厦门大学林子雨编著

《大数据技术原理与应用》

第 15 章 Hadoop 架构再探讨

（版本号：2016 年 4 月 13 日版本）

（备注：2015 年 8 月 1 日第一版教材中没有本章，本章为 2016 年新增内容，将被放入第二版教材中）

主讲教师：林子雨
厦门大学数据库实验室
二零一六年四月

中国高校大数据课程
公 共 服 务 平 台

中国高校大数据课程公共服务平台，由中国高校首个“数字教师”的提出者和建设者——林子雨老师发起，由厦门大学数据库实验室全力打造，由厦门大学云计算与大数据研究中心、海峡云计算与大数据应用研究中心携手共建。这是国内第一个服务于高校大数据课程建设的公共服务平台，旨在促进国内高校大数据课程体系建设，提高大数据课程教学水平，降低大数据课程学习门槛，提升学生课程学习效果。

平台为教师开展大数据教学和学生学习大数据课程，提供全方位、一站式**免费**服务，包括**讲义 PPT**、教学大纲、备课指南、**学习指南**、上机习题、**授课视频**、技术资料等。

百度搜索“厦门大学数据库实验室”，访问平台主页，或直接访问平台地址：
http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

扫一扫访问平台主页

《大数据技术原理与应用——概念、存储、处理、分析与应用》，由厦门大学计算机科学系教师林子雨博士编著，是中国高校第一本系统介绍大数据知识的专业教材。本书定位为大数据技术入门教材，为读者搭建起通向“大数据知识空间”的桥梁和纽带，以“构建知识体系、阐明基本原理、引导初级实践、了解相关应用”为原则，为读者在大数据领域“深耕细作”奠定基础、指明方向。

全书共有 13 章，系统地论述了大数据的基本概念、大数据处理架构 Hadoop、分布式文件系统 HDFS、分布式数据 库 HBase、NoSQL 数据库、云数据库、分布式并行编程模型 MapReduce、流计算、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在 Hadoop、HDFS、HBase 和 MapReduce 等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：

http://dblab.xmu.edu.cn/post/bigdata

扫一扫访问教材官网

目录

第 15 章 Hadoop 架构再探讨

Hadoop 作为一种开源的大数据处理架构，在业内得到了广泛的应用，几乎成为大数据技术的代名词。但是，Hadoop 在诞生之初，在架构设计和应用性能方面，仍然存在一些不尽人意的地方，并在后续发展过程中得到了逐渐的改进和完善。Hadoop 的优化与发展主要体现在两个方面，一方面是 Hadoop 自身两大核心组件 MapReduce 和 HDFS 的架构设计改进，另一方面是 Hadoop 生态系统其它组件的不断丰富。通过这些优化和提升，Hadoop 可以支持更多的应用场景，提供更高的集群可用性，同时也带来了更高的资源利用率。

本章首先介绍 Hadoop 的局限与不足，并从全局视角系统总结针对 Hadoop 的改进与提升；然后，介绍 Hadoop 在自身核心组件方面的新发展，包括 HDFS2.0 新特性和新一代资源管理调度框架 YARN 框架；最后，介绍 Hadoop 推出之后陆续涌现的具有代表性的新功能组件，包括 Pig、Tez、Spark 和 Kafka 等，这些组件对 Hadoop 的局限进行了有效的改进，进一步丰富和发展了 Hadoop 生态系统。

15.1 Hadoop 的优化与发展

15.1.1Hadoop 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。需要手工编写代码来完成，有时只是为了实现一个简单的功能，也需要编写大量的代码；
- 表达能力有限。MapReduce 把复杂分布式编程工作高度抽象到两个函数，即 Map 和 Reduce，在降低开发人员程序开发复杂度的同时，却也带来了表达能力有限的问题。实际生产环境中的一些应用，是无法用简单的 Map 和 Reduce 来完成的。
- 开发者自己管理作业之间的依赖关系。一个作业（Job）只包含 Map 和 Reduce 两个阶段，通常的实际应用问题需要大量的作业进行协作才能顺利解决，这些作业之间往往存在复杂的依赖关系，但是，MapReduce 框架本身并没有提供相关的机制对这些依赖关系进行有效管理，只能由开发者自己管理。
- 难以看到程序整体逻辑。用户的处理逻辑都隐藏在代码细节中，没有更高层次的抽象机制对程序整体逻辑进行设计，这就给代码理解和后期维护带来了障碍。
- 执行迭代操作效率低。对于一些大型的机器学习、数据挖掘任务，往往需要多轮迭代才能得到结果。采用 MapReduce 实现这些算法时，每次迭代都是一次执行 Map、Reduce 任务的过程，这个过程的数据来自分布式文件系统 HDFS，本次迭代的处理结果也被存放到 HDFS 中，继续用于下一次迭代过程。反复读写 HDFS 文件中的数据，大大降低了迭代操作的效率。
- 资源浪费。在 MapReduce 框架涉及中，Reduce 任务需要等待所有 Map 任务都完成后才可以开始，造成了不必要的资源浪费。
- 实时性差。只适用于离线批数据处理，无法支持交互式数据处理、实时数据处理。

15.1.2 针对 Hadoop 的改进与提升

针对 Hadoop1.0 存在的局限和不足，在后续发展过程中，Hadoop 对 MapReduce 和 HDFS 的许多方面做了有针对性的改进提升（如表 15-1 所示），同时，在 Hadoop 生态系统中也融入了更多的新产品，来更好地弥补 Hadoop1.0 中存在的问题，比较有代表性的产品包括 Pig、Spark、OOzie、Tez、Kafka 等（表 15-2 所示）。

表 15-1 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

表 15-2 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 Job 依赖关系管理机制，需要用户自己处理 Job 之间的依赖关系
Tez	支持 DAG 作业的计算框架，对 Job 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

在下面的内容中，将首先介绍 HDFS 的新特性（包括 HDFS HA 和 HDFS Federation），然后介绍 Hadoop 中新的资源管理框架 YARN，它是在 MapReduce1.0 框架基础之上发展起来的，最后，介绍 Hadoop 生态系统中具有代表性的几个组件及其解决的问题。

15.2HDFS2.0 的新特性

15.2.1HDFS HA

对于分布式文件系统 HDFS 而言，名称节点（NameNode）是系统的核心节点，存储了

各类元数据信息，并负责管理文件系统的命名空间和客户端对文件的访问。但是，在 HDFS1.0 中，只存在一个名称节点（NameNode），一旦这个唯一的名称节点发生故障，就会导致整个集群变得不可用，这就是常说的“单点故障问题”。虽然 HDFS1.0 中存在一个“第二名称节点（Secondary NameNode）”，但是，第二名称节点并不是名称节点的备用节点，它与名称节点有着不同的职责（二者的服务可以运行在一台机器上），其主要功能是周期性地从名称节点获取命名空间镜像文件（FsImage）和修改日志（EditLog），进行合并后再发送给名称节点，替换掉原来的 FsImage，以防止日志文件 EditLog 过大，导致名称节点失败恢复时消耗过多时间。合并后的命名空间镜像文件 FsImage 在第二名称节点中也保存一份，当名称节点失效的时候，它首先会把自己的 FsImage 和 EditLog 进行恢复，如果自己的 FsImage 发生丢失，还可以使用第二名称节点中的 FsImage 进行恢复。

由于第二名称节点无法提供“热备份”功能，即在名称节点发生故障的时候，系统无法借助于第二名称节点继续提供服务，仍然需要进行停机恢复，因此，HDFS1.0 的设计是存在单点故障问题的。为了解决单点故障问题，HDFS2.0 采用了 HA（Hign Aailability）架构。如图 15-1 所示，在一个典型的 HA 集群中，一般设置两个名称节点，其中一个名称节点处于“活跃（Active）”状态，另一个处于“待命（Standby）”状态。其中。处于活跃状态的名称节点负责对外处理所有客户端的请求，而处于待命状态的名称节点则作为备用节点，保存了足够多的系统元数据，当名称节点出现故障时提供快速恢复能力。也就是说，在 HDFS HA 中，处于待命状态的名称节点提供了“热备份”，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，不会影响到系统的正常对外服务。

