

单位代码： 10293 密 级：           

南京邮电大学

专 业 学 位 硕 士 论 文



论文题目： Hadoop 平台下的海量小文件处理研究

学 号 1212043110

姓 名 马 越

导 师 黄 刚

专业学位类别 工程硕士

类 型 全 日 制

专业（领域） 软件工程

论文提交日期 2015 年 2 月

# **The Research on Massive Small Files Processing under the Hadoop**

Thesis Submitted to Nanjing University of Posts and  
Telecommunications for the Degree of  
Master of Engineering



By

Yue Ma

Supervisor: Prof. Gang Huang

February 2015

## 南京邮电大学学位论文原创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南京邮电大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

本人学位论文及涉及相关资料若有不实，愿意承担一切相关的法律责任。

研究生签名：\_\_\_\_\_ 日期：\_\_\_\_\_

## 南京邮电大学学位论文使用授权声明

本人授权南京邮电大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档；允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索；可以采用影印、缩印或扫描等复制手段保存、汇编本学位论文。本文电子文档的内容和纸质论文的内容相一致。论文的公布（包括刊登）授权南京邮电大学研究生院办理。

涉密学位论文在解密后适用本授权书。

研究生签名：\_\_\_\_\_ 导师签名：\_\_\_\_\_ 日期：\_\_\_\_\_

# 摘要

近年来随着大数据的被大量应用到社会生活中，导致无论是在数据量上还是在数据类型上都呈现了一种爆炸性的增长，从而使得在对数据分析时需要有更高的存储能力，因此，作为在云计算概念上延伸和发展出来的大数据处理技术，业已成为炙手可热的研究方向。而 Hadoop 平台作为当前应用最广的云计算大数据处理平台，其在面对海量小文件处理时，则由于先天的设计因素，在直接应用时存在着处理效率低下的问题。

针对 Hadoop 平台下海量小文件的处理效率低下的问题，本文结合实际工程部署情况，主要从下面三个方面进行分析优化：首先，在集群的底层架构方面，通过分析比较传统虚拟化技术与容器技术在处理性能方面的差异，使用更好的 Hadoop 平台的底层架构方案，即不使用传统的虚拟机架构方案，而采用容器作为底层的部署环境来搭建 Hadoop 平台，从而提高 Hadoop 平台下各应用的数据处理能力；在文件存储方面，基于 Hadoop 平台下现有的 HDFS，采用分类汇总的基本思想，结合实际情况来寻找合适的方案来提高 Hadoop 平台对于海量小文件的存储效率；在数据处理层面上，结合 Hadoop 生态系统中的新的计算框架，即使用基于内存的计算模型来提高数据处理海量小文件的效率。本文通过构建各项测试平台，结合实验数据说明了在 Hadoop 平台下海量小文件的优化处理方案。

**关键词：** Hadoop，小文件，虚拟化，容器，云计算

# Abstract

In recent years, with the big data are applied to social life, resulting in either in the amount of data or in the data type has been an explosive growth, so that in a storage capacity, higher need for data analysis therefore, as data in the cloud computing concept extension and development out of the processing technology, has become a hot research direction. While the Hadoop platform as the most widely used cloud data processing platform, which in the face of massive small file processing, due to factors inherent in the design, the direct application of existing when handling the problem of low efficiency.

Aiming at the low efficiency of the management of massive small file under the platform of Hadoop, this paper combined with the actual project deployment optimization, mainly carries on the analysis from three aspects below: firstly, the underlying architecture of cluster, through the analysis and comparison of the differences between traditional virtualization technology and container technology in the processing performance, using the better underlying architecture of the scheme Hadoop platform, virtual machine architecture scheme that does not use traditional, while using the container as the environment to deploy the bottom to build the Hadoop platform, so as to improve the application of Hadoop platform data processing ability; in the file storage, the existing HDFS based on Hadoop platform, using the basic idea of the summary and classification, combining with the actual situation to find the right method to improve the Hadoop platform for the storage efficiency of large amount of small files; in the data processing level, combined with the new computing framework in the Hadoop eco-system, that is used to improve the efficiency of processing mass data of small files based on the memory computational model. In this paper, through the construction of the test platform, combined with experimental data to demonstrate that the optimum processing scheme in Hadoop platform for massive small file.

**Key words:** Hadoop, Small File, Virtualization, Container, Cloud computing

# 目录

专用术语注释表 .....	V
第一章 绪论 .....	1
1.1 课题背景 .....	1
1.2 课题来源及论文的主要内容 .....	3
1.3 论文的组织架构 .....	4
第二章 相关背景知识介绍 .....	5
2.1 Hadoop 集群概述 .....	5
2.1.1 Hadoop 数据存储 .....	5
2.1.2 使用 HDFS 文件 .....	7
2.1.3 HDFS 联盟 .....	8
2.2 内存计算框架 .....	8
2.2.1 Spark 的架构 .....	9
2.2.2 Spark 的编程模型 .....	9
2.3 容器技术 .....	10
2.3.1 Docker 技术介绍 .....	11
2.3.2 Docker 的优势 .....	12
2.4 物理设备上的优化 .....	14
2.4.1 固态存储 .....	14
2.4.2 将 SSD 应用于 Ceph 集群 .....	14
2.5 本章小结 .....	16
第三章 基于容器构建 Hadoop 平台 .....	17
3.1 传统的 Hadoop 平台架构 .....	17
3.2 虚拟机与容器的比较 .....	18
3.3 Docker 的读写性能分析 .....	19
3.3.1 读文件时性能比较 .....	20
3.3.2 写文件时性能比较 .....	21
3.3.3 文件 I/O 性能比较 .....	21
3.4 在 Docker 上构建 Hadoop .....	22
3.4.1 构建 Hadoop 镜像 .....	23
3.4.2 搭建 Docker 的集群环境 .....	27
3.5 本章小结 .....	29
第四章 软件层面优化处理 .....	30
4.1 方案构架说明 .....	30
4.1.1 序列化文件方法 .....	30
4.1.2 归档文件方法 .....	31
4.1.3 方案分析 .....	32
4.2 优化方案设计 .....	33
4.2.1 小文件存储优化 .....	34
4.2.2 小文件读取优化 .....	35
4.2.3 方案逻辑结构 .....	37
4.3 本章小结 .....	40
第五章 方案性能评估 .....	41
5.1 方案架构 .....	41
5.1.1 底层环境构建 .....	41

5.1.2 软件搭建 .....	42
5.2 实验分析 .....	43
5.2.1 小文件存储性能的比较 .....	43
5.2.2 小文件处理性能的比较 .....	44
5.3 本章小结 .....	45
第六章 总结与展望 .....	46
6.1 总结 .....	46
6.2 展望 .....	47
参考文献 .....	48
附录 1 攻读硕士学位期间撰写的论文 .....	50
附录 2 攻读硕士学位期间参加的科研项目 .....	51
致谢 .....	52

## 专用术语注释表

### 符号说明:

NameNode	Hadoop 名称节点服务器
DataNode	Hadoop 数据节点服务器
DevOps	开发与运维
VFS	虚拟文件系统
OSD	对象存储设备即 Object Store Device 的缩写, 是 Ceph 软件里的一个存储单元

### 缩略词说明:

HDFS	Hadoop Distributed File System	Hadoop 分布式文件系统
HAR	Hadoop archive	Hadoop 文件归档
LXC	Linux Container	Linux 容器技术
RDD	Resilient Distributed Datasets	弹性分布式数据集



# 第一章 绪论

本章讨论了 Hadoop 平台下海量小文件处理的课题研究背景及意义，说明了课题来源及主要内容，之后给出了论文的组织架构。

## 1.1 课题背景

从 2010 年开始，大数据技术的在世界范围内得到了快速发展，根据关注信息技术发展的研究机构的预测，在 2015 年到 2020 年间，无论是数据量还是数据类型都将会大幅增长，与之相对应的，PB 级的数据量将成为大数据时代的基本要求。如此大的数据的增长，势必使得企业提升对数据分析和存储能力的需求。而数据量的增加，势必会导致数据中心的扩展变革，因为现在的很多大型数据中心都是从二十世纪 90 年代开始建设的，从那时开始各大公司的服务器主要都是机架式的，取代了之前的大型机，这种模式的数据中心机架一直持续到现在。而随后各大公司为了集约化成本，开始把数据中心托管在异地机房中，从而出现了各类云服务提供商，进而促使出现了专门的数据中心提供公司，这类公司主要以亚马逊和谷歌为代表，主要提供数据的处理、存储等服务，也就是各类云服务。这几年，由于各类云服务提供商的兴起，以及各类云应用的出现，特别是数据量的爆炸式的增长，越来越多的数据处理移植到云服务上，例如根据思科公司的预计，相比 2011 年全球范围内有 61% 的流量使用传统数据中心处理，到 2016 年这个数据则会降低到 36%，大部分数据将会在云平台上进行处理。所以，日后的数据处理将会是基于云计算的环境的。正是由于这些原因，促使各大大型的 IT 基础设施和服务提供商不得不不断提高自身的技术水平以应对瞬息万变的市场环境。虽然不是所有企业都适合使用云服务来处理业务，例如工业控制上的业务，由于工业生产中的强酸碱的环境因素<sup>[1]</sup>，传感器之间不能使用网络连接，所以无法使用云服务处理，但由于工业信息化的发展需求，工业生产中使用的 ERP 系统则可以使用云服务。所以，从科技历史发展的大趋势来看，未来会有更多的数据处理应用汇迁移到云服务上来，也因此云服务数据处理能力会受到各大厂商的关注。

与传统数据库技术相比，云服务处理数据的一大优势是——简单快捷，主要原因是云服务数据处理的平台 Hadoop 在数据分析上具有很好的可编程性，其生态系统规模庞大，许多不懂技术的业务人员可以通过一些开源工具简单快捷的访问和分析数据。Hadoop 技术从 2006 年发展至今，凭借着其特有的高性价比的处理能力，在各大中小企业中得以快速发展，且又因为其开源的特性，使得其生态系统得以不断的发展，各大厂商不断对性能进行优化，

使得 Hadoop 的使用成本越来越低，从而也使得更多的企业投入到 Hadoop 的阵营里，所以可以预计的是以 Hadoop 为主的云计算这一产业的市场价值在接下来的几年的内还会保持较高的增长势头。

未来的几年，Hadoop 技术将会成为企业运营发展所需要的重要依托。特别是在对 Hadoop 的实时分析的能力进一步提高后，Hadoop 将会成为企业数据处理的一大利器。海量数据与即时服务要求企业能过从海量数据中及时有效地提取出有用信息，高效的处理海量数据，并利用数据进行各种分析，为企业业务决策提供支持<sup>[2]</sup>。

作为与云计算同样是信息技术产业发展热点的物联网，也在一方面促进了云计算的发展。我们知道物联网的重要组成部分是传感器，而物联网需要有大量的传感器，所以会产生大量的传感器数据，但是传统的数据库技术在面对物联网下的大量传感器数据时就会显得力不从心。而 Hadoop 却恰好适合处理大量数据的情况，它能够承担大量的存储和分析工作。例如，在对微博等社交平台进行数据挖掘时，由于数据量过于庞大，单机处理和挖掘速度不可接受，有大量重复性的工作，所以，一个基于 Hadoop 的，包含高效率的爬虫和数据处理和挖掘算法的微博数据挖掘系统（WDMS）可以解决上述困难和挑战<sup>[3]</sup>。然而，Hadoop 在设计之初，就是为了处理大文件而设计的，对于处理类似于视频文件这类的大文件，使用 Hadoop 同样可以进行处理，因为视频类文件会被作为文本文件或二进制文件进行处理，这也是 Hadoop 处理各种类型文件的思路，这样能够使开发人员更关注于视频处理算法，而不必在 Hadoop MapReduce 复杂的数据处理细节上浪费精力<sup>[4]</sup>。但是，对于处理海量小文件，由于 Hadoop 平台本身的原因，在处理效率上面显得性能比较低下。例如，随着我国航空航天事业的发展，遥感数据正以指数级增长，传统的遥感数据处理效率已经很难满足人们的需求<sup>[5]</sup>，这些海量的遥感数据，往往会以小文件的形式进行保存，Hadoop 在对于此类数据的处理时，往往会显得性能低下。目前，解决基于 Hadoop 平台的存储系统中小文件存储效率问题的主要思想是合并，即寻找一种合适的方案，将小文件合并或组合为大文件，目前采用的方法无外乎以下两种，一种是利用 Hadoop 自带的归档技术等方案实现小文件合并，另一种则是针对具体的应用进行具体分析，从而进一步提出的文件组合归并的方法。

从上面的分析可以看出，在当前磁盘存储的价格已经相当廉价的情况下，对于现在的计算机来说，想要提高的更多的是读写速度，特别是在大数据时代，读写速度已经在很大程度上制约了云计算的性能，所以，如何提高文件的读写速度是提高云计算效率的一个重要方面。特别是在当前云存储普及的情况下，如何提高云存储服务的响应速度，是企业吸引客户使用其产品的一个重要的决定性因素。而要提高云存储的速度，一个很重要的方面就是提高在海量小文件情况下的存储效率，因为从各方面的情况来看，在云计算的存储数据中，影响读写

速度的一个重要原因就是因为在存在大量的小文件，从而影响了整体的处理速度。

对于在 Hadoop 平台下进行海量小文件处理，因为 Hadoop 平台的设计者，其设计初衷是解决大文件问题的，所以其在解决小文件时效率很不理想，而又没有统一的、一劳永逸的解决方案，因此，就需要结合具体所涉及的工程进行具体分析。这也将是本论文研究的主要方面。

## 1.2 课题来源及论文的主要内容

本研究生毕业设计课题为“Hadoop 平台下的海量小文件处理研究”，课题来源于黄刚教授所主持的：“南京邮电大学——江苏电信虚拟化技术实验室”。

本文主要的研究内容有以下几个方面：

本文先是分析了现有的 Hadoop 平台对于处理海量小文件的局限所在。由于 Hadoop 平台的主要部分是由底层的 HDFS 和上层的 MapReduce 组成，而作为以处理大数据为目标的 Hadoop 平台，在设计之初，是通过流式数据的访问模式来存储文件的，所以，在处理像大文件这种比较集中的数据时具有比较好的处理速度，而在处理小文件时，则因为 Hadoop 平台在存放文件的物理地址时，使用的是 NameNode 节点的内存，在海量小文件的情况下，会造成内存极大的消耗。对于任何一个文件、目录或块，在 HDFS 中都会被视作为一个对象，存储在 NameNode 的内存中，且每个对象所占用的内存空间为 150 bytes。所以，假设我们使用常见的 4G 内存大小的主机作为我们 Hadoop 平台的 NameNode，那么其 NameNode 可用来存储文件地址信息的大小小于 4G，那么可以存储的文件数大概也就只有一千多万个，总的文件数据规模甚至达不到 TB 级，这就达不到大数据所要求的数据量。在这种情况下唯一的解决方案就是去增加 NameNode 的内存，但很难做到跨数量级的增长。而且，由于存在大量的小文件，数据在读写时会产生大量的 I/O 请求，这会极大的影响处理速度。因此，在云计算越来越普及的今天，提高 Hadoop 平台处理海量小文件的性能，是具有现实意义的。

本文其次是针对传统的基于虚拟机的 Hadoop 平台的部署方式，提出了使用容器虚拟化的方式来部署 Hadoop 平台。本文通过比较容器与虚拟机的性能，验证了在处理文件 I/O 时，容器比虚拟机具有更好的性能，从而可以把 Hadoop 平台部署在容器上。之后通过实际分析与验证，给出了两套构建基于容器的分布式 Hadoop 平台的方案，并在最后通过实验验证了该方案比传统的 Hadoop 构建方案具有更好的性能表现。

接着，通过分析现有的 Hadoop 技术，结合我们实际遇到的一些情况，通过使用分类海量文件、合并部分小文件的方案，在软件层面上优化 Hadoop 平台对于海量小文件的处理。

最后我们通过把我们经过一些列措施优化后的平台环境与一般环境的处理平台相比较，说明了我们构建的这一套 Hadoop 平台相关优化方案能够提高处理小文件的能力。

### 1.3 论文的组织架构

本文共分为六章，主要内容与章节组织结构如下所示：

**第一章：绪论。**本章主要介绍本文的课题背景，课题来源及主要内容，以及简单列出了论文章节的安排。

**第二章：相关背景知识介绍。**本章先简单介绍了 Hadoop 平台，其次介绍了有关在 Hadoop 平台上优化处理小文件所需的相关技术，着重介绍了我们实验所需要的容器技术。

**第三章：基于容器构建 Hadoop 平台。**本章通过分析了容器技术相比于虚拟机的优势，并介绍了通过 Docker 构建 Hadoop 平台的方案来替代传统的基于虚拟机的 Hadoop 平台构建方案，从而在底层架构上就为 Hadoop 平台处理海量小文件进行了优化。

**第四章：软件层面优化处理的方案构建说明。**本章首先通过分析了现有的 Hadoop 平台对于处理小文件时的优缺点，从而引出优化策略，介绍了本文所使用的优化方案。

**第五章：方案性能评估。**本章首先对我们所用的优化方案做了总结，并给出了一个完整的架构方案，并通过实验数据，验证了我们所使用的方案的可行性。

**第六章：总结与展望。**对本课题的相关内容进行了总结和展望，并分析了其中的不足，指出了其今后可能的研究方向与发展目标。

## 第二章 相关背景知识介绍

本章首先简单介绍了 Hadoop 平台及其相关技术，其次介绍了有关在 Hadoop 平台上优化处理小文件所需的相关技术。

### 2.1 Hadoop 集群概述

在过去的这几年来里，数据的存储、管理和处理发生了巨大的变化<sup>[6]</sup>。存储的数据比以前更多，数据来源更加多样，数据格式也更加丰富。数据的存储、处理存在困难，并不是一个新问题，但由于当前数据量的爆炸性的增长，传统的单机数据处理方式已不能满足需要，因此出现了基于云计算的数据处理平台——Hadoop。Hadoop 是 Apache 基金会下的一个开源分布式计算平台<sup>[7]</sup>，Apache Hadoop 的一个重要生态系统是 Hadoop 分布式文件系统 HDFS。传统的大型存储区域网络 SAN 和网络附加存储 NAS 虽然能够解决 TB 级的块设备的集中式访问的问题，但对于处理成百的 TB 级的数据处理，则需要一个高性价比的处理系统。而 Hadoop 平台就是这样一个系统，这个系统中能够使每台机器都拥有自己的 I/O 子系统、磁盘、RAM、网络接口、CPU，且支持部分 POSIX 功能。

