第 3 章

大数据架构

本章学习目标

- 了解大数据架构的概念及类型
- 了解 Hadoop 架构的发展史及核心组件
- 了解 HDFS 概念及操作
- 了解 MapReduce 的概念及设计方式
- 掌握 Hadoop 的搭建
- 掌握 MapReduce 的应用

本章先向读者介绍大数据架构的概念及类型，再介绍 Hadoop 架构，然后介绍 HDFS 和 MapReduce 架构，重点讲解 Hadoop 的搭建及 MapReduce 的应用。

3.1 大数据架构概述

3.1.1 大数据架构介绍


视频讲解

大数据架构是用于摄取和处理大量数据（通常称为“大数据”）的总体系统，因此可以针对业务目的进行分析。该架构可视为基于组织业务需求的大数据解决方案的蓝图。大数据架构旨在处理以下类型的工作。

（1）批量处理大数据源。

（2）实时处理大数据。

（3）预测分析和机器学习：精心设计的大数据架构可以节省企业资金，并帮助其预测未来趋势，从而做出明智的业务决策。

企业中数据处理平台的基础架构如图 3-1 所示，底层的数据经过数据平台处理后，最

终为决策者所使用。

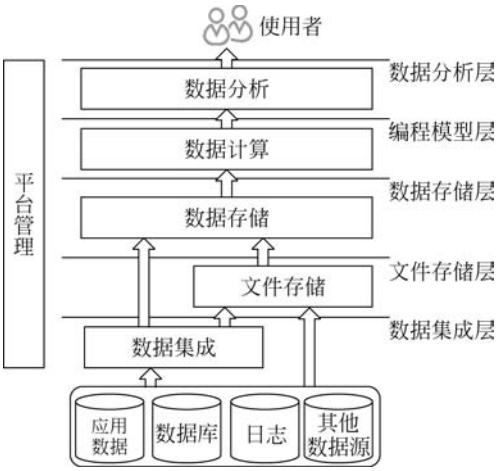

图 3-1　数据处理平台的基础架构

大数据架构因企业的基础设施和需求而异，通常包含以下组件。

（1）数据源。所有大数据架构都从源代码开始，可以包括来自数据库的数据、来自实时源（如物联网设备）的数据，以及从应用程序（如 Windows 日志）生成的静态文件。

（2）实时消息接收。如果有实时数据源，则需要在架构中构建一种机制摄取数据。

（3）数据存储。企业需要存储将通过大数据架构处理的数据。通常，数据将存储在数据湖中，这是一个可以轻松扩展的大型非结构化数据库。

（4）批处理和实时处理的组合。企业需要同时处理实时数据和静态数据，因此应在大数据架构中内置批处理和实时处理的组合。这是因为批处理可以有效地处理大量数据，而实时数据需要立即处理才能带来价值。批处理涉及长时间运行的作业，用于筛选、聚合和准备数据进行分析。

（5）分析数据存储。准备好要分析的数据后，需要将它们放在一个位置，以便对整个数据集进行分析。分析数据存储的重要性在于，企业的所有数据都集中在一个位置，因此其分析将是全面的，并且针对分析而非事务进行了优化。可能采取基于云计算的数据仓库或关系数据库的形式，具体取决于企业的需求。

（6）分析或报告工具。在摄取和处理各种数据源之后，需要一个分析数据的工具。通常，企业将使用商业智能（Business Intelligence，BI）工具完成这项工作，并且可能需要数据科学家来探索数据。

使用大数据架构可以帮助企业节省资金并做出关键决策，其主要作用至少包括以下几点。

（1）降低成本。在存储大量数据时，Hadoop 和基于云计算的分析等大数据技术可以显著地降低成本。

（2）做出更快、更好的决策。使用大数据架构的流组件，企业可以实时做出决策。

（3）预测未来需求并创建新产品。大数据可以帮助企业衡量客户需求并分析预测未来趋势。

3.1.2 大数据架构分类

目前围绕 Hadoop 体系的大数据架构主要有传统大数据架构、流式架构、Lambda 架构、Kappa 架构和 Unifield 架构等。

1. 传统大数据架构

这种架构之所以称为传统大数据架构，是因为其目标定位是为了解决传统商业智能所存在的问题。简单来说，基本的数据分析业务没有发生任何本质上的变化，但是因为数据量越来越大、性能越来越低等问题导致商业智能系统无法正常使用，因此需要进行升级改造，传统的大数据架构便是为了解决这些问题，如大数据量存储、提高应用系统等。可以看到，其依然保留了抽取、转换、加载的动作，将数据经过抽取、转换、加载数据采集操作存入数据存储。这种架构在很多场景中都有作用。

- 优点：简单、易懂，对于 BI 系统，基本思想没有发生变化，变化的仅仅是技术选型，用大数据架构替换 BI 的组件。
- 缺点：对于大数据，没有 BI 下如此完备的 Cube 架构，虽然目前有 kylin，但是 kylin 的局限性非常明显，远远没有 BI 下 Cube 的灵活度和稳定度，因此对业务支撑的灵活度不够，所以对于存在大量报表者复杂的钻取的场景，需要太多的手工定制化。同时该架构依旧以批处理为主，缺乏实时的支撑。
- 适用场景：数据分析需求依旧以 BI 场景为主，但是因为数据量、性能等问题无法满足日常使用。

2. 流式架构

在传统大数据架构的基础上，流式架构非常激进，直接去掉了批处理，数据全程以流的形式处理，所以在数据接入端没有 ETL，转而替换为数据通道。经过流处理加工后的数据，以消息的形式直接推送给消费者。该架构虽然有一个存储部分，但是更多地以窗口的形式进行存储，所以存储并非发生在数据湖，而是在外围系统。

- 优点：没有臃肿的 ETL 过程，数据的实时性非常高。
- 缺点：对于流式架构，不存在批处理，因此对于数据的重播和历史统计无法很好地支撑。对于离线分析，仅支撑窗口之内的分析。
- 适用场景：预警、监控等对数据有有效期要求的场景。

3. Lambda 架构

Lambda 架构算是大数据系统中举足轻重的架构，大多数架构基本都是 Lambda 架构或基于它的变种。Lambda 架构的数据通道分为两条分支：实时流和离线流。实时流依照流式架构，保障了其实时性，在具体应用中通常包含实时处理查询和实时数据处理。而离线流则以批处理方式为主，保障了最终一致性，在具体应用中通常包含批处理查询、批处理预计算和批处理存储。Lambda 架构组成如图 3-2 所示。

- 优点：既有实时又有离线，对于数据分析场景涵盖得非常到位。

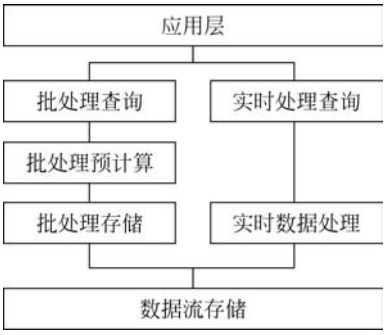

图 3-2　Lambda 架构组成

- 缺点：离线流和实时流虽然面临的场景不相同，但是其内部处理的逻辑却相同，因此有大量冗余的模块存在。
- 适用场景：同时存在实时和离线需求的场景。

4. Kappa 架构

Kappa 架构在 Lambda 架构的基础上进行了优化，将实时和流部分进行了合并，将数据通道用消息队列进行替代。因此，对于 Kappa 架构，依旧以流处理为主，但是数据在数据湖层面进行了存储，当需要进行离线分析或再次计算时，则将数据湖的数据再次经过消息队列重播一次则可。

- 优点：Kappa 架构解决了 Lambda 架构中的冗余问题，以数据可重播的思想进行了设计，整个架构非常简洁。
- 缺点：虽然 Kappa 架构看起来简洁，但是难度相对较高，尤其是对于数据重播部分。
- 适用场景：和 Lambda 架构类似，该架构是针对 Lambda 架构的优化。

