 资源管理不单是MapReduce系统所需要的，而是通用的
- 作业控制管理高度集中
  - JobTracker存在单点故障风险
  - JobTracker“大包大揽”内存开销大

Hadoop as Next-Gen Platform

6



YARN: Yet Another Resource Negotiator

7

- 设计思路：资源管理与计算相分离
  - MapReduce 1.0既是计算系统，也是资源管理系统
  - Yarn是独立出来的**资源管理系统**

Hadoop 1.0	功能	Yarn
JobTracker	资源管理	ResourceManager
	作业管理	ApplicationMaster
TaskTracker	节点管理	NodeManager
Task	执行计算	Container

大纲

8

- 设计思想
  - 再论MapReduce的局限性
  - 平台与框架
- 体系架构
- 工作原理
- 容错机制
- 典型应用

应用 vs. 作业

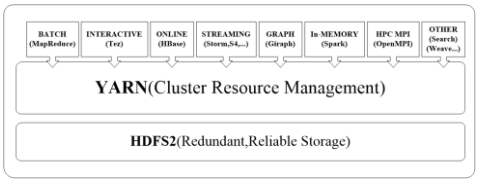
9

- 对于大多数系统来说，应用(Application)与作业(Job)是同一概念
- MapReduce: application = Job
- Spark: application = 一个或多个Job
- Yarn管理的粒度是Application

平台 vs. 框架

10

- 系统
  - 平台：Yarn
  - 框架：MapReduce、Spark……



大纲

11

- 设计思想
- 体系架构
  - 架构图
  - 单平台多框架
- 工作原理
- 容错机制
- 典型应用

YARN体系架构

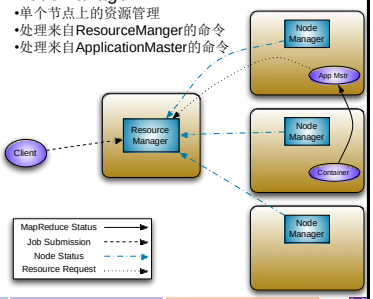
12

ResourceManager
- 处理客户端请求
- 启动/监控ApplicationMaster
- 监控NodeManager
- 资源分配与调度

NodeManager
- 单个节点上的资源管理
- 处理来自ResourceManger的命令
- 处理来自ApplicationMaster的命令

ApplicationMaster
- 为应用程序申请资源，并分配给内部任务
- 任务调度、监控与容错

Container
- 提供运行任务的资源



ResourceManager

13

- ResourceManager(RM)是一个全局的资源管理器，包括
  - 资源调度器(Resource Scheduler)：资源分配
  - 应用程序管理器(Application Manager)：所有应用程序的管理工作(轻量作业监控)
    - 检查是否由足够资源部署application master
    - 维护application id
    - Web UI展示信息
    - ……

NodeManager

14

- 监控所在节点每个Container(容器)的资源（CPU、内存等）使用情况、健康状况
- 向ResourceManager汇报作业的资源使用情况和每个容器的运行状态
- NodeManager主要负责管理抽象的容器，只处理与容器相关的事情，而不具体负责每个任务（如，Map任务或Reduce任务）自身状态的管理

ApplicationMaster

15

- 与ResourceManager(RM)的交互
  - 与RM协商获取资源，把获得的资源进一步分配给内部的各个任务（如Map或Reduce任务）
  - 定时向RM报告资源使用情况和应用进度信息
  - 作业完成时，向RM注销容器，执行周期完成
- 与NodeManager保持通信进行应用程序的管理
- 监控任务的执行进度和状态，并在任务发生失败时重新申请资源重启任务

Container

16

- 容器（Container）作为动态资源分配单位，每个容器中都封装了一定数量的CPU、内存、磁盘等资源，从而限定每个应用程序可以使用的资源量
- 这些资源用于执行计算任务

大纲

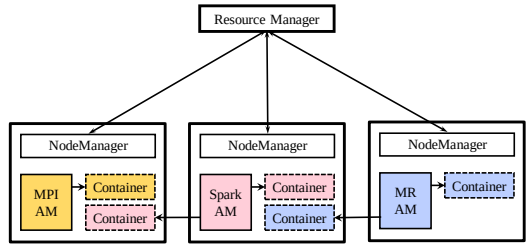
17

- 设计思想
- 体系架构
  - 架构图
  - 单平台多框架
- 工作原理
- 容错机制
- 典型应用

一个平台多个框架

18

- 一个资源管理平台运行多个计算框架



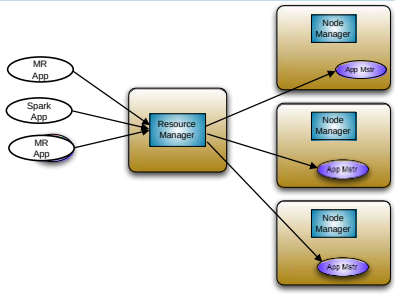
一个系统一个集群

19

- 企业当中同时存在不同的业务，需要采用不同的计算框架，如Hadoop、Spark
- 一个系统一个集群：为了避免干扰，把内部的服务器拆分，分别部署不同的系统
  - 集群资源利用率低
  - 维护代价高
- 多个系统一个集群：Yarn
  - 部署多个计算系统并共享资源