将 Hadoop 平台之所以能够运用于海量数据处理上，主要是因为其有下面四个优势：

1) 方便：Hadoop 能够在许多提供云服务的主机上（例如 AWS 等提供的云主机）或一般的主机上进行部署，所以对于商业应用来说能够方便的部署。

2) 弹性：可以根据实际情况，动态的调整 Hadoop 平台集群的节点数目，从而在数据量大时方便的增加节点，数据量小时减少节点。

3) 健壮：该特性具有一定的相对性，只有在 Hadoop 平台的非 NameNode 节点出现单点故障时，可以保证仍然能够正常运行，并能够在故障解决时进行数据修复。

4) 简单：Hadoop 平台支持多语言编写分布式处理代码，所以用户能够根据自己熟悉的快速编写代码，从而快速上手。

由于 Hadoop 是由许多生态系统所组成，而本论文所关注的是小文件处理的问题，所以，下面本论文关于 Hadoop 的讨论重点将主要在 HDFS 与 MapReduce 相关的技术上。

#### 2.1.1 Hadoop 数据存储

HDFS 作为 Hadoop 平台的分布式文件系统，其设计思想是基于 Google 的文件系统，当

然，考虑到为了在处理大数据时有扩展性和高性能上获得提高，做了一些取舍，主要体现在下面几个方面<sup>[8]</sup>：

1) HDFS 针对高速流式读取性能做了优化，代价就是降低了随机查找的性能，所以 HDFS 适合顺序读取的方式。

2) HDFS 仅支持有限的文件操作，即不支持更新，所有的更新都是数据的再次写入，适合一次写入，多次读取。

3) HDFS 不提供本地数据缓存机制。

HDFS 的系统架构采用的是主从架构<sup>[9]</sup>，架构中的各节点及对应的功能如表 2.1 所示：

表 2.1 HDFS 主要节点及功能<sup>[10]</sup>

节点名称	主要功能
NameNode (名称节点服务器)	NameNode 是整个分布式文件系统的主服务器，是整个系统的核心，NameNode 作为 HDFS 中文件目录和文件分配的管理者。
DataNode (数据节点服务器)	DataNode 负责存储数据，一个数据块在多个 DataNode 中有备份；而一个 DataNode 对于一个数据块最多只包含一个备份。
Client (客户端)	Client 即是 HDFS 的客户端，它主要是负责连接 HDFS 系统并执行文件操作，通常我们直接使用 Hadoop 自带的命令行操作。

图 2.1 是 HDFS 的架构图，从图中我们可以看出，在 HDFS 中，一个给定的名字节点 NameNode 管理着一些文件系统名称空间操作，如打开、关闭以及重命名文件和目录<sup>[11]</sup>。名字节点 NameNode 通过将数据块映射到数据节点 DataNode 的方法来管理数据，并且负责处理来自 HDFS 客户端的读写请求。DataNode 数据节点则是根据名字节点 NameNode 的要求来负责创建、删除和复制数据块。HDFS 在处理单个文件时，单个文件会被拆分成固定大小的块，并保存在 Hadoop 集群上，这是 Hadoop 平台存储数据的一大特性。数据节点 DataNode 在其本地文件系统上以单独文件的形式保存各个 HDFS 数据块，并且，为了提高吞吐量，数据节点 DataNode 不会将所有文件创建在相同的文件夹中。对于小于块大小的文件，在 HDFS 中只会占用所需大小的磁盘空间，不在额外占用资源。

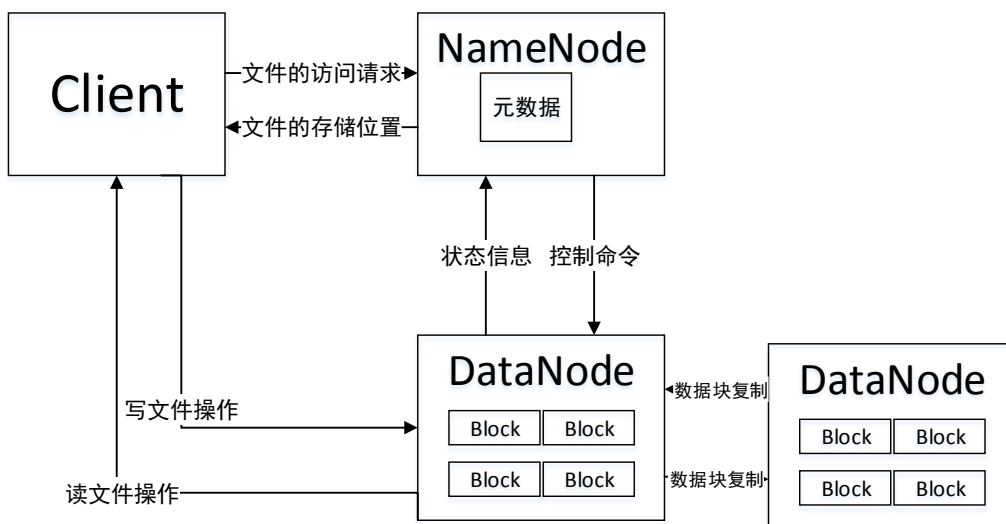


图 2.1 HDFS 架构

### 2.1.2 使用 HDFS 文件

Hadoop 提供了访问 HDFS 的多种方法，但无论使用何种方法，其原理都是一样的。在 HDFS 写入文件时，其步骤如下：

- 1) 客户端 (Client) 向名称节点 (NameNode) 发出写文件的请求。
- 2) NameNode 根据文件大小和块的信息，在可以写入的前提下，向客户端返回一个数据节点列表；否则告知无法写入。
- 3) 客户端将文件划分为多个 Block，根据 NameNode 返回的数据节点列表以包 (packet) 为单位向 DataNode 请求写数据，当能够写入数据时，即可写入数据，否则报告问题。
- 4) 客户端先把包 (packet) 写到第一个 DataNode，然后 DataNode 把 packet 转发给第二个 DataNode，这样一直转发到最后一个 DataNode。
- 5) 只有当所有 DataNode 都响应写入成功，写入文件的流程结束。

在读取文件时，其过程如下：

- 1) 客户端 (Client) 向名称节点 (NameNode) 发出读文件的请求。
- 2) 系统远程调用询问名称节点 NameNode，在查询得到结果后，则能得到文件的 Block (包括副本) 的位置信息；若查找不到相关信息则告知。
- 3) 客户端根据 NameNode 返回的信息向相应的 DataNode 请求读数据，当数据可以读取时，即可直接从 DataNode 读取数据，否则则报告出现的问题。
- 4) 读取数据完成后，系统关闭输入流，断开与这个 DataNode 的连接。读取文件过程结束。

无论是读还是写文件，访问 HDFS 都要依赖一个 FileSystem 对象实例，FileSystem 是

Hadoop 中所有的文件系统的抽象父类，它定义了文件系统所具有的基本特征和基本操作。对于 HDFS，则是由 DFSClient 来实现具体操作。DDFSClient 使用 ClientProtocol 协议通过 RPC 机制和 NameNode 进行通信并获得文件的元数据信息，然后连接到 DataNode 并通过 DFSOutputStream 和 DFSInputStream 来进行数据块的真正读写操作。

### 2.1.3 HDFS 联盟

通过前面的分析我们知道，在处理大量小文件时不能有效利用 HDFS 的一个重要原因是无论文件块大小如何，只要其小于最小文件块的大小，其元数据在 NameNode 中占据的内存完全相同，而 NameNode 又只有一个，所以 NameNode 的内存大小将会限制了所能存储的文件数量。为了克服单个 NameNode 内存的限制并能够水平地扩展 NameNode 的服务，所以在 Hadoop 0.23 中开始引入了 HDFS 联盟，其主要优势如下：

- 1) 名称空间可扩展性提高：添加更多的 NameNode 到 Hadoop 集群中能够很好的扩展名称空间，从而能够在 Hadoop 集群中存储更多数量的文件。
- 2) 性能提高：通过添加更多的 NameNode 可以使得 Hadoop 集群能够提高文件系统读写时的吞吐量。
- 3) 隔离性提高：通过添加更多的 NameNode 可以使得实现将不同类别的数据放置在不同名称空间，提高安全性。

虽然 HDFS 联盟解决了 HDFS 的扩展性问题，使得可管理的文件数量得到很大的提升，但对于本文来说，我们关注的重点将是优化海量小文件的处理，所以在后面的实验架构中，我们并没有使用 HDFS 联盟的方式。

## 2.2 内存计算框架

Hadoop 生态系统最核心的计算框架是 MapReduce，但由于 MapReduce 计算框架在计算过程中较多的使用了磁盘读写操作，主要是因为映射阶段所产生的中间结果不会保存在内存中，而是写回了磁盘，只有混合操作时需要从该集群中的各个运算节点处进行数据传输，所以可以看出，Hadoop 的大量时间花费，是在处理网络磁盘 IO，从而影响了计算处理，所以其在性能上收到一定的限制。为了解决这些限制，最大限度地使用计算资源，美国加州大学伯克利分校的 AMPLab 实验室开发了 Spark，高效的使用内存来提高运算处理性能。



## 2.2.1 Spark 的架构

Spark 起源于 Hadoop 开源社区，可以使用基于 HDFS 来构建系统，其系统生态圈如图 2.2 所示：



图 2.2 Spark 组件生态圈

而 Spark 的运行环境不仅和 Hadoop 一样需要 JDK 环境，还需要有 Scala 环境的支持。Scala 是一门混合了函数式和面向对象的语言<sup>[12]</sup>。Scala 可以看成是一种基于 Java 的高度集成的语言。Scala 的语法十分简洁易学，而 Java 则是一种样板代码，所以 Scala 是一种能让人眼前一亮的交互式开发语言<sup>[13]</sup>。可见 Scala 的一大好处就是支持交互式运行，所以在使用 Spark 时可以很方便的使用交互方式对 Spark 进行测试。

## 2.2.2 Spark 的编程模型

Spark 的主要思想是通过一种新的作业和数据容错方式，从而减少磁盘以及网络的 IO 开销<sup>[14]</sup>。为了极大的减少磁盘及网络中产生的 IO 开销，Spark 计算框架主要是利用了基于内存的计算抽象数据结构 RDD<sup>[15]</sup> (Resilient Distributed Datasets)，RDD 能够被缓存到内存中，供其它计算使用而不必像 MapReduce 那样每次都从 HDFS 上重新加载数据，所以 Spark 在某些应用上的实验结果比 MapReduce 高一百多倍<sup>[16]</sup>。Spark 提供了两种针对 RDD 的操作：

- 1) 转换：根据已有的 RDD 数据集创建一个新的 RDD 数据集。
- 2) 动作：在 RDD 数据集上运行计算后，返回一个值或者将结果写入外部存储系统。

需要说明的是，对一个 RDD 进行操作，所产生的结果是一个新的 RDD，且 RDD 里面保存的并不是真实的数据，而是元数据的记录信息，记录了该 RDD 是通过哪些 RDD 进行何种操作后得到的。

根据 Spark 的内核机制，在使用 Spark 计算框架时，在计算发生的时刻，Spark 会绘制一

张记录计算路径的有向无环图 DAG。当该有向无环图完成后，Spark 内核就会把所需要的计算划分到对应的任务集，也就是分配到对应的 Stage 中。计算节点真正处理的就是这些任务集。为了优化计算，Spark 的一大特点就是在划分任务集时会尽量把可在分布式计算中进行流水线计算的部分提取出来，从而提高处理效果，并且默认状态下 Spark 运算的中间结果会保存在内存中，从而避免与 HDFS 的读写操作。图 2.3 说明了 Spark 的计算过程。

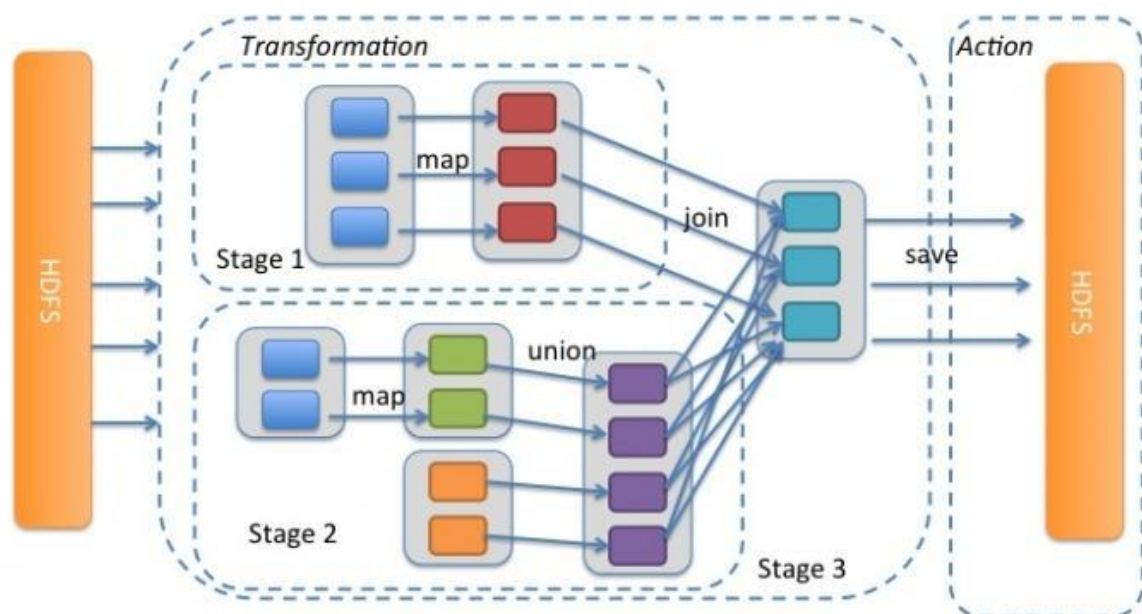


图 2.3 Spark 的运行过程

从图 2.3 中我们可以看出从 HDFS 中读入数据将会生成多个不同的 RDD，然后会进入到各自的转换操作中，操作结束后才将计算汇总结果保存到 HDFS 中。

正是因为 Spark 减少了对磁盘的操作，所以在进行大量文件处理时，其能极大的减少读写次数，从而优化海量文件处理，也因此，我们在我们的 Hadoop 平台构建中也引入了该框架。

## 2.3 容器技术

本文所讨论的容器技术是指 Linux 容器（Linux Container, LXC）技术。虚拟化是持续推动云计算的数据中心进行变革的一种重要的 IT 技术，目前它已显示出了众多好处，例如能够提供快速部署，能够方便的提供测试环境，以及在企业应用中最重要系统快速备份还原。虽然使用虚拟化技术作为数据中心的备份技术还处于一个探索阶段，但作为一种新技术，还是被寄予了很大的希望的，而在众多的虚拟化技术中，容器技术又被认为是一个最具有发展前景的虚拟化技术。

容器的特点是能够有效地将由单个操作系统管理的资源划分到孤立的组中，即可以认为是把运行的程序与操作系统进行了解耦和，这样，相互孤立的组在使用有冲突的资源时能够做到互不影响<sup>[17]</sup>。容器的思想很简单，就是把一个进程（包括其子进程）独立打包在一个“集装箱”内，而不影响系统内的其它进程。这样做最大的好处就是，容器内的进程所运行的环境是独立于底层物理操作系统的，当容器启动的时候，也仅是通过进程间调度，而不需要引导整个系统，这样能够提高效率。

### 2.3.1 Docker 技术介绍

容器技术在 Linux 内核的 2.6.29 版本就产生了，目前已经产生了很多的实现方案，但之前的技术并没有引起业界的太大关注，而随着 2013 年 Docker, Inc 公司发布的一款基于 LXC 技术的开源的软件——Docker 的出现，使得容器技术开始真正进入到普通企业的应用中来，使得容器技术变得具有商业价值。现在 Docker 技术已经被 Google、IBM、RedHat、Amazon 和微软等业界知名公司所支持，可以认为容器现在和虚拟机一起在改变应用程序运行、设计、开发和部署的模式。

Docker 虽然利用的是现有的容器技术，但其意义在于其构建了一种基于容器的可互操作的应用创建方式，这使得 Docker 技术变得如此受欢迎，而且 Docker 还主要解决了下面这些问题：

(1) 部署速度：在传统的软件发布过程中，开发者需要考虑许多因素，其中很重要的一点就是运行环境的问题，因为现实情况下很多大型软件在运行过程中会有许多依赖条件，这一点在 Linux 环境下尤为明显，而且这些东西都是难于管理，软件开发人员不得不面对这一问题。而 Docker 则简化了这些工作，例如，我们若构建一个基于 Stuct 和 impress.js 的演示文稿类的应用，可以直接把其所依赖的诸如 Web 应用、后台应用等都打包成一个镜像，然后再快速部署出去。

(2) 软件管理的自动化：Docker 的出现给软件运维带来了一个新的解决方案，在此之前，无论是使用虚拟云主机还是使用真实物理主机，应用的管理与配置依然是主要依靠人工完成，而 Docker 因为采用的是一种可交互式的方式，且可以通过编排对其需要运行的容器进行配置，所以能够更好的实现软件管理的自动化。

(3) 虚拟化方式的优化：云计算的一大优势是成本的降低，其主要是通过使用集约化的硬件设备来控制硬件成本的投入，而通过软件来实现对用户资源需求的按需分配，而起主要作用的技术就是虚拟化技术。然而传统的虚拟化技术，无论是基于硬件虚拟化的 KVM 还是

通用的 Xen 技术，都存在一定的资源浪费，因为很多情况下用户需要并不是一个操作系统而仅是需要一个运行环境，虚拟主机虽然能够起到很好的资源隔离作用，但其过于浪费资源，所以在很多情况下，轻量级的容器则能够起到优化的效果。

(4) 便捷性：容器技术在 Linux 2.6 版本的内核中就已经被引入了，一直都在不温不火的发展完善中，但由于容器在初期的发展中并没有考虑到云计算的需求，所以容器没有一个便携性的操作方法，虽然出现了很多解决方案，但都难于管理。而 Docker 的创新之处就在于 Docker 把容器的操作变得更为便捷<sup>[18]</sup>。

从上面的描述可以看出，Docker 具有很多传统虚拟化技术所不具有的优势，但其也存在着一个软肋，即 Docker 不能商品化的数据中心<sup>[19]</sup>，因为从软件工程的模块化开发的思想上来说，除了数据中心外，我们可以通过使用 Docker 把软件应用所需要的例如运行环境、配置进行整合或分拆在不同的封装好的容器中，进而把这些容器进行商品化，提供给需要的用户。

### 2.3.2 Docker 的优势

Docker 作为一种容器技术的新的解决方案，无论是对于开发人员还是管理人员来说，都具有很大的意义。特别是对于开发人员来说，以往的应用往往需要考虑软件的发布问题，而现在可以直接把应用封装在 Docker 中，所有的开发更具有模块化的方式，更接近于工业生产的形式，这无疑是很好的推动了信息技术产业的发展。另外，Docker 能够使得应用更加方便的进行跨云服务的迁移，因为只要通过封装应用到 Docker 中，就能够使得在所有支持 Docker 的云服务上运行自己的应用。

