项目 1

Hadoop 基础知识

本项目主要讲述 Hadoop 的发展历史、体系架构，介绍 Hadoop 与分布式开发及 Hadoop 的应用案例分析。本项目共分 4 个任务，任务 1 主要介绍 Hadoop 的发展历史；任务 2 主要介绍 Hadoop 的体系架构；任务 3 主要介绍 Hadoop 与分布式开发；任务 4 对 Hadoop 的应用案例进行分析。

通过本项目的学习，应达到以下目标。

- 了解 Hadoop 的发展历史。
- 理解 Hadoop 的体系架构。
- 掌握 Hadoop 与分布式开发。
- 了解 Hadoop 的应用案例。

任务 1　认识 Hadoop

1.1.1　Hadoop 的由来

Hadoop 是通过开源代码形式提供的软件平台产品，近年来 Hadoop 在很多开源项目中受到了广泛的关注，虽然它的历史并不长，但在近期产生的云计算生态界中，Hadoop 引起了广泛的关注，并被许多企业应用。想要了解 Hadoop 的相关信息及最新进展，读者可以关注 http://hadoop.apache.org/。下面对 Hadoop 的由来进行一下梳理。

2002 年，开源组织 Apache 成立开源搜索引擎项目 Nutch，但在 Nutch 开发过程中，始终无法有效地将计算任务分配到多台计算机上。2004 年前后，Google 陆续发表三大论文 GFS、MapReduce 和 BigTable，于是 Apache 在其 Nutch 里借鉴了 GFS 和 MapReduce 思想，实现了 Nutch 版的 NDFS 和 MapReduce。但 Nutch 项目侧重搜索，而 NDFS 和 MapReduce 则更像是分布式基础架构，因此，2006 年，开发人员将 NDFS 和 MapReduce 移出 Nutch，形成独立项目，称为 Hadoop。

说到 Hadoop，就不得不提到 Doug Cutting，如图 1-1 所示。1985 年，Cutting 毕业于美

国斯坦福大学。他并不是一开始就决心投身 IT 行业，在大学时代的头两年，Cutting 学习了诸如物理、地理等常规课程。因为学费的压力，Cutting 开始意识到，自己必须学习一些更加实用、有趣的技能。这样一方面可以帮助自己还清贷款，另一方面也是为自己未来的生活做打算。因为斯坦福大学坐落在 IT 行业的“圣地”——硅谷，所以学习软件对年轻人来说是再自然不过的事情了。

图 1-1 “Hadoop 之父”Doug Cutting

Cutting 的第一份工作是在 Xerox 做实习生，Xerox 当时在激光扫描仪上运行着 3 个不同的操作系统，其中的一个操作系统还没有屏幕保护程序。因此，Cutting 就开始为这套系统开发屏幕保护程序。由于这套程序是基于系统底层开发的，因此，其他同事可以给这个程序添加不同的主题。这份工作给了 Cutting 一定的满足感，也是他最早的“平台”级的作品。

可以说，Xerox 对 Cutting 后来的研究搜索技术起到了决定性的影响，除了短暂的在苏格兰工作的经历外，Cutting 事业的起步阶段大部分都是在 Xerox 度过的，这段时间让他在搜索技术的知识上有了很大提高。他花了 4 年的时间搞研发，这 4 年中，他阅读了大量的论文，同时，自己也发表了很多论文，用 Cutting 自己的话说：“我的研究生是在 Xerox 读的。”

尽管 Xerox 让 Cutting 积累了不少技术知识，但他却认为，自己当时搞的这些研究只是纸上谈兵，没有人试验过这些理论的可实践性。于是，他决定勇敢地迈出这一步，让搜索技术可以为更多人所用。1997 年年底，Cutting 开始以每周两天的时间投入，在家里试着用 Java 把这个想法变成现实，不久之后，Lucene 诞生了。作为第一个提供全文文本搜索的开源函数库，Lucene 的伟大自不必多言。

之后，Cutting 再接再厉，在 Lucene 的基础上将开源的思想继续深化。2004 年，Cutting 和同为程序员出身的 Mike Cafarella 决定开发一款可以代替当时的主流搜索产品的开源搜索引擎，这个项目被命名为 Nutch。在此之前，Cutting 所在的公司 Architext（其主要产品为 Excite 搜索引擎）因没有顶住互联网经济泡沫的冲击而破产，那时的 Cutting 正处在自由职业者的生涯中，所以他希望自己的项目能通过一种低开销的方式来构建网页中的大量算法。幸运的是，Google 这时正好发布了一项研究报告，报告中介绍了两款 Google 为支持自家的搜索引擎而开发的软件平台。这两个平台，一个是 GFS（Google File System），用于存储不同设备所产生的海量数据；另一个是 MapReduce，它运行在 GFS 之上，负责分布式大规模数据。基于这两个平台，Cutting 最引人注目的作品——Hadoop 诞生了。谈到 Google 对他们的“帮助”，Cutting 说：“我们开始设想用 4～5 台计算机来实现这个项目，但在实际运行中牵涉了大量烦琐的步骤，需要靠人工来完成。Google 的平台让这些步骤得以自动化，为我们实现整体框架打下了良好的基础。”

说起 Google，Cutting 也是它成长的见证人之一，这里有一段鲜为人知的故事。早在 Cutting 供职于 Architext 期间，有两个年轻人曾去拜访这家公司，并向他们兜售自己的搜索技术，但当时他们的 Demo 只检索出几百万条网页信息，Excite 的工程师们觉得他们的技术太小儿科，于是就把他们给送走了。但故事并未到此结束，这两个年轻人回去之后痛定思

痛，决定自己创业。于是，他们开了一家自己的搜索公司，取名为 Google。这两个年轻人就是 Larry Page 和 Sergey Brin。在 Cutting 看来，Google 的成功主要取决于反向排序之后再存储的设计和对自己技术的自信。

关于 Hadoop 名字的来源也很有意思，这个名字不是一个缩写，而是一个虚构的名字。Doug Cutting 如此解释 Hadoop 的得名：这个名字是我孩子给一头吃饱了的棕黄色大象命名的。我的命名标准就是简短，容易发音和拼写，没有太多的意义，并且不会被用于别处。小孩子是这方面的高手。Google 就是由小孩命名的。Hadoop 的图标如图 1-2 所示。

图 1-2　Hadoop 的图标

Hadoop 使分散处理环境中的数据处理变得更加快速，在 Hadoop 作为顶尖的项目到达巅峰时，大部分的硬盘容量为 1 TB，每秒可读取的容量为 100 MB 左右。因此，如果要读取硬盘全部 1 TB 的容量，至少需要等待两个半小时。然而，Hadoop 因为建立在分布式环境中，所以它可以用其他方式无法比拟的速度对数据进行快速处理。2007 年，Hadoop 完成 1 TB 磁盘数据读取和排序仅需要 297 秒。但 Hadoop 并没有止步于此，通过技术的不断发展，2008 年将上述时间缩短到 209 秒，同年 11 月又缩短到 68 秒。2009 年 5 月，Yahoo 通过使用 Hadoop 在 62 秒内便完成了磁盘 1 TB 数据的读取和排序后的再次存储。实质上，在极短的时间内，Hadoop 便实现了技术的高速发展。

在此以后，Hadoop 仍然在不断发展，根据多种多样的需求不断地变化。然而，由于种种 Bug 和性能的问题，在使用了几年 1.0 以下版本后，Hadoop 终于在 2011 年 12 月 27 日推出了划时代的 1.0 版本。

1.1.2　关于 Hadoop 的版本

由于 Hadoop 的开源特性及组件众多，对于初学者来说还是有一定难度的。目前 Hadoop 的发行版除了 Apache 的开源版本之外，还有华为发行版、Intel 发行版、Cloudera 发行版(CDH)、Hortonworks 发行版(HDP)、MapR 等，所有这些发行版均是基于 Apache Hadoop 衍生出来的，因为 Apache Hadoop 的开源协议允许任何人对其进行修改并作为开源或商业产品发布。国内大多数公司发行版是收费的，如 Intel 发行版、华为发行版等。不收费的 Hadoop 版本主要有国外的 4 个，分别是 Apache 基金会 Hadoop、Cloudera 版本(CDH)、Hortonworks 版本(HDP)、MapR 版本。

1. 第一代和第二代

Apache Hadoop 版本分为两代，第一代 Hadoop 称为 Hadoop 1.0，第二代 Hadoop 称为 Hadoop 2.0。第一代 Hadoop 包含 3 个大版本，分别是 0.20.x，0.21.x 和 0.22.x，其中，0.20.x 最后演化成 1.0.x，变成了稳定版，而 0.21.x 和 0.22.x 增加了 NameNode HA 等新的重大特性。第二代 Hadoop 包含两个版本，分别是 0.23.x 和 2.x，它们完全不同于 Hadoop 1.0，是一套全新的架构，均包含 HDFS Federation 和 YARN 两个系统，相比于 0.23.x，2.x 增加了 NameNodeHA 和 Wire-compatibility 两个重大特性。经过上面的大体解释，大家可能明白 Hadoop 以重大特性区分各个版本，总结起来，用于区分 Hadoop 版本的

特性有以下几个。

(1)Append 支持文件追加功能,如果想使用 HBase,需要这个特性。

(2)RAID 在保证数据可靠的前提下,通过引入校验码减少数据块数目。

(3)Symlinks 支持 HDFS 文件链接。

(4)Security Hadoop 安全。

需要注意的是,Hadoop 2.0 主要由 Yahoo 独立出来的 Hortonworks 公司主持开发。2013 年 10 月,Hadoop 2.0 发布,关键特性包括以下内容。

(1)YARN。YARN 是"Yet Another Resource Negotiator"的简称,它是 Hadoop 2.0 引入的一个全新的通用资源管理系统,可在其上运行各种应用程序和框架,如 MapReduce、Tez、Storm 等,它的引入使得各种应用运行在一个集群中成为可能。YARN 是在 MRv1 基础上衍化而来的,是 MapReduce 发展到一定程度的必然产物,它的出现使得 Hadoop 计算类应用进入平台化时代。

(2)HDFS 单点故障得以解决。Hadoop 2.2.0 同时解决了 NameNode 单点故障问题和内存受限问题,其中,单点故障是通过主备 NameNode 切换实现的,这是一种古老的解决服务单点故障的方案,主备 NameNode 之间通过一个共享系统存储同步元数据信息,因此共享存储系统的选择成为关键,而 Hadoop 则提供了 NFS、QJM 和 Bookeeper 三种可选的共享存储系统。

(3)HDFS Federation。NameNode 内存受限问题也在 2.2.0 版本中得到了解决。这是通过 HDFS Federation 实现的,它允许一个 HDFS 集群中存在多个 NameNode,每个 NameNode 分管一部分目录,而不同 NameNode 之间彼此独立,共享所有 DataNode 的存储资源。注意,NameNode Federation 中的每个 NameNode 仍存在单点问题,需要为每个 NameNode 提供一个 backup 以解决单点故障问题。

(4)HDFS 快照。HDFS 快照是指 HDFS 文件系统(或子系统)在某一时刻的只读镜像,它的出现使得管理员可定时为重要文件或目录做快照,以防止数据误删、丢失等。具体可阅读 Snapshots for HDFS(使用说明),Support for RW/RO snapshots in HDFS。

(5)通过 NFSv3 访问 HDFS。NFS 允许用户像访问本地文件系统一样访问远程文件系统,而将 NFS 引入 HDFS 后,用户可像读写本地文件一样读写 HDFS 上的文件,这大大简化了 HDFS 的使用,这是通过引入一个 NFS gateway 服务实现的,该服务能将 NFS 协议转换为 HDFS 访问协议。

(6)支持 Windows 操作系统。在 2.2.0 版本之前,Hadoop 仅支持 Linux 操作系统,而 Windows 仅作为实验平台使用。从 2.2.0 版本开始,Hadoop 支持 Windows 操作系统。

(7)兼容 1.x 上运行的 MapReduce 应用程序,与 Hadoop 生态系统其他系统进行了充分的集成测试。除了 HDFS、MapReduce 和 YARN 这 3 个核心系统外,Hadoop 生态系统还包括 HBase、Hive、Pig 等系统,这些系统底层依赖于 Hadoop 内核,而相比于 Hadoop 1.0,Hadoop 2.0 的最大变化出现在内核(HDFS、MapReduce 和 YARN),但与生态系统中其他系统进行集成测试是必需的。

除了以上特性外,Apache 官方还给出了两个特殊说明。

(1)HDFS 变化。HDFS 的 symlinks(类似于 Linux 中的软链接)被迁移到 2.3.0 版本中。

(2)YARN/MapReduce 注意事项。管理员在 NodeManager 上设置 ShuffleHandler

service 时，要采用 mapreduce_shuffle，而非之前的 mapreduce. shuffle 作为属性值。

2. Hadoop 2. 3. 0

新版本不仅增强了核心平台的大量功能，还修复了大量 bug。新版本对 HDFS 做了两个非常重要的增强：支持异构的存储层次和通过数据节点为存储在 HDFS 中的数据提供内存缓存功能。

借助 HDFS 对异构存储层次的支持，人们将能够在同一个 Hadoop 集群上使用不同的存储类型。此外，还可以使用不同的存储媒介（如商业磁盘、企业级磁盘、SSD、内存等），更好地权衡成本和收益。类似的，在新版本中还能使用 Hadoop 集群中的可用内存集中地缓存并管理数据节点内存中的数据集。MapReduce、Hive、Pig 等类似的应用程序将能够申请内存进行缓存，然后直接从数据节点的地址空间中读取内容，通过完全避免磁盘操作极大地提高了扫描效率。Hive 现在正在为 ORC 文件实现一个非常有效的零复制读取路径，该功能就使用了这项新技术。

在 YARN 方面，资源管理器自动故障转移功能的研发已经进入尾声。此外，2. 3. 0 版本还对 YARN 做了一些关键的运维方面的增强，如更好的日志、错误处理和诊断等。

MapReduce 的一个关键增强是 MAPREDUCE-4421。借助该功能，人们已经不再需要在每一台机器上安装 MapReduce 二进制程序，仅仅通过 YARN 分布式缓存将一个 MapReduce 包复制到 HDFS 中就可以了。当然，新版本还包含大量的 bug 修复及其他方面的增强。例如：

(1) YarnClientImpl 类中的异步轮询操作引入了超时。

(2) 修复了 RMFatalEventDispatcher 没有记录事件原因的问题。

(3) HA 配置不会影响节点管理器的 RPC 地址。

(4) RM Web UI 和 REST API 统一使用 YarnApplicationState。

(5) 在 RpcResponseHeader 中包含 RPC 错误信息，而不是将其分开发送。

(6) 向 jetty/httpserver 中添加了请求日志。

(7) 修复了将 dfs. checksum. type 定义为 NULL 之后写文件和 hflush 会抛出 java. lang. ArrayIndexOutOfBoundsException 的问题。

3. Hadoop 2. 4. 0

2014 年 4 月，Hadoop 2. 4. 0 发布，关键特性包括：

(1) HDFS 支持访问控制列表(access control lists，ACLs)。

(2) 原生支持 HDFS 滚动升级。

(3) HDFS FsImage 用到了 protocol-buffers，从而可以平滑地升级。

(4) HDFS 完全支持 HTTPS。

(5) YARN ResourceManager 支持自动故障转移，解决了 YARN ResourceManager 的单点故障。

(6) 对 YARN 的 Application History Server 和 Application Timeline Server 上的新应用加强了支持。

(7) 通过抢占使得 YARN Capacity Scheduler 支持强 SLAs 协议。

安全对于 Hadoop 来说至关重要，所以在 Hadoop 2. 4. 0 版本中对 HDFS 的所有访问

(包括 WebHDFS、HSFTP 甚至是 Web interfaces)都支持 HTTPS。在 Hadoop 2.4.0 中解决了 ResourceManager 的单点故障,这样会在集群中存在两个 ResourceManager,其中一个处于 active,另一个处于 standby。当 active 出现故障时,Hadoop 可以自动平滑地切换到另外一个 ResourceManager,这个新的 ResourceManager 将会自动地重启那些提交的 Applications。在下一阶段,Hadoop 将会增加一个热 standby,这个 standby 可以继续从故障点运行应用程序,以保存任何已经完成的工作。

4. Hadoop 2.5.0

2014 年 8 月,Hadoop 2.5.0 发布,关键特性包括:

(1)Common。

①使用 HTTP 代理服务器时的认证改进。当通过代理服务器使用 WebHDFS 时,这是非常有用的。

②增加了一个新的 Hadoop 指标监控 sink,允许直接写到 Graphite。

③Hadoop 文件系统兼容相关的规范工作。

(2)HDFS。

①支持 POSIX 风格的扩展文件系统。更多的细节可以查看 Extended Attributes in HDFS 文档。

②支持离线 image 浏览,客户端现在可以通过 WebHDFS 的 API 浏览一个 FsImage。

③NFS 网关得到大量可支持性的改进和 bug 修复。Hadoop portmapper 不再需要运行网关,网关现在可以拒绝没有权限的端口的连接。

④SecondaryNameNode、JournalNode 和 DataNode 的 Web UI 已经使用 HTML5 和 JavaScript 美化。

(3)YARN。

①YARN 的 REST API 支持写/修改操作。用户可以用 REST API 提交和杀死应用程序。

②时间线存储到 YARN,用来存储一个应用通用的和特殊的信息,支持 Kerberos 认证。

③公平调度器支持动态分层用户队列,运行时,用户队列在任一指定的父队列中被动态地创建。

5. Hadoop 2.6.0

2014 年 11 月,Hadoop 2.6.0 发布。它是市场上企业应用最多、与其他发行版结合得最好的版本,推荐用这个版本。其关键特性包括:

(1)Common。Hadoop Key Management Server(KMS)是一个基于 HadoopKeyProvider API 编写的密钥管理服务器。它提供了一个 client 和一个 server 组件,client 和 server 之间基于 HTTP,使用 REST API 通信。client 是一个 KeyProvider 的实现,使用 KMS HTTP REST API 与 KMS 交互。KMS 和它的 client 有内置的安全机制,支持 HTTP SPNEGO Kerberos 认证和 HTTPS 安全传输。KMS 是一个 Java Web 应用程序,运行在与 Hadoop 发行版绑定在一起的预先配置好的 Tomcat 服务器上。

(2)Tracing。HDFS-5274 增加了追踪通过 HDFS 的请求的功能,此功能使用了开源的库 HTrace。

(3)HDFS。

①TransparentEncryption,HDFS 实现了一个透明的、端到端的加密方式。一旦配置了加密,从 HDFS 读出数据解密和写入数据加密的过程对用户应用程序代码来说都是透明的。加密过程是端到端的,这意味着数据只能在客户端被加密和解密。HDFS 从来不存储也不访问未加密的数据和数据加密密钥。这样满足了加密过程的两个典型的需求:at-rest encryption(静态加密,也就是说,数据持久化在像磁盘这样的媒介上),in-transit encryption(在途加密,如当数据在网络中传输时)。

②StorageSSD&&Memory。ArchivalStorage(档案存储器)是将计算能力与不断增长的存储能力分离。拥有高密度低成本的存储,但计算能力较低的节点将变得可用,可以在集群中做冷存储。增加更多的节点作为冷存储可以提高集群的存储能力,与集群的计算能力无关。

(4)MapReduce。这一部分主要是一些 bug 的修复和改进,增加了两个新的特性,在 2.5.2里已经有所描述。

①ResourceManagerRestart。

②允许 AM 发送历史事件信息到 Timeline Server。

(5)YARN。

①NodeManagerRestart:这个特性可以使 NodeManager 在不丢失运行在节点中活动的 container 的情况下重新启动。

②DockerContainer Executor:DockerContainer Executor(DCE)允许 YARN NodeManager 在 Docker container 中启动 YARN container。用户可以指定他们想用来运行 YARN container 的 Docker 的镜像。这些 container 提供了一个可以自定义的软件环境,用户的代码可以运行在其中,与 NodeManager 运行的环境隔离。这些运行用户代码的 container 可以包含应用程序需要的特定的库,它们可以拥有与 NodeManager 不同版本的 Perl、Python 甚至是 Java。事实上,这些 container 可以运行与 NodeManager 所在的 OS 不同版本的 Linux。尽管 YARN container 必须定义运行 Job 所需的所有的环境和库,但是 NodeManager 中的所有东西都不会共享。

Docker 为 YARN 提供了一致和隔离两种模式。在一致模式下,所有的 YARN container 将拥有相同的软件环境;在隔离模式下,不管物理机器安装了什么都互不干扰。

6. Hadoop 2.7.0 和 Hadoop 2.7.1

2015 年 7 月,Hadoop 2.7.0 发布,关键特性包括:

(1)Common。支持 Windows Azure Storage,BLOB 作为 Hadoop 中的文件系统。

(2)HDFS。

①支持文件截断(file truncate)。

②支持每个存储类型配额。

③支持可变长度的块文件。

(3)MAPREDUCE。

①能够限制运行的 MapReduce 作业的任务。

②为非常大的 Job(有许多输出文件)加快了 FileOutputCommitter。

2015 年 7 月,Hadoop 2.7.1 发布。本版本属于稳定版本,是自 Hadoop 2.6.0 以来又一

个稳定版，同时也是 Hadoop 2.7.x 版本线的第一个稳定版本，也是 2.7 版本线的维护版本，变化不大，主要是修复了一些比较严重的 bug。

关于 Hadoop 版本的最新信息可参见 https://hadoop.apache.org/release.html。

任务 2　理解 Hadoop 体系架构

1.2.1　Hadoop 1.x 和 Hadoop 2.x 的区别

Hadoop 1.x 时期的系统架构如图 1-3 所示。

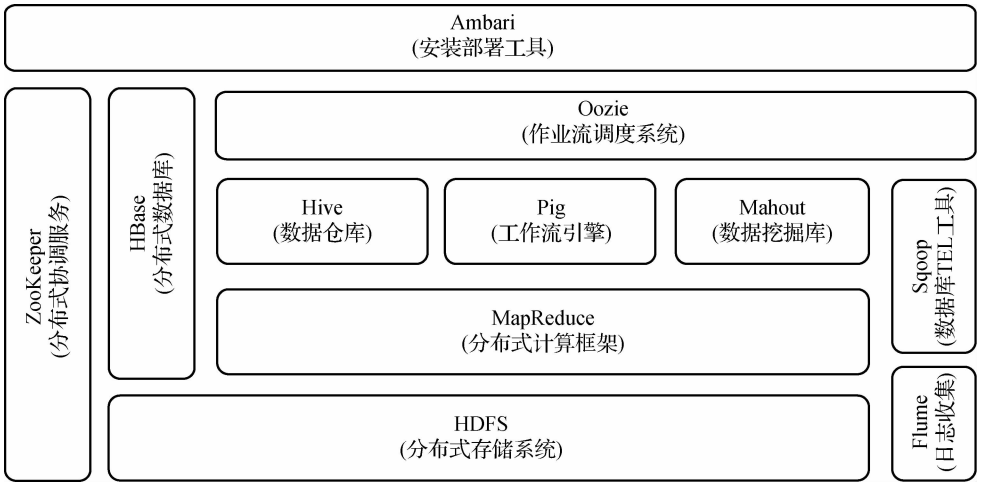

图 1-3　Hadoop1.x 时期的系统架构

Hadoop 2.x 时期的系统架构如图 1-4 所示。

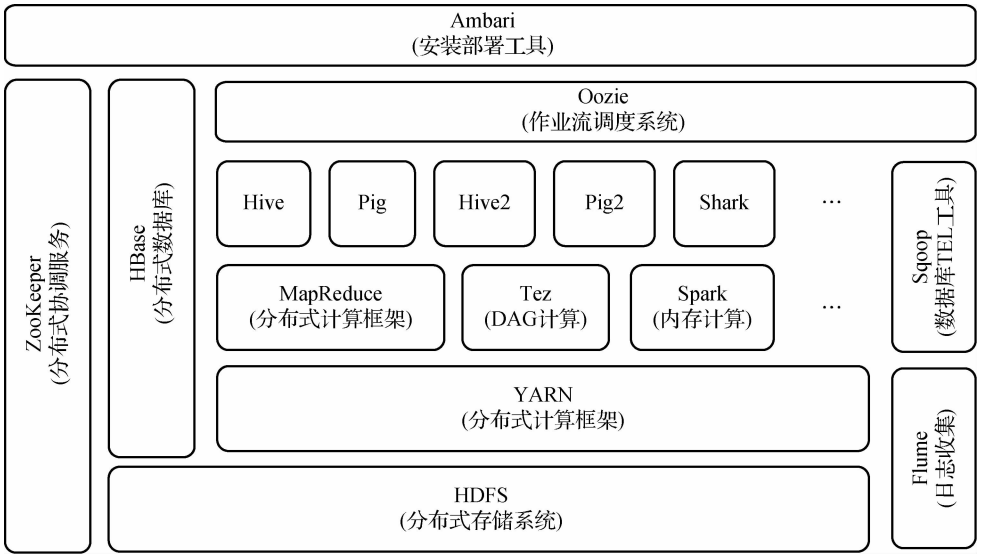

图 1-4　Hadoop 2.x 时期的系统架构

Hadoop 2. x 相比 Hadoop 1. x 最大的变化是增加了 YARN 组件，YARN 是一个资源管理和任务调度的框架，主要包含三大模块：ResourceManager（RM）、NodeManager（NM）和 ApplicationMaster（AM）。其中，ResourceManager 负责所有资源的监控、分配和管理；ApplicationMaster 负责每一个具体应用程序的调度和协调；NodeManager 负责每一个节点的维护。对于所有的 applications，RM 拥有绝对的控制权和对资源的分配权。而每个 AM 则会和 RM 协商资源，同时和 NodeManager 通信来执行和监控 task。几个模块之间的关系如图 1-5 所示。

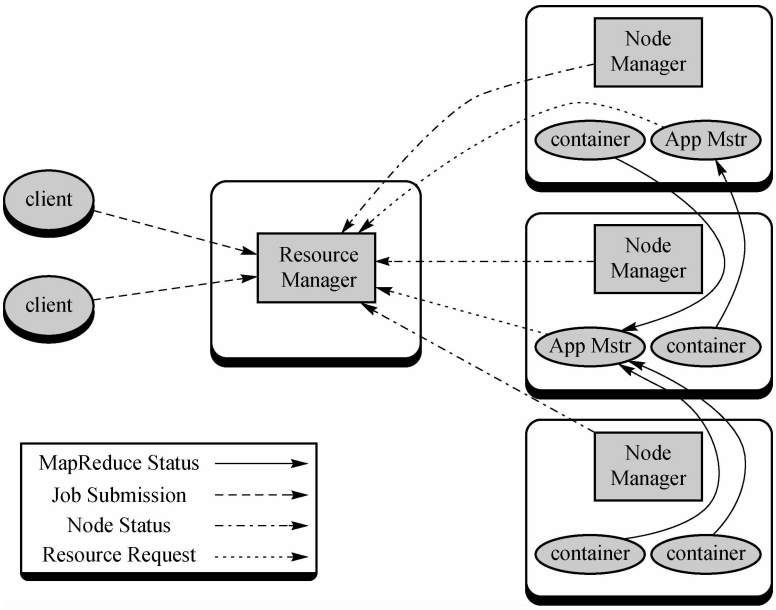

图 1-5　YARN 的模块结构

1. ResourceManager

（1）ResourceManager 负责整个集群的资源管理和分配，是一个全局的资源管理系统。

（2）NodeManager 以心跳的方式向 ResourceManager 汇报资源使用情况（目前主要是 CPU 和内存的使用情况）。RM 只接受 NM 的资源回报信息，对于具体的资源处理，则交给 NM 自行处理。

（3）YARN Scheduler 根据 application 的请求为其分配资源，不负责 application job 的监控、追踪、运行状态反馈、启动等工作。

2. NodeManager

（1）NodeManager 是每个节点上的资源和任务管理器，它是管理这台机器的代理，负责该节点程序的运行，以及该节点资源的管理和监控。YARN 集群每个节点都运行一个 NodeManager。

（2）NodeManager 定时向 ResourceManager 汇报本节点资源（CPU、内存）的使用情况和 container 的运行状态。当 ResourceManager 宕机时，NodeManager 自动连接 RM 备用节点。

（3）NodeManager 接收并处理来自 ApplicationMaster 的 container 启动、停止等各种请求。

3. ApplicationMaster

(1)用户提交的每个应用程序均包含一个 ApplicationMaster,它可以运行在 ResourceManager 以外的机器上。

(2)负责与 RM 调度器协商以获取资源(用 container 表示)。

(3)将得到的任务进一步分配给内部的任务(资源的二次分配)。

(4)与 NM 通信以启动/停止任务。

(5)监控所有任务运行状态,并在任务运行失败时重新为任务申请资源以重启任务。

(6)当前 YARN 自带了两个 ApplicationMaster 实现,一个是用于演示 AM 编写方法的实例程序 DistributedShell,它可以申请一定数目的 container 以并行运行一个 Shell 命令或 Shell 脚本;另一个是运行 MapReduce 应用程序的 AM,即 MRAppMaster。

注意:RM 只负责监控 AM,并在 AM 运行失败时启动它。RM 不负责 AM 内部任务的容错,任务的容错由 AM 完成。

YARN 的运行流程如图 1-6 所示。

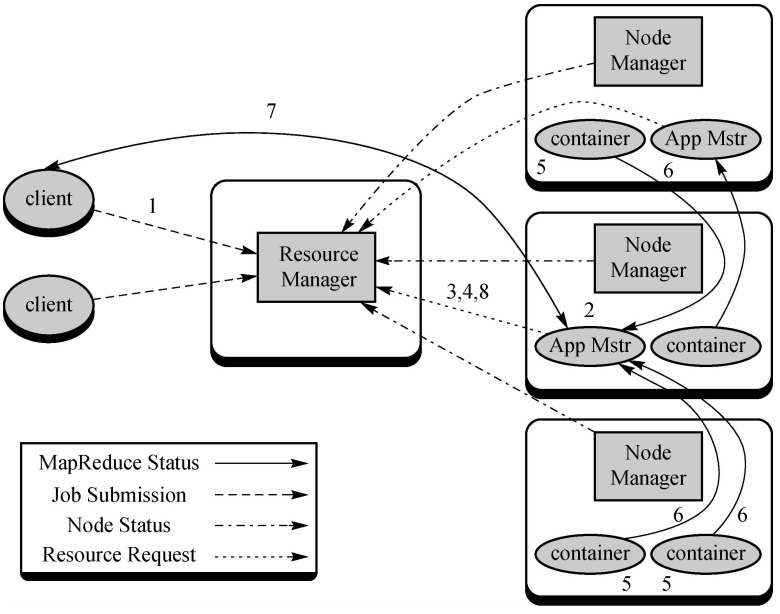

图 1-6 YARN 的运行流程

(1)client 向 RM 提交应用程序,其中包括启动该应用的 ApplicationMaster 的必需信息,如 ApplicationMaster 程序、启动 ApplicationMaster 的命令、用户程序等。

(2)ResourceManager 启动一个 container 用于运行 ApplicationMaster。

(3)启动中的 ApplicationMaster 向 ResourceManager 注册自己,启动成功后与 RM 保持心跳。

(4)ApplicationMaster 向 ResourceManager 发送请求,申请相应数目的 container。

(5)ResourceManager 返回 ApplicationMaster 申请的 container 信息。申请成功的 container 由 ApplicationMaster 进行初始化。container 的启动信息初始化后,AM 与对应的 NodeManager 通信,要求 NM 启动 container。AM 与 NM 保持心跳,从而对 NM 上运行的

任务进行监控和管理。

(6)container 运行期间,ApplicationMaster 对 container 进行监控。container 通过 RPC 协议向对应的 AM 汇报自己的进度、状态等信息。

(7)应用运行期间,client 直接与 AM 通信获取应用的状态、进度更新等信息。

(8)应用运行结束后,ApplicationMaster 向 ResourceManager 注销自己,并允许属于它的 container 被收回。

1.2.2　HDFS 架构

Hadoop 使用的底层文件系统是分布式文件系统 HDFS,它是一个能够面向大规模数据使用的、可进行扩展的文件存储与传递系统,是一种允许文件通过网络在多台主机上分享的文件系统,可让多机器上的多用户分享文件和存储空间。使实际上是通过网络来访问文件的动作,在程序与用户看来,就像是访问本地磁盘一样。即使系统中有某些节点脱机,整体来说,系统仍然可以持续运行而不会有数据损失。

HDFS 的体系结构如图 1-7 所示。

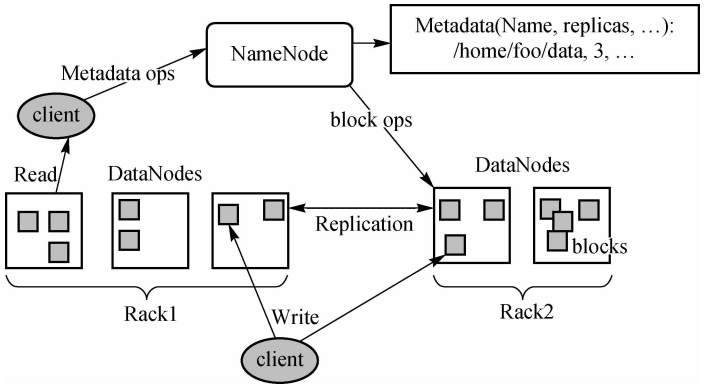

图 1-7　HDFS 的体系结构

1. NameNode

NameNode 是整个文件系统的管理节点。它维护着整个文件系统的文件/目录树,文件/目录的元信息和每个文件对应的数据块列表,接收用户的操作请求。文件包括:

(1)FsImage:元数据镜像文件。存储某一时段 NameNode 内存元数据信息。

(2)edits:操作日志文件。

(3)fstime:保存最近一次 checkpoint 的时间。

以上这些文件是保存在 Iinux 的文件系统中的,通过 hdfs-site. xml 的 dfs. namenode. name. dir 属性进行设置。

2. DataNode

DataNode 提供真实文件数据的存储服务。

(1)文件块(block):最基本的存储单位。对于文件内容而言,一个文件的长度大小是 size,那么,从文件的 0 偏移开始,按照固定的大小顺序对文件进行划分并编号,划分好的每

一个块称一个 block。HDFS 默认的 block 大小是 128 MB，因此，一个 256 MB 的文件，共有 256/128＝2 个 block。

与普通文件系统不同的是，在 HDFS 中，如果一个文件小于一个数据块的大小，并不占用整个数据块存储空间。

(2)Replication：多复本。默认是 3 个，通过 hdfs-site. xml 的 dfs. replication 属性进行设置。

HDFS 的主要优点如下。

(1)支持超大文件。超大文件在这里指的是几百 MB、几百 GB，甚至几 TB 大小的文件。一般来说，Hadoop 的文件系统会存储 TB 级别或 PB 级别的数据。因此，在企业级应用中，数据节点可能有上千个。