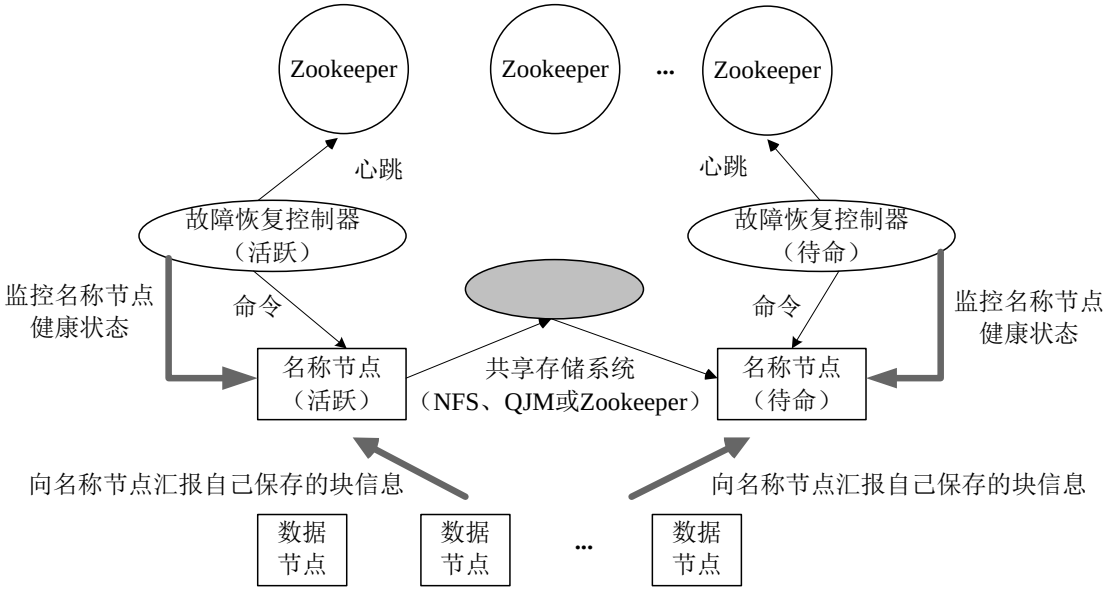


图 15-1 HDFS HA 架构

由于待命名称节点是活跃名称节点的“热备份”，因此，活跃名称节点的状态信息必须实时同步到待命名称节点。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，比如 NFS（Network File System）、QJM（Quorum Journal Manager）或者 Zookeeper。活跃名称节点将更新数据写入到共享存储系统，待命名称节点会一直监听该系统，一旦发现有新的写入，就立即从公共存储系统中读取这些数据并加载到自己的内存中，从而保证与活跃名称节点状态完全同步。

此外，名称节点中保存了数据块（block）到实际存储位置的映射信息，即每个数据块是由哪个数据节点存储的。当一个数据节点加入 HDFS 集群时，它会把自己所包含的数据

块列表告知给名称节点，此后会定期执行这种告知操作，以确保名称节点的块映射是最新的。因此，为了实现故障时的快速切换，必须保证待命名称节点一直包含最新的集群中各个块的位置信息。为了做到这一点，需要给数据节点配置两个名称节点的地址（即活跃名称节点和待命名称节点），并把块的位置信息和心跳信息同时发送到这两个名称节点。为了防止出现“两个管家”现象，HA 还要保证任何时刻都只有一个名称节点处于活跃状态，否则，如果有两个节点处于活跃状态，HDFS 集群中出现“两个管家”，就会导致数据丢失或者其他异常，这个任务是由 Zookeeper 来实现的，Zookeeper 可以确保任意时刻只有一个名称节点提供对外服务。

15.2.2HDFS Federation

1.HDFS1.0 中存在的问题

HDFS1.0 采用单名称节点的设计，不仅会带来单点故障问题，还存在可扩展性、性能和隔离性等问题。在可扩展性方面，名称节点把整个 HDFS 文件系统中的元数据信息都保存在自己的内存中，HDFS1.0 中只有一个名称节点，不可以水平扩展，而单个名称节点的内存空间是有上限的，这限制了系统中数据块、文件和目录的数目。是否可以通过纵向扩展的方式（即为单个名称节点增加更多的 CPU、内存等资源）解决这个问题呢？答案是否定的。纵向扩展带来的第一个问题就是，会带来过长的系统启动时间，比如，一个具有 50GB 内存的 HDFS 启动一次大概需要消耗 30 分钟到 2 个小时，单纯增大内存空间，只会让系统启动时间变得更长。其次，当在内存空间清理时发生错误，就会导致整个 HDFS 集群宕机。

在系统整体性能方面，整个 HDFS 文件系统的性能会受限于单个名称节点的吞吐量。在隔离性方面，单个名称节点难以提供不同程序之间的隔离性，一个程序可能会影响其他运行的程序（比如一个程序消耗过多资源导致其他程序无法顺利运行）。HDFS HA 虽然提供了两个名称节点，但是，在某个时刻，也只会有一个名称节点处于活跃状态，另一个则处于待命状态，因而，HDFS HA 在本质上还是单名称节点，只是通过“热备份”设计方式解决了单点故障问题，并没有解决可扩展性、系统性能和隔离性等三个方面的问题。

2.HDFS Federation 的设计

HDFS Federation 可以很好解决上述三个方面的问题。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，相互之间是联盟关系，不需要彼此协调。HDFS Federation 并不是真正的分布式设计，但是，采用这种简单的“联合”设计方式，在实现和管理复杂性方面要远低于真正的分布式设计，而且可以快速满足需求。在兼容性方面，HDFS Federation 具有良好的向后兼容性，可以无缝地支持单名称节点架构中的配置，所以，原有针对单名称节点的部署配置，不需要做任何修改就可以继续工作。

HDFS Federation 中的名称节点提供了命名空间和块管理功能。如图 15-2 所示，在 HDFS Federation 中，所有名称节点会共享底层的数据节点存储资源。每个数据节点要向集群中所有的名称节点注册，并周期性地向名称节点发送“心跳”和块信息，报告自己的状态，同时也会处理来自名称节点的指令。

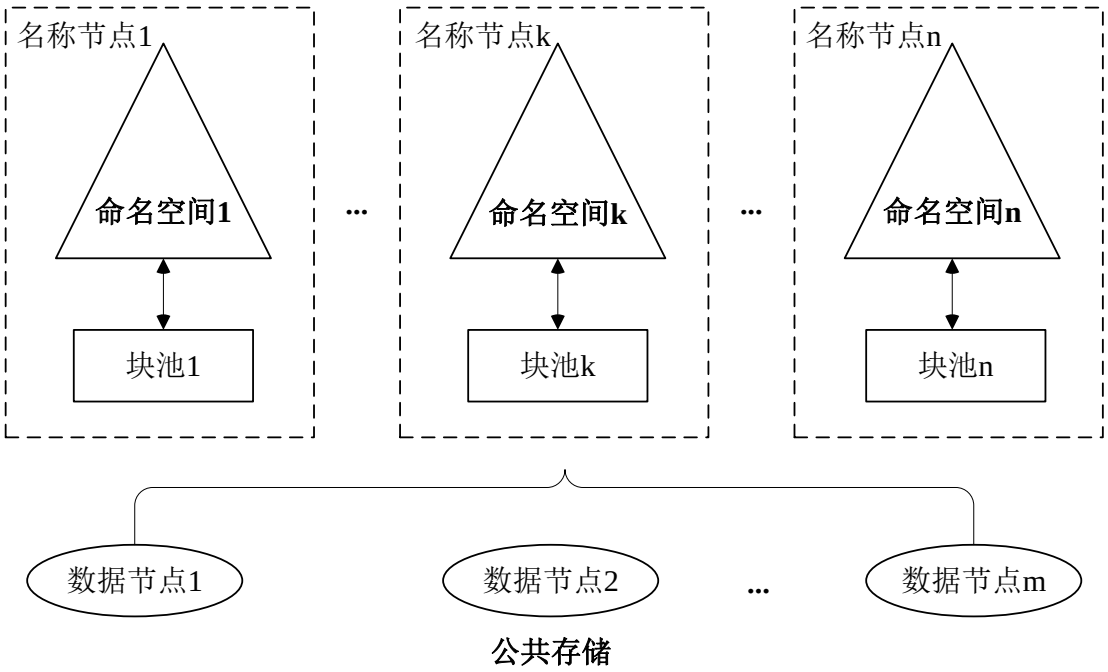


图 15-2 HDFS Federation 架构

HDFS1.0 只有一个命名空间，这个命名空间使用底层数据节点全部的块。与 HDFS1.0 不同的是，HDFS Federation 拥有多个独立的命名空间，其中每一个命名空间管理属于自己的一组块，这些属于同一个命名空间的块构成一个“块池”（block pool）。每个数据节点会为多个块池存储块。可以看出，数据节点是一个物理概念，而块池则属于逻辑概念，一个块池是一组块的逻辑集合，块池中的各个块实际上是存储在各个不同的数据节点中的。因此，HDFS Federation 中的一个名称节点失效，也不会影响到与它相关的数据节点继续为其他名称节点提供服务。

