本 科 毕 业 论 文

课题：面向大规模视频处理的 **Hadoop** 云系统

学员姓名：谭瀚霖　专业：系统工程

培养类型：技术类　学号：200905017018

所属学院：信息系统与管理学院　年级：2009 级

指导教员：涂　丹　职称：副教授

所属单位：信息系统与管理学院系统工程系

国防科学技术大学训练部制

目　录

图表目录

摘要

论文研究了面向大规模视频的计算平台关键技术，将开源云计算平台 Hadoop 引入视频分析领域，从平台建设、算法移植、性能实验三个方面进行了详细阐述。

平台建设方面，论文在原始的 Hadoop 云系统基础上，结合其子项目 Fuse-DFS 和视频处理库 OpenCV、FFMPEG 及其 Java 接口项目 JavaCV，从而实现了面向大规模视频处理的 Hadoop 云系统。为验证平台效能，通过人脸检测和运动目标检测与跟踪两个算法移植实例展示利用 Hadoop 框架逐帧读取视频并送入 MapReduce 计算模型的方法，利用 JavaCV 实现已有的视频处理算法的方法和利用 Map 对输入视频进行冗余分组方法。论文中通过组建六个节点的Hadoop集群进行性能实验。实验发现视频处理程序在 Hadoop 上的运行时间与输入视频的大小存在线性关系，与集群子节点数存在反比例关系，并据此提出运行时间模型。实验表明，Hadoop 云系统对视频处理有明显的加速效果，五个子节点的集群即可将处理时间缩减到单节点的 25%以下，而且集群的加速潜力极大。只要有足够的集群规模，理论上可以达到任何加速比。

关键词：Hadoop　MapReduce　大规模视频数据　Hadoop 性能

Abstract

This paper proposes a solution, A Hadoop Cloud System Facing Large Scale of Video Data Processing, which introduces the open source cloud platform, Apache Hadoop, to process large quantities of video data. The paper focuses on three aspects: platform construction, algorithm port and performance analysis, to demonstrate how and how well the solution works.

The introduced Hadoop system consists of the original Apache Hadoop, one of its sub projects named Fuse-DFS, the well-known video processing libraries FFMPEG and OpenCV as well as their Java interface project JavaCV. JavaCV ports FFMPEG and OpenCV to Java language. And Fuse-DFS makes Hadoop's distributed file system(HDFS) available to FFMPEG and OpenCV. Thus, the introduced Hadoop system takes advantages of the two video processing libraries and works fine with current video processing algorithms.

There are many things to be noticed before you successfully porting video processing algorithms to the Hadoop system. Chapter Three gives two demos of porting face detection algorithm and moving object detection and tracking algorithm, separately, to illustrate how to get video frames from input, how to group frames with Map method and how to use JavaCV to realize video algorithms. In addition, this Chapter introduces other techniques to deal with huge input video files and real time input video streams.

The author built up a Hadoop cluster of six computers to conduct performance experiments. The results show that the system is able to reduce the running time to below 25% of that of a single computer. However, there exists a limit to the acceleration. Impressing conclusions also includes two laws about running time: Linear Law and Inverse Proportion Law. Combining the two laws and the author gets a mathematical model that can predict the running time of the Hadoop system with different numbers of cluster slave nodes and different input video sizes.

Keywords: Apache Hadoop, video processing, MapReduce, Hadoop performance

第一章 绪论

1.1 研究背景及问题提出

随着信息技术的发展，人类产生了越来越大规模的数据，特别是视频数据。大规模视频监控系统在交通控制，如交通路口、高速公路等，敏感公共场所，如机场、火车站、银行等，的部署越来越多，由此产生了大量视频数据。

越来越多的海量视频数据有着处理和分析的需求。这种需求超越了大大超出了人工处理的能力，就连传统的计算机处理程序面对大规模视频数前处理的时间超出需求能承受的范围。超级计算是解决问题的途径之一。可是超级计算机本身就价格昂贵、运行维护费用也不容小觑，普通用户难以承受。云平台（Cloud Platform，也称“分布式计算平台”）正是在这种情况下应运而生的。

作为 Apache 顶级开源项目的 Hadoop 云平台软件框架更是以其免费、运行于廉价设备、高可靠性、高容错性和部署灵活性成为众多大型互联网企业的应用选择和云计算领域的研究热点。目前的 Hadoop 分布式处理框架常用于分布式排序、Web 访问日志分析、反向索引构建、文档聚类、机器学习、数据分析、基于统计的机器翻译和生成整个搜索引擎的索引等大规模数据处理工作，并且已经在很多国内外知名的互联网公司内部得到广泛应用，比如百度和淘宝，雅虎和谷歌。但目前将 Hadoop 用于多媒体数据处理的研究尚不成熟，相比于文本处理，此方面的文献资料也很少。面向大规模视频数据处理的研究更是还处于起步阶段。

本文的工作，正是要架起 Hadoop 云平台和大规模视频数据分析之间的桥梁，充分利用 Hadoop 云平台的云存储、云计算能力和现有视频分析技术，实现对大规模视频数据的快速而有效的分析。

1.2 国内外研究现状

随着大数据时代的到来，Hadoop 是当前十分热门的研究对象。大致可以分为算法移植和性能优化两类。目前已有利用 Hadoop 做图像分类、视频转码的论文发表。它们提出了一些利用实验性的方案，在 4-8 个节点的小集群上做了相关实验[3,

[4, 5]。但是它们对于平台建设和编程方法的描述往往比较模糊，缺乏可操作性。

Hadoop 大集群的研究和应用前沿不在院校，而在需要使用它解决实际大数据分析问题的各大互联网公司。Yahoo! 通过集群运行 Hadoop，以支持广告系统和 Web 搜索的研究；Facebook 借助集群运行 Hadoop，以支持其数据分析和机器学习；百度则使用 Hadoop 进行搜索日志的分析和网页数据的挖掘工作；淘宝的 Hadoop 系统用于存储并处理电子商务交易的相关数据；中国移动研究院基于 Hadoop 的“大云”（BigCloud）系统用于对数据进行分析和并对外提供服务。但是工程师所做的大量应用并没有形成论文或理论发表，难于借鉴。研究领域出现发表论文研究人员不能接触大集群，接触大集群的工程师很少发表论文的尴尬局面。这很不利于对 Hadoop 大集群性能的研究。

目前对于利用 Hadoop 云平台处理视频的研究并不成熟：无论是平台建设方面、还是算法移植方面。如何让视频处理搭上 Hadoop 云平台的快车，是视频处理和云技术应用的交叉研究方向。其科研工程和商业应用前景十分广阔。

1.2.1 现有的大规模视频分析技术概况

存储是分析的基础。因此本小结将从大规模视频数据的存储和分析两个层面来介绍现有的大规模视频分析技术概况。

1.2.1.1 大数据及其处理方法

大数据通常是指大小超出普通软件在可容忍的时间内能获取、存储、处理能力的数据集[6]。它的大小一直随时间增长，已由最初的几十 TB 增长到 PB 级。

大数据有以下三个特点[7]：

a) 大存储量(high-volume)；

b) 高存取速度(high-velocity)；

c) 形式多样(high-variety)。

因而需要新手段来分析大数据，以支持决策、发现规律和优化处理。目前，大数据的存储和分析主要有两大类方法：一是利用映射/聚合系统(以下简称 MR 系统)；二是利用并行数据库管理系统(Parallel Database Management System，以下简

称并行 DBMS)方法[8]。

这两种方法各有优缺点，并且优缺点有互补性。并行 DBMS 有如下优点：

a) 并行 DBMS 的性能明显优于 MR 系统。
b) 并行 DBMS 比 MR 系统更加节能。
c) 并行 DBMS 有 25 年技术发展的积累，在访问速度、新型数据存储、数据压缩、复杂并发查询方面有明显的优势。

并行 DBMS 的缺点是可拓展性差。而作为 MR 系统的典范，Hadoop 具有如下优势：

a) Hadoop 系统的安装使用比并行 DBMS 容易。（并行 DBMS 较难配置）
b) 当节点故障时，Hadoop 系统恢复的工作量比并行 DBMS 工作量小（当然这是以数据冗余并牺牲性能为代价的）。
c) Hadoop 系统可以处理非结构化数据，并行 DBMS 不适合处理非结构化数据。

1.2.1.2 大规模视频数据的分析

大规模视频数据除了具备大数据的一般特点之外，还具备一个重要特点：非结构化。这个特点使得大规模视频数据难于使用关系型数据库进行存储和检索。因而，我将优先考虑另一个云存储方案：分布式文件系统—作为解决大规模视频数据存储的解决方案。

研究大规模视频处理的需求多来自商用领域，目前市场上已经出现一些大规模视频存储和分析的解决方案。大体思路是将包括云计算操作系统、分布式存储系统、分布式资源管理系统的云平台用于视频数据的存储和分析。例如：国内的友友（Yoyo Systems）海量高清视频云存储解决方案提供了完整了视频云存储服务[9]。2012 年 09 月 04 日，新加坡 NEC 公司发布了自主研制的大规模监控视频分析技术，据其网站主页称，可运用到需要人脸识别、人物检索等高精度视频分析功能的监控摄像系统中，通过自动调节监控频率，在一台服务器上的处理能力将是现有技术的 3 倍[10]。

但是这些商用解决方案的技术细节不公开，部署和维护需收费，效果缺乏研究和实验证明。这正是研究开源系统作为大规模视频数据分析平台的意义所在。

受商用解决方案的启发，本课题采用 Linux 操作系统作为云计算操作系统，Hadoop 的 HDFS 分布式文件系统作为云存储系统，Hadoop 核心作为分布式资源管理系统，力求建立开源大规模视频数据分析的解决方案。

1.2.2 现有云平台简介

在介绍 Hadoop 云平台之前，本文首先介绍其他的著名云平台，以使读者对云平台有一个比较全面的认识。许多大型互联网企业推出了面向普通用户的商用云平台。一些著名的云平台如表 1-1 所示。

表 1-1 一些大型互联网企业的云平台

云平台	描述
微软在线 (Microsoft Live@edu)	提供免费的电子邮件，文件存储和许多其他工具。它服务是软件层次的。 **http://www.microsoft.com/liveatedu/**
谷歌应用教育版 (Google Apps Education Edition)	除了微软在线云平台提供的服务，还提供谷歌文档办公处理应用服务等。它也是免费的。 **http://www.google.com/a/help/intl/en/edu/index.html**
谷歌应用引擎 (Google App Engine)	这是一个提供平台服务的引擎，为开发人员提供面向 Web 的开发服务，封装常用的 Web API，将繁琐的 Web 应用部署变为流水线式的简单操作。用户只为其使用的功能付费。 **http://code.google.com/appengine/**
亚马逊网络服务 (Amazon Web Services, AWS)	它提供与谷歌应用引擎类似的功能，还提供更加底层、分化的功能。如简单存储服务（Simple Storage Service, S3），只提供云存储服务，以及弹性云计算平台（the Elastic Compute Cloud , EC2），只提供计算服务。它也是付费使用的。 **http://aws.amazon.com/**

表注：表中内容主要参考了文献[11]，并综合了文献[12]。

表 1-1 中列出的云平台并没有直接面向视频处理的，可见面向视频处理的云平台还并未成熟到商用程度。事实上，大规模视频数据的在云平台的处理并没有一套成熟理论和实践支撑。因而本课题探索 Hadoop 开源云平台在视频数据处理方面的应用是很有价值的。

1.2.3 Hadoop 云平台简介

1.2.3.1 什么是 Hadoop

阿帕奇(Apache) Hadoop 是一个支持数据密集型分布式应用的软件框架。它由 Java 语言编写，支持在大型商用集群上运行应用程序。Hadoop 框架透明地提供可靠的数据移动应用程序。Hadoop 成功地实现了被称为映射/缩减(Map/Reduce)的计算模式，在这个模式下，一个应用程序被分解为许多片段，每一个片段可能在集群的任何一个节点上。并且它提供了一个分布式文件系统将数据存储在计算节点上，在整个集群上提供非常高的聚合带宽。映射/缩减和分布式文件系统都在设计层面上考虑了节点故障的自动处理。它能够使应用程序在上千个独立的计算机上运行并处理 PB 数量级的数据。Hadoop 是从谷歌（Google）的映射/缩减和谷歌文件系统(Google File System, GFS)论文演化而来的。[11]

目前一般认为整个阿帕奇 Hadoop 平台由 Hadoop 核心(Hadoop Kernel)，映射聚合(Map/Reduce)和 Hadoop 分布式文件系统(HDFS)以及一些相关项目——包括阿帕奇蜂巢(Apache Hive)，阿帕奇 Hadoop 数据库(Apache HBase)等组成。[11]

1.2.3.2 Hadoop 平台的结构

Hadoop 云平台可以分为两大系统：任务调度系统和存储控制系统。任务调度系统由 JobTracker 总控，存储控制系统由 Namenode 总控，运行这两个部分的节点都称为主节点。JobTracker 调度任务到 TaskTracker 执行，Namenode 控制数据在 Datanode 的存储。而 TaskTracker 和 Datanode 同时运行于各个子节点上。系统结构如图 1.1 所示。