(2)检测和快速应对硬件故障。在集群的环境中，硬件故障是常见的问题。因为有上千台服务器连接在一起，这样会导致高故障率。因此，故障检测和自动恢复是 HDFS 文件系统的一个设计目标。

(3)流式数据访问。HDFS 的数据处理规模比较大，应用一次需要访问大量的数据，同时，这些应用一般都是批量处理，而不是用户交互式处理。应用程序能以流的形式访问数据集，重要的是数据的吞吐量，而不是访问速度。

(4)简化的一致性模型。大部分 HDFS 操作文件时，需要一次写入、多次读取。在 HDFS 中，一个文件一旦经过创建、写入和关闭，一般就不需要修改了。这样简单的一致性模型有利于提高吞吐量。

HDFS 的主要缺点如下。

(1)不适合低延迟数据访问。例如，和用户进行交互的应用，需要数据在毫秒或秒的范围内得到响应。由于 Hadoop 针对高数据吞吐量做了优化，牺牲了获取数据的延迟，因此对于低延迟来说，不适合用 Hadoop 来做。

(2)大量的小文件。HDFS 支持超大的文件，是通过数据分布在数据节点、数据的元数据保存在名字节点上来实现的。名字节点的内存大小决定了 HDFS 文件系统可保存的文件数量。虽然现在的系统内存都比较大，但大量的小文件还是会影响名字节点的性能的。

(3)不支持多用户写入和修改文件。HDFS 的文件只能有一次写入，不支持多用户写入和修改文件。只有这样，数据的吞吐量才能大。

(4)不支持超强的事务。不像关系型数据库那样对事务有强有力的支持。

1.2.3 MapReduce 架构

MapReduce 采用 Master/Slave(M/S)架构，如图 1-8 所示，主要包括 client、JobTracker、TaskTracker 和 Task 组件。

1. client

用户编写的 MapReduce 程序通过 client 提交到 JobTracker 端；同时，用户可通过 client 提供的一些接口查看作业运行状态。在 Hadoop 内部，用作业(job)表示 MapReduce 程序。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MapReduce任务(task)。

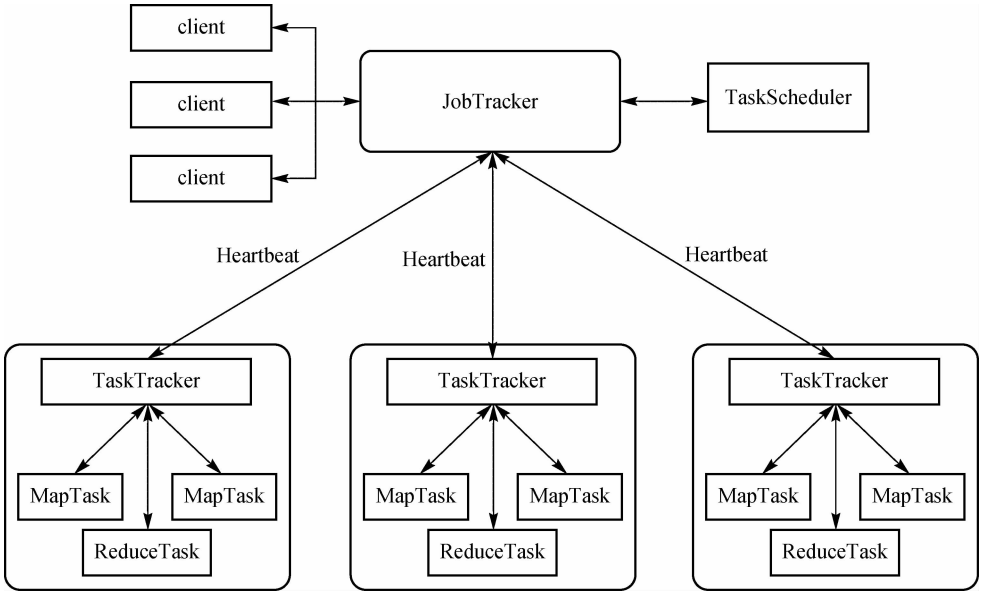

图 1-8　MapReduce 架构

2. JobTracker

JobTracker 主要负责资源监控和作业调度。JobTracker 监控所有 TaskTracker 与作业的健康状况，一旦发现失败情况后，其会将相应的任务转移到其他节点；同时 JobTracker 会跟踪任务的执行进度、资源使用量等，并将这些信息告诉给任务调度器（TaskScheduler），而任务调度器会在资源出现空闲时，选择合适的任务使用这些资源。在 Hadoop 中，任务调度器是一个可插拔的模块，用户可以根据自己的需要设计相应的 Scheduler。

3. TaskTracker

TaskTracker 会周期性地通过 Heartbeat 将本节点上资源的使用情况和任务的运行进度汇报给 JobTracker，同时接收 JobTracker 发送过来的命令并执行相应操作（如启动新任务、杀死任务等）。TaskTracker 使用 slot 等量划分本节点上的资源量。slot 代表计算资源（CPU、内存等）。一个 task 获取到一个 slot 后才有机会运行，而 Hadoop 调度器的作用就是将各个 TaskTracker 上的空闲 slot 分配给 task 使用。slot 分为 MapSlot 和 ReduceSlot 两种，分别供 MapTask 和 ReduceTask 使用。TaskTracker 通过 slot 数目（可配置参数）限定 task 的并发度。

4. Task

Task 分为 MapTask 和 ReduceTask 两种，均由 TaskTracker 启动。我们知道，HDFS 以固定大小的 block 为基本单位存储数据，而对于 MapReduce 而言，其处理单位是 split。split 与 block 的对应关系如图 1-9 所示。split 是一个逻辑概念，它只包含一些元数据信息，如数据起始位置、数据长度、数据所在节点等。它的划分方法完全由用户自己决定。但需要注意的是 split 的多少决定 MapTask 的数目，因为每个 split 会交由一个 MapTask 处理。

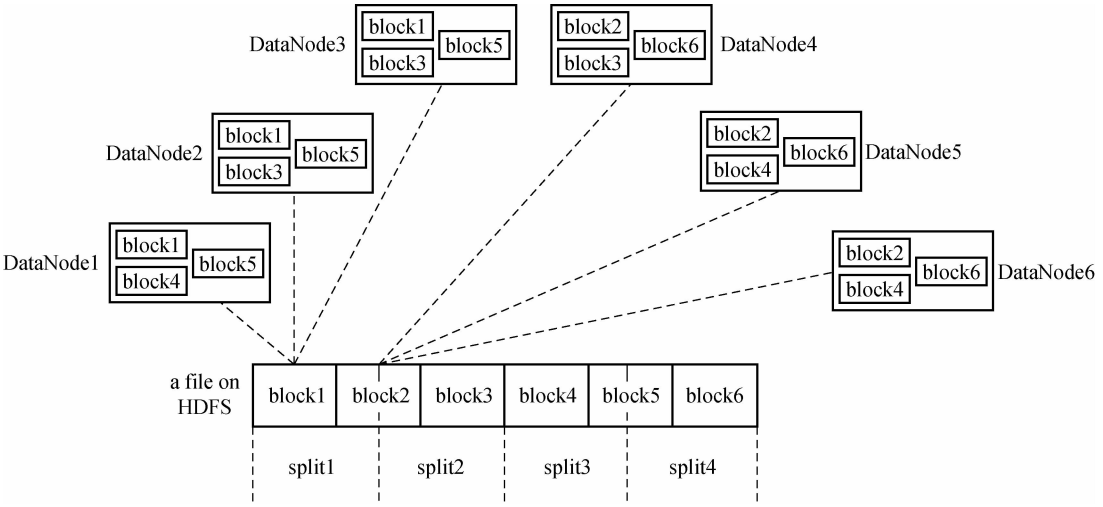

图 1-9　split 与 block 的对应关系

任务 3　理解 Hadoop 与分布式开发

Hadoop 提供的是一个分布式文件系统和一个框架，可以进行分布式应用系统的开发与搭建，下面来了解一下分布式开发的相关概念。

分布式系统(distributed system)是建立在网络之上的软件系统。正是因为软件的特性，所以分布式系统具有高度的内聚性和透明性。因此，网络和分布式系统之间的区别更多体现在高层软件(特别是操作系统)，而不是硬件。

在一个分布式系统中，一组独立的计算机展现给用户的是一个统一的整体，就好像是一个系统似的。系统拥有多种通用的物理和逻辑资源，可以动态地分配任务，分散的物理和逻辑资源通过计算机网络实现信息交换。系统中存在一个以全局的方式管理计算机资源的分布式操作系统。通常，对用户来说，分布式系统只有一个模型或范型。在操作系统之上有一层软件中间件(middleware)负责实现这个模型。一个著名的分布式系统的例子是万维网(World Wide Web)，在万维网中，所有的一切看起来就好像是一个文档(Web 页面)一样。

在计算机网络中，这种统一性、模型及其中的软件都不存在。用户看到的是实际的机器，计算机网络并没有使这些机器看起来是统一的。如果这些机器有不同的硬件或不同的操作系统，那么，这些差异对于用户来说都是完全可见的。如果一个用户希望在一台远程机器上运行一个程序，那么，他必须登录到远程机器上，然后在那台机器上运行该程序。

分布式系统和计算机网络系统的共同点是：多数分布式系统是建立在计算机网络之上的，所以分布式系统与计算机网络在物理结构上是基本相同的。

分布式系统和计算机网络系统的区别在于：分布式操作系统的设计思想和网络操作系统是不同的，这决定了它们在结构、工作方式和功能上也不同。网络操作系统要求网络用户在使用网络资源时，首先必须了解网络资源，网络用户必须知道网络中各个计算机的功能与配置、软件资源、网络文件结构等情况。在网络中如果用户要读一个共享文件，就必须知道

这个文件放在哪一台计算机的哪一个目录下；分布式操作系统是以全局方式管理系统资源的，它可以为用户任意调度网络资源，并且调度过程是透明的。当用户提交一个作业时，分布式操作系统能够根据需要在系统中选择最合适的处理器，将用户的作业提交到该处理程序，在处理器完成作业后，将结果传给用户。在这个过程中，用户并不会意识到有多个处理器的存在，这个系统就像是一个处理器一样。

内聚性是指每一个数据库分布节点高度自治，有本地的数据库管理系统。透明性是指每一个数据库分布节点对用户的应用来说都是透明的，看不出是本地还是远程。在分布式数据库系统中，用户感觉不到数据是分布的，即用户不需要知道关系是否分割、有无副本、数据存于哪个站点及事务在哪个站点上执行等。

分布式部署方案适合以下情况。

(1)公司有不同分支机构或较小的分散站点，与公司总部的网络连接通常是低带宽、高滞后或不可靠的。

(2)公司总部网络无法处理中心位置的服务流量。

(3)分支机构有自己的服务器、企业网络、域控制器和系统管理员，包含数目不定的用户。

(4)用户要求有更快的邮箱访问速度、更佳的用户体验和可用性。

(5)邮箱用户数量大，并发线程多。

(6)对于安全要求高，需要把邮件服务器不同的功能分开部署。

系统倾向于分布式发展潮流的真正驱动力是经济。20 多年前，计算机权威和评论家 Herb Grosch 指出，CPU 的计算能力与它的价格的平方成正比，后来成为 Grosch 定理。也就是说，如果用户付出两倍的价钱，就能获得 4 倍的性能。这一论断与当时的大型机技术非常吻合，因而使得许多机构都尽其所能购买最大的单个大型机。

随着微处理机技术的发展，Grosch 定理不再适用了。到了 21 世纪初期，人们只需花几百美元就能买到一个 CPU 芯片，这个芯片每秒执行的指令比 20 世纪 80 年代最大的大型机的处理机每秒所执行的指令还多。如果你愿意付出两倍的价钱，将得到同样的 CPU，但它却以更高的时钟速率运行。因此，最节约成本的办法通常是在一个系统中使用集中在一起的大量的廉价 CPU。所以，倾向于分布式系统的主要原因是它可以潜在地得到比单个的大型集中式系统好得多的性价比。实际上，分布式系统是通过较低廉的价格来实现相似的性能的。

与这一观点稍有不同的是，发现微处理机的集合不仅能产生比单个大型主机更好的性能价格比，而且能产生单个大型主机无论如何都不能达到的绝对性能。例如，按 21 世纪初期的技术，能够用 10 000 个现代 CPU 芯片组成一个系统，每个 CPU 芯片以 50 MIPS(每秒百万指令)的速率运行，那么整个系统的性能就是 500 000 MIPS。而如果单个处理机(CPU)要达到这一性能，就必须在 2×10^{-12} s(2 ps，0.002 ns)的时间内执行一条指令，然而没有一个现存的计算机能接近这个速度，从理论和工程上考虑都认为能达到这一要求的计算机都不可能存在。理论上，爱因斯坦的相对论指出光的传播速度最快，它能在 2 ps 内传播 0.6 mm。实际上，一个包含于边长为 0.6 mm 大小的立方体内的具有上面所说的计算速度的计算机产生的大量热量就能将它自己立即熔掉。所以，无论是要以低价格获得普通的性能，还是要以较高的价格获得极高的性能，分布式系统都能够满足要求。

另外，一些学者对分布式系统和并行系统进行了区分。他们认为分布式系统是用来允许众多用户一起工作的，而并行系统的唯一目标就是以最快的速度完成一个任务，就像速度为 500 000 MIPS 的计算机那样。实际上，上述说法是难以成立的，因为这两个设计领域是统一的。人们更愿意在最广泛的意义上使用“分布式系统”一词来表示任何一个有多个互连的 CPU 协同工作的系统。

建立分布式系统的另一个原因在于一些应用本身是分布式的。一个超级市场连锁店可能有许多分店，每个商店都需要采购当地生产的商品（可能来自本地的农场）、进行本地销售，或者要对本地的哪些蔬菜因时间太长或已经腐烂而必须扔掉做出决定。因此，每个商店的本地计算机能明了存货清单是有意义的，而不是集中于公司总部。毕竟，大多数查询和更新都是在本地进行的。然而，连锁超级市场的高层管理者也会不时地想要了解他们还有多少甘蓝。实现这一目标的一种途径就是将整个系统建设成对于应用程序来说就像一台计算机一样，但是在实现上它是分布的。这就是一个商业分布式系统。

另一种固有的分布式系统是通常被称为计算机支持下的协同工作系统（computer supported cooperative work，CSCW）。在这个系统中，一组相互之间在物理上距离较远的人员可以一起进行工作。例如，写出同一份报告。就计算机工业的长期发展趋势来说，人们可以很容易地想象出一个全新领域——计算机支持的协同游戏（computer supported cooperative games，CSCG）。在这个游戏中，不在同一地方的玩家可以实时地玩游戏。可以想象，在一个多维迷宫中玩电子捉迷藏游戏，甚至是一起玩一场电子空战游戏，每个人操纵自己的本地飞行模拟器去试着击落别的玩家，每个玩家的屏幕上都显示出其飞机外的情况，包括其他飞入其的视野的飞机。

同集中式系统相比，分布式系统的另一个潜在的优势在于它的高可靠性。通过把工作负载分散到众多的机器上，单个芯片故障最多只会使一台机器停机，而其他机器不会受任何影响。理想条件下，某一时刻如果有 5%的计算机出现故障，系统将仍能继续工作，只不过损失 5%的性能。对于关键性的应用，如核反应堆或飞机的控制系统，采用分布式系统来实现主要是考虑到它可以获得高可靠性。

最后，渐增式的增长方式也是分布式系统优于集中式系统的一个潜在的、重要的原因。通常，一个公司会买一台大型主机来完成所有的工作。而当公司规模扩大时，工作量就会增大，当其增大到某一程度时，这个主机就不再适用了。解决办法是要么用更大型的机器（如果有）代替现有的大型主机，要么再增加一台大型主机。这两种做法都会引起公司运转混乱。相比较之下，如果采用分布式系统，仅给系统增加一些处理机就可能解决这个问题，而且这也允许系统在需求增长时相应进行扩充。以上这些优点见表 1-1。

表 1-1　分布式系统的优点

项　　目	描　　述
经济性	微处理机提供了比大型主机更好的性价比
速度	分布式系统总的计算能力比单个大型主机更强
固有的分布性	一些应用涉及空间上分散的机器
可靠性	如果一个机器崩溃，整个系统还可以运转
渐增	计算能力可以逐渐增加

从长远的角度来看，主要的驱动力将是大量个人计算机的存在和人们共同工作与信息共享的需要，这种信息共享必须是以一种方便的形式进行的，而不受地理或人员、数据、机器的物理分布的影响。

分布式系统尽管有许多优点，但也有缺点，如前面已经提到的最棘手的问题——软件。目前最新的技术发展水平，在设计、实现及使用分布式系统上都没有太多的经验。什么样的操作系统、程序设计语言和应用适合这一系统呢？用户对分布式系统中分布式处理又了解多少呢？系统应当做多少而用户又应当做多少呢？专家们的观点不一（这并不是因为专家们与众不同，而是因为对于分布式系统他们也很少涉及）。随着更多研究的进行，这些问题将会逐渐减少。但是不应该低估这个问题。

第二个潜在的问题是通信网络。由于分布式系统会损失信息，因此就需要用专门的软件进行恢复。同时，网络还会产生过载。当网络负载趋于饱和时，必须对它进行改造替换或加入另外一个网络扩容。在这两种情况下，一个或多个建筑中的某些部分必须花费很高的费用进行重新布线，或更换网络接口板（如用光纤）。一旦系统依赖于网络，那么网络的信息丢失或饱和将会抵消通过建立分布式系统所获得的大部分优势。