5. Unifield 架构

以上各种架构主要围绕海量数据处理，Unifield 架构则更激进，将机器学习和数据处理融为一体，从核心上来说，Unifield 架构依旧以 Lambda 架构为基础，不过对其进行了改造，在流式层新增了机器学习层。可以看到，数据在经过数据通道进入数据湖后，新增了模型训练部分，并且将其在流式层进行使用。同时，流式层不单使用模型，也包含对模型的持续训练。

- 优点：Unifield 架构提供了一套数据分析和机器学习结合的架构方案，非常好地解决了机器学习如何与数据平台进行结合的问题。
- 缺点：Unifield 架构实施复杂度更高。对于机器学习架构，从软件包到硬件部署都和数据分析平台有着非常大的差别，因此在实施过程中的难度更高。
- 适用场景：有大量数据需要分析，同时对机器学习又有非常大的需求或有规划的场景。

3.2　Hadoop 架构

视频讲解

3.2.1　Hadoop 介绍

1. Hadoop 概述

Hadoop 是 Apache 软件基金会旗下的一个开源分布式计算平台。以 Hadoop 分布式文件系统（Hadoop Distributed File System，HDFS）和 MapReduce（Google MapReduce

的开源实现）为核心的 Hadoop 为用户提供了系统底层细节透明的分布式基础架构。HDFS 的高容错性、高伸缩性等优点允许用户将 Hadoop 部署在低廉的硬件上，形成分布式系统，为海量的数据提供存储方法；MapReduce 分布式编程模型允许用户在不了解分布式系统底层细节的情况下开发并行应用程序，为海量数据提供计算方法。所以，用户可以利用 Hadoop 轻松地组织计算机资源，从而搭建自己的分布式计算平台，并可以充分利用集群的计算和存储能力完成海量数据的处理。经过业界和学术界长达 10 年的锤炼，目前 Hadoop 已经趋于完善，在实际的数据处理和分析任务中担当着不可替代的角色。

Hadoop 本质上起源于 Google 的集群系统，Google 的数据中心使用廉价 Linux PC 组成集群，运行各种应用，即使是分布式开发新手也可以迅速学会使用 Google 的基础设施。如今广义的 Hadoop 已经包括 Hadoop 本身和基于 Hadoop 的开源项目，并已经形成了完备的 Hadoop 生态链系统。

狭义的 Hadoop 就是单独指代 Hadoop 这个软件；广义的 Hadoop 指代大数据的一个生态圈，还包括很多其他的软件。

2. Hadoop 生态圈

Hadoop 生态圈泛指大数据技术相关的开源组件或产品，如常见的 Hbase、Hive、Spark、Pig、ZooKeeper、Kafka、Flume、Sqoop、Hue、Storm、Mahout、Shark 等。生态圈中的这些组件或产品相互之间会有依赖，但又各自独立。Hadoop 生态圈组件分布如图 3-3 所示。

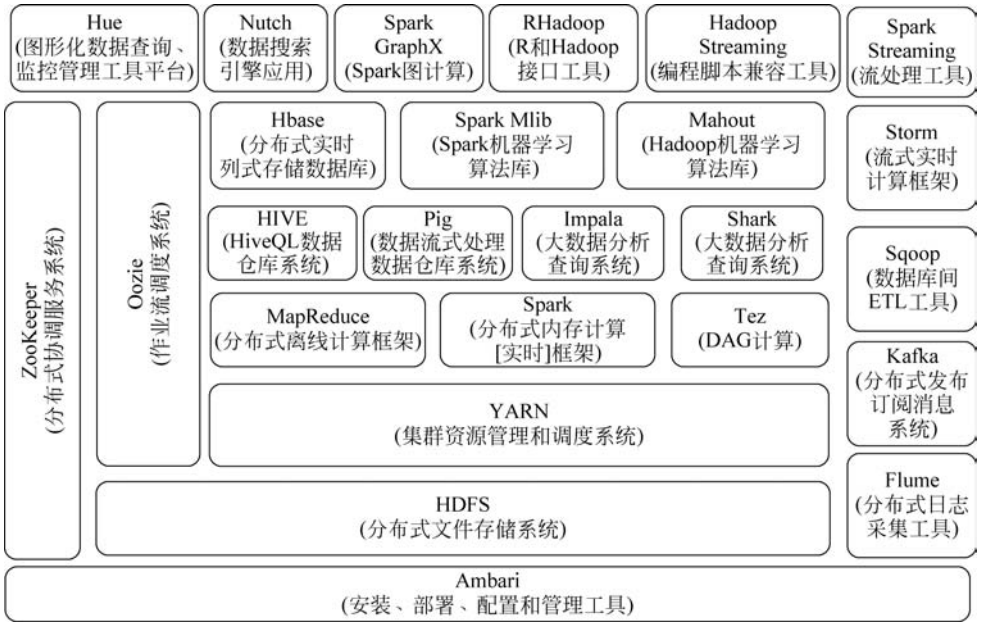

图 3-3　Hadoop 生态圈组件分布

值得注意的是，图 3-3 中并没有包括当前生态圈中的所有组件。因为 Hadoop 生态圈技术在不断发展，会不断有新的组件出现，而一些老的组件也可能被新组件替代。

3. Hadoop 特点

Hadoop 有以下几个特点。

图 3-4　Hadoop 的标志

1）Hadoop 是一个框架

很多初学者在学习 Hadoop 时对其本质并不十分了解。Hadoop 其实是由一系列的软件库组成的框架。这些软件库也可称作功能模块，它们各自负责了 Hadoop 的一部分功能，其中最主要的是 Common、HDFS 和 YARN。Common 提供远程过程调用（Remote Procedure Call，RPC）、序列化机制；HDFS 负责数据的存储；YARN 则负责统一资源调度和管理等。

Hadoop 的标志如图 3-4 所示，欢快的黄色大象如今已深入人心。

2）Hadoop 适合处理大规模数据

这是 Hadoop 一个非常重要的特点和优点。Hadoop 处理海量数据的能力十分可观，并能够实现分布式存储和分布式计算，有统一的资源管理和调度平台，扩展能力十分优秀。2008 年，Hadoop 打破 297s 的世界纪录，成为最快的 TB 级数据排序系统，仅用时 209s。

3）Hadoop 被部署在一个集群上

承载 Hadoop 的物理实体是一个集群。所谓集群，是一组通过网络互联的计算机，集群中的每台计算机称为一个节点。Hadoop 被部署在集群之上，对外提供服务。当节点数量足够多时，故障将成为一种常态而不是异常现象，Hadoop 在设计之初就将故障的发生作为常态进行考虑，数据的灾备及应用的容错对于用户都是透明的，用户得到的只是一个提供高可用服务的集群。图 3-5 所示为 Hadoop 的分布式集群。

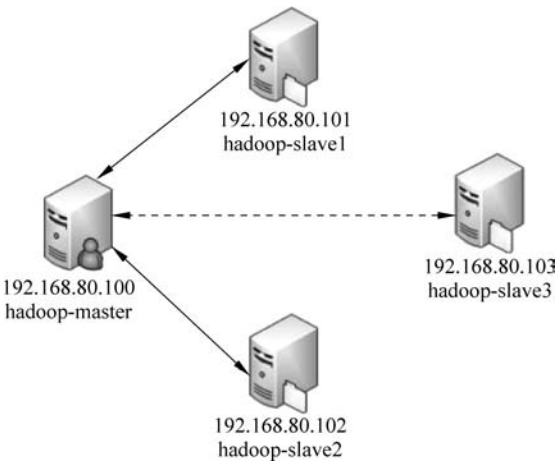

图 3-5　Hadoop 的分布式集群

3.2.2 Hadoop发展史

Hadoop原本来自Google一款名为MapReduce的编程模型包。Google的MapReduce框架可以把一个应用程序分解为许多并行计算指令，跨大量的计算节点运行巨大的数据集。使用该框架的一个典型例子就是在网络数据上运行的搜索算法。Hadoop最初只与网页索引有关，后来迅速发展为分析大数据的领先平台。