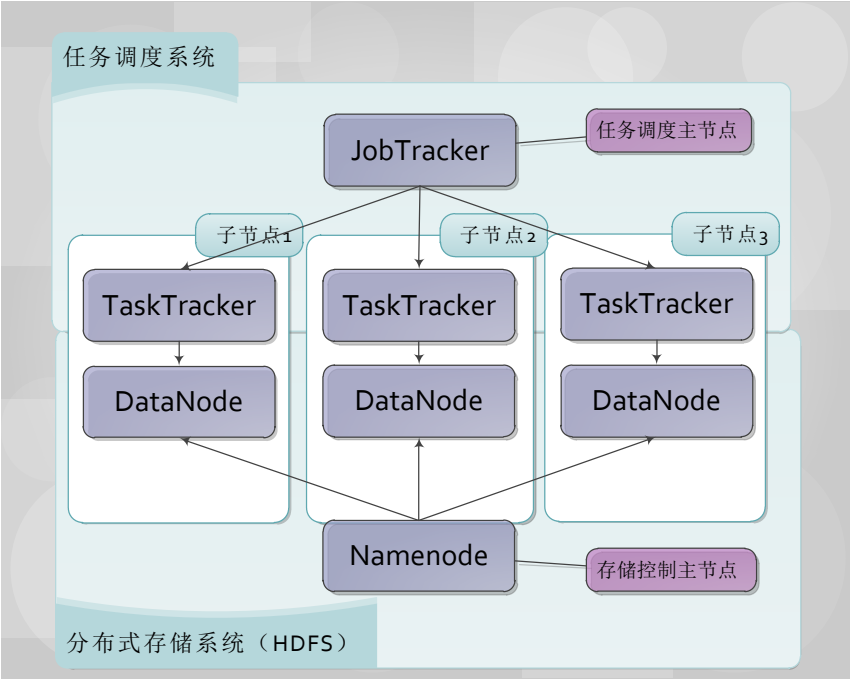


图 1.1 Hadoop 云平台结构图

1.3 主要研究内容及贡献

本文致力于架起 Hadoop 云平台和大规模视频数据分析之间的桥梁，充分利用 Hadoop 云平台的云存储、云计算能力和现有视频分析技术，实现对大规模视频数据的快速而有效的分析。围绕此目标，开展了以下研究和实验：

1） **面向大规模视频处理的 Hadoop 集群的建立。**这里包括两部分工作：一是 Hadoop 集群建立；二是 HDFS 分布式文件系统的本地挂载。在进行分布式文件系统的本地挂载后，利用 OpenCV、FFMEPG 库处理视频具备了可行性。

2） **利用 OpenCV、FFMPEG 视频处理库在 Java 语言下编写视频处理程序。**OpenCV 和 FFMEPG 是视频处理领域著名的库，很多现有算法是基于它们的。然而他们均是 C/C++语言编写的，而 Hadoop 是 Java 语言编写的。这里就有一个从 C/C++到 Java 接口问题。幸运的是，Google 开发了 JavaCV 项目部分解决了这个问题。

3） **在 Hadoop 集群上实现以“人脸检测”算法为代表面向图像的视频处理程序并进行性能分析。**本文以人脸检测算法为单帧视频处理程序的代表，将视频人脸检测程序移植到 Hadoop 的 MapReduce 框架下。尝试不同的具体编程方案，在

获取性能数据后进行对比分析，寻求性能最佳方案。

4） **在 Hadoop 集群上实现以“运动检测与跟踪”算法为代表的面向图像序列的视频处理程序。**人脸检测算法只是单帧视频处理算法，实际应用中“运动检测与跟踪”等连续多帧分析应用很常见。因此本文以“运动检测与跟踪算法”为多帧视频处理程序的代表，也探索将此类程序移植到 Hadoop 框架下的可行性和效果。

5） **在 Hadoop 集群上进行视频分析性能实验并提出运行时间模型。**性能实验发现 Hadoop 云系统的运行时间与输入数据大小和集群子节点数之间都存在明显的关联。据此本文通过建模、验证，提出描述 Hadoop 运行时间的线性关系、指数率以及运行时间模型。

本课题作为科学实验型课题，致力于为工程实践探索面向大规模视频数据分析的云平台解决方案。课题选定阿帕奇顶级项目 Hadoop 作为开源云平台，试图将其应用到大规模视频分析领域，从 Hadoop 云平台建立到适应性部署，从 MapReduce 计算模型介绍到视频处理算法移植，为实际应用这一技术解决大规模视频数据存储分析的难题提供了现实可行的具体方案和性能数据。

1.4 论文结构

本文按“面向大规模视频处理的 Hadoop 云系统构建”、“视频处理技术到 Hadoop 云系统的移植”、“Hadoop 云系统的性能实验与分析”的顺序组织全文。

第一章为绪论，介绍论文的研究背景和研究意义，对与本论文研究相关的国内外云计算和视频处理研究成果进行总结和分析。着重介绍论文的总体研究思路以及需要研究的主要内容，并给出论文的结构安排。

第二章阐述面向大规模视频处理的 Hadoop 云系统，对系统的组成、模型和软硬件配置进行详细阐述。

第三章以人脸检测算法、运动目标检测算法为代表，详细阐述了如何将面向单幅图像、面向图像序列的视频处理算法移植到 Hadoop 云系统上。此外，还介绍了如何分割存储大视频以提升处理速度。

第四章进行了 Hadoop 云系统性能实验，验证了 Hadoop 云系统对视频处理的

加速效果；并提出运行时间模型，对于中型、大型集群的建立具有指导意义。

第五章对全文进行总结，指出本系统的优缺点，并指明下一步工作的方向。

附录详细说明了本系统的安装配置方法和运行调试方法，对于希望部署 Hadoop 集群或本系统的读者是有益的帮助。

第二章 面向大规模视频处理的 Hadoop 云系统

本文研究面向大规模视频处理的 Hadoop 云系统(以下简称“Hadoop 云系统”或“系统”)是在原始的 Hadoop 云系统基础上，结合其子项目 Fuse-DFS 和视频处理库 OpenCV、FFMPEG 及其 Java 接口项目 JavaCV 所构成的。

本章的行文思路是先介绍系统核心：Apache Hadoop 的 MapReduce 并行计算模型。然后依次介绍系统的各个重要组成部分：Hadoop 的 Fuse-DFS 子项目，著名的视频处理库 OpenCV 和 FFMPEG 以及谷歌代码(GoogleCode)旗下的 JavaCV 项目。最后实现系统的整体框架——面向大规模视频处理的 Hadoop 云系统模型，并说明系统的实际部署方法。

2.1 Hadoop 并行计算模型

首先介绍 Hadoop 并行运算模型，如图 2.1 所示。HDFS 是 Hadoop 分布式存储

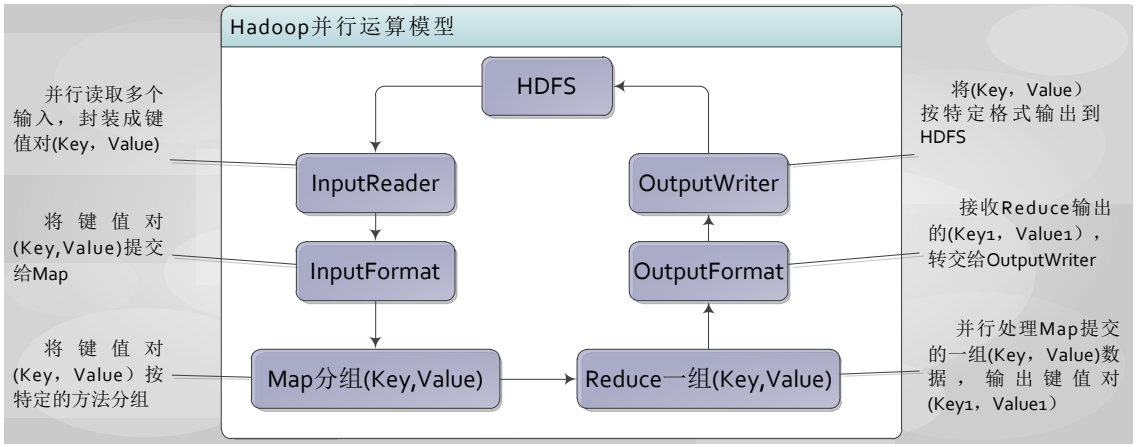


图 2.1 Hadoop 并行计算模型

系统。InputReader 读取 HDFS 中的数据，封装成键值对(Key, Value)后提交给 InputFormat。一个 InputFormat 可以接收多个不同类型的 InputReader 提交的数据，处理之后以键值对(Key, Value)的形式提交给 Map 处理。Map 根据算法需求将键值对(Key, Value)分组之后提交给 Reduce。每个 Reduce 实例接收到一组键值对(Key, Value)，并行地对它们进行处理，结果以键值对(Key1,Value1)的形式提交给 OutputFormat。OutputFormat 调用 OutputWriter 以指定的方式输出结果到 HDFS。

但是，如果将上述模型直接用于视频处理，将难以利用 OpenCV、FFMPEG 等现有的视频处理库，现有的算法便很难用在 Hadoop 中。这个问题的症结在于现有的视频处理库输入环节的封装性较高，可以读取本地文件系统但是不能读取 HDFS 分布式文件系统。因此需要在 HDFS 和本地文件系统之间提供一个接口。这个接口就是 Hadoop 的子项目 Fuse-DFS。

2.2 Fuse-DFS 子项目

Fuse-DFS 是 Hadoop 项目下的一个子项目，它的目的是为 Hadoop 的 HDFS 分布式文件系统提供到本地文件系统的接口，从而使得 Hadoop 能够利用众多专门为本地文件系统设计的库和接口，极大地提升开发效率。在本课题中，Fuse_DFS 子项目可以将存储在 HDFS 分布式文件系统挂载到 Linux 本地文件系统，从而使得利用 OpenCV、FFMEPG 等成熟的视频处理库变得可能。这里我们只是将它作为一个工具使用，不详细介绍其原理和实现。有关此项目的详细情况，可以参看参考文献[13]。

有了 Fuse–DFS 子项目，解决了文件系统接口问题。但是还有另外一个编程接口问题，这便是下一小结要解决的问题。

2.3 OpenCV、FFMPEG 和 JavaCV

在视频处理领域，OpenCV、FFMPEG 两大视频处理库是最成熟、使用频率最高的。众多的算法、软件是在它们的基础上实现的。如果不能利用它们，那么视频处理几乎只是原理，而难以实现。

这里的编程接口问题就在于 OpenCV、FFMPEG 等视频处理库均是用 C/C++语言编写，而 Hadoop 本身是 Java 语言编写。如果要利用 OpenCV、FFMPEG，需要有到 Java 的编程接口。幸运的是，GoogleCode 的 JavaCV 项目就提供了这样一个接口。（Hadoop 的 Streaming 是另一个利用 C++语言的选项，但没有 Java 易用。）

JavaCV 是 GoogleCode 旗下的一个开源项目，它是第一个向包括 Android 在内的 Linux 内核系统提供视频处理库 Java 接口的开源项目，提供的视频处理库接口包括：OpenCV、FFMEPG、lib1394、PRG、FlyCapture、OpenKinect、VideoInput、

ARToolKitPlus，并支持硬件加速处理和显示。但此项目还需要提升稳定性，另外相关文档和教程还比较匮乏。同样这里我们只是将它作为一个工具使用，不详细介绍其原理和实现。有关此项目的详细情况，可以参看参考文献[14]。

2.4 系统模型

有了 2.1 节 Hadoop 并行计算模型和 2.2、2.3 节介绍的众多项目的支持，建立面向大规模视频处理的 Hadoop 云系统有了现实可能。图 2.2 展示了系统的粗略模型。

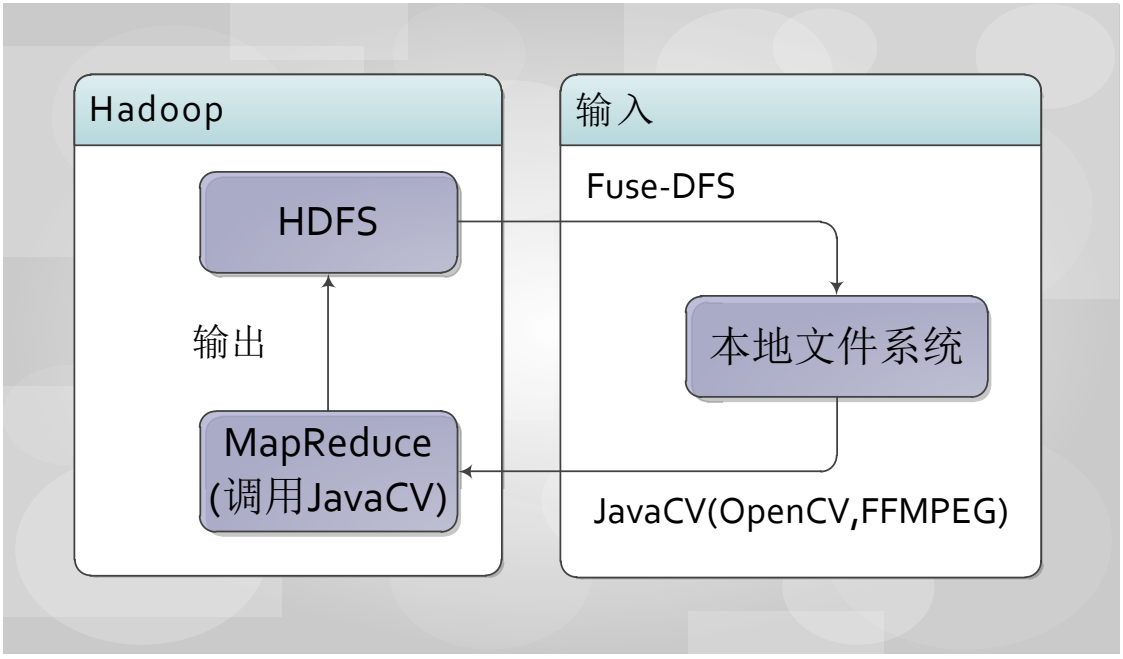


图 2.2 面向大规模视频处理的 Hadoop 系统粗略模型

系统的构建思路可以概括为：

1. 以 Hadoop 分布式文件系统（HDFS）存储海量视频；
2. 以 Fuse-DFS 实现从 HDFS 到本地文件系统的接口；
3. 以 JavaCV 调用 OpenCV，FFMEPG 视频处理库完成视频数据从本地文件系统的输入和在 Hadoop 集群上的处理；
4. 以 Hadoop 的 MapReduce 并行计算模型结合 JavaCV 实现对视频数据的并行分析处理。