3. HDFS Federation 的访问方式

对于 HDFS Federation 中的多个命名空间，可以采用客户端挂载表（Client Side Mount Table）方式进行数据共享和访问。如图 15-3 所示，每个阴影三角形代表一个独立的命名空间，上方空白三角形表示从客户方向去访问下面子命名空间。客户可以访问不同的挂载点来访问不同的子命名空间。这就是 HDFS Federation 中命名空间管理的基本原理，即把各个命名空间挂载到全局“挂载表”（mount-table）中，实现数据全局共享；同样的命名空间挂载到个人的挂载表中，就成为应用程序可见的命名空间。

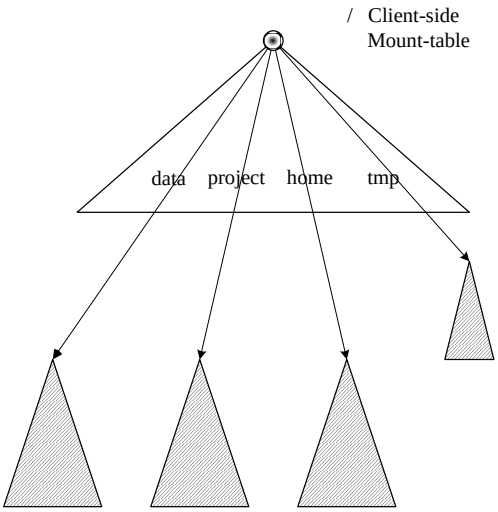


图 15-3 客户端挂载表方式访问多个命名空间

4.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 集群可扩展性。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，以应对名称节点宕机后对业务产生的影响。

15.3 新一代资源管理调度框架 YARN

15.3.1 MapReduce1.0 的缺陷

MapReduce1.0 采用 Master/Slave 架构设计（如图 15-4 所示），包括一个 JobTracker 和若干个 TaskTracker，前者负责作业的调度和资源的管理，后者负责执行 JobTracker 指派的具体任务。这种架构设计具有一些很难克服的缺陷，具体如下：

（1）存在单点故障。由 JobTracker 负责所有 MapReduce 作业的调度，而系统中只有一个 JobTracker，因此会存在单点故障问题，即这个唯一的 JobTracker 出现故障，就会导致系统不可用；

（2）JobTracker“大包大揽”导致任务过重。JobTracker 既要负责作业的调度和失败恢复，又要负责资源管理分配。执行过多的任务，需要消耗大量的资源，例如，当存在非常多的 MapReduce 任务时，JobTracker 需要巨大的内存开销，这也潜在地增加了 JobTracker 失败的风险。正因如此，业内普遍总结出 MapReduce1.0 支持主机数目的上限为 4000 个；

（3）容易出现内存溢出。在 TaskTracker 端，资源的分配并不考虑 CPU、内存的实际

使用情况，而只是根据 MapReduce 任务的个数来分配资源，当两个具有较大内存消耗的任务被分配到同一个 TaskTracker 上时，很容易发生内存溢出的情况。

（4）资源划分不合理。资源（CPU、内存）被强制等量划分成多个 slot，slot 又进一步划分为 Map slot 和 Reduce slot 两种，分别供 Map 任务和 Reduce 任务使用，彼此之间不能使用分配给对方的 slot，也就是说，当 Map 任务已经用完 Map slot 时，即使系统中还有大量剩余的 Reduce slot，也不能拿来运行 Map 任务，反之亦然。这就意味着，当系统中只存在单一 Map 任务或 Reduce 任务时，会造成资源的浪费。

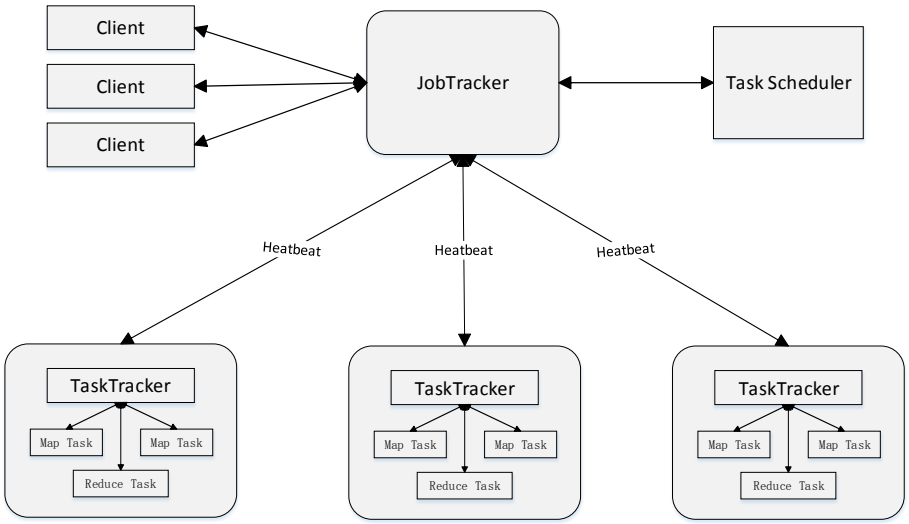


图 15-4 MapReduce1.0 体系结构

15.3.2 YARN 设计思路

为了克服 MapReduce1.0 版本的缺陷，Hadoop2.0 以后的版本，对其核心子项目 MapReduce1.0 的体系结构进行了重新设计，生成了 MapReduce2.0 和 YARN（Yet Another Resource Negotiator）。如图 15-5 所示，基本思路就是“放权”，即不让 JobTracker 这一个组件承担过多的功能，把原 JobTracker 三大功能（资源管理、任务调度和任务监控）进行拆分，分别交给不同的新组件去处理。重新设计后得到的 YARN 包括 ResourceManager、ApplicationMaster 和 NodeManager，其中，由 ResourceManager 负责资源管理，由 ApplicationMaster 负责任务调度和监控，由 NodeManager 负责执行原 TaskTracker 的任务。通过这种“放权”的设计，大大降低了 JobTracker 的负担，提升了系统运行的效率和稳定性。

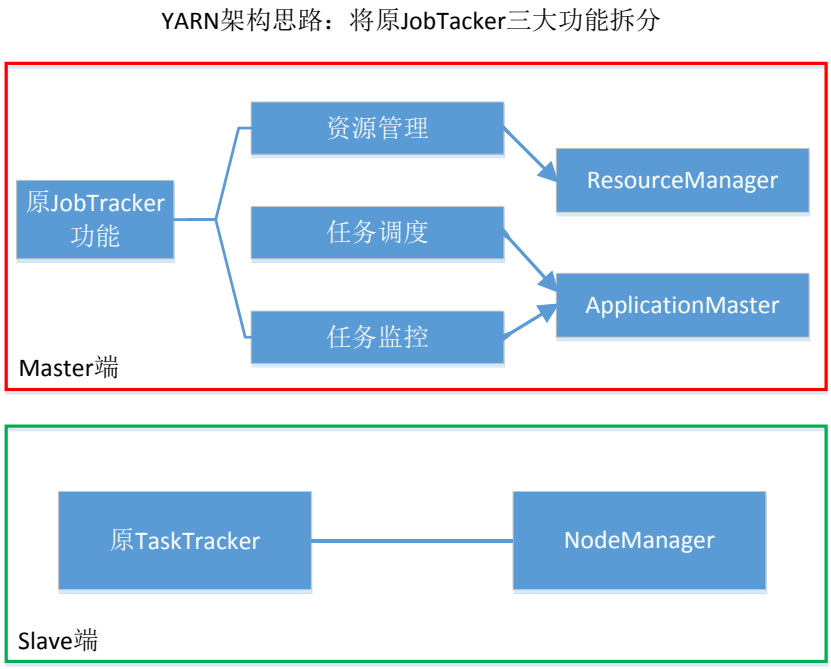


图 15-5 YARN 架构设计思路

在 Hadoop1.0 中，其核心子项目 MapReduce1.0 既是一个计算框架，也是一个资源管理调度框架。到了 Hadoop2.0 以后，MapReduce1.0 中的资源管理调度功能，被单独分离出来形成了 YARN，它是一个纯粹的资源管理调度框架，而不是一个计算框架；而被剥离了资源管理调度功能的 MapReduce 框架就变成了 MapReduce2.0，它是运行在 YARN 之上的一个纯粹的计算框架，不再自己负责资源调度管理服务，而是由 YARN 为其提供资源管理调度服务。