最后，上面作为优点来描述的分布式系统使数据易于共享的特点也具有两面性。如果人们能够很方便地存取整个系统中的数据，那么他们同样也能很方便地存取与他们无关的数据。换句话说，经常要考虑系统的安全性问题。通常，对必须绝对保密的数据，使用一个专用的、不与其他任何机器相连的孤立的个人计算机进行存储的方法更可取。而且这个计算机被保存在一个上锁的十分安全的房间中，与这台计算机配套的所有磁盘存放在这个房间中的一个保险箱中。分布式系统的缺点见表 1-2。

表 1-2　分布式系统的缺点

项　　目	描　　述
软件	为分布式系统开发的软件还很少
网络	网络的信息丢失或饱和将引起其他问题
安全	保密数据的访问变得不安全

尽管存在这些潜在的问题，许多人还是认为分布式系统的优点多于缺点，并且普遍认为分布式系统在未来几年中会越来越重要。实际上，在几年之内许多机构会将它们的大多数计算机连接到大型分布式系统中，为用户提供更好、更廉价和更方便的服务。而在 10 年之后，中型或大型商业或其他机构中可能将不再存在一台孤立的计算机了。

任务 4　Hadoop 应用案例简介

1. 全球最大超市业者 WalMart

WalMart 分析顾客搜索商品的行为，找出超越竞争对手的商机。全球最大连锁超市 WalMart 利用 Hadoop 来分析顾客搜寻商品的行为，以及用户通过搜索引擎寻找到 WalMart 网站的关键词，利用这些关键词的分析结果发掘顾客需求，以规划下一季商品的促

销策略，甚至打算分析顾客在 Facebook、Twitter 等社交网站上对商品的讨论，期望能比竞争对手提前一步发现顾客需求。

2. 全球最大拍卖网站 eBay

eBay 用 Hadoop 拆解非结构性巨量数据，降低数据仓储负载。经营拍卖业务的 eBay 则用 Hadoop 来分析买卖双方在网站上的行为。eBay 拥有全世界最大的数据仓储系统，每天增加的数据量有 50 TB，光是储存就是一大挑战，更别说要分析这些数据了，而且更困难的挑战是这些数据包括结构化的数据和非结构化的数据，如照片、影片、电子邮件、用户的网站浏览 log 记录等。

eBay 分析平台高级总监 Oliver Ratzesberger 也坦言，大数据分析最大的挑战就是要同时处理结构化及非结构化的数据。

eBay 在 2013 年建立了一个软硬件整合的平台 Singularity，搭配压缩技术来解决结构化和非结构化数据分析问题。2015 年 eBay 在这个平台整合了 Hadoop 来处理非结构化数据，通过 Hadoop 来进行数据预处理，将大块的非结构化数据拆解成小型数据，再放入数据仓储系统的数据模型中分析，来加快分析速度，也减轻数据仓储系统的分析负载。

3. 全球最大信用卡公司 Visa

Visa 公司拥有一个全球最大的付费网络系统 VisaNet，作为信用卡付款验证之用。2009 年，Visa 公司每天就要处理 1.3 亿次授权交易和 140 万台 ATM 的联机存取。为了降低信用卡各种诈骗、盗领事件的损失，Visa 公司得分析每一笔事务数据，来找出可疑的交易。虽然每笔交易的数据记录只有短短 200 位，但每天 VisaNet 要处理全球上亿笔交易，两年累积的资料多达 36 TB，过去光是要分析 5 亿个用户账号之间的关联，得等一个月才能得到结果，所以，Visa 也在 2009 年导入了 Hadoop，建置了两套 Hadoop 丛集（每套不到 50 个节点），让分析时间从一个月缩短到 13 分钟，更快速地找出了可疑交易，也能更快对银行提出预警，甚至能及时阻止诈骗交易。

这套被众多企业赖以解决大数据难题的分布式计算技术，并不是一项全新的技术，早在 2006 年就出现了，而且 Hadoop 的核心技术原理，更是源自 Google 打造搜索引擎的关键技术，后来由 Yahoo 支持的开源开发团队发展成一套 Hadoop 分布式计算平台，也成为 Yahoo 内部打造搜索引擎的关键技术。

总结与回顾

本项目主要介绍了 Hadoop 项目的发展历程、Hadoop 产生的历史背景，详细介绍了 Hadoop 版本的发展过程。详细分析了 Hadoop 的体系结构，分析了分布式系统开发的特点，介绍了 Hadoop 的一些经典案例应用。

习　题

1. Hadoop 是由哪个项目发展来的？
2. Hadoop 主要有哪些版本？
3. 简要描述 Hadoop 的体系结构，分析 1.x 与 2.x 版本间的区别。
4. 简要描述分布式系统的优点有哪些。

项目 2

Hadoop 安装与配置管理

本项目主要讲述搭建和配置 Hadoop 环境的具体过程。Hadoop 是在 Linux 平台下运行的，本书选用 CentOS 作为运行平台。本项目使用 VMware 虚拟机虚拟化若干台 CentOS 虚拟机，完成单机、集群、伪集群环境的搭建。本项目共分 3 个任务，任务 1 主要介绍 Hadoop 环境的搭建与配置；任务 2 主要介绍 Hadoop 的安装模式；任务 3 主要介绍 Hadoop 的启动与验证。

通过本项目的学习，应达到以下目标。

- 能熟练使用 VMware 虚拟机软件创建和复制虚拟机。
- 能熟练使用 VMware 构建虚拟网络。
- 能熟练在 CentOS 下完成 Vmtools、JDK 的安装与配置。
- 能在 CentOS 下配置主机名、IP 地址、SSH 等。
- 能完成单机、伪分布、分布式 Hadoop 的安装与部署。
- 能完成 Hadoop 的启动和停止及验证软件配置的正确性。

任务 1　Hadoop 环境的搭建与配置

Hadoop 只能在 Linux 环境下运行，所以需要在 Windows 平台下使用 VMware 虚拟机虚拟出 3 台设备来安装和运行 Linux。

本书实例使用 CentOS，读者可以从 CentOS 官网下载。

2.1.1　安装 VMware

从 VMware 的官网（www. vmware. com）上下载一个 VMware Workstation Pro for Windows，借助 VMware Workstation Pro，可以在单台 Windows PC 上同时运行多个虚拟机系统。安装 VMware Workstation Pro 非常简单，一直单击“下一步”按钮即可安装成功，安装成功后启动软件，如图 2-1 所示。

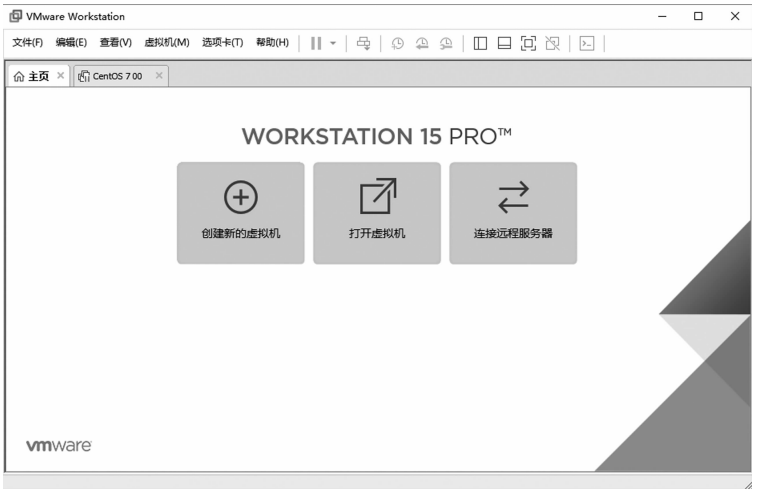

图 2-1　VMware 运行界面

2.1.2　安装 CentOS

CentOS(community enterprise operating system,社区企业操作系统)是 Linux 发行版之一,它是由 Red Hat Enterprise Linux 依照开放源代码规定释出的源代码编译而成的。

读者可以从 CentOS 的官网(www.centos.org)上下载发行版本,选择 DVD ISO 镜像。

1. 创建虚拟机

启动 VMware 虚拟机,单击“创建新的虚拟机”按钮,进入“欢迎使用新建虚拟机向导”界面,如图 2-2 所示。

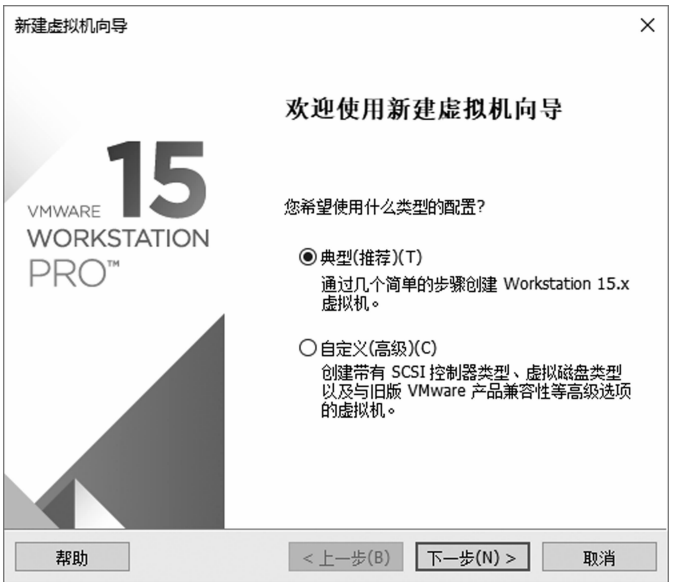

图 2-2　“欢迎使用新建虚拟机向导”界面

VMware 新建虚拟机向导提供了两种安装模式，即典型安装和自定义安装。

(1)典型安装：默认安装，对 VMware 不熟悉的用户，可以使用这种安装模式。

(2)自定义安装：用户可以按照自己的需要进行安装，使用自定义安装可以提高不同 VMware 版本的兼容性，配置 CPU、虚拟内存的大小、网络类型及 I/O 控制器类型等。

选择典型安装后，单击“下一步”按钮，进入“安装客户机操作系统”界面，如图 2-3 所示。

图 2-3　“安装客户机操作系统”界面

①安装程序光盘。这需要有光盘才能安装，把系统安装光盘放到光驱里即可。

②安装程序光盘映像文件。这是 ISO 镜像文件安装，需要找到 ISO 文件存放的位置。

③稍后安装操作系统。选中这个单选按钮后，VMWare 会先创建一个空磁盘的虚拟机，将来在虚拟机设置里重新设置安装光盘位置再继续安装系统。

单击“下一步”按钮，进入图 2-4 所示的“命名虚拟机”界面，输入虚拟机名称并选择存储位置，选择存储路径需要磁盘有足够的存储空间。

图 2-4　“命名虚拟机”界面

单击“下一步”按钮，进入图 2-5 所示的“指定磁盘容量”界面，Hadoop 实验机大概需要 20 GB 的空间。如果虚拟磁盘存储在具有文件大小限制的文件系统上，那么需要将虚拟磁盘拆分成多个文件。如果拆分的虚拟磁盘大小不到 950 GB，VMWare 会创建一系列 2 GB 大小的虚拟磁盘文件。如果虚拟磁盘大小超过 950 GB，就会创建两个虚拟磁盘文件。第一个虚拟磁盘文件最大可以达到 1.9 TB，第二个虚拟磁盘文件则存储剩余的数据。

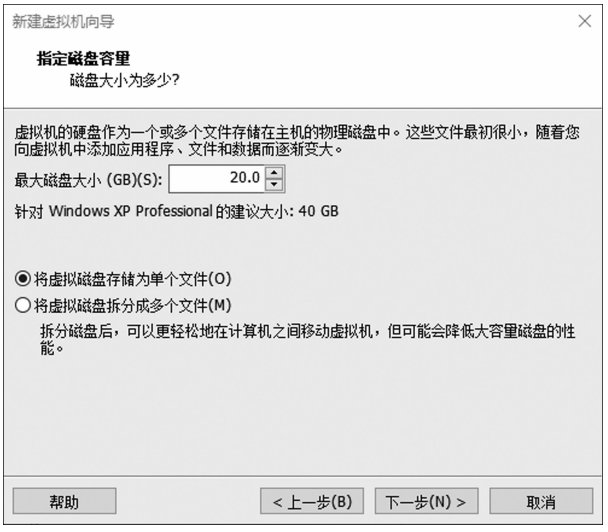

图 2-5 “指定磁盘容量”界面

这里建议选中“将虚拟磁盘存储为单个文件”单选按钮，这既便于虚拟机在不同机器上使用，也便于克隆新的虚拟机，并且能够提高文件的访问速度。

创建完虚拟机后，还可以编辑虚拟磁盘设置并添加其他虚拟磁盘。单击“下一步”按钮，进入图 2-6 所示的虚拟机确认界面，在这里可以检查前面对虚拟机的各项设置，若需要修改，则可以单击“上一步”按钮回退到设置内容，确认后，单击“完成”按钮，虚拟机就创建成功了。

图 2-6 虚拟机创建完成

2. 安装 CentOS

现在要在刚刚建立的虚拟机上安装 CentOS 操作系统。启动虚拟机前，还可以对虚拟机的硬件配置做一些修改，单击选中需要修改的设备，如图 2-7 所示。

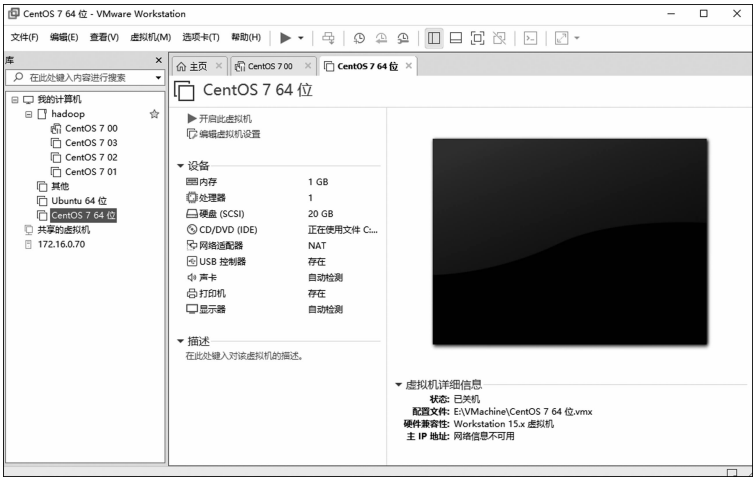

图 2-7　单击选中需要修改的设备

(1)内存：设置虚拟机内存大小，一般建议不超过主机内存的一半。

(2)处理器：设置虚拟机的处理器个数，包括核数和进程数。核数和进程数都不能超过主机 CPU 的性能参数。

(3)硬盘：磁盘的大小，可以将扩大的磁盘映射到系统的本地卷里。

(4)网络适配器：设置网络连接的方式。

(5)CD/DVD(IDE)：选择光驱的来源和设备连接的状态。

单击工具栏中的绿色按钮开启此虚拟机，启动机器进入安装操作系统的界面，如图 2-8 和图 2-9 所示。

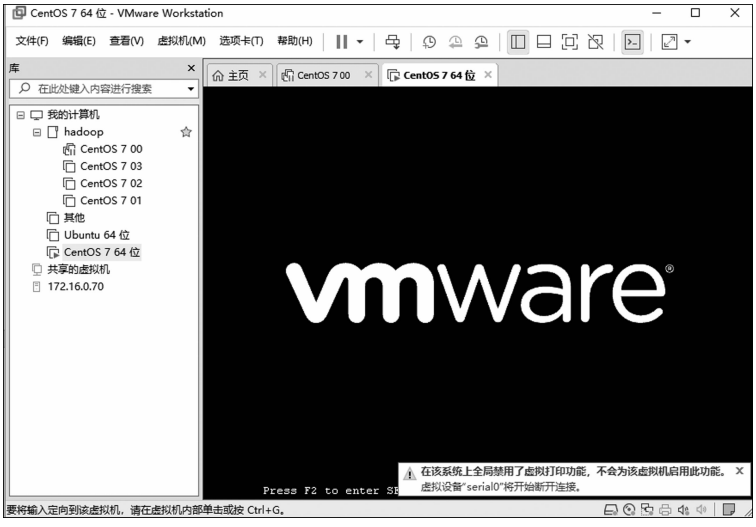

图 2-8　启动虚拟机

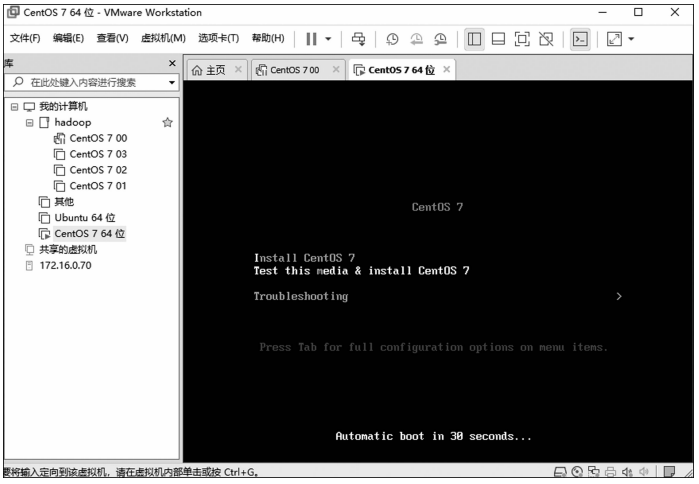

图 2-9　开始安装 CentOS

选择“Install CentOS 7”，按任意键开始安装 CentOS 7，如图 2-10 所示。

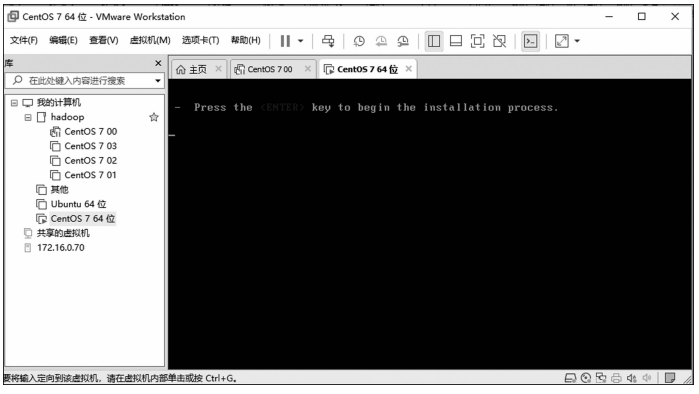

图 2-10　按任意键开始安装

CentOS 的安装界面是图形化的，只需进行必要的设置就能完成安装，第一步是语言设置，选择“简体中文”，单击“下一步”按钮，如图 2-11 所示。

图 2-11　选择语言

打开“安装信息摘要”界面，如图 2-12 所示。这里主要设置安装位置和选择软件。安装位置类似 Windows 安装过程的磁盘选择；CentOS 默认的软件安装为最小安装，这里建议打开“软件选择”界面，如图 2-13 所示，对安装软件进行设置。特别要添加 GNOME 桌面和一些常用工具，否则系统安装后仅有命令行模式，会给后续的软件安装带来很多不便。