软件工程从初期的独立主义到现在，已经发展成一个高度工程化的产业链形式，为了更好的交付软件和服务，人们认识到传统的把开发和运营分隔开来的做法是不再适合当前的软件服务开发需求的，所以，就需要把开发与运营作为一个整体来对待，这也就是所谓的 DevOps 思想，业界也随之有了对 DevOps 系统的需求，即希望有一套平台或软件能够来整合开发和运营交付以及之后的管理工作。所以，从 DevOps 的思想上来看，云平台都是为了解决开发人员的在开发测试过程中快速搭建环境以及后期运营支持过程中恢复升级监控系统所服务的。近年来，随着云计算产业的发展，DevOps 的概念也在业界变得流行起来，第一代的 DevOps 系统是基于物理主机部署的，主要靠人工协调来进行业务的自动部署，第二代的 DevOps 系统则是基于 IaaS 进行部署的，借助云计算的资源监控特性，能够实现很好的智能感知等，能够快捷的通过迁移方便的解决硬件单点故障问题，自动做到负载均衡，这是目前很多云计算公司都在极力达到的目标，现在很多公司还不能很好的实现第二代的 DevOps 系统。然而，

随着容器技术的发展，我认为 DevOps 系统会朝着一个新的方向发展，即基于容器的部署，容器能够实现应用的跨云平台架构迁移，即使得应用不会被限定在固有的 IaaS 中，例如，我们搭建了一套集群，是混合了 VMware 和 Xen 这两种虚拟化技术的云平台，两套系统的虚拟机若想实时互相迁移，是办不到的，因为底层的系统存储格式是不一样的，而容器则不存在这种问题，只需要通过启用对应的容器镜像，就可以启动一个新的服务，这样做就能够使得运维人员无需关心底层的 IaaS 架构了。

另外，随着将 DevOps 实践引入应用程序生命周期管理，应用程序性能管理（APM）方案的出现则逐渐成为企业实现 DevOps 投资回报的重要前提<sup>[20]</sup>。对于应用程序的管理，在如今云应用程序普及的情况下，如何整合云应用则是 IT 团队所面临的严峻挑战，而容器技术则为其提供了一个比较好的解决方案。所以，从商业效益上来说，个人认为容器技术可以说是云计算和 DevOps 发展的下一个重要方向。目前已有超过 14000 应用建立在 Docker 上，eBay 也正在测试建立在 Docker 上的数据中心软件<sup>[21]</sup>。

我们知道，Docker 对于底层操作系统并没有太多的限制，从而使得 Docker 同样可以在虚拟机中的系统中，这一特性使得其能够将其相关的应用很好的部署在现有的虚拟化框架中，例如企业所使用的各类云服务。而且，从理论上来说，甚至可以在容器里安装虚拟机程序，虽然目前来看这样做似乎没有意义，但根据一些谷歌公司开发人员的说法，谷歌的云平台就是用这种方式来使用容器的，因为谷歌公司早在十年前，就开始在其生产环境中使用容器技术了，考虑到 Hadoop 也是基于谷歌公司的一些理论而发展出来的这一现实，所以我们可以相信，容器与虚拟机在未来将会共存发展，当然，也有可能将容器管理和虚拟化技术进行融合从而提供一种新的虚拟化管理方法<sup>[22]</sup>。同样，因为容器的平台特性，使得其能够满足一定的 PaaS 功能，各个封装好的容器就如同可以采取按需付费的 IaaS 产品上的功能模块<sup>[23]</sup>，所以容器很可能使得 PaaS 与 IaaS 变得更为融合，甚至使得这两者融为一体。

当前 Docker 的应用方式主要为以下几种模式：

- 通过提供经过认证的软件包，从而保证实现与现有的各类运行模型协作完成任务；
- 通过微服务 POD 的形式来提供一个按需的运行环境给用户；
- 在现有的 IaaS 之上使用，作为与物理机环境相同的平台进行提供。

总体来说，具体如何使用 Docker 是需要根据实际需求来确定的，无论是基于云基础设施还是使用物理主机来构建 Docker 环境，都是需要综合考虑的，在出于运维便捷性考虑的企业中考虑建立一套小型自动化开发与测试环境，使用物理主机来直接构建 Docker 环境则是更为适合的<sup>[24]</sup>。

虽然 Docker 还处在发展阶段，性能上还有待提高，但基于上述所说的 Docker 优势，所

以我们认为在 Docker 上架设 Hadoop 平台是一种新的平台架构方案。

## 2.4 物理设备上的优化

我们知道，对于提高数据处理的方法，一般可以从两方面进行考虑，一个是软件方面，一个是硬件方面。对于硬件方面的优化，由于各个部署 Hadoop 平台的组织在资金上面的投入程度不同，所以在硬件层面上进行优化并不完全可行，但作为一种优化方法，本文会做一些简单介绍。

### 2.4.1 固态存储

固态存储技术虽然兴起不久，但由于其防震抗摔快速低耗的特点，使得其在诸如智能手机、平板电脑和超极本中得到广泛应用。固态硬盘相比于传统的磁盘，具有明显的优势：在处理随机读写数据时是传统磁盘的五倍，读写延迟分别缩短了十倍和七倍，而耗电量则减少了一半。正是由于固态硬盘拥有如此大的性能优势，所以数据中心中也开始引入固态硬盘，市面上也出现了许多企业级固态硬盘，但由于固态硬盘阵列的价格因素，至今仍没有企业大批量部署。虽然固态硬盘在过去几年来价格持续降低，但相比于传统磁盘，其需要的资金投入过于巨大。另外，固态硬盘还有一个性能上的缺陷，即固态硬盘的“写入耐久性”，在固态硬盘达到写入周期的最大值后，其内部闪存块就可能会出现故障，从而影响数据的保存，虽然已有一些解决方案，但仍限制了固态硬盘的大批量使用。

目前，固态硬盘的一些企业级应用都是采用将固态硬盘与普通磁盘混合组成的方式，形成一个混合存储阵列。这样即能够在大幅提高性能的情况下，又兼顾价格，不至于投入过多的资金。

正是基于以上几点，在优化 Hadoop 处理海量小文件时，如在基础硬件架构上使用固态硬盘，将能极大的提高数据读写速度，从而提高处理性能。当然，考虑到资金投入的因素，可以使用混合存储阵列的方式。

### 2.4.2 将 SSD 应用于 Ceph 集群

上面我们已经分析说明了固态硬盘引入到 Hadoop 架构中将会极大的提高处理性能，且在实际应用中，固态硬盘往往会以混合存储阵列的方式存在，要充分使用固态硬盘的性能，在软件架构上需要进行一定的设置。目前比较流行的趋势是使用 Ceph。

Ceph 是当前 SAN 服务的最典型代表，我们知道随着云服务的快速发展，数据中心的建设成为一个重点，而其中最比较重要的部分则是存储问题。因为随着技术的发展，新存储设备与旧有的设备在性能上有差异，所以为了充分利用现有设备，就需要有专门的存储管理软件，Ceph 就是这样一款分布式存储管理软件。

由于 Ceph 可以很好的实现分布式存储的统一管理，并能以块、对象、文件服务的方式对外提供服务，所以目前被广泛应用，例如云计算提供商 DreamHost、欧洲核子研究中心 CERN 等企业，已使用 Ceph 部署了 PB 级的生产环境。

针对 Hadoop 平台处理海量小文件的需求情况，可以把处理小文件的 Hadoop 平台部署在基于 SSD 的分区上。其实现方法就是通过编辑 Ceph 的 CRUSH MAP，标记出固态硬盘的 OSD 以及磁盘的 OSD，通过分类规则，创建不同的资源池，让指定数据在对应的池中操作。这样一来，对于处理海量小文件的 Hadoop 平台，就可以在一个混合 SSD 和 SATA 磁盘的集群中，单独使用 SSD，从而提高处理速度。

当然，这样做对于硬件的成本还是很高，等于就是要求所有处理小文件的 Hadoop 平台都得使用 SSD 磁盘，这对于海量数据来说，投入的成本将会很大，但因为使用 Ceph 这一分布式统一存储项目，可以使用 Ceph 的 Cache Tiering 技术来解决该问题。正如大家所知的那样，使用 Hadoop 平台处理数据时，都是一次读入，元数据不再进行修改的，Hadoop 本身也是不支持对于元数据进行修改的，在 Hadoop 平台中，数据的修改意味着重新写入新的数据，所以，这意味着对于数据写入时，无论数据是多零碎的小文件，瓶颈是在初始写入时，因此，可以用固态硬盘来做 Ceph 的缓存功能，来提高小文件的处理性能。

因为 Hadoop 存在冷热数据的情况，所以根据 Ceph 的特性，我们可以使用 Ceph 的两种处理模式：

#### 1) Writeback 模式

在这种模式下，数据写入时，是直接写入到 Cache 层，写完立即返回，代理负责把数据及时传输到冷数据的存储池中；而在数据读取时，同样先在 Cache 中尝试读取，读取不到时在从冷数据池中获取。所以，在此种模式下，适合处理数据可变的情况。

#### 2) 只读模式

在这种模式下，所有的写操作是直接对后端的冷数据池进行写，而在获取数据时，Ceph 负责把数据先提取到 Cache 层，在提供给用户程序，这种模式比较适合处理不可变数据。

在应对 Hadoop 处理小文件的情况时，我们的思想是把小文件进行合并后再进行处理，然后再进行数据处理，所以使用 Writeback 模式更符合需求。

以上就是通过硬件架构改造对 Hadoop 处理小文件进行优化，所以在处理海量小文件时，

我们可以针对具体的读写需求选择上述的方案，这样就可以在很大程度上的提高 Hadoop 平台下海量小文件的读写速度，从而提高处理速度。然而该方案还是需要企业进行额外的硬件投入的。

## 2.5 本章小结

本章对 Hadoop 平台的相关内容做了一个简单介绍，并介绍了我们在为构建一个处理海量小文件的 Hadoop 平台而所需要的相关技术，在后续章节中，相关内容会在我们的工作中进行进一步说明。



## 第三章 基于容器构建 Hadoop 平台

本章主要通过分析比较容器与虚拟机在读写性能上的差异，从而提出在架构 Hadoop 平台时，以容器技术替代原有的虚拟化来进行底层的平台架构，从而提高 Hadoop 平台的处理性能，并详细说明了架构方案。

### 3.1 传统的 Hadoop 平台架构

对于传统的 Hadoop 平台的部署，一般来说有两种选择，一种是直接部署在物理主机上，一种是部署在虚拟机上。当然，大多数时候是根据情况，混合使用物理主机和虚拟机来部署 Hadoop 平台。但对于商业公司来说，一般都会把 Hadoop 平台部署在 IaaS 提供的虚拟机上，不会直接部署在物理主机上面，这主要是出于对提高单点故障等问题的处理速度的考虑，并且这样部署也有利于运维人员管理维护各个节点，因此，我们可以发现，许多科技企业都是把 Hadoop 平台部署在虚拟机上，所以说传统的 Hadoop 平台架构是基于虚拟机的。然而，由于虚拟机本身需要消耗一定的资源来实现底层的虚拟化，所以运行虚拟机势必会消耗物理主机的资源，从而造成计算资源的浪费，这也是传统的 Hadoop 平台架构的一大缺点。

对于企业级的 Hadoop 平台部署，底层的虚拟化方案组成有很多，当前业界提供的虚拟机方案大致可分为三类：XEN、KVM 以及 VMware。对于企业应用来说，其虚拟化方案组成可能会选择使用多种虚拟机方案的混合组合的方式，也有很多厂商专门提供混合组合虚拟化技术的解决方案。但无论这些虚拟化解决方案是如何组成的，他们都可归类为使用全虚拟化技术或使用部分虚拟化技术，这些虚拟化技术都是要提供一个完整的虚拟硬件环境，从而能够让一个操作系统完整的安装在虚拟机中。现在各大云计算虚拟化厂商的 IaaS 层，无论是 AWS、Google 还是国内的 BAT 公司，基本都是使用这种虚拟主机的方式。这种方式有其自身的优势，即能够充分利用物理机的资源，能够自由的选择所要创建的主机的虚拟硬件，能够构建各种系统环境。但由于每台虚拟机都是完整的安装整个操作系统，虚拟机之间不能共享相同的底层操作系统的功能，并且，在实际使用过程中，我们使用操作系统时往往并不需要一个完整的操作系统支持，只需要其中的一些软件，或其中的运行环境，但出于安全与维护方便的考量，才需要虚拟机，所以，虚拟机提供了一个完整的操作系统，虽然功能强大，但这也造成了一定的资源浪费。正是基于这一原因，近年来则出现了新的轻量级的虚拟化技术——容器。

## 3.2 虚拟机与容器的比较

虚拟主机技术是能够让一台物理主机上运行多个操作系统，在这个环境中，物理主机自身的操作系统被称为 Host OS，而在虚拟机中运行的操作系统则被称为 Guest OS。每个 Guest OS 都有自己的计算、存储和网络组件，这些可以通过硬件虚拟化，或者 Host OS 通过把 Guest OS 的指令进行翻译成物理主机的指令来实现，所以理论上来说 Guest OS 可以是任何一款操作系统，与底层的 Host OS 无关。

容器则是一款轻量级的操作系统，它运行在物理机的系统上，直接使用物理主机的 CPU 指令，而不需要像虚拟机那样要对指令进行翻译。所以容器不需要像虚拟机那样有太多的开销，且也能提供很好的隔离性。虚拟机能够利用 RAM 的过度承诺技术(RAM over commitment) 来提高性能，能够减少内存的开销，但是容器也表现出比虚拟机更低的系统负载，所以同样的应用，在容器中相比在虚拟机中，性能通常会相当或者更好。经过我们实际测试，在 Docker 启动时，不加载额外软件的话，即只使用基本的系统镜像，所消耗的物理主机的内存仅只有几兆，甚至更少。因为 Docker 对于内存消耗主要是在所加载的程序上的，而不像虚拟机一样需要额外提供给一些系统级的服务，所以对于一台服务器来说，Docker 能够比虚拟机提供更多的与操作系统相隔离的容器给用户使用，容器的这一性能大大提高了物理主机的使用效率。

表 3.1 比较了虚拟机与容器在性能、隔离性、安全性、网络和存储上的不同之处。

表 3.1 虚拟机和容器特性的比较<sup>[25]</sup>

参数	虚拟机	容器
客户机系统	每个虚拟机都会有自己的虚拟硬件，都会在自己的内存中加载内核	所有的客户机系统共享系统和内核，内核镜像是加载在物理内存中的
通信	通过网卡设备	使用标准 IPC 机制，例如信号，管道，套接字等
安全性	取决于虚拟机的安全性	使用的是强制的分层访问控制
性能	取决于虚拟机指令转换成物理主机指令的性能	几乎是提供与底层物理主机一样的性能
隔离性	虚拟机之间共享库、文件等，但不与物理机共享	都能互相访问
开机时间	每个虚拟机启动都需要几分钟的时间	可在几秒内完成启动
存储	需要更多的存储，要存放系统	由于操作系统是共享的，所以需要的存储很少

简单来说，容器技术相比虚拟机有下面几个优势：

- 1) 简化部署方案, 简化命令操作
- 2) 能够快速获得可用的环境, 简化流程
- 3) 微服务化, 主要体现在容器可以按照服务提供给用户, 而不需额外开销

同时, 由于容器本身也可以安装在 IaaS 所分配的虚拟机上, 相比现有的 PaaS, 容器技术也更具灵活性, 因为现有的 PaaS, 大部分都是厂商驱动的, 例如 Cloud Foundry(原先由 VMware 开发)、OpenShift(红帽)等, 这些厂商驱动的 PaaS, 无法达到应用程序的跨平台迁移, 或者说实现起来比较麻烦。而容器则不会出现这种问题, 因为容器底层就完全可以看作是一个操作系统, 就如同一个虚拟机。也正因为如此, 所以现在业界很看好把容器构建成一个分布式服务, 各方都在做这方面的开发与研究, 目前比较成熟的就是 Google 的 Kubernetes, 这是一款开源的容器管理系统, 谷歌使用的容器引擎就是基于 Kubernetes 的。当然, 在 2015 年初, 也新出了许多与容器分布式管理相关的软件。

### 3.3 Docker 的读写性能分析

由于 Docker 本身也是一种虚拟化技术, 我们在使用 Docker 作为 Hadoop 集群部署的环境时, 需要保证其在读写性能上不能低于现有的虚拟机技术。因为本论文的目标就是要提高 Hadoop 小处理海量小文件时的性能, 而我们知道, Hadoop 在处理小文件时的瓶颈就是 I/O 读写, 要使 Hadoop 集群能够在存储处理海量小文件时得到优化, 就必须保证使用 Docker 作为底层虚拟化环境时, 能够保证其 I/O 读写速度的提高, 所以在对于 I/O 的读写性能上要求高于传统的虚拟化环境。为了验证容器在 I/O 读写上优于虚拟机, 我们需要把 Docker 与虚拟机在 I/O 读写上的性能进行比较。我们选择 KVM 作为虚拟机类软件的代表, 主要因为 KVM 技术的虚拟机应用面更广, 而 XEN 技术则相对比较老旧, 服务器系统中的龙头企业红帽公司在 2011 年发布 RHEL6 时就开始使用 KVM 来替代 Xen 了, VMware 的虚拟化技术则由于其属于高度商业化的商品, 在许多高科技企业中反而使用的少, 例如谷歌, 阿里, 而这些企业则恰恰是使用云计算最多的企业; 因此, 我们选择 KVM 来作为虚拟机类技术的代表更为恰当。

为了比较 Docker 与 KVM 在文件 I/O 读写时的性能, 我们构建了一个测试环境, 实验环境如下:

硬件环境: 4G 内存, 双核 CPU, 200G 磁盘;

软件环境: sysbench 0.4.12, 文件块大小为 16k;

sysbench 是一款开源软件, 主要是用于多线程性能测试, 我们使用该软件来进行文件 I/O

测试。

因为实验环境为 Linux 系统，而 Linux 在实现文件系统时采用了两层结构：第一层是虚拟文件系统（VFS），它把各种实际文件系统的公共结构抽象出来，统一的以 inode 为中心来建立其组织结构，从而达到为兼容实际文件系统<sup>[26]</sup>。所以，虽然 Docker 在运行过程中会部分使用自己的一套文件系统，不像 KVM 的虚拟机可以保证使用标准的 Linux 文件系统，但由于虚拟文件系统能够自动适配各种实际的文件系统，所以在实验中的影响不大。但同时，因为是随机读写的小文件，而 Linux 在处理小文件时，为了取得更好的处理效果，我们需要对内核进行参数调优，根据红帽的培训资料显示，需要把 `/sys/block/sda/queue/scheduler` 的值设为 `deadline`。该值的设定是我们经过实验验证后确认的，我们把所产生的数据分别在 `deadline`、`anticipator`、`noop` 和 `cfq` 的调度算法下，使用 `time` 命令进行了打包大量小文件测试，之所以使用 `time` 命令是因为这是一个在 UNIX 系统下常见的工具，使用 Unix 工具而非编写代码<sup>[27]</sup>，这样更具代表性。结果显示 `deadline` 是整体耗时最少的，所以为了提高效率，我们把实验环境都同样设定为相同的调度算法。当然，我们在架设处理海量小文件的 Hadoop 平台时，也应该设定该值，以提高 Hadoop 平台处理海量小文件时的速度。

