基于数据挖掘的 P2P 流量识别技术

（申请清华大学工程硕士专业学位论文）

培 养 单 位： 计算机科学与技术系

工 程 领 域： 计算机技术

申 请 人： 白 利 彬

指 导 教 师： 王 鼎 兴 教 授

联合指导教师： 张 崇 轲 高 工

二〇一三年十二月

P2P Flow Recognition Based on Data Mining Technology

Thesis Submitted to

Tsinghua University

in partial fulfillment of the requirement

for the professional degree of

Master of Engineering

By

Bai Libin

(Computer Technology)

Thesis Supervisor: Professor Wang Dingxing

Associate Supervisor: Senior Engineer Zhang Chongke

December, 2013

基于数据挖掘的P2P流量识别技术

白利彬

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：

清华大学拥有在著作权法规定范围内学位论文的使用权，其中包括：（1）已获学位的研究生必须按学校规定提交学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（2）为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆、资料室等场所供校内师生阅读，或在校园网上供校内师生浏览部分内容。

本人保证遵守上述规定。

（保密的论文在解密后遵守此规定）

作者签名：_____ 导师签名：_____

日　　期：_____ 日　　期：_____

摘要

随着 P2P 技术的发展，经统计 P2P 应用产生的流量已经占据了整个互联网流量的 60~90%。P2P 各类应用所具备的文件共享，抢占带宽，分布式计算等特点消耗了大量的网络带宽，甚至在某些情况下造成网络拥塞，同时对网络上的其他应用造成了影响。因此，要维护网络空间的正常秩序和正常应用，必然要首先实现对 P2P 流量的准确、高效的识别。

一方面流量识别本质上就是一个分类问题，因而，可以将数据挖掘方面的分类算法应用到流量识别上。本文利用开源数据挖掘工具 WEKA 建立一个单机流量识别系统，对小规模的网络流量数据，可以做到较好的识别。另外一方面，网络流量识别从规模上来看，是一个大数据集以及超大数据集的存储和处理的问题。针对数据规模的问题，本文提出使用 Hadoop ， HDFS ， Mahout 三个开源系统共同组建一个分布式数据挖掘系统，处理并识别 P2P 网络流量。

论文工作包括：

1 提出一个基于数据挖掘的 P2P 流量识别系统的架构。包含了从网络流量采集、上传、存储、预处理、识别、结果查询分析等各个方面。

2 基于架构设计，给出了基于 Weka 的 P2P 流量的识别系统。能够比较有效的识别出网络上的 P2P 流量。但是系统在面对大规模数据集的情况下，系统存在性能上的瓶颈。

3 提出了一个基于云计算改进的 P2P 流量识别系统。实验结果表明，利用这个分布式计算的数据挖掘系统，处理并识别 P2P 流量，在面对海量数据的时候，比基于 Weka 数据挖掘的 P2P 流量识别系统具有更高的性能，并且系统有比较好的伸缩性。

关键词：P2P；数据挖掘；云计算；Mahout；WEKA

Abstract

With the development of P2P technology, P2P traffic consumes 60 to 90% of the entire Internet traffic. With the increasing number all kinds of P2P applications sharing the file, preempt the bandwidth, distributed computing. All of these characteristics have already caused the huge consumption of network bandwidth, leading to network congestion and even poses a threat to the quality of service for other applications. Therefore, it is of great significance to implement an accurate and efficient P2P traffic identification system for network management and rational utilization of network resources.

On one hand, traffic identification is an essential classification problem. Data mining methods can be applied to the field of traffic identification and classification. Based on the transport layer identify P2P traffic, this paper combines open source data mining tool WEKA and P2P traffic characteristics to complete stand-alone system. For small-scale network traffic data, it can be done better recognition.

On the other hand, the network traffic identification on the scale is the problem about storage and processing of big data or even a large data. Aiming at the problem of large data size, this paper proposes using three open source systems Hadoop, HDFS, and Mahout to combine a distributed data mining system, processing and identification of P2P network traffic.

This paper finishes the work as follows:

First, completed a P2P traffic identification system based on data mining of architecture design, system contains from network traffic capture, upload, storage, preprocessing, identification and analysis of the results query.

Then based on the architectural design, Weka based P2P traffic identification system is established. This system can effectively

identify the P2P traffic from the network. But when system processes the large data sets, there are system performance bottlenecks.

Finally, this paper creates a P2P traffic identification system based on cloud computing to improve the performance and resolve the problem of bottlenecks. The experiment suggests that in use of the proposed distributed data mining system for processing and identifying the P2P traffic under the scenario of big data, better performance and system scalability can be achieved.

Key Words: P2P; Data Mining; Cloud Computing; Mahout; WEKA

目 录

第 1 章 绪论

1.1 研究背景

随着 P2P 技术的发展，经统计 P2P 应用产生的流量已经占据了整个互联网流量的 60~90%。P2P 各类应用所具备的文件共享，抢占带宽，分布式计算等特点消耗了大量的网络带宽，甚至在某些情况下造成网络拥塞，同时对网络上的其他应用造成了影响。因此，要维护网络空间的正常秩序和正常应用，必然要实现 P2P 流量的准确、高效的识别。

1.2 P2P 概述

1.2.1 P2P 的定义和发展

点对点技术（Peer To Peer， 简称 P2P）又称对等互联网络技术。作为一种网络模型，它包含的所有的节点都是对等的。P2P 技术极大的促进网络的发展，使用户更加方便的共享各类资源。与有中心服务器的中央网络系统不同，无需依赖中心服务器，对等网络的每个用户端既是一个节点，也有服务器的功能。单独任何一个节点虽然无法直接找到其他节点，但是成规模的用户其户群之间进行信息交流，就可以找到所有用户。

随着 P2P 技术的发展，最大的变化就是改变了互联网上资源的分布。互联网上信息的提供者不再局限于“中心服务器”， 而是采用“人人为我，我为人人”的分布式模式，每个点都可以变成资源的提供者。

P2P 技术的出现颠覆了以往互联网信息资源的不对称的应用场景，从信息的中心集中模式转向了信息用户共享甚至创造的模式。互联网上的各个节点上的信息和资源都可以共享和发布，这样就会使互联网的信息和资源的分布和利用更为均衡。

目前国内外采用 P2P 网络分布模式的应用包含以下一些，国外：BitTorrent, eDonkey, FastTrack, Gnutella, Skype。国内：Maze, KuGoo， PPLive， PPStream，

Xunlei，Youku。

1.2.2 P2P 应用的分类

目前互联网上普遍使用的 P2P 应用主要可以分为以下几种：

- 文件共享软件，例如 eDonkey、emule、BitTorrent、Xunlei 等；
- 流媒体（包括音视频）共享，例如: PPLive、PPStream、风行、Youku 等；
- 即时通信软件，例如：MSN、QQ、Yahoo Messenger、skype 等；
- 安全的 P2P 通讯与信息共享，例如 Skype、Crowds、Onion Routing 等；
- 虚拟化数字货币，例如 Bitcoin , Litecoin。

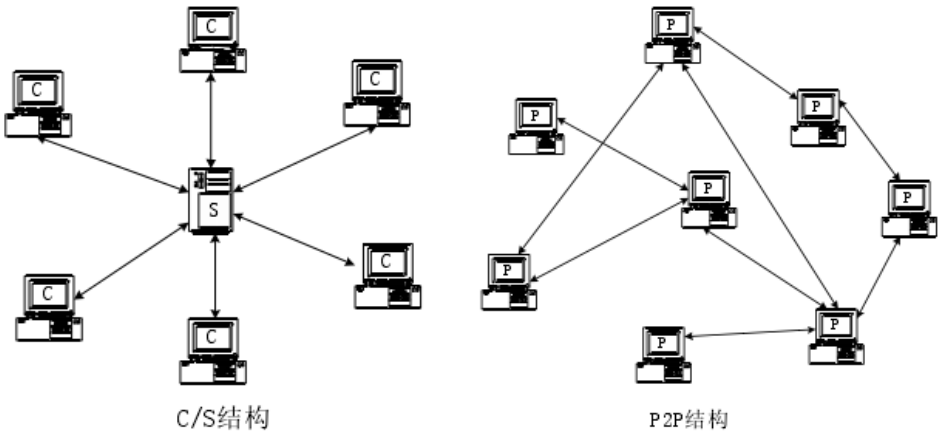


图 1.1 服务器客户端结构与 P2P 结构

1.3 P2P 研究的意义

P2P 大量消耗带宽资源，尤其是一些视频、大型软件等的下载，很容易导致网络的拥塞，使其他互联网应用的性能降低，增大了电信运营商的网络负荷和运营成本。

对传统电信运营商而言，P2P 技术及应用带来的冲击和挑战是非常巨大的。P2P 应用在吸引大量客户的同时，会使网络流量急剧上升。如何应对 P2P 带来的冲击和挑战，包括识别和控制 P2P 流、版权保护等, 这些已经成为摆在网络运营商面前的重要课题。

要维护网络空间的正常秩序和正常应用，必然要实现 P2P 流量的准确、高

效的识别。研究 P2P 网络流量的识别，可以为网络管理提供很好的依据，能够提升整体网络的质量。

1.4 本文结构

本文首先对传统的 P2P 网络流量的识别技术进行了分析，分析了各种识别技术的优缺点。本文在第三章提出了一种单机模式下的基于开源数据挖掘软件 Weka 的 P2P 流量监测方案，然后在第四章提出了一种结合数据挖掘技术以及云计算并行计算技术的 P2P 流量检测系统方案。

本文一共包含五章：

1. 绪论，描述了 P2P 的定义、发展以及应用分类。然后，简单阐述了对 P2P 流量研究的意义。最后，阐述了本文的结构和研究内容。

2. 基于数据挖掘的 P2P 流量识别系统架构设计。首先对常见 P2P 流量识别技术进行综述，介绍了几种传统的流量识别技术，简要介绍了各种技术的原理。最后在分析了传统技术所遇到的问题后，点明了本文要解决的问题，进而提出一种基于数据挖掘的 P2P 流量识别系统的架构设计。

3. 基于 WEKA 的单机 P2P 流量识别系统。首先简要介绍了数据挖掘的概念，然后说明了本系统所使用的三种相关算法。最后使用开源数据挖掘工具 WEKA，实现了一个单机 P2P 流量识别系统。

4. 基于云计算改进的 P2P 流量识别系统。首先介绍了数据挖掘在云端的优势，然后介绍了系统所用到的相关技术。之后结合云计算的优点，提出了一种基于 Kafka、Hadoop、Hive 以及开源数据挖掘工具 Mahout 的改进 P2P 流量识别系统设计。最后利用系统对大数据的网络流量进行 P2P 识别的实验。

5. 总结与展望，在总结本文系统的研究成果基础上，进一步展望未来的研究工作及主要方向。

第 2 章 基于数据挖掘的 P2P 流量识别系统架构

2.1 现有 P2P 流量识别技术

首先，简要描述一下，传统的 P2P 流量识别技术，主要如下[1]：。

2.1.1 基于端口号的识别技术

在早期，P2P 常规流量检测一般通过常用的端口来进行识别。例如，某些早期的 P2P 软件使用固定端口 6881～6889 传输数据。这样只需要根据软件对应使用的端口，就可以判断出 P2P 流量。

应用层一些服务和软件固定使用端口的对应表：

表 2.1 通用服务端口对照表

服务	端口	协议
文件传输服务	21	FTP
远程登录服务	23	TELNET
传输邮件服务	25	SMTP
用于万维网(WWW)的文本传输服务	80	HTTP
访问远程服务器上的邮件服务	110	POP3
互联网消息存取服务	143	IMAP4
安全的超文本传输服务	443	HTTPS
安全的远程登录服务	992	TELNETS
安全的消息存取服务	993	IMAPS

2.1.2 基于会话关键字的识别

用关键字进行 P2P 流量识别是基于人工经验。在抓取到 P2P 软件的网络包之后，对网络包中出现频率高的特征字符串进行归纳总结。最终确定出 P2P 软件的会话关键字。之后就可以对未知网络流量和会话关键字进行匹配，从而可以确定未知流量是否是 P2P 流量，甚至能具体辨别出是哪种 P2P 软件产生的流

量。例如 BT 应用软件的握手协议如下表 2.2

表 2.2 典型的 BitTorrent protocol Handshake

名称	内容
Protocol Name Length	19
Protocol Name	BitTorrent protocol
Reserved Extension Bytes	0000000000100001
SHA1 Hash of info dictionary	611407d1c530ba4ed4e7b70dd9470d4d41cfb799
Peer ID:	2d4243303133362dd61d246bae111a58c288d9e6
Client is BitComet v0136	

但是随着 P2P 软件的快速更新换代，各种 P2P 软件会根据运行环境、版本升级等的情况改变关键字符串。所以利用关键字识别 P2P 流量，仅能对全部 P2P 流量的一部分起作用。

2.1.3 基于 URL 的过滤方法

此方法通过判断 HTTP 请求的 URL 中是否含有特定的 P2P 关键字，来进行 P2P 流量的过滤。其主要是针对需要首先通过 http 协议来获取和初始化连接信息的一些 P2P 应用，如 BT。

这种控制局限性比较大，各种类型的 BT 软件经常采用其他方式提供种子文件，用户自己也可以通过非 http 协议下载。

2.1.4 基于流统计特性的识别

一般 P2P 网络应用作为一种分布式部署的端对端的节点网络，与一般正常网络协议，如 HTTP、FTP 等所表现出来的流量特征大有不同。每个节点之间都会频繁的建立连接，判断状态，然后传送数据。这样会造成大量的系统网络连接消耗和带宽消耗。而且表现为：持续性， 以及网络包载荷变大，上下行网络数据比较平均。

而普通的网络应用，一般只有在需要更新数据时才进行网络连接，发生网

络流量。其流量特征表现为突发，持续时间短。

这种方式的优点： P2P 应用的适用领域决定了其流量特征具有普遍性，任何 P2P 应用也离不开抢占资源，表现为持续数据上下行。所以这种方法能识别出新的应用而且即使加密的应用也可能被识别出来。