图 2-12　“安装信息摘要”界面

图 2-13　“软件选择”界面

如果使用多个磁盘分区，还需要对安装目标位置进行设置，如图 2-14 所示。安装过程如图 2-15 所示。

图 2-14　安装目标位置设置

图 2-15　正在安装

在安装过程中或安装结束后都允许设置 ROOT 密码和创建用户账户，单击“ROOT 密码”按钮，进入图 2-16 所示的界面；单击“创建用户”按钮，进入图 2-17 所示的界面，按照提示操作即可。需要注意的是，ROOT 具有系统最高权限，应该牢记这个密码。

图 2-16　创建 ROOT 密码

图 2-17　创建用户账户

最后直到“重启”按钮从灰色变为蓝色，如图 2-18 所示，表示 CentOS 安装完成了。

图 2-18　安装完成

2.1.3　安装 JDK

JDK(Java development kit)是 Sun MicroSystems 针对 Java 开发的产品。JDK 是整个 Java 的核心，包括 Java 运行环境、Java 工具和 Java 基础类库。JDK 只要 1.5 版本以上就可以运行，读者到 Oracle 官网上下载即可。

(1)查询系统自带的 JDK。在终端输入运行的命令，查看系统是否安装了 Java，如图 2-19 所示。

```
rpm -qa|grep java
```

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.el7.x86_64 on an x86_64

localhost login: root
Password:
Last login: Thu Jan  3 19:06:05 on tty1
[root@localhost ~]# rpm -qa|grep java
tzdata-java-2018e-3.el7.noarch
javapackages-tools-3.4.1-11.el7.noarch
java-1.7.0-openjdk-headless-1.7.0.191-2.6.15.5.el7.x86_64
java-1.7.0-openjdk-1.7.0.191-2.6.15.5.el7.x86_64
python-javapackages-3.4.1-11.el7.noarch
[root@localhost ~]#
```

图 2-19　查询 Java 版本

(2)删除自带 JDK。使用 rpm-e-nodeps 命令，删除自带的 JDK，如图 2-20 所示。

```
rpm -e --nodeps java-1.7.0-openjdk-headless-1.7.0.191-2.6.15.5.el7.x86_64
rpm -e --nodeps java-1.7.0-openjdk-1.7.0.191-2.6.15.5.el7.x86_64
```

运行上面的命令，一定要注意不同的 JDK 的版本号会有区别，读者要按照自己查询到的版本号来输入。

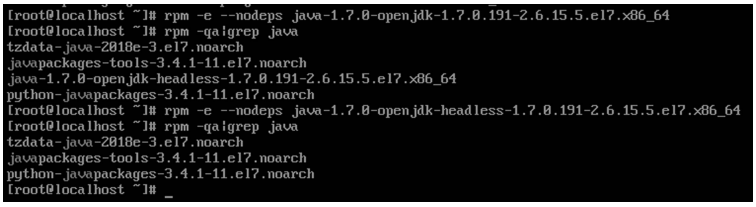

图 2-20　删除自带的 JDK

(3)安装新的 JDK,读者可以到 Oracle 网站下载最新版本的 JDK。新建 Java 目录:

```
mkdir /usr/java
```

(4)将压缩包 jdk-8u191-linux-x64.tar.gz 上传到该目录下,执行解压缩命令:

```
tar -zxvf jdk-8u191-linux-x64.tar.gz
```

再次执行 ls,该目录下会多出文件夹 jdk1.8.0_191,如图 2-21 所示。

图 2-21　建立 JDK 目录

(5)修改 profile 文件。执行命令:gedit /etc/profile,打开 profile 文件,将下列代码放入文件末尾,如图 2-22 所示。

```
export JAVA_HOME=/usr/java/jdk1.8.0_191
export JRE_HOME=JAVA_HOME/jre
export CLASSPATH=$JAVA_HOME/lib/:$JRE_HOME/lib:$CLASSPATH
export PATH=$JAVA_HOME/BIN:$JRE_HOME/BIN:$PATH
export PATH=$PATH:JAVA_HOME
```

图 2-22　修改 profile 文件

单击“保存”按钮保存并退出后,在命令行执行下列命令,使环境变量生效。

```
source /etc/profile
```

完成后，执行命令：java -version，屏幕显示如图 2-23 所示的版本信息，说明 JDK 配置成功。

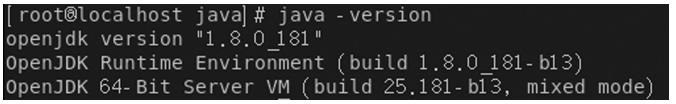

图 2-23　Java 版本信息

任务 2　Hadoop 的安装模式

Hadoop 的安装模式分为 3 种，分别是单机模式、全分布模式和伪分布模式，需要从 Hadoop 的官方网站（hadoop.apache.org）下载 hadoop-2.8.5.tar.gz 文件。

2.2.1　单机安装

单机模式是 Hadoop 的默认安装模式，这种模式所需的系统资源最少。在这种模式下，Hadoop 的 core-site.xml、mapred-site.xml 和 hdfs-site.xml 配置文件均为空。

安装过程如下。

（1）解压 Hadoop 的 tar 文件，在 usr 文件夹下运行：

```
tar -zxvf hadoop-2.8.5.tar.gz
```

（2）配置 Hadoop 环境变量。

执行命令：gedit /etc/profile，在文件末尾添加如下代码，如图 2-24 所示。

```
export HADOOP_HOME=/usr/hadoop-2.8.5
export PATH=$HADOOP_HOME/bin:$PATH
```

图 2-24　配置 Hadoop 环境变量

保存后，执行命令：source /etc/profile，完成环境加载。再执行命令：hadoop version，查看 Hadoop 是否安装成功。若出现图 2-25 所示的界面，则说明 Hadoop 安装成功。

```
[root@localhost ~]# hadoop version
Hadoop 2.8.5
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r 0b8464d75227fce
e2c6e7f2410377b3d53d3d5f8
Compiled by jdu on 2018-09-10T03:32Z
Compiled with protoc 2.5.0
From source with checksum 9942ca5c745417c14e318835f420733
This command was run using /mnt/hadoop-2.8.5/share/hadoop/common/hadoop-common-2
.8.5.jar
[root@localhost ~]#
```

图 2-25　查看 Hadoop 版本

(3)到目前为止，我们没有对 Hadoop 的配置文件做任何修改，可以执行 sbin 文件夹里的 start-all.sh 命令，测试一下 Hadoop 是否可以正常启动，如图 2-26 所示。

```
[root@localhost sbin]# ./start-dfs.sh
Incorrect configuration: namenode address dfs.namenode.servicerpc-address or dfs
.namenode.rpc-address is not configured.
Starting namenodes on []
localhost: ssh_exchange_identification: read: Connection reset by peer
localhost: ssh_exchange_identification: read: Connection reset by peer
Starting secondary namenodes [0.0.0.0]
0.0.0.0: ssh_exchange_identification: read: Connection reset by peer
[root@localhost sbin]# ./start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Incorrect configuration: namenode address dfs.namenode.servicerpc-address or dfs
.namenode.rpc-address is not configured.
Starting namenodes on []
localhost: ssh_exchange_identification: read: Connection reset by peer
localhost: ssh_exchange_identification: read: Connection reset by peer
Starting secondary namenodes [0.0.0.0]
0.0.0.0: ssh_exchange_identification: read: Connection reset by peer
starting yarn daemons
starting resourcemanager, logging to /usr/hadoop-2.8.5/logs/yarn-root-resourcema
nager-localhost.localdomain.out
localhost: ssh_exchange_identification: read: Connection reset by peer
```

图 2-26　启动单机 Hadoop

运行 hadoop fs -ls 命令，查看 HDFS 单机部署目录结构，如图 2-27 所示。

```
[root@localhost sbin]# hadoop fs -ls /
Found 20 items
drwxr-xr-x   - root     root         61440 2018-12-15 14:59 /bin
dr-xr-xr-x   - root     root          4096 2019-01-08 09:29 /boot
drwxr-xr-x   - root     root          3300 2019-01-10 16:45 /dev
drwxr-xr-x   - root     root          8192 2019-01-12 13:55 /etc
drwxr-xr-x   - root     root            21 2018-12-14 17:20 /home
drwxr-xr-x   - root     root          4096 2018-12-14 13:46 /lib
drwxr-xr-x   - root     root        106496 2018-12-14 17:15 /lib64
drwxr-xr-x   - root     root             6 2018-04-11 12:59 /media
drwxr-xr-x   - root     root            38 2019-01-10 15:54 /mnt
drwxr-xr-x   - root     root            16 2018-12-14 17:14 /opt
dr-xr-xr-x   - root     root             0 2019-01-10 16:45 /proc
dr-xr-x---   - root     root          4096 2019-01-12 14:00 /root
drwxr-xr-x   - root     root          1360 2019-01-12 10:38 /run
drwxr-xr-x   - root     root         20480 2018-12-14 13:46 /sbin
drwxr-xr-x   - root     root             6 2018-04-11 12:59 /srv
dr-xr-xr-x   - root     root             0 2019-01-10 16:45 /sys
drwxrwxrwt   - root     root          4096 2019-01-12 14:08 /tmp
drwxr-xr-x   - liumiao liumiao        214 2019-01-10 16:15 /usr
drwxr-xr-x   - root     root          4096 2018-12-14 20:44 /var
drwxr-xr-x   - root     root            67 2018-12-14 13:46 /vmware-tools
```

图 2-27　HDFS 单机部署目录结构

2.2.2 分布式安装

1. 集群的构建和拓扑图

Hadoop 主要由两部分构成：分布式文件系统 HDFS 及统一资源和调度系统 YARN，分布式文件系统主要用于海量数据的存储；YARN 主要管理集群的计算资源，并根据计算框架的需求进行调度。

HDFS 集群和 YARN 集群是由一些守护进程组成的，这些守护进程和运行它们的节点就构成了 Hadoop 集群。如图 2-28(a)所示，这个集群的 NameNode 进程和ResourceManager 进程在一个节点上运行，而 DataNode 和 NodeManager 在同一个节点上运行。出于性能和稳定性的考虑，建议将 NameNode 和 ResourceManager 分开部署，如图 2-28(b)所示，这样的部署也是一个标准的 Hadoop 集群。

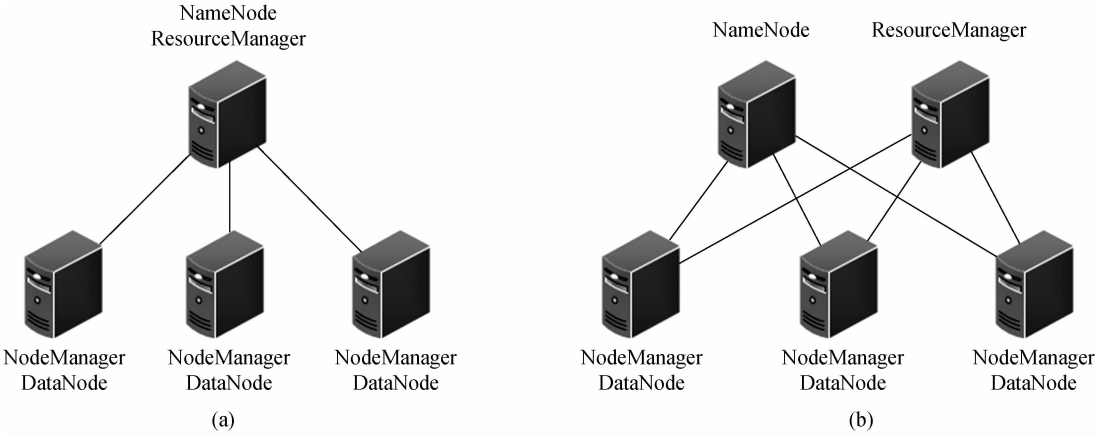

图 2-28 Hadoop 集群

2. 搭建服务器环境

首先要克隆多台服务器。在 VMWare 的库列表中找到前面使用的 CentOS 服务器，右击并选择“管理”→“克隆”命令，进入“克隆虚拟机向导”对话框。单击“下一步”按钮，在“克隆类型”界面中选中“创建完整克隆”单选按钮，如图 2-29 所示。

继续单击“下一步”按钮，设置虚拟机名称和位置，如图 2-30 所示，单击“完成”按钮，VMWare 开始克隆新的机器。

按照上述过程，还须再复制出 3 台 CentOS 虚拟机，将虚拟机按照集群的功能进行重命名，如图 2-31 所示。

虚拟机准备好后，下面开始配置虚拟网络。选择 VMware 的菜单，执行“编辑”→“虚拟网络编辑器”命令，打开“虚拟网络编辑器”对话框，如图 2-32 所示。

单击“添加网络”按钮，选中“NAT 模式”单选按钮，再选中“将主机虚拟适配器连接到此网络”复选框，如图 2-33 所示。

在 Hadoop 集群里的每一台虚拟机里，都需要将其 IP 地址设置为 192.168.201.*，将子网掩码设置为 255.255.255.0。

在 VMWare 库里，右击虚拟机，在弹出的快捷菜单中选择“设置”命令，打开“虚拟机设置”对话框，将网络连接设置为“自定义”连接，并选择“VMnet1(NAT 模式)”选项，如图 2-34 所示。

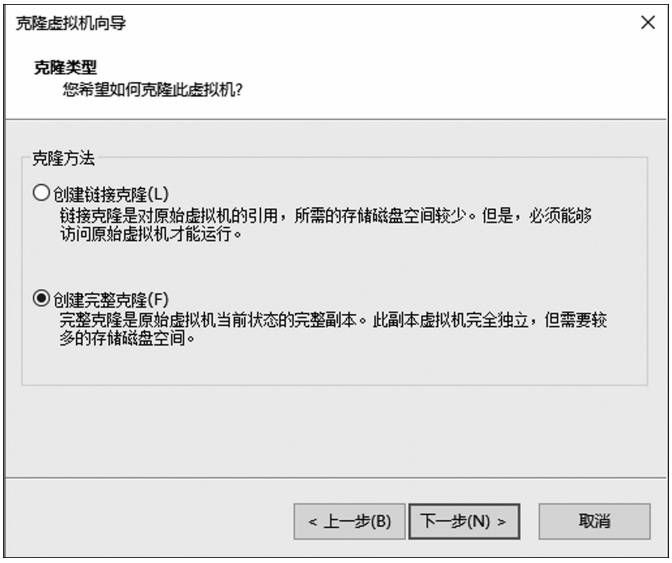

图 2-29　“克隆类型”界面

图 2-30　设置虚拟机名称和存储位置

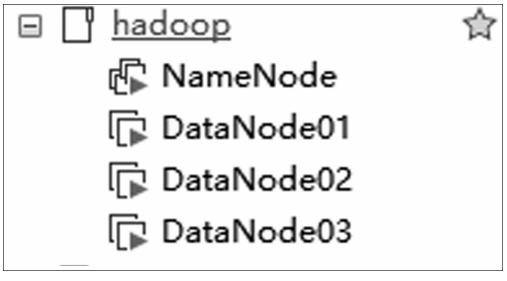

图 2-31　hadoop 机器列表

图 2-32 “虚拟网络编辑器”对话框

图 2-33 虚拟网络配置

图 2-34　“虚拟机设置”对话框

3. 修改机器配置

为了让集群里的机器能互相访问，统一开发环境，需要对主机进行配置。

1）修改用户密码、主机名和 IP 地址

执行“useradd hadoop”命令，添加以 hadoop 为用户名的用户；执行“passwd hadoop”命令，修改该用户的密码，集群里的机器密码要统一，如图 2-35 所示。