### 3.3.1 读文件时性能比较

首先，通过使用 `sysbench` 软件，我们获得了随机读 150G 的文件的实验数据，我们把得出的结果用 `gnuplot` 绘图后，结果如下图 3.1 所示：

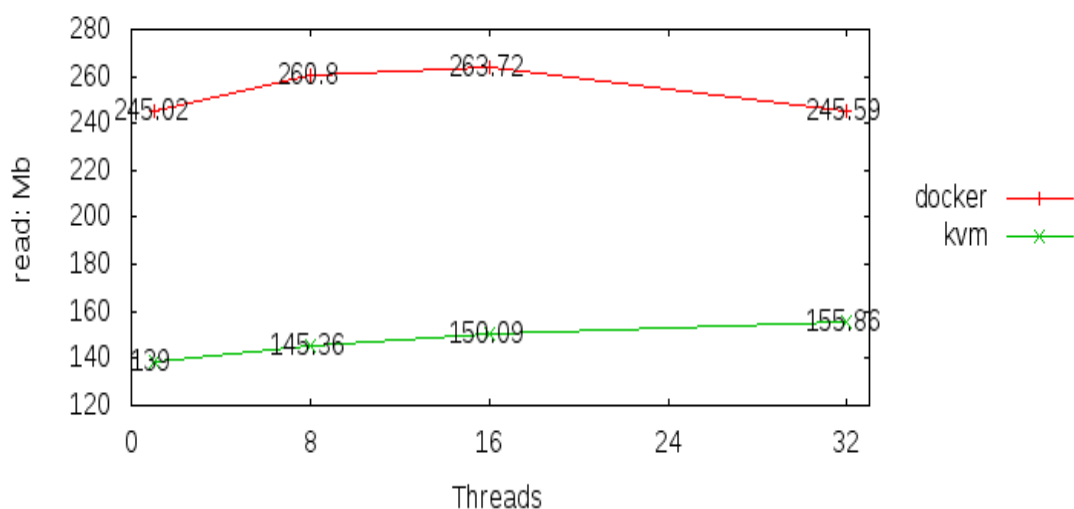


图 3.1 Docker 与 KVM 在文件 I/O 读时的性能比较

虽然实验结果不能作为定量分析的依据，毕竟由于实验硬件环境的不同，可能具体数值上会有所不同，但我们还是能够做一个定性分析，即通过该实验，我们可以看出，Docker 与

KVM 在文件 I/O 读时的性能是有明显差异的，使用 Docker 在文件 I/O 读时具有明显的优势。

### 3.3.2 写文件时性能比较

同样，通过使用 sysbench 软件，我们获得了随机写 150G 的文件的实验数据，我们把得出的结果同样用 gnuplot 绘图后，结果如下图 3.2 所示：

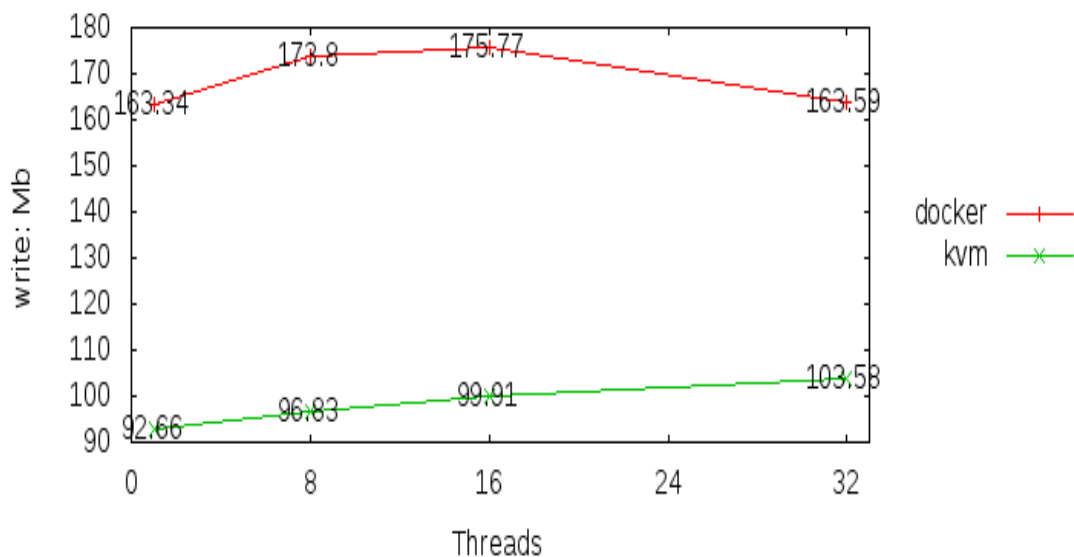


图 3.2 Docker 与 KVM 在文件 I/O 写时的性能比较

同样虽然该实验结果只能作为一个定性分析的依据，但通过该实验，我们也可以看出，Docker 与 KVM 在文件 I/O 写时的性能也是有明显差异的，使用 Docker 在文件 I/O 写时具有明显的优势。

### 3.3.3 文件 I/O 性能比较

最后，通过使用 sysbench 软件，我们同样获得了随机读写 150G 文件时传输速度的实验数据，我们把得出的结果同样用 gnuplot 绘图后，结果如下图 3.3 所示，从中我们可以得知，Docker 容器相比于 KVM 虚拟机，在读写文件传输数据时，性能上依然具有较大优势。

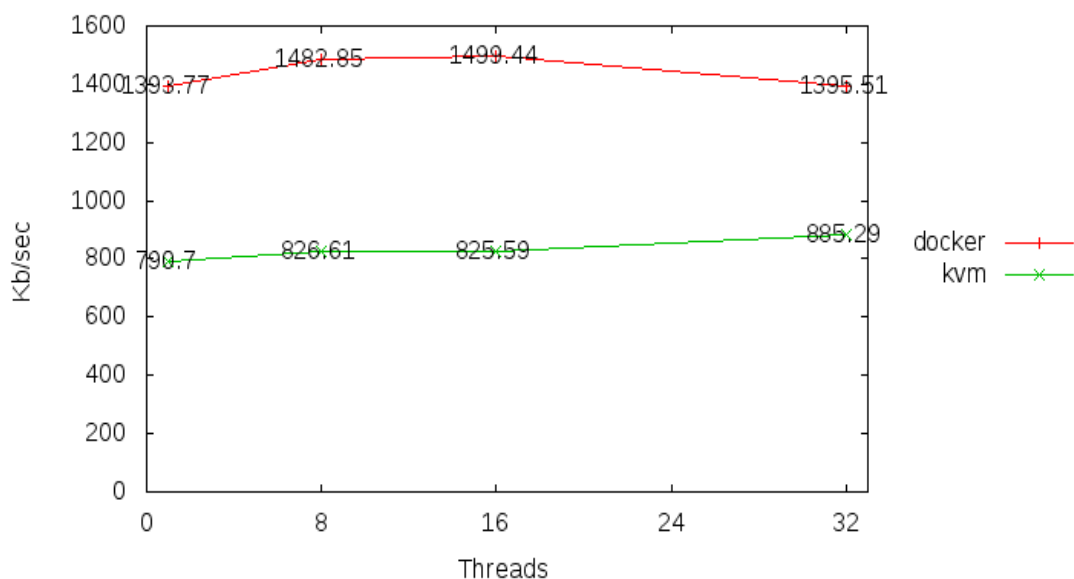


图 3.3 Docker 与 KVM 在读写文件时的传输速率比较

结合上面的所有实验的结果，我们可以看出，使用 Docker 技术，在 I/O 的读写性能上都是能够优于 KVM 技术的，虽然无法确定性能优化程度是否与线程数有关。同时，容器也显示出比虚拟机更低的系统负载，因此同样的应用，在容器中使用会比在虚拟机中使用在性能上获得更好的表现。国外也有一些机构已经开始把 Docker 用于构建大数据处理平台的了，例如罗马尼亚的 Cluj-Napoca Cluj-Napoca 大学就把处理地球观测数据的平台构建在 Docker 下<sup>[28]</sup>，获得较好的性能表现。所以说，Docker 技术在 I/O 的性能上是能够胜任作为大数据处理工具平台。

### 3.4 在 Docker 上构建 Hadoop

从上面的分析可知，我们可以在 Docker 上构建 Hadoop，且在理论上说明了在处理文件读写时也是具有较大优势的。

而因为 Docker 容器的架构不同于传统的虚拟机，更为确切的说它不是虚拟机，它是一种容器实现方案，使用了 Linux 系统中的 Libcontainer，该运行环境是为不同的 Linux 内核提供隔离特性的一个接口。Docker 容器的架构如图 3.4 所示，从图中我们可以看出，Docker 通过使用命名空间和控制组来达到其隔离性的要求，这样的架构设计使得多个容器在共享同一个系统内核时也能完全隔离地运行，到达解耦和的目的。

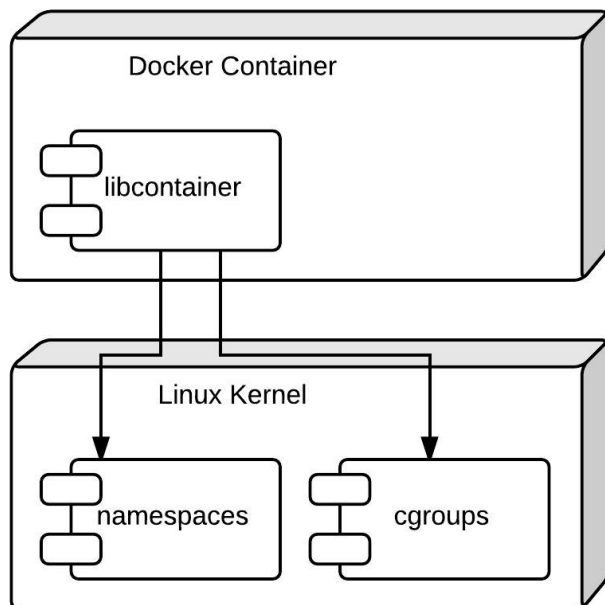


图 3.4 Docker 容器架构

Docker 作为一种特殊的虚拟化软件，要制作特定的软件镜像，Docker 有其自己的安装打包方式，主要有两种方式<sup>[29]</sup>：

- (1) 直接在基础镜像上安装软件，然后使用 Docker 命令将其封装成一个新的镜像；
- (2) 使用 Dockerfile 文件的方法，类似于 GNU 的 make 操作，在拉取初始系统镜像后，让镜像根据 Dockerfile 文件的定义，自己编译生成新的镜像。

第(1)种方法的优点是所有操作与真实操作一台虚拟机一样，无需重新学习新内容，缺点是在部署这些镜像时，可能会由于所处环境的不同而造成需要重新修改部分内容，而且，下载完整的镜像所需要消耗的时间较多；而第(2)种方法则只需要在部署时下载该 Dockerfile 文件，然后让系统自己去拉取对应的数据，这样能够减少所需保存的镜像的大小，但由于镜像完全是按照 Dockerfile 文件的内容来生成的，所以能够减少人为的干预，从而减少出错，但其缺点就是需要花费时间去学习 Dockerfile 文件的机制。

### 3.4.1 构建 Hadoop 镜像

我们使用第(2)种方法来部署 Hadoop，主要完成如下几步工作：

- 1) 下载基础系统镜像；
- 2) 使用 Dockerfile 的内建指令下载安装软件；
- 3) 使用 Dockerfile 内建的指令加载对应的配置文件。

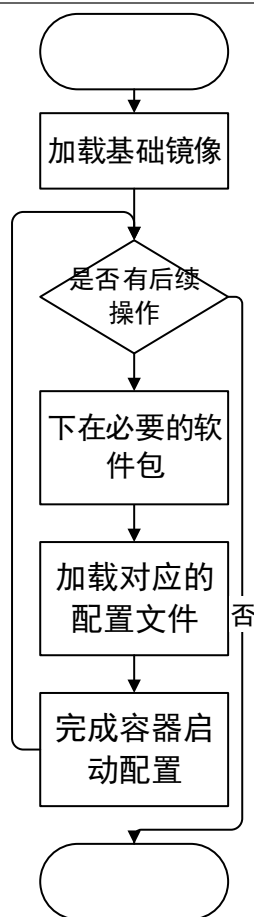


图 3.5 Dockerfile 工作流程

图 3.5 是 Dockerfile 文件执行的流程图,下面就是构建镜像所需的 Dockerfile 文件的内容:

1) 首先确保初始基础镜像完整,且下载 SSH 服务端软件,由于 Hadoop 在分布式环境下需要 SSH 无密钥登陆的支持,所以需要进行设置,下面的配置中我们是让容器在创建时自己建立自己的密钥,从而使在容器启动时能够使用 SSH 访问:

```
FROM ubuntu
MAINTAINER mayue
USER root
RUN apt-get install -y curl tar sudo openssh-server openssh-client rsync
RUN rm -f /etc/ssh/ssh_host_dsa_key /etc/ssh/ssh_host_rsa_key /root/.ssh/id_rsa
RUN ssh-keygen -q -N "" -t dsa -f /etc/ssh/ssh_host_dsa_key
RUN ssh-keygen -q -N "" -t rsa -f /etc/ssh/ssh_host_rsa_key
RUN ssh-keygen -q -N "" -t rsa -f /root/.ssh/id_rsa
RUN cp /root/.ssh/id_rsa.pub /root/.ssh/authorized_keys
```

上面几步通过产生的密钥能够使本机无密钥登陆,若在分布式部署中,则需要把密钥文件作为参数文件加载到容器中,例如后续的对 Hadoop 配置文件进行设置一样。



2) 完成 Java 环境的配置，由于采用纯净的系统作为基础镜像，所以需要手动安装 Java 运行环境，下面的操作是使用的从 Oracle 官网上抓取所需要的 Java 版本的软件包，若在实际使用中，可以把所需要的软件包放在本地，在 Dockerfile 中设置成从本地提取，但我们在初始构建镜像时最好选择直接从官网下载的方式：

```
RUN mkdir -p /usr/java/default && \  
    curl -Ls 'http://download.oracle.com/otn-pub/java/jdk/7u51-b13/jdk-7u51-linux-x64.tar.gz'  
-H 'Cookie: oraclelicense=accept-securebackup-cookie' | \  
    tar --strip-components=1 -xz -C /usr/java/default/  
ENV JAVA_HOME /usr/java/default/  
ENV PATH $PATH:$JAVA_HOME/bin
```

3) 完成 Hadoop 软件的安装，同 Java 软件包一样，需要在容器镜像中安装，下面的操作也是基于从网络上下载的方式进行安装，这样做能够减少构建 Docker 镜像时的文件数量，但会导致在构建镜像时耗费大量的时间从网络下载资源，该方法适合在公共云主机上进行使用，能够极大的简化操作：

```
RUN curl -s http://www.eu.apache.org/dist/hadoop/common/hadoop-2.5.0/hadoop-2.5.0.tar.gz |  
tar -xz -C /usr/local/  
RUN cd /usr/local && ln -s ./hadoop-2.5.0 hadoop  
ENV HADOOP_PREFIX /usr/local/hadoop  
RUN sed -i '/^export JAVA_HOME/ s:.*:export JAVA_HOME=/usr/java/default\nexport  
HADOOP_PREFIX=/usr/local/hadoop\nexport          HADOOP_HOME=/usr/local/hadoop\n:'  
$HADOOP_PREFIX/etc/hadoop/hadoop-env.sh  
RUN sed -i '/^export HADOOP_CONF_DIR/ s:.*:export HADOOP_CONF_DIR=/usr  
/local/hadoop/etc/hadoop/:' $HADOOP_PREFIX/etc/hadoop/hadoop-env.sh
```

4) 把本地的 Hadoop 相关配置文件传到容器中，因为每个容器的环境不同，所以要想使得搭建的 Hadoop 的配置文件可以被使用者方便的修改，可以采取下面的方式，即把配置文件卸载本地，然后通过 Dockerfile 把这些配置文件传到容器中：

```
RUN mkdir $HADOOP_PREFIX/input  
RUN cp $HADOOP_PREFIX/etc/hadoop/*.xml $HADOOP_PREFIX/input  
ADD core-site.xml.template $HADOOP_PREFIX/etc/hadoop/core-site.xml.template  
RUN sed s/HOSTNAME/localhost/ /usr/local/hadoop/etc/hadoop/core-site.xml.template >  
/usr/local/hadoop/etc/hadoop/core-site.xml
```

```

ADD hdfs-site.xml $HADOOP_PREFIX/etc/hadoop/hdfs-site.xml
ADD mapred-site.xml $HADOOP_PREFIX/etc/hadoop/mapred-site.xml
ADD yarn-site.xml $HADOOP_PREFIX/etc/hadoop/yarn-site.xml
RUN $HADOOP_PREFIX/bin/hdfs namenode -format
RUN rm /usr/local/hadoop/lib/native/*

```

5) 为了让容器能够在启动时启动 Hadoop 及相关服务, 需要对服务软件进行配置, 以便在启动时能够自动运行, 这也是 Dockerfile 中比较重要的内容, 下面的操作主要就是对容器的启动项进行配置以及开放对应的权限, 让容器在启动时开启对应的服务:

```

RUN curl -Ls http://dl.bintray.com/sequenceiq/sequenceiq-bin/hadoop-native-64-2.5.0.tar|tar -xz
-C /usr/local/hadoop/lib/native/
ADD ssh_config /root/.ssh/config
RUN chmod 600 /root/.ssh/config
RUN chown root:root /root/.ssh/config
ADD bootstrap.sh /etc/bootstrap.sh
RUN chown root:root /etc/bootstrap.sh
RUN chmod 700 /etc/bootstrap.sh
ENV BOOTSTRAP /etc/bootstrap.sh
# workingaround docker.io build error
RUN ls -la /usr/local/hadoop/etc/hadoop/*-env.sh
RUN chmod +x /usr/local/hadoop/etc/hadoop/*-env.sh
RUN ls -la /usr/local/hadoop/etc/hadoop/*-env.sh
RUN sed -i "/^[^#]*UsePAM/ s/.*#&/" /etc/ssh/sshd_config
RUN echo "UsePAM no" >> /etc/ssh/sshd_config
RUN echo "Port 2122" >> /etc/ssh/sshd_config
RUN service ssh start && $HADOOP_PREFIX/etc/hadoop/hadoop-env.sh &&
$HADOOP_PREFIX/sbin/start-dfs.sh && $HADOOP_PREFIX/bin/hdfs dfs -mkdir -p /user/root
RUN service ssh start && $HADOOP_PREFIX/etc/hadoop/hadoop-env.sh &&
$HADOOP_PREFIX/sbin/start-dfs.sh && $HADOOP_PREFIX/bin/hdfs dfs -put
$HADOOP_PREFIX/etc/hadoop/ input
CMD ["/etc/bootstrap.sh", "-d"]