同时这种方式也存在缺点：由于这些流量的统计特征只是对网络流量的粗略分类，所以这种方法对应用分类的能力较弱。需要配合其他识别技术，对流量进行进一步甄别。

2.2 现有技术所面临问题

2.2.1 不确定性

由于 P2P 应用的多种多样，其所用的网络结构也不同。P2P 网络流量在资源共享的行为特征上各不相同，有些更偏向于硬盘资源共享，有些则偏向于计算能力共享，有些则偏向于占用网络带宽资源频繁传输。这种应用行为的多样性，为 P2P 的识别带来巨大困难。

2.2.2 海量性

目前互联网上的 P2P 应用极多，表现形式不同，但是都会造成大量数据流动在网络上 (如支持 BT 协议的各种应用在线客户端节点数可高达 100 万，一些视频网站的在线人数也会达到百万级别)。

要对这些海量流量数据进行识别，几乎不可能，这就对识别方法的性能和可扩展性提出了很高的要求。

2.2.3 加密性

针对 P2P 的识别，有些 P2P 应用加密了其传输的 Payload，例如 Skype 软件采用加密技术。流量的加密性使得常规的基于会话关键字的匹配算法在 P2P 流量识别中失效。因此，必须寻求新的流量识别方法才能解决 P2P 流量识别的准确性和可靠性问题。

2.2.4 动态端口

最新的 P2P 应用程序技术上逐渐发展到随机动态端口甚至伪装端口。

采用伪装端口(SMTP port 25 或 HTTP port 80) 可以穿越防火墙，躲避路由器的访问控制。目前大多数 P2P 应用可以随机利用空闲端口进行通信，甚至采用 80 等 HTTP 传统应用的端口，伪装成传统应用的流量，导致利用端口识别方法的识别率下降。用这种方式躲避检查的应用如下：

- 随机端口：如风行、PPLive 等；
- 可变端口：如 Skype 等；
- 伪装端口：Kazaa、Gnutella 等。

2.3 本文要研究的问题

文献[5]的研究结果表明，传统识别方法很难适应对日益复杂的多种 P2P 软件产生的流量进行识别。传统 P2P 流量识别技术的优缺点如**错误!未找到引用源。**：

随着数据挖掘理论的不断完善以及各种算法的提出，利用数据挖掘方式识别 P2P 网络流量成为区别于传统识别技术的一个突破点。本文研究的重点是在已有的技术上有效地结合贝叶斯分类算法，决策树方法和支持向量机(SVM)等数据挖掘算法来实现 P2P 流量的分类问题。

具体而言，要研究的主要问题：先在单机上基于数据挖掘的 P2P 流量识别, 然后基于云计算进行数据挖掘，提高 P2P 流量识别的性能。

2.4 系统设计原则

2.4.1 可靠性

系统对于用户的网络活动影响很大，如果产生误识别，则会非常影响用户体验，因此软件系统必须具有很小的误判，即可靠性。

2.4.2 安全性

由于网络流量中能暴露出很多个人信息，企业信息，政府信息，所以对这些敏感数据的存储，分析要注意隐私以及信息安全。

2.4.3 伸缩性

系统必须能够在用户数目爆发增加的情况下，保持稳定的性能。本系统设

计能支持对数据级别在 GB 的大数据量进行采集和分析的能力。

在系统面临更大性能压力的时候，通过很小的改动甚至只要增加硬件进行横向扩展，就能支持更多的吞吐量。

表 1.3 传统 P2P 流量识别的优缺点

	基本思想	优点	缺点
基于端口	利用传输层的特定端口	误判率低 简单	识别率低
基于内容的深层报文检测	利用报文负荷的内容是否包含特定 P2P 应用的特征字符串	误判率低 可以利用已有的一些技术手段	隐私 容易失效 对加密报文无效 处理开销大
基于启发式流量特征	基于流量的一些启发式特征，比如 P2P 节点之间的通信拓扑	对未知流量识别性较好 无隐私或加密报文问题	有误判可能 与协议相关
基于主动探测	模拟节点主动加入 P2P 网络	简单，误判率低	适用范围窄 识别率不高 侵入性

2.4.4 可扩展性

系统中的各个模块松耦合，在新技术或者新的算法出现的时候，系统可以方便的替换掉旧有的模块，而不改变系统整体结构，从而对现有系统进行功能和性能的扩展。

2.4.5 可维护性

软件系统的可维护性，即健壮性，首先体现在现有系统的 Bug 率处在一个较低水平，即使出现错误，也能够在规定的时间内解决

其次是针对新的需求，能够很方便的在不改变系统整体结构的情况下，将

需求实现融入系统中。

2.5 系统架构设计及各模块功能

P2P流量识别系统，架构设计及各部分功能模块如图2.2：

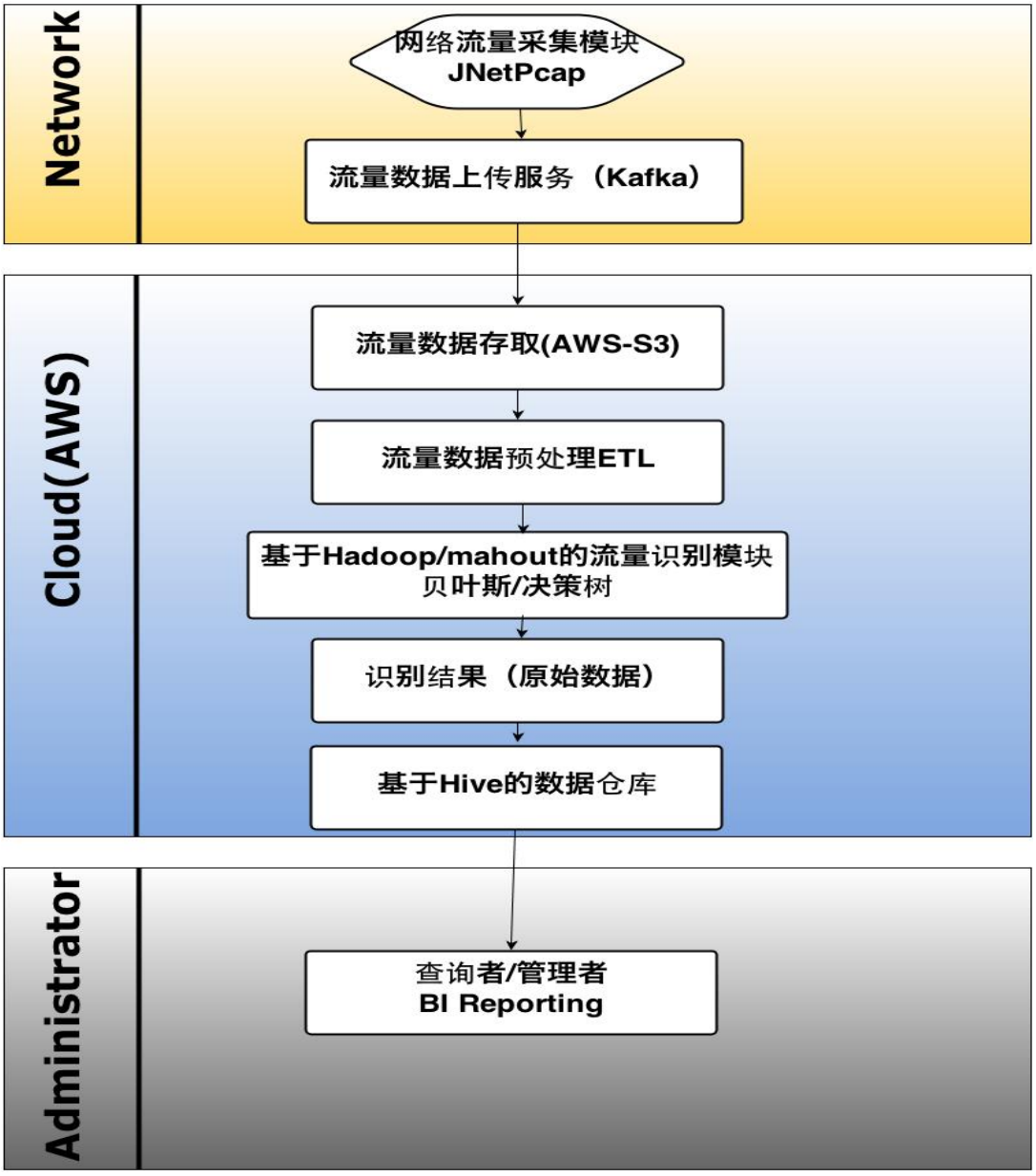


图 2.2 基于云计算的P2P数据挖掘识别架构

2.5.1 流量采集

流量采集上传是整个系统的基础，采集模块本系统采用JNetPcap。

2.5.3 流量上传

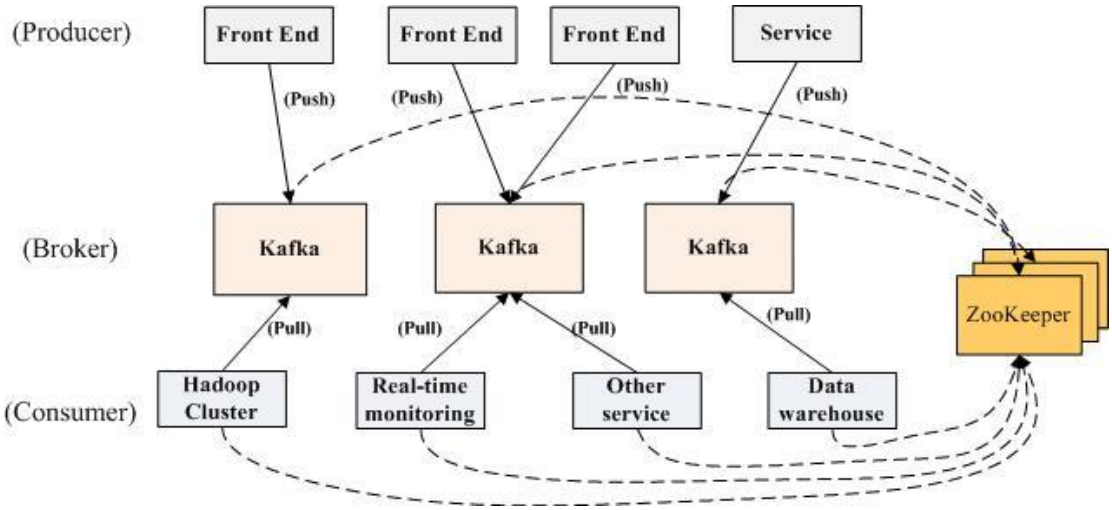
数据上传服务采用Kafka，这是一个开源的消息系统。其主要用于处理活跃的流式数据，具有很高的吞吐率。Kafka能很好的处理离线和在线数据。而且针对网络流量的海量性和突发性，使用Kafka是一个不错的选择。

Kafka的优势具体体现在：

- 消息持久化非常快，服务端存储消息的开销为O(1)，并且基于文件系统，能够持久化TB级的消息而不损失性能；
- 完全的分布式系统，broker、producer、consumer三种组件一开始就完全支持分布式部署，并可以自动实现复杂均衡；
- 高吞吐率，即使是一般机器所构成的节点，在上边处理消息的能力达到几千条每秒；
- Kafka支持将数据并行加载到Hadoop中。

Kafka工作原理如下图2.3，我们将流量采集的角色称作生产者(producer)，流量预处理以及流量识别称为消费者(consumer)，将中间的存储称作broker：

图2.3 Kafka工作原理图



生产者（采集端）采集数据，传输给broker（S3）进行存储。消费者（数据处理）是核心，它负责处理数据。消费者从broker中读取相应的数据，并进行处理。

2.5.2 流量存储

流量通过 Kafka 消息系统，上传至亚马逊的云平台 S3(S3 Simple Storage Service)简单存储服务上，S3 表现为一个超大硬盘，系统可以通过 API 可以在其中存储和检索数据。

2.5.4 流量预处理

网络流量进入S3存储平台后，首先要进行预处理。针对数据的抽取（Extract）、清洗（Cleaning）、转换（Transform）、装载（Load）等 ETL 过程主要是为了保证数据的质量问题。具体表现经过预处理的数据，可以提高其正确性、完整性、一致性、完备性、有效性、时效性等几个特性。

2.5.5 流量识别

流量识别如前文所述，有多种手段。本文在单机系统中采用Weka开源工具进行识别，在基于云计算的系统中，采用Mahout来进行并行的数据挖掘，从而提高效率。

本系统中使用了Mahout的决策树算法以及贝叶斯算法，对网络流量进行分类。

2.5.6 识别结果的存储和查询

对网络流量数据进行识别之后，要对识别结果进行归类，存储，以供后续处理使用。本文在基于云计算的识别系统中，采用数据仓库工具 Hive，对识别出来的流量这种大规模数据，进行规范化，提供便捷和快速的查询工具。

2.6 小结

本章首先列举了传统的 P2P 流量识别技术，接下来阐述了现有技术所面临的问题。然后基于这些存在的问题，提出了本文研究的重点，即在已有的技术上有效地结合贝叶斯分类算法，决策树方法和支持向量机(SVM)等数据挖掘算法来实现 P2P 流量的分类问题。

接下来，描述了一个基于数据挖掘的 P2P 流量识别系统的系统架构设计，并简要描述了系统中各个模块的设计要点以及设计目的。

第 3 章 基于 WEKA 的 P2P 流量识别系统

Weka（Waikato Environment for Knowledge Analysis），是 Java 环境下的机器学习（machine learning）以及数据挖掘（data mining）软件。WEKA 编写者实现了大量数据挖掘和机器学习的算法。Weka 是一款基于 GNU 协议的开源软件。软件功能包括对数据进行预处理，分类，回归、聚类、关联规则以及在新的交互式界面上的可视化[13] 。

基于 Weka 提供的数据挖掘方法来处理 P2P 流量识别问题是本章研究的重点。

3.1 数据挖掘简介

数据挖掘（Data Mining，DM）又称 KDD（Knowledge Discover in Database）。

简单的说，数据挖掘是指使用智能方法从大规模的数据中提取数据模式，然后通过对数据模式的分析将其转化为知识，或者利用这种数据模式，进行其他分析。

数据挖掘通常可分为如下七个领域:

