第八章 hadoop2.0再探讨

CONTENTS

01

Hadoop2.0新机制

8.1.1Hadoop的局限与不足

Hadoop1.0的核心组件（仅指MapReduce和HDFS，不包括Hadoop生态系统内的Pig、Hive、HBase等其他组件），主要存在以下不足：

- 抽象层次低，需人工编码
- 表达能力有限
- 开发者自己管理作业（Job）之间的依赖关系
- 难以看到程序整体逻辑
- 执行迭代操作效率低
- 资源浪费（Map和Reduce分两阶段执行）
- 实时性差（适合批处理，不支持实时交互式）

8.1.2针对Hadoop的改进与提升

表 Hadoop框架自身的改进：从1.0到2.0

组件	Hadoop1.0的问题	Hadoop2.0的改进
HDFS	单一名称节点，存在单点失效问题	设计了HDFS HA，提供名称节点热备机制
HDFS	单一命名空间，无法实现资源隔离	设计了HDFS Federation，管理多个命名空间
MapReduce	资源管理效率低	设计了新的资源管理框架YARN

8.1.2针对Hadoop的改进与提升

表 不断完善的Hadoop生态系统

组件	功能	解决Hadoop中存在的问题
Pig	处理大规模数据的脚本语言，用户只需要编写几条简单的语句，系统会自动转换为MapReduce作业	抽象层次低，需要手工编写大量代码
Spark	基于内存的分布式并行编程框架，具有较高的实时性，并且较好支持迭代计算	延迟高，而且不适合执行迭代计算
Oozie	工作流和协作服务引擎，协调Hadoop上运行的不同任务	没有提供作业（Job）之间依赖关系管理机制，需要用户自己处理作业之间依赖关系
Tez	支持DAG作业的计算框架，对作业的操作进行重新分解和组合，形成一个大的DAG作业，减少不必要操作	不同的MapReduce任务之间存在重复操作，降低了效率
Kafka	分布式发布订阅消息系统，一般作为企业大数据分析平台的数据交换枢纽，不同类型的分布式系统可以统一接入到Kafka，实现和Hadoop各个组件之间的不同类型数据的实时高效交换	Hadoop生态系统中各个组件和其他产品之间缺乏统一的、高效的数据交换中介

关于Secondar yNameNode下面哪项是正确的___（单选题）。

A. 它是 NameNode 的热备

B. 它对内存没有要求

C. 它的目的是帮助NameNode合并编辑日志，减少NameNode 启动时间

D. SecondaryNameNode 应与NameNode 部署到一个节点

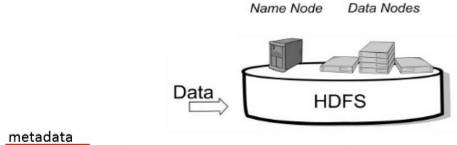
8.2.1HDFS HA

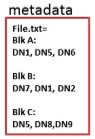
HDFS1.0组件及其功能回顾（具体请参见第3章HDFS）

名称节点保存元数据：

（1）在磁盘上：FsImage和EditLog

（2）在内存中：映射信息，即文件包含哪些块，每个块存储在哪个数据节点

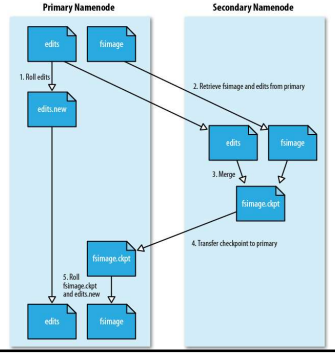




NameNode	DataNode
• 存储元数据	• 存储文件内容
•元数据保存在内存中	•文件内容保存在磁盘
• 保存文件,block，datanode之间的映射关系	•维护了block id到datanode本地文件的映射关系