```

6) 开放容器所运行服务的对应端口, 以便其它终端能够访问及使用相关服务, 该步骤是

十分需要注意的，因为若不执行此操作，会导致在部署时无法指定系统开放的端口：

```
EXPOSE 50020 50090 50070 50010 50075 8031 8032 8033 8040 8042 49707 22 8088 8030
```

通过上述内容构成的 Dockerfile 文件，我们完成了 Hadoop 环境的 Docker 镜像的构建，且经过我们后续的测试，能够正常使用 Hadoop。我们把其与安装在物理主机上的 Hadoop 相比较，通过 time 命令测试两者运行时间，结果显示两者耗时相差不大，部分情况下，Docker 下的耗时才略高一些。虽然因为所部署的 Docker 容器数量不多，但依然可以发现在 Docker 上部署的 Hadoop 平台在数据读写上的表现是出色的。

通过该步，我们就已经获得了一个基于 Docker 的 Hadoop 平台的镜像，此镜像可以用来创建 Hadoop 节点，关于 Hadoop 的相关镜像，我们也可以从 DockerHub 上进行拉取，在大陆地区由于网络环境原因，则可以从 docker.cn 上进行拉取。

### 3.4.2 搭建 Docker 的集群环境

制作好容器镜像后，就可以在部署好的 Docker 上启动对应的容器镜像了。由于 Hadoop 平台是一个分布式平台，因此我们要在基于 Docker 的容器虚拟化环境下部署 Hadoop 平台，需要进一步对 Docker 进行集群化部署。所以下一步的工作就是把各个运行的容器组成一个集群，从而完成 Hadoop 集群的搭建。传统的对于 Docker 的操作都是使用 Docker 客户端来进行操作的，Docker 客户端与 Docker 的关系如图 3.6 所示。从图中我们可以看出，客户端并不需要与运行的容器直接通信，而是通过 Docker 的 Daemon 来进行交互信息，且客户端可以不与 Docker 服务器安装在同一服务器上。从图中我们也可以知道，用户也可以通过使用 REST 接口来操作，因此可以通过脚本编程的方式对 Docker 容器进行操作。

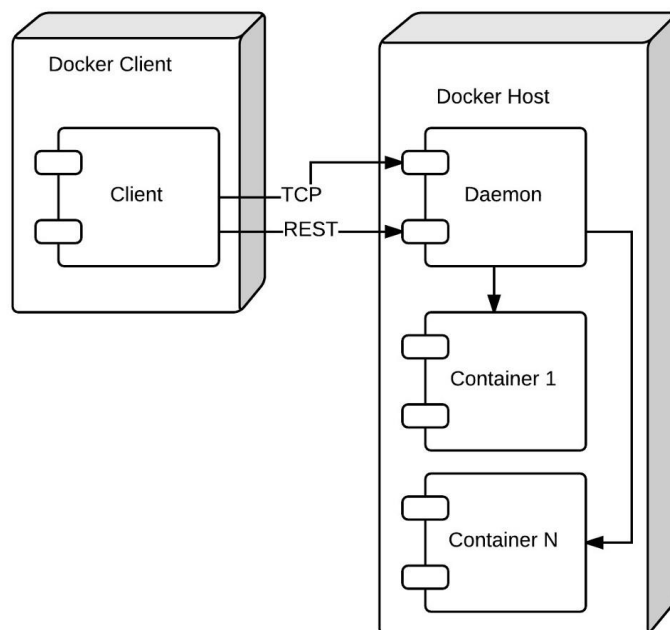


图 3.6 客户端与 Docker 的关系

通过前面的分析，我们可以知道容器技术能够通过相关的命令操作对容器资源进行有效控制，但对于大型的服务器应用，例如需要千万台服务器时，例如 Hadoop 平台，单纯的容器技术无法达到高效操作的目的，那时因为目前的 Docker 技术在单一的机器操作上技术比较成熟，但在集群化操作上仍无法很好的胜任。虽然在 2015 年初，Docker 开始在其环境中支持多容器编排分布式应用，即引进了 Swarm 和 Compose 这两个基于集群方面的应用，但都还处在测试版，性能上并不够稳定。目前来看，由于 Docker 技术发展的时间还不长，虽然成长迅速，但在这方面的技术还不够成熟，根据我们的调研，目前比较成熟的一个解决方案是使用 Google 公司的最新开源软件 Kubernetes，该软件是基于谷歌十年来的容器使用经验来开发的，所以具有比较大的实用性。

Kubernetes 的主要思想是对资源进行抽象，并以 REST 的形式开放给开发人员和管理人员使用，Kubernetes 的可操作对象为：pod、service、和 replicationController，分别在不同的级别上对容器进行控制，然而由于 Kubernetes 仍然处于早期的发展阶段，性能上并不是很稳定，根据我们的实际部署情况来看，Kubernetes 甚至出现过宕机的情况，且由于 Hadoop 的各个节点所需要开放的端口较多，纯粹使用 Kubernetes 管理的话，在使用容器时是极不方便的。因此，虽然 Kubernetes 具有很好的集群化管理容器的能力，但目前来看，还不能很好的适应于 Hadoop 平台的部署。因此，虽然可以使用 Kubernetes 进行 Hadoop 平台的部署，但出于对稳定性的考虑，我们就不建议使用 Kubernetes。

若出于稳定性的考虑的话，目前我们使用另一种方案来构建 Docker 集群，即使用 Docker

的桥接方式，把每一个 Docker 所创建的容器都作为一个虚拟机进行管理。要使用 Docker 创建能够被外网访问的容器，需要进行一些额外设置，我们利用容器的底层实现原理，通过脚本，动态的为每个创建的容器创建对应的桥接网口。这样能够实现跨服务器间的容器间的互相访问，也更接近真实的应用环境。当然，该方法的缺点就是管理容器变得比较麻烦，对每个容器都需要进行设置。

综上，共给出了两台搭建 Docker 集群环境的方案，基于这两个方案，在结合之前所创建的 Hadoop 容器镜像，我们就可以在容器上建立部署分布式 Hadoop 平台了。

### 3.5 本章小结

通过本章内容，我们不但验证说明了容器技术相比于传统虚拟化技术具有无可比拟的优势，更重要的是，我们实现了 Hadoop 平台基于容器技术的部署，虽然就现阶段的情况来看，基于 Docker 来部署 Hadoop 集群相比于传统的部署方式在人员操作要求上更高，但是随着 Docker 技术的发展，相信不久就会有更好的管理平台出现，但有一点我们可以确定的是，Hadoop 平台作为需要大量读写数据的云计算平台，部署在 Docker 下比部署在传统的虚拟机上有更好的性能表现。

就本文而言，为了优化 Hadoop 对海量小文件的处理，我们验证了在 Docker 下，读写数据的性能比 KVM 下的读写性能更高，同时，在 Docker 上部署 Hadoop 具有接近物理主机的资源利用率。

并且，由于 Docker 把数据与运行环境进行了分离，所以可以把构建好的 Hadoop 平台作为镜像发布，方便随时添加或替换 Hadoop 节点，这能够提高部署 Hadoop 时的工作效率。Docker 作为容器技术的最佳实践，其性能优势使得其能够替换现有的虚拟化技术，现在国内已经有结合 Docker 与 OpenStack 的研究（如文献<sup>[30]</sup>）和在基于 Docker 的 PaaS 平台研究（如文献<sup>[31]</sup>），这将使得 IaaS 与 PaaS 的界限变得更加模糊，Docker 势必会引领下一场云计算技术浪潮，未来将会有越来越多的企业把 Hadoop 平台构建在容器上。

所以，本课题把 Docker 技术引入到 Hadoop 平台部署中来，是符合 Hadoop 平台以及云计算未来发展方向的。

## 第四章 软件层面优化处理

本章主要介绍我们在为构建处理海量小文件的 Hadoop 平台时软件实现的方案，通过分析 Hadoop 对于小文件的处理方案，讨论具体实现过程，从而给出了一个自己的小文件处理方案。

### 4.1 方案构架说明

本文是针对 Hadoop 平台下处理海量小文件的研究，在调研过程中，我们发现一般在处理 Hadoop 平台的小文件时，通常都是需要根据具体问题具体分析，结合对应的项目，来编写对应的处理模块，而通用的方法无非都是合并文件的思想，且目前的文件合并都是基于 Hadoop 的归档及序列化文件的方法。

#### 4.1.1 序列化文件方法

序列化文件方法即 SequenceFile，是 Hadoop 的 API<sup>[32]</sup>提供的一种针对二进制小文件处理方案，它是一种文件合并方案，通过将多个二进制小文件合并而进行统一存储。其主要的数据结构是一系列的二进制键值对，即在这种方案下，所有的文件名都将存入数据结构的 key 中，而文件的内容则存入该种数据结构的 value 中，通过此种方法就能够将大量的二进制小文件合并成一个大文件。其原理图如图 4.1 所示。

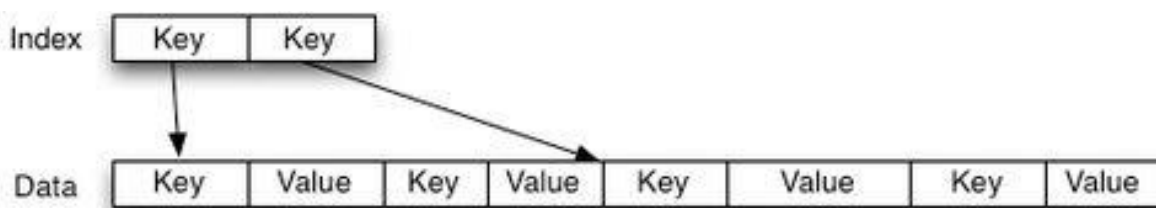


图 4.1 序列化文件原理图

根据资料，我们可以直观的进行一个说明，当某文件夹下有 100000 个 10KB 左右的小文件时，就能够通过编写一个序列化文件处理程序，把这些小文件存入到 SequenceFile 中，这样就能够极大的减少文件数量。使用序列化文件技术还有一大好处就是该技术因为是官方提供的技术支持，所有能够有许多基于此技术的扩展支持，比如支持数据压缩，我们知道，对于海量数据来说，哪怕是很小的数据压缩都能给整个系统带来很大的数据空间节约，且有利于集群中的数据传输，所以说序列化文件技术在对应处理小文件时还是存在一定优势的。但

序列化文件技术却还是有一个比较大的劣势的，那就是序列化文件方法中的数据结构并没有建立小文件到大文件的映射关系，即没有建立类似于索引的方法，所以在查询小文件时，就需遍历整个序列化文件，从而在文件读取时会降低读取效率。

### 4.1.2 归档文件方法

在处理小文件问题时，Hadoop 平台有一个比较简单的处理方案即归档文件方法（HAR），在 Hadoop 的 0.18 版本中开始引入。我们知道在使用 Hadoop 处理小文件时一个不可避免且难以很好解决的问题就是 NameNode 因大量存储文件名而造成的内存负荷，而归档文件方法就是主要用来缓解这一问题的。对所有的客户端来说，可以直接使用归档文件，不会有任何影响，所有的原始文件都是可见的且是可获得的，HAR 文件仅仅是通过在 HDFS 上建立了一个层次化的文件系统来工作，其结构原理如图 4.2<sup>[33]</sup>所示，在 HDFS 端使得其内部的文件数减少了，从而使得 NameNode 的内存开销得以减少。

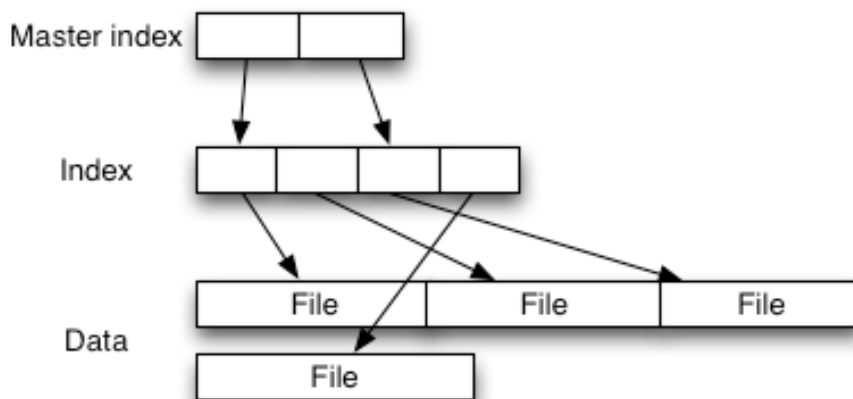


图 4.2 归档文件原理图

在处理小文件时，我们可以直接使用 Hadoop 的交互式命令中 `archive` 来生成相应的.har 文件，这个命令实际上是会运行一个 MapReduce 任务来将小文件进行打包成.har 文件。例如，要归档“/fc/src/2014/1[0-2]”这三个文件夹到“/fc/har/2014/”，可以直接使用如下命令：

```
hadoop archive -archiveName combine.har -p /fc/src/2014/ 10 11 12 /fc/har/2014/
```

在需要在客户端获得 HAR 文件的具体包含的小文件信息时，也可以直接使用命令来获得，例如在包含有多个子文件及文件夹的目录下，假如路径为/user/heipark/fc/，可以通过 `archive` 命令获得相关信息，命令可以仿照如下：

```
hadoop archive -archiveName src.har -p /user/heipark/fc/ /user/heipark
```

运行上述命令过程中会生成一个“/user/hdfs/.staging/har\_93ftj7/\_har\_src\_files”文件，其中的 `har_93ftj7` 中，`har` 为固定前缀，后面为随机字符串，`_har_src_files` 为固定文件名，其主

主要内容会包含原始文件的结构信息。

通过对比原始文件夹和 `_har_src_files` 中的数据，我们可以看到每个原始的小文件的对应实际存储位置情况，从而我们可以理解对于使用 HAR 文件来说，所有的原始文件都是可见的且是可获得的，HAR 文件只是在 HDFS 上建立了一个层次化的文件系统，我们可以随时获得对应的小文件。

然而，虽然采用归档文件方法处理小文件能够减少 NameNode 的内存使用，但是通过生成的 .har 文件来读取一个文件并不会比直接从 HDFS 中读取文件更快，甚至可能更慢，我们在测试中的结果也显示了其可能出现花费时间增多的情况，因为从图 4.2 归档文件的原理图中我们可以知道，对每一个 .har 文件的访问，在抽取需要的文件时，都需要完成两层文件的索引读取和文件本身数据的读取，所以势必会花费较多的时间。

而且，我们知道，Hadoop 的核心计算框架是 MapReduce，Map tasks 通常是每次处理一个文件块的输入。如果在文件大小与数据块的大小的情况下，此类文件的数量较多时，那么每一个 map 任务都只能处理相对较少的输入数据，并且会产生大量的 map 任务，而每一个 map 任务都是需要消耗资源的，众多的 map 任务势必会导致资源的极大开销。例如在处理一个大小为 2GB 的文件时，默认文件块大小为 64M 的话，消耗的时间肯定是远远小于处理两万多个大小为 100KB 的文件，因为后者虽然总大小也为 2GB，但产生的 map 任务也会有两万多个，这么多的工作任务会使处理时间增加千百倍。所以，虽然归档文件的方法通过 MapReduce 来处理大量文件的输入，但归档文件中并没有比较好的方法来将 .har 文件中的被打包过的文件当作一个 HDFS 文件来处理，所以无法做到很好的文件处理操作，所以可见归档文件方案局限较大。

### 4.1.3 方案分析

从上面两节中我们可以看出，现阶段的对于在 Hadoop 平台下的小文件处理的通用方案都是存在一定缺陷的，而且上述两种方案也仅是考虑了对于小文件存储的优化，特别是归档文件的方案，仅仅是缓解了小文件对于 NameNode 的内存使用过多的问题，但对于 Hadoop 平台来说，小文件的产生其实是有两种情况的，一种就是源文件为小文件的情况，而另一种情况就是在对文件的操作时而产生的文件例如日志文件等，特别是 MapReduce 计算过程中会不断进行数据的读写，因此，纯粹的使用归档技术和序列化文件技术是不够的。

当然，也有一些针对 Hadoop 下特定的小文件类型的处理方案，在论文中常见的有如 WebGIS 系统的小文件处理优化，在处理 WebGIS 系统的数据时，浏览器与服务器之间数据的



传输都是以小文件形式的，因为需要考虑到网络中的数据传输速度，所以存储到 Hadoop 中的都是被切分成 KB 级别的小文件。所以设计人员为了提高存储速度，便利用 WebGIS 系统中被切分文件的相关性特征，将相邻地理位置信息的小文件合并成一个大的文件进行保存，同时建立索引，以便对小文件进行访问<sup>[34]</sup>。另外，针对中国电子教学共享系统 Bluesky，在解决 Hadoop 下的小文件存储时提出的优化方案也是基于所需存储文件的特性的，由于主要存储的内容是演示文稿文件和视频文件，所以设计人员在实现时，做了两方面的工作：第一，将属于同一个课件的文件合并成一个大文件，这样能提高小文件的存储速度；第二，使用了一种 two-level prefetching 机制，该机制主要包括两项操作索引文件的预取和数据文件的预取，这样同样能提高小文件的读取速度<sup>[35]</sup>。

然而，若需要使 Hadoop 处理的小文件更具有普遍性，则一般需要在系统层面上改变 HDFS 架构来提高 Hadoop 存储小文件的速度。例如可以考虑将大量的小文件存储到一个数据块中，并且使该数据块所在 DataNode 的内存空间存储这些小文件的元数据信息<sup>[36]</sup>。这一种方法虽然是直接在 HDFS 系统层面上实现了小文件的合并，而且能够减少 NameNode 的内存空间的浪费问题，但是使用 DataNode 的内存空间来存储元数据或索引文件，还是会对以后大量小文件的读取或检索速度造成一定影响。

综合 Hadoop 在处理小文件时所遇到的性能瓶颈的原因，我们可以发现，Hadoop 在处理小文件时，若是简单的使用归档文件方法或序列化文件的方法是不够的，但我们通过分析，可以确认我们在优化 Hadoop 平台处理小文件的性能时，需要在下面两个方面进行优化：