Hadoop的源头是Apache Nutch，该项目始于2002年，是Apache Lucene的子项目之一。Apache Nutch的设计目标是构建一个大型的全网搜索引擎，包括网页抓取、索引、查询等功能。但随着抓取网页数量的增加，遇到了严重的可扩展性问题——如何解决数十亿网页的存储和索引问题。之后，Google发表的两篇文章为该问题提供了可行的解决方案。一篇是2003年发表的关于Google分布式文件系统(GFS)的文章。该文章描述了Google搜索引擎网页相关数据的存储架构，该结构可以解决Apache Nutch遇到的网页抓取和索引过程中超大文件存储需求的问题。但由于Google未开源代码，Apache Nutch项目组便根据文章完成了一个开源实现：Apache Nutch的分布式文件系统(NDFS)。另一篇是2004年Google在“操作系统设计与实现”(Operating System Design and Implementation，OSDI)会议上公开发表了题为*MapReduce：Simplified Data Processing on Large Clusters*(《MapReduce：简化大规模集群上的数据处理》)的文章。该文章描述了Google内部最重要的分布式计算框架MapReduce的设计艺术，该框架可用于处理海量网页的索引问题。之后，受到启发的Doug Cutting等开始尝试实现MapReduce计算框架，并将它与NDFS(Nutch Distributed File System)结合，用于支持Apache Nutch引擎的主要算法。由于NDFS和MapReduce在Apache Nutch引擎中有着良好的应用，所以它们于2006年2月被分离出来，成为一套完整而独立的软件，并命名为Hadoop。到了2008年年初，Hadoop已成为Apache的顶级项目，包含众多子项目，被应用到很多互联网公司。Hadoop 1.0.1版本已经发展成为包含HDFS、MapReduce子项目，与Pig、ZooKeeper、Hive、HBase等项目相关的大型应用工程，迎来了它的快速发展期。

Hadoop的历史版本一共有3个，分别如下。

(1) 0.x系列版本：Hadoop中最早的一个开源版本。

(2) 1.x版本系列：Hadoop的第2代开源版本，主要修复0.x版本的一些Bug等。

(3) 2.x版本系列：架构产生重大变化，引入了YARN平台等许多新特性。

3.2.3 Hadoop核心组件

Hadoop的三大核心组件分别是HDFS、YARN和MapReduce。HDFS(Hadoop Distribute File System)是Hadoop的数据存储工具；YARN(Yet Another Resource Negotiator)是Hadoop的资源管理器；MapReduce是分布式计算框架。

1. HDFS

HDFS是一个文件系统，用于存储文件，通过目录树定位文件；另外，它是分布式的，

由很多服务器联合起来实现其功能，集群中的服务器有各自的角色。

HDFS 适合一次写入，多次读出的场景，且不支持文件的修改；适合进行数据分析，并不适用于网盘应用。

2. YARN

YARN 是一种新的 Hadoop 资源管理器，它是一个通用资源管理系统，可为上层应用提供统一的资源管理和调度，它的引入为集群在利用率、资源统一管理和数据共享等方面带来了巨大好处。通过 YARN，不同计算框架可以共享同一个 HDFS 集群上的数据，享受整体的资源调度。

YARN 的基本思想是将 JobTracker 的两个主要功能（资源管理和作业调度/监控）进行分离，主要方法是创建一个全局的 ResourceManager（RM）和若干针对应用程序的 ApplicationMaster（AM）。这里的应用程序是指传统的 MapReduce 作业或作业的有向无环图（DAG）。

YARN 分层架构的本质是 ReduceManager。这个实体控制整个集群并管理应用程序向基础计算资源的分配。ReduceManager 将各个资源部分（计算、内存、带宽等）精心安排给基础 NodeManager（YARN 的每节点代理）。ReduceManager 还与 ApplicationMaster 一起分配资源。

3. MapReduce

MapReduce 是 Google 公司于 2004 年提出的能并发处理海量数据的并行编程模型，其特点是简单易学，应用广泛，能够降低并行编程难度，让程序员从繁杂的并行编程工作中解脱出来，轻松地编写简单、高效的并行程序。

MapReduce 是一种编程模型，用于大规模数据集（大于 1TB）的并行运算。概念“Map（映射）”和“Reduce（归约）”是它们的主要思想，都是从函数式编程语言中借来的，还有从矢量编程语言中借来的特性。它极大地方便了编程人员在不会分布式并行编程的情况下将自己的程序运行在分布式系统上。

视频讲解

3.3 HDFS 概述

3.3.1 HDFS 的概念

1. HDFS 介绍

HDFS 是基于流数据模式访问和处理超大文件的需求而开发的，是一个分布式文件系统。它是 Google 的 GFS 提出之后出现的另一种文件系统。HDFS 具有一定高度的容错性，且提供了高吞吐量的数据访问，非常适合大规模数据集。

HDFS 的设计特点如下。

（1）大数据文件。非常适合 TB 级别的大文件或一堆大数据文件的存储，如果文件只有 GB 级甚至更小就没有什么意义了。

(2) 文件分块存储。HDFS会将一个完整的大文件平均分块存储到不同计算器上，其意义在于读取文件时可以同时从多个主机读取不同区块的文件，多主机读取比单主机读取效率要高得多。

(3) 流式数据访问。一次写入，多次读写，这种模式与传统文件不同，它不支持动态改变文件内容，而是要求文件一次写入就不做变化，要变化也只能在文件末尾添加内容。

(4) 廉价硬件。HDFS可以应用在普通个人计算机(Personal Computer，PC)上，这种机制能够让一些企业用几十台廉价的计算机就可以撑起一个大数据集群。

(5) 硬件故障。HDFS认为所有计算机都可能会出问题，为了防止某个主机失效读取不到该主机的块文件，它将同一个文件块副本分配到其他某些主机上，如果其中一台主机失效，可以迅速找另一块副本读取文件。

2. HDFS的优缺点

基于上述设计特点，HDFS有一系列的优点和缺陷。

HDFS的优点如下。

(1) 处理超大文件。这里的超大文件通常是指数百兆字节大小的文件。但是，目前在实际应用中，HDFS已经能用来存储管理PB级的数据了。雅虎的Hadoop集群也已经扩展到了4000个节点。

(2) 流式数据访问。HDFS的设计建立在“一次写入，多次读写”任务的基础上。这意味着一个数据集一旦由数据源生成，就会被复制分发到不同的存储节点中，然后响应各种各样的数据分析任务请求。在多数情况下，分析任务都会涉及数据集中的大部分数据，也就是说，对于HDFS，请求读取整个数据集要比读取一条记录更加高效。

(3) 运行于廉价的商用机集群上。Hadoop设计对应急需求比较低，只需运行在低廉的商用硬件集群上，而无须运行在昂贵的高可用性机器上。廉价的商用机也就意味着大型集群中出现节点故障情况的概率非常高。HDFS遇到了上述故障时，被设计成能够继续运行且不让用户察觉到明显的中断。

正是出于以上种种考虑，人们会发现，现在HDFS在处理一些特定问题时不但没有优势，反而存在很多局限性。具体局限性及应对策略如下。

(1) 不适合低延迟数据访问。HDFS不适合处理一些用户要求时间比较短的低延迟应用请求。HDFS是为了处理大型数据集分析任务的，主要是为达到高的数据吞吐量而设计的，这就可能要求以高延迟作为代价。

改进策略：对于那些有低延时要求的应用程序，HBase是一个更好的选择，通过上层数据管理项目尽可能地弥补这个不足。HBase在性能上有了很大的提升，它的口号是goes real time。使用缓存或多个Master设计可以降低Client的数据请求压力，以减少延时。

(2) 无法高效存储大量的小文件。小文件是指文件大小小于HDFS上块大小的文件。这样的文件会给Hadoop的扩展性和性能带来严重问题。当Hadoop处理很多小文件时，由于FileInputFormat不会对小文件进行划分，所以每个小文件都会被当作一个Split并分配一个Map任务，导致效率低下。

例如，一个 1GB 的文件，会被划分成 16 个 64MB 的 Split，并分配 16 个 Map 任务处理，而 10 000 个 100KB 的文件会被 10 000 个 Map 任务处理。