将系统的 MapReduce 并行计算模型展开，我们可以得到视频帧数据在系统中从输入到处理再到输出的具体流程模型。图 2.3 展示了这一模型：

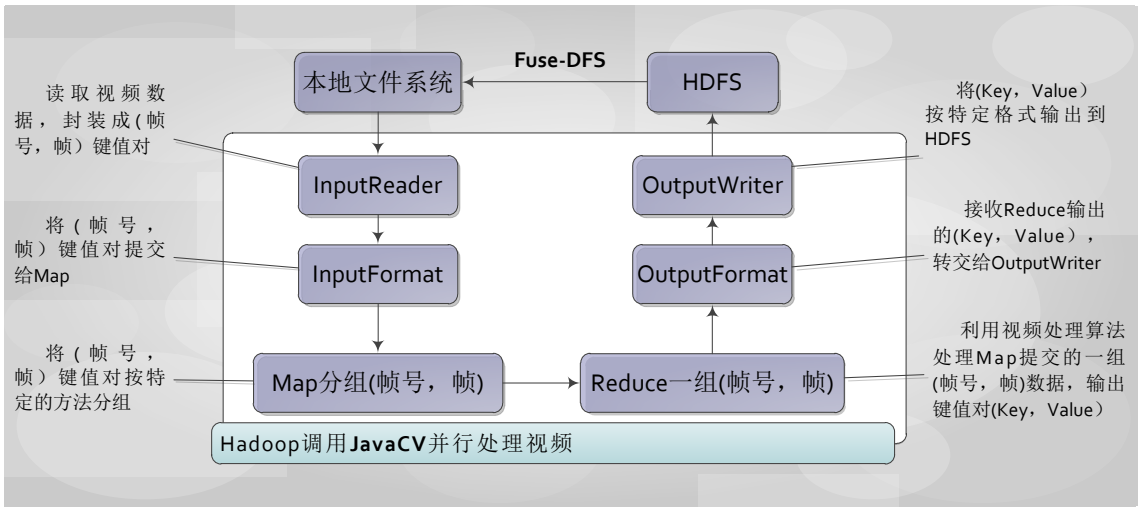


图 2.3 面向大规模视频处理的 Hadoop 系统模型

这一流程模型可以概括为：

1. HDFS 存储海量视频数据；
2. Fuse-DFS 将 HDFS 挂载到本地文件系统；
3. InputReader 调用 JavaCV 从挂载到本地的 HDFS 读取视频数据，封装成(帧号，帧）键值对，提交给 InputFormat。
4. InputFormat 多个 InputReader 提交的(帧号，帧）键值对数据提交给 Map 函数。
5. Map 函数将众多(帧号，帧）键值对按算法需求分组，并提交给 Reduce 函数。
6. Reduce 函数根据特定算法调用 JavaCV 处理一组(帧号，帧）键值对数据，将输出提交给 OutputFormat。
7. OutputFormat 将输出任务划分到多个 OutputWriter 上。
8. OutputWriter 执行输出任务：将输出写入 HDFS。

至此，面向大规模视频处理的 Hadoop 系统模型已经介绍完毕。

2.5 系统部署

系统部署的主要工作包括硬件和软件环境的建立和部署。

2.5.1 硬件环境

系统的硬件需求为：若干台计算机，路由器，网线。备齐了硬件设备后，需要组建局域网。硬件上，用网线将各台计算机分别连接到路由器上并开机即可。软件上，还需要在各台计算机的操作系统上配置好 IPv4 地址、子网掩码、默认网关、主机名等，并调整好防火墙设置，才能组成一个可以相互 ping 通的局域网。

具体硬件连接方法这里不再赘述，请读者自行查阅相关教程。

2.5.2 软件环境

对于每一台计算机节点，我们需要安装麒麟操作系统（或者其他 Linux 操作系统），SUN JDK 6（或者更新版本），Hadoop 环境，OpenCV、FFMPEG 和与之兼容的 JavaCV，以及 Fuse，Fuse-DFS。下面将细说各个步骤。

1. **安装麒麟操作系统。**将 NeoKylinV3.2.1 光盘放入光驱，启动计算机，从光盘启动（启动时按 Esc 或 F2 或 F10 或 F12 进行引导选择或者进入 BIOS 设置，根据机器不同而不同）。出现麒麟安装向导程序界面时，按提示操作：设置 root 用户密码、进行磁盘分区格式化后，选择安装模式为“编程开发”（或者包含此模式的“全部”模式），开始安装。等待大约 40 到 60 分钟系统安装完成并自动弹出光盘、重启。
2. **安装 ssh**。ssh 是 Linux 上的系统远程登录工具。对于麒麟操作系统或 Redhat 这样的服务器操作系统，ssh 是自带的。对于其他 Linux 系统，如 Ubuntu，可能并不自带 ssh。这时，需要到 Google 中搜索“zlib”，“openssl”，“openssh”三个源码包，分别依次解压、编译(命令为：configure&make)、安装(命令为：make install)。具体方法请参看源码包中的“readme”文件或附录 1.1。
3. **配置 ssh 无密码相互登录。**请参看附录 1.1。
4. **安装配置 Hadoop。**请参看附录 1.2。
5. **安装 Fuse、Fuse-DFS。**值得注意的是，在编译 Apache 官网发布的 Fuse-DFS

源码包时出现了编译错误，需要手动修正多处源码才能编译安装。具体请参看附录 1.3。

6. **安装 OpenCV、FFMPEG、JavaCV。**参看附录 1.4。
7. **安装 Eclipse 和 Hadoop 插件。**对于需要进行 Hadoop 编程调试的节点安装即可。步骤参看附录 1.5。

2.6 本章小结

本章介绍了面向大规模视频处理的 Hadoop 系统模型。系统的基础是 Hadoop 的 MapReduce 并行计算模型，综合集成 Fuse-DFS、OpenCV、FFMPEG、JavaCV 四个开源项目，使系统具备了对大规模视频数据进行并行分析的能力，为后续的视频分析算法移植奠定了基础。

第三章 Hadoop 云系统视频分析技术

Hadoop 云系统视频分析技术的核心是移植现有的视频分析算法和技术，使之符合 Hadoop 的 MapReduce 并行计算模型。在尽量不降低运行效果的条件下利用 Hadoop 云系统大幅度地提高视频分析速度，从而使得大规模视频数据的分析变得可能。

3.1 视频文件的存储与分段

3.1.1 常见的存储策略

本文利用 HDFS 存储大规模视频数据，但是对于具体应用，相适应的存储策略可以提高运行效率。下面讨论两种常见的情形：

1. **输入视频文件的大小基本相同。**这种情况无需对视频做分段，节点的 map 负载已经接近均衡。
2. **输入视频文件的大小存在显著差异。**有的输入文件只有数十 MB，有的却有几 GB。这种情况下，如果不进行预处理分段，会使单个节点 map 负载过大而导致运行时间较长。

3.1.2 视频独立性分割

视频文件的帧与帧之间由于压缩算法而存在关联性，要使得分段独立需采用 FFMPEG 或者 mencoder 等软件分割出独立的视频段。

采用 ffmpeg 分割视频的命令示例如下：

```
ffmpeg -ss 0:0:30 -t 0:1:00 -i input.avi -vcodec copy -acodec copy output.avi
```

其中“-ss”之后的参数为起始时间，“-t”之后的参数为截图视频的持续时间，“-i”之后的参数是输入文件路径，“output.avi”是输出路径。采用 ffmpeg 在分割视频文件速率远远快于逐帧读取再压缩成新视频，分割视频占用时间的很少，因此能加速 Hadoop 程序的执行。

3.2 实时视频流输入

在分析了两种算法移植方法后，再回到更复杂的输入方式上。在实际应用中，输入数据往往以视频流的方式输入，而 Hadoop 程序处理的输入是存储在 HDFS 中的文件，这里同样存在一个接口问题。对于实时视频流输入，可以采取以下方式处理：

1. 将实时接收的视频流转储为一系列固定大小的视频文件。
2. 视频文件在命名上区分“正在转储”、“转储完毕”和“分析完毕”三种状态。
3. 在 Hadoop 视频处理程序总控程序中加入一个 while(true)循环，一次 MapReduce 运行完毕后立即启动下一次运行，并只处理处于“转储完毕”状态的视频，处理完成后标记为“分析完毕”。

从上述过程可以看出 Hadoop 系统在虽然能够接受实时视频流输入，却难以做到“实时响应”，因为不论是视频转储、还是 MapReduce 任务的建立，都有延迟。

3.3 面向单帧图像的 Hadoop 云系统视频分析技术

Hadoop 用于视频分析的巨大优势在于缩短视频处理时间。本节将以经典人脸检测算法为实例，展示如何将基于单帧图像的视频分析算法实现在 Hadoop 云系统上。在第四章中，将通过实验验证 Hadoop 集群对缩短分析时间的显著效果。

基于单帧图像的视频分析算法的特点在于分析逻辑只考虑当前帧，所做的工作是针对每一帧的简单重复。这一类型的常见应用有：目标检测与识别，视频转码等等。此类算法在实现上帧与帧之间相互独立，没有耦合性，故逻辑上易于移植到 Hadoop 的 MapReduce 计算模型上。

3.3.1 基于 Haar 特征的单帧图像人脸检测算法

由于本文重点在于研究 Hadoop 平台对视频处理算法性能的改进而不在人脸检测算法上，这里只是简单介绍一下面向单帧视频（或者说单幅图像）的人脸检测算法中的 Haar 特征识别方法。

Haar 特征（Haar-like features）是用于物体识别的一种数字图像特征。它们因

为与 Haar 小波转换极为相似而得名，是第一种实时的人脸检测算子。在维奥拉-琼斯目标检测框架的检测阶段，一个与目标物体同样尺寸的检测窗口将在输入图像上滑动，在图像的每一个子区域都计算一个 Haar 特征。然后这个差值会与一个预先计算好的阈值进行比较，将目标和非目标区分开来。这样的一个 Haar 特征是一个弱分类器，为了达到一个可信的判断，就需要多个这样的特征。维奥拉-琼斯目标检测框架会将这些 Haar 特征组合成一个**级联分类器**，最终形成一个强分类群。[15]

OpenCV 库已经实现了维奥拉-琼斯目标检测框架并制作好了人脸 Haar 特征数据文件，因而可以直接利用 OpenCV 的提供的算法来实现人脸检测。对于 Hadoop 平台来说，需要使用 JavaCV 实现。

3.3.2 人脸检测算法在 Hadoop 云系统上的实现

首先是利用 JavaCV 将 OpenCV 的 Haar 特征人脸检测算法编写为一个静态方法，以便在 MapReduce 方法中直接调用。

```
//加载一帧图像
        String OUT_FILE = "out.jpg";
        IplImage origImg = cvLoadImage("inputImg.png", 1);
//获取灰度图像
        IplImage grayImg = IplImage.create(origImg.width(), origImg.height(),
IPL_DEPTH_8U, 1);
        cvCvtColor(origImg,grayImg,CV_BGR2GRAY);
//缩放灰度图像
        IplImage smallImg = IplImage.create((int) (grayImg.width() / SCALE), (int)
(grayImg.height() / SCALE), IPL_DEPTH_8U, 1);
        cvResize(grayImg, smallImg, CV_INTER_LINEAR);
//直方图均化灰度图像
        IplImage equImg = IplImage.create(smallImg.width(), smallImg.height(),
IPL_DEPTH_8U, 1);
        cvEqualizeHist(smallImg, equImg);
//创建 cvHaarDetectObjects 所需的临时存储空间
        CvMemStorage storage = CvMemStorage.create();
//加载 Haar 特征数据集
        String CASCADE_FILE = HAAR_DIR +
"haarcascade_frontalface_alt2.xml";
        CvHaarClassifierCascade cascade = new
CvHaarClassifierCascade(cvLoad(CASCADE_FILE));
//进行人脸检测
        CvSeq faces = cvHaarDetectObjects(equImg, cascade, storage, 1.1, 1,
CV_HAAR_DO_CANNY_PRUNING);
……
```