- 1) 在写入小文件时，需要对小文件进行适当的合并，以减少在写文件时对 NameNode 的请求数量，从而减少对 HDFS 的性能的影响<sup>[37]</sup>。
- 2) 在读文件时，需要抵消因文件合并而造成读取文件时的额外计算开销，例如在归档文件和序列化文件中读取文件所产生性能开销。

## 4.2 优化方案设计

根据前文所做的分析，我们可以明确，要优化 Hadoop 平台处理小文件的性能，需要从两方面进行处理，分别需要优化读操作和写操作。当然，虽然说是分别进行优化操作，但因为读操作所需要处理的文件数据都是是基于写操作存入的文件的，所以这两者也需要做到互相配合。

### 4.2.1 小文件存储优化

在 Hadoop 平台下，应对其存储系统中小文件存储效率问题的主流思想是将小文件合并或组合为大文件，目前主要的方法分为 2 种，一种是利用 Hadoop 归档(Hadoop archive, HAR)等技术实现小文件合并的方法，另一种则是针对具体的应用而提出的文件组合方法。当然，通过我们前面的分析可知，这两种通用解决方案在性能上还是有待提高的，所以我们需要设计一套自己的方案来对小文件进行合并。

对于文件合并，其方法有很多种，有些合并是需要考虑到所需存储的文件的特性的，例如在处理“中华字库”工程数据时<sup>[38]</sup>，因为文件具有很好的目录结构，所以在合并时能够直接根据其相关逻辑而产生对应的索引，这样在文件读取时能够根据其逻辑结构而方便的找到对应的文件。而我们知道，在使用 Hadoop 平台时，很多情况下我们不能很好的保证我们的文件具有很好的逻辑结构，所以，我们在探讨 Hadoop 处理小文件时的优化方案时，需要尽可能的找寻一种较为通用的方法。当然，我们知道由于文件种类的繁多，要找寻一种通用的方案还是比较困难的，但考虑到使用 Hadoop 的情况通常是要进行数据处理的，而要进行数据处理通常是基于文本信息的，而非文本在 Hadoop 中通常是作为一种“冷数据”存在的，很少使用这些非文本文件进行计算，例如，我们参与的中科院声学所的语音标注项目中，所要处理的文件有两种格式，一种是语音文件，一种是每个语音所对应的文本文件，而语音文件虽然不大，但能够达到 Hadoop 平台中数据块的大小，而因为 Hadoop 中的 NameNode 存储信息是以文件块或单个文件来作为单位的，所以即使对其进行文件合并，意义也不是很大，其消耗的 NameNode 内存资源几乎一样，而且额外的操作也会增加时间开销。而对于文本文件才是我们需要关注的，文本文件本身就小，所以为了后续的处理性能，需要对这些文本文件进行合并。

一种合并文本文件的方案就是根据“局部性”原理，即认为相邻的小文件在读取时被需要的可能性较大，因此可以在使用时进行预读取，一些针对 Hadoop 的小文件优化也正是基于这一点来实现的<sup>[39]</sup>。而我们也知道，Hadoop 是用来处理大量数据的，处理海量的小文件时，其关注的重点是文件中的数据，如果我们把大量的小文件合并成一个大文件，只要保证其文件中的内容可分析，则其不会影响最终的数据计算结果，所以我们可以直接对小文件进行合并操作，汇总成一个大文件。

我们也通过实验简单测试了一下上传一个拥有较多小文件的文件夹至 Hadoop 和合并小文件后上传到 Hadoop 所分别花的时间，结果显示，直接上传所有小文件，使用 time 命令计时的话 real 时间是 0m29.919s，而合并小文件花费 0m0.435s，上传花费 0m18.403s，合计不超

过 20s，低于上传小文件所花的时间，所以可以说明我们的方法具有可行性。

### 4.2.2 小文件读取优化

由于我们在对小文件存储时采用合并文件的方法，所以在读取文件时，我们将会获得一个包含许多小文件内容的大文件，然而，若该大文件内的内容中的所有数据都是同一类型的，则在处理数据时，那么我们就可以直接进行读取操作并处理。例如，在语音文件中，众多的语音文本文件汇总成一个较大的语音文本文件后，其中所包含的内容还是不变的，要进行分析处理的话是不需要再关注其原本的小文件的，例如要统计其中所有的短句中最长用的语句，这样的操作都是可以直接进行的。

这样一来，我们对于小文件的读取操作的优化则是需要在其处理操作上了。我们知道，传统的处理方案都是使用 **MapReduce** 计算方式的，这种计算方式有一个比较大的不足之处，即在运算过程中会不断的进行读写操作，其在读写操作过程中势必会产生一些小文件，这些额外的小文件会极大的影响，因此，为了提高处理性能，我们在进行计算时，使用 **Spark** 进行数据处理。

在前面的介绍中我们已经对 **Spark** 进行了简单的说明了，因为其可以使用 **Hadoop** 平台的 **HDFS** 作为数据源，所以我们可以把我们写入的合并后的小文件的数据提取出来进行处理。为了比较 **Spark** 和 **MapReduce** 在性能上的差异，我们分别使用 **MapReduce** 和 **Spark** 对我们的数据进行处理，对大约 10G 的数据进行字数统计处理。我们同时也使用 **Ganglia** 监控我们的 **Hadoop** 平台，**Ganglia** 是伯克利大学发起的一个开源项目，主要是用来监控集群系统的性能，例如：cpu、内存、硬盘利用率，读写负载、网络流量等性能都能进行监控<sup>[40]</sup>。因此，我们可以使用该软件的数据来分析其性能差异。截图见图 4.3 和图 4.4 所示。

通过比较图 4.3 和图 4.4，我们可以发现，使用 **Spark** 是会使用更多的系统内存，且无论是内存还是磁盘读写，其处在峰值的时间都更长，说明其在数据处理时充分利用了系统的资源。

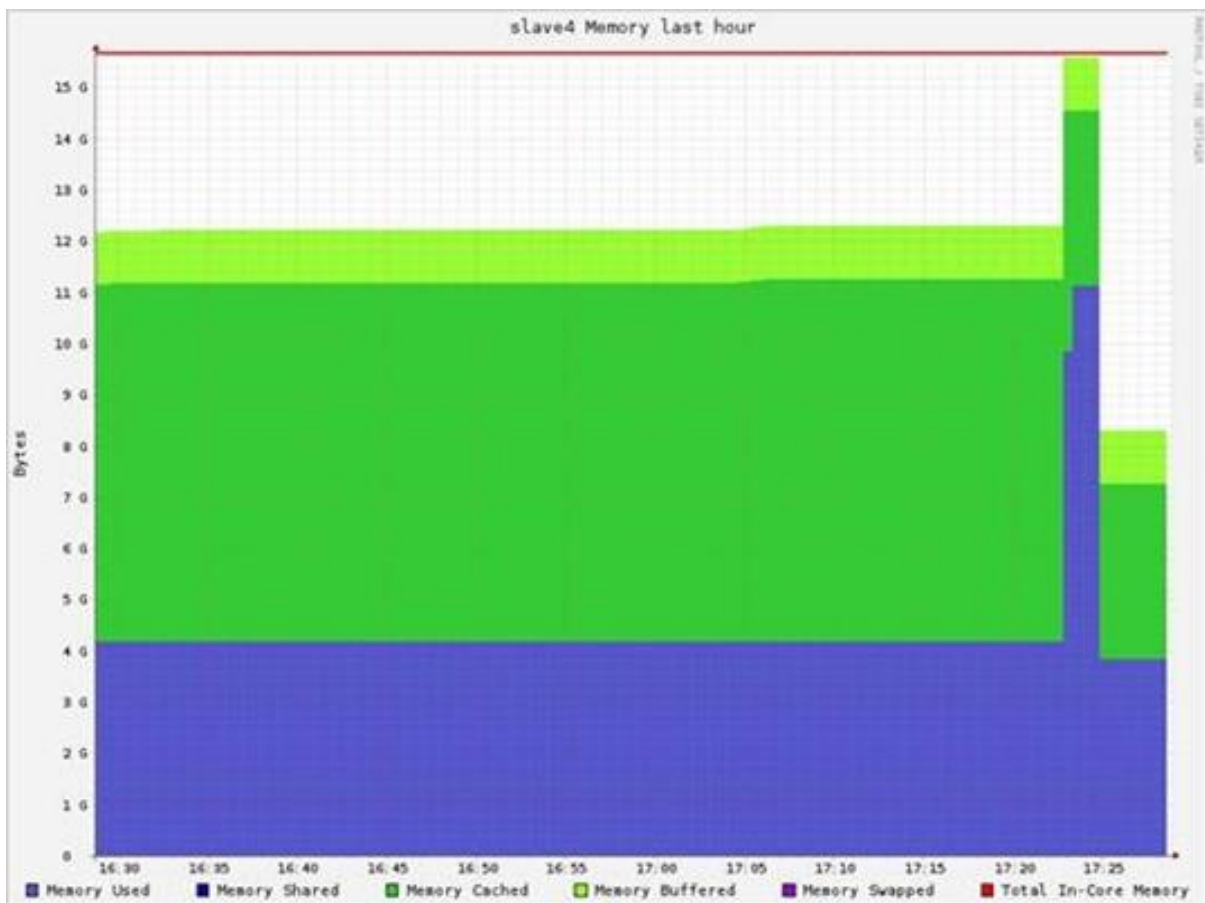


图 4.3 Spark 运行单词计数后的情况

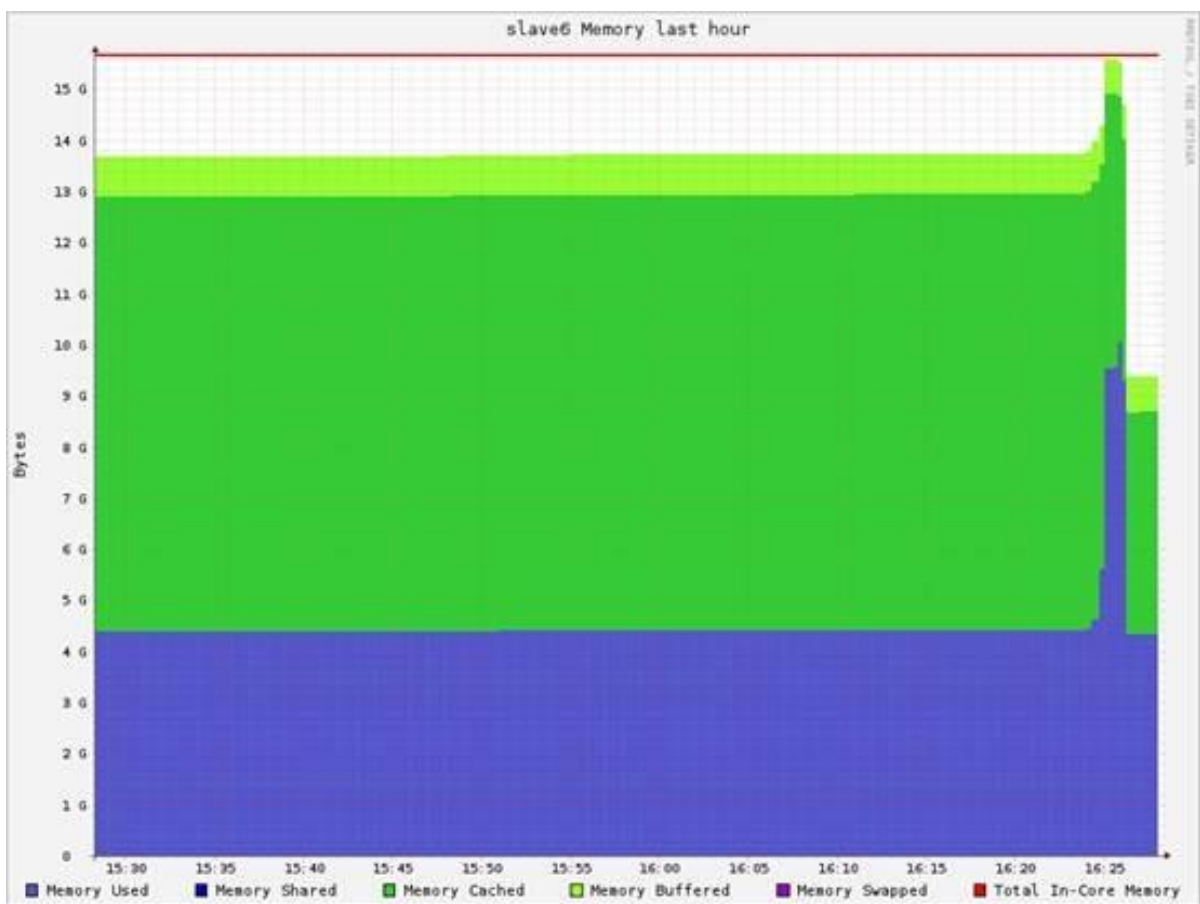


图 4.4 MapReduce 方法运行单词计数后的情况

同时,我们也测试了 MapReduce 与 Spark 在做运算时的性能差异,图 4.5 是使用 MapReduce 进行一次简单文件运算的时间,图 4.6 是使用 Spark 对同样的文件进行计算时所耗费的时间。

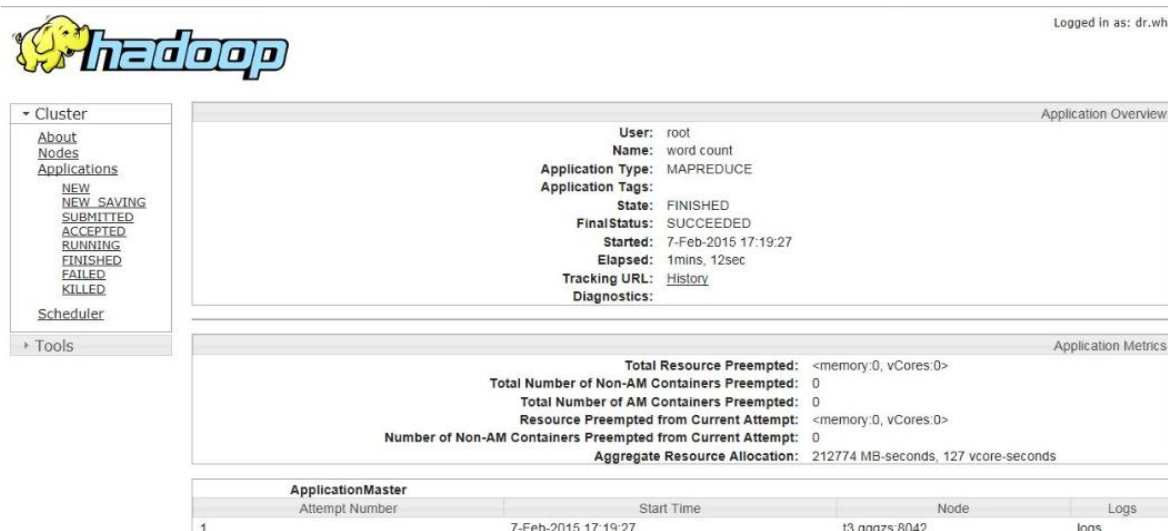


图 4.5 MapReduce 处理一个任务的耗时情况

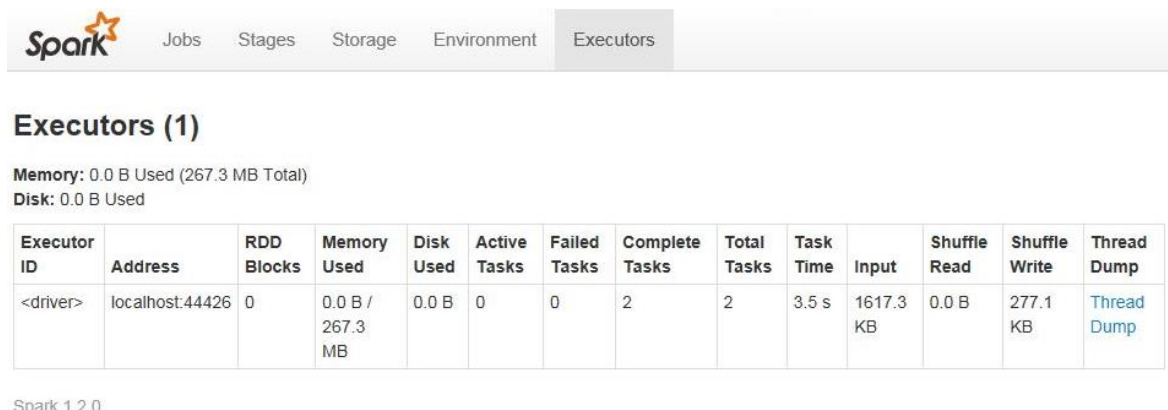


图 4.6 Spark 处理一个任务的耗时情况

对比图 4.5 和图 4.6,我们可以清楚的了解到,相比于 MapReduce 的一分多钟的耗时,Spark 只需几秒就能处理结束,由此可见 Spark 在处理能力。

总结上面的两次比较,可以明确通过使用 Spark 能比 MapReduce 更高效的处理数据。也因此,使用 Spark 来对 Hadoop 平台下的合并后的小文件进行处理是较好的。

### 4.2.3 方案逻辑结构

结合前两节的分析,我们对 Hadoop 处理小文件的优化方案是:在向 Hadoop 平台进行写入数据时,并不是直接向 NameNode 发起请求,而是通过客户端进行先进行小文件分类及合并工作,客户端类似于一个缓冲区,当这个“缓冲区”写满后,再把这些文件与 Hadoop 平台进行对接,通过调用 `FSDDataOutputStream.write(buffer,0,bytesRead)`来实现文件上传。在进行文件读取时,则是通过使用 Spark 来进行读取计算的。具体的软件组织结构见图 4.7。