改进策略：要想让 HDFS 处理好小文件，有不少方法。利用 SequenceFile、MapFile、Har 等方式归档小文件，这个方法的原理就是把小文件归档起来管理，HBase 就是基于此的。

（3）不支持多用户写入及任意修改文件。在 HDFS 的一个文件中只有一个写入者，且写操作只能在文件末尾完成，即只能执行追加操作。目前 HDFS 还不支持多个用户对同一文件的写操作，以及在文件任意位置进行修改。

3. HDFS 的构成

HDFS 的关键元素包含 Block、NameNode 和 DataNode。

Block：将一个文件进行分块，通常一个块的大小为 64MB。

NameNode：保存整个文件系统的目录信息、文件信息和分块信息，这是由唯一一台主机专门保存，当然如果这台主机出错，NameNode 就失效了。从 Hadoop 2.x 开始支持 Activity-Standy 模式——如果主 NameNode 失效，就启动备用主机运行 NameNode。

DataNode：分布在廉价的计算机上，用于存储块文件。

一个完整的 HDFS 运行在一些节点上，这些节点运行着不同类型的守护进程，如 NameNode、DataNode、SecondaryNameNode 等。不同类型的节点相互配合，相互协作，在集群中扮演了不同的角色，一起构成了 HDFS。

如图 3-6 所示，在一个典型的 HDFS 集群中，有一个 NameNode、一个 SecondaryNameNode 和至少一个 DataNode，而 HDFS 客户端数量并没有限制。所有的数据均存放在运行 DataNode 进程的节点的块（Block）中。

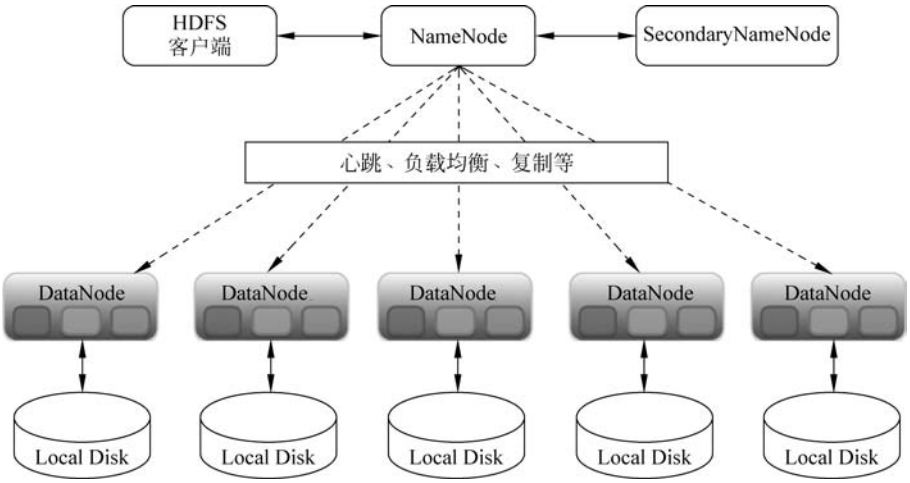

图 3-6　HDFS 架构

（1）HDFS 客户端（HDFS Client）。HDFS 客户端是指用户和 HDFS 交互的手段，HDFS 提供了非常多的客户端，包括命令行接口、Java API、Thrift 接口、C 语言库、用户

空间文件系统等。

(2) NameNode(元数据节点)。NameNode 是管理者,一个 Hadoop 集群只有一个 NameNode,通常是一个在 HDFS 实例中的单独机器上运行的软件。NameNode 主要负责 HDFS 文件系统的管理工作,具体包括命名空间(Namespace)管理和文件块管理。NameNode 决定是否将文件映射到 DataNode 的复制块上。对于最常见的 3 个复制块,第 1 个复制块存储在同一个机架的不同节点上,最后一个复制块存储在不同机架的某个节点上。

NameNode 是 HDFS 的大脑,它维护着整个文件系统的目录树以及目录树中所有的文件和目录,这些信息以两种文件形式存储在本地文件中:一种是命名空间镜像,也称为文件系统镜像(File System Image,FSImage),即 HDFS 元数据的完整快照,每次 NameNode 启动时,默认会加载最新的命名空间镜像;另一种是命名空间镜像的编辑日志(Edit Log)。

(3) SecondaryNameNode(第 2 名字节点)。SecondaryNameNode 是用于定期合并命名空间镜像和命名空间镜像的编辑日志的辅助守护进程。每个 HDFS 集群都有一个 SecondaryNameNode,在生产环境下,一般 SecondaryNameNode 也会单独运行在一台服务器上。

FSImage 文件其实是文件系统元数据的一个永久性检查点,但并非每个写操作都会更新这个文件,因为 FSImage 是一个大型文件,如果频繁地执行写操作,会使系统运行极为缓慢。解决方案是 NameNode 只改动内容预写日志(Write-Ahead Logging,WAL),即写入命名空间镜像的编辑日志。随着时间的推移,编辑日志会变得越来越大,那么一旦发生故障,将花费非常多的时间回滚操作。所以,就像传统的关系型数据库一样,需要定期地合并 FSImage 和编辑日志。如果由 NameNode 进行合并操作,那么 NameNode 在为集群提供服务时可能无法提供足够的资源。为了彻底解决这一问题,SecondaryNameNode 应运而生。NameNode 与 SecondaryNameNode 的交互如图 3-7 所示。

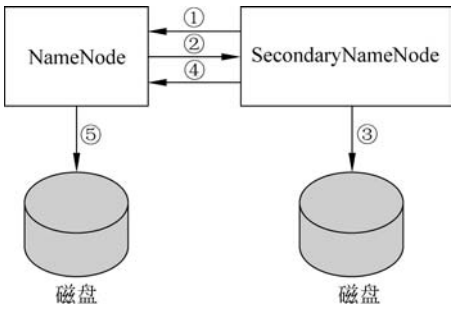

图 3-7 NameNode 与 SecondaryNameNode 的交互

① SecondaryNameNode 引导 NameNode 滚动更新编辑日志文件,并开始将新的内容写入 EditLog. new。

② SecondaryNameNode 将 NameNode 的 FSImage 和编辑日志文件复制到本地的检查点目录。

③ SecondaryNameNode 载入 FSImage 文件，回放编辑日志，将其合并到 FSImage，将新的 FSImage 文件压缩后写入磁盘。

④ SecondaryNameNode 将新的 FSImage 文件送回 NameNode，NameNode 在接收新的 FSImage 后，直接加载和应用该文件。

⑤ NameNode 将 EditLog. new 更名为 EditLog。

默认情况下，该过程每小时发生一次，或者当 NameNode 的编辑日志文件达到默认的 64MB 时也会被触发。

从名称来看，初学者会以为当 NameNode 出现故障时，SecondaryNameNode 会自动成为新的 NameNode，也就是 NameNode 的“热备”。通过上面的介绍，可以清楚地认识到这是错误的。

(4) DataNode(数据节点)。DataNode 是 HDFS 的主从架构中从角色的扮演者，它在 NameNode 的指导下完成输入/输出(I/O)任务。如前文所述，存放在 HDFS 的文件都是由 HDFS 的块组成的，所有的块都存放于 DataNode。实际上，对于 DataNode 所在的节点，块就是一个普通的文件，可以在 DataNode 存放块的目录下(默认为 $(dfs. data. dir)/current)查看，块的文件名为 blk. blkID。

DataNode 会不断地向 NameNode 报告。初始化时，每个 DataNode 将当前存储的块告知 NameNode，在集群正常工作时，DataNode 仍会不断地更新 NameNode，为其提供本地修改的相关信息，同时接收来自 NameNode 的指令，创建、移动或删除本地磁盘上的数据块。

(5) Block(块)。每个磁盘都有默认的数据块大小，这是磁盘进行数据读/写的最小单位，而文件系统也有文件块的概念，如 ext3、ext2 等。文件系统的块大小只能是磁盘块大小的整数倍，磁盘块的大小一般是 512B，文件系统的块大小一般为几千字节，如 ext3 的文件块大小为 4096B，Windows 的文件块大小为 4096B。用户在使用文件系统对文件进行读取或写入时，完全不知道块的细节，这些对于用户是透明的。