- 分类 （Classification）
- 估计（Estimation）
- 预测（Prediction）
- 相关性分组或关联规则（Affinity grouping or association rules）
- 聚类（Clustering）
- 描述和可视化（Description and Visualization）
- 复杂数据类型挖掘(Text, Web ,图形图像，视频，音频等)

3.2 相关算法

3.2.1 朴素贝叶斯

贝叶斯 Thomas Bayes，英国数学家。他首先将归纳推理法用于概率论基础理论，并创立了贝叶斯统计理论，对于统计决策函数、统计推断、统计的估算等做出了贡献。贝叶斯分类是一类分类算法的总称，这类算法均以贝叶斯定理

为基础，故统称为贝叶斯分类。[19] 贝叶斯决策理论是主观贝叶斯派归纳理论的重要组成部分。

朴素贝叶斯定理：

$$P(B[j]|A[i])=P(A[i]|B[j])P(B[j]) / P(A[i])$$

朴素贝叶斯分类法是基于条件独立性这一假设。即给定元组的类标号，假定属性值有条件地相互独立，即其属性间不存在依赖关系。贝叶斯的核心思想：用先验概率估计后验概率。

3.2.2 决策树挖掘算法

决策树的构造不需要领域的特有知识或参数设置，适合于探测式知识发现。决策树可以处理高维数据。基于决策树算法获得的知识，其表现形式非常直观且易于理解。

决策树算法主要包含 ID3（Iterative Dichotomiser，迭代的二分器），C4.5，分类与回归树（CART）。这三种算法都采用贪心（即非回溯的）的方法，决策树以自顶向下递归的分治方法构造，从训练元组集和他们的相关联的类标号开始构造决策树。

决策树的作用是：计算给定记录归属于某一个类别的概率或给定某条记录，寻找其最有可能的分类。

决策树方法的适应性好，能够容忍训练数据中的错误和实例的属性值缺失，可以应用到流量识别问题中。

3.2.3 支持向量机挖掘算法

支持向量机(SVM，Support vector machine)是一种既支持线性数据，更支持非线性数据的分类方法。简单的说，它将线性不可分的数据按照非线性映射，将训练数据映射到高维空间。在高维空间，就可以寻找一种线性最佳分离的超平面。

3.3 基于 Weka 的流量识别系统

3.3.1 系统模块组成及处理流程

使用朴素贝叶斯算法、决策树算法、支持向量机算法对 P2P 流量进行建模，处理和识别。本 P2P 流量识别系统的主要功能模块包括，如图 3.1：

1. 流量采集模块：网络流量的采集是整个系统实现的基础。本模块采用软件方式对网络流量进行采集。采集到的数据存储为文件形式。
2. 数据处理模块：本模块将采集到的流量格式转换为可供 WEKA 识别的格式。
3. 流量识别模块：根据本章提到的基于数据挖掘的 P2P 流量识别算法，对采集到的网络流量进行区分，判断其是否属于 P2P，将结果进行标记。

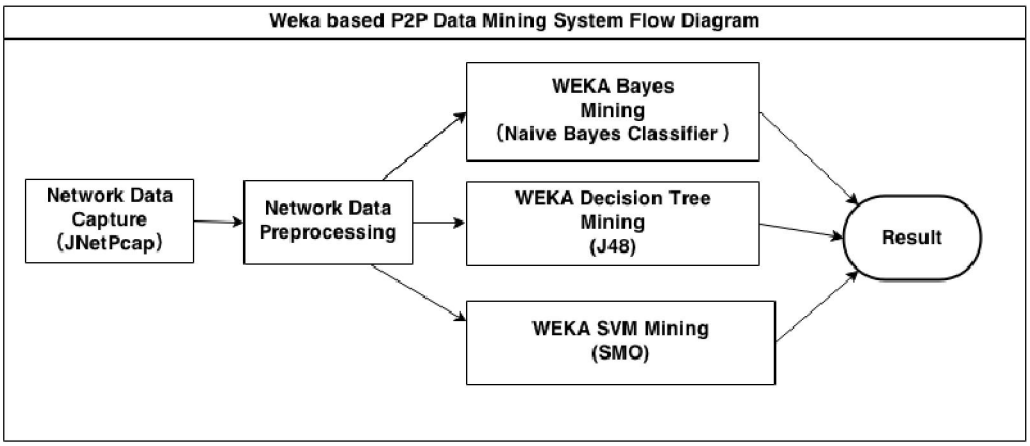


图 3.1 基于 Weka 的 P2P 流量识别系统组成及处理流程

3.3.2 度量标准

流量识别的度量标准是识别或者分类模型对未知数据对象进行分类的准确率。接下来简要介绍一下通常用于衡量分类准确率的标准。

首先定义一个分类问题的预测结果，一般可以总结为下表所示的四种可能：

表 3.1 通用服务端口对照表

	正确的、相关的	不正确的、不相关的
预测出来	true positives (真正 TP)	false positives（假正 FP）
未预测出来	false negatives（假负 FN）	true negatives (真负 TN)

- TP：预测的情况和实际相符；
- FN：实际存在或者符合，但是没有预测出来，属于漏报；
- FP：实际不存在或者不符合，但是预测存在或者符合，属于误报；
- TN：实际不存在或者不符合，也没有预测出来；

下边可以定义准确率(precision)和召回率(recall).

- Precision = TP/(TP + FP) 即正确的预测占所有预测的比值
- Recall = TP / (TP + FN) 即正确预测与真实中正确的比值
- F-Measure:

F-Measure 又称为 F-Score，是 IR（信息检索）领域的常用的一个评价标准，计算公式为：

F = (β ^2+1)P*R / β ^2P +R

β 是参数，P 是准确率(Precision)，R 是召回率(Recall)。

当参数 β =1 时，就是最常见的 F1-Measure 了：

F1 = 2P*R / (P+R)

3.3.3 算法运行的测试方法

在分类算法运行过程中，我们选用十折交叉验证十次（英文名叫做 10-fold cross-validation），用来测试算法准确性。

十折交叉验证是常用的测试方法，具体做法为：

数据十等分，其中 9/10 的数据当做训练数据，剩余 1/10 当做测试数据。轮流替换训练数据和测试数据进行测试。多次进行十折交叉验证并取其平均值作为对算法精度的评估。

3.4 实验环境

实验环境为，采用 Jcap 采集网络数据，经过预处理之后，采用 WEKA 怀卡托智能分析环境(Waikato Environment for Knowledge Analysis)所提供的数据挖掘算法(Naive Bayes Classifier，J48，SMO)，对数据进行分析。如图 3.1 所示。

3.4.1 流量采集

本文采用 JNetPcap[17]开源包进行流量采集。JNetPcap 是一个开源的 Java 包，它包含了几乎所有的对 libcap[18]包的包装，它将截获的包进行实时解码。它支持了大量的网络协议公共包以及自定义协议的嵌入。

3.4.1.1 数据采集原理：

一个包的捕捉分为三个主要部分，包括面向底层包捕获、面向中间层的数据

包过滤和面向应用层的用户接口。如图 3.2 所示：

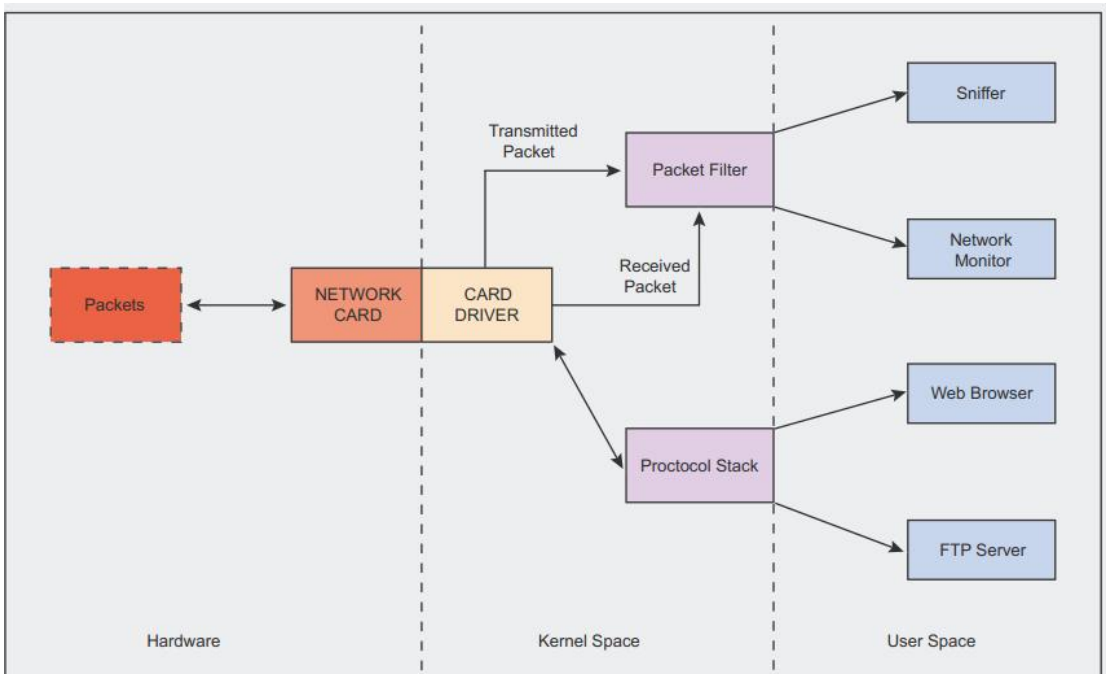


图 3.2 Libcap 工作原理示意图

3.4.1.2 数据采集程序

具体采集程序伪代码如下：

```
//获取所有的网卡设备：
int r = Pcap.findAllDevs(alldevs, errbuf);

//选择实际用到的网卡进行流量采集
PcapIf device = alldevs.get(2);
//截获所有的网络流量包，不做截断
int snaplen = 64 * 1024;
//将网卡置于混杂模式下，截获所有的网络包
int flags = Pcap.MODE_PROMISCUOUS;
//超时设置为10s
int timeout = 10 * 1000;
//打开指定网卡，开始截获该网卡上所有网络包
Pcap pcap = Pcap.openLive(device.getName(), snaplen, flags, timeout,
                errbuf);
//依次读取所截获的网络包(截取所有IP协议下的包信息<协议，目的端口，
源端口，目的IP地址，源IP地址，Payload长度>)
```

```
PcapPacketHandler<String> jpacketHandler = new PcapPacketHandler<String>()
{
      While(nextPacket(PcapPacket packet) {
           if (pcapPacket.hasHeader(Ip4.ID)) {
          pcapPacket.getHeader(ip4);
          protocal = "IP";
          payload = ip4.getPayload();
          payloadLength = ip4.getPayloadLength();
          }
          if (pcapPacket.hasHeader(Tcp.ID)) {
                pcapPacket.getHeader(tcp);
                srcPort = tcp.source();
                destPort = tcp.destination();
                protocal = "TCP";
          }
          if (pcapPacket.hasHeader(Icmp.ID)) {
                protocal = "ICMP";
          }
          if (pcapPacket.hasHeader(Udp.ID)) {
                pcapPacket.getHeader(udp);
                srcPort = udp.source();
                destPort = udp.destination();
                protocal = "UDP";
          }

      //将截获的网络流量写入文件，格式为CSV

      FileWriter fw = new FileWriter("e:\\capture-001.csv", true);
      BufferedWriter bw = new BufferedWriter(fw);
      bw.write(protocal + COMMA + destPort + COMMA + srcPort
                  + COMMA + FormatUtils.ip(ip4.source()) + COMMA
                  + FormatUtils.ip(ip4.destination()) + COMMA
                  + payloadLength);
      bw.newLine();
      }
      bw.flush();
      bw.close();
      fw.close();
```

采集示意图如下图3.3：

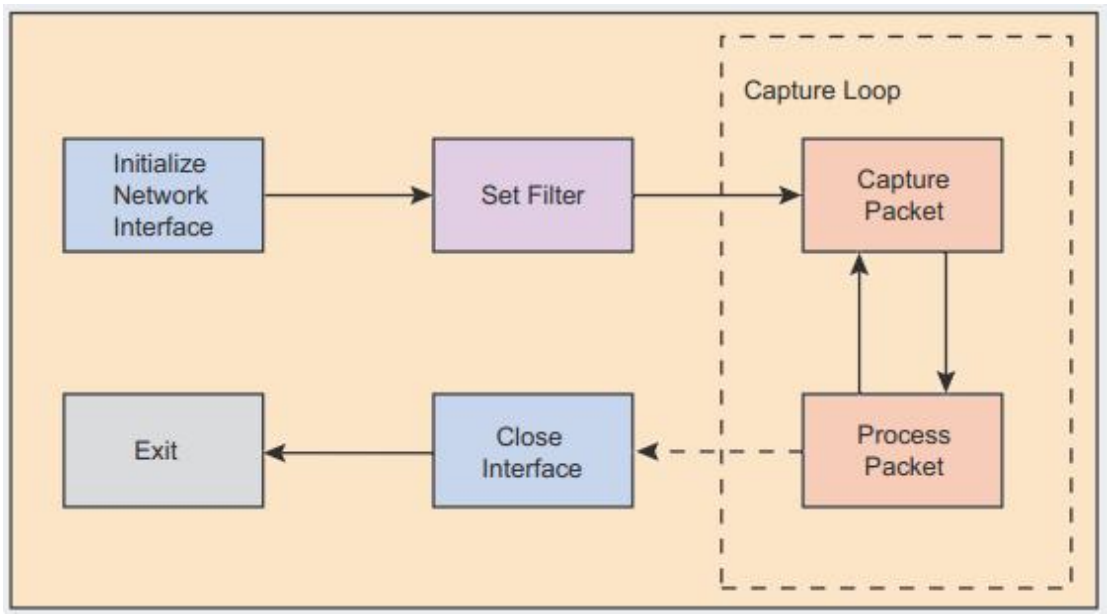


图 3.3 网络流量采集示意图

3.4.1.3 数据采集步骤：

在电脑上关闭其他使用网络的程序，分别打开 Xunlei，PPLive，BT 三种 P2P 软件，用数据采集程序各采集 5 万条网络流量的七元组。然后关闭所有 P2P 软件，采集正常上网的网络流量七元组。共采集到 Xunlei.csv, PPLive.csv, BT.csv，以及 Non-P2P.csv 四组数据，每组约 5 万条。

