Massive Data Storage and Association Rule Mining of Electronic Health Recording Based on Cloud Computing

Dissertation Submitted to

Nanjing University of Technology

in partial fulfillment of the requirements

for the degree of

Master of Engineering

By

Jian Hou

Supervisor: Prof. Renjun Shuai

June 2011

摘 要

随着计算机的不断更新与发展，各行各业都引入计算机作为使用工具，在不断方便我们的日常的生活和工作的同时，随之而来的是产生了越来越多的数据。数据是信息的载体，随着信息化的不断发展，数据在现代社会生活中承担着越来越重要的角色。

医疗行业正在逐步迈进信息化的时代，随着新医改的发布，各地大中城市纷纷考虑建立电子健康档案数据中心，以及搭建以电子健康档案为中心的区域卫生信息平台。要建立一个真正以人为本的健康医疗体系，就要使医疗服务的成本、质量以及可及性得到平衡发展，任何一方面的缺失或者偏重都会使得整个医疗改革走样，甚至崩塌。南京市医疗信息化系统由南京市卫生局下的信息中心牵头，涵盖下属区县、社区的多家医院，数据中心设在信息中心数据机房。信息中心通过云计算技术整合现有的小型机、PC 服务器、存储设备、交换机、中间件等硬、软件设备，构建了一个易于扩展的居民电子健康档案平台。目的是为了实现南京市医疗信息的共享，实现医疗数据的统一管理，为医生提供实时的、集成的、可操作的数据，为患者提供准确诊断，从而降低成本，减少医疗事故。

南京市医疗信息化系统必将产生规模庞大的数据，而传统的存储方式 DAS、NAS、SAN，在面对网络产生的海量数据的时候，其缺点就明显的暴露了出来。DAS 存储方式可扩展性差，系统性能低，存储分散。NAS 虽然使用方便，成本低廉，但是存储性能最差。SAN 存储效能优异，能大幅提升网络上工作效能与资料传输效率，但是其架构为封闭式架构，无法整合不同系统，且规模过大成本较高。这对数据的存储和管理提出了挑战。

而另一方面，数据与数据之间存在联系，面对数据的多样性和复杂性、数据库的分布性和异构性以及服务对象的多样性和层次性，如果能够发现数据之间的联系，则可以从大量数据之间提取有用信息，从而准确预测病患的发展趋势，发病群体，发病年龄，以及发病时间，提高准确性。数据挖掘就是从大量数据中提取或挖掘知识的技术，关联规则挖掘发现大量数据项集之间有趣的关联，从大量的数据记录中发现有趣的关联关系，可以更方便的管理数据，制定决策。

基于以上分析，本文在云计算的基础上，依据云计算的核心计算模式 MapReduce，结合分布式存储系统的原理，提出一种基于 MapReduce 开源框架（Hadoop 平台）的分布式数据存储模型和挖掘关联规则的 Apriori 算法模型。数据的分块通过 Map 函数处理

以及 Reduce 函数合并，实现了数据的并行存储和有效关联。为共享医疗数据提供了可能性，为医生全面分析病人病情提供了实际性。

关键词： 云计算　MapReduce　数据挖掘　Apriori 算法　关联规则

ABSTRACT

As the development of computing, every walk of life uses computer as their necessary tool, computer has made a large convenient for our lives and works, but at the same time it also gives us large amount of datas. Data carries information, and as the development of informationization, data will play a very important role in our lives.

The medical trade is striding up to the era of informationization, after the issue of health care reform, lots of cities are thinking of constructing a data center of electronic health recording and a platform of area sanitation that gives priority to electronic health recording. To construct a health care reform aimed at people oriented, we should balance its costs and quality, and if one of them is weak, the medicare will be falling and aberrancy. The medical informationization of Nanjing is tractived by information center of board of health of Nanjing,it includes lots of hospital of communities and counties in Nanjing, the dispatch center of data lacate at IDC of the information center. The center assemble PC servers and memory devices together with hardware and software by cloud computing to construct easily expanding platform of electronci health recording. Its aim is to get to share the information and also uniform supervising, to supply real-time and compositive data for doctors and at the same time exact diagnose for sufferers. At last it can lose the charge that the sufferers have to pay for.