HDFS 同样也有块的概念，但是 HDFS 的块比一般文件系统的块大得多，默认为 64MB，并可以随着实际需要而变化，配置项为 hdfs-site. xml 文件中的 dfs. block. size 项。与单一文件系统相似，HDFS 上的文件也被划分为多个分块，它是 HDFS 存储处理的最小单元。

例如，data. txt 文件大小为 150MB，如果此时 HDFS 的块大小没有经过配置，默认为 64MB，那么该文件在 HDFS 中存储的情况如图 3-8 所示。

图 3-8 中，圆形为保存该文件的第 1 个块，大小为 64MB；正方形为保存该文件的第 2 个块，大小为 64MB；五边形为保存文件的第 3 个块，大小为 22MB。与其他文件系统不同的是，HDFS 小于一个块大小的文件不会占据整个块的空间，所以第 3 个块的大小为 22MB，而不是 64MB。

HDFS 中的块如此大的原因是为了最小化寻址开销。如果块设置得足够大，从磁盘传输数据的时间可以明显大于定位这个块开始位置所需的时间。这样，传输一个由多个块组成的文件的时间取决于磁盘传输的效率。得益于磁盘传输速率的提升，块的大小可以被设置为 128MB 甚至更大。

在 hdfs-site. xml 文件中，还有一个 dfs. relication 配置项，该项配置为每个 HDFS 的块在 Hadoop 集群中保存的份数，值越高，冗余性越好，占用存储也越多。dfs. relication 默认为 3，即有两份冗余，如果在 hdfs-site. xml 文件中将 dfs. relication 设置为 2，那么该文件在 HDFS 中存储的情况如图 3-9 所示。

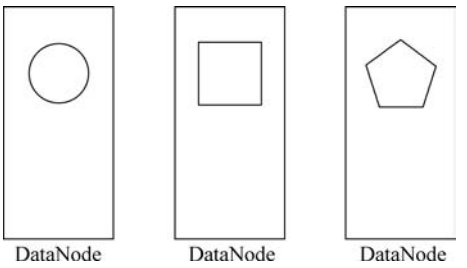

图 3-8 HDFS 中的块(默认大小)

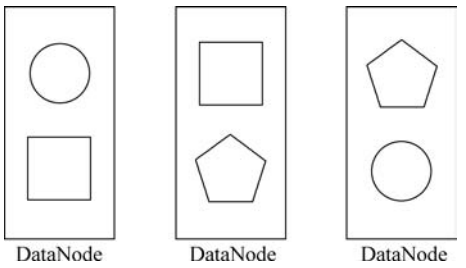

图 3-9 HDFS 中的块(dfs. relication 为 2)

使用块的好处如下。

首先，可以保存比存储节点单一磁盘大的文件。块的设计实际上就是对文件进行分片，分片可以保存在集群的任意节点，从而使文件存储跨越磁盘甚至机器的限制，如 data. txt 文件被切分为 3 个块，并存放于 3 个 DataNode 之中。

其次，简化存储子系统。将存储子系统控制单元设置为块，可简化存储管理，并且也实现了元数据和数据的分开管理和存储。

最后，容错性高。这是块非常重要的一点，在 HDFS 中任意一个块损坏，都不会影响数据的完整性，用户在读取文件时，并不会察觉到异常。之后集群会将损坏的块的副本从其他候选节点复制到集群中能正常工作的节点，从而使副本数回到配置的水平。

3.3.2 HDFS 操作

1. HDFS 命令操作简单示例

HDFS 命令可以执行所有其他文件系统都有的操作，如读取文件、创建目录、移动文件、删除数据、列出索引目录等。执行 hadoop fs-help 命令，即可看到所有命令详细的帮助文件。

首先，从本地文件系统复制一个文件到 HDFS。

```
% hadoop fs - copyFromLocal  /test/hadoop/test.txt   hdfs://localhost/user/tom/test.txt
```

上述命令调用 Hadoop 文件系统的 shell 命令 fs，该命令提供了一系列子命令，在本例中执行的是-copy FromLocal，本地文件 test. txt 被复制到运行在 localhost 上的 HDFS 实例中，路径为 user/tom/test. txt。事实上，可以简化命令格式以省略主机的统一资源标识符(Uniform Resource Identifier，URI)默认设置，即省略 hdfs://localhost，因为该项已在 core-sile. xml 中指定。

```
% hadoop fs - copyFromLocal  /test/hadoop/test.txt  user/tom/ test.txt
```

也可以使用相对路径，并将文件复制到 HDFS 的 home 目录中，在本例中为 /user/tom。

```
% hadoop fs -copyFromLocal  /test/hadoop/test.txt   test.txt
```

把文件复制回本地文件系统，并检查是否一致。

```
% hadoop fs - copyToLocal test.txt test.copy.txt
% md5 input/docs/test.txt test.copy.txt
MD5{input/docs/test.txt} = a16f231da6b05e2ba7a339320e7dacd9
MD5{ test.copy.txt } = a16f231da6b05e2ba7a339320e7dacd9
```

由于 MD5 键值相同，表明这个文件在 HDFS 之旅中得以幸存并保存完整。

2. HDFS 中的文件访问权限

针对文件和目录，HDFS 具有与可移植操作系统接口（Portable Operating System Interface of UNIX，POSIX）非常相似的权限模式。

HDFS 提供 3 类权限模式：只读权限（R）、写入权限（W）和可执行权限（X）。读取文件或列出目录内容时需要只读权限。写入一个文件或在一个目录上创建及删除文件或目录，需要写入权限。对于文件，可执行权限可以忽略，因为不能在 HDFS 中执行文件（与 POSIX 不同），但在访问一个目录的子项时需要该权限。对于目录，当列出目录内容时需要具有 R 权限，当新建或删除子文件或子目录时需要具有 W 权限，当访问目录的子节点时需要具有 X 权限。

每个文件和目录都有所属用户（Owner）、所属组别（Group）和模式（Mode）。这个模式是由所属用户的权限、组内成员权限及其他用户的权限组成。

默认情况下，可以通过正在运行进程的用户名和组名唯一确定客户端的标识。但由于客户端是远程的，任何用户都可以简单地在远程系统上以他的名义创建一个账户进行访问。因此，作为共享文件系统资源和防止数据意外损失的一种机制，权限只能供合作团体中的用户使用，而不能在一个不友好的环境中保护资源。注意，最新版本的 Hadoop 已经支持 Kerberos 用户认证，该认证去除了这些限制。但是，除了上述限制以外，为防止用户或自动工具及程序意外修改或删除文件系统的重要部分，启用权限控制还是很重要的（这也是默认的配置，参见 dfs.permissions 属性）。

如果启用权限检查，就会检查所属用户权限，以确认客户端的用户名与所属用户是否匹配。另外，也将检查所属组别权限，以确认该客户端是否是该用户组的成员，若不符，则检查其他权限。

注意，这里有一个超级用户（Super-User）的概念，超级用户是 NameNode 进程的标识。宽泛地讲，如果启动了 NameNode，你就是超级用户。另外，管理员可以用配置参数指定一组特定的用户，如果做了设定，这个组的成员也会是超级用户。对于超级用户，系统不会执行任何权限检查。


视频讲解

3.4 MapReduce 概述

3.4.1 MapReduce 的概念

在云计算和大数据技术领域被广泛提到并成功应用的一项技术是 MapReduce，MapReduce 是 Google 系统和 Hadoop 系统中的一项核心技术。它是一个软件框架，可以将单个计算作业分配给多台计算机执行，从而缩短运行时间。

1. MapReduce 简介

MapReduce 是一种分布式计算模型，在处理海量数据上具有很明显的优势，因此常应用于大规模数据集的并行计算。MapReduce 是开源的，任何人都可以借助这个框架进行并行编程。这个框架使之前复杂的分布式编程变得更容易实现。