数据格式如下：

```
<flag,protocal,destPort,srcPort,sourceIP,destinationIP,payloadLength>
P2P-PPLIVE,UDP,5041,37775,124.126.164.255,192.168.1.103,1083
P2P-PPLIVE,UDP,5041,37775,124.126.164.255,192.168.1.103,1083
P2P-PPLIVE,UDP,5041,37775,124.126.164.255,192.168.1.103,1083
…
<flag,protocal,destPort,srcPort,sourceIP,destinationIP,payloadLength>
P2P-XUNLEI,UDP,4448,15005,192.168.1.103,85.17.56.21,14
P2P-XUNLEI,UDP,6264,24409,192.168.1.103,59.44.152.167,10
P2P-XUNLEI,UDP,7128,24409,192.168.1.103,60.48.41.32,10
…
<flag,protocal,destPort,srcPort,sourceIP,destinationIP,payloadLength>
NON-P2P,TCP,53700,80,123.150.49.13,192.168.1.103,1460
NON-P2P,TCP,53700,80,123.150.49.13,192.168.1.103,1460
NON-P2P,UDP,51711,53,101.226.4.6,192.168.1.103,81
…
<flag,protocal,destPort,srcPort,sourceIP,destinationIP,payloadLength>
```

P2P-BT,UDP,4070,14982,192.168.1.103,222.168.41.176,106
P2P-BT,UDP,14982,18746,201.245.233.99,192.168.1.103,299
P2P-BT,UDP,26315,14982,192.168.1.103,58.55.62.207,106

3.4.2 数据处理

由于 Weka 单机处理能力的问题，我们取出四组数据中各 1 万条，组成一个混合网络流量的子集，将这个子集作为使用 Weka 进行单机流量识别的数据。

使用 Weka 转换工具，将存储的 CSV 格式文件转换为 Weka 分析软件可以直接用来计算的 arff 格式。

3.4.3 流量识别

3.4.3.1 朴素贝叶斯 Naive Bayes Classifier

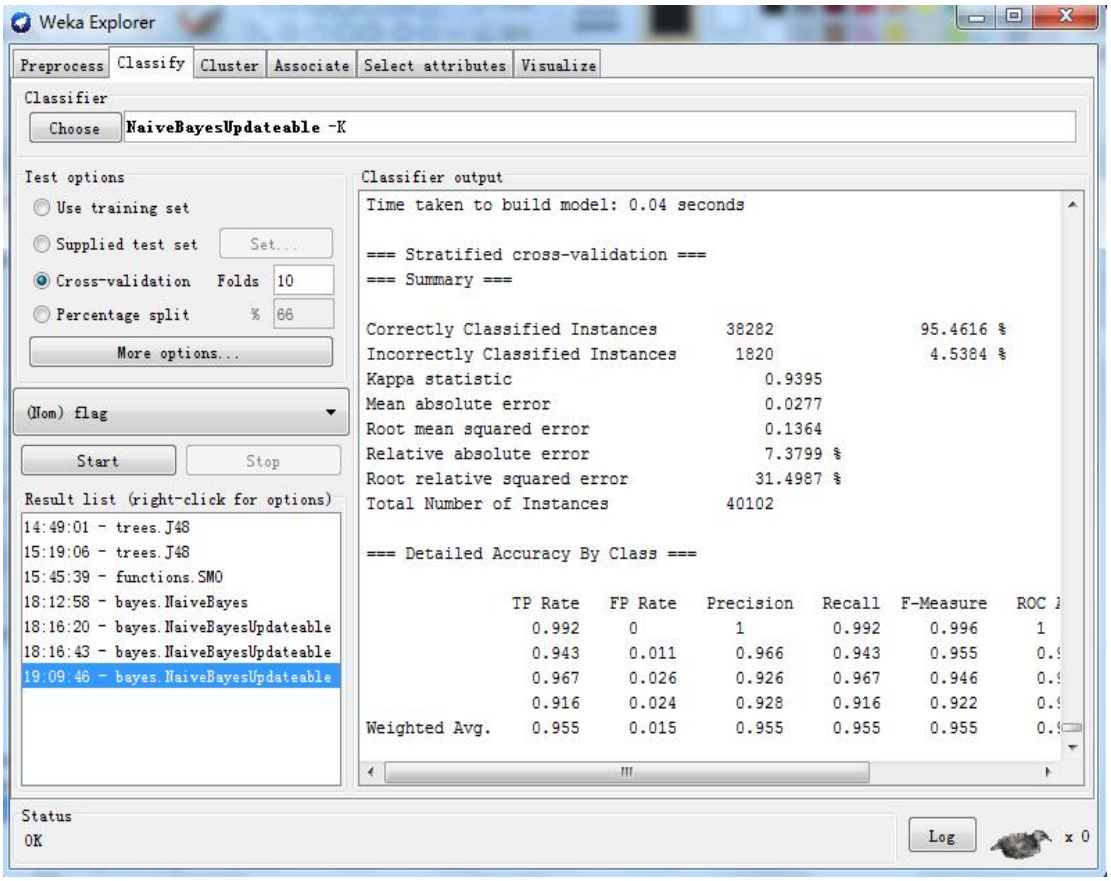


图 3.4 基于 Weka 的朴素贝叶斯流量识别过程

使用 Weka 的朴素贝叶斯算法，对所采集的数据进行流量识别

算法运行结果为：

表 3.2 Weka 朴素贝叶斯网络流量识别结果

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.992	0	1	0.992	0.996	1	P2P-PPLIVE
0.943	0.011	0.966	0.943	0.955	0.991	P2P-XUNLEI
0.967	0.026	0.926	0.967	0.946	0.998	NON-P2P
0.916	0.024	0.928	0.916	0.922	0.99	P2P-BT

3.4.3.2 J48 决策树

WEKA 里的 J48 决策树模型是对 Ross Quinlan 的 C4.5 决策树算法的实现,并加入了比较好的剪枝过程, 有非常好的精度。[20]

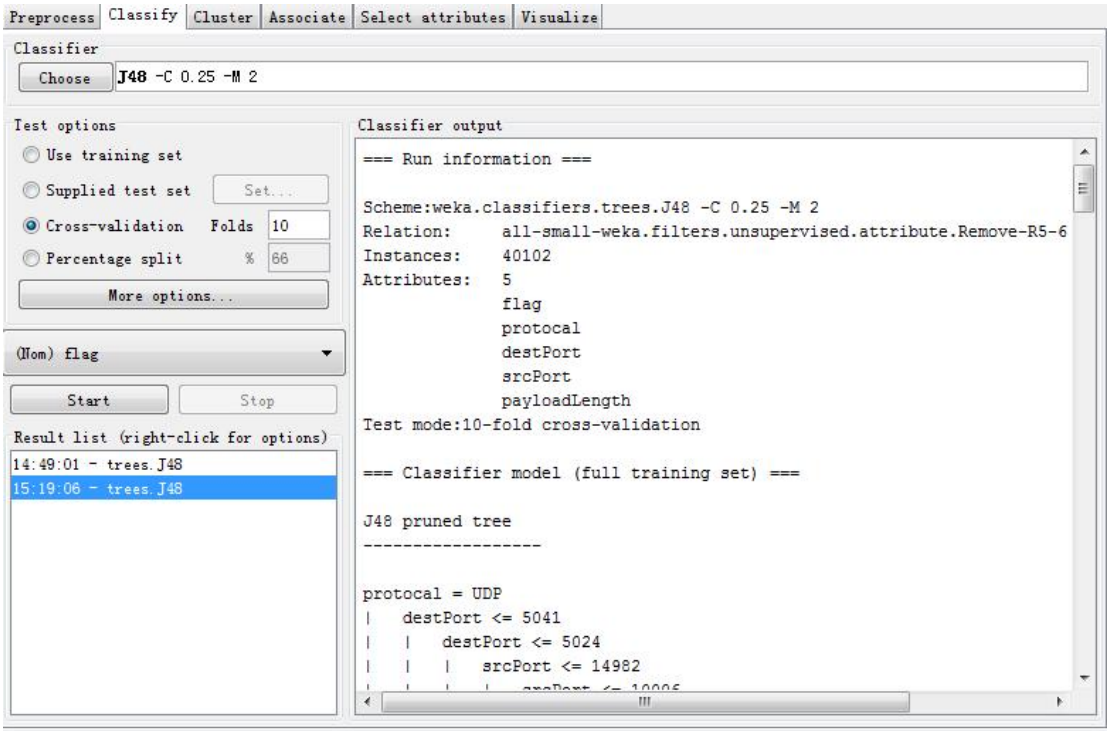


图 3.5 基于 Weka 的 J48 决策树流量识别过程

J48 决策树算法运行的结果为：

表 3.3 Weka J48 决策树网络流量识别结果

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.997	0.001	0.997	0.997	0.997	1	P2P-PPLIVE
0.994	0.001	0.997	0.994	0.995	0.999	P2P-XUNLEI
0.994	0.002	0.993	0.994	0.993	0.999	NON-P2P
0.994	0.003	0.992	0.994	0.993	0.999	P2P-BT

3.4.3.3 SMO 支持向量机

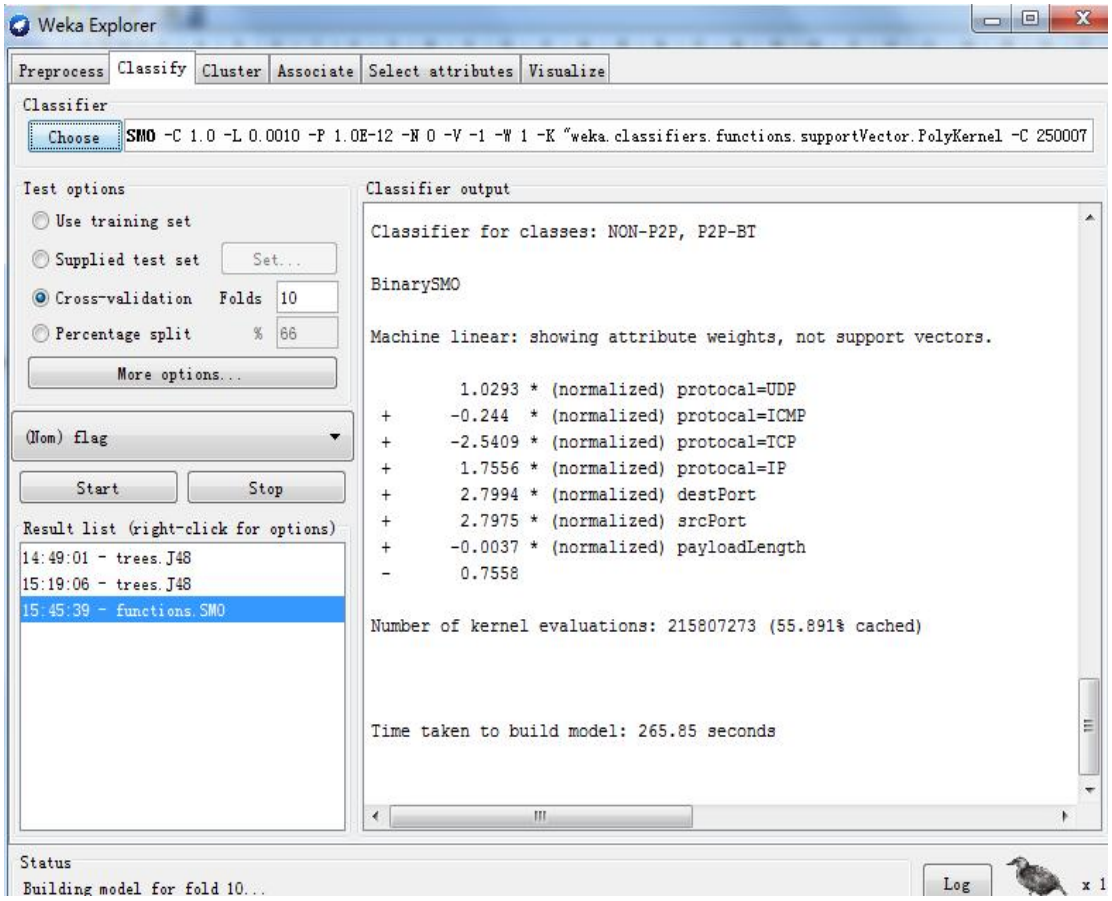


图 3.6 基于 Weka 的 SMO 支持向量机流量识别过程

使用 Weka 中 SMO 算法，加载我们采集的数据，得出的结果为：

表 3.4 Weka SMO 支持向量机网络流量识别结果

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.994	0.097	0.774	0.994	0.87	0.965	P2P-PPLIVE
0.108	0.033	0.528	0.108	0.18	0.572	P2P-XUNLEI
0.96	0.196	0.619	0.96	0.753	0.891	NON-P2P
0.717	0.083	0.741	0.717	0.729	0.879	P2P-BT

3.5 小结

通过基于 Weka 的 P2P 流量识别系统，其中利用三种不同的数据挖掘算法，得出对一组相同网络流量数据的识别结果。我们列出不同算法识别 F-Measure 值进行对比，见下表 3.5。

表 3.5 Weka 数据挖掘算法的网络流量识别结果

	朴素贝叶斯分类	J48 决策树	SMO 支持向量机
P2P-PPLIVE	0.996	0.997	0.87
P2P-XUNLEI	0.955	0.995	0.18
NON-P2P	0.946	0.993	0.753
P2P-BT	0.922	0.993	0.729

从中我们可以看出朴素贝叶斯分类和 J48 决策树两种算法，对流量识别具有很高的识别度和精确性。

同时我们在 Weka 执行算法的过程中发现，对于大数据量，超过 20 万条数据的分析过程，会占用大量内存，我们在这里分配了 6G 内存，但是在运行 J48 和 SMO 的过程中，都会出现 OutOfMemory 即内存不足的情况。