The system will bring out a large amount of data, but the traditional ways of storage including DAS NAS and SAN will expose its weakness when they confront massive data. The extended ability of DAS is low, and its performance is bad together with it's divert storage. The capacity of storage of NAS is very bad even though it has a lower price.SAN has an outstanding performance in storage and it could heigthen the speed of transmission, but its stuct is close and can not be conformitied.All that is made attention to give a large challenge in the storage and manage of data.

On the oter hand, it has relations between data, and because of the variety and complexity of data, the distributed and isomerous database and diversity and arragement of the serving object, finding the contact of data is very pre-requisite, we can distill the useful information between data, so we can predict the trading of the illness, the age of patient, the

time they will be ill and who wil be ill in the future. Data mining is a technology which can get knowledy from massive data, find the interesting association relationship in large amount of data, so we can manage data easier and make veracious decision.

By analysis of above, and based on cloud computing, a model of distributed data storage and the model of apriroi are brought out in this passage, its base is mapreduce which is the core compute pattern of cloud computing. Map here is used to analyse the blocks of data, reduce here deal with the results of map, all what they do are going to achieve the parallel storage and efficiency association. It provides possibility of sharing medical data and practicality of curing patient all around.

KEYWORDS: Cloud Computing; MapReduce; Data Mining; Apriori; Association Role

目 录

摘 要 I

ABSTRACT III

第 1 章 绪论 1

1.1 课题研究背景 1
1.2 课题来源与意义 2
1.3 论文主要工作 3

第 2 章 云计算及其关键技术 4

2.1 云计算技术背景 4
2.1.1 云计算的应用 4
2.1.2 Google 文档系统 GFS 5
2.1.3 Google 的 MapReduce 算法模型 7
2.1.4 Google 的 BigTable 数据库 8
2.2 Hadoop 平台框架 9
2.3 HDFS 分布式文档系统 10
2.4 Hbase 分布式数据库 12
2.5 Hbase 体系结构 13
2.5.1 Hbase 访问接口 14
2.5.2 Hregion Server 14
2.5.3 Hmaster Server 15
2.6 Hbase 存储格式 16
2.7 Hbase 数据模型 18
2.8 Hadoop 平台 MapReduce 模型 19

第 3 章 基于 HADOOP 的电子健康档案的数据存储模型设计 23

3.1 数据定义 23
3.2 文件分块策略分析 25
3.3 电子健康档案数据模型 28
3.4 元数据表模型 30
3.5 海量数据存储模型设计 31
3.6 存储模型的实现 33

第 4 章 基于 MAPREDUCE 的电子健康档案关联规则挖掘 36

4.1 医疗信息化数据特点 36
4.2 关联规则挖掘的定义 37
4.3 Apriori 算法 38
4.4 MapReduce 下的 Apriori 算法 40
4.5 性能分析 43
4.6 关联规则的产生 45

第 5 章　系统实施 46
5.1　硬件配置 46
5.2　软件配置 48
5.2.1　SSH 配置 48
5.2.2　Hadoop 平台搭建 49
5.2.3　HDFS 配置 51
5.2.4　Hbase 环境搭建 53
5.2.5　与客户端系统结合 54
5.3　运行结果与分析 55
5.3.1　不同数据规模的实验 55
5.3.2　不同任务粒度的实验 56
5.4　后续展望 57
参考文献 59
攻读学位期间科研成果 62
致　谢 63

第1章 绪论

1.1 课题研究背景

随着越来越多的人使用计算机，网络产生了数量巨大的数据，如何存储网络中产生的这些海量数据，已经是一个摆在面前亟待解决的问题。常用的存储介质[1]为磁盘和磁带，数据存储组织方式因存储介质而异。在磁带上数据仅按顺序文件方式存取；在磁盘上则可按使用要求采用顺序存取或直接存取[2]方式。数据存储方式与数据文件按组织密切相关，其关键在于建立记录的逻辑与物理顺序间的对应关系，确定存储地址，以提高数据存取速度。