多个系统一个集群

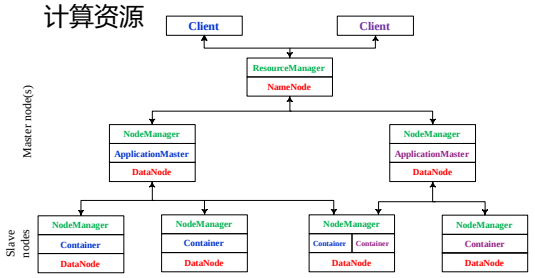
20



YARN集群部署

21

- 多租户：利用Yarn实现多个计算系统共享计算资源



大纲

22

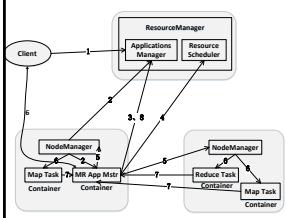
- 设计思想
- 体系架构
- 工作原理
  - 应用生命周期
  - 平台资源分配
- 容错机制
- 典型应用

应用的生命周期

23

步骤1：用户编写客户端应用程序，**向YARN提交应用程序**，提交的内容包括AppMaster程序、启动AppMaster的命令、用户程序等

步骤2：YARN中的RM负责接收和处理来自客户端的请求，为应用程序分配一个容器，在该容器中启动一个AppMaster



步骤3：AppMaster被创建后会首先向RM注册

步骤4：AppMaster**向RM申请资源**

步骤5：RM以“容器”的形式向提出申请的AppMaster分配资源

步骤6：在容器中启动任务（运行环境、脚本）

步骤7：各个任务**向AppMaster汇报自己的状态和进度**

步骤8：应用程序运行完成后，AppMaster向RM的应用程序管理器注销并关闭自己

应用的结束

24

- 随着部分任务执行结束，Application Master向Resource Manager和Node Manager逐步释放所占用的资源，最终AM注销并关闭自己
- 短时间应用与长时间应用
  - 短时间应用：运行一段时间终止
  - 长时间应用：随着输入不断运行
- MapReduce和Spark应用是哪一类?

大纲

25

- 设计思想
- 体系架构
- 工作原理
  - 应用生命周期
  - 平台资源分配
- 容错机制
- 典型应用

资源分配

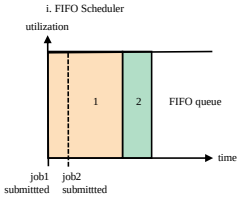
26

- Resource Manager中的调度器维护了一个或多个应用队列（queue），每个队列拥有一定量的资源，位于同一队列中的应用共享该队列所拥有的资源
- Yarn进行资源分配对象是应用，用户提交的每个应用会分配到其中一个队列当中，而队列决定了该应用能使用的资源上限
- 资源调度实际上是决定如何将资源分配给队列、以及如何分配给队列中应用的过程

资源分配策略-FIFO

27

- FIFO Scheduler只维护一个队列，该队列拥有集群中所有的资源，调度器的资源分配方式是先提交的应用先得到资源

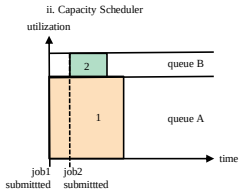


应用1占用所有资源，应用2需要等待应用1执行完毕后才会被执行。

资源分配策略-Capacity

28

- Capacity Scheduler维护了层级式的队列，集群中的资源划分给这些队列，队列内部的资源分配方式是FIFO



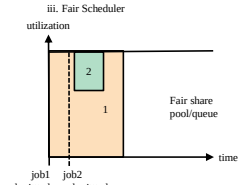
Capacity Scheduler调度方式可以避免某一长时间运行的应用独占集群资源而其它应用得不到运行的情况。