15.3.3 YARN 体系结构

如图 15-6 所示，YARN 体系结构中包含了三个组件：ResourceManager、ApplicationMaster 和 NodeManager，表 15-3 给出了 YARN 各个组件的功能。

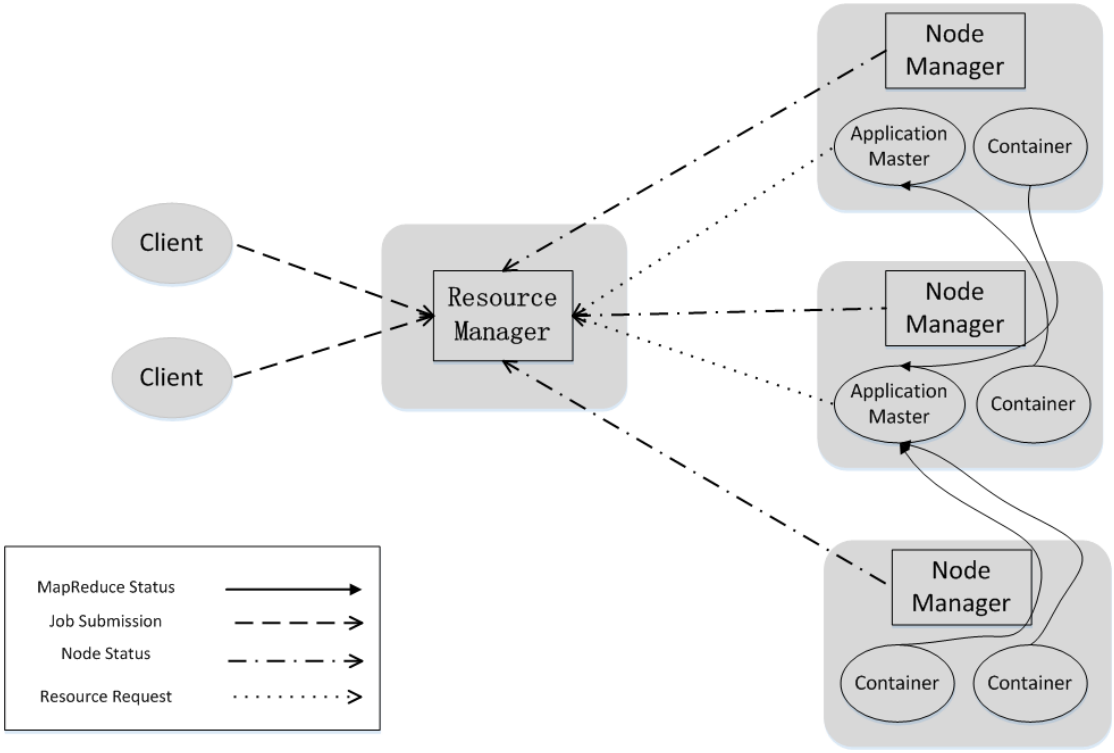


图 15-6 YARN 体系结构

表 15-3 YARN 各个组件的功能

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

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）。调度器主要负责资源管理和分配，不再负责跟踪和监控应用程序的执行状态，也不负责执行失败恢复，因为这些任务都已经交给 ApplicationMaster 组件来负责。调度器接收来自 ApplicationMaster 的应用程序资源请求，并根据容量、队列等限制条件（如每个队列分配一定的资源，最多执行一定数量的作业等），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”。在 MapReduce1.0 中，资源分配的单位是 slot，而在 YARN 中，是以容器（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。同时，在 YARN 中，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。应用程序管理器（Applications Manager）负责系统中所

有应用程序的管理工作，主要包括应用程序提交、与调度器协商资源以启动 ApplicationMaster、监控 ApplicationMaster 运行状态并在失败时重新启动等。

在 Hadoop 平台上，用户的应用程序是以作业（Job）的形式提交的，然后，一个作业会被分解成多个任务（包括 Map 任务和 Reduce 任务）进行分布式执行。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。

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，并以“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，同时，它还要接收来自 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。

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。如图 15-4 所示，YARN 的 ResourceManager 组件和 HDFS 的名称节点（NameNode）部署在一个节点上，YARN 的 ApplicationMaster 和 NodeManager 是和 HDFS 的数据节点（DataNode）部署在一起的。YARN 中的容器代表了 CPU、内存、网络等计算资源，它也是和 HDFS 的数据节点一起的。

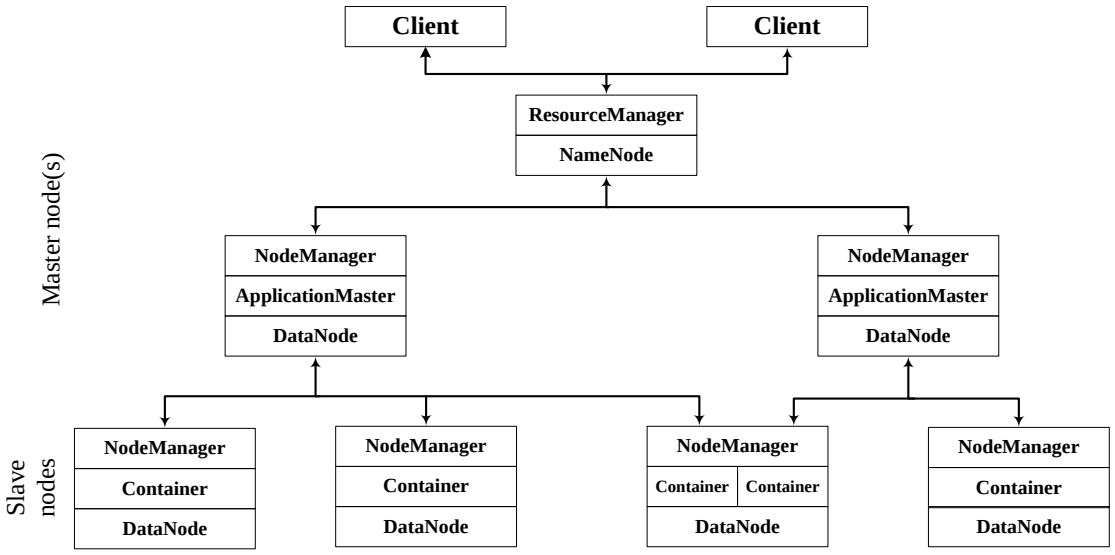


图 15-7 YARN 和 Hadoop 平台其他组件的统一部署

15.3.4 YARN 工作流程

如图 15-8 所示，在 YARN 框架中执行一个 MapReduce 程序时，从提交到完成需要经历如下 8 个步骤：

- 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。接到客户端应用程序请求后，ResourceManager 里面的调度器会为应用程序分配一个容器。同时，ResourceManager 的应用程序管理器会与该容器所在的 NodeManager 通信，为该应用程序在该容器中启动一个 ApplicationMaster（即图 15-5 中的“MR App Mstr”）。
- 步骤 3：ApplicationMaster 被创建后会首先向 ResourceManager 注册，从而使得用户可以通过 ResourceManager 来直接查看应用程序的运行状态。接下来的步骤 4~7 是具体的应用程序执行步骤。
- 步骤 4：ApplicationMaster 采用轮询的方式通过 RPC 协议向 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，一旦 ApplicationMaster 申请到资源后，就会与该容器所在的 NodeManager 进行通信，要求它启动任务。
- 步骤 6：当 ApplicationMaster 要求容器启动任务时，它会为任务设置好运行环境（包括环境变量、JAR 包、二进制程序等），然后将任务启动命令写到一个脚本中，最后通过在容器中运行该脚本来启动任务。
- 步骤 7:各个任务通过某个 RPC 协议向 ApplicationMaster 汇报自己的状态和进度，让 ApplicationMaster 可以随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务。
- 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。若 ApplicationMaster 因故失败，ResourceManager 中的应用程序管理器会监测到失败的情形，然后将其重新启动，直到所有的任务执行完毕。