MapReduce 分布式编程模型是 Google 引以为豪的三大云计算相关的核心技术(GFS、Big Table 和 MapReduce)之一，被设计用于并行运算处理大于 1TB 的海量数据集。MapReduce 的最初灵感来源于函数式编程语言中经常用到的映射(Map)和规约(Reduce)函数，它将复杂的并行算法处理过程抽象为一组概念简单的接口，用来实现大规模海量信息处理的并行化和分布化，从而使没有多少并行编程经验的开发人员也能轻松地进行并行编程。

MapReduce 分布式编程模型可以用于中小规模的能灵活调整的普通 PC 构成的集群之上，典型的 MapReduce 系统能运行于由数以千计普通廉价 PC 所组成的集群中，这已经在 Google 中得到实现与应用。MapReduce 分布式编程模型的主要贡献在于：通过实现一组概念简单却又强大的接口，以实现大规模计算的并行化和分布化，并通过实现这些接口，MapReduce 能够组建由普通廉价 PC 作为成员的高性能集群。在采用 MapReduce 分布式模式的系统上，一个单独的节点上可以同时运行一个 Map 任务和一个 Reduce 任务，所以 MapReduce 的处理效率非常高。

2. MapReduce 的发展历史

MapReduce 的出现要追溯到 1956 年，图灵奖获得者、著名的人工智能专家 McCarthy 首次提出了 LISP 语言的构想，而在 LISP 语言中就包含了现在所使用的 MapReduce 功能。LISP 语言是一种用于人工智能领域的语言，在 1956 年设计时主要是希望有效地进行“符号运算”，它是一种表处理语言，逻辑简单但结构不同于其他高级语言。1960 年，McCarthy 更是极有预见性地提出“今后计算机将会作为公共设施提供给公众”的观点，这一观点已与现在人们对云计算的定义极为相近，所以 McCarthy 被誉为“云计算之父”。MapReduce 在 McCarthy 提出时并没有考虑到其在分布式系统和大数据上会有如此大的应用前景，只是作为一种函数操作来定义的。

2004 年，Google 公司的 Dean 发表文章，将 MapReduce 这一编程模型在分布式系统中的应用进行了介绍，从此 MapReduce 分布式编程模型进入了人们的视野。可以认为 MapReduce 是由 Google 公司首先提出的，Hadoop 跟进了这一思想。Hadoop 是一个开

源版本的 Google 系统，正是由于 Hadoop 的跟进才使普通用户得以开发自己的基于 MapReduce 框架的云计算应用系统。

3. MapReduce 的优缺点

MapReduce 主要有两个优点：①通过 MapReduce 分布式处理框架，不仅能处理大规模数据，而且能将很多烦琐的细节隐藏起来，如自动并行化、负载均衡和灾备管理等，这将极大地简化程序员的开发工作；②MapReduce 的伸缩性非常好，也就是说，每增加一台服务器，MapReduce 能将差不多的计算能力接入集群中，而过去的大多数分布式处理框架在伸缩性方面都与 MapReduce 相差甚远。

但是，MapReduce 毕竟是一个离线计算框架，其不足之处如下。

（1）启动时间长。一个 Map 和 Reduce 作业前有启动任务步骤，后有清理任务步骤，这就使最简单的作业也会消耗几秒钟的时间。

（2）调度开销大。一个作业包含很多任务时，MapReduce 将任务调度到各个节点上会消耗比较长的时间，当资源不足时作业还得排队。

（3）由于作业要容错，计算的中间结果要写回文件系统，这导致了不必要的输入/输出操作，严重降低短作业处理速度。

（4）数据必须先存储才能运算。MapReduce 在搜索的应用中，先将爬虫爬取的网页数据放在一个分布式存储上，然后间断性地对这些数据进行批量处理（MapReduce），即先存储数据，后对数据进行运算。

3.4.2 MapReduce 设计方式

MapReduce 是一个简单、方便的分布式编程模型，主要面向存储在 HDFS 中的数据。采用“分而治之”的思想，MapReduce 将一个大规模数据分解为多个小规模数据，并将其分发给集群中的多个节点共同完成，这样可以有效降低每部分的运算复杂度，达到提高运算效率的目的。

1. MapReduce 的集群结构

一个 MapReduce 任务需要由以下 4 部分协作完成。

（1）客户端。客户端与集群进行交互的接口，可以进行任务提交、结果获取等工作。

（2）Job Tracker。Job Tracker 是集群的总负责节点，主要起到集群的调度作用，一个集群中只能有一个 Job Tracker。

（3）Task Tracker。Task Tracker 是作业的真正执行者，可以执行两类任务：Map 任务和 Reduce 任务。执行 Map 任务的 Task Tracker 称为 Mapper，执行 Reduce 任务的 Task Tracker 称为 Reducer，一个集群中可以有多个 Task Tracker。

（4）分布式文件系统。分布式文件系统用来存储输入/输出的数据，通常使用 HDFS。

2. MapReduce 的执行过程

MapReduce 的编程框架是由一个单独运行在主节点上的 Job Tracker 和运行在每个集群的从节点上的 Task Tracker 共同组成的。用户用 Map()和 Reduce()两个函数表达计算。Map()函数的输入是一个< Key, Value >键值对,输出一个< Key, Value >键值对的集合的中间结果。MapReduce 集合所有相同 Key 的 Value,然后提供给 Reduce()函数。Reduce()函数收到 Key 和对应的 Value 的集合,通过计算得到较小的 Value 值的集合。

MapReduce 任务分为两个阶段:Map 阶段和 Reduce 阶段。Job Tracker 将一个大规模的任务根据数据量分解,Map 阶段执行分解后的小任务并得到中间结果,Reduce 阶段负责把这些中间结果汇总。具体执行过程如下。

(1) 数据预处理。在任务开始前,首先调用类库,将输入文件分为多个分片。

(2) 任务分配。Job Tracker 为集群中空闲的节点分配 Map 任务或 Reduce 任务。设集群中有 M 个 Map 任务和 R 个 Reduce 任务(Reduce 任务数通常小于 Map 任务数)。

(3) Map 任务。Mapper 读取自己所属的文件分片,将每条输入数据转换为< Key, Value >键值对,使用 Map()函数对每个键值对进行处理,得到一个新的< Key, Value >键值对,并作为中间结果缓存在当前节点上。

(4) 缓存文件定位。Map 任务得到的中间结果被周期性地写入 Mapper 所在的本地硬盘中,并把文件的存储位置信息经由 Job Tracker 传递给 Reducer。

(5) Reducer 拉取文件。Reducer 通过位置信息到相应的 Mapper 处拉取这些文件,将同一 Key 对应的所有取值合并,得到< Key, List(Value)>键值组。

(6) Reduce 任务。Reducer 将所读取到的< Key, List(Value)>键值组使用 Reduce()函数进行计算,得到最终结果并将其输出。

(7) 结束。当所有的 Map 任务和 Reduce 任务运行完毕后,系统会自动结束各个节点上的对应进程并将任务的执行情况反馈给用户。

每个 Map 操作都是针对不同的初始数据,不同的 Map 操作之间彼此独立,互不影响,因而 Map 可以并行操作。Reduce 操作是对 Map 操作之后产生的一部分结果进行规约操作。每个 Reduce 操作和 Map 操作一样,也是互相独立的,所以 Reduce 也能够并行执行。由此得知,MapReduce 编程框架的共同特征是 MapReduce 的数据都是被分割成设定大小的数据块,这些数据块都能够被并行处理。

MapReduce 运行流程如图 3-10 所示。

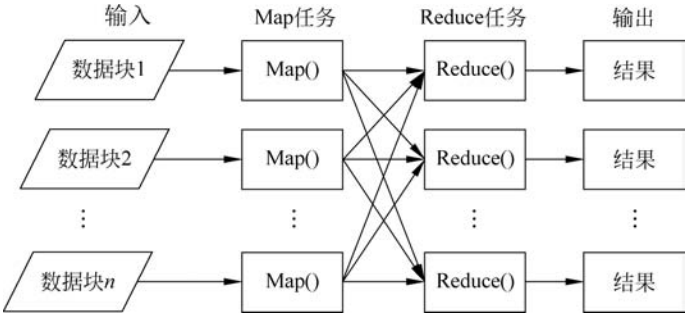

图 3-10 MapReduce 运行流程

MapReduce 是一个简便的分布式编程框架，此框架下并行程序中需要 Map()、Reduce() 和 main() 3 个主要函数。编程人员只要实现其中的 Map()函数和 Reduce()函数，其他问题（如分布式存储、工作调度、负载平衡等）都由 MapReduce 分布式框架负责完成。