8.2.1HDFS HA

•HDFS 1.0存在单点故障问题
•第二名称节点（SecondaryNameNode）无法解决单点故障问题



•SecondaryNameNode会定期和NameNode通信
•从NameNode上获取到FsImage和EditLog文件，并下载到本地的相应目录下
•执行EditLog和FsImage文件合并
•将新的FsImage文件发送到NameNode节点上
•NameNode使用新的FsImage和EditLog（缩小了）

第二名称节点用途：
•不是热备份
•主要是防止日志文件EditLog过大，导致名称节点失败恢复时消耗过多时间
•附带起到冷备份功能

HDFS HA

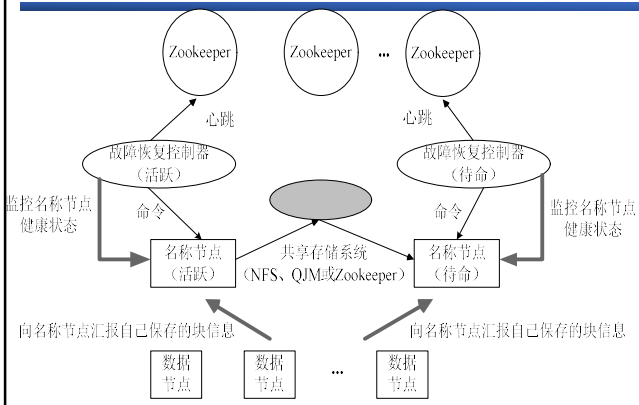


图 HDFS HA架构

•HDFS HA（High Availability）是为了解决单点故障问题
•HA集群设置两个名称节点，“活跃（Active）”和“待命（Standby）”
•两种名称节点的状态同步，可以借助于一个共享存储系统来实现
•一旦活跃名称节点出现故障，就可以立即切换到待命名称节点
•Zookeeper确保一个名称节点在对外服务
•名称节点维护映射信息，数据节点同时向两个名称节点汇报信息

HDFS Federation

HDFS HA是热备份，提供高可用性，但是无法解决可扩展性、系统性能和隔离性

HDFS Federation的设计

•在HDFS Federation中，设计了多个相互独立的名称节点，使得HDFS的命名服务能够水平扩展，这些名称节点分别进行各自命名空间和块的管理，相互之间是联盟（Federation）关系，不需要彼此协调。并且向后兼容

•HDFS Federation中，所有名称节点会共享底层的数据节点存储资源，数据节点向所有名称节点汇报

•属于同一个命名空间的块构成一个“块池”

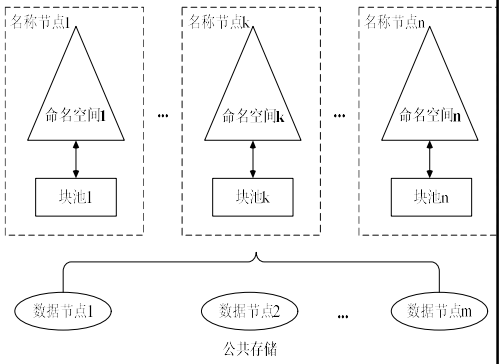


图 HDFS Federation架构

HDFS Federation

3. HDFS Federation的访问方式

“客户端挂载表”（mount-table）实现数据全局共享

各挂载点

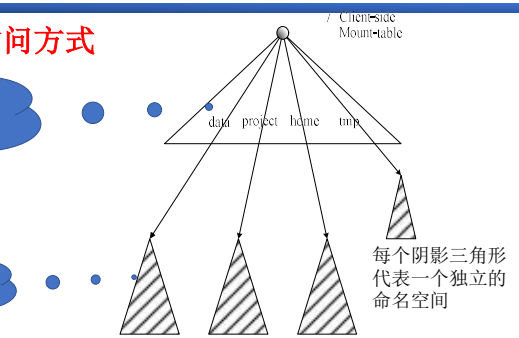


图 客户端挂载表方式访问多个命名空间

HDFS Federation

HDFS Federation相对于HDFS1.0的优势

HDFS Federation设计可解决单名称节点存在的以下几个问题：
（1）HDFS集群扩展性。多个名称节点各自分管一部分目录，使得一个集群可以扩展到更多节点，不再像HDFS1.0中那样由于内存的限制制约文件存储数目
（2）性能更高效。多个名称节点管理不同的数据，且同时对外提供服务，将为用户提供更高的读写吞吐率
（3）良好的隔离性。用户可根据需要将不同业务数据交由不同名称节点管理，这样不同业务之间影响很小