```
[root@localhost ~]# useradd hadoop
[root@localhost ~]# passwd hadoop
更改用户 hadoop 的密码 。
新的 密码：
```

图 2-35　添加用户和密码

如果安装了图形界面，这里也可以使用图形界面来完成。在系统工具里找到设置功能，选择网络，单击有线网络右侧的齿轮图标，如图 2-36 所示。

图 2-36　有线连接

在“身份”选项卡中设置主机名，如图 2-37 所示；在 IPv4 选项卡中设置 IP 地址等信息，如图 2-38 所示。

图 2-37　设置主机名

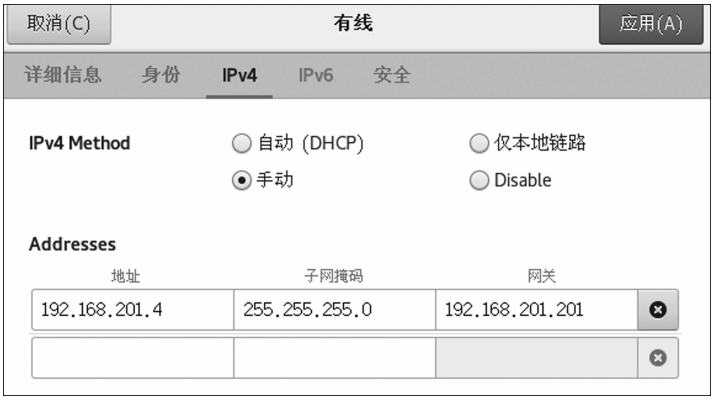

图 2-38　设置 IP 地址

分布式安装需要将一个节点的主机修改为 master1，其他节点的主机修改为 slave1、slave2 和 slave3。

将几个节点的 IP 地址设置为同一网段的机器，这个网段要和虚拟机里 VMnet 的网段一致。本书中的设置见表 2-1。

表 2-1　集群机器列表

名　　称	主机名	IP 地址
虚拟网络	VMnet	192.168.201.1～192.168.201.100
NameNode、ResourceManager	master1	192.168.201.1
DataNode	slave1	192.168.201.3
DataNode	slave2	192.168.201.4
DataNode	slave3	192.168.201.5

我们需要把规划好的 IP 地址、主机名添加到 4 台虚拟机的 hosts 文件里，如图 2-39 所示。

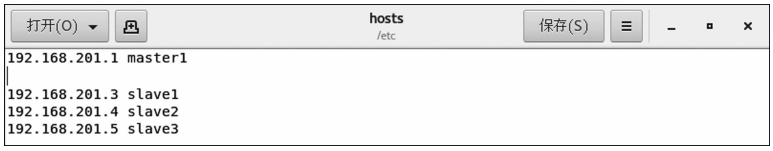

图 2-39　修改 host 文件

2)配置 SSH 无密码链接

远程管理环境中最常用的是 SSH(secure shell)。SSH 是一个可在应用程序中提供的安全通信协议,通过 SSH 可以安全地进行网络数据传输,SSH 采用非对称加密体系,即对所有传输的数据进行加密,保证数据在传输时不被恶意破坏、泄露和篡改。Hadoop 仅在启动和停止时需要主节点通过 SSH 协议启动或停止进程。

在配置 SSH 无密码连接之前,需要先关闭防火墙。执行以下命令。

```
systemctl stop firewalld.service
systemctl disable firewalld.service
```

然后,执行“firewall-cmd --state”命令,查看是否成功关闭了防火墙,如图 2-40 所示。

图 2-40　查看是否成功关闭了防火墙

登录 NameNode(192.168.201.1)节点,切换至 Hadoop 用户,在 Hadoop 的 home 目录下创建“.ssh”目录。

```
$cd
$mkdir .ssh
```

在这台机器上生成密钥对,如图 2-41 所示。

```
$ssh-keygen -t rsa
```

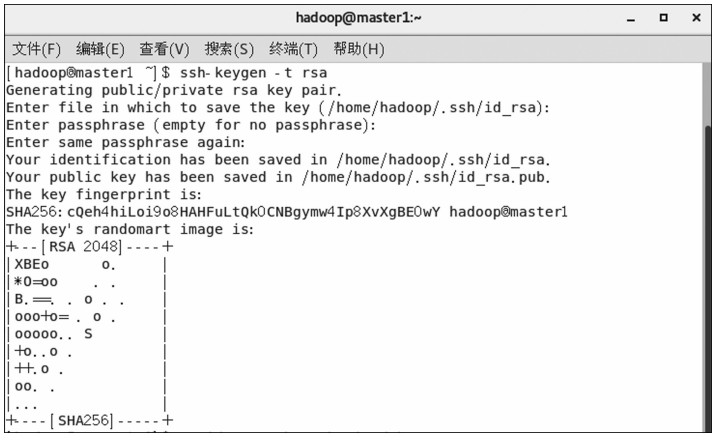

图 2-41　生成密钥对

将生成的 id_rsa.pub 文件复制一份，命名为 authorized_keys，然后分别复制到其他 3 台 slave 机器上，执行以下命令。

```
$cp id_rsa.pub authorized_keys
$scp authorized_keys 192.168.201.3:/home/hadoop/.ssh
$scp authorized_keys 192.168.201.4:/home/hadoop/.ssh
$scp authorized_keys 192.168.201.5:/home/hadoop/.ssh
```

按照屏幕提示完成复制，如图 2-42 所示。

```
[hadoop@master1 ~]$ scp authorized_keys slave1:/home/hadoop/.ssh
hadoop@slave1's password:
authorized_keys                               100%  396   275.0KB/s   00:00
[hadoop@master1 ~]$ scp authorized_keys slave2:/home/hadoop/.ssh
The authenticity of host 'slave2 (192.168.201.4)' can't be established.
ECDSA key fingerprint is SHA256:XLyhq69A2JNzjNlkx3rPFlCXSbOwbZaPWTO9qoAnt88.
ECDSA key fingerprint is MD5:b5:bc:14:81:bb:d0:d0:ac:b2:f0:c5:ee:67:74:eb:6f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'slave2,192.168.201.4' (ECDSA) to the list of known hosts.
hadoop@slave2's password:
authorized_keys                               100%  396   403.0KB/s   00:00
[hadoop@master1 ~]$ scp authorized_keys slave3:/home/hadoop/.ssh
The authenticity of host 'slave3 (192.168.201.5)' can't be established.
ECDSA key fingerprint is SHA256:XLyhq69A2JNzjNlkx3rPFlCXSbOwbZaPWTO9qoAnt88.
ECDSA key fingerprint is MD5:b5:bc:14:81:bb:d0:d0:ac:b2:f0:c5:ee:67:74:eb:6f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'slave3,192.168.201.5' (ECDSA) to the list of known hosts.
hadoop@slave3's password:
authorized_keys                               100%  396    20.7KB/s   00:00
[hadoop@master1 ~]$ \
```

图 2-42　复制 SSH 文件

最后需要从 master1 机器向其他机器发起 SSh 连接，如果不需要输入密码，则表示配置成功了。

```
$ssh slave1
$ssh slave2
$ssh slave3
```

注意：“.ssh”是一个隐藏文件夹，用“ll -a”命令可以看到隐藏的文件夹，“ssh”文件夹权限必须是 700；“.ssh”里面的文件权限最好是 600，如图 2-43 所示。在 SSH 安装过程中，经常由于权限遇到麻烦，因此在操作时要特别注意，SSH 的安装对“.ssh”文件夹和文件权限要求非常严格，高了或低了都连不上。

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
[hadoop@master1 ~]$ cd .ssh
[hadoop@master1 .ssh]$ touch authorized_keys
[hadoop@master1 .ssh]$ chmod 600 authorized_keys
[hadoop@master1 .ssh]$ ll
总用量 12
-rw-------. 1 hadoop hadoop    0 1月  17 22:14 authorized_keys
-rw-rw-rw-. 1 hadoop hadoop 1679 1月  17 20:48 id_rsa
-rw-rw-rw-. 1 hadoop hadoop  396 1月  17 20:48 id_rsa.pub
-rw-rw-rw-. 1 hadoop hadoop  546 1月  16 17:14 known_hosts
[hadoop@master1 .ssh]$
```

图 2-43　修改文件权限

4. 配置 Hadoop

Hadoop 的配置文件都在/etc/hadoop 中，进入该文件夹下，会发现有若干配置文件。Hadoop 集群的安装是在单机安装的基础上修改表 2-2 中的文件。

表 2-2　Hadoop 的配置文件

文件名称	格　式	描　述
hadoop-env. sh	bash 脚本	记录 Hadoop 要用的环境变量
core-site. xml	Hadoop 配置 XML	Hadoop core 的配置项，如 HDFS 和 MapReduce 常用的 I/O 设置等
hdfs-site. xml	hdfs 配置 XML	HDFS 守护进程的配置项，包括 NameNode、blocksize 等
yarn-site. xml	Hadoop 配置 XML	YARN 守护进程的配置项，包括 ResourceManager、NodeManager 等
mapred-site. xml	Hadoop 配置 XML	MapReduce 计算框架的配置项
slaves	纯文本	运行 DataNode 和 NodeManager 的机器列表
hadoop-metrics. properties	properties 文件	控制 metrics 在 Hadoop 上如何发布的属性
log4j. properties	properties 文件	系统日志文件、NameNode 审计日志、DataNode 子进程的任务日志的属性

(1)修改 hadoop-env. sh。在文件 hadoop-env. sh 末尾追加环境变量。

```
export JAVA_HOME=/usr/java/jdk1.8.0_191
export HADOOP_HOME=/usr/hadoop-2.8.5
```

(2)修改 core-site. xml。

```
<property>
    <name>fs.default.name</name>
    <value>hdfs://master1:9000</value>
</property>
```

该项配置设置提供 HDFS 服务的主机号和端口号，也就是说，HDFS 通过 master1 的 9000 端口提供服务，这项配置也指明了 NameNode 所运行的节点(主节点)。

(3)修改 hdfs-site. xml。在<configuration></configuration>之间添加如下属性。

```
<property>
    <name>dfs.replication</name>
    <value>3</value>
</property>
<property>
    <name>dfs.name.dir</name>
    <value>/usr/hadoop-2.8.5/name</value>
</property>
<property>
    <name>dfs.data.dir</name>
    <value>/usr/hadoop-2.8.5/data</value>
</property>
```

dfs. replication 配置项设置 HDFS 中文件副本数为 3。HDFS 会自动对文件做冗余处理，该项配置就是配置文件的冗余数，3 表示有两份冗余。dfs. name. dir 配置项设置 NameNode 数据存放的本地文件系统路径，dfs. data. dir 设置 DataNode 存放数据的本地文件系统路径。

(4)修改 mapred-site. xml，在 mapred-site. xml 中添加：

```
<property>
    <name>mapreduce. job. tracker</name>
     <value>master1:9001</value>
</property>
```

在 core-site. xml 和 mapred-site. xml 文件中分别指定 NameNode 和 job. tracker 的主机名。

(5)修改 slaves 文件，slaves 指定子节点的位置，因为要在 master1 上启动 HDFS、在 master2 上启动 ResourceManager，所以 master1 上的 slaves 文件指定的是 DataNode 的位置，master2 上的 slaves 文件指定的是 NodeManager 的位置。

```
slave1
slave2
slave3
```

任务 3　Hadoop 的启动与验证

2. 3. 1　格式化 HDFS

在第一次启动 Hadoop 之前，必须先将 HDFS 格式化。执行如下命令。

```
hadoop namenode -format
```

按照提示输入“Y”，格式化成功后会出现格式化成功的信息，如图 2-44 所示。

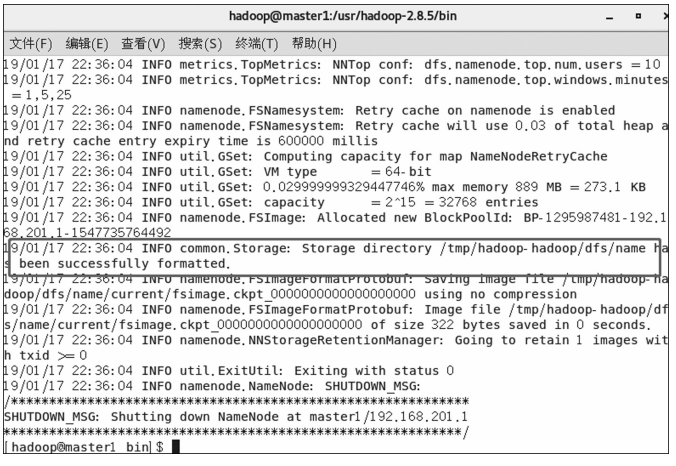

图 2-44　Hadoop 格式化

2.3.2　Hadoop 的守护进程

在 sbin 目录下有很多启动脚本，可以根据自己的需求来启动 Hadoop 的守护进程，Hadoop 的启动和停止说明见表 2-3。

表 2-3　Hadoop 的启动和停止说明

启动和停止脚本	脚本说明
start-all. sh	启动所有 Hadoop 守护进程
stop-all. sh	停止所有 Hadoop 守护进程
start-dfs. sh	启动 Hadoop HDFS 守护进程
stop-dfs. sh	停止 Hadoop HDFS 守护进程
hadoop-daemons. sh start namenode	单独启动 NameNode 守护进程
hadoop-daemons. sh stop namenode	单独停止 NameNode 守护进程
hadoop-daemons. sh start secondarynamenode	单独启动 SecondaryNameNode 守护进程
hadoop-daemons. sh stop secondarynamenode	单独停止 SecondaryNameNode 守护进程
start-mapred. sh	启动 Hadoop MapReduce 守护进程 JobTracker 和 TaskTracker
stop-mapred. sh	停止 Hadoop MapReduce 守护进程 JobTracker 和 TaskTracker
hadoop-daemons. sh start jobtracker	单独启动 JobTracker 守护进程
hadoop-daemons. sh stop jobtracker	单独停止 JobTracker 守护进程
hadoop-daemons. sh start tasktracker	单独启动 TaskTracker 守护进程
hadoop-daemons. sh stop tasktracker	单独停止 TaskTracker 守护进程

如果 Hadoop 集群是第一次启动，可以用 start-all. sh。比较常用的启动方式是依守护进程次序来启动，启动步骤如下。

(1)启动 Hadoop 的 HDFS 模块里的守护进程。

启动 HDFS 守护进程的顺序是：启动 NameNode 守护进程—启动 DataNode 守护进程—启动 SecondaryNameNode 守护进程。

Hadoop 启动成功后，执行 jps 命令可以查看运行的进程，如图 2-45 所示。

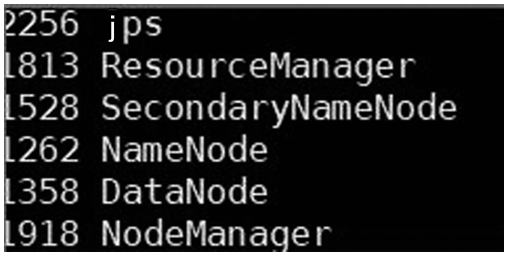

图 2-45　查看运行的进程

(2)启动 MapReduce 模块里的守护进程:启动 JobTracker 守护进程—启动 TaskTracker 守护进程。

如果关闭,可以用 stop-all. sh 命令,或者按上述步骤相反的步骤执行即可。

注意:正常情况下,不使用 start-all. sh 和 stop-all. sh 来启动和停止 Hadoop 集群。因为这样出错了不好找原因。建议一个一个地启动守护进程,建议使用 start-dfs. sh 命令,这样可以缩小找错的范围。

2.3.3 验证集群 HDFS

如果使用 jps 命令查看守护进程运行正常,则可以在 master1 的虚拟机上打开浏览器,访问 http://master1:50070,可以打开图 2-46 所示的界面。

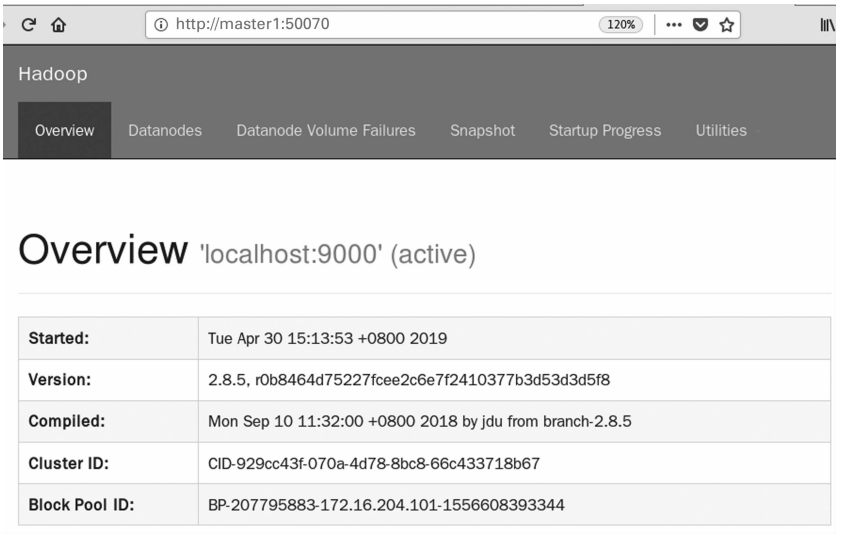

图 2-46 HDFS 界面

总结与回顾

本项目主要完成了搭建、配置 Hadoop 环境的具体过程,包括单机模式和分布模式。

首先,通过 VMware 安装了一个 CentOS 虚拟机,在这个虚拟机上安装了 JDK、Hadoop,并完成了 JDK 和 Hadoop 的配置。然后,根据 Hadoop 集群的拓扑结构,配置了一个虚拟网络环境,并克隆出 3 台虚拟机作为 DataNode 的节点,通过修改 Hadoop 的若干配置文件,最终正常启动 Hadoop 的一系列守护进程。