这表明基于 Weka 在单个机器上处理大规模数据，确实存在性能上的瓶颈。基于此存在的问题以及云计算的良好的可扩展性，下一章我们将提出一个基于云计算的改进的 P2P 流量识别系统。

第 4 章 基于云计算改进的 P2P 流量识别系统

4.1 云计算

云计算（Cloud Computing），是一种基于互联网的计算方式，通过这种方式，共享的软硬件资源和信息。

4.1.1 为何使用云计算

基于单机的数据挖掘，虽然也可以对网络流量进行分析，但是随着网络带宽的增加，网络流量数据将达到一个爆炸性的增长。根据思科公司发布的预测(Visual Networking Index (VNI) Forecast)，到 2017 年全球年度 IP 流量将会超过 1.4 zettabytes(1ZB = 1,000,000,000,000 GB) [21]

这样的话，流量识别分析工具面对的将是超大规模的数据，而传统的单点识别技术面对这些数据就会显得效率低下，并难以应付。

而云计算，基于它最主要的可扩展性的特性，将能很好的担负起这个任务。接下来对云计算及本文所用到的技术做一些介绍。

4.1.2 数据挖掘在云端的优势

云计算模式和数据挖掘各自有自身的特点，将二者结合就能产生极大的优势。优势如下：

计算资源按需申请并获得：将海量数据挖掘任务化，利用 Mapreduce 并行数据挖掘算法，基于 Hadoop 框架，使其具备任务自动分配运行以及资源灵活调度。

任务的分布式运行以及任务内计算资源动态分配：海量数据挖掘的计算任务一般需要消耗大量的计算资源；而且不同的挖掘任务队对算法、性能、实时性等需求不同。因而在云计算框架下任务的负载均衡调节的策略与任务迁移策略能够很好的服务于数据挖掘的特殊需求。

复杂数据挖掘任务服务平台：在 Hadoop 等云平台上，设计支持复杂数据挖掘任务服务化的中间件系统。支持复杂数据分析任务的流定义、复杂数据分析任务的动态配置、并行算法的注册、云平台资源的调度和管理的透明化，最终实现复杂数据分析任务的按需服务。

在来自 Stanford 大学的一篇论文中[20]，对一些常用数据挖掘算法按照 Map-Reduce 机制实现后，将算法复杂度在单核和多核之间做了对比。从下表中可以看出，在多核下，使用 Map-Reduce 机制，能有效的降低数据挖掘算法的复杂度。由于使用了 Map-Reduce 机制的算法能很容易的分布在多台机器上进行运算，所以可以得出结论，在云端进行数据挖掘，速度更快，更具扩展性。

	single	multi
LWLR	$O(mn^2+n^3)$	$O(\frac{mn^2}{P}+\frac{n^3}{P'}+n^2\log(P))$
LR	$O(mn^2+n^3)$	$O(\frac{mn^2}{P}+\frac{n^3}{P'}+n^2\log(P))$
NB	$O(mn+nc)$	$O(\frac{mn}{P}+nc\log(P))$
NN	$O(mn+nc)$	$O(\frac{mn}{P}+nc\log(P))$
GDA	$O(mn^2+n^3)$	$O(\frac{mn^2}{P}+\frac{n^3}{P'}+n^2\log(P))$
PCA	$O(mn^2+n^3)$	$O(\frac{mn^2}{P}+\frac{n^3}{P'}+n^2\log(P))$
ICA	$O(mn^2+n^3)$	$O(\frac{mn^2}{P}+\frac{n^3}{P'}+n^2\log(P))$
k-means	$O(mnc)$	$O(\frac{mnc}{P}+mn\log(P))$
EM	$O(mn^2+n^3)$	$O(\frac{mn^2}{P}+\frac{n^3}{P'}+n^2\log(P))$
SVM	$O(m^2n)$	$O(\frac{m^2n}{P}+n\log(P))$

图 4.1 常用数据挖掘算法在单核和多核上的时间复杂度

4.1.3 云计算简介

4.1.2.1 云计算的特征

云计算服务应该具备以下几条重要特征：按需获取资源，按使用付费，共享物理主机虚拟化，资源可重用并可灵活复制，资源管理和监控工具丰富，减少运维成本。

4.1.2.2 云计算的服务模式

如图 4.2，直观的展示了云计算所包含的 SaaS，PaaS 以及 IaaS 三种服务模式的服务内容。

4.1.2.3 云计算的应用模式

目前基于云计算的应用如下：云物联、云安全、云存储、私有云、云游戏、云教育、云会议、云社交、云统计分析等。

4.2 开源框架 Hadoop

4.2.1 Hadoop 简介

Hadoop 是 Apache 软件基金会（ASF）旗下的一个开源的分布式、具备可靠性以及可伸缩性的计算平台。[14]

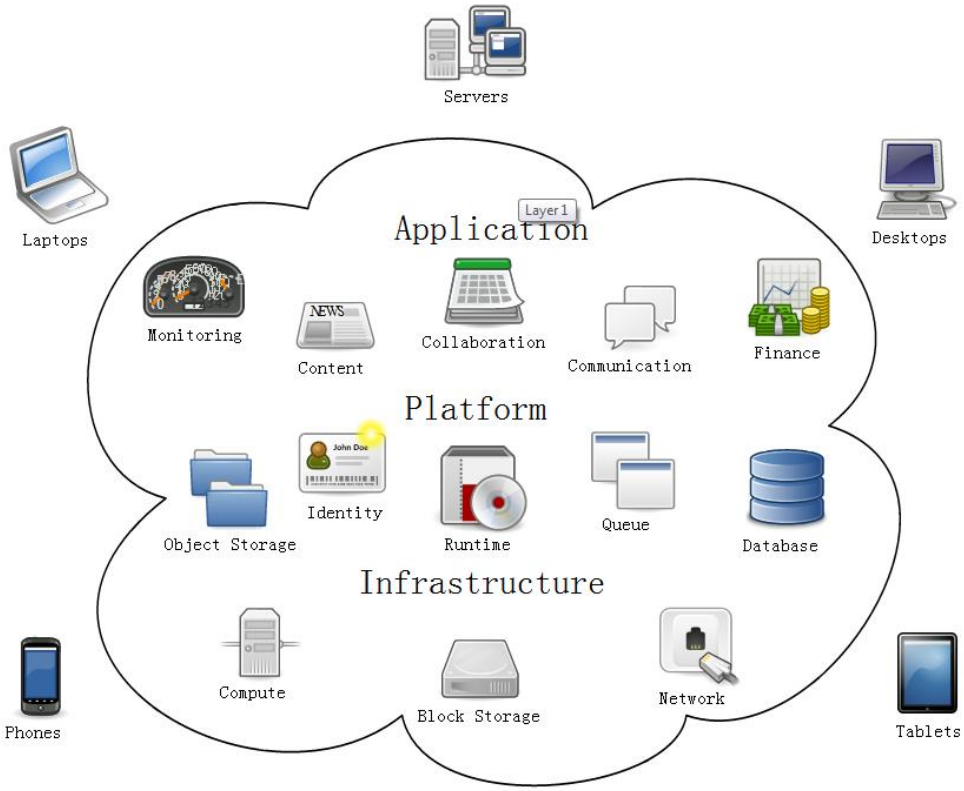


图 4.2 云计算的服务模式

Hadoop 是以 HDFS(分布式文件系统)和 MapReduce 为核心的 Hadoop 软件包，提供了一种可以基于简单的编程模型，在大规模集群计算机上进行分布式大数据处理的架构。Hadoop 架构被设计成可以利用从单台服务器扩充到成千上万服务器的计算和存储能力。Hadoop 架构不依赖于硬件去实现它的高可用性，它本身的设计就是在应用层探测并处理各种失效情况，从而保证了发布在 Hadoop 平台上的服务的高可用性。

Hadoop 作为一个 Apache 的顶级项目，其中包含了很多子项目，并且有很多相关项目是基于 Hadoop 的。如图 4.3 所示：

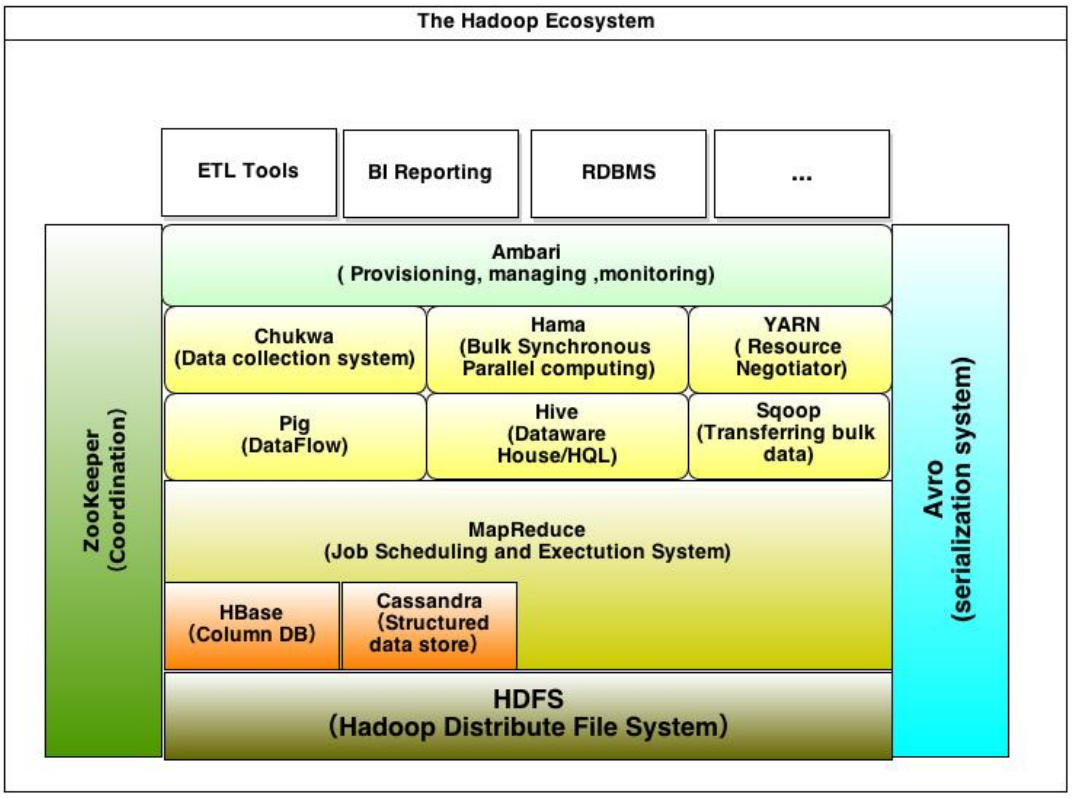


图 4.3 Hadoop 结构图

4.2.2 Hadoop 的优势

Hadoop 是一个开源的分布式计算平台。它有下边几个优势：高可靠性、高扩展性、高效性、高容错性。

- 扩展性好，可以很方便地增加节点，扩展计算能力，扩大存储容量
- 任务的分布式运行，计算资源动态分配
- 健壮性强，可自动处理失败节点，具有高容错能力

4.3 计算框架 MapReduce

Hadoop 分布式并行计算框架 MapReduce 是用来在集群环境中并发处理海量数据的具有高容错性的软件架构[2]。Hadoop 根据 Google 的论文实现了 MapReduce 这个编程框架，并完全开源。下边我们介绍一下 MapReduce 的计算模型.

在 Hadoop 中，每个 MapReduce 任务都被初始化为一个 Job。每个 Job 又可

以分为两个阶段：map 阶段和 reduce 阶段。Map 函数接收一个<key, value>形式的输入，然后同样产生一个<key, value>形式的中间输出。Hadoop 会负责将所有具有相同中间 key 值得 value 集合到一起传递给 reduce 函数，reduce 函数接收一

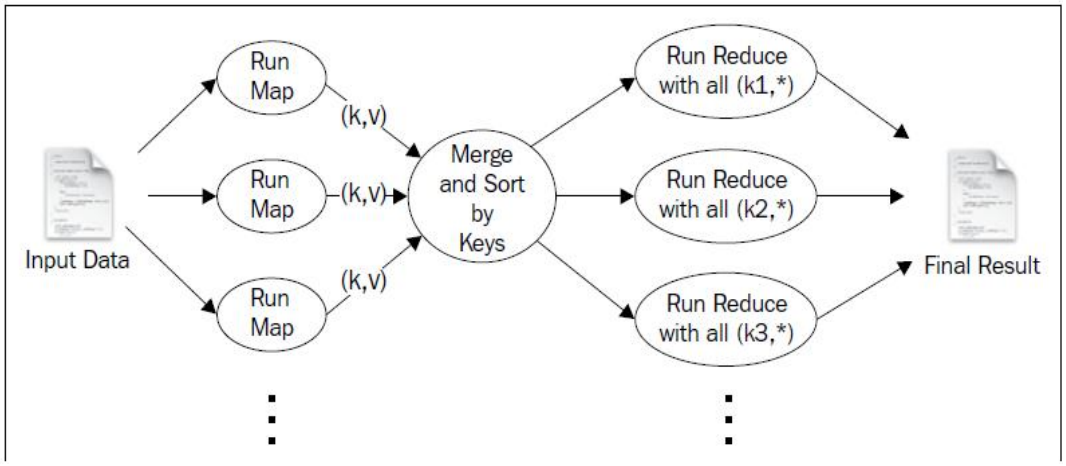


图 4.4 MapReduce 计算流程简图

个如<key , (list of values)>形式的输入，然后对这个 value 集合进行处理，每个 reduce 产生 0 或 1 个输出，reduce 的输出也是<key, value>形式的。如图 4.4 和图 4.5 所示，为 MapReduce 中 Map 阶段和 Reduce 阶段的数据处理流程图：