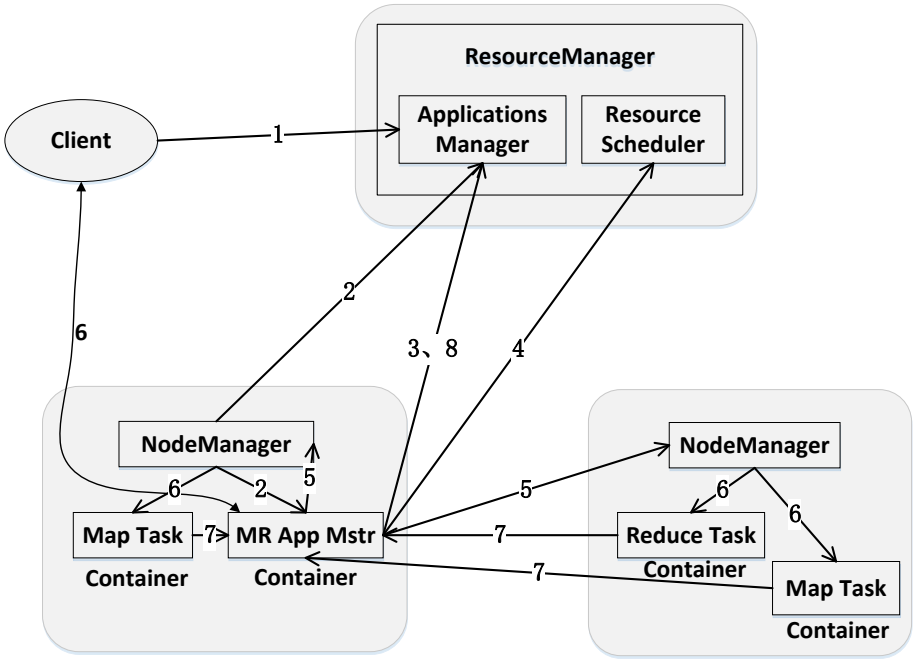


图 15-8 YARN 的工作流程

15.3.5YARN 框架与 MapReduce1.0 框架的对比分析

从 MapReduce1.0 框架发展到 YARN 框架，客户端并没有发生变化，其大部分调用 API 及接口都保持兼容，因此，原来针对 Hadoop1.0 开发的代码不用做大的改动，就可以直接放到 Hadoop2.0 平台上运行。

在 MapReduce1.0 框架中的 JobTracker 和 TaskTracker，在 YARN 框架中变成了三个组件，即 ResourceManager、ApplicationMaster 和 NodeManager。ResourceManager 要负责调度、启动每一个作业所属的 ApplicationMaster，监控 ApplicationMaster 运行状态并在失败时重新启动，而作业里面的不同任务的调度、监控、重启等，则不再由 ResourceManager 负责，而是交给专门为某个作业启动的 ApplicationMaster 来负责，ApplicationMaster 要负责一个作业生命周期内的所有工作，也就是说，它承担了 MapReduce1.0 中 JobTracker 的“作业监控”的功能。

总体而言，YARN 相对于 MapReduce1.0 来说具有以下优势：

（1）大大减少了承担中心服务功能的 ResourceManager 的资源消耗。MapReduce1.0 中的 JobTracker 需要同时承担资源管理、任务调度和任务监控等三大功能，而 YARN 中的 ResourceManager 只需要负责资源管理，需要消耗大量资源的任务调度和监控重启工作，则交由 ApplicationMaster 来完成。由于每个作业都有与之关联的独立的 ApplicationMaster，所以，系统中存在多个作业时，就会同时存在多个 ApplicationMaster，这就实现了监控任务的分布化，不再像 MapReduce1.0 那样监控任务只集中在一个 JobTracker 上；

（2）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，默认类型是 MapReduce。因为，YARN 中的 ApplicationMaster 是可变更的，针对不同的计算框架，用户可以采用任何编程语言自己编写服务于该计算框架的 ApplicationMaster，比如，可以编写一个面向 MapReduce 计算框架的 ApplicationMaster，从而使得 MapReduce 计算框架可以运行在 YARN 框架之上。同理，还可以编写面向 Spark、Storm 等计算框架的 ApplicationMaster，从而使得 Spark、Storm 等计算

框架也可以运行在 YARN 框架之上。

（3）YARN 中的资源管理比 MapReduce1.0 更加高效。YARN 采用容器为单位进行资源管理和分配，而不是以 slot 为单位，避免了 MapReduce1.0 中 slot 闲置浪费的情况，大大提高了资源的利用率。

15.3.6 YARN 的发展目标

YARN 的提出，并非仅仅为了解决 MapReduce1.0 框架中存在的缺陷，实际上，YARN 有着更加“宏伟”的发展构想，即发展成为集群中统一的资源管理调度框架，在一个集群中为上层的各种计算框架提供统一的资源管理调度服务。

在一个企业当中，会同时存在各种不同的业务应用场景，各自的数据处理需求截然不同，为了满足各种应用场景的不同数据处理需求，就需要采用不同的计算框架，比如使用 MapReduce 实现离线批处理，使用 Impala 实现实时交互式查询分析，使用 Storm 实现流式数据实时分析，使用 Spark 实现迭代计算等。而这些产品通常来自不同的开发团队，具有各自的资源调度管理机制，于是，为了避免不同类型应用之间互相干扰，企业就需要把内部的服务器拆分成多个集群，分别安装运行不同的计算框架，即“一个框架一个集群”，一个集群运行 MapReduce，一个运行 Spark，还有的运行 Storm 或者其他计算框架。企业内部服务器集群被分拆成不同的独立小集群运行，带来的一个显而易见的问题就是，集群资源利用率低，因为，在某个时刻，不同集群的负载水平分布很不均匀，有些小集群可能处于极度繁忙状态，而另外一些集群可能处于闲置浪费状态，由于各个小集群之间彼此隔离，因而繁忙小集群的负载无法分发到空闲小集群上执行，这就导致了服务器资源的浪费。另外，不同集群之间无法直接共享数据，造成集群间大量的数据传输开销，同时需要多个管理员维护不同的集群，大大增加了运维成本。

因此，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（如图 15-9 所示，包括 MapReduce、Tez、HBase、Storm、Giraph、Spark、OpenMPI 等），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。通过这种方式，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，同时，不同计算框架可以共享底层存储，在一个集群上集成多个数据集，使用多个计算框架来访问这些数据集，从而避免了数据集跨集群移动，最后，这种部署方式也大大降低了企业运维成本。

目前，可以运行在 YARN 之上的计算框架包括离线批处理框架 MapReduce、内存计算框架 Spark、流计算框架 Storm 和 DAG 计算框架 Tez 等。和 YARN 一样提供类似功能的其他资源管理调度框架还包括 Mesos、Torca、Corona、Borg 等。

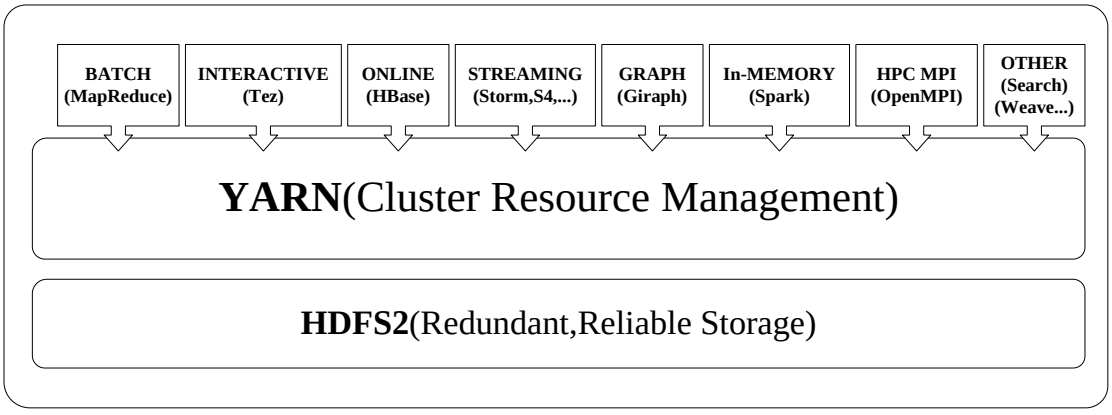


图 15-9 在 YARN 上部署各种计算框架

15.4Hadoop 生态系统中具有代表性的功能组件

15.4.1Pig

Pig 是 Hadoop 生态系统的一个组件，提供了类似 SQL 的 Pig Latin 语言（包含 Filter、GroupBy、Join、OrderBy 等操作，同时也支持用户自定义函数），允许用户通过编写简单的脚本来实现复杂的数据分析，而不需要编写复杂的 MapReduce 应用程序，Pig 会自动把用户编写的脚本转换成 MapReduce 作业在 Hadoop 集群上运行，而且具备对生成的 MapReduce 程序进行自动优化的功能，所以，用户在编写 Pig 程序的时候，不需要关心程序的运行效率，这就大大减少了用户编程时间。因此，通过配合使用 Pig 和 Hadoop，在处理海量数据时就可以实现事半功倍的效果，比使用 Java、C++等语言编写 MapReduce 程序的难度要小很多，并且用更少的代码量实现了相同的数据处理分析功能。