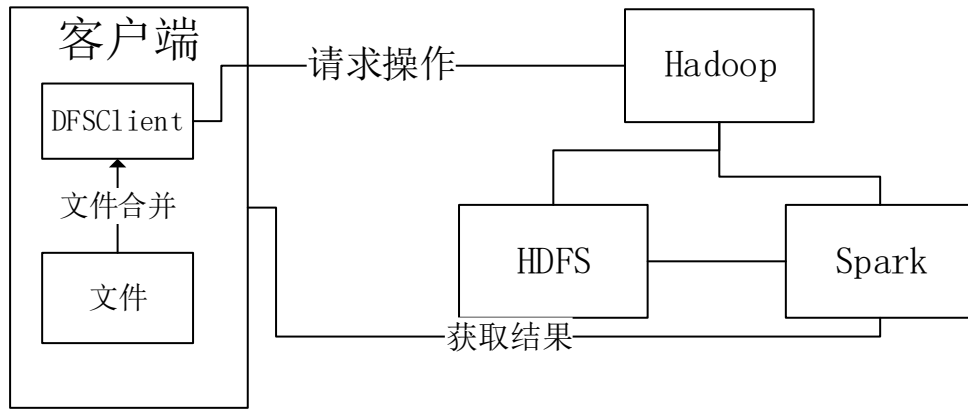


图 4.7 软件组织结构图

由图 4.7 可知，我们在客户端上首先对小文件在写入前进行了处理，而在处理数据时，则依托 Spark 进行数据处理。软件运行过程见图 4.10。

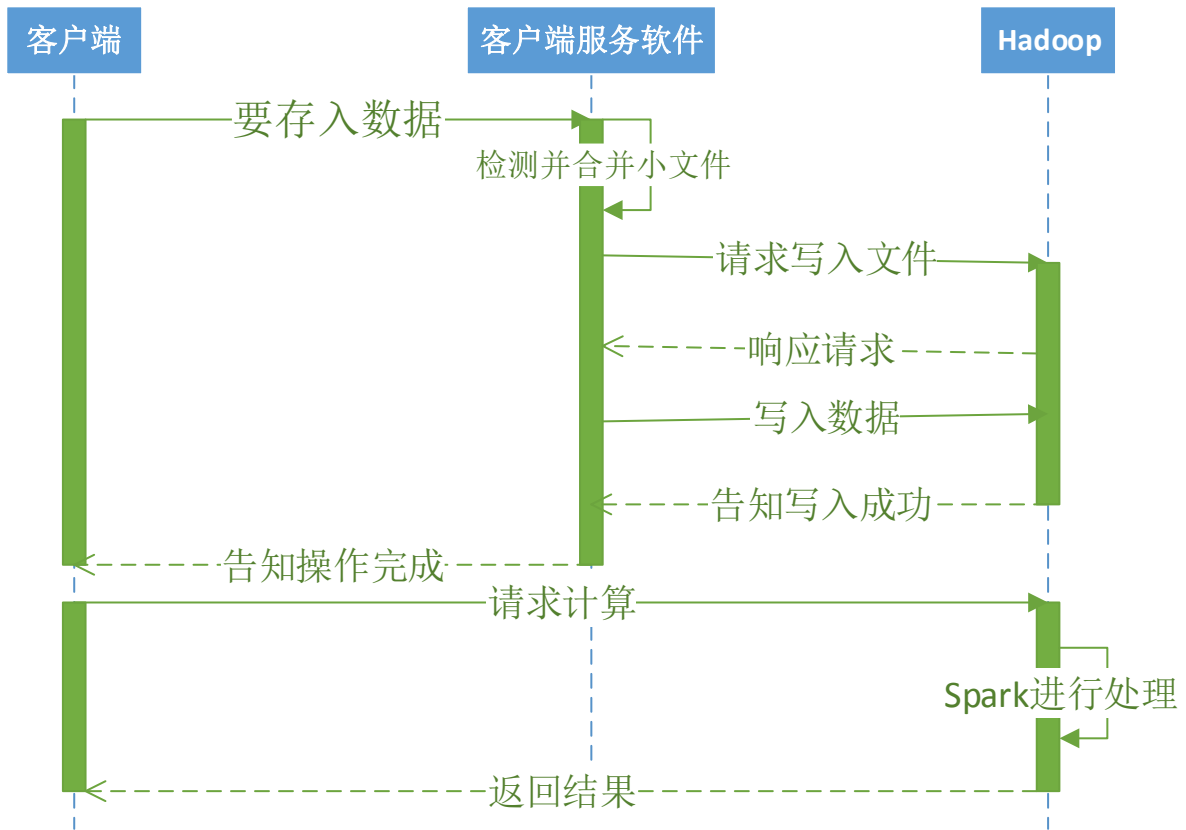


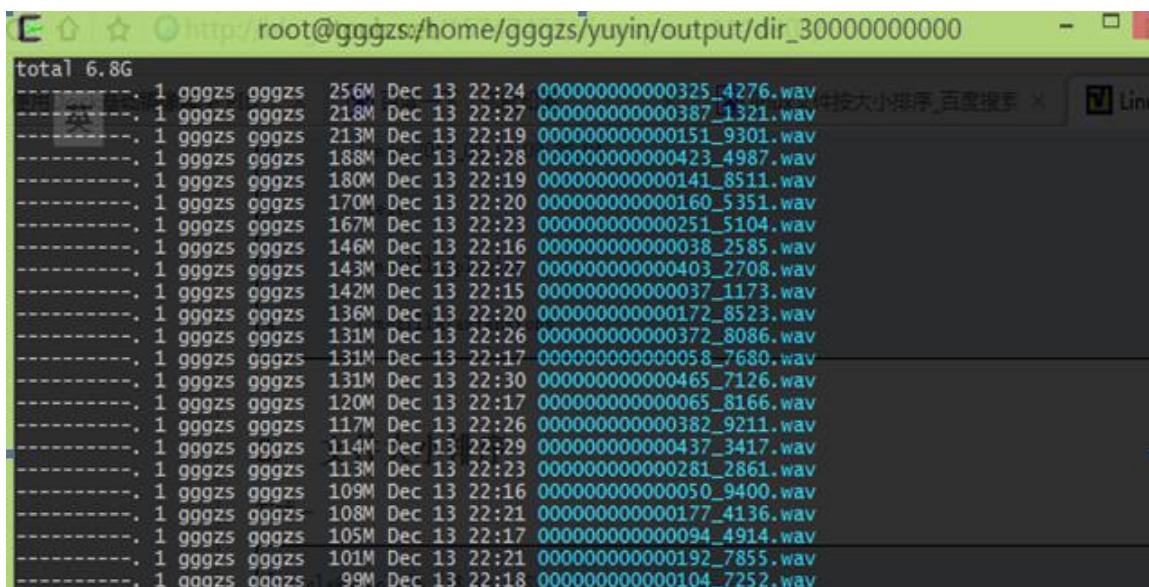
图 4.8 操作序列图

由图 4.7 和图 4.8 可知，我们在对小文件进行合并时，有个重要的客户端服务软件，它主要就是实现了小文件的合并。该客户端服务软件处在客户端，主要是在客户端直接对所要处理的文件进行前期处理，首先对文件进行判断，把文本小文件进行合并，并把与这些小文件相关联的非文本文件保存在对应的同一文件夹内。因为是直接利用客户端进行文件处理，首先，肯定是可以减轻服务器的资源，其次，因为在 Hadoop 集群中，其文件块都是属于大文件，所以在内核优化上回选择使用 anticipator 来作为/sys/block/sda/queue/scheduler 的值，而客



客户端作为处理小文件的场所，为了提高处理速度，应使用 `deadline` 参数，该参数是针对小文件处理的，所以把小文件合并放在客户端进行也是有利于提高处理速度的。

关于小文件的合并方法，我们在合并小文件时，考虑到合并文件主要都为文本文件的情况，而在读取时又是采用整个文件块同时读取，再提取文件的方法，所以我们对文本文件的合并的策略是：首先，每个小文件都有一个唯一的文件名，类似于磁盘的 UUID 号，且该文件名与其相关联的非文本文件除文件类型标识的后缀名外都一致，以便于根据文本文件提取对应的非文本文件。如图 4.9 所示的音频文件，其文件名都是有一个唯一的编号的，而文本文件也都是有唯一编号的，如图 4.10 所示，相同的编号的文件即为相互关联的文件。且由图中我们可以看到只有文本文件才是小文件，对应的非文本文件都不算是小文件了，所以，这已是我们只选取文件文件进行文件合并的原因，减少合并文件的数量，有助于提高我们的整体处理速度，当然，由于语音数据中会存在一些错误或不可用语音，该类语音没有对应的文本文件，且文件较小，对于此类语音，我们采取的做法是依据语音数据包文件夹名进行分类后采用序列文件的形式进行合并后存储，其相关的文件信息在文本文件合并时会被记录到文本文件中。因为此类文件被访问使用的几率极低，所以其需要读取处理的操作较少，使用序列文件的方式足够应对了。



```
root@ggz: /home/ggz/yuyin/output/dir_3000000000
total 6.8G
-----, 1 ggz ggz 256M Dec 13 22:24 00000000000325_4276.wav
-----, 1 ggz ggz 218M Dec 13 22:27 00000000000387_1321.wav
-----, 1 ggz ggz 213M Dec 13 22:19 00000000000151_9301.wav
-----, 1 ggz ggz 188M Dec 13 22:28 00000000000423_4987.wav
-----, 1 ggz ggz 180M Dec 13 22:19 00000000000141_8511.wav
-----, 1 ggz ggz 170M Dec 13 22:20 00000000000160_5351.wav
-----, 1 ggz ggz 167M Dec 13 22:23 00000000000251_5104.wav
-----, 1 ggz ggz 146M Dec 13 22:16 00000000000038_2585.wav
-----, 1 ggz ggz 143M Dec 13 22:27 00000000000403_2708.wav
-----, 1 ggz ggz 142M Dec 13 22:15 00000000000037_1173.wav
-----, 1 ggz ggz 136M Dec 13 22:20 00000000000172_8523.wav
-----, 1 ggz ggz 131M Dec 13 22:26 00000000000372_8086.wav
-----, 1 ggz ggz 131M Dec 13 22:17 00000000000058_7680.wav
-----, 1 ggz ggz 131M Dec 13 22:30 00000000000465_7126.wav
-----, 1 ggz ggz 120M Dec 13 22:17 00000000000065_8166.wav
-----, 1 ggz ggz 117M Dec 13 22:26 00000000000382_9211.wav
-----, 1 ggz ggz 114M Dec 13 22:29 00000000000437_3417.wav
-----, 1 ggz ggz 113M Dec 13 22:23 00000000000281_2861.wav
-----, 1 ggz ggz 109M Dec 13 22:16 00000000000050_9400.wav
-----, 1 ggz ggz 108M Dec 13 22:21 00000000000177_4136.wav
-----, 1 ggz ggz 105M Dec 13 22:17 00000000000094_4914.wav
-----, 1 ggz ggz 101M Dec 13 22:21 00000000000192_7855.wav
-----, 1 ggz ggz 99M Dec 13 22:18 00000000000104_7252.wav
```

图 4.9 语音文件截图

Go to parent directory								
Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
000000000013331_7830.textgrid	file	3.00 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup
000000000013332_9984.textgrid	file	3.88 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup
000000000013333_1845.textgrid	file	4.61 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup
000000000013334_8212.textgrid	file	2.13 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup
000000000013335_0494.textgrid	file	1.57 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup
000000000013336_2417.textgrid	file	7.64 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup
000000000013337_1321.textgrid	file	1.93 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup
000000000013338_6243.textgrid	file	3.94 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup
000000000013339_5357.textgrid	file	4.07 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup
000000000013340_9441.textgrid	file	1.66 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup
000000000013341_0180.textgrid	file	2.22 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup
000000000013342_1801.textgrid	file	2.34 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup
000000000013343_7188.textgrid	file	2.58 KB	2	128 MB	2015-02-06 15:47	rw-r--r--	root	supergroup

图 4.10 语音相关联的文本文件截图

其次，我们在写入大文件过程中，该文件名会作为重要的信息插入到合并后的文件中，以便于读取时的文本小文件查找对应的非文本文件。我们在合并文件时，采用把文本文件写成 XML 的形式，小文件的唯一文件名作为元素的属性值，而文本文件内容则是元素内容，使用 XML 的好处是便于一些程序语言的 XML 编程接口，降低使用成本，且 XML 文本形式能够很好的在一个文本文件中保存具有各种格式的文本文件，具有很好的兼容性。因此，我们在合并小文件时，把小文件是汇总成大的文本文件的，而且，在进行数据分析时，我们也可以直接分析该大文件文本，而不需要再拆分成具体的小文件，减少的数据的再处理量。

综上，就是我们在本文中提出的在 Hadoop 平台处理小文件时进行的软件层面的优化方案的结构及实现思想，即通过写文件时的合并处理，以及在读取时的整体读取，优化了处理海量小文件时的性能。

### 4.3 本章小结

这一章我们详细介绍了 Hadoop 下处理小文件所用到的技术，以及简单介绍了现有技术的优缺点，并根据我们实际使用情况，提出了在软件层面上优化处理的思想，分析了我们使用的文件合并优化的方法的优势，以及在读取处理时的优化方案的优势，从而说明了本文提出的方案的可行性。



## 第五章 方案性能评估

本文是针对 Hadoop 平台下处理海量小文件的研究，主要通过架构上以及通过编写对应的处理软件模块来提高处理海量小文件的能力，并寻找一条适合企业进行部署的具有高性价比的部署架构方案。本章正是通过完整的介绍我们所做的工作了，并结合实验数据，来说明我们的优化方案能够在一定程度上适用于企业使用 Hadoop 平台处理海量小文件的情形。

### 5.1 方案架构

我们发现如果仅从 Hadoop 平台的文件读写的算法上进行优化，在性能上的提高相对有限，因此我们从 Hadoop 平台的整体上开始进行优化，并仿照企业级应用来搭建我们的测试平台。

#### 5.1.1 底层环境构建

如我们在前文中提到的，Hadoop 平台在企业级应用时往往是构建在基于虚拟机的 IaaS 层上，而我们在第三章中也已经介绍了我们在容器上构建 Hadoop 平台的方案。所以，我们的实验平台也正是以该方案为基础。

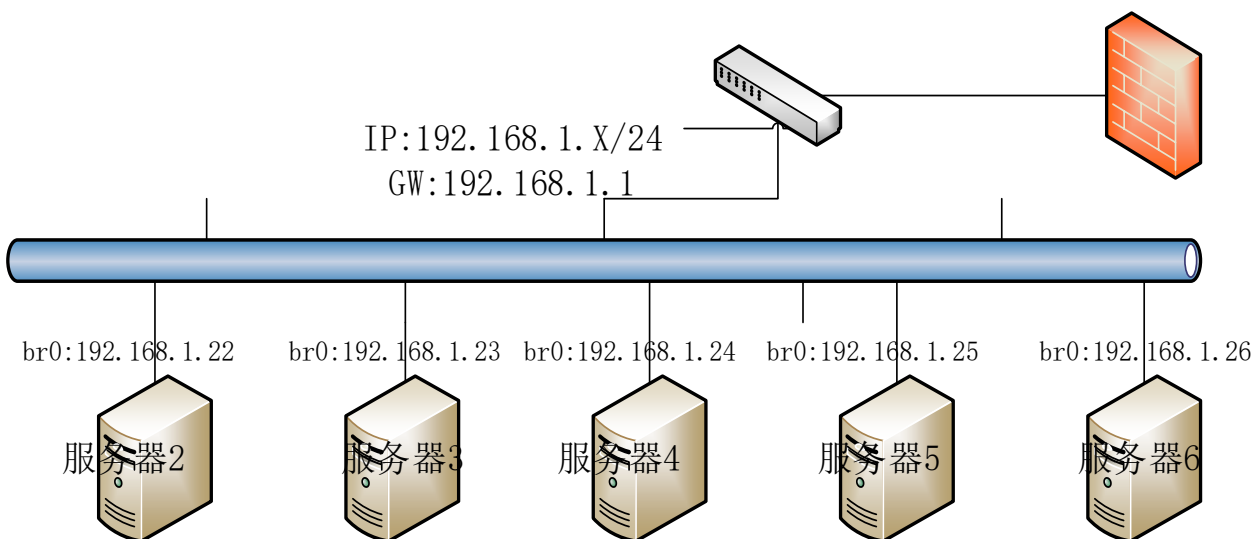


图 5.1 系统硬件架构环境图

首先，在现有的五台服务器上安装 Docker，使用 Docker 搭建基于容器的 Hadoop 集群，其架构图见图 5.1。另外在现有的提供 IaaS 服务的超云平台上进行部署一组作为对比用的基于虚拟机的 Hadoop 平台，超云平台的硬件架构见图 5.2。

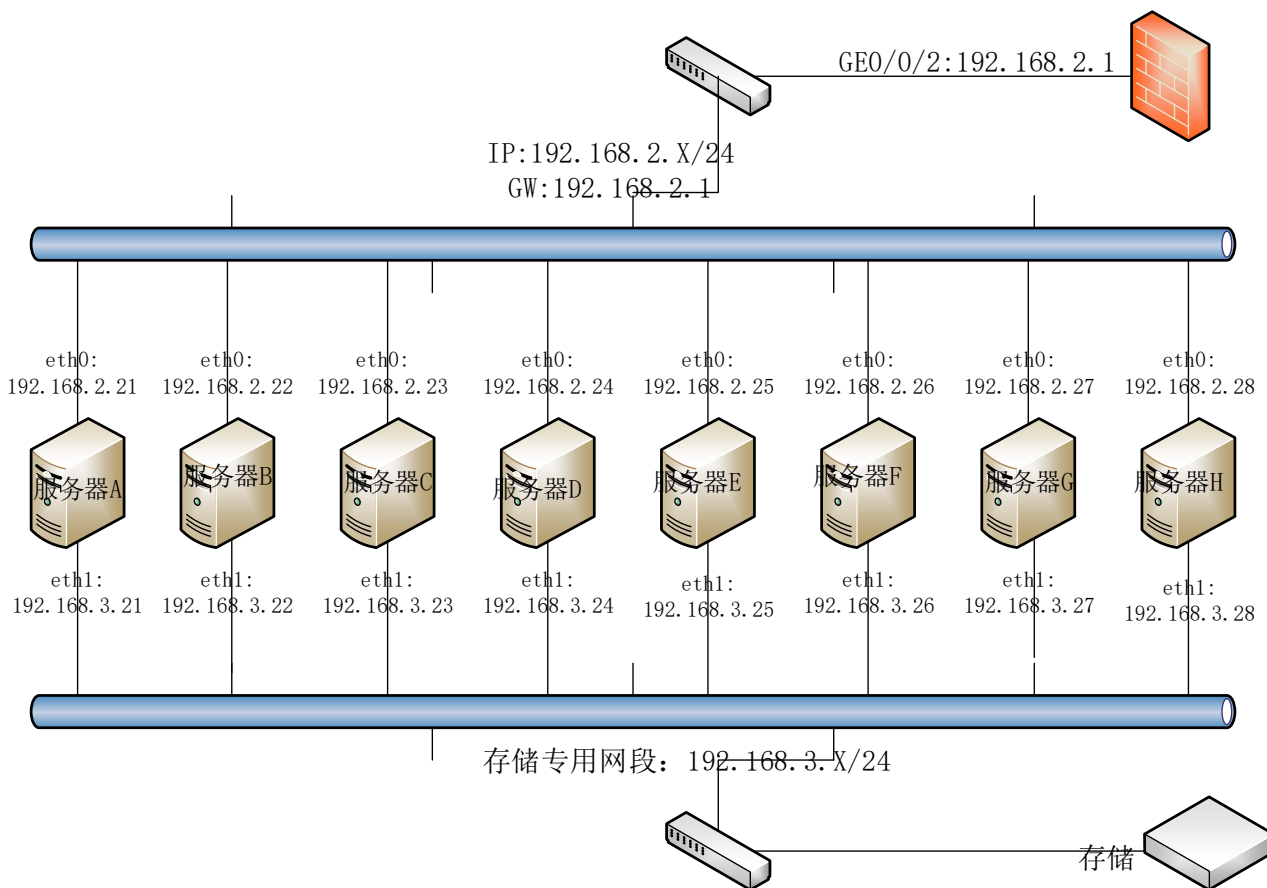


图 5.2 超云集群架构图

以上便是我们为验证优化方案所构建的实验环境的底层架构。

### 5.1.2 软件搭建

针对 Docker 的环境，前面我们分析了 Docker 搭建集群环境的两种方案，由于 Kubernetes 在性能上还不够稳定，所以我们在搭建 Docker 集群时使用程序进行控制，通过自写的程序对 Docker 集群进行控制，以实现容器跨主机间的互通，并使用自行通过标准镜像及 Dockerfile 及相关配置文件构建所需要的运行时的 Hadoop 容器。主要操作为：