代码 3-1 JavaCV 人脸检测核心代码

其次是分别继承 Hadoop 的 FileInputFormat<Text, BytesWritable>类和 RecordReader<Text, BytesWritable>类编写 VideoInputFormat 和 VideoRecordReade 两个类(参看代码 3-3，代码 3-2)，实现利用 JavaCV 从本地文件系统读取视频并拆解为帧数据，再转化为 Hadoop 的 BytesWritable 数据类型，并以“文件名+帧号”做键、帧内容做值组成键值对，提交给 Map 函数处理。

代码 3-2 具体展示了如何将 JavaCV 获取的 IplImage 数据类型转化为 Hadoop 的 BytesWritable 数据类型。

```
public class VideoRecordReader implements RecordReader<Text, BytesWritable> {
    byte[] buf;
     FileInputStream fin;
     String localPath;
     public static final String exportPath = "/export/hdfs";
     ……
     private FFmpegFrameGrabber grabber;
     private int currentFrameNumber;
     private int totalFrameNumber;
     private IplImage frame;
     private java.nio.ByteBuffer byteBuf;
     private BufferedImage bufImage;
     private ByteArrayOutputStream baos;
     ……
    @Override
//重载 next 方法读取视频帧数据，生成键值对；
//其中键为“文件名+帧号”，值为视频帧数据
    public boolean next(Text key, BytesWritable value) throws IOException {
        try {
            while ((frame = grabber.grab()) != null) {
                ++currentFrameNumber;
                key.set(keyName.concat("." + currentFrameNumber));
                bufImage = frame.getBufferedImage();
                baos = new ByteArrayOutputStream();
                baos.close();
                value.set(new BytesWritable(baos.toByteArray()));
                return true;
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return false;
        }
        return false;
    }
}
```

代码 3-2 VideoRecordReader 核心代码

代码 3-3 展示了如何利用 VideoRecordReader 类自定义 FileInputFormat 类，实现 VideoInputFormat。关键之处在于重载 getRecordReader 方法。

```
public class VideoInputFormat extends FileInputFormat<Text, BytesWritable> {
    @Override//是否对文件进行切分? return TRUE  是，  FALSE  否
    protected boolean isSplitable(FileSystem fs, Path path) {
        return false;   //这里我们一律不对文件进行切分
    }
    @Override
    public RecordReader<Text, BytesWritable> getRecordReader(InputSplit split,
JobConf job,Reporter reporter) throws IOException {
        // TODO Auto-generated method stub
        try {
            return new VideoRecordReader(job, FileSplit.class.cast(split));
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return null;
        }
    }
}
```

代码 3-3 VideoInputFormat 核心代码

代码 3-4 展示了如何实现 Map 方法。

```
public static class VideoMapper extends MapReduceBase implements
            Mapper<Text, BytesWritable, Text, BytesWritable> {
        private IplImage frame;
        private ByteBuffer byteBuf;
        @Override
//这里 map 不需要分组数据，可以直接将键值对提交给 Reduce 处理
        public void map(Text key, BytesWritable value,
                OutputCollector<Text, BytesWritable> collector,
                Reporter reporter) throws IOException {
            // TODO Auto-generated method stub
            collector.collect(key, value);
        }
    }
```

代码 3-4 Hadoop 的 Mapper 核心代码

由于是单帧处理，负责将视频分组的 Map 函数其实并没有实际任务。这里 Map 函数直接将收到的键值对提交给 Reduce 函数(参看代码 3-5)处理。Reduce 函数将键值对中的字节视频数据还原为 JavaCV 的 IplImage 数据类型，并调用人脸检测的静态方法处理，检测结果以“键+文本”形式提交输出。

```
    public static class VideoReducer extends MapReduceBase implements
            Reducer<Text, BytesWritable, Text, Text> {
        private IplImage frame;
        private ByteBuffer byteBuf;
        private byte[] buf;
        @Override
        public void reduce(Text key, Iterator<BytesWritable> values,
                OutputCollector<Text, Text> collector, Reporter reporter)
                throws IOException {
            while (values.hasNext()) {
//将视频帧从 BytesWritable 类型还原为 IplImage  类型
                buf = values.next().getBytes();
                ByteArrayInputStream bais = new ByteArrayInputStream(buf);
                BufferedImage bufImage = ImageIO.read(bais);
                bais.close();
                frame = IplImage.createFrom(bufImage);
                IplImage image = cvCreateImage(
                        cvSize(frame.width(), frame.height()), IPL_DEPTH_8U, 3);
//调用 FaceDetection 类中的 detectFaces 静态方法检测人脸
                int faceNumber = FaceDetection.detectFaces(frame, image);
//提交检测结果
                collector.collect(key,
                        new Text("reducing frame: " + frame.width() + " * "
                                + frame.height() + " FaceNumber = "
                                + faceNumber));
                cvReleaseImage(image);
            }
        }
    }
```

代码 3-5 Hadoop 的 Reducer 核心代码

值得注意的是，从性能上说，针对本例的特殊情况，在 Map 中直接进行人脸检测将结果而省略 Reduce 可能更高效（因为节约了从 Map 到 Reduce 的视频数据传输带宽）。但它违背了 MapReduce 计算模型的设计意图，而且 Map 任务的数量不能直接控制，不利于充分发挥集群性能，所以本例中没有采用。

3.4 面向图像序列的 Hadoop 云平台视频分析技术

本节将以基于统计背景模型的运动目标检测算法为实例，展示如何将基于图像序列的视频分析算法实现在 Hadoop 云系统上。

基于图像序列的视频分析算法的特点在于分析逻辑需考虑连续多帧图像，综

合分析才能得出结果。而其结果往往对后续工作也有影响。这一类型的常见应用有：运动目标检测与跟踪，行为识别等等。此类算法在实现上帧与帧之间相互关联，具有耦合性，故在移植到 Hadoop 的 MapReduce 计算模型上时，需要对视频图像序列进行合理的、冗余的分割。[16,17]

3.4.1 基于统计背景模型的运动目标检测与跟踪算法

此算法是计算机视觉领域的一个经典算法，其算法流程如图 3.1 所示。

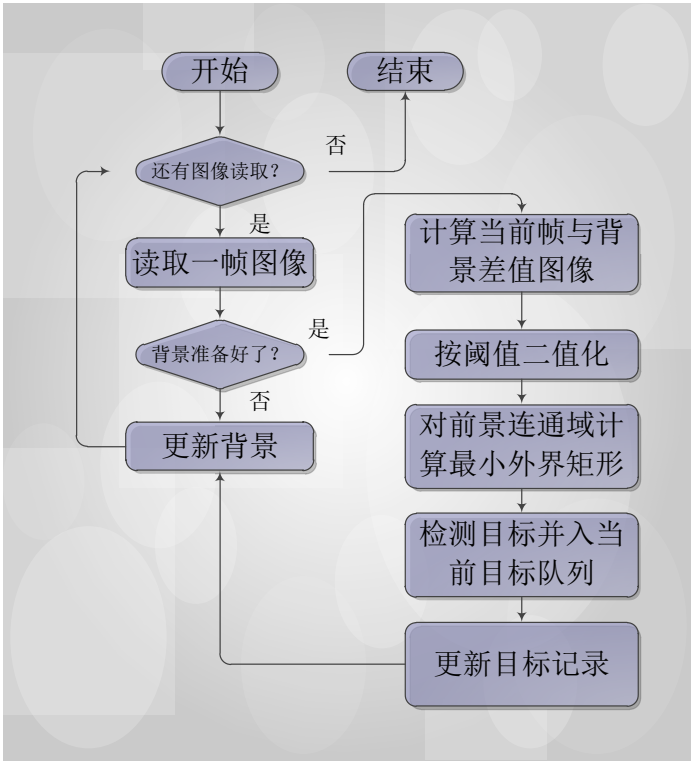


图 3.1 基于统计背景模型的运动目标检测与跟踪算法流程图

下面对于算法中的关键步骤进行说明：

1. **更新背景。**在没有背景时将读取到的若干帧图像的平均值作为背景。在有背景时，将当前帧按比例与背景帧混合得到新的背景。
2. **计算当前帧与背景帧差值图像。**这里计算的是对应像素的亮度差值。
3. **按阈值二值化差值图像。**对于上一步得到的差值图像，设定一个阈值，以分为前景和背景。
4. **对前景图像连通域计算最小外接矩形。**

5. **检测目标并入当前目标队列。**算法维护一个“当前目标队列”。这个队列保存还出现在当前帧中的运动目标。而合并的方法有很多种，常见的是用 Kalman 滤波器预测位置与当前位置比对。而这里出于简化，将目标上一位置与当前帧目标位置的公共面积占目标上一位置面积的比例作为合并依据。
6. **更新目标记录。**对于“当前目标队列”中一定帧数（例如 15 帧）之内都没有再次出现的目标，认为已经离开画面，生成记录；同时将该目标踢出“当前目标队列”。

值得注意的是，运动目标检测与跟踪算法有很多种版本。更复杂的、效果更好的算法包括带有 Kalman 预测的跟踪，基于特征点骨架的跟踪等等。由于本文关注的 Hadoop 平台对算法性能的改进，为了简单起见，不再探讨更复杂的算法。

3.4.2 运动目标检测与跟踪算法在 Hadoop 云系统上的实现

在 Hadoop 云系统上设计运动目标检测与跟踪算法，不同之处在于需要将算法处理的视频分段，然后并行处理，最后再将每段的检测跟踪结果合并。为了提高合并时的准确度，分段应当是有冗余的。具体做法如下：

1. Mapper 将同一个视频的每 300 帧分为一组，前后相邻两组间各保留 10 帧重复帧。即分组为：1-300 帧，290-600 帧，590-900 帧……以此类推。
2. Reducer 采用图 3.1 所示的流程对每一组视频数据进行运动目标检测与跟踪，并生成文本记录。
3. 对相邻组的文本记录，根据对象出现消失的帧号、位置信息，进行对象匹配与合并，得到最终的检测结果。

代码 3-6 展示了 Mapper 分组视频帧的过程。首先从 VideoInputFormat 提交的<Text, BytesWritable>键值对中提取出视频帧号和帧数据；其次将帧号和视频帧数据重新封装为 BytesWritable 数据类型（代码中通过自己编写的 EncodeFrame 类的 encodeFrame 静态方法实现）；最后按照帧号将封装有帧号的视频帧数据分组。

```
@Override
public void map(Text key, BytesWritable frame,
                    OutputCollector<Text, BytesWritable> collector,
                    Reporter reporter) throws IOException {
    //获取视频帧号和帧数据
    String tmp = key.toString();
    int ind = tmp.lastIndexOf('.');
    String videoFilePath = tmp.substring(0, ind + 1);
    long frameNumber = Long.valueOf(tmp.substring(ind + 1));
   //将视频帧号与帧数据重新封装成 BytesWritable 类型
    BytesWritable encodedFrame = EncodedFrame.encodeFrame(frameNumber,
frame);
    //计算组号
     long groupId = (frameNumber - 1) /
     CommonVariables.FRAME_GROUP_LENGTH;
    collector.collect(new Text(videoFilePath + groupId), encodedFrame);
    //如果是重复帧也压入下一组
    long groupId2 = (frameNumber + CommonVariables.FRAME_GROUP_INTER) /
CommonVariables.FRAME_GROUP_LENGTH;
    if (groupId2 > groupId)
        collector.collect(new Text(videoFilePath + groupId2), encodedFrame);
}
```

代码 3-6 运动目标检测与跟踪 Map 方法

在代码 3-6 中，CommonVariables.FRAME_GROUP_LENGTH 为 100，而 CommonVariables.FRAME_GROUP_INTER 为 10。如此实现了原视频每 300 帧（加上组间重复帧或为 310 帧）分成一组，每组与前一组有 10 帧重复。

Map 将分组好的数据提交给 Reduce 处理。对于每一组数据，Reduce 首先解压出视频帧号和帧数据，并按照帧号从小大到对帧数据排序；其次创建一个 MotionDetector，将排序后的帧数据提交给 MotionDetector 的 analyzeFrame 方法执行运动目标检测与跟踪算法。而 MotionDetector 实现了图 3.1 描述的算法流程。

```
@Override
public void reduce(Text key, Iterator<BytesWritable> values,
                   OutputCollector<Text, Text> collector, Reporter reporter)
                   throws IOException {

    //提取本组的帧
    ArrayList<EncodedFrame> frameArray = new ArrayList<EncodedFrame>();
    while (values.hasNext()) {
        EncodedFrame eFrame = EncodedFrame.decodeFrame(values.next());
        frameArray.add(eFrame);
    }
    //按帧号排序
    EncodedFrame[] eFrames =    new EncodedFrame[frameArray.size()];
    eFrames = frameArray.toArray(eFrames);
    Arrays.sort(eFrames, new Comparator<EncodedFrame>() {
        @Override
        public int compare(EncodedFrame a, EncodedFrame b) {
            // TODO Auto-generated method stub
            return (int)(a.frameNumber - b.frameNumber);
        }
    });

    // 创建 MotionDetector 并进行检测
    MotionDetector motionDetector = new MotionDetector();
    for(int i = 0; i < eFrames.length; ++i) {

motionDetector.analyzeFrame(CommonOperations.parseFrame(eFrames[i].frame),
eFrames[i].frameNumber);
    }
    motionDetector.finishAnalysis();
    // 显示结果
    ……
    collector.collect(key, new Text(result));
}
```