Pig 可以加载数据、表达转换数据以及存储最终结果，因此，在企业实际应用中，Pig 通常用于 ETL（Extraction、Transformation、Loading）过程，即来自各个不同数据源的数据被收集过来以后，采用 Pig 进行统一加工处理，然后加载到数据仓库 Hive 中，由 Hive 实现对海量数据的分析。需要特别指出的是，每种数据分析工具都有一定的局限性，Pig 的设计和 MapReduce 一样，都是面向批处理的，因此，Pig 并不适合所有的数据处理任务，特别是当需要查询大数据集中的一小部分数据时，Pig 仍然需要对整个或绝大部分数据集进行扫描，因此，实现性能不会很好。

图 15-10 Pig 在企业数据分析系统中的作用

Pig 语句通常按照如下的格式来编写：

- 通过 LOAD 语句从文件系统读取数据；
- 通过一系列“转换”语句对数据进行处理；
- 通过一条 STORE 语句把处理结果输出到文件系统中，或者使用 DUMP 语句把处理结果输出到屏幕上。

下面是一个采用 Pig Latin 语言编写的应用程序实例，实现对用户访问网页情况的统计分析。

```
visits= load ‘/data/visits’ as (user, url, time); //导入用户访问日志 visits
gVisits = group visits by url;//根据网址 url 对用户访问数据进行分组
visitCounts = foreach gVisits generate url, count(visits);//对于每个 url，计算用户访问量
//上面语句执行后得到的表的结构 visitCounts(url,visits)
urlInfo = load ‘/data/urlInfo’ as (url, category, pRank);//导入用户信息
visitCounts = join visitCounts by url, urlInfo by url;//对 visitCounts 和 urlInfo 表进行连接操作
//上面语句执行后得到的连接结果表的结构 visitCounts(url,visits,category,pRank)
gCategories = group visitCounts by category;//根据用户类别进行分组
topUrls = foreach gCategories generate top(visitCounts,10);//每个用户类别取访问量 TOP10
```

```
store topUrls into '/data/topUrls';//把访问量排名信息写入 topUrls
```

对于上述 Pig Latin 脚本，Pig 会自动转换成如下 MapReduce 任务，如图 15-11 所示，图中，group by 和 join 操作都“跨越”了 Map 和 Reduce 两个阶段，这是因为，根据“第 14 章基于 Hadoop 的数据库仓库 Hive”可以知道，group 和 join 操作都涉及到 Shuffle 过程，根据“第 7 章 MapReduce”可以知道，Shuffle 过程包含了 Map 端和 Reduce 端，所以，图中表示 group by 和 join 操作的矩形框，和 Map 和 Reduce 两个阶段都存在重叠区域。

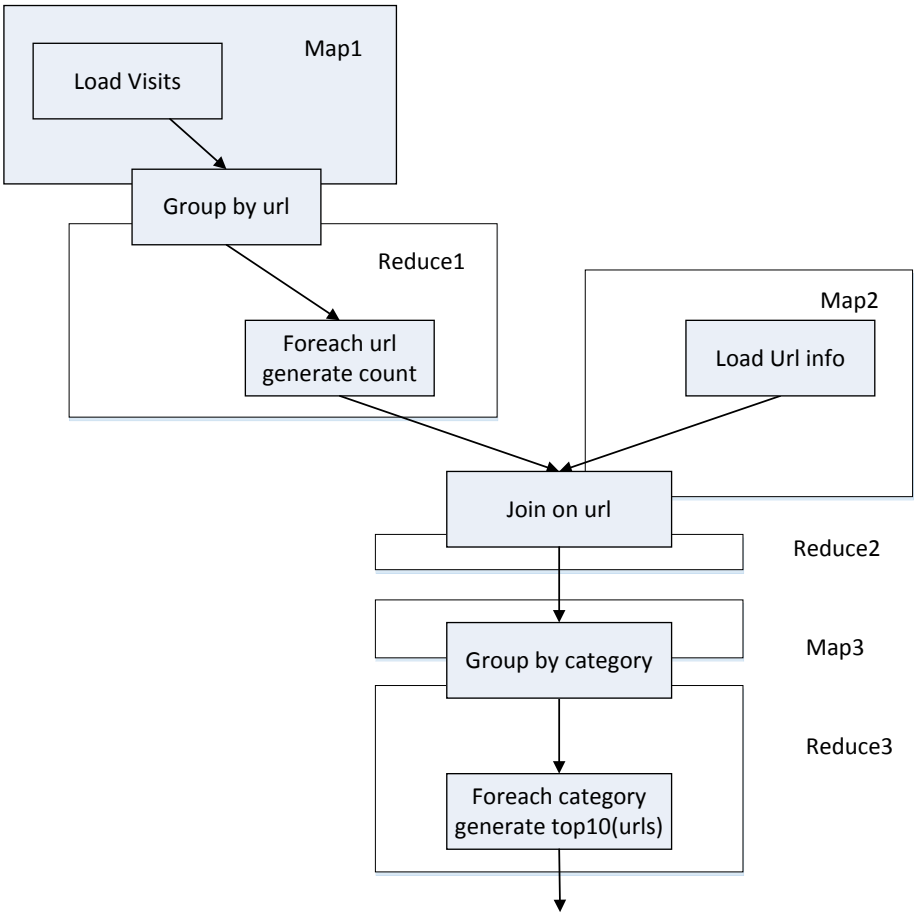


图 15-11 从 Pig Latin 脚本转化得到的 MapReduce 作业

当数据查询只面向相关技术人员，并且属于即时性的数据处理需求的时候，比较适合采用 Pig 编写一个脚本完成快速运行处理，从而避免创建表等相关操作。目前，Pig 在很多公司得到了应用，在 Yahoo!中 90%以上的 MapReduce 作业是 Pig 生成的，Twitter 公司 80%以上的 MapReduce 作业是 Pig 生成的，Linkedin 公司中的大部分 MapReduce 作业是 Pig 生成的。Pig 其他主要用户还包括 Salesforce、Nokia、AOL 和 comScore 等。

15.4.2Tez

Tez 是 Apache 开源的支持 DAG 作业的计算框架，直接源于 MapReduce 框架，核心思想是将 Map 和 Reduce 两个操作进一步进行拆分，即 Map 被拆分成 Input、Processor、Sort、Merge 和 Output，Reduce 被拆分成 Input、Shuffle、Sort、Merge、Processor 和 Output 等，经过分解后的这些元操作可以进行自由任意组合产生新的操作，然后经过一些控制程序组装后就可形成一个大的 DAG 作业。

通过 DAG 作业的方式运行 MapReduce 作业，提供了程序运行的整体处理逻辑，就可以

去除工作流当中多余的 Map 阶段，减少不必要的操作，提升数据处理的性能。Hortonworks 把 Tez 应用到数据仓库 Hive 的优化中，使得性能提升了约 100 倍。

这里以一个具体实例说明 Tez 的优化效果。假设有三个表 a、b 和 c，表 a 的属性包括 state、id 和 itemId，表 b 的属性包括 id 和其他属性，表 c 的属性包括 itemId 和 price。下面是数据仓库 Hive 中执行数据分析的一段 HiveQL 语句，其功能是对三个表在公共属性上进行连接操作，并根据 state 属性进行分组，并统计每个分组的元组个数和平均价格。

```
SELECT a.state, COUNT(*), AVERAGE(c.price)
FROM a
JOIN b ON(a.id = b.id)
JOIN c ON(a.itemId = c.itemId)
GROUP BY a.state
```