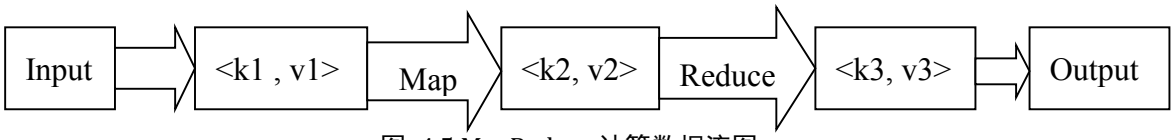


图 4.5 MapReduce 计算数据流图

图是 Google 关于 MapReduce 的论文[4]中给出的流程图。从最上方的用户程序（User Program）开始，用户程序和 MapReduce 库进行编译链接，实现了最基本的 Map 函数和 Reduce 函数。图中执行的顺序用数字标记。

4.4 针对大规模数据集的数据挖掘工具 Mahout

Apache Mahout 是 Apache 基金会下的一个开源软件。Apache Mahout™机器学习库的目标是构建可扩展的机器学习库。

目前 Mahout 的主要支持四个用例：挖掘用户的行为，试图找到用户可能会

喜欢的项目，进行推荐。基于文本文件，然后将他们分为局部相关文件进行聚类。然后从分类学习现有文档和分类文件产生一个特定类别的模型，从而使未标记的文件通过该模型能够分配到正确的类别。频繁项集挖掘项目组采用一组（查询会话中，购物车的内容），确定其中哪些项目通常一起出现。

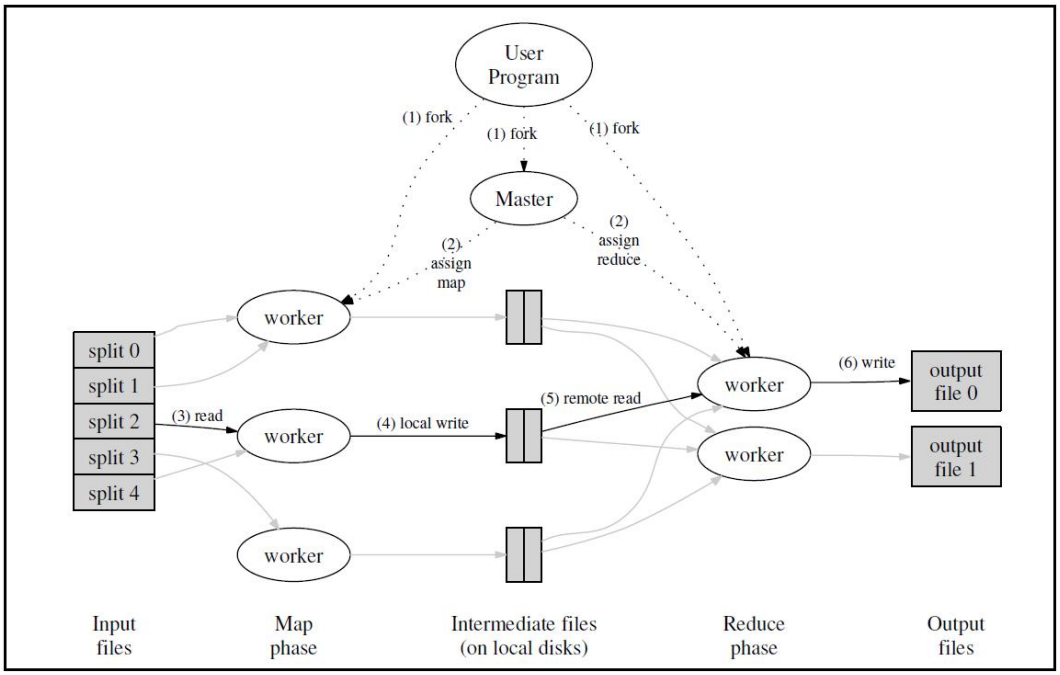


图 4.6 MapReduce 数据任务调度流程图

目前 Apache Mahout 项目主要包含以下算法：

- 协同过滤
- 基于用户和项目推荐系统
- k - means,模糊 k -均值聚类
- 均值漂移聚类
- Dirichlet process clustering
- Latent Dirichlet Allocation LDA
- 奇异值分解
- 并行频繁模式挖掘
- 互补朴素的贝叶斯分类器
- 基于随机森林分类器的决策树
- 高性能 java 集合(以前柯尔特集合)

使用 Mahout 创建数据集群的步骤包括：

1. 准备输入。注意，需要将非数值的属性转化为数值表示。
2. 使用 Mahout 中可用的集群算法。
3. 计算结果。
4. 如果有必要，执行迭代。

4.5 开源数据仓库工具 Hive

4.5.1 Hive 简介

Hive 是 Hadoop 中的一个重要子项目，其官方定义为：

Apache Hive 数据仓库软件可以方便的查询和管理在分布式存储系统上的大数据集。Hive 构建在 Hadoop 系统之上，提供下列功能：

1. 容易使用的数据 ETL (data extract/transform/load)工具；
2. 针对不同数据格式的强制统一的机制；
3. 直接访问 HDFS 或者 HBase 数据存储系统；
4. 通过 MapReduce 来执行查询。

4.5.2 Hive 工作原理

由于 Hadoop 是批量处理系统，任务是高延迟性的，所以在任务提交和处理过程中会消耗一些时间成本。同样，即使 Hive 处理的数据集非常小（几百兆），在执行时也会出现延迟现象。与传统的关系型大型数据库相比，Hive 不能提供数据排序和查询 cache 功能，也不能提供在线事务处理，不提供实时的查询和记录级的更行。但 Hive 能更好的处理不变的大规模数据集（例如本文所要研究的网络流量）上的批量任务。所以，Hive 最大的价值是可扩展性（基于 Hadoop 分布式计算平台）、可延展性（结合 MapReduce 和用户定义的函数库），并且拥有良好的容错性和低约束的数据输入格式。

Hive 本身建立在 Hadoop 的体系结构上，提供了一个 SQL 解析过程，并从外部接口中获取命令，以对用户指令进行解析。Hive 可将外部命令解析成一个 MapReduce 的可执行计划，并按照此计划生产 MapReduce 任务后交给 Hadoop 集群处理，Hive 的体系结构如图 4.7 所示。

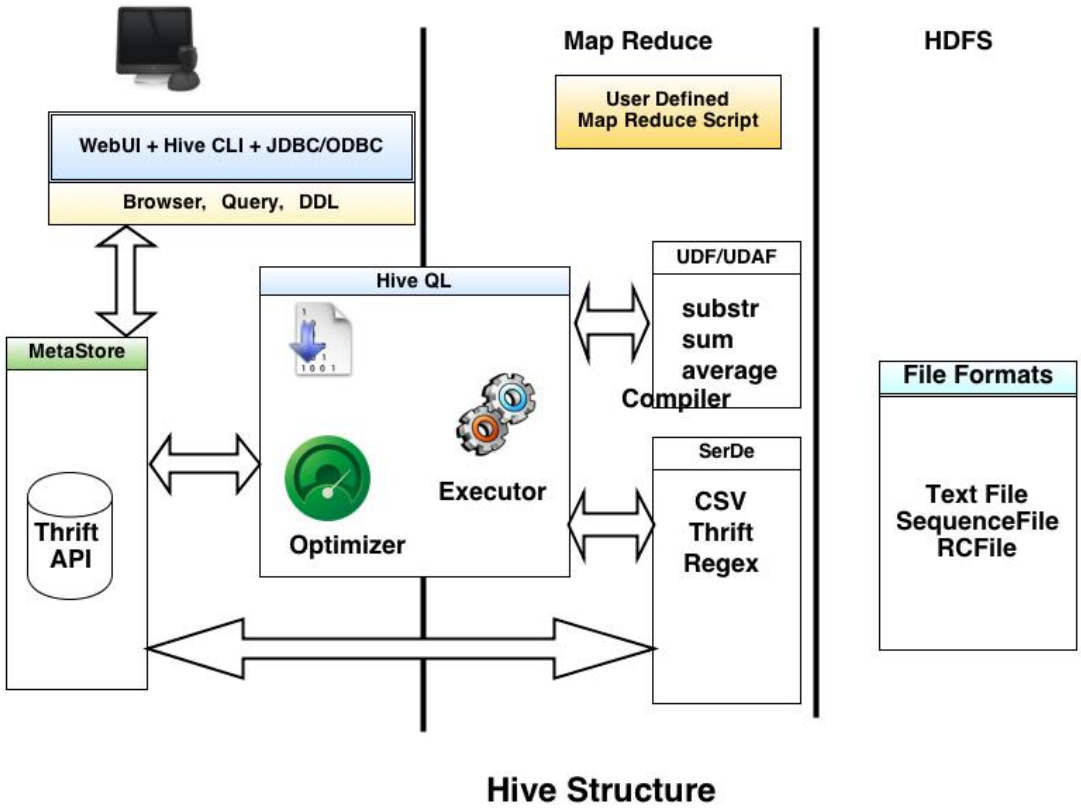


图 4.7 Hive 体系结构图

4.6 系统结构设计

基于云计算的并行计算框架对 P2P 流量进行识别的系统结构如图 4.8 所示。系统各个部分的工作原理如下：

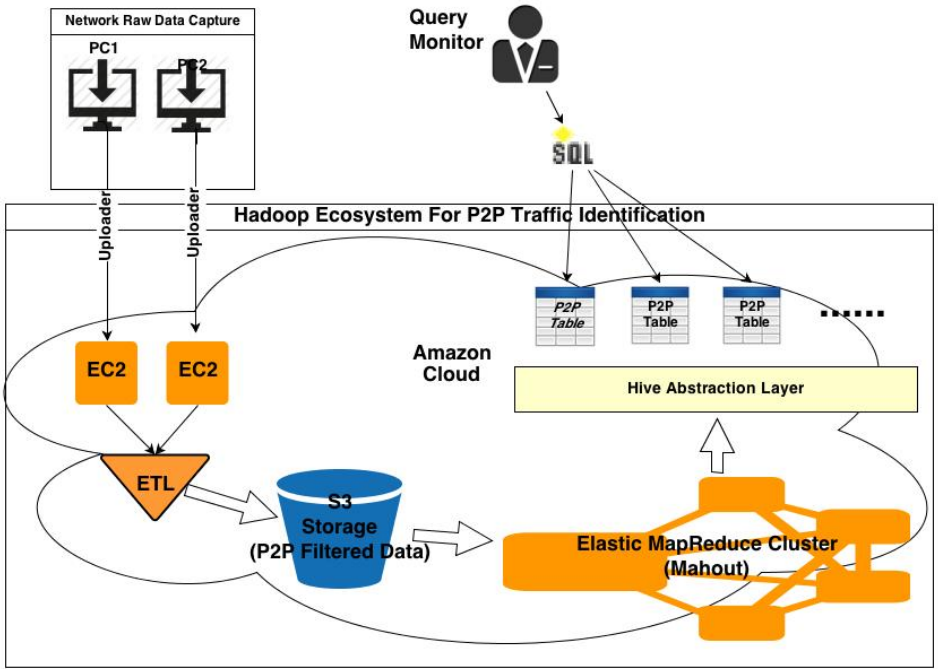


图 4.8 基于云计算的数据挖掘系统结构图

4.6.1 网络流量的采集

首先，通过基于 Pcap 的 Java 工具 JNetPcap，截获 PC 平台上的网络数据流。数据流的形式为，七元组（传输层协议，源 IP 地址，目的 IP 地址，源端口，目的端口, 大小，内容），例如：{协议类型:TCP 源 IP 地址:/192.168.1.103 目的 IP 地址:/140.211.11.4 源端口:57818 目的端口:80 大小:40 内容:abdcdefg…}

4.6.2 基于 Kafka 的网络流量数据的上传系统

通过一个流数据上传服务（Upload Service），将特定时间内截获的网络数据流，经过压缩上传到亚马逊的云平台上 S3 上（Simple Storage Service 简单存储服务）。此时数据文件的格式为原始数据格式（Raw Data），每一行为一个网络通信的七元组。我们使用 Kafka 消息系统，作为上传数据的主要解决方案。

Kafka 是一个消息系统，原先开发自 LinkedIn，用作 LinkedIn 的活动流（activity stream）和运营数据处理管道（pipeline）的基础。现在它已为多家不同类型的公司作为多种类型的数据管道（data pipeline）和消息系统使用。

传统的网络日志文件统计分析对报表和批处理这种离线处理的情况来说，是

一种很不错且很有伸缩性的方法；但是这种方法对于实时处理来说，延时太大，而且还具有较高的复杂度。另一方面，现有的消息队列系统（messaging and queuing system）却很适合于在实时或近实时（near-real-time）的情况下使用，但它们对很长的未被处理的消息队列的处理很不给力，往往并不将数据持久化作为首要的事情考虑。这样就会造成一种情况，就是当把大量数据传送给 Hadoop 这样的离线系统后，这些离线系统每个小时或每天仅能处理掉部分源数据。Kafka 的目的就是要成为一个队列平台，仅仅使用它就能够既支持离线又支持在线使用这两种情况。这就需要一套更加复杂的基础设施对其提供支持。下边详细说明 Kafka 在系统中的应用。

4.6.2.1 定义网络包

```
public class NetworktEntry {
    private long eventId;
    private long time;
    private String sourceIp;
    private String destIp;
    private int sourcePort;
    private int destPort;
    private String protocal;
    private byte[] payload;
}
```

4.6.2.2 建立 Producer

```
public ProducerFactoryExtImpl() {
        Properties sysProps = System.getProperties();
        Properties props = new Properties();
        props.put(SERIALIZER, NetworktEncoder.class.getName());
        ProducerConfig config = new ProducerConfig(props);
        producer = new Producer<String, NetworktEntry>(config) {
    }
```

4.6.2.3 发送网络数据包

```
public void logEvent(String topic, NetworktEntry entry) {
    Producer<String,NetworktEntry>producer=producerFactory.getProducer();
        if (entry.getEventId() == 0) {
            long eventId = idGenerator.getId(entry.getTime());
            entry.setEventId(eventId);
        }
    ProducerData<String, NetworktEntry> todo = new ProducerData<String,
    NetworktEntry >(topic, entry);
    try {
        producer.send(todo);
    } finally {
        producer.close();
    }
}
```