代码 3-7 运动目标检测与跟踪算法的 Reduce 方法

Reduce 之后算法并没有结束，对相邻组的文本记录，还要根据对象出现消失的帧号、位置信息，进行对象匹配与合并，得到最终的检测结果。

3.5 本章小结

Hadoop 云系统视频分析技术的核心是改造现有的视频分析算法和技术，使之符合 Hadoop 的 MapReduce 并行计算模型。本章介绍了如何用 ffmpeg 等工具将大视频分段存储，以及如何处理实时视频流输入，详细阐述了如何用 JavaCV 在

Hadoop 云系统上实现面向图像的、面向图像序列的两种视频分析算法，力求在尽量不降低运行效果的条件下利用 Hadoop 云系统大幅度地提高视频分析速度。

第四章 性能实验和 Hadoop 云系统运行时间模型

4.1 性能实验

4.1.1 实验环境

实验采用六台台式算机，其中一台作为主控节点(Namenode 兼 JobTracker)，其余五台作为从节点(DataNode 和 TaskTracker)。每台机器的性能如表 4-1。

表 4-1 节点机器硬件性能参数

主要组件	性能参数和品牌
CPU	主频 1.80GHz，Intel 奔腾双核处理器 E2160
主板	华硕 P5VD2-VM SE(威盛 P4M900/VN896/CN896)
内存	容量 1GB，金士顿 DDR2(667MHz)内存（原本为 1G 内存条，有 128M 分配为显存）
硬盘	容量 160GB，IDE 接口，迈拓 STM3160215AS 硬盘

集群环境保证每台从节点是同构的，即目录结构和软件环境完全相同。具体软件配置见表 4-2。

表 4-2 节点机器软件配置

主要组件	配置或版本
操作系统	中标麒麟操作系统 v3.2.1（一款红帽系列的 32 位 Linux 操作系统，由中标软件有限公司和国防科大计算机学院联合研发）
JDK	SUN JDK 1.7 Update 13
Hadoop	Apache Hadoop 1.0.4 稳定发行版(stable release)
FFMEPG	ffmpeg 0.6.1
OpenCV	OpenCV 2.4.0
JavaCV	OpenCV 2.4.0 配套版本
Fuse	Fuse 2.8.5
Fuse-DFS	Fuse-DFS 2005 年版（截至 2013 年 03 月的最新版本）

4.1.2 实验方案

实验过程主要分为三大步骤：

1. 部署系统：包括集群节点安装麒麟操作系统，JDK，Hadoop 环境，OpenCV，FFMPEG，Fuse，Fuse-DFS。请参考 2.5 节。
2. 编写、调试 Hadoop 单帧视频人脸检测程序。请参考 3.2 节。
3. 准备输入视频数据，在不同节点数，不同输入视频数目的情况下分别测试 Hadoop 单帧视频人脸检测程序和运动目标检测跟踪程序的运行时间并分析。将测试程序打包为 TestHadoopVideo.jar 包（大小：2.5MB），并将所依赖的 JavaCV 库放在 jar 包的 lib 目录下。通过终端运行命令

   hadoop –jar TestHadoopVideo.jar hdfs://27.132.210.79:9000/data/video/input hdfs://27.132.210.79:9000/data/video/output

   将任务在集群上运行。其中最后两个参数分别为输入视频目录和输出目录在 HDFS 上的位置。而 IP“27.132.210.79”是主节点的 IP，“9000”是 HDFS 使用的端口号。

4.1.3 实验结果

在 3.3.1 节描述的实验环境下分别运行 3.2 节描述的人脸检测算法测试程序和运动目标检测跟踪程序，选取大小为 2.6MB、总帧数 690 帧（15fps×46s）、分辨率为 320×240 的一段 flv 格式视频作为**标准输入视频**，得出的实验结果如表 4-3、表 4-4 所示。

表 4-3 人脸检测程序在 Hadoop 云系统上的运行时间 (秒)

视频数 节点数	1	2	3	4	5
1	166.55	108.51	89.32	74.65	69.18
2	287.80	172.58	131.43	110.27	102.75
3	419.28	235.87	173.60	143.27	134.59
4	554.76	303.76	219.95	177.86	160.73
5	678.80	384.20	264.57	208.44	191.71
6	808.47	433.91	311.81	245.81	218.29

注：这里的“运行时间”是从启动 Hadoop 程序到运行结束的总运行时间。表中行号为 Hadoop 从节点数，列号为标准输入视频数。表中的数据为指定条件下五次运行程序时间的平均值。

表 4-4 运动目标检测与跟踪程序在 Hadoop 云系统上的运行时间 (秒)

视频数 节点数	1	2	3	4	5
1	48.16	46.56	45.16	45.76	46.87
2	53.99	49.97	49.49	48.29	49.88
3	71.68	55.20	54.59	48.10	50.11
4	82.31	57.83	55.88	49.15	55.27
5	101.57	71.53	61.36	54.66	54.56
6	111.31	75.87	64.50	58.12	60.09

注：这里的“运行时间”是从启动 Hadoop 程序到运行结束的总运行时间。表中行号为 Hadoop 从节点数，列号为标准输入视频数。表中的数据为指定条件下五次运行程序时间的平均值。

4.2 性能分析

4.2.1 人脸检测程序性能分析

首先对 4.1.3 节的人脸检测程序的实验结果进行可视化。

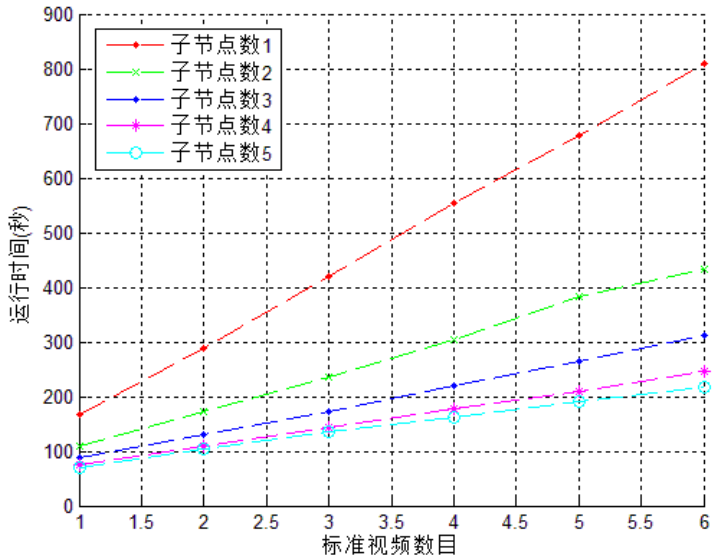


图 4.1 运行时间与输入视频数的关系

图 4.1 展示了测试程序运行时间岁输入的标准视频数据、集群子节点数目的变化情况。可以看出：

1. **对于相同集群，测试程序运行时间与输入标准视频数大致成线性关系。**综观五条曲线的趋势不难看出线性关系的存在。从理论上分析人脸检测算法的运行时间与输入帧数确实是线性关系，二者相互印证。
2. **对于相同数目的标准视频，测试程序运行时间岁集群子节点数增加而显著减少。**这说明在一定范围内扩大集群规模对于减少计算时间是有益的。

综合两条规律，可以推出：**输入视频数据量越大，大集群处理节约的时间越多。**

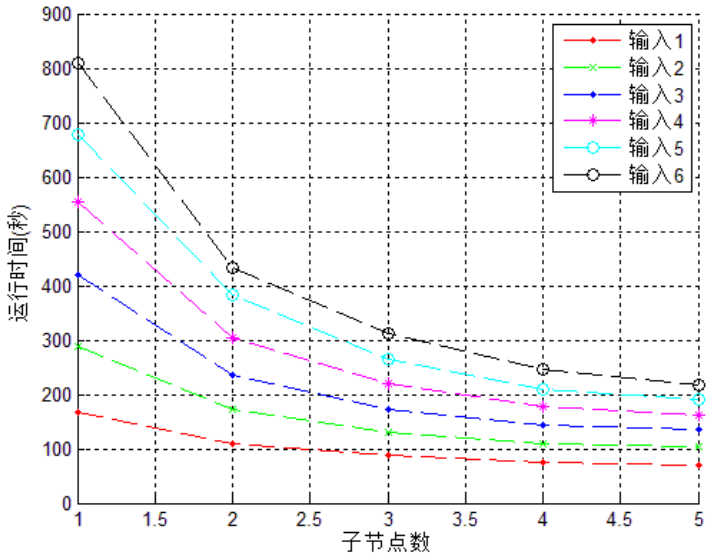


图 4.2 运行时间与集群子节点数的关系

图 4.2 展示了运行时间与 Hadoop 集群子节点数的关系。从图中可以发现：

1. **对于同一输入，运行时间随着子节点数的增加而减小，并呈现出类似反比例函数的特点**。这里的“负指数分布特点”包括：1）随着子节点数增加，运行时间变小；2）随着子节点数增加，运行时间减小的幅度变小。
2. **同一个集群，输入视频数据越小运行时间越少**。

这两条规律表明：增加节点总能提高系统性能，但增加节点的费效比在提高，应当综合效果与费用考虑来确定集群大小。

4.2.2 运动目标检测与跟踪程序性能分析

同样，先对实验结果进行可视化。

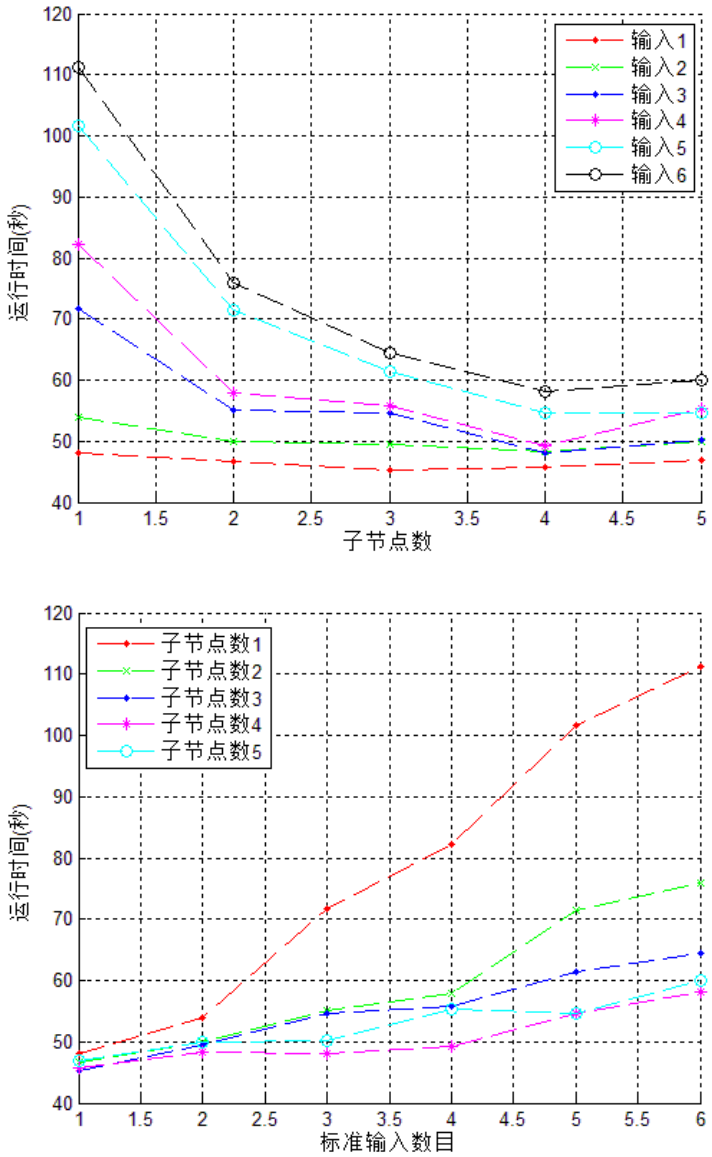


图 4.3 运行时间与标准输入数目、集群子节点数的关系

图 4.3 反应出运动目标检测与跟踪程序与人脸检测程序的性能有以下不同：

1. **总运行时间较少。**由于输入视频相同，这反映出运动目标检测与跟踪程序计算密集性小于人脸检测程序。计算时间在总运行时间中的比例不如前者。
2. **总体上运行时间随输入数据量增长，但增长幅度有明显波动。**这点与人脸检测程序明显的线性增长不同。
3. **运行时间与子节点数关系图出现极小值点。**总体上看，子节点数为 5 比子节点数为 4 时运行时间并没有明显减少。输入数目为 1、3、4、6 时，子节点数为 4 的运行时间比子节点数为 5 的运行时间还要小，成为“极小值点”。**这说明 4**