数据以某种格式存储在计算机内部或外部存储介质上。数据存储是数据流[3]在加工过程中产生的临时文件或加工过程中需要查找的信息。数据存储要命名，这种命名要反映信息特征的组成含义。数据流反映了系统中流动的数据，表现出动态数据的特征；数据存储反映系统中静止的数据，表现出静态数据的特征。网络中常见的三种存储[4]方式是开放系统的直接式存储(DAS，Direct-Attached Storage)、网络附属存储(NAS，Network Attached Storage)和存储域网络(SAN，Storage Area Network)。三种方式各有其优缺点。

NAS 数据存储方式[5]是基于现有的企业 Ethernet 而设计的，按照 tcp/ip 协议进行通信，以文件的 I/O 方式进行数据传输。NAS 的优点：1、真正的即插即用，独立的存储节点存在于网络中，与用户的操作系统平台无关；2、存储部署简单，不依赖通用的操作系统，而是采用一个面向用户设计的，专门用于数据存储的简化操作系统，内置了与网络连接所需要的协议，因此是整个系统的管理和设置较为简单；3、存储设备位置非常灵活；4、管理容易且成本低。其缺点是：存储性能较低并且可靠性不高。SAN 的优势[6]在于：1、网络部署容易；2、高速存储性能，因为 SAN 采用了光纤通道技术，所以它具有更高的存储带宽，存储性能明显提高。SAN 的光纤通道使用全双工串行通道原理传输数据，传输速率高达 1062.5Mb/s；3、良好的扩展能力。由于 SAN 采用了网络结构，扩展能力更强。

从近几年频繁项集挖掘算法的研究趋势来看，为了提高算法的效率，国内外学者提出了一系列的混合搜索和高效剪枝策略[7]。当数据集中所包含的项目个数较多时，马占欣等[8]提出了只有恰当地设置 2 个额外参数，才能够保证挖掘过程的正常进行。但这样

做的代价是可能会遗漏部分包含更多负项目的关联规则。在基于 Apriori 算法性质的基础上，胡学钢等在《基于剪枝概念格模型的频繁项集表示集挖掘》一文[9]提出了机基于 CL 模型频繁项集求解算法，改善了频繁挖掘算法的时空性能，但是构造剪枝概念格需要花费一定时间。基于概念格[10]及其分布式关联规则挖掘算法及其在实际应用需要进行大量的研究，尤其是分布式概念的关联规则挖掘算法等都需要进一步深入研究解决。

本文提出的基于云计算的海量数据存储模型和电子健康档案的关联规则挖掘的算法模型，是依据云计算的核心计算模式 MapReduce，并依托实现了 MapReduce 计算模式的开源分布式并行编程框架 Hadoop，将存储模型和云计算结合在一起，从而实现海量数据的分布式存储和数据之间关联规则的挖掘。

1.2 课题来源与意义

本课题来源于南京市卫生局下的南京市卫生信息中心申请课题：研究十一五国家科技支撑计划项目区域医疗卫生综合信息系统开发与运用示范（基金号为：2010BAI88B00），课题名称为云计算在居民电子健康档案平台中研究，作者主要负责云计算平台的搭建与电子健康档案的管理工作。

电子健康档案记录的是居民的基本信息、就诊和用药，检查记录等。大型医院之间将首先联网共享居民健康档案。去医院看病，患者只要提供一个 ID 号，其家庭成员情况、病史、历次诊断情况等均可在不同医院共享。电子健康档案将每个人从生到死的健康记录、病例完整的记录和保存下来，在合适的时候能够被诊疗机构，大型医院合理的利用。甚至医疗保险机构、政府监管部门，甚至药厂、科研单位都可以用到这些信息。对于人员流动密集的地区，电子健康档案能确保信息方便快捷准确的在任何时间、任何地点被存取和利用。