- 1) 为容器所在的物理机配置桥接网络；
- 2) 配置 Docker 守护进程，使 Docker 启用对应的桥接网卡；
- 3) 启用容器；
- 4) 启动编写的程序，为容器设置网络环境，以实现容器间通信；

由于 Docker 的集群功能还在不断的完善当中，无法做到智能的进行 Hadoop 容器的创建，Kubernetes 之类的平台只适合于创建短生命周期的容器，所以我们根据 Docker 的相关文档，

编写了相关程序，调用容器的相关接口，解决容器间通信问题，来满足我们创建基于 Hadoop 集群的相关需求。

虚拟机上的 Hadoop 集群环境的搭建则按照 Hadoop 的官方文档的搭建方式进行搭建，虚拟机则是通过超云平台进行创建。为了简化操作，我们把客户端处理程序放置在 Hadoop 集群的主节点上。

以上，便是本文的实验环境的软硬件搭建情况，处理过程中使用的系统版本为 ubuntu14.04，Hadoop 版本使用的是在实验时可获得稳定版本 2.6.0，由于 Hadoop 更新较快，所以在实验时选择了更为稳定的版本。

## 5.2 实验分析

根据本论文提出的方案，我们通过搭建相关的环境，进行了实验验证，主要通过比较基于虚拟机的 Hadoop 平台处理大量小文件和基于容器的 Hadoop 平台处理大量小文件，以及比较未经优化的 Hadoop 平台与使用本论文思想优化后的 Hadoop 平台在处理 Hadoop 平台下大量小文件的性能，来验证本文的设计思路。

### 5.2.1 小文件存储性能的比较

测试所使用的文件夹每个包括 1000 个语音文件以及与其对应的文本文件信息，文本文件的大小都属于 KB 级，平均一个文件夹的大小为 8G 左右。我们分布使用 1 个文件夹、10 个文件夹、20 个文件夹的数据来进行测试，其结果如表 5.1 所示。

表 5.1 文件存储时的性能比较

对比状态	原文件数	耗时
优化前	2000	7m26.067s
	20000	60m56.502s
	40000	137m34.506s
优化后	2000	4m18.403s
	20000	46m36.444s
	40000	104m49.486s

由于实验所使用的文件是由大量的大文件和小文件的混合组成的，所以在存储过程中还

是会有比较多的大文件的存储耗时，但从实验结果中我们还是可以清楚的看出，经过优化后的 Hadoop 平台在处理小文件的存储过程中起到了不错的优化效果，且可以看出随着文件数量的增加，其所减少的时间更多，即起到了较大的提速效果，所以可以说，使用本文所设计的方案来进行海量小文件的存储是能够有效降低时间消耗的。

### 5.2.2 小文件处理性能的比较

在进行小文件处理性能的比较时，我们采用对文本信息进行简单的 wordcount 运算的方法来测试性能，我们采用的数据是在小文件存储性能测试时所上传到 Hadoop 中的数据，实验结果见表 5.2。

表 5.2 小文件数据处理的性能比较

对比状态	原文件数	耗时
不合并小文件处理	2000	21m12s
	20000	223m21s
	40000	463m34s
合并小文件后处理	2000	1m12s
	20000	26m36s
	40000	54m49s
使用 Spark 处理	2000	3.5s
	20000	1m52s
	40000	4m13s

虽然我们的实验数据只能定性的说明优化前后的情况优劣，但还是可以看出，随着数据量的增加，能够明显体现出优化后的方案的优势，时间消耗得到了很好的降低，这说明了在面对海量小文件处理的情况下，我们的对于小文件的优化做法是可取的，可以在企业应用中进行部署。

总体来看，我们的优化方案无论是在小文件的存储还是读取处理上，都能够起到一定的作用，且因为我们在对小文件进行处理时，先把小文件与其相关联的大文件进行了分类，这样更具有通用性，因为现实情况下很多时候是大文件与小文件混合存在的，所以我们这样做预先处理有利于减轻 Hadoop 平台的负担。

### 5.3 本章小结

通过本章我们完成了对 Hadoop 平台的优化方案的实验说明，本章首先介绍了使用本文方案构建的实验环境架构，并通过实验，来验证本文的优化方案的可行性。通过实验数据的分析，我们可以认为，采用本文的方案对 Hadoop 平台进行优化后，对海量小文件进行存储和处理都能在性能上得到一定的提高，虽然没有在硬件层面是使用增加固态硬盘的方式来提高处理速度，但正是因为我们的方案并不需要在硬件层面上的进行额外的资金投入，所以也比较易于在企业中现有的环境中进行部署实施。

## 第六章 总结与展望

### 6.1 总结

本文我们通过从架构上对 Hadoop 的优化，以及在软件上对 Hadoop 的处理方法的改进，提高了 Hadoop 平台在针对小文件时的存储效率，同时减少了由于大量小文件的 MapReduce 处理而造成的读写消耗，从而提高处理效率。

我们在本文中的工作主要在下面几个方面：

- 1) 引入容器来作为 Hadoop 的架构平台，实现了 Hadoop 平台基于容器技术的部署，Hadoop 平台作为需要大量读写数据的云计算平台，部署在 Docker 下比部署在传统的虚拟机上有更好的性能表现。我们通过实验，验证了在 Docker 下，读写数据的性能比 KVM 虚拟机下的读写性能更高，使得在 Docker 上部署 Hadoop 具有接近物理主机的资源利用率，且使用 Docker 部署 Hadoop 的容器可以比传统部署安装方式更为便捷简单，一次制作好镜像就可以在后续直接使用。
- 2) 优化小文件的存储方案，采取小文件合并的方式，来缓解因存储大量小文件而造成的 NameNode 的内存消耗，同时能够有效的减少 DataNode 的读写次数，从而提高 Hadoop 的整体存储速度，且能够有效提高在文件读取处理时的文件读取效率，从而有助于 Hadoop 的数据处理，我们的方案正是实现了根据文件大小，通过适当合并小文件，来达到优化目的的。
- 3) 把小文件的部分处理工作与 Hadoop 平台进行了分离，存储文件与处理文件相分离，因为 Hadoop 平台适合处理大文件，在 Hadoop 优化时，会对部署 Hadoop 平台的系统内核进行优化，调整系统的读写调度算法，而小文件的合并时涉及到大量小文件的读写，通过在客户端调整读写算法，使其以适合小文件读写的方式进行处理，这样有助于提高处理速度。且因为在处理 Hadoop 的数据时，我们使用了 Spark 计算框架，充分利用内存，减少磁盘读写，同样能够提高处理性能。

综上，我们在本文中通过分析 Hadoop 平台在处理海量小文件所遇到的问题瓶颈，分析了现有的优化方案，并提出了自己的优化方案，即使用 Docker 构建 Hadoop 平台，小文件采取合并存储，使用 Spark 框架处理数据的方法，我们通过对优化方案的解析并构建了实验平台，通过实验验证了优化方案在处理小文件时的有效性。

## 6.2 展望

本文主要是针对在 Hadoop 平台下处理海量小文件的性能优化，通过对 Hadoop 平台的分析与研究，我们提出了相应的解决方案，并通过实验，验证了其有效性，能够有效助力于海量文本数据的分析处理，例如，中移动在为了辨识用户投诉问题的真实原因，并为了发现问题和改进产品，中移动使用了天睿公司开发的客户投诉智能识别系统，把投诉内容作为源头，通过智能文本分析，实现了从发现、分析、解决问题以及跟踪评估的全套闭环管理，而我们提出的优化方案正好适用于此类文本分析情景，针对客服电话录音所产生的大量文本小文件，我们的方案能够有效的提高数据处理的速度。但因为我们的方案的优化重点是针对文本文件的，所以还存在一定的局限性，但可以进一步进行深入下去，把各类非文本信息转化成文本信息后，同样适用于我们的这套方案。

另外，本文的一个重要工作就是实现了使用 Docker 构建分布式部署 Hadoop。Docker 作为现在使用最多的容器软件，已逐渐成为一个容器软件标准，即使后续出现了一些诸如 Rocket 之类的新的容器软件，都已无法撼动其行业地位了，而容器作为一个云计算虚拟化产业的新的增长点，越早开始进行基于 Docker 的应用研究就能越早进行生成环境的部署实施，从而在应用的获得收益。例如，在南京青奥会期间，我们就把 Docker 应用在青奥比赛的成绩系统的 ODF 消息的测试当中，有效的提高了工作效率。当然，在 2015 年初，出现了许多分布式容器管理相关的生态圈软件，这些软件虽然目前还都处在测试版中，但其应用前景还是比较大的。但就目前来看，为了保证稳定性，诸如 etcd 与 Kubernetes 之类资源调度分配的软件，都还不够成熟，所以我们在实现基于 Docker 部署 Hadoop 时并没有完全实现部署的自动化，只做到了半自动化的部署，但相信随着容器技术的完善以及凭借着容器技术所拥有的性能上的优势，会有更多的厂商把其 Hadoop 平台移植到基于容器的管理平台中来。

## 参考文献

- [1] 马爱诚.在线pH测量系统在农药生产中的应用[J].化工自动化及仪表,2007,34(4): 87-88.
- [2] 朱娜娜, 李丽萍.Hadoop 平台的集群故障监控的研究与实现 [J]. 软件,2013,34(12): 73-77.
- [3] 李冠辰. 一个基于 hadoop 的并行社交网络挖掘系统 [J]. 软件,2013,34(12): 127-131
- [4] 高东海, 李文生, 张海涛. 基于 Hadoop 的离线视频处理技术研究及实现 [J]. 软件,2013,34(11): 5-9.
- [5] 田野.Hadoop下海量遥感数据的处理 [J]. 软件,2014,35(3): 91-93.
- [6] Eric Sammer.Hadoop Operations[M]. O'Reilly Media, Inc, USA,2012.
- [7] 蔡斌, 陈湘萍.Hadoop技术内幕: 深入解析Hadoop Common和HDFS架构设计与实现原理[M].北京: 机械工业出版社,2013.
- [8] Lublinsky B, Smith K T, Yakubovich A.Professional Hadoop Solutions [M].Wrox,2013.
- [9] 张鑫. 深入云计算: Hadoop源代码分析[M]. 北京: 中国铁道出版社,2013.
- [10] 陈光景.Hadoop小文件处理技术的研究和实现[D]. 南京:南京邮电大学图书馆,2013.
- [11] 万川梅, 谢正兰.深入云计算: Hadoop应用开发实战详解[M].北京: 中国铁道出版社,2013.
- [12] Venkat Subramaniam.Programing Scala:Tackle Multi-Core Complexity on the Java Virtual Machine[M]. The Pragmatic Programmers,2009.
- [13] Horstmann,C S. Scala for the Impatient[M]. Addison-Wesley Professional,2012.
- [14] 冯琳.集群计算引擎Spark中的内存优化研究与实现[D].北京: 清华大学图书馆,2013.
- [15] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: cluster computing with working sets[C]. Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, 2010: 10.
- [16] XIN R S, ROSEN J, ZAHARIA M, et al. Shark: SQL and rich analytics at scale[C].Proceedings of the 2013 international conference on Management of data, 2013: 13-24.
- [17] 杨保华,戴王剑,曹亚仑.Docker技术入门与实战[M].北京: 机械工业出版社,2014.12.
- [18] 肖德时.深入浅出 Docker[EB/OL].[2015-01-05] [http://www.infoq.com/cn/articles/docker-core-technology-preview?utm\\_source=infoq&utm\\_medium=related\\_content\\_link&utm\\_campaign=relatedContent\\_articles\\_click](http://www.infoq.com/cn/articles/docker-core-technology-preview?utm_source=infoq&utm_medium=related_content_link&utm_campaign=relatedContent_articles_click).
- [19] Daniel Compton.Why Docker and CoreOS' split was predictable[EB/OL]. [2015-01-05] <http://danielcompton.net/2014/12/02/modular-integrated-docker-coreos>.
- [20] Gabriel Lowy. Application Performance Management Enables DevOps ROI [EB/OL]. [2015-01-05] <http://www.apmdigest.com/application-performance-management-enables-devops-roi>.
- [21] Garber L.News Briefs[N].IEEE, 2014, 47(11): 14-18.
- [22] Chris Swan.Docker: Present and Future[EB/OL]. [2015-01-05] <http://www.infoq.com/articles/docker-future>.
- [23] Mike Kavis.Blurring The Line Between PaaS And IaaS[EB/OL]. [2015-01-05] <http://www.forbes.com/sites/mikekavis/2014/06/02/blurring-the-line-between-paas-and-iaas/>.
- [24] Nati Shalom. Do I need OpenStack if I use Docker [EB/OL]. [2015-01-05] <http://pensource.com/business/14/11/do-i-need-openstack-if-i-use-docker>.
- [25] Dua R, Raja A R, Kakadia D.Virtualization vs Containerization to Support PaaS[J].IEEE, 2014(41): 610-614.
- [26] 黄刚,徐小龙,段卫华.操作系统教程[M].北京: 人民邮电出版社,2009.
- [27] Dustin Boswell,Trevor Foucher.The Art of Readable Code[M]. O'Reilly Media, Inc, USA,2012.
- [28] Bica M, Bacu V, Mihon D, Gorgan D. Architectural Solution for Virtualized Processing of Big Earth.IEEE, 2014 (10): 399-404.
- [29] James Turnbull.The Docker Book[M]. Amazon Digital Services, Inc.,2014.



- [30] 张忠琳, 黄炳良. 基于OpenStack 云平台的 Docker 应用[J]. 软件, 2014, 35(11): 73-76.
- [31] 鞠春利, 刘印锋. 基于Docker的私有PaaS系统构建[J]. 轻工科技, 2014(10): 80-83.
- [32] Parker D, Dasdan A, Hsiao R, et al. "Map-reduce-merge: simplified relational dataprocessing on large clusters", Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pages 1029-1040, New York, NY, USA, 2007. ACM Press.
- [33] 李智. Hadoop 关于处理大量小文件的问题和解决方法 [EB/OL]. (2010-11-22) <http://www.csdn.net/article/2010-11-22/282301>.
- [34] Liu XH, Han JZ, Zhong YQ, Han CD, He XB. Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS [C]. Proc. of the 2009 IEEE Conf. on Cluster Computing and Workshops, 2009: 1-8.
- [35] Bo Dong, Jie Qiu, Qinghua Zheng, et al. A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: A Case Study by PowerPoint Files [C]. IEEE International Conference on Services Computing, 2010: 65-72.
- [36] Liu Jiang, Bing Li, Meina Song. The optimization of HDFS based on small files [C]. Broadband Network and Multimedia Technology (IC-BNMT), 2010: 912-915.
- [37] HDFS Users Guide [EB/OL]. [2015-01-23] [http://Hadoop.apache.org/common/docs/current/hdfs\\_user\\_guide.html](http://Hadoop.apache.org/common/docs/current/hdfs_user_guide.html).
- [38] 张春明, 芮建武, 何婷婷. 一种 Hadoop 小文件存储和读取的方法 [J]. 计算机应用与软件, 2012, 29(11): 95-100.
- [39] 杨彬. 分布式文件系统 HDFS 处理小文件的优化方案 [J]. 软件, 2014, 35 (6) : 65-69.
- [40] Ganglia Development Team [EB/OL]. [2015-01-05] <http://ganglia.info/>.

## 附录 1 攻读硕士学位期间撰写的论文

- (1) 马越、黄刚，基于 Docker 的应用软件虚拟化研究，软件，已录用；

## 附录 2 攻读硕士学位期间参加的科研项目

- (1) 江苏省高校教育学会基金 (JSSY1201);
- (2) 南京邮电大学基金(SG1107)。

## 致谢

从 2012 年 9 月入校以来，转瞬间，在南邮的研究生生活即将结束。在过去的三年间，通过学习相关专业知识，通过实际项目经验，我在实践中不断提高。对我来说，这三年是个很重要的时间，是个承上启下的三年，把本科所学的知识与实际项目相结合，初步融入社会，可以说是个学习和收获的三年。在此临毕业之际，我要在此向那些给予我关怀和帮助的人表达我内心最诚挚的谢意。

首先，我要感谢我的导师——黄刚老师。在我读研究生期间，黄老师不仅在生活上同时在工作学习中，都给予了我很大的帮助与指导。黄老师为人十分随和，跟着黄老师，我不单是学到了专业知识，更多的，是学到了很多为人处事的道理，这是对我这即将走向社会的同学最有价值的知识。对黄老师的感谢之情也绝非一句谢谢所能够表达。作为黄老师的开门弟子，我既感到万分荣幸，同时又觉得万分惶恐，总是害怕自己不够优秀无法为师门争光。祝您在今后的日子里身体健康，万事如意！

其次，我还要感谢所有在 308 和 304 教研室的各位同学，你们是一个充满活力的集体，其中的每一个人对我来说，遇到你们都是我人生中莫大的幸运。感谢你们每一个人，正是你们，才有了这么个互勉互励、积极进取的团队，也正因为有你们的帮助，我的能力才会取得如此迅速地提升，能和你们结识，是我莫大的荣幸，谢谢你们了！

此外，我还要特别感谢我的父母，你们的养育之恩已使我无法用语言表达谢意，现在更要谢谢你们能支持我进入到如此好的学校来完成研究生学业。

最后，谢谢江苏省电信对于我们实验室的支持，感谢你们为我们提高如此好的实验设备平台，谢谢！感谢南京 Linux 用户组的各位，感谢你们在技术上对我的指导！感谢 InfoQ 的各位不断把最先进的技术资料介绍给我，让我受益匪浅！感谢 VRM 团队的各位在实践中给予我的帮助！感谢优之杰的李正锦、于伟，以及嘉环、群顶、曙光、超云的各位工程师在技术上给予的支持！感谢在 51CTO 论坛上给予我解答的董西成老师等人！

最后，再次向那些给予我指导和帮助的老师、同学和朋友表示感谢！感谢南京邮电大学对我的培养，同时，感谢各位审查论文的专家、评委，谢谢！