需要注意的，HDFS Federation并不能解决单点故障问题，也就是说，每个名称节点都存在在单点故障问题，需要为每个名称节点部署一个后备名称节点，以应对名称节点挂掉对业务产生的影响

02

资源管理调度框架Yarn

8.3.1 MapReduce1.0的缺陷

（1）存在单点故障
（2）JobTracker“大包大揽”导致任务过重（任务多时内存开销大，上限4000节点）
（3）容易出现内存溢出（分配资源只考虑MapReduce任务数，不考虑CPU、内存）
（4）资源划分不合理（强制划分为slot，包括Map slot和Reduce slot）

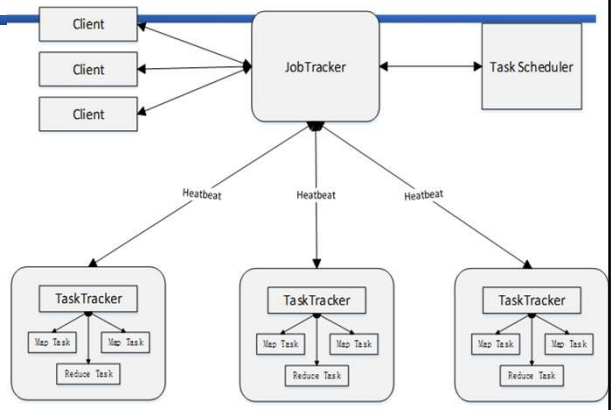


图 MapReduce1.0体系结构

MapReduce1.0的缺陷

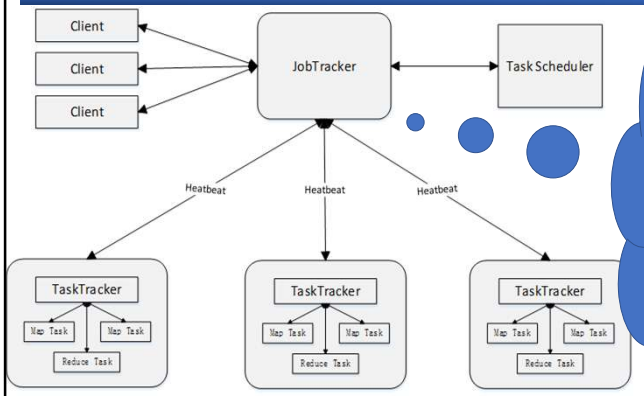
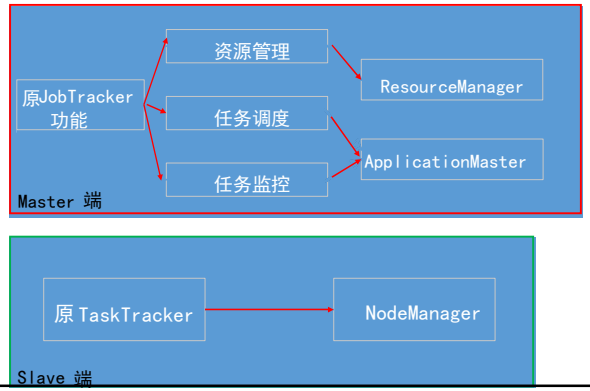