然而，我们也能观察到，在提交应用程序2之前队列B中的资源处于空闲状态，这造成了集群资源的浪费。

资源分配策略-Fair

29

- Fair Scheduler维护层级式的队列，集群中的资源划分给这些队列，但是这些队列可以共享资源，因而这些队列逻辑上可以看作是一个共享队列



当只有一个应用运行时，这个应用可以独占整个集群。但当其它应用提交到集群时，将空出部分资源给新的应用，最终所有的应用会根据所需使用内存的大小得到分配的资源。

大纲

30

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 典型应用
  - 第二代MapReduce
  - 基于Yarn部署Spark

故障类型

31

- Resource Manager故障
- Node Manager故障
- Application Master故障：重启
- Container中的任务故障：重启

Resource Manager故障

32

- 如果Resource Manager发生故障，那么它在进行故障恢复时需要从某一持久化存储系统中恢复状态信息，所有应用将会重新执行。
- 我们可以部署多个Resource Manager并通过ZooKeeper进行协调，从而保证Resource Manager的高可用性

Node Manager故障

33

- Resource Manager认为Node Manager所在节点上所有容器运行的任务也都执行失败，并把执行失败的信息告诉Application Master。
  - AM将向RM重新申请资源运行这些任务
  - RM将分配其它节点的Container执行这些任务
- 如果发生故障的Node Manager进行恢复，那么它将向Resource Manager重新注册，重置本地的状态信息

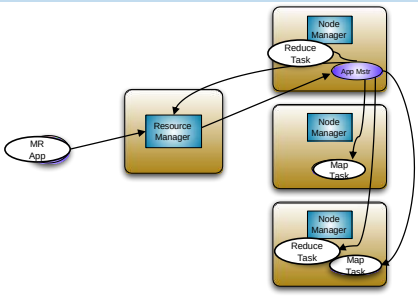
大纲

34

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 典型应用
  - 第二代MapReduce
  - 基于Yarn部署Spark

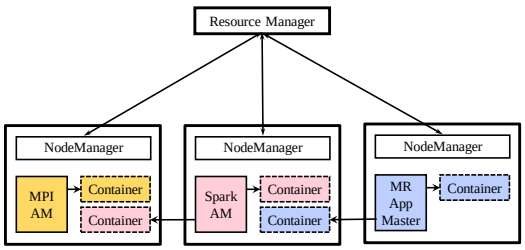
MapReduce v2 工作流程

35



MapReduce V2

36



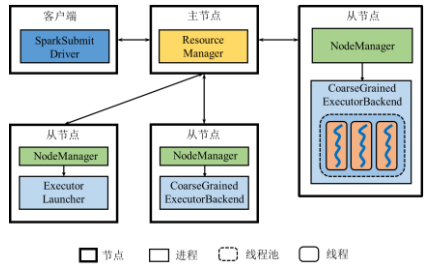
大纲

37

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 典型应用
  - 第二代MapReduce
  - 基于Yarn部署Spark

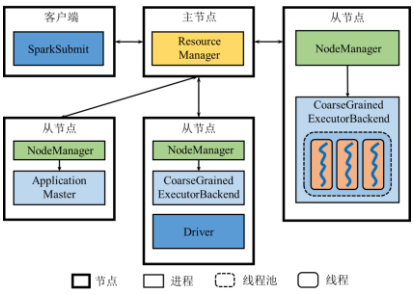
Yarn-client

38



Yarn-cluster

39



Yarn-client vs. Yarn-cluster

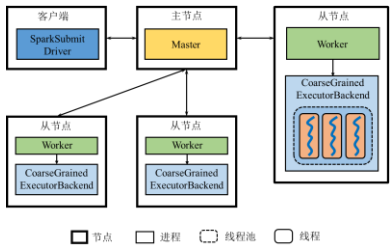
40

- Yarn-client
  - Driver：在客户端启动的进程中
  - ApplicationMaster：表现为ExecutorLauncher，向ResourceManager申请资源，用container资源去链接其他的NodeManager，然后去启动executor
- Yarn-cluster
  - Driver存在于NodeManager上的某一个ApplicationMaster

Standalone Client

41

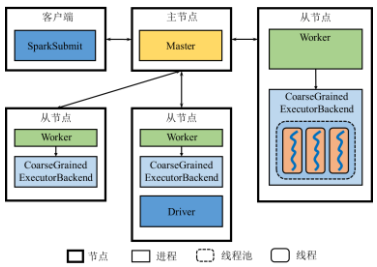
- Driver和客户端以同一个进程存在



Standalone Cluster

42

- 某一Worker启动一个进程作为Driver



课后阅读

43

- 论文
  - Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., … Saha, B. (2013). Apache Hadoop yarn: Yet another resource negotiator. In SoCC (pp. 5:1-5:16).

本讲小结

44

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 典型应用

谢谢！Q&A