图 15-12 给出了上述 HiveQL 语句在 MapReduce 和 Tez 中的执行情况对比。在 MapReduce 框架中执行时，需要采用三个 MapReduce 作业才能完成，每个作业完成以后，都需要写入到分布式文件系统 HDFS 中，供下一个作业读取，带来了较大的处理延迟；而在 Tez 框架中执行，只需要一个 Tez 作业就可以完成，处理全过程不需要把数据写入到 HDFS，可以直接在不同的 Map 和 Reduce 任务之间传输数据，同时，也减少了一些不必要的 MapReduce 操作，从而大大提升了程序执行效率。

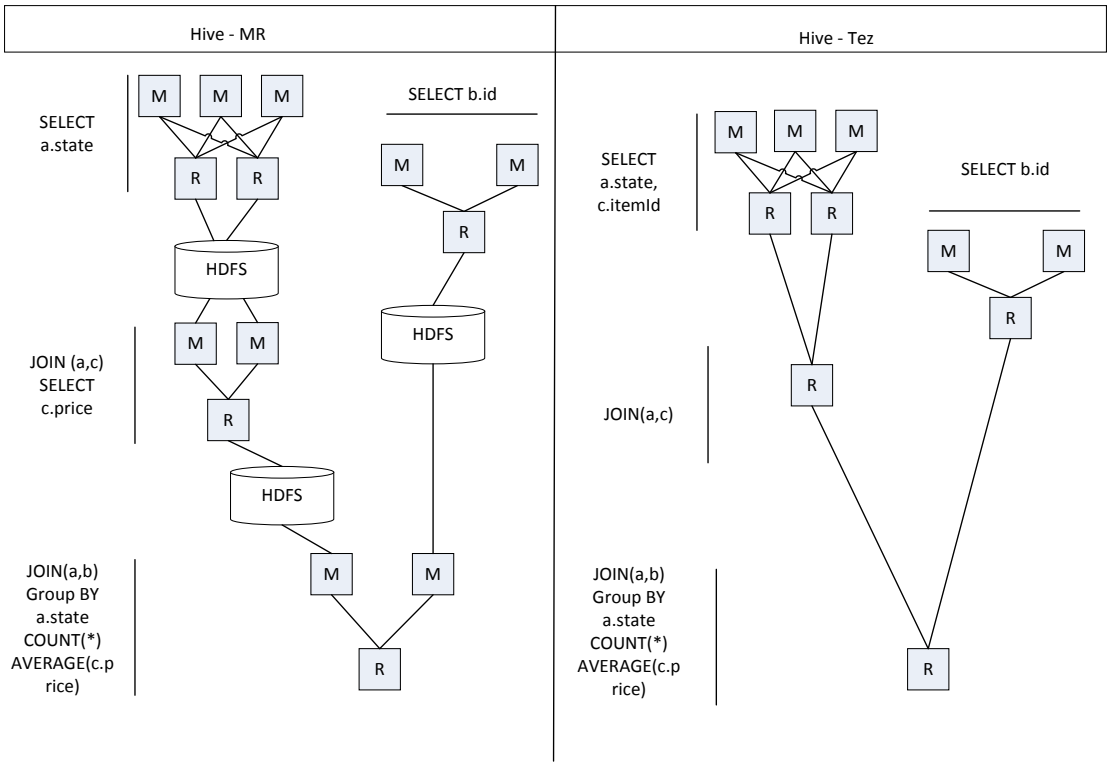


图 15-12 HiveQL 语句在 MapReduce 和 Tez 中的执行情况对比

在 Hadoop2.0 生态系统中，MapReduce、Hive、Pig 等计算框架，都需要最终以 MapReduce 任务的形式执行数据分析，因此，Tez 框架可以发挥重要的作用。如图 15-13 所示，可以让 Tez 框架运行在 YARN 框架之上，然后让 MapReduce、Pig 和 Hive 等计算框架运行在 Tez 框架之上，从而借助于 Tez 框架实现对 MapReduce、Pig 和 Hive 等的性能优化，更好地解决现有 MapReduce 框架在迭代计算（如 PageRank 计算）和交互式计算方面存在的问题。

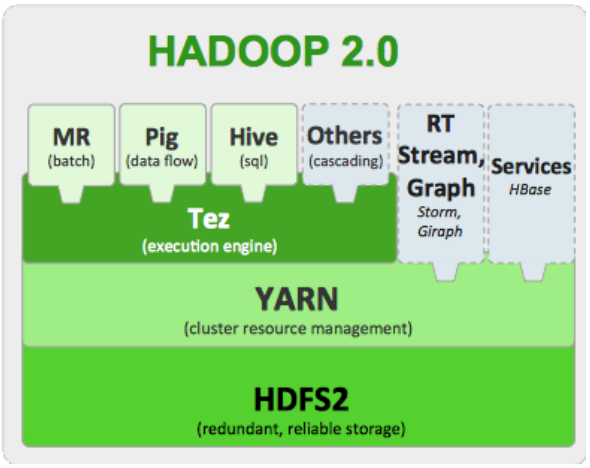


图 15-13 Tez 框架在 Hadoop 生态系统中的作用

可以看出，Tez 在解决 Hive、Pig 延迟大、性能低等问题的思路，是和那些支持实时交互式查询分析的产品（如 Impala、Dremel 和 Drill 等）是不同的。Impala、Dremel 和 Drill 的解决问题思路是抛弃 MapReduce 计算框架，不再将类似 SQL 语句的 HiveQL 或者 Pig 语句翻译成 MapReduce 程序，而是采用与商用并行关系数据库类似的分布式查询引擎，可以直接从 HDFS 或者 HBase 中用 SQL 语句查询数据，而不需要把 SQL 语句转化成 MapReduce 任务来执行，从而大大降低了延迟，很好地满足了实时查询的要求。但是，Tez 则不同，比如，针对 Hive 数据仓库进行优化的“Tez+Hive”解决方案，仍采用 MapReduce 计算框架，但是对 DAG 的作业依赖关系进行了裁剪，并将多个小作业合并成一个大作业，这样，不仅计算量减少了，而且写 HDFS 次数也会大大减少。

15.4.3Spark

Hadoop 虽然已成为大数据技术的事实标准，但其本身还存在诸多缺陷，最主要的缺陷是其 MapReduce 计算模型延迟过高，无法胜任实时、快速计算的需求，因而只适用于离线批处理的应用场景。在 MapReduce 中，每次执行时都需要从磁盘读取数据，并且在计算完成后需要将中间结果写入到磁盘中，IO 开销较大；而且，在前一个任务执行完成之前，其他任务无法开始，难以胜任复杂、多阶段的计算任务。

Spark 最初诞生于伯克利大学的 APM 实验室，是一个可应用于大规模数据处理的快速、通用引擎，如今是 Apache 软件基金会下的顶级开源项目之一。Spark 使用简练、优雅的 Scala 语言编写，基于 Scala 提供了交互式的编程体验。Spark 在借鉴 Hadoop MapReduce 优点的同时，很好地解决了 MapReduce 所面临的问题，比如，Spark 提供了内存计算，中间结果直接放到内存中，带来了更高的迭代运算效率；Spark 基于 DAG 的任务调度执行机制，要优于 MapReduce 的迭代执行机制。因而，Spark 更适合于迭代运算比较多的数据挖掘与机器学习运算。当前，Spark 正以其结构一体化、功能多元化的优势，逐渐成为当今大数据领域最热门的大数据计算平台。

15.4.4Kafka

Kafka 是由 LinkedIn 公司开发的一种高吞吐量的分布式发布订阅消息系统，用户通过 Kafka 系统可以发布大量的消息，同时也能实时订阅消费消息。Kafka 设计的初衷是构建一

个可以处理海量日志、用户行为和网站运营统计等的数据处理框架。为了满足上述应用需求，就需要同时提供实时在线处理的低延迟和批量离线处理的高吞吐量。现有的一些消息队列框架，通常设计了完备的机制来保证消息传输的可靠性，但是，由此会带来较大的系统负担，在批量处理海量数据时无法满足高吞吐率的要求；另外有一些消息队列框架则被设计成实时消息处理系统，虽然可以带来很高的实时处理性能，但是在面对批量离线场合时却无法提供足够的持久性，即可能发生消息丢失。同时，在大数据时代涌现的新的日志收集处理系统（Flume、Scribe 等）往往更擅长批量离线处理，而不能较好地支持实时在线处理。相对而言，Kafka 可以同时满足在线实时处理和批量离线处理。