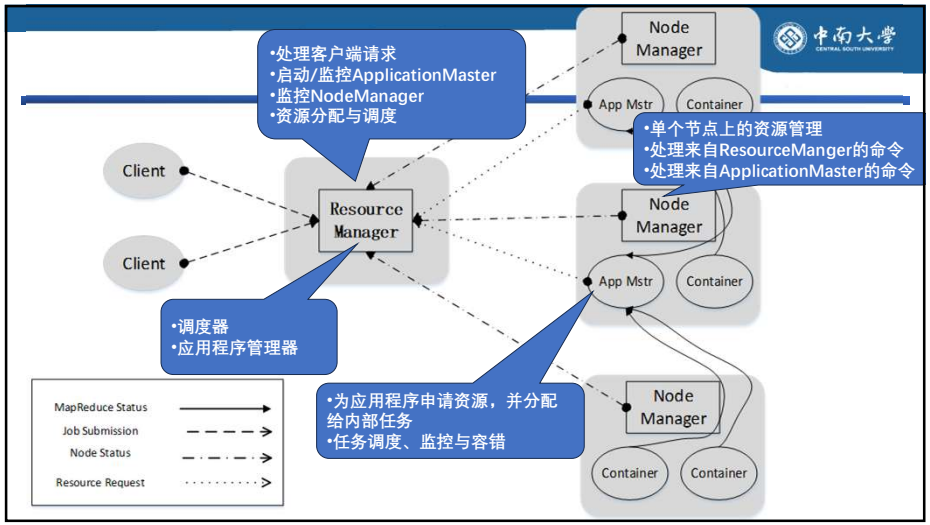
图 MapReduce1.0体系结构

- JobTracker“大包大揽”导致任务过重（任务多时内存开销大，上限4000节点）
- 容易出现内存溢出（分配资源只考虑MapReduce任务数，不考虑CPU、内存）
- 资源划分不合理（强制划分为slot，包括Map slot和Reduce slot）

YARN设计思路

YARN 架构思路：将原 JobTacker 三大功能拆分





YARN体系结构

ResourceManager

•ResourceManager（RM）是一个全局的资源管理器，负责整个系统的资源管理和分配，主要包括两个组件，即调度器（Scheduler）和应用程序管理器（Applications Manager）

•调度器接收来自ApplicationMaster的应用程序资源请求，把集群中的资源以“容器”的形式分配给提出申请的应用程序，容器的选择通常会考虑应用程序所要处理的数据的位置，进行就近选择，从而实现“计算向数据靠拢”

•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

•调度器被设计成是一个可插拔的组件，YARN不仅自身提供了许多种直接可用的调度器，也允许用户根据自己的需求重新设计调度器

YARN体系结构

ApplicationMaster

ResourceManager接收用户提交的作业，按照作业的上下文信息以及从NodeManager收集来的容器状态信息，启动调度过程，为用户作业启动一个ApplicationMaster

ApplicationMaster的主要功能是：

（1）当用户作业提交时，ApplicationMaster与ResourceManager协商获取资源，ResourceManager会以容器的形式为ApplicationMaster分配资源；

（2）把获得的资源进一步分配给内部的各个任务（Map任务或Reduce任务），实现资源的“二次分配”；

（3）与NodeManager保持交互通信进行应用程序的启动、运行、监控和停止，监控申请到的资源的使用情况，对所有任务的执行进度和状态进行监控，并在任务发生失败时执行失败恢复（即重新申请资源重启任务）；

（4）定时向ResourceManager发送“心跳”消息，报告资源的使用情况和应用的进度信息；

（5）当作业完成时，ApplicationMaster向ResourceManager注销容器，执行周期完成。

8.3.3 YARN体系结构

NodeManager

NodeManager是驻留在一个YARN集群中的每个节点上的代理，主要负责：

- 容器生命周期管理
- 监控每个容器的资源（CPU、内存等）使用情况
- 跟踪节点健康状况
- 以“心跳”的方式与ResourceManager保持通信
- 向ResourceManager汇报作业的资源使用情况和每个容器的运行状态
- 接收来自ApplicationMaster的启动/停止容器的各种请求

需要说明的是，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（Map任务或Reduce任务）自身状态的管理，因为这些管理工作是由ApplicationMaster完成的，ApplicationMaster会通过不断与NodeManager通信来掌握各个任务的执行状态