在南京市展开试点，采用云计算的技术，可以覆盖支援南京市各区县的电子健康档案及社区应用的建设，以及支援医疗 IT 投入不足的地区的信息化建设。基于云计算网络环境的电子健康档案将医疗机构的系统和设备通过互联网或专用 VPN 互联起来，大幅度减少运行成本，并提高医疗资源的使用率。同时，通过 SaaS 的创新模式向医疗机构和个人提供一整套在线服务，包括电子健康档案，注册预约等，大大缩减医疗机构的投资，同时可为病人提供便利。

1.3 论文主要工作

本文主要研究的内容有以下几个方面：

1）医疗信息化数据采用传统关系型数据库处理数据存在的问题以及解决这些问题首先需要考虑的各方面的因素。

2）Hadoop 平台搭建，名字节点与数据的节点的配置。HDFS 的数据容错机制，工作粒度的确定。

3）MapReduce 编程模型的基本原理与具体实现，医疗信息化数据存储与关联规则挖掘的 MapReduce 算法模型的设计与实现。

4）分析基于云计算的数据存储模型和关联规则挖掘方法，与传统的存储方法和关联规则挖掘方法比较，验证算法的可行性。

Hadoop 平台[11]将并行计算“平民化”，它对程序员屏蔽了并行应用开发的细节，程序员只需要将更多的放在业务逻辑上即可。另外 Hadoop 平台只需要普通 PC 机群就可以运行，大大节约了开发成本。这也就让个人实施、研究分布式系统更加简单。

第 2 章 云计算及其关键技术

云计算是一种计算模式，也是一种全新的商业模式[12]。云计算(Cloud Computing)是分布式处理(Distributed Computing)、并行处理(Parallel Computing)和网格计算(Grid Computing)的发展或者说是这些计算机科学概念的商业实现。云计算是随着网络中产生的越来越多的数据而被提出的，在云计算中，无数的软件和服务都置于云中，这里的云是指可以自我维护和管理的虚拟计算资源[13]。这些软件和服务均构筑于各种标准和协议之上，可以通过各种设备来获得。云计算是一种超级的计算模式，可以把网络中的计算机虚拟为一个资源池，将所有的计算资源集中起来，并用特定软件实现自动管理，使得各种计算资源可以协同工作，这就使得处理数量巨大的数据成为了可能。

MapReduce 是云计算的核心计算模式[14]，是一种分布式运算技术，也是简化的分布式编程模式，用于解决问题的程序开发模型，也是开发人员拆解问题的方法。MapReduce 模式的主要方法是将自动分割要执行的问题(例如程序)，拆解成 Map(映射)和 Reduce(化简)的方式。在数据被分割后通过 Map 函数的程序将数据映射成不同的区块，分配给计算机机群处理达到分布式运算的效果，在通过 Reduce 函数的程序将结果汇整，输出要得到的结果。

Hadoop 是一个实现了 MapReduce 计算模型的开源分布式并行编程框架[15]，程序员可以借助 Hadoop 编写程序，将所编写的程序运行于计算机机群上，从而实现对海量数据的处理。此外，Hadoop 还提供一个分布式文件系统(HDFS)及分布式数据库(Hbase)用来将数据存储或部署到各个计算节点上。

2.1 云计算技术背景

2.1.1 云计算的应用

云计算技术作为一种创新的基础架构管理方法，通过一个公共服务的平台，动态灵活的监控和部署资源，从而提高资源的使用率，为各行各业带来了一种创新的解决方案，并成为未来 IT 管理的趋势。

2006 年底，Google 第一次提出了“云”的概念[16]，为我们更好的处理网络中产生的海量数据带来了希望。Google 与 IBM 在 2007 年底耗资数千万美元，创建了一个大型