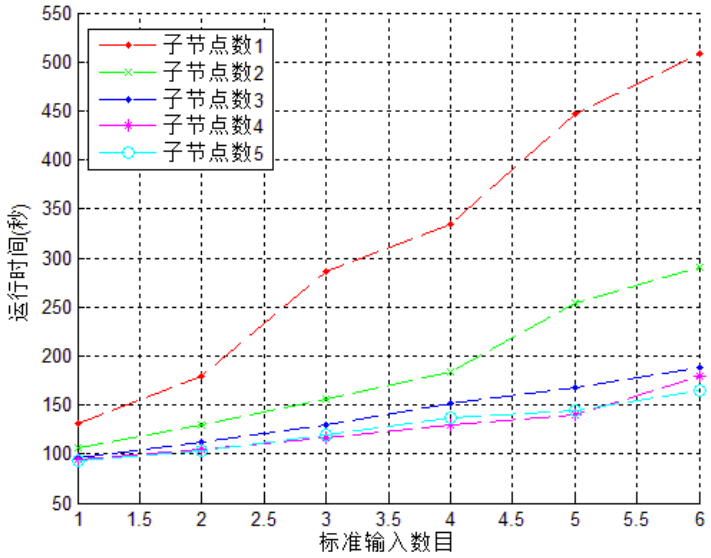
个子节点已经能满足此任务需要，再增加子节点数反而可能因为通信开销增加而增加运行时间。

从“运行时间与子节点数图”输入个数为 5，6 时规律性较好与输入个数小于 5 时规律性较差分析，推测运动目标检测程序表现出的不同特性是否是因为其输入数据太小，还没有体现 Hadoop 运算性能的规律性。为了验证这个推测是否正确，需要补充一个实验：增大单个输入视频文件到 20.9MB（原来为 2.6MB），重复运动目标检测与跟踪实验。

表 4-5 “大输入时”运动目标检测与跟踪程序在 Hadoop 云系统上的运行时间 (秒)

视频数 节点数	1	2	3	4	5
1	130.54	105.78	96.20	93.93	92.81
2	180.05	129.63	112.35	104.53	104.09
3	285.75	156.24	129.22	116.85	119.50
4	333.82	183.61	152.06	129.33	137.68
5	447.28	253.74	167.41	139.48	144.90
6	507.63	289.80	188.04	179.35	165.10

注：这里的“运行时间”是从启动 Hadoop 程序到运行结束的总运行时间。表中行号为 Hadoop 从节点数，列号为标准输入视频数。表中的数据为指定条件下五次运行程序时间的平均值。单个输入视频大小为 20.9MB。



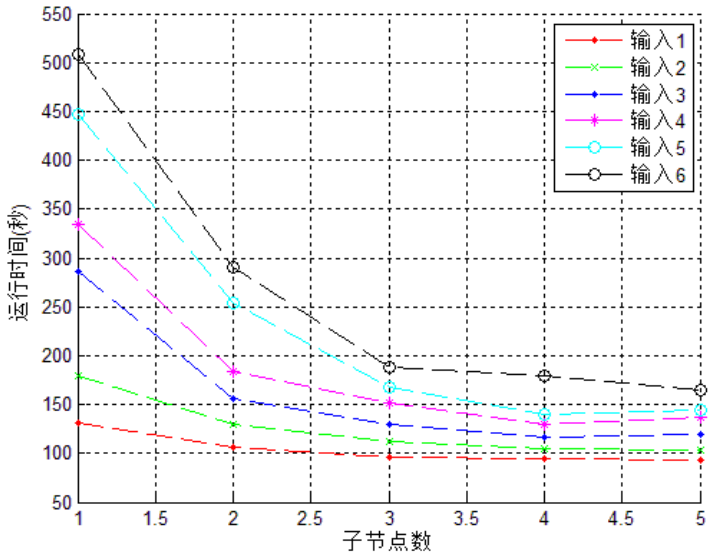


图 4.4“大输入时”运行时间与标准输入数目、集群子节点数的关系

可以看出，增大输入规模后，运动目标检测与跟踪程序的运行时间曲线与人脸检测程序运行时间曲线更为相似，表现出运行时间与输入的线性关系和运行时间与集群子节点数的反比关系的特点。

4.3 Hadoop 云系统运行时间模型

从 4.2 节的性能分析中不难看出 Hadoop 云系统的运行时间与输入数据大小和集群子节点数之间都存在明显的关联。本小结将尝试挖掘这种关联的本质，即用数学模型来描述运行时间与输入数据大小和集群子节点数的关系。

4.3.1 运行时间模型的建立

首先分析运行时间与输入数据的关系。对同一个集群，设运行时间 *runTime* 是关于变量输入标准视频数 *s* 的函数。由图 4.1 反应的线性关系，可得

$$runTime(s) = \beta s + \gamma \tag{4-1}$$

其中 β，γ 为工作任务相关待定常数。本文称之为运行时间与输入数据的“**线性关系**”。

然后分析运行时间与集群子节点数的关系。对于同一个工作任务，设运行时间 *runTime* 是关于集群子节点数 *n* 的函数。由图 4.2 反应的反比例关系，可得

$$runTime(n) = \alpha / n + \delta \tag{4-2}$$

其中α，δ为与集群计算能力相关待定常数。本文称之为运行时间与集群子节点数的“**反比例关系**”。

综合以上线性关系与反比例关系，可以得出集群运行时间与输入标准视频数s、集群子节点数n的函数关系，即**集群运行时间模型**：

$$runTime(n,s) = \alpha s / n + \beta \tag{4-3}$$

其中变量n，s分别为集群子节点数和输入数据大小；参数α，β为与集群计算能力和工作任务相关待定常数。不难看出，参数α表征任务的计算密集度，参数β是集群建立任务的常数时间。

4.3.2 运行时间模型的验证

使用人脸检测程序、运动目标检测程序和 Hadoop 示例程序 WordCount 做了三个实验来验证运行时间模型。

首先利用 4.1 节的实验数据来验证模型。对于 4.1 节描述的实验，利用表 4-3 的实验结果数据和公式(4-3)，并运用 Matlab 拟合工具得到参数估计值为$\alpha = 126.4$，$\beta = 50.67$。拟合结果如图 4.5。

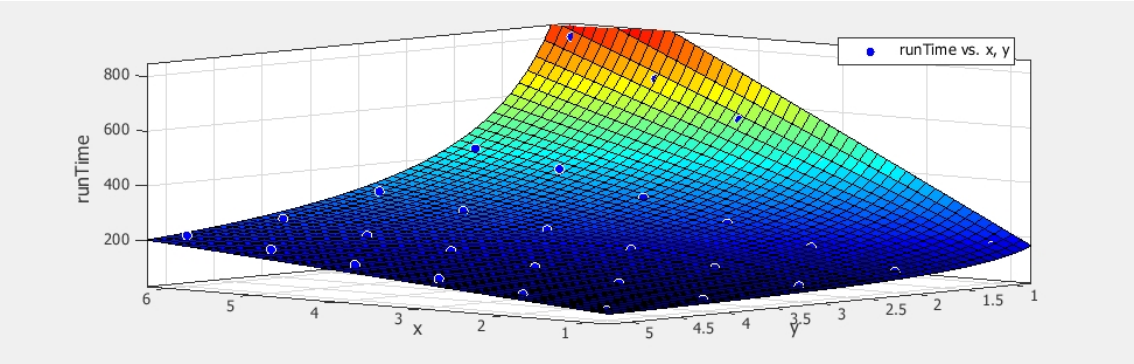


图 4.5 运行时间模型预测结果与实际运行时间比对效果

图注：图中 x 轴表示集群子节点数，y 轴表示输入标准视频数，runTime 轴（z 轴）表示运行时间。蓝色原点表示表 4-3 中的实际运行时间；曲面表示模型预测时间。可以看出模型的预测结果与实际数据几乎一致，预测效果很好。

由 Matlab 拟合工具同时得出了修正拟合系数$adj.R\text{-square} = 0.9980$，均方根$RMSE = 7.9368$秒。这表明模型的预测结果与实际数据几乎一致，预测效果非常好。

对于运动目标跟踪与检测程序，由表 4-5 中的实验数据和公式（4-3），可以得到拟合结果如图 4.6。

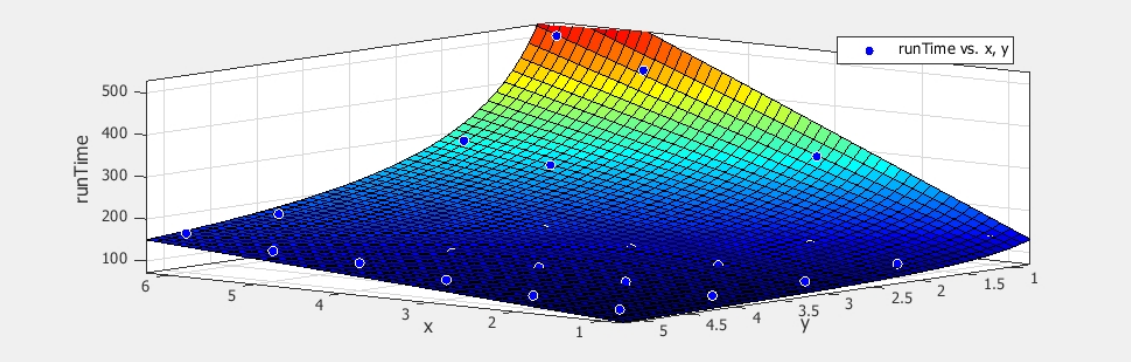


图 4.6 运行时间模型预测结果与实际运行时间比对效果

图注：图中 x 轴表示集群子节点数，y 轴表示输入标准视频数，runTime 轴（z 轴）表示运行时间。蓝色原点表示表 4-5 中的实际运行时间；曲面表示模型预测时间。模型参数：$\alpha = 72.06$，$\beta = 64.05$。修正拟合系数 $adj.R\text{-}\text{square} = 0.9822$，均方根 $RMSE = 13.79$ 秒。可以看出模型的预测结果与实际数据基本一致，预测效果较好。

其次，采用一个不同类型的程序来检验模型的适用性。利用 4.1 节描述的实验环境运行 Hadoop 的示例程序 WordCount。该程序的功能是统计所有输入文本文件中每个单词的出现次数。得到的运行时间数据如表 4-5 所示：

表 4-6 WordCount 程序在 Hadoop 云系统上的运行时间 (秒)

视频数 节点数	1	2	3	4	5
1	45.83	42.90	41.78	42.15	42.60
2	58.10	53.44	49.63	50.38	48.64
3	83.73	55.09	52.39	49.88	49.72
4	95.19	65.14	59.42	51.92	51.31
5	120.36	76.42	61.90	56.24	53.02
6	138.89	89.39	66.81	59.63	55.46

注：这里的“运行时间”是从启动 Hadoop 程序到运行结束的总运行时间。表中行号为 Hadoop 从节点数，列号为标准文本文件（每个大小为 10M）输入个数。表中的数据为指定条件下五次运行程序时间的平均值。