3.4.3 MapReduce 架构

在 Hadoop 体系结构中，MapReduce 是一个简单、易用的软件框架。基于 MapReduce 可以将任务分发到由上千台商用机器组成的集群上，并以一种可靠容错的方式并行处理大量的数据集，实现 Hadoop 的并行任务处理功能。

1. MapReduce 架构

MapReduce 主要采用 Master/Slave（M/S）架构，其主要包括 Client、Job Tracker、Task Tracker 和 Task Scheduler 4 个组件。MapReduce 架构如图 3-11 所示，下面分别对这 4 个组件进行介绍。

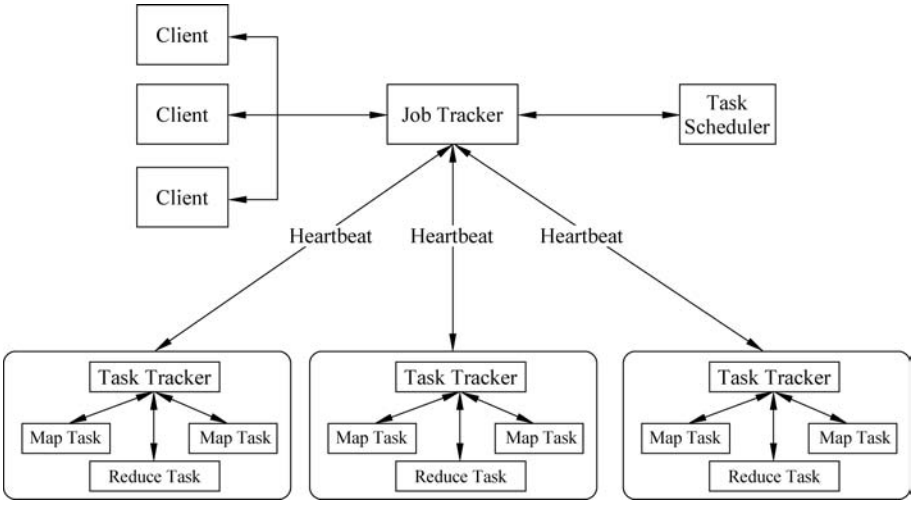

图 3-11 MapReduce 架构

1）Client

用户通过 Client 将编写的 MapReduce 程序提交到 Job Tracker 端，作业运行状态也是通过 Client 提供的部分接口来查询的。在 Hadoop 内部，MapReduce 程序是用作业（Job）表示的，一个 MapReduce 程序可对应若干个作业，而每个作业会被分解成若干个 Map/Reduce 任务（Task）。

2）Job Tracker

Job Tracker 主要实现资源监控和作业调度功能。Job Tracker 用来监控所有 Task Tracker 和作业的健康状况，在发现失败的情况下，Job Tracker 会跟踪其任务执行进度、资源使用量等信息，并将此信息生成报表发送给任务调度器，任务调度器在接收到命令之后及时选择合适的任务并将这些资源进行分配。在 Hadoop 中，任务调度器以模块形式存在，具有可插拔的特征，用户可以根据自己的需要设计相应的任务调度器。

3) Task Tracker

Task Tracker 使用 Heartbeat 将本节点上资源的使用情况和任务的进行情况汇报给 Job Tracker,同时接收 Job Tracker 发送过来的命令并响应启动新任务、杀死任务等操作。Slot 表示 CPU、内存上的计算资源,Slot 可以帮助 Task Tracker 将每个节点上的资源进行等量划分,每个任务只有在获得 Slot 才可以运行。Slot 包括 Map Slot 和 Reduce Slot 两种,Map Slot 供 Map Task 使用,Reduce Slot 供 Reduce Task 使用。

4) Task Scheduler

Task 可以分成 Map Task 和 Reduce Task 两种,且都由 Task Tracker 启动。HDFS 是以块为固定大小存储数据的,它是存储数据的基本单位。Split 主要包括数据起始位置、数据长度和数据所在节点等基本的元数据信息,它是 Map Reduce 的处理单元,每个 Split 会由一个 Map Task 处理,Split 的数量决定了 Map Task 的数目。Map Task 将接收到对应的 Split 通过迭代解析得到多个键值对,并使用用户自定义的 Map()函数处理进程。经 Map()函数处理过的数据被分成多个 Partition,每个 Partition 被对应的 Reduce Task 处理,并将数据保存在本地磁盘中。Reduce Task 执行过程包括以下 3 个阶段。

(1) 从数据节点中读取 Map Task 的中间结果,此阶段称为 Shuffle 阶段。

(2) 根据键值对排序,此阶段称为 Sort 阶段。

(3) 依次读取 Key 和 Value List 的值,并调用用户自定义的 Reduce()函数处理结果,并将此结果保存到 HDFS 中,此阶段称为 Reduce 阶段。

2. MapReduce 作业的生命周期

一个 MapReduce 作业的生命周期大致分为以下 5 个阶段。

1) 作业提交与初始化

用户在提交作业之后,Job Client 将程序 jar 程序包、作业配置文件、分片元信息文件等作业相关信息上传至分布式文件系统上,分片元信息文件的作用是记录每个输入分片的逻辑位置信息。当 Job Tracker 接收到 Job Client 的请求后,会立即进行初始化,之后在运行过程中监控作业运行情况,这就需要建立 Job in Progress 对象,而且可以同时监控多个任务的运行状况。

2) 任务调度与监控

Job Tracker 是用来对任务进行调度和监控的。Task Tracker 通过 Heartbeat 周期性地向 Job Tracker 发送本节点资源的使用情况,在有空闲资源的情况下,任务调度命令 Job Tracker 按照一定的计划选择合适的空闲资源。任务调度器是具有双层架构、比较独立的结构,可以完成对任务的选择,选择任务需要充分考虑数据的本地性。此外,Job Tracker 的作用保证任务可以成功运行,并可以跟踪作业的整个运行过程。如果 Task Tracker 或任务运行失败,则重新进行任务运行时间的计算;如果运行进度落后,也会重新进行计算;如果其他运行结束,就重新启动一个相同的任务;最终选取计算最快的任务结果作为最终结果。

3) 任务运行环境准备

通过启动 Java 虚拟机(Java Virtual Machine,JVM),将资源进行隔离,这就基本准备

好了运行环境，都是通过 Task Tracker 来实现的。Task Tracker 为每个任务启动一个独立的 JVM，为了防止任务滥用资源，采用操作系统进程实现隔离。

4）任务执行

Task Tracker 准备好任务的执行环境之后，就可以执行任务。在运行过程中，每个任务都汇报给 Task Tracker 之后再发送至 Job Tracker。

5）作业完成

如果所有任务都执行完成，整个作业就完成了。

3.5 本章小结

（1）大数据架构是用于获取和处理大量数据（通常称为“大数据”）的总体系统，因此可以针对业务目的进行分析。

（2）Hadoop 是一个能够对大量数据进行分布式处理的软件框架，实现了 Google 的 MapReduce 编程模型和框架，能够把应用程序分割成许多小的工作单元，并把这些单元放到任何集群节点上执行。在 MapReduce 中，一个准备提交执行的应用程序称为作业（Job），而从一个作业划分出运行于各个计算节点的工作单元称为任务（Task）。

（3）HDFS 将文件进行切块处理，再通过文件信息服务器 NameNode 存放切块的文件信息存放地址，实际存放数据的服务器 DataNode 存放切块后的数据。系统默认每个片块大小为 64MB，以保证寻址速度；数据会写入 3 个 DataNode 中，以保证更高的容错性。HDFS Client 帮助 NameNode 对写入读取数据进行预处理，进行文件的分块与发送读取操作。NameNode 负责为数据任务寻址。