习 题

1. 练习安装 VMware 和 CentOS。
2. 练习下载并安装 Hadoop 系统。
3. 简要描述配置伪分布式 Hadoop 和分布式 Hadoop 的主要区别。
4. 练习启动和停止 Hadoop。

项目3 HDFS 技术

HDFS 是 Hadoop 分布式文件系统。本项目主要讲述 Hadoop 的文件系统 HDFS，包括 3 个任务，任务 1 主要介绍 HDFS 的特点；任务 2 主要介绍 HDFS 架构；任务 3 主要介绍 Hadoop Shell 命令。

通过本项目的学习，应达到以下目标。

- 能理解 HDFS 体系架构。
- 能掌握 HDFS 命令。
- 能理解 HDFS 流的操作过程。
- 能掌握 HDFS API 编程方法。

任务 1　认识 HDFS

HDFS(Hadoop distributed file system)是 Apache Hadoop 分布式文件系统，其设计初衷是支持高吞吐和超大文件的流式读/写操作。这样的结构能够使成百上千台计算机共同存储一个文件。也就是说，通过 HDFS 访问 TB、PB 级的数据时，就像在一台计算机上访问一个文件一样。

HDFS 的设计思想是当数据的大小超过单台计算机的存储能力时，将其进行分区并存储到若干台单独的计算机上。管理网络中多台计算机存储的文件系统就是分布式文件系统。

3.1.1　HDFS 产生的背景

在大数据时代，数据不仅由互联网产生，还会由智能终端、科学计算、工业设备、智能交通设施等产生。这些海量数据每天都会产生，使用单个操作系统的存储方式显然不能满足大数据的存储需求，因此，需要一种系统来存储这些海量数据，于是分布式文件系统(distributed file system，DFS)就诞生了。

3.1.2 HDFS 简介

HDFS 是 Hadoop 项目的核心子项目，用于大数据领域的数据存储。HDFS 适合通用硬件上的分布式文件系统。它与现有的分布式系统有很多相近的地方，也有很多明显的不同。

3.1.3 HDFS 的特点

作为 Hadoop 的基础，HDFS 非常适合运行在廉价硬件集群上，以数据流的形式访问数据以存储超大文件。其主要优势有以下几点。

(1)适合大数据处理。HDFS 的节点规模非常大，能够达到 10 KB。存储在 HDFS 中的文件大多为 GB、TB 级别，目前，很多互联网企业(如百度、阿里等)集群存储的数据都是 PB 级别的，随着时间的推移，数量级还会成级数增长，很快会达到 EB(1 EB=1 024 PB)甚至 ZB(1 ZB=1 024 EB)。

(2)高容错性。HDFS 在设计之初就认为文件存储在大规模集群中，每个节点发生故障的概率会很高，甚至认为会是一种常态。数据自动保存多个副本，从而当某一个副本丢失以后，它可以自动恢复，提高容错性。当节点发生故障时，HDFS 能够继续运行并且不让用户察觉。

(3)适合批处理。

(4)通过移动计算而不是移动数据，会把数据位置暴露给计算框架。

(5)流式文件访问。一次写入，多次读取。文件一旦写入不能修改，只能追加。它能保证数据的一致性。在数据集生成后，会长时间在这个数据集上进行各种分析，每次分析都会读取大部分数据甚至全部数据，采用流式文件访问，提高了读取数据集的时间延迟。

(6)可构建在廉价机器上。它通过多副本机制，提高了可靠性。它提供了容错和恢复机制。例如，某一个副本丢失，可以通过其他副本来恢复。因此，HDFS 在设计之初就考虑了节点的故障率，所以 HDFS 并不需要在高可靠且昂贵的服务器上运行。

任务 2　了解 HDFS 架构

HDFS 为 Hadoop 分布式计算框架提供了高性能、高可靠、高可扩展的存储服务。HDFS 采用 Master/Slave 的架构来存储数据，这种架构主要由四个部分组成，分别为 client、NameNode、DataNode 和 SecondaryNameNode。这些部分在集群里以节点的形式存在，各个节点运行不同类型的守护进程，各个节点互相配合，一起构成了 HDFS。

如图 3-1 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 进程的节点的块里。

1. client

文件上传 HDFS 时，client 将文件切分成一个个的 block，然后进行存储。client 从 NameNode 获取文件的位置信息，从 DataNode 读取或写入数据。client 提供一些命令来管

理 HDFS，如启动或关闭 HDFS。client 可以通过一些命令来访问 HDFS。

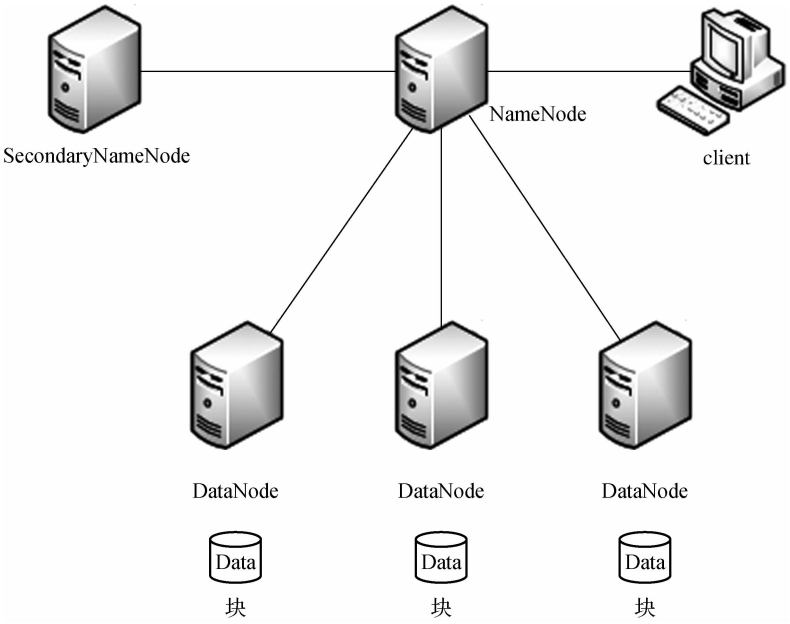

图 3-1　HDFS 架构

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

NameNode 也被称为名字节点，即 master。它是一个主管、管理者，是 HDFS 的大脑，维护整个文件系统的目录树，管理 HDFS 的名称空间，管理数据块(block)映射信息，并在配置副本策略的同时处理客户端读写请求。

3. DataNode

DataNode 就是 slave。NameNode 下达命令，DataNode 执行实际的操作，存储实际的数据块，执行数据块的读/写操作。

每个 DataNode 都是以块为单位进行存储的。每个磁盘都有默认的数据块大小，块是磁盘读写数据的最小单位。HDFS 的块比一般文件系统的块大很多，默认为 64 MB，并可以根据实际文件大小而变化，配置项为 hdfs-site. xml 文件中的 dfs. block. size 项。例如，某个 150 MB 大小的文件，默认配置为 64 MB，该文件在 HDFS 的实际存储情况如图 3-2 所示。

图 3-2　默认配置下文件块的分布

HDFS 存储时，当最后一个块不足一个块的大小时，HDFS 不会占用整个块的空间，所以第三个块的大小是 22 MB 而不是 64 GB。

在 HDFS 中如果块设置得足够大，寻址的开销就会被最小化。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。

为了保证文件在廉价设备上的安全，在 hdfs-site. xml 文件中，还有一项配置为 dfs. replication，该项配置为每个 HDFS 块在 Hadoop 集群中保存的份数，数值越高，冗余性越好，但占用的存储空间也越多。

4. SecondaryNameNode

SecondaryNameNode 并非 NameNode 的热备，而是 NameNode 的辅助节点。当 NameNode 失效时，它并不能马上替换 NameNode 并提供服务。它辅助 NameNode 分担其工作量，定期合并 FsImage 和 fsedits，并推送给 NameNode。在紧急情况下，可辅助恢复 NameNode。

3.2.1 HDFS 读取和写入数据

HDFS 将文件切分成块并存储至各个 DataNode 中，文件数据块在 HDFS 的布局情况由 NameNode 和 hdfs-site. xml 中的配置 dfs. replication 共同决定。dfs. replication 表示该文件在 HDFS 中的副本数，默认为 3，就是有两份冗余，如图 3-3 所示。

图 3-3　dfs. replication 为 3 的文件块分布

Hadoop 的默认布局是在 HDFS 的客户端节点上放第一个副本，但是由于 HDFS 的客户端往往会运行在集群之外，因此 Hadoop 会随机选择一个存储不太慢或不太忙的节点。第二个副本放在与第一个不同且随机选择的机架的节点上，第三个副本放在与第二个副本相同的机架上。这样的方法不仅提供了很好的冗余性，也实现了很好的负载均衡。

1. 数据读取

HDFS 客户端可以通过多种不同的方式对 HDFS 进行读取，这些操作都遵循同样的流程。HDFS 客户端需要使用 Hadoop 库函数，函数库封装了大部分与 NameNode 和 DataNode 通信相关的细节，同时也考虑分布式文件系统的错误机制。

假设请求读取文件 ss. avi，这个读取流程如图 3-4 所示。

(1)使用 HDFS 提供的客户端开发库，向远程的 NameNode 发起 RPC 请求。

(2) NameNode 会视情况返回文件的部分或全部 block 列表，对于每个 block，NameNode 都会返回有该 block 拷贝的 DataNode 地址。

(3)客户端拿到 block 的位置信息后，调用 FSDataInputStream API 的 read 方法并行地读取 block 信息，图 3-4 中 4 和 5 流程是并发的，block 默认有 3 个副本，所以每一个 block 只

需要从一个副本读取就可以。客户端开发库会选取离客户端最接近的 DataNode 来读取 block。

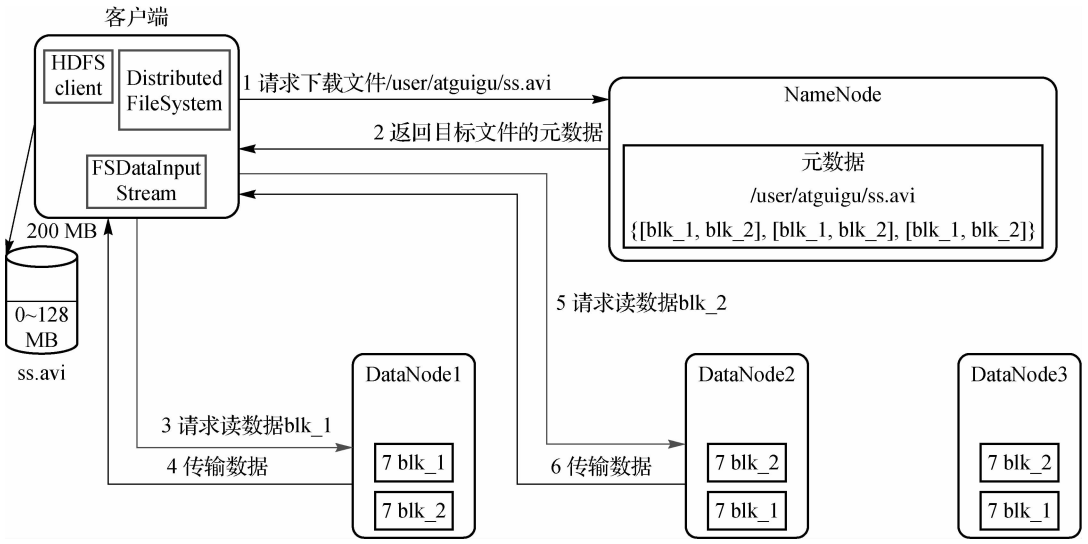

图 3-4　数据读取流程

(4)读取完当前 block 的数据后,关闭与当前 DataNode 的连接,并为读取下一个 block 寻找最佳的 DataNode;返回给客户端。

(5)当读完列表的 block 且文件读取还没有结束时,客户端开发库会继续向 NameNode 获取下一批 block 列表。

(6)读取完一个 block 都会进行 checksum 验证,如果读取 DataNode 时出现错误,客户端会通知 NameNode,然后从下一个拥有该 block 拷贝的 DataNode 继续读取。

2. 数据写入

数据写入流程如图 3-5 所示。

(1)使用 HDFS 提供的客户端开发库,向远程的 NameNode 发起 RPC 请求。

(2)NameNode 会检查要创建的文件是否已经存在,创建者是否有权限进行操作,成功则会为文件创建一个记录,否则会让客户端抛出异常。

(3)当客户端开始写入文件时,开发库会将文件切分成多个 packet(信息包),在内部以“data queue”的形式管理这些 packet,并向 NameNode 申请新的 block,获取用来存储 replicas(复制品)的合适的 DataNode 列表,列表的大小根据在 NameNode 中对 replication 的设置而定。

(4)开始以 pipeline(管道)的形式将 packet 写入所有的 replicas 中。开发库把 packet 以流的方式写入第一个 DataNode,该 DataNode 把该 packet 存储之后,再将其传递给在此 pipeline(管道)中的下一个 DataNode,直到最后一个 DataNode,这种写数据的方式呈流水线的形式。

(5)最后一个 DataNode 成功存储之后会返回一个 ack packet,在 pipeline 里传递至客户端,在客户端的开发库内部维护着 ack queue,成功收到 DataNode 返回的 ack packet 后,会从 ack queue 中移除相应的 packet。

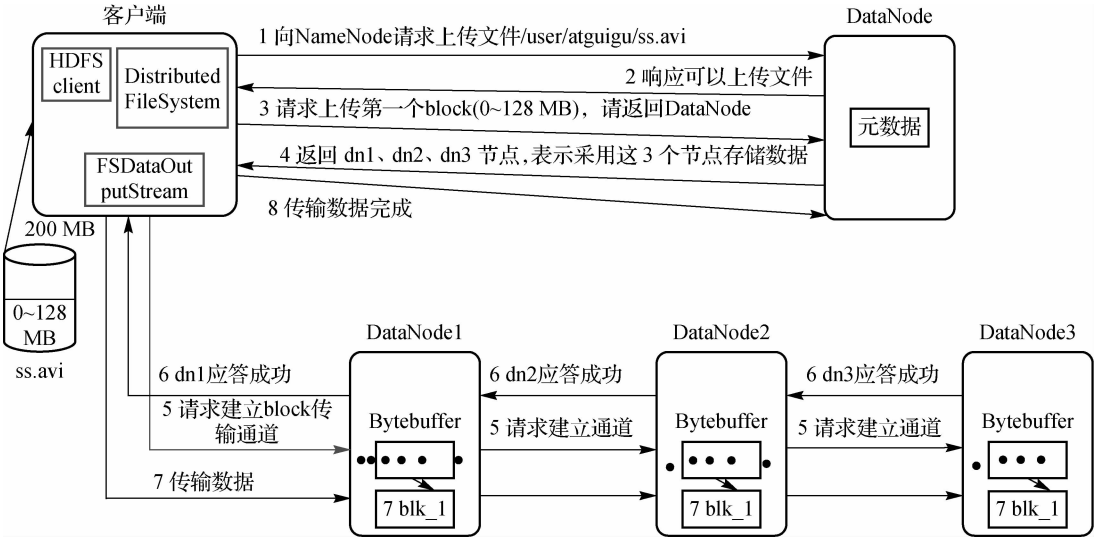

图 3-5　数据写入流程

(6)如果传输过程中有某个 DataNode 出现了故障，那么当前的 pipeline 会被关闭，出现故障的 DataNode 会从当前的 pipeline 中被移除，剩余的 block 会继续剩下的 DataNode，继续以 pipeline 的形式传输，同时 NameNode 会分配一个新的 DataNode，保持 replicas 设定的数量。

3.2.2　元数据节点和数据节点

元数据节点 NameNode 是管理者，一个 Hadoop 集群有一个 NameNode 节点，是一个通常在 HDFS 实例中的单独机器上运行的软件。它负责管理文件系统名字空间和控制外部客户机的访问。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，第一个复制块存储在同一机架的不同节点上，最后一个复制块存储在不同机架的某个节点上。

实际的 I/O 事务并没有经过 NameNode，只有表示 DataNode 和块的文件映射的元数据经过 NameNode。当外部客户机发出请求要求创建文件时，NameNode 会以块标识和该块的第一副本的 DataNode IP 地址作为响应。这个 NameNode 还会通知其他将要接收该块的副本的 DataNode。

NameNode 在一个 FsImage 文件中存储所有关于文件系统名称空间的信息，这个文件和一个包含所有事务的记录文件(这里是 EditsLog)将存储在 NameNode 的本地文件系统上。FsImage 和 EditsLog 文件也需要复制副本，以防文件损坏或 NameNode 系统丢失。NameNode 是 master，负责管理多个 DataNode，所以容易出现单点的问题，这里比较常用的解决方案是使用 NFS 对 NameNode 上的 FsImage 和 EditsLog 进行备份。

数据节点 DataNode 也是一个通常在 HDFS 实例中的单独机器上运行的软件。Hadoop 集群包含一个 NameNode 和大量 DataNode。DataNode 通常以机架的形式组织，机架通过一个交换机将所有系统连续起来。Hadoop 的一个假设是：机架内部节点之间的传输速度快于机架间节点的传输速度。

DataNode 响应来自 HDFS 客户机的读写请求，它们还响应来自 NameNode 的创建、删除和复制块的命令。NameNode 依赖来自每个 DataNode 的定期心跳消息。每条消息都包含一个块报告，NameNode 可以根据这个报告验证块映射和其他文件系统元数据。如果 DataNode 不能发送心跳消息，NameNode 将采取修复措施重新复制在该节点丢失的块。DataNode 的功能如下。