4.6.2.4 将网络包上传到 S3

利用多线程，将网络包上传的代码如下：

```
public void startConsuming() {
    directory.deleteLockFiles();
    directory.fixDataFileWithLogRange();
// create threads to consume from each of the partitions, and to upload event files
to                              s3.executorService                              =
Executors.newFixedThreadPool(s3ConsumerProperties.NUM_CONSUMER_TH
```

```
 RE
ADS +s3ConsumerProperties.NUM_UPLOADER_THREADS);
Map<String, Integer> topicMap = new HashMap<String, Integer>();
topicMap.put(s3ConsumerProperties.CONSUMER_TOPICS,
s3ConsumerProperties.NUM_CONSUMER_THREADS);
Map<String,List<KafkaMessageStream>>          topicMessageStreams          =
consumerConnector.createMessageStreams(topicMap);
List<KafkaMessageStream>                      streams                      =
topicMessageStreams.get(s3ConsumerProperties.CONSUMER_TOPICS);
// consume the messages in the threads
for (final KafkaMessageStream stream : streams) {
executorService.submit(new S3ConsumerCallable(directory,
s3ConsumerProperties.CONSUMER_TOPICS, stream));
}
for (int i = 0; i < s3ConsumerProperties.NUM_UPLOADER_THREADS; i++) {
      executorService.submit(new S3UploaderCallable(directory,
      s3ConsumerProperties.AWAIT_UPLOAD_FAIL));
}
```

4.6.3 网络流量数据预处理 ETL

进入云端（AWS Cloud）并存储在 S3 之上的 Raw Data 首先需要经过 ETL，ETL 是数据抽取（Extract）、清洗（Cleaning）、转换（Transform）、装载（Load）的过程。实现 ETL，首先要实现 ETL 转换的过程。它可以集中地体现为以下几个方面：

空值处理：可捕获字段空值，进行加载或替换为其他含义数据，并可根据字段空值实现分流加载到不同目标库。例如：在捕获的网络流量中，不含有 payload。

规范化数据格式：可实现字段格式约束定义，对于数据源中时间戳、数值、字符等数据，可自定义加载格式。例如将捕获时间，转换为 Unix 时间戳。

验证数据正确性：可利用网络端口以及 IP 分配规则，对捕获的数据进行校验。将那些无效数据过滤掉。例如：端口范围从 0~65535，并且过滤掉确定的非 P2P 流量。

数据替换：可实现无效数据、缺失数据的替换。

本系统中的数据预处理，举例如下，我们直截获基于 TCP/UDP 协议的 IP 数据包：

```
if (pcapPacket.hasHeader(Ip4.ID)) {
 pcapPacket.getHeader(ip4);
 protocal = "IP";
 payload = ip4.getPayload();
 payloadLength = ip4.getPayloadLength();
}
if (pcapPacket.hasHeader(Tcp.ID)) {
 pcapPacket.getHeader(tcp);
 srcPort = tcp.source();
 destPort = tcp.destination();
 protocal = "TCP";
}
if (pcapPacket.hasHeader(Udp.ID)) {
 pcapPacket.getHeader(udp);
 srcPort = udp.source();
 destPort = udp.destination();
 protocal = "UDP";
}
```

4.6.4 基于 Mahout 网络流量识别

从 S3 中读取经过 ETL 的数据，将数据交给基于 Mahout 的数据挖掘流量识别模块。分别是基于贝叶斯、以及决策树方法的识别模块。由于各个 Mahout 的数据挖掘算法是可以并行化的，所以这些识别过程通过 MapReduce 机制，可以并行运行，能极大地提高识别速度。注：由于 SVM 在 Mahout 开源工具中，仍然处在开发当中，并没有将开发中的 branch 合并的 trunk 上，所以 SVM 算法并没有实现并行化。本文暂时不对网络流量数据进行 SVM 挖掘。[16]

下边，对 Mahout 的贝叶斯算法以及决策树算法做一个简单的介绍：

4.6.5.1Mahout 贝叶斯算法：

Mahout 实现了 Traditional Naive Bayes 和 Complementary Naive Bayes，后者是在前者的基础上增加了结果分析功能(Result Analyzer).

主要相关的 Mahout 类:

```
org.apache.mahout.classifier.naivebayes.NaiveBayesModel
org.apache.mahout.classifier.naivebayes.StandardNaiveBayesClassifier
org.apache.mahout.classifier.naivebayes.ComplementaryNaiveBayesClassifier
```

Mahout 的 trainnb 调用的是 TrainNaiveBayesJob 完成训练模型任务。所在包：
org.apache.mahout.classifier.naivebayes.training
TrainNaiveBayesJob 的输入是在 tfidf 文件上 split 出来的一部分，用作训练。
TrainNaiveBayesJob 代码分析：

首先是一些命令行选项，决定贝叶斯算法的一些参数如

```
LABEL       -L

ALPHA_I   -a

LABEL_INDEX   -li

TRAIN_COMPLEMENTARY      -c
```

第二步从输入文件中读取 label，将 label 保存于 label index，其实也就是将分类建一个索引。例如 20news group 的例子，读取的 label 有两个，label index 如下:

```
Key class: class org.apache.hadoop.io.Text

Value Class: class org.apache.hadoop.io.IntWritable
```

第三步，将相同 label 的 vectors 相加。也就是将同一个类别的所有数据的 vector 相加。这里 vector 其实是一个 key/value vector，每项由数据的 id 和 tfidf 值组成。这样相加后就是一个一个类的 vector，相同 id 的 tfidf 相加，没有的则插入，类似两个递增的链表的合并。由一个 job 来完成：

```
// Key class: class org.apache.hadoop.io.Text
// Value Class: class org.apache.mahout.math.VectorWritable
//add up all the vectors with the same labels, while mapping the labels into our
index
```

```
Job indexInstances = prepareJob(getInputPath(), //input path
      getTempPath(SUMMED_OBSERVATIONS),            //output path
      SequenceFileInputFormat.class,               //input format
      IndexInstancesMapper.class,                  //mapper class
      IntWritable.class,                           //mapper key
      VectorWritable.class,                        //mapper value
      VectorSumReducer.class,                      //reducer class
      IntWritable.class,                           //reducer key
      VectorWritable.class,                        //reducer value
      SequenceFileOutputFormat.class);         //output format
    indexInstances.setCombinerClass(VectorSumReducer.class);
    boolean succeeded = indexInstances.waitForCompletion(true);
```

Mapper 为 IndexInstancesMapper，Reducer 为 Reducer VectorSumReducer，代码如下：

```
  protected void map(Text labelText, VectorWritable instance, Context ctx)
throws IOException, InterruptedException {

    String label = labelText.toString().split("/")[1];

if (labelIndex.containsKey(label)) {

     //从文件中读取的类的 index 作为 key

     ctx.write(new IntWritable(labelIndex.get(label)), instance);

    } else {

     ctx.getCounter(Counter.SKIPPED_INSTANCES).increment(1);

    }

  }

  //相同 key 的 vector 相加

  protected void reduce(WritableComparable<?> key, Iterable<VectorWritable>
values, Context ctx)
```

```
    throws IOException, InterruptedException {

    Vector vector = null;

    for (VectorWritable v : values) {

     if (vector == null) {

       vector = v.get();

     } else {

       vector.assign(v.get(), Functions.PLUS);

     }

    }

    ctx.write(key, new VectorWritable(vector));

    }
```

第四步：统计每个 label 的所有 ITIDF 权值和，输入为上一步的输出，并由一个 job 来执行：

```
//sum up all the weights from the previous step, per label and per feature
  Job weightSummer =
prepareJob(getTempPath(SUMMED_OBSERVATIONS),
                  getTempPath(WEIGHTS)
  weightSummer.getConfiguration().set(WeightsMapper.NUM_LABELS,
String.valueOf(labelSize));
  weightSummer.setCombinerClass(VectorSumReducer.class);
  succeeded = weightSummer.waitForCompletion(true);
```

最后一步，其实 model 有 weightsPerFeature 和 weightsPerLabel 就完成了。这一步也就是把它们变成矩阵形式，如下，每行一个权重 vector。

Item1 ,iterm2, item3…
lab1
lab2

代码如下：

```
//得到 SparseMatrix 矩阵

NaiveBayesModel naiveBayesModel =
BayesUtils.readModelFromDir(getTempPath(), getConf());

naiveBayesModel.validate();

//序列化，写到 output/naiveBayesModel.bin

naiveBayesModel.serialize(getOutputPath(), getConf());
```

4.6.5.2Mahout 决策树算法：

第一步：新建 treeBuilder，设置每次随机选择属性的样本个数，默认是所有属性的 1/3，设置 complemented，默认是 true 的，其他的属性参数基本也是默认的

DecisionTreeBuilder treeBuilder = new DecisionTreeBuilder();

第二步：新建 PartialBuilder，设置相关的参数，得到下面的 forestBuilder 的值如下：

Builder forestBuilder = new PartialBuilder(treeBuilder, dataPath, datasetPath, seed, getConf());

第三步：决策树的 build 方法，这个是重点：

DecisionForest forest = forestBuilder.build(nbTrees);

进入到 Builder 中的 build 方法中，看到是一些设置相关变量的代码：setRandomSeed、setNbTrees、setTreeBuilder。然后把 dataset 的路径加入到了 distributedCache 中，这样在 Mapper 中就可以直接读出这个路径了（相当于放在了内存中）。然后就是新建 Job 了，名字为 decision forest builder，初始化这个 Job 并运行：

setRandomSeed(conf, seed);

```
setNbTrees(conf, nbTrees);
setTreeBuilder(conf, treeBuilder);
// put the dataset into the DistributedCache
.addCacheFile(datasetPath.toUri(), conf);
Job job = new Job(conf, "decision forest builder");
log.debug("Configuring the job...");
configureJob(job);
runJob(job);
最终对决策树进行输出，供以后使用
DFUtils.storeWritable(getConf(), forestPath, forest);
在测试决策树的时候，使用以下代码：
    //首先确保输出路径是否已存在
      outFS = outputPath.getFileSystem(getConf());
      if (outFS.exists(outputPath)) {
    //确保决策树模型存在
    FileSystem mfs = modelPath.getFileSystem(getConf());
    if (!mfs.exists(modelPath)) {
      throw new IllegalArgumentException("The forest path does not exist");
    }
    //确保要分析的数据存在
    dataFS = dataPath.getFileSystem(getConf());
    if (!dataFS.exists(dataPath)) {
      throw new IllegalArgumentException("The Test data path does not
exist");
//如果要并行处理，用 mapreduce 函数，否则进行顺序处理
    if (useMapreduce)
      mapreduce();
    else
      sequential();
```

并行处理过程如下，首先基于决策树的模型以及相关参数创建一个分类器。

一些基本参数的设置主要有：输入、输出（这个是最基本的了）、dataset 路径、model 路径（BuildForest 的路径）、是否显示分析结果、是否采用 mapreduce 模式运行。

```
Classifier classifier = new Classifier(modelPath, dataPath, datasetPath, outputPath, getConf());
//运行此分类器
classifier.run();
//分别设置 dataset、converter、forest，从路径中把文件读出
dataset = Dataset.load(conf, new Path(files[0].getPath()));
converter = new DataConverter(dataset);
forest = DecisionForest.load(conf, new Path(files[1].getPath()))
```

下边是 Map-Reduce 中的 Map 函数，首先判断输入是否为空，否则由 converter 把输入的一行转换为 Instance 变量，然后由 setup 函数中读出来的 forest 去分析这个 Instance，看它应该是属于哪一类的，然后把 key 就设置为 instance 原来的分类，value 设置为 forest 的分类结果。这里最重要的操作其实就是 forest.classify 函数了。

```
protected void map(LongWritable key, Text value, Context context)
        String line = value.toString();
         if (!line.isEmpty()) {
            Instance instance = converter.convert(line);
            double prediction = forest.classify(dataset, rng, instance);
            lkey.set(dataset.getLabel(instance));
            lvalue.set(Double.toString(prediction));
            context.write(lkey, lvalue);
//forest.classify 函数如下，遍历决策树，对每一条数据做出分类。
public void classify(Data data, double[][] predictions) {
      int treeId = 0;
      for (Node tree : trees) {
         for (int index = 0; index < data.size(); index++) {
            if (predictions[index] == null) {
```

```
predictions[index] = new double[trees.size()];
            }
            predictions[index][treeId] = tree.classify(data.get(index));
        }
        treeId++;
```