将表 4-6 中的数据可视化可得图 4.7：

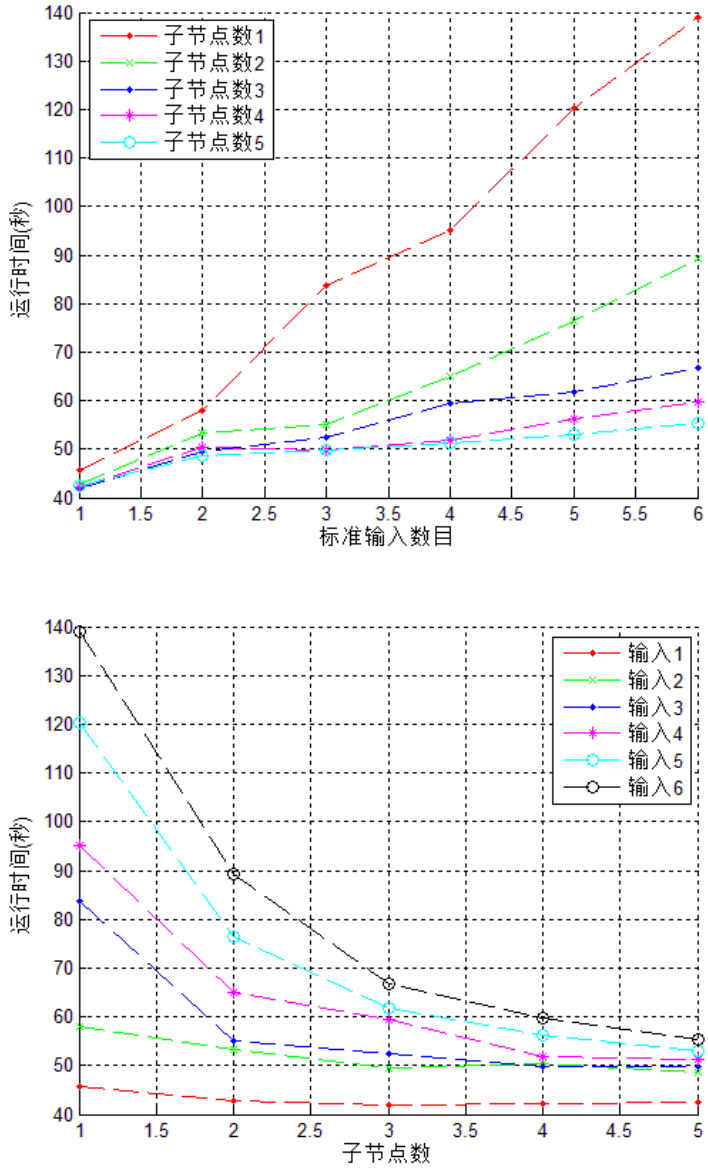


图 4.7 WordCount 程序运行时间与输入大小、集群子节点数关系

图 4.6 依然反映出线性关系和反比例关系两条规律，但是也存在与规律不一致的波动。这一方面是由于 WordCount 的实验运行时间较短。

同样利用 Matlab 拟合工具箱获取模型拟合结果。

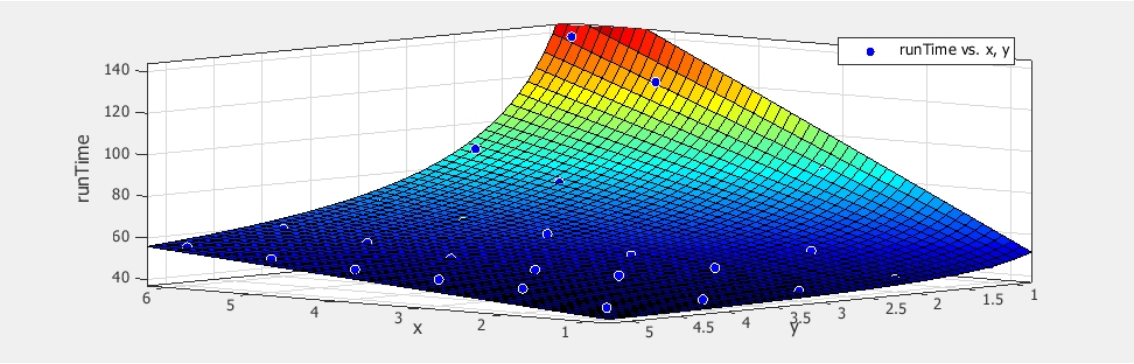


图 4.8 运行时间模型预测结果与 WordCount 实际运行时间比对效果

图注：图中 x 轴表示集群子节点数，y 轴表示输入标准视频数，runTime 轴（z 轴）表示运行时间。蓝色原点表示表 4-6 中的实际运行时间；曲面表示模型预测时间。模型参数为：$\alpha = 16.05$，$\beta = 36.52$。可以看出模型的预测结果与实际数据基本相符，预测效果较好。

由 Matlab 拟合工具得出的修正拟合系数 $adj.R\text{-square} = 0.9714$，均方根 $RMSE = 3.8543$ 秒。这表明模型的预测结果与实际数据基本一致，预测效果较好。

从以上三个实例对模型进行独立的验证，修正拟合系数均在 0.95 以上，说明了模型对于不同的类型的程序都具有较好的预测能力。

4.3.3 运行时间模型的优缺点

运行时间模型是从 Hadoop 人脸检测程序的运行实验数据中归纳出来的。在集群同构子节点不超过五个的条件下，通过了 Hadoop 示例程序 WordCount 的验证。

运行时间模型可被用于预测 Hadoop 任务时间和确定任务所需的集群大小。建立中型、大型集群，利用小集群实验或许性能数据并利用模型预测运行时间，可以科学确定集群规模，有效节约集群建立的时间和成本。其优点在于：

1. **简单、快速。**只需小集群、少量时间对某一任务采集运行数据，通过简单的数学计算就可以预测该任务在大集群上、面对大输入的运行时间。
2. **发现了线性关系、反比例关系。**这对于后续深入挖掘运行机理和适用范围更广的规律具有很大的启发意义。

然而，运行时间模型也存在的局限性：

1. **模型有效性没有经过中型（几十个节点）、大型集群（上百个节点）的验证。**本文没有条件建立中型或者大型集群去验证模型的有效性。

2. **模型没有考虑网络带宽的制约。**这意味着使用模型时应保证局域网带宽不会成为任务的瓶颈。
3. **模型仅适用于同构机器组成的集群。**如果集群中的节点计算性能不一，模型的预测结果可能有偏差。
4. **模型要求运行的算法与输入规模呈线性关系。**对于大多数视频处理算法，处理时间与输入视频帧数成线性关系是成立的。但是对于其他算法程序，这一点不一定成立。此时“线性关系”不再成立，模型不再适用。

4.3.4 模型的应用

应用模型进行运行时间预测，需要建立一个同构节点组成的小集群采集和小输入来采集数据，估计模型的三个参数。以 4.1 节描述的实验条件和 4.3.2 节估计的参数来预测，可以得到以下结果：

表 4-7 模型对人脸检测程序运行时间的预测

输入视频（1 个=2MB）	集群子节点数	预估运行时间
100MB=50 个	1	6370.7 秒=1.8 小时
100MB=50 个	10	682.67 秒=0.2 小时
100MB=50 个	100	113.87 秒=0.03 小时
1GB=500 个	1	63251 秒=17.6 小时
1GB=500 个	10	6370.7 秒=1.8 小时
1GB=500 个	100	682.64 秒=0.2 小时
1TB=5000 个	1	632050 秒=175.6 小时
1TB=5000 个	10	63251 秒=17.6 小时
1TB=5000 个	100	6370.7 秒=1.8 小时

表注：由于客观条件限制，本实验采用的机器性能远远落后于 2013 年的机器平均水平（详见表 4-1）。如果能购置 2013 年的台式电脑，其单机计算性能至少可以提升 4 倍（2013 年 GPU 为四核，主频 3.4GHz 的台式电脑十分常见），集群性能也至少可以提升 4 倍。

从表 4-7 的预测结果可以看出：

1. **集群具有明显的加速效果。**10 个子节点的集群可以将运行时间按压缩到单机运行的 12%以下。而这个集群规模普通监控视频用户（政府部门、企业）也是

可以负担的。多路监控视频的实时分析、预警的推广在技术上、经济上是现实可行的。

2. **集群的加速潜力极大。**100 个子节点的运行时间几乎是 1 个子节点的 1%。只要有足够的集群规模，理论上可以达到任何加速比。当然，实际上还会受到网络带宽的制约。

4.4 本章小结

本章详细阐述了人脸检测程序在六台同构机器组成的Hadoop云系统上的运行实验。本文将输入大小、集群子节点数作为变量收集运行时间数据，在充分分析实验数据的基础上，发现了两条规律：

1. 运行时间与输入大小之间的**线性关系**：$runTime(s) = \beta s + \gamma$，其中 s 输入数据大小；β，γ 为工作任务相关待定常数；
2. 运行时间与集群子节点数之间的**反比例关系**：$runTime(n) = \alpha / n + \delta$，其中 n 为集群子节点数；α，δ 为与集群计算能力相关待定常数。

综合这两条规律得出了 Hadoop 云系统**运行时间模型**：

$$runTime(n, s) = \alpha s / n + \beta$$

其中 n，s 分别为集群子节点数和输入数据大小；参数 α 表征任务的计算密集度，参数 β 是集群建立任务的常数时间。该模型通过了人脸检测程序、运动目标检测与跟踪程序和 Hadoop 示例程序 WordCount 的检验并取得了较好的预测效果。

实验表明，Hadoop 云系统对视频处理有明显的加速效果，五个子节点的集群即可将处理时间缩减到单节点的 25%以下。而且集群的加速潜力极大。只要有足够的集群规模，理论上可以达到任何加速比。

第五章 总结与展望

5.1 总结

本文选题紧贴第五次 IT 革命的核心：“云计算”，提出了将开源云计算平台 Hadoop 引入视频分析领域的构想，并从平台建设、算法移植、性能实验三个方面详细阐述了如何实现这一构想，以及实现这一构想能取得的对大规模视频处理的加速效果。

平台建设方面，本文提出的面向大规模视频处理的 Hadoop 云系统是在原始的 Hadoop 云系统基础上，结合其子项目 Fuse-DFS 和视频处理库 OpenCV、FFMPEG 及其 Java 接口项目 JavaCV 所构成的。

算法移植方面，首先要掌握利用 Hadoop 框架逐帧读取视频并送入 MapReduce 计算模型的方法。其次是利用 JavaCV 实现已有的视频处理算法，一种比较好的方式是将视频处理算法封装为一个静态方法，让算法模块与 MapReduce 框架即相对独立，又能有机结合。最后，如果是面向图像序列的算法，需要利用 Map 对输入视频进行冗余分组，由 Reduce 进行处理，最后利用第三方程序或者一个新的 MapReduce 程序来合并输出结果。另外，如果输入视频大小严重不平衡，可以利用 ffmpeg 对大视频进行独立性分割，以避免单个节点 Map 负载过大。

性能实验方面，视频处理程序在 Hadoop 上的运行时间与输入视频的大小存在线性关系，与集群子节点数存在反比例关系。这两条规律被概括为“线性关系”和“反比例关系”。综合这两条规律可以得出运行时间模型。利用此模型可以对集群运行时间进行预测。实验表明，Hadoop 云系统对视频处理有明显的加速效果，五个子节点的集群即可将处理时间缩减到单节点的 25%以下。而且集群的加速潜力极大。只要有足够的集群规模，理论上可以达到任何加速比。

5.2 亮点

本文具有以下四个方面的突出亮点：

1. **融合四个开源项目的面向大规模视频处理的 Hadoop 云系统。**本文提出的 Hadoop 云系统融合了 Fuse-DFS（包括 Fuse）、FFMEPG、OpenCV 和 JavaCV，

使 Hadoop 系统具备了处理大规模视频的能力。

2. **两种经典视频处理算法的移植思路和实现。**本文详细阐述了如何将面向单帧图像和面向图像序列的传统图像处理算法移植到 Hadoop 云系统上。并探讨了大视频分割和实时视频输入的处理方法。
3. **格点对比实验和详细的数据。**本文针对 Hadoop 系统上的人脸检测程序进行了详细实验。实验以输入视频数、集群子节点数作为输入变量，运行时间作为观测变量，进行了格点对比实验。
4. **深入的实验分析和数学建模。**本文针对实验数据进行了详细而深入的分析，发现了运行时间的线性关系和指数律，提出了适用于中小集群的 Hadoop 运行时间模型并对其进行了有效性验证。

5.3 展望

本文在平台建设、算法移植、性能实验都做了一些工作，但依然有很多需要拓展的部分。

在平台建设方面，Hadoop 云系统如何与实际监控系统对接还有待细化。

在算法移植方面，如何利用 MapReduce 计算模型处理超高分辨率卫星图像可以作为一个研究点。

在性能实验方面，本文利用 5 个子节点的小集群对 Hadoop 云系统视频处理能力进行了实验、分析和评估，得出的推论中有一点：Hadoop 集群的加速潜力很大（由反比例关系推出）。此推论对于中型、大型集群是否成立有待实际实验数据来验证。另外，中型、大型集群如果执行单任务可能是对计算资源的巨大浪费。在多任务并行的情况下集群的性能也是今后分析的一个选点。

致谢

本文的完成离不开信息系统与管理学院系统工程系涂丹老师的指导和信息系统与管理学院学员大队学院三队段晓峰教导员的支持。

涂丹老师选准了云计算这个热点，并将视频处理与之相结合的命题为我指明了研究方向，他高水平的耐心指导使得本文的内容和表达都得到了充实和提高。

段晓峰教导员在得知我因缺乏云计算实验条件而陷入困境时当即允许我使用学员队机房进行实验。我在搭建真实系统中积累了宝贵的安装和配置经验。而且，机房搭建的系统使得本文第四章的实验数据真实可靠，为实验结果的分析打下了坚实基础。

此外，系统工程系张茂军老师、教学科研办公室老松杨主任在论文写作期间给与我热情的关心和亲切的指导；学员三队陈一帆同学和我毕业设计题目均与 Hadoop 有关，彼此交流启发，在此一并感谢。

四年大学时光，在我身边的是五院三队 102 个兄弟和 10 个姐妹，是同学也是战友。相聚是一团火，不会忘记在一起的疯狂与快乐，感谢你们开启了我这个“学习机器”沉睡的另一半大脑！散开是满天星，祝福我们，每一个三队人，在新的学位和岗位上，唱响时代的最强音！

从出生到二十二岁，没有哪一天不在关注我的是我亲爱的妈妈和姥姥。感谢你们的养育、爱护和教育！我必将奋发图强，学成报国，做一个对得起家人期望的孩子，做一个不辜负母校培养的科技人，用创造和成果去书写无悔的青春！

谭瀚霖，于长沙国防科学技术大学

2013 年 06 月 06 日

参考文献

[1] Tom White.《Hadoop 权威指南(Hadoop The Definitive Guide)》[M]. 曾大聃、周傲英 译. 北京：清华大学出版社，2010.

[2] Peter Mell, Timothy Grance. The NIST Definition of Cloud Computing[R]. National Institute of Standards and Technology Special Publication 800-145. 2011.

[3] 杨帆，沈奇威. 分布式系统 Hadoop 平台的视频转码[J]. 计算机系统应用. 2011，20(11):80-85.

[4] 李倩，施霞萍. 基于 Hadoop MapReduce 图像处理的数据类型设计[J]. 软件导刊. 2012，11(4):182-183.

[5] 朱义明. 基于 Hadoop 平台的图像分类[J]. 西南科技大学学报. 2012，26(2):70-73.

[6] 维基百科. big_data 主页. http://en.wikipedia.org/wiki/Big_data,2012 年 12 月 28 日.

[7] Beyer, Mark. Gartner Says Solving 'Big Data' Challenge Involves More Than Just Managing Volumes of Data. http://www.gartner.com/it/page.jsp?id=1731916，2012 年 12 月.