(1)保存 block，每个块对应一个元数据信息文件。这个文件主要描述这个块属于哪个文件、第几个块等信息。

(2)启动 DataNode 线程时会向 NameNode 汇报 block 信息。

(3)通过向 NameNode 发送心跳消息，保持与其联系(3 s 一次)，如果 NameNode 10 min 没有收到 DataNode 的心跳消息，则认为其已经丢失，并将其上的 block 复制到其他 DataNode。

3.2.3　辅助元数据节点

辅助元数据节点 SecondaryNameNode 会周期性地将 EditsLog 文件中对 HDFS 的操作记录合并到一个 FsImage 文件中，然后清空 EditsLog 文件。NameNode 重启就会加载最新的 FsImage 文件，并重新创建一个 EditsLog 文件来记录 HDFS 操作，由于 EditsLog 中记录的是从上次 FsImage 以后到现在的操作列表，因此会比较小。

如果没有 SecondaryNameNode 这个周期性的合并过程，当每次重启 NameNode 时，就会花费很长的时间。而这样周期性的合并就能减少重启的时间，同时也能保证 HDFS 的完整性。SecondaryNameNode 合并 FsImage 和 EditsLog 文件的过程如下。

(1)文件系统客户端(client)进行写操作时，首先把它记录在修改日志(EditsLog)中。

(2)元数据节点在内存中保存了文件系统的元数据信息。在记录修改日志后，元数据节点修改内存中的数据结构。

(3)每次写操作成功之前，修改日志都会同步到文件系统。

(4)FsImage 文件即名字空间映像文件，是内存中的元数据在磁盘上的 CheckPoint，它是一种序列化的格式，并不能在磁盘上直接修改。

(5)同数据的机制相似，当元数据节点失败时，最新 CheckPoint 的元数据信息从 FsImage 加载到内存中，然后逐一重新执行修改日志的操作。

(6)从元数据节点就是用来帮助元数据节点将内存中的元数据信息 CheckPoint 到磁盘上的。

CheckPoint 的过程如下。

(1)从元数据节点通知元数据节点生成新的日志文件，以后的日志都写到新的日志文件中。

(2)从元数据节点用 HTTP Get 从元数据节点获得 FsImage 文件及旧的日志文件。

(3)从元数据节点将 FsImage 文件加载到内存中，并执行日志文件中的操作，然后生成新的 FsImage 文件。

(4)从元数据节点将新的 FsImage 文件 HTTP Post 传回元数据节点。

(5)元数据节点可以将旧的 FsImage 文件及旧的日志文件换为新的 FsImage 文件和新的日志文件(第一步生成的)，然后更新 FsTime 文件，写入此次 CheckPoint 的时间。

(6)这样元数据节点中的 FsImage 文件保存了最新 CheckPoint 的元数据信息，日志文件也重新开始，不会变得很大了。

SecondaryNameNode 会周期性地将 EditsLog 文件进行合并，合并前提条件如下。

(1)EditsLog 文件到达某一阈值时，对其进行合并。

(2)每隔一段时间进行一次合并。

如果 NameNode 损坏或丢失，Hadoop 就无法启动，这时可以人工恢复到 SecondaryNameNode 中最近保存的状态，在集群数据损坏时 SecondaryNameNode 或多或少能够挽回一些损失，因此，要尽量避免将 SecondaryNameNode 和 NameNode 放在同一台机器上。

3.2.4 安全模式

安全模式主要是为了系统启动时检查各个 DataNode 上的数据块的有效性，同时根据策略复制或删除部分数据块。运行期间可以通过命令进入安全模式。

用户可以通过 hadoop dfsadmin -safemode value 来操作安全模式，如图 3-6 所示。

```
[root@localhost sbin]# hadoop dfsadmin -safemode get
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Safe mode is OFF
[root@localhost sbin]# hadoop dfsadmin -safemode enter
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Safe mode is ON
[root@localhost sbin]#
```

图 3-6 查看安全模式状态和进入安全模式

参数 value 包括：

- enter：进入安全模式。
- leave：离开安全模式。
- get：返回安全模式状态。
- wait：等待，直到安全模式结束。

任务 3 Hadoop shell 命令

和 HDFS 系统交互可以使用 HDFS shell 命令，这是最直接、最简单的交互方式，调用文件系统(FS)shell 命令可以使用

```
bin/hadoop fs
```

的形式。所有的 FS shell 命令都是使用 URI 路径作为参数的。

URI 格式是 scheme://authority/path。URI 的 scheme 是 hdfs，表示分布式 HDFS 系统，scheme 是 file，表示本地系统。其中，scheme 和 authority 参数都是可选的，如果未指定，则会使用配置中默认的 scheme。

例如，/parent/child 可以表示成 hdfs://namenode:namenodePort/parent/child，或者更简单的/parent/child(假设配置文件是 namenode:namenodePort)。

3.3.1　命令格式

文件系统 shell 包含各种 shell like 命令，可以直接和 HDFS 文件系统进行交互，就像对其他系统的支持一样，如 Local FS、HFTP FS、S3 FS。shell 命令执行格式如下。

```
hadoop fs {args}
hadoop dfs {args}
hdfs dfs {args}
```

hadoop fs 是使用最广泛的命令，可以操作任何文件系统。

3.3.2　HDFS 命令

用户命令对于集群的用户来说是很有用的，命令也不多。例如，创建文件、创建文件夹、移动文件、删除文件、重命名等，命令的使用方法和格式与 Linux 的命令符相似，用户可以执行如下命令。

```
hdfs dfs -help
```

查看命令列表，如图 3-7 所示。

```
[root@localhost sbin]# hdfs dfs -help
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] [-d] <localsrc> ... <dst>]
        [-copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] [-h] [-v] [-t [<storage type>]] [-u] [-x] <path> ...]
        [-cp [-f] [-p | -p[topax]] [-d] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] [-x] <path> ...]
        [-expunge]
        [-find <path> ... <expression> ...]
        [-get [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-getfacl [-R] <path>]
        [-getfattr [-R] {-n name | -d} [-e en] <path>]
        [-getmerge [-nl] [-skip-empty-file] <src> <localdst>]
        [-help [cmd ...]]
        [-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [<path> ...]]
        [-mkdir [-p] <path> ...]
        [-moveFromLocal <localsrc> ... <dst>]
        [-moveToLocal <src> <localdst>]
        [-mv <src> ... <dst>]
        [-put [-f] [-p] [-l] [-d] <localsrc> ... <dst>]
        [-renameSnapshot <snapshotDir> <oldName> <newName>]
        [-rm [-f] [-r|-R] [-skipTrash] [-safely] <src> ...]
        [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
        [-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>]|[--set <acl_spec> <path>]]
        [-setfattr {-n name [-v value] | -x name} <path>]
        [-setrep [-R] [-w] <rep> <path> ...]
        [-stat [format] <path> ...]
```

图 3-7　部分用户命令列表

下面来看一下几个常见的 HDFS 命令，在 simple 下执行“touch words. txt”命令新建 words. txt 文本文件，并对其进行编译。

1. cat 命令

使用方法：hadoop fs -cat URI [URI …]。

将路径指定文件的内容输出到 stdout。例如，

```
hadoop fs -cat hdfs://host1:port1/file1 hdfs://host2:port2/file2
hadoop fs -cat file:///file3 /user/hadoop/file4
```

返回值：成功返回 0，失败返回－1。

2. chgrp 命令

使用方法：hadoop fs -chgrp [-R] GROUP URI [URI …]。

改变文件所属的组。使用-R 将使改变在目录结构下递归进行。命令的使用者必须是文件的所有者或超级用户。

3. chmod 命令

使用方法：hadoop fs -chmod [-R] <MODE[,MODE]… | OCTALMODE> URI [URI …]。

改变文件的权限。使用-R 将使改变在目录结构下递归进行。命令的使用者必须是文件的所有者或超级用户。

4. chown 命令

使用方法：hadoop fs -chown [-R] [OWNER][:[GROUP]] URI [URI]。

改变文件的拥有者。使用-R 将使改变在目录结构下递归进行。命令的使用者必须是超级用户。

5. copyFromLocal 命令

使用方法：hadoop fs -copyFromLocal <localsrc> URI。

除了限定源路径是一个本地文件外，和 put 命令相似。

6. copyToLocal 命令

使用方法：hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>。

除了限定目标路径是一个本地文件外，和 get 命令相似。

7. cp 命令

使用方法：hadoop fs -cp URI [URI …] <dest>。

将文件从源路径复制到目标路径。这个命令允许有多个源路径，此时目标路径必须是一个目录。例如，

```
hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2
hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2 /user/hadoop/dir
```

返回值：成功返回 0，失败返回－1。

8. du 命令

使用方法：hadoop fs -du URI [URI …]。

显示目录中所有文件的大小，或者当只指定一个文件时，显示此文件的大小。例如，

```
hadoop fs -du /user/hadoop/dir1 /user/hadoop/file1 hdfs://host:port/user/hadoop/dir1
```

返回值：成功返回 0，失败返回－1。

9. dus 命令

使用方法：hadoop fs -dus <args>。

该命令用于显示文件的大小。

10. expunge 命令

使用方法：hadoop fs -expunge。

使用该命令能够清空回收站。

11. get 命令

使用方法：hadoop fs -get [-ignorecrc] [-crc] <src> <localdst>。

这个命令用来复制文件到本地文件系统。可以使用-ignorecrc 选项复制 CRC 校验失败的文件，使用-crc 选项复制文件及 CRC 信息。例如，

```
hadoop fs -get /user/hadoop/file localfile
hadoop fs -get hdfs://host:port/user/hadoop/file localfile
```

返回值：成功返回 0，失败返回－1。

12. getmerge 命令

使用方法：hadoop fs -getmerge <src> <localdst> [addnl]。

这个命令用来接收一个源目录和一个目标文件作为输入，并将源目录中所有的文件连接成本地目标文件。addnl 是可选的，用于指定在每个文件结尾添加一个换行符。

13. ls 命令

使用方法：hadoop fs -ls <args>。

如果是文件，则按照如下格式返回文件信息。

```
文件名 <副本数> 文件大小 修改日期 修改时间 权限 用户 ID 组 ID
```

如果是目录，则返回它直接子文件的一个列表，就像在 UNIX 中一样。目录返回列表的信息如下。

```
目录名 <dir> 修改日期 修改时间 权限 用户 ID 组 ID
```

例如，

```
hadoop fs -ls /user/hadoop/file1 /user/hadoop/file2 hdfs://host: port/user/hadoop/dir1/nonexistentfile
```

返回值：成功返回 0，失败返回－1。

14. lsr 命令

使用方法：hadoop fs -lsr <args>。

这个命令是 ls 命令的递归版本。

15. mkdir 命令

使用方法：hadoop fs -mkdir <paths>。

这个命令接收路径指定的 URI 作为参数，创建这些目录。其行为类似于 UNIX 的

mkdir -p，它会创建路径中的各级父目录。例如，

```
hadoop fs -mkdir /user/hadoop/dir1 /user/hadoop/dir2
hadoop fs -mkdir hdfs://host1:port1/user/hadoop/dir hdfs://host2:port2/user/hadoop/dir
```

返回值：成功返回 0，失败返回－1。

16. movefromLocal 命令

使用方法：dfs -moveFromLocal <src> <dst>。

输出一个“not implemented”信息。

17. mv 命令

使用方法：hadoop fs -mv URI [URI …] <dest>。

将文件从源路径移动到目标路径。这个命令允许有多个源路径，此时目标路径必须是一个目录。不允许在不同的文件系统间移动文件。例如，

```
hadoop fs -mv /user/hadoop/file1 /user/hadoop/file2
hadoop fs -mv hdfs://host:port/file1 hdfs://host:port/file2 hdfs://host:port/file3 hdfs://host:port/dir1
```

返回值：成功返回 0，失败返回－1。

18. put 命令

使用方法：hadoop fs -put <localsrc>…<dst>。

从本地文件系统中复制单个或多个源路径到目标文件系统，也支持从标准输入中读取输入并写入目标文件系统。例如，

```
hadoop fs -put localfile /user/hadoop/hadoopfile
hadoop fs -put localfile1 localfile2 /user/hadoop/hadoopdir
hadoop fs -put localfile hdfs://host:port/hadoop/hadoopfile
hadoop fs -put - hdfs://host:port/hadoop/hadoopfile
```

从标准输入中读取输入。

返回值：成功返回 0，失败返回－1。

19. rm 命令

使用方法：hadoop fs -rm URI [URI …]。

删除指定的文件。只删除非空目录和文件。请参考 rmr 命令了解递归删除。例如，

```
hadoop fs -rm hdfs://host:port/file /user/hadoop/emptydir
```

返回值：成功返回 0，失败返回－1。

20. rmr 命令

使用方法：hadoop fs -rmr URI [URI …]。

例如，

```
hadoop fs -rmr /user/hadoop/dir
hadoop fs -rmr hdfs://host:port/user/hadoop/dir
```

返回值：成功返回 0，失败返回－1。

21. setrep 命令

使用方法：hadoop fs -setrep [-R] <path>。

改变一个文件的副本系数。-R 选项用于递归改变目录下所有文件的副本系数。例如，

```
hadoop fs -setrep -w 3 -R /user/hadoop/dir1
```

返回值：成功返回 0，失败返回－1。

22. stat 命令

使用方法：hadoop fs -stat URI [URI …]。

返回指定路径的统计信息。例如，

```
hadoop fs -stat path
```

返回值：成功返回 0，失败返回－1。

23. tail 命令

使用方法：hadoop fs -tail [-f] URI。

将文件尾部 1 KB 字节的内容输出到 stdout。支持-f 选项，行为和 UNIX 中一致。例如，

```
hadoop fs -tail pathname
```

返回值：成功返回 0，失败返回－1。

24. test 命令

使用方法：hadoop fs -test -[ezd] URI。

选项：

- -e：检查文件是否存在。若存在则返回 0。
- -z：检查文件是否是 0 字节。若是则返回 0。
- -d：若路径是个目录，则返回 1，否则返回 0。

```
hadoop fs -test -e filename
```

25. text 命令

使用方法：hadoop fs -text <src> 。

将源文件输出为文本格式。允许的格式是 zip 和 TextRecordInputStream。

26. touchz 命令

使用方法：hadoop fs -touchz URI [URI …]。

创建一个 0 字节的空文件。例如，

```
hadoop fs -touchz pathname
```

返回值：成功返回 0，失败返回－1。

3.3.3　HDFS 管理员命令

管理员命令对于集群管理员是非常有用的，命令比较多，如平衡管理、缓存管理、DFS 管

理、DataNode 管理等，用户可以查看 DFS 管理员命令的帮助，执行命令：

```
hdfs dfsadmin -help
```

查看命令列表，执行命令的部分结果如图 3-8 所示。

```
[root@localhost sbin]# hdfs dfsadmin -help
hdfs dfsadmin performs DFS administrative commands.
Note: Administrative commands can only be run with superuser permission.
The full syntax is:

hdfs dfsadmin
        [-report [-live] [-dead] [-decommissioning]]
        [-safemode <enter | leave | get | wait>]
        [-saveNamespace]
        [-rollEdits]
        [-restoreFailedStorage true|false|check]
        [-refreshNodes]
        [-setQuota <quota> <dirname>...<dirname>]
        [-clrQuota <dirname>...<dirname>]
        [-setSpaceQuota <quota> [-storageType <storagetype>] <dirname>...<dirname>]
        [-clrSpaceQuota [-storageType <storagetype>] <dirname>...<dirname>]
        [-finalizeUpgrade]
        [-rollingUpgrade [<query|prepare|finalize>]]
        [-refreshServiceAcl]
        [-refreshUserToGroupsMappings]
        [-refreshSuperUserGroupsConfiguration]
        [-refreshCallQueue]
        [-refresh <host:ipc_port> <key> [arg1..argn]
        [-reconfig <datanode|...> <host:ipc_port> <start|status|properties>]
        [-printTopology]
        [-refreshNamenodes datanode_host:ipc_port]
        [-deleteBlockPool datanode_host:ipc_port blockpoolId [force]]
        [-setBalancerBandwidth <bandwidth in bytes per second>]
        [-getBalancerBandwidth <datanode_host:ipc_port>]
        [-fetchImage <local directory>]
        [-allowSnapshot <snapshotDir>]
        [-disallowSnapshot <snapshotDir>]
        [-shutdownDatanode <datanode_host:ipc_port> [upgrade]]
        [-evictWriters <datanode_host:ipc_port>]
        [-getDatanodeInfo <datanode_host:ipc_port>]
```

图 3-8　部分管理员命令

其中：

• -report：报告 HDFS 的基本统计信息。有些信息也可以在 NameNode Web 服务首页看到。

• -safemode：通常并不需要，但是管理员可以手动让 NameNode 进入或离开安全模式。

• -finalizeUpgrade：删除上一次升级时制作的集群备份。

总结与回顾

本项目主要介绍了 HDFS 技术，讲解 HDFS 在大数据领域的重要作用及 HDFS 分布式存储的架构特点，最后介绍了 Hadoop shell 命令。

我们在认识 HDFS 的基础上，进一步了解了 HDFS 的架构，在 HDFS 下如何读取和写入一个数据，以及元数据节点和数据节点在 HDFS 中的作用。

习　题

1. HDFS 的特点有哪些？

2. HDFS 是如何读取数据的？

3. 元数据节点、数据节点和辅助元数据节点的功能分别是什么？

4. HDFS shell 的命令格式是什么样的？