4.6.5 基于 Hive 的数据仓库存储和查询

将基于 Mahout 的流量识别模块产生的结果，经过 Hive QL 转化为数据仓库，并提供接口供管理员查询当前分辨出的 P2P 流量。具体见下图 4.9：

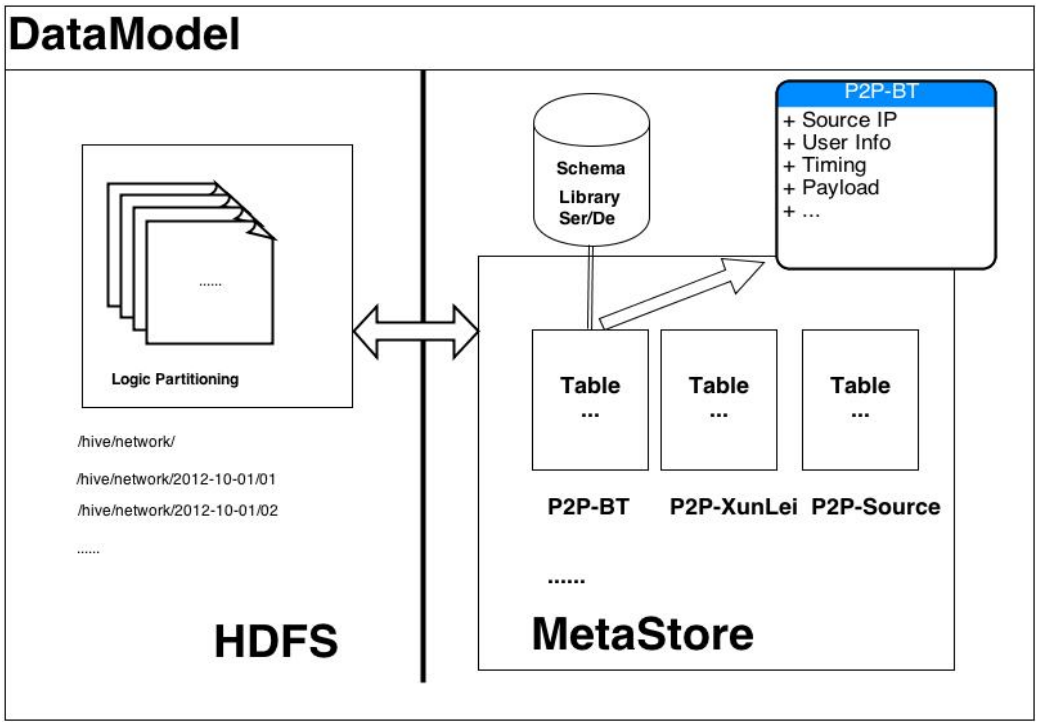


图 4.9 基于 Hive 数据仓库示意图

4.6.5.1 基于 Hive 的数据仓库

首先，建立 P2P-BT，P2P-Xunlei, Non-P2P 等 Hive 表（元数据）：

```
create table nonp2p_data(protocal STRING, srPort INT, destPort INT, sourceIP STRING, destIP STRING, lable STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;
create table xunleip2p_data(protocal STRING, srPort INT, destPort INT, sourceIP STRING, destIP STRING, lable STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;
create table otherp2p_data(protocal STRING, srPort INT, destPort INT, sourceIP STRING, destIP STRING, lable STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;
…..
```

第二步，加载经过分析过的数据

```
LOAD DATA LOCAL INPATH '/usr/data/otherp2p.data' OVERWRITE INTO TABLE otherp2p_data
LOAD DATA LOCAL INPATH '/usr/data/nonp2p.data' OVERWRITE INTO TABLE nonp2p_data
LOAD DATA LOCAL INPATH '/usr/data/xunleip2p.data' OVERWRITE INTO TABLE xunleip2p_data
```

第三步，管理员通过 Hive 工具，可以很方便的通过 Hive SQL 来进行查询。例如下方的 HQL，统计了同一时间段，迅雷 P2P 流量，针对不同目的 IP 的流量总和：

```
select sourceIP，collect_set(time)[0], sourcePort, sum(paylength) from xunleip2p_data
Group by destIP,
substr(from_unixtime(time,'yyyyMMddHHmmss'),9,2);
```

4.7 系统模拟实验和测试

4.7.1 实验环境

实验环境为(AWS)Amazon Web Service ，在 AWS 上申请 EC2 中型实例（M1

Medium Instance），共 5 台，配置如下：

3.75 GiB memory，

2 EC2 Compute Unit (1 virtual core with 2 EC2 Compute Unit)，

410 GB instance storage，

64-bit platform，

I/O Performance: Moderate，

EBS-Optimized Available: No

同时申请 S3 存储，作为网络流量 RawData 的存放位置。

在四台机器上部署并配置 Hadoop 环境，我们配置一台 master，作为运行 JobTracker 的 NameNode 节点，三台 Slave 作为运行 TaskTracker 的 DataNode 节点，第五台配置 Hive 环境。

4.7.2 实验过程

4.7.2.1 Naive Bayes Classifier 朴素贝叶斯分类

Mahout 目前有两种实现贝叶斯分类器。一个是传统的朴素的贝叶斯方法,和其他被称为互补朴素的贝叶斯(Complementary Naive Bayes)。

Mahout 中的 bayes 实现步骤以及贝叶斯算法的执行步骤如下：

首先，在 $MAHOUT_HOME 下的 examples/bin/ 下建立 work 文件夹（mkdir /home/hadoop/mahout-distribution-0.8/examples/bin/work/）上传网络流量数据集 all-network.arff，test-network.arff 到 work 文件夹下。

接下来，产生 Input 数据集，即对训练数据集进行预处理，数据准备阶段，将各类中的文件读入到一个大文件中，使得每类最后只有一个文件包含起初所有的文件，mahout 下处理的文件必须是 SequenceFile 格式的，还需要把 txtfile 转换成 sequenceFile。

命令为:

```
$MAHOUT_HOME/bin/./mahout org.apache.mahout.classifier.bayes.NetwokPreload \
-p /home/hadoop/mahout-distribution-0.8/examples/bin/work/Network-train \
-o /home/hadoop/mahout-distribution-0.8/examples/bin/work/Network-train-input \
-a org.apache.mahout.vectorizer.DefaultAnalyzer \
-c UTF-8
```

完毕后将 work 下的 bayes-train-input 放到 hadoop 的分布式文件系统上的 network-input，

在 hadoop/hadoop-0.20.2/bin 下输入命令:

```
hadoop dfs -put /home/hadoop/mahout-distribution-0.8/examples/bin/work/
bayes-train-input network-input
```

第三，用处理好的训练数据集进行训练得出分类模型即中间结果，模型保存在分布式文件系统上，在 mahout 的目录下输入命令:

```
$MAHOUT_HOME/bin/ ./mahout trainclassifier \
-i network-input \
-o networkmodel \
-type bayes \
-ng 3 \
-source hdfs
```

可以在查看 networkmodel 里的内容时，先查看其里面都有什么，命令：

```
hadoop fs -lsr /user/hadoop/networkmodel
```

将其导入到本地 txt 格式，进行查看，例如命令：

```
./mahout seqdumper -s
/user/hadoop/networkmodel /trainer-tfIdf//trainer-tfIdf/part-00000 -o /home/hadoop/out/part-1
```

最后，用模型进行测试，输入命令$MAHOUT_HOME/bin/ ./mahout testclassifier \

```
-m networkmodel \
-d network \
-type bayes \
-ng 3 \
-source hdfs \
-method mapreduce
```

4.7.2.2 决策树分类

我们按照以下步骤，对网络流量数据进行基于决策树的流量分类

第一步：将数据集放到 hdfs 上，以下为命令：

```
hadoop fs -mkdir testdata
hadoop fs -put allNetwork.arff testdata
hadoop fs -put testNetwork.arff testdata
```

第二步：生成数据集的解释文件，以下为命令：

```
../hadoop-0.20.2/bin/hadoop
jar
mahout-core-0.8-job.jar
org.apache.mahout.classifier.df.tools.Describe
-p testdata/all-network.arff -f testdata/all-network.info -d L C N N C C N
```

第三步：运行生成决策树

```
hadoop
jar
mahout-examples-0.8-job.jar
org.apache.mahout.classifier.df.mapreduce.BuildForest-Dmapred.max.split.size=
1874231 -oob -d testdata/allNetwork.arff
-ds testdata/allNetwork.info -sl 5 -p -t 100
```

第四步：使用决策树对测试数据分类

```
hadoop
jar
mahout-examples-0.8-job.jar
org.apache.mahout.classifier.df.mapreduce.TestForest -i testdata/testNetwork.arff
-ds testdata/allNetwork.info -m ob -a -mr -o predictions
```

第五步输出：

```
=======================================================
Summary
-------------------------------------------------------
Correctly Classified Instances          :       36854       91.9007%
Incorrectly Classified Instances        :        3248        8.0993%
Total Classified Instances              :       40102

=======================================================
Confusion Matrix
-------------------------------------------------------
a       b       c       d       <--Classified as
9979    21      0       0       |  10000   a     = NON-P2P
206     9888    4       2       |  10100   b     = P2P-XUNLEI
2903    41      7054    2       |  10000   c     = P2P-BT
67      2       0       9933    |  10002   d     = P2P-PPLIVE

=======================================================
Statistics
-------------------------------------------------------
Kappa                                     0.8916
Accuracy                                 91.9007%
Reliability                              73.5082%
Reliability (standard deviation)          0.4291
```

4.7.3 结果分析

下表为在不同时间，不同机器上截取的真实网络数据流量，进行的流量识别后的结果：

从结果中我们可以看出，随着数据规模的增大，网络流量数据的识别用时基本上和数据量成线性关系。这表明，基于云计算改进的 P2P 流量识别系统在性能上具有很好的伸缩性。

表 4.1 基于云计算改进的 P2P 流量识别系统识别结果

基于 Hadoop 云计算（4 运算节点）	平均识别率	数据规模 (40,102 条) 用时 min	数据规模 (3,000,000 条) 用时 min	数据规模 (16，000，000 条) 用时 min
贝叶斯算法	93.7	30s	3m27s	18m27s
决策树算法	95.2	50s	5m47s	27m37s

第 5 章 总结和展望

5.1 总结

从以上几章来看，要维护网络空间的正常秩序和正常应用，必然要首先实现对 P2P 流量的准确、高效的识别。而对 P2P 网络流量的准确、高效识别，作为本文的研究重点。

本文将流量识别看做分类问题，利用数据挖掘方面的分类算法应用到流量识别上。利用开源数据挖掘工具 WEKA 建立一个单机流量识别系统，对小规模的网络流量数据，可以做到较好的识别。另外一方面，网络流量识别从规模上来看，是一个大数据集以及超大数据集的存储和处理的问题。针对数据规模的问题，本文提出使用 Hadoop ， HDFS ， Mahout 三个开源系统共同组建一个分布式数据挖掘系统，处理并识别 P2P 网络流量。

本文提出一个基于数据挖掘的 P2P 流量识别系统的架构。包含了从网络流量采集、上传、存储、预处理、识别、结果查询分析等各个方面。基于架构设计，给出了基于 Weka 的 P2P 流量的识别机制。能够比较有效的识别出网络上的 P2P 流量。但是系统在面对大规模数据集的情况下，系统存在性能上的瓶颈。

针对性能上存在的瓶颈，提出了一个基于云计算改进的 P2P 流量识别系统。实验结果表明，利用这个分布式计算的数据挖掘系统，处理并识别 P2P 流量，在面对海量数据的时候，比基于 Weka 数据挖掘的 P2P 流量识别系统具有更高的性能。

通过实验结果表明，通过基于云计算改进的 P2P 流量识别系统，增加了系统的伸缩性和可扩展性。在数据量上，可以应对大规模网络数据流的采集和识别。在系统可用性上，优于基于 Weka 的单机流量识别系统。

此系统针对大规模网络流量数据的离线分析，具有较强的实用性和扩展性。相比较其他流量分类的技术与实验，基本上都是在小规模网络环境下进行的实验性质的测试。而基于本系统的伸缩性和可扩展性，系统可以处理大规模的网络流量数据，并给出相对较高的识别率以及较好人机查询接口和界面。

5.2 下一步工作

本文将 3 种识别方法有机地结合在一起，识别的准确率达到了 85%以上。

但由于时间和人力的原因，本系统不足主要表现在以下几点：

1、流量采集目前采用软件采集方法，采集并上传的同时会对现有网络造成影响；

2、流量识别部分，目前只采用了数据挖掘以及简单的基于端口的预处理；

3、本系统最终还没有放到实际的网络环境中去进行测试，只是按照同等级数据流进行了模拟实验。

下一步，将现有模型增加前置预处理模块，使用已有技术对流量进行预处理。此外，还将进一步完善监控和查询接口，使系统更加易用。

参考文献

[1] 陈云菁，P2P 流量识别技术的研究， 扬州大学， 2009 年 05 月

[2] http://www.cs.stanford.edu/people/ang//papers/nips06-mapreducemulticore.pdf

[3] http://csl.stanford.edu/~christos/publications/2011.phoenixplus.mapreduce.pdf

[4] MapReduce : Simplified Data Processing on Large Clusters. In proceedings of OSDI'04

[5] ADHUKARA, WILLIAMSON C. A longitudinal study of P2P traffic classification [C] Proceedings of 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Sep 11-13, 2006, Monterey, CA, USA. Los Alamitos, CA, USA: IEEE Computer Society, 2006:179-188.

[6] http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//archive/mapreduce-osdi04.pdf

[7] Wikipedia. http://en.wikipedia.org/wiki/Mapreduce MapReduce 算法说明

[8] Evaluating MapReduce for Multi-core and Multiprocessor Systems. In proceedings of HPCA'07.

[9] http://sortbenchmark.org/

[10] Accurate Classification of the Internet Traffic Based on the SVM Method. By Zhu Li

[11] http://aws.amazon.com 亚马逊云平台

[12] http://code.google.com/p/cascadesvm/ SVM classifier for multi-class classification on Hadoop (Cloud Computing)

[13] http://www.cs.waikato.ac.nz/ml/weka/ Weka 数据挖掘工具集

[14] http://hadoop.apache.org/

[15] http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf

[16] https://issues.apache.org/jira/browse/MAHOUT-232 关于 SVM 在 Mahout 上的说明

[17] http://static.usenix.org/events/osdi2000/full_papers/gribble/gribble_html/dds.html

[18] Ross Quinlan (1993). "C4.5: Programs for Machine Learning", Morgan Kaufmann Publishers, San Mateo, CA.

[19] [http://home.cnblogs.com/group/topic/40112.html]

[20] Cheng-Tao Chu ..*. CS. Department, Stanford University 353 Serra Mall Stanford University, “Map-Reduce for Machine Learning on Multicore” Stanford CA 94305-9025. Rexee Inc.

[21] [http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf]

致　谢

衷心感谢导师王鼎兴教授在论文期间对本人的精心指导，他的言传身教将使我终生受益。

我要感谢我的夫人王婧，她在我写论文期间担负起绝大部分的家庭劳动。

感谢我的父母，感谢我工作上的同事在学业及论文期间对我的宽容与支持。

最后衷心的感谢在百忙之中评阅论文和参加答辩的各位专家、教授！

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____日　期：_____

个人简历、在学期间发表的学术论文与研究成果

个人简历

1983 年 1 月 30 日出生于山西省大同市。

2001 年 9 月考入北方交通大学计算机科学与技术专业，2005 年 7 月本科毕业并获得工学学士学位。

2009 年 12 月考入清华大学计算机科学与技术系攻读工程硕士至今。

发表的学术论文