的数据中心，并提出了云计算的概念。随后，很多大型的公司相继推出了自己的云计算服务。Yahoo 参与了 Apache Hadoop 计划，这是一个开放源代码的计划，用于海量数据的处理。另外，Yahoo 将开源云计算框架 Hadoop 应用在自家搜索服务的两千台服务器上，来处理超过 5PB 的网页属性，创建整个互联网的网页索引数据。亚马逊公司把云计算应用与 Web Services，通过虚拟化技术，让 Amazon EC2 提供计算服务，再搭配 Amazon S3 存储服务，提供各种不同规格的虚拟主机和存储空间，使用户和软件开发者根据自己的需要在上面安装或运行所需的程序。微软公司则创建了 Microsoft Azure 云计算操作系统，Microsoft Azure 并不是替换原有机器上的 Windows 操作系统，而是一个开发平台替代方案。Windows Azure 主要是让开发人员所编写是程序直接在微软的数据中心运行，而不用靠本地端的服务器。IBM 主推的蓝云计划主要专注于提供云计算服务所需拥有的各种硬件设备与管理软件，允许企业将计算任务分成不同的组件，将服务器集群与计算机系统调整到最高的运行效率，以解决企业尖峰或离峰时间的系统负载问题。升阳发表的开放式云计算平台，包括多项内核技术、API 与协议，并将提供云服务器和存储等硬件服务，支持在 Sun Virtual Box 虚拟软件以及 Windows、Solaris、Linux 等操作系统环境中运行应用程序。

云计算提供了一种在虚拟环境中开发应用程序的方法[17]，在虚拟云计算环境中，我们可以开发、部署和管理应用程序，根据不断变化的业务需求调整应用程序。云计算最重要的概念之一就是可伸缩性，实现它的关键则是虚拟化(virtualization)。虚拟化就是在一台共享计算机上聚集多个操作系统和应用程序，以便更好地利用服务器。虚拟化允许在线迁移，因此，当一个服务器负载超荷时，可以将其中一个系统的一个实例(以及它的应用程序)迁移到一个新的、相对闲置的服务器上。从外部看，云计算只是将计算和存储资源从企业迁出，并迁入到云中。用户定义资源需求[18] (例如计算和广域网、带宽需求)，云提供者在它的基础设施中虚拟地装配这些组件。放弃对自己资源的控制，而让其虚拟地存在于云中最重要的两个原因是成本和可伸缩性。因此云计算可以看成是网格计算与虚拟化技术的结合。利用网格的分布式计算能力将各种 IT 资源筑成一个资源池，然后结合成熟的存储虚拟化和服务虚拟化技术，让用户实时透明地监控和调配资源。

2.1.2 Google 文档系统 GFS

Google 文档系统[19] (GFS，Google File System)主要用来处理云计算的数据迅速增长问题，GFS 文档系统是一个可扩展的分布式文件系统，用于大型的、分布式的、对大量

数据进行访问的应用。GFS 具备分布式文档系统的所有特性，包括存储效率、可伸缩性、可靠性以及可再用性等。大型的 GFS 分布式文档系统不需要使用昂贵的高阶存储设备，它的存储设备可由几千个甚至几万个普通的硬盘串联而成，同样可以达到维持文档的高质量存储的目的。GFS 所处理的数据量通常都是 GB 级别的，有时甚至高达 TB 级，主要是因为 GFS 的文档系统更改不同于其他的系统。GFS 文档系统中的文档更改不是覆盖原来数据的操作，而是在原有的文档中追加新数据，所以造成 GFS 文档系统数据量越来越大。GFS 具备容错功能，它运行在普通的 PC 机上，当实时操作过程中发生应用程序 bug、网络中断或者人为造成的突然断电和操作系统问题，都可以通过自身的容错功能检测以及自动恢复系统将损毁的文档。图 2-1 是 GFS 的运作架构。