最近几年，Kafka 在大数据生态系统中开始扮演越来越重要的作用，在 Uber、Twitter、Netflix、LinkedIn、Yahoo、Cisco、Goldman Sachs 等公司得到了大量的应用。目前，在很多公司的大数据平台中，Kafka 通常扮演数据交换枢纽的角色。

传统的关系数据库一直是企业关键业务系统的首选数据库产品，能够较好地满足企业对数据一致性和高效复杂查询的需求。但是，关系数据库只能支持规范的结构化数据存储，无法有效应对各种不同类型的数据，比如各种非结构化的日志记录、图结构数据等，同时面对海量大规模数据也显得“捉襟见肘”。因此，关系数据库无法实现“一种产品满足所有应用场景”。在这样的大背景下，各种专用的分布式系统纷纷涌现，包括离线批处理系统（如 MapReduce、HDFS）、NoSQL 数据库（如 Redis、MongoDB、HBase、Cassandra）、流计算框架（如 Storm、S4、Spark Streaming、Samza）、图计算框架（如 Pregel、Hama）、搜索系统（如 ElasticSearch、Solr）等。这些系统不追求“大而全”，而是专注于满足企业某一方面的业务需求，因此，取得了很好的性能。但是，随之而来的问题是如何实现这些专用系统与 Hadoop 系统各个组件之间数据的导入导出。一种朴素的想法是，为各个专用系统单独开发数据导入导出工具。这种解决方案在技术上没有实现难度，但是，带来了较高的实现代价，因为，每当有一款新的产品加入到企业的大数据生态系统中，就需要为这款产品开发和 Hadoop 各个组件的数据交换工具。因此，有必要设计一种通用的工具，起到数据交换枢纽的作用，其他工具加入大数据生态系统后，只需要开发和这款通用工具的数据交换方案，就可以通过这个交换枢纽轻松实现和其他 Hadoop 组件的数据交换。Kafka 就是一款可以实现这种功能的产品。

如图 15-14 所示，在公司的大数据生态系统中，可以把 Kafka 作为数据交换枢纽，不同类型的分布式系统（关系数据库、NoSQL 数据库、流处理系统、批处理系统等），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，较好地满足各种企业应用需求。

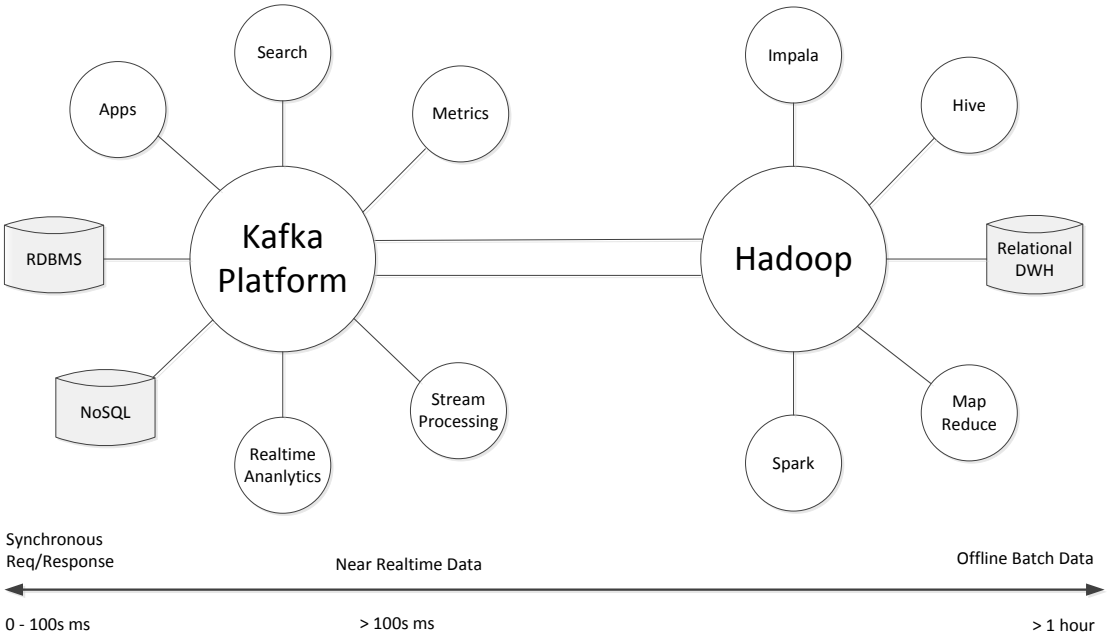


图 15-14 Kafka 作为数据交换枢纽

习题

1.试述在 Hadoop 推出之后其优化与发展主要体现在哪两个方面。

2.试述 HDFS1.0 中只包含一个名称节点会带来哪些问题。

3.请描述 HDFS HA 架构组成组件及其具体功能。

4.请分析 HDFS HA 架构中数据节点如何和名称节点保持通信。

5.请阐述为什么需要 HDFS Federation，即它能够解决什么问题。

6.请描述 HDFS Federation 中“块池”的概念，并分析为什么 HDFS Federation 中的一个名称节点失效，也不会影响到与它相关的数据节点继续为其他名称节点提供服务。

7.请阐述 MapReduce1.0 体系结构中存在的问题。

8.请描述 YARN 架构中个组件的功能。

9.请描述在 YARN 框架中执行一个 MapReduce 程序时，从提交到完成需要经历的具体步骤。

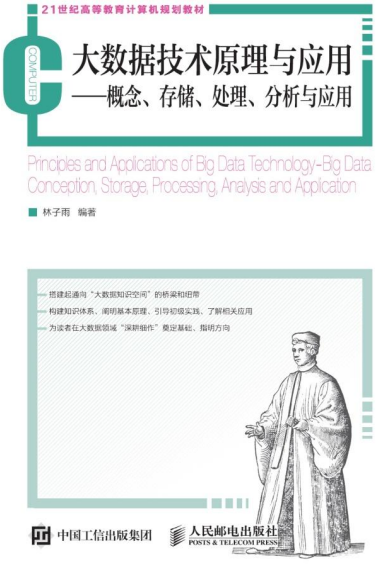
10.请对 YARN 和 MapReduce1.0 框架进行优劣势对比分析。

11.请分别描述 Pig、Tez、Spark 和 Kafka 的功能。

附录 1:任课教师介绍

林子雨(1978－),男,博士,厦门大学计算机科学系助理教授,主要研究领域为数据库,实时主动数据仓库,数据挖掘.
主讲课程：《大数据技术基础》
办公地点：厦门大学海韵园科研 2 号楼
E-mail: ziyulin@xmu.edu.cn
个人主页：http://www.cs.xmu.edu.cn/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

附录 2：课程教材介绍



《大数据技术原理与应用——概念、存储、处理、分析与应用》，由厦门大学计算机科学系教师林子雨博士编著，是中国高校第一本系统介绍大数据知识的专业教材。本书定位为大数据技术入门教材，为读者搭建起通向“大数据知识空间”的桥梁和纽带，以“构建知识体系、阐明基本原理、引导初级实践、了解相关应用”为原则，为读者在大数据领域“深耕细作”奠定基础、指明方向。

全书共有 13 章，系统地论述了大数据的基本概念、大数据处理架构 Hadoop、分布式文件系统 HDFS、分布式数据 库 HBase、NoSQL 数据库、云数据库、分布式并行编程模型 MapReduce、流计算、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在 Hadoop、HDFS、HBase 和 MapReduce 等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：http://dblab.xmu.edu.cn/post/bigdata

扫一扫访问教材官网

附录 3：中国高校大数据课程公共服务平台介绍

中国高校大数据课程公共服务平台，由中国高校首个“数字教师”的提出者和建设者——林子雨老师发起，由厦门大学数据库实验室全力打造，由厦门大学云计算与大数据研究中心、海峡云计算与大数据应用研究中心携手共建。这是国内第一个服务于高校大数据课程建设的公共服务平台，旨在促进国内高校大数据课程体系建设，提高大数据课程教学水平，降低大数据课程学习门槛，提升学生课程学习效果。平台服务对象涵盖高校、教师和学生。平台为高校开设大数据课程提供全流程辅助，为教师开展教学工作提供一站式服务，为学生学习大数据课程提供全方位辅导。平台重点打造“9 个 1 工程”，即 1 本教材（含官网）、1 个教师服务站、1 个学生服务站、1 个公益项目、1 堂巡讲公开课、1 个示范班级、1 门在线课程、1 个交流群（QQ 群、微信群）和 1 个保障团队。

平台主页：http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

扫一扫访问平台主页