（4）MapReduce 中有两类节点：一类是 Job Tracker，它也是一个 Master 管理节点。客户端提交一个任务，Job Tracker 把它放到一个后续队列中，在适当的时机选择一个 Job，将这个 Job 拆分成多个 Map 任务和 Reduce 任务，将任务分发给另一类节点——Task Tracker。在部署时，Task Tracker 和 HDFS 中的 DataNode 往往是同一种物理节点。这样可以保证计算是跟着数据走的，保证读取数据开销最小，即移动计算。


视频讲解

3.6 实训

1. 实训目的

通过本实训掌握 Hadoop 平台的安装和配置。

2. 实训内容

搭建 Hadoop 平台的第 1 步，就是根据实际情况选择最合适的 Hadoop 版本。目前，由于 Hadoop 飞速发展，功能更新和错误修复在不断地迭代，所以版本特别多，显得有些杂乱。结合想要的功能和是否稳定，这里选择 CDH5，该版本是目前生产环境中装机量最大的版本之一，涵盖了 Hadoop 的主要功能和模块，稳定且有很多有用的新特性。下载地址为 https://archive.cloudera.com/cdh5/cdh/5/hadoop-2.6.0-cdh5.6.0.tar.gz。

Hadoop 的运行环境有以下两种。

(1) Windows：虽然目前 Hadoop 社区已经支持 Windows，但由于 Windows 操作系统本身不适合作为服务器操作系统，所以本书不介绍 Windows 下 Hadoop 的安装方式。

(2) Linux：Hadoop 的最佳运行环境无疑是世界上最成功的开源操作系统 Linux。Linux 的发行版本众多，常见的有 CentOS、Ubuntu、RedHat 等。本书选择 CentOS。

使用 VMware 虚拟机创建一个 Linux 系统，如图 3-12 和图 3-13 所示。

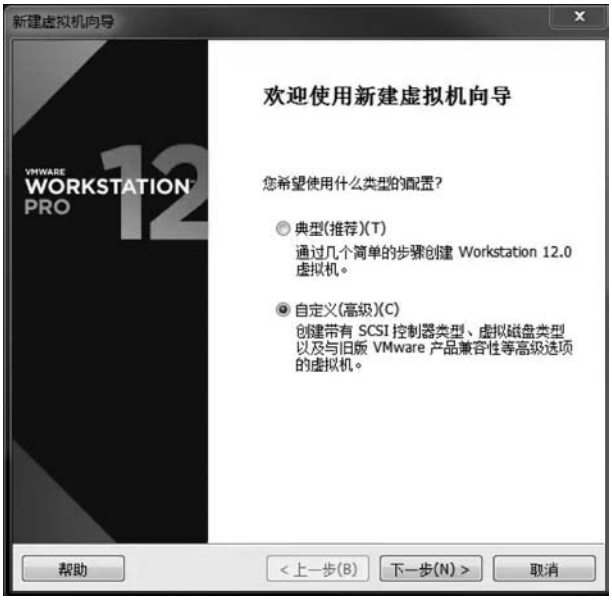

图 3-12　新建虚拟机

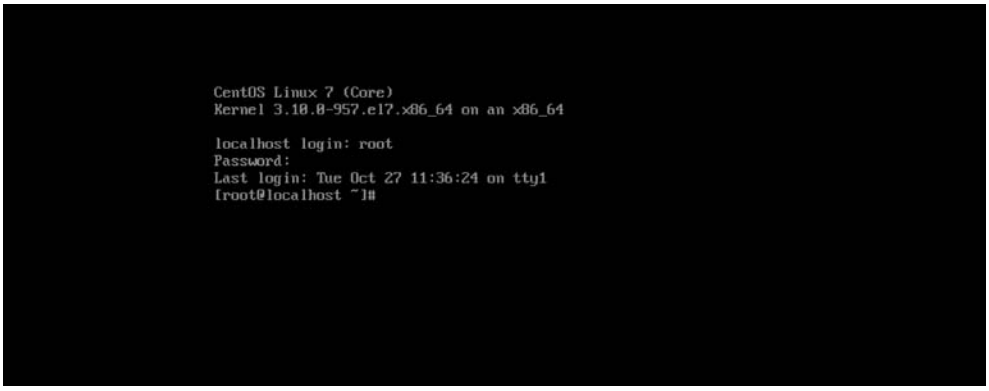

图 3-13　创建 Linux 系统

在 Linux 中搭建 Hadoop 伪分布式平台，如图 3-14 所示。

在 HDFS 上创建一个目录 input，将 Hadoop 的配置文件 core-site.xml 上传至该目录，如图 3-15 所示。

使用 MapReduce 自带的 wordcount 程序对 core-site.xml 文件进行词频统计，如

图 3-16 和图 3-17 所示。

```
[root@node01 hadoop_data]# jps
10336 NodeManager
9907 DataNode
12419 JobHistoryServer
9782 NameNode
12662 Jps
10073 SecondaryNameNode
10635 RunJar
```

图 3-14　搭建 Hadoop 伪分布式平台

/input　　Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	1.1 KB	2020/9/9 下午7:20:51	1	128 MB	core-site.xml

图 3-15　将 Hadoop 的配置文件 core-site. xml 上传至 input 目录

```
[root@node01 hadoop]# yarn jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar wordcount /input/core-site.xml /
output
20/09/09 19:27:42 INFO client.RMProxy: Connecting to ResourceManager at node01/192.168.242.101:8032
20/09/09 19:27:43 INFO input.FileInputFormat: Total input paths to process : 1
20/09/09 19:27:43 INFO mapreduce.JobSubmitter: number of splits:1
20/09/09 19:27:43 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1599644045105_0001
20/09/09 19:27:44 INFO impl.YarnClientImpl: Submitted application application_1599644045105_0001
20/09/09 19:27:44 INFO mapreduce.Job: The url to track the job: http://node01:8088/proxy/application_1599644045105_0001/
20/09/09 19:27:44 INFO mapreduce.Job: Running job: job_1599644045105_0001
20/09/09 19:27:56 INFO mapreduce.Job: Job job_1599644045105_0001 running in uber mode : false
20/09/09 19:27:56 INFO mapreduce.Job:  map 0% reduce 0%
20/09/09 19:28:02 INFO mapreduce.Job:  map 100% reduce 0%
20/09/09 19:28:09 INFO mapreduce.Job:  map 100% reduce 100%
20/09/09 19:28:10 INFO mapreduce.Job: Job job_1599644045105_0001 completed successfully
20/09/09 19:28:10 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=1452
                FILE: Number of bytes written=240389
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1228
                HDFS: Number of bytes written=1090
                HDFS: Number of read operations=6
```

图 3-16　统计词频

```
[root@node01 hadoop]# hadoop fs -text /output/part*
"AS     1
"License");     1
(the    1
-->     5
2.0     1
<!--    5
</configuration>        1
</property>     2
<?xml   1
<?xml-stylesheet        1
<configuration> 1
<name>fs.defaultFS</name>       1
<name>hadoop.tmp.dir</name>     1
<property>      2
<value>/usr/local/hadoop/data/tmp</value>       1
<value>hdfs://node01:8020</value>       1
ANY     1
Apache  1
BASIS,  1
CONDITIONS      1
IS"     1
KIND,   1
LICENSE 1
License 3
License,        1
```

图 3-17　统计词频结果

访问 MapReduce 的 Web UI 界面，并查看详细的日志信息，如图 3-18 所示。

图 3-18　查看详细的日志信息

习题 3

（1）举例说明 Hadoop 的体系结构。

（2）HDFS 中数据副本的存放策略是什么？

（3）NameNode 和 DataNode 的功能分别是什么？

（4）根据自己的理解画出 HDFS 文件系统中文件读取的流程，并解释其中的各个步骤。

（5）根据自己的理解画出 HDFS 文件系统中文件写入的流程，并解释其中的各个步骤。