YARN体系结构

在集群部署方面，YARN的各个组件是和Hadoop集群中的其他组件进行统一部署的

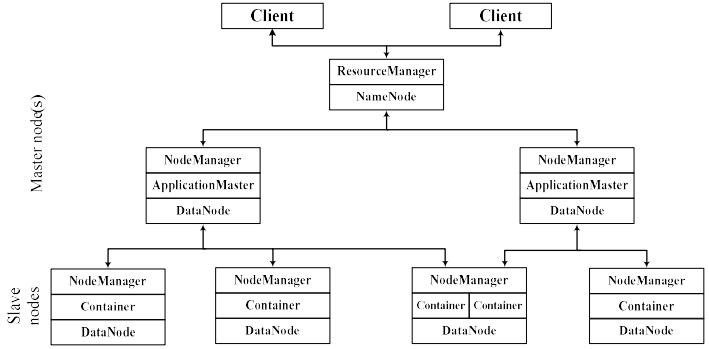
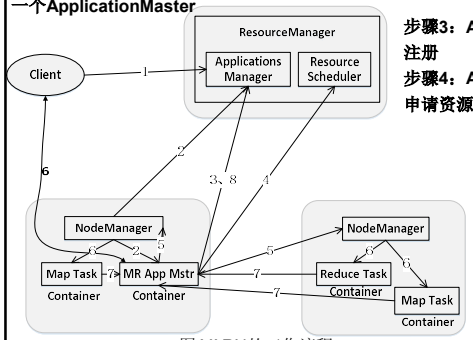


图 YARN和Hadoop平台其他组件的统一部署

YARN工作流程

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ApplicationMaster程序、启动ApplicationMaster的命令、用户程序等
步骤2：YARN中的ResourceManager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ApplicationMaster



步骤3：ApplicationMaster被创建后会首先向ResourceManager注册
步骤4：ApplicationMaster采用轮询的方式向ResourceManager申请资源

步骤5：ResourceManager以“容器”的形式向提出申请的ApplicationMaster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ApplicationMaster汇报自己的状态和进度
步骤8：应用程序运行完成后，ApplicationMaster向ResourceManager的应用程序管理器注销并关闭自己

图 YARN的工作流程

YARN框架与MapReduce1.0框架的对比分析

YARN相对于MapReduce1.0来说具有以下优势：

- 大大减少了承担中心服务功能的ResourceManager的资源消耗
  - ApplicationMaster来完成需要大量资源消耗的任务调度和监控
  - 多个作业对应多个ApplicationMaster，实现了监控分布化

- MapReduce1.0既是一个计算框架，又是一个资源管理调度框架，但是，只能支持MapReduce编程模型。而YARN则是一个纯粹的资源调度管理框架，在它上面可以运行包括MapReduce在内的不同类型的计算框架，只要编程实现相应的ApplicationMaster

- YARN中的资源管理比MapReduce1.0更加高效
  - 以容器为单位，而不是以slot为单位

YARN的发展目标

•YARN的目标就是实现“一个集群多个框架”，即在一个集群上部署一个统一的资源调度管理框架YARN，在YARN之上可以部署其他各种计算框架

•由YARN为这些计算框架提供统一的资源调度管理服务，并且能够根据各种计算框架的负载需求，调整各自占用的资源，实现集群资源共享和资源弹性收缩

•可以实现一个集群上的不同应用负载混搭，有效提高了集群的利用率

•不同计算框架可以共享底层存储，避免了数据集跨集群移动

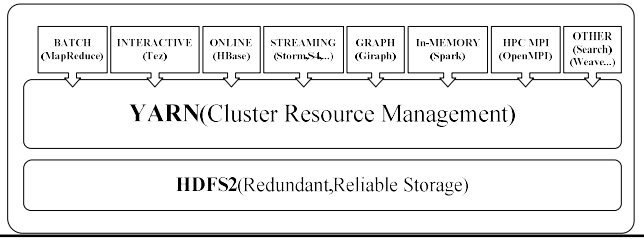


图 在YARN上部署各种计算框架