[8] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Michael Stonebraker. A comparison of approaches to large-scale data analysis[J]. SIGMOD '09: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. 2009.

[9] 中国友友系统（公司）. 友友官方网站. http://www.yoyosys.com/index.php?id=142，2013 年 01 月 09 日.

[10] 新加坡 NEC 公司. NEC 官方网站. http://www.nec.com.sg/index.php，2013 年 01 月 09 日.

[11] 维基百科. Hadoop 主页. http://en.wikipedia.org/wiki/Apache_Hadoop，2012 年 12 月 27 日.

[12] Wilbert Kraan and Li Yuan. Cloud Computing in Institutions: A Briefing Paper[J], JISC Cetis Centre for Educational Technology & Interoperability Standards, 2009.

[13] 维基百科. Fuse-DFS 主页.http://en.wikipedia.org/wiki/MountableHDFS，2013 年 01 月 20 日.

[14] GoogleCode. JavaCV 主页. http://code.google.com/p/javacv，2012 年 08 月 20 日.

[15] Viola and Jones. Rapid object detection using a boosted cascade of simple

features[J]. Computer Vision and Pattern Recognition. 2001.

[16] Muhammad Usman Ghani Khan, Atif Saeed. HUMAN DETECTION IN VIDEOS[J]. Journal of Theoretical and Applied Information Technology. 2009:212-220.

[17] Kalman. R. E. A New Approach to Linear Filtering and Prediction Problems. Transactions of the ASME - Journal of Basic Engineering. 1960. 82:35-45.

[18] 赵晓永，杨扬，孙莉莉，陈宇. 基于 Hadoop 的海量 MP3 文件存储架构[J]. 计算机应用. 2012，32(6)：1724—1726.

[19] 王峰,雷葆华. Hadoop 分布式文件系统的模型分析[J]. 电信科学. 2010. 26(16)：95-99.

[20] Hadoop 1.0.4 文档. mapred_tutorial. http://hadoop.apache.org/releases.html,2013 年 01 月 26 日.

附录

1 Hadoop 集群的软件安装和配置

本章对于文本提出的面向的规模视频处理的 Hadoop 云系统的软件安装配置进行详细的说明。

1.1 安装 ssh

对于安装服务器 Linux 系统的读者，这一步通常可以省略，因为服务器一般自带 ssh。对于没有 ssh 的 Linux 系统，采用如下方法安装。对于已经安装 ssh 的 Linux 系统，按 7)、8)配置无密码登录。

a) 安装 zlib（将 zlib-1.2.7.tar.gz 拷贝到主文件夹中）

a) 解压：

```
@ubuntu:~$ tar –xzvf zlib.-1.2.7.tar.gz
```

b) 进入 zlib 目录：

```
@ubuntu:~$ cd zlib.-1.2.7
```

c) 配置：

```
@ubuntu:~$ sudo ./configure
```

d) 编译：

```
@ubuntu:~$ make
```

e) 安装：

```
@ubuntu:~$ sudo make install
```

b) 安装 jdk（将 jdk-6u34-linux-i586.bin 或更新版本的 jdk 拷贝到主文件夹中）

f) 为 jdk-6u34-linux-i586.bin 赋予可执行权限：

```
@ubuntu:~$ sudo chmod u+x jdk-6u34-linux-i586.bin
```

g) `@ubuntu:~$ sudo –s ./jdk-6u34-linux-i586.bin`

h) 添加环境变量到 profile：

```
@ubuntu:~$ sudo gedit /etc/profile
```

i) 在文件末尾添加环境变量：

```
#set java environment

JAVA_HOME=/home/administrator/jdk1.6.0_34

export JRE_HOME=/home/administrator/jdk1.6.0_34/jre
```

```
export CLASSPATH=.:$JAVA_HOME/lib:$JRE_HOME/lib:$CLASSPATH

export PATH=$JAVA_HOME/bin:$JRE_HOME/bin:$PATH
```

c) 重启

d) 检验是否安装成功：

```
@ubuntu:~$ java –version
```

如果出现 `java version “1.6.0_34”`，则安装成功

e) 安装 openssl（将 openssl-1.0.1c.tar.gz 拷贝到主文件夹中）

a) 解压 openssl-1.0.1c.tar.gz：

```
@ubuntu:tar –xzvf openssl-1.0.1c.tar.gz
```

b) 进入 openssl-1.0.1c 目录中：

```
@ubuntu:cd openssl-1.0.1c
```

c) 配置：

```
@ubuntu:~$ sudo ./Configure
```
(Configure 大写首字母)

d) 编译：

```
@ubuntu:~$ make
```

e) 安装：

```
@ubuntu:~$ sudo make install
```

f) 安装 openssh（将 openssh-6.0p1.tar.gz 拷贝到主文件夹中）

a) `@ubuntu:tar –xzvf openssh-6.0p1.tar.gz`

b) `@ubuntu: cd openssh-6.0p1`

c) `@ubuntu:~$ sudo ./configure`

d) `@ubuntu:~$ make`

e) `@ubuntu:~$ sudo make install`

f) `@ubuntu:~$sshd`

g) 生成 SSH 密钥对

a) `@ubuntu:~$ ssh-keygen –t rsa`

之后连续输入三个回车 ENTER

b) `@ubuntu:~$ cd .ssh`

c) 将 id_rsa.pub 添加到 authorized_keys 中

```
@ubuntu:~$ cat id_rsa.pub >> authorized_keys
```

h) 配置 ssh 无密码登录

修改/etc/passwd 文件，在文件末尾加入

```
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
```

a) 设置权限

```
@ubuntu:~$ chmod 700 .ssh
@ubuntu:~$ chmod 644 .ssh/authorized_keys
```

b) 添加 id_rsa 到 ssh_agent

```
@ubuntu:~$ ssh-add id_rsa
```

c) 验证无密码登录是否设置成功

```
@ubuntu:~$ sudo ‘/usr/local/sbin/sshd’
@ubuntu:~$ ssh localhost
```

如果能够登录到 localhost ,且不用输入密码，则设置成功

注意：

1. **如果无密码登录失败，查看 /var/log/secure， 看看是否是某个目录或者文件的权限不够，修改相应的目录权限为 600 即可。**
2. **对于集群免 ssh 配置，只需要在一台机器上生成 id_rsa, id_rsa.pub, authorized_keys, 将其复制到其他机器上，并将/etc/passwd 复制到其他机器上对应位置即可。注意不要每台机器去生成 id_rsa，否则你将面临组合爆炸的 ssh 配置量。**

1.2 安装 Hadoop

1) 解压：`@ubuntu:~$ tar -xzvf hadoop-1.0.3.tar.gz`

2) 配置:

a) 配置 hadoop-env.sh:

```
@ubuntu:~$ sudo gedit hadoop-1.0.3/conf/hadoop-env.sh
```

找到相应位置，并修改如下

```
# set java environment
export JAVA_HOME=/home/administrator/jdk1.6.0_34
```

这里同样应该设置为本机的 JAVA_HOME 目录。

b) 打开配置文件 core-site.xml:

```
@ubuntu:~$ sudo gedit hadoop-1.0.3/conf/core-site.xml
```

在<configuration></configuration>中添加：

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

```
</property>
```

c) 打开配置文件 hdfs-site.xml:

```
@ubuntu:~$ sudo gedit hadoop-1.0.3/conf/hdfs-site.xml
```

在<configuration></configuration>中添加：

```
<property>
  <name>dfs.name.dir</name>
  <value>/home/administrator/name</value>
  <description>配置 namenode 的本地存储路径</description>
</property>
<property>
  <name>dfs.data.dir</name>
  <value>/home/administrator/data</value>
  <description>配置 datanode 的本地存储路径</description>
</property>
<property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>副本个数，配置默认为 3 个</description>
</property>
```

d) 打开配置文件 mapred-site.xml:

```
@ubuntu:~$ sudo gedit hadoop-1.0.3/conf/mapred-site.xml
```

在<configuration></configuration>中添加：

```
<property>
  <name>mapred.job.tracker</name>
  <value>localhost:9001</value>
</property>
```

3) 添加环境变量：

a) 打开 profile 文件：

```
@ubuntu:~$ sudo gedit /etc/profile
```

在文件末尾添加：

```
#set hadoop environment
export HADOOP_HOME=/home/administrator/hadoop-1.0.3
export PATH=$HADOOP_HOME/bin:$PATH
```

b) 重启计算机

3) 格式化 namenode:

重新启动后，打开终端（Ctrl+Alt+T for Ubuntu）

格式化：

```
@ubuntu:~$ hadoop namenode –format
```

4) 开启 hadoop:

a) 开启 sshd:

```
@ubuntu:~$ sudo ‘/usr/local/sbin/sshd’
```

b) 启动 hadoop:

```
@ubuntu:~$ start-all.sh
```

c) 打开浏览器，在地址栏中输入：

localhost:50070

若能正常打开网页，则 hadoop 安装成功

d) 上传文件到 HDFS：

```
@ubuntu:~$ hadoop fs –mkdir input
@ubuntu:~$ hadoop fs –put text input/text
```

e) 运行 wordcount 程序：

```
@ubuntu:~$ hadoop jar hadoop-1.0.3/hadoop-examples-1.0.3.jar
wordcount input output
```

5) 更改 ip 映射：

a) 打开 hosts 文件：

```
@ubuntu:~$ sudo gedit /etc/hosts
```

b) 修改 ip 映射：

```
节点 1 的 IP        节点 1 的主机名
节点 2 的 IP        节点 2 的主机名
……
```

注意：

1. **修改 IP 映射是必不可少的一步!在每台机器的 hosts 文件中要存放整个集群的每台机器的 ip 和 hostname，并且要保持唯一对应!**
2. **关闭集群机器的防火墙或进行适当配置：开放 9000，9001，50070 等端口。**
3. **如果感觉 Hadoop 配置得当又无法正常启动，注意查看$HADOOP_HOME/logs 下的日志文件，对于快速发现问题很有帮助。**

c) 重启计算机

d) 打开 sshd 服务，并开启 hadoop（同上）

e) 关闭 hadoop 安全模式：

```
@ubuntu:~$ hadoop dfsadmin -safemode leave
```

否则 HDFS 是只读的。

1.3 安装 fuse-dfs

1.3.1 安装 fuse2.8.5

解压， configure, make, makeinstall，在 RedHat 系列 Linux 上一切比较顺利。在 Ubuntu12.04 上编译会出错。

1.3.2 安装 fuse-dfs

解压，进入解压后的目录，在该目录打开终端，终端中执行./configure，然后是 make 源码包编译有错，很多宏定义错误和一个函数参数个数有问题（可能版本不匹配吧）。一次 make 修正一个源码错误，耐心修正，直到 make 成功，执行 make install。

fuse-dfs 的运行还需要 Hadoop 的 libhdfs.so.0 动态库文件，此文件在 Hadoop 根目录下：

```
$HADOOP_HOME/c++/linux-i386-32/lib
```

将 bin 目录下的可执行文件 fuse_dfs 和 fuse_dfs_wrapper.sh 文件置于同一个目录下。编辑 fuse_dfs_wrapper.sh 文件设置环境变量。注意要根据操作系统设置 OS_ARCH 环境变量，32 位系统设为 i386，64 位系统设为 AMD64。然后可以利用 fuse_dfs_wrapper.sh 来挂载 HDFS 文件系统到本地。例如执行：

```
sh fuse_dfs_wrapper.sh dfs://localhost:9000 /export/hdfs
```

就将 HDFS 文件系统挂载到本地/export/hdfs 目录下。

再次挂载到此目录前需要卸载已经挂载的目录，命令为：

```
fusermount -u /export/hdfs
```

1.4 安装 OpenCV，FFMPEG，JavaCV

在 Google 中搜索 opencv 可以在 SourceForge 网站上找到 OpenCV 源码包，按安装 ssh 的方法：configure 配置、make 编译、make install 安装。

FFMPEG 在麒麟系统中也是自带的，如果没有请自行 Google 下载相应的 RPM 安装包并使用如下命令安装：

```
rpm -ivh 安装包路径
```

在 http://code.google.com/p/javacv 网址可以下载与 OpenCV 版本对应的 JavaCV（无需安装，作为库直接导入 eclipse 工程中）。

1.5 安装 eclipse

a) 解压：`@ubuntu:~$ tar -xzvf eclipse-jee-juno-linux-gtk.tar.gz`

b) 将 hadoop 插件添加到 eclipse 中，即将 hadoop-eclipse-plugin.jar 文件复制到 eclipse 安装目录下的 plugins 目录下。

c) 新建 Map/Reduce Project，configure hadoop install directory，选择 hadoop 安装目录，即 $HADOOP_HOME 所指向的目录。然后就可以开始编写 Hadoop 程序了。

d) 注意 Hadoop 程序的输入输出目录设置，例如：

```
hdfs://localhost:9000/data/project/input
hdfs://localhost:9000/data/project/output
```

指导教员审核意见：   签名：　　年　　月　　日
教研室(研究室、实验室)意见：   领导签名：　　年　　月　　日
系(研究所、重点实验室)意见：   领导签名：　　年　　月　　日
学院训练部意见：   (公章) 年　　月　　日

注：开题报告由学员撰写，答辩结束交指导教员。