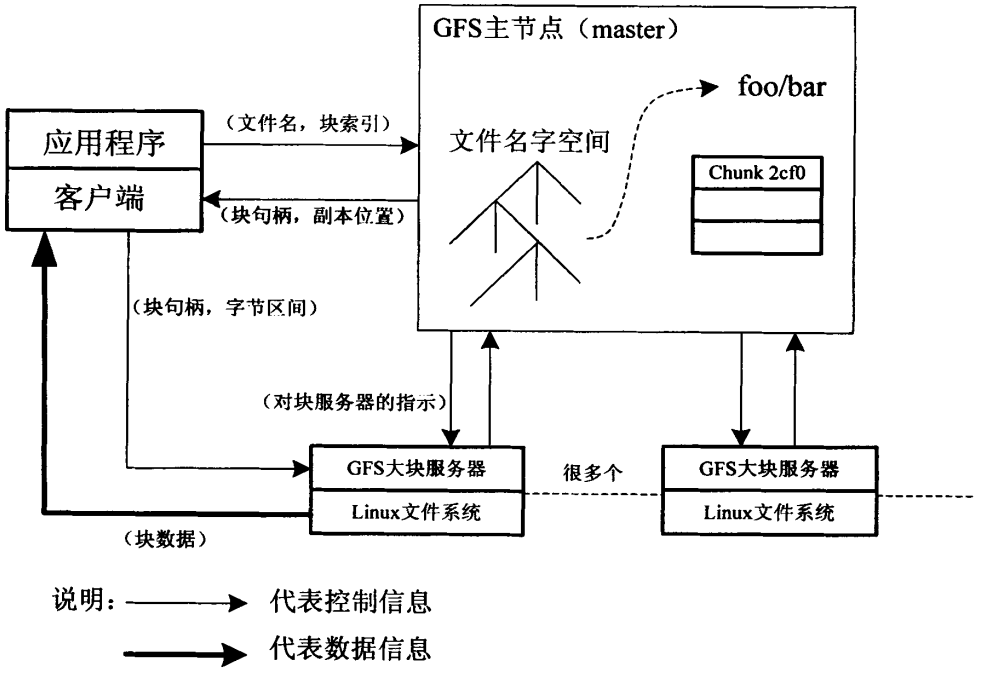

图 2-1 GFS 运作架构

Fig.2-1 The Operational Architecture of GFS

如图 2-1 所示，一个 GFS 的运作架构包含一个 master 主服务器和多个 chunkserver 服务器，chunk 在这里指将大文档切割成多个小文档块。master 和 chunkserver 通常是运行用户层服务进程的 Linux 机器，当资源和可靠性允许时，chunkserver 和 client 可以运行在同一个机器上。在 GFS 系统中，提供类似传统文档系统的操作方式包括创建、删除、打开、关闭、读/写以及目录层次组织等。

当大量的 GFS Client 客户端应用程序向 GFS 主服务器 master 发出访问请求时，GFS 主服务器将这些请求文件化分成固定大小的块，每个块的大小是 64MB。每个块由一个

不变的、全局唯一的 64 为的 Chunk Handle 标识，Chunk Handle 是在创建时初由 master 分配的，然后分配给底下大量的 chunkserver 去处理。chunkserver 将块当做 Linux 文件存储在本地磁盘并可以读和写出 Chunk Handle 和位区间指定的数据，GFS 文档系统为了确保文档的安全，每一个块被复制到多个 chunkserver 上，默认情况下，会将这些 chunk 数据自动保存到至少三个以上的 chunkserver 服务器上，进行三份以上的的备份操作，副本个数可以由用户自己定义。

在 GFS 运作架构中[20]，master 主服务器用来维护文件系统所有的元数据，包括文档的 namespace 名字空间、分割后的 chunk namespace，以及从文件到块的映射以及块的当前位置。存储在主服务器 master 上的元数据通常只有几十个 MB，每一个文档平均只有 100 个字符，而真正的文档则保存在底下的 GFS chunkserver 中，从而更好的提高了数据的可靠性。同时，GFS master 主服务器也控制系统的活动，如块租约管理，孤块的垃圾收集，chunkserver 间的块迁移。master 通过定期心跳消息控制所有 chunkserver，给 chunkserver 传递指令并收集它的状态。

2.1.3 Google 的 MapReduce 算法模型

Google 运用 MapReduce 关键技术处理云计算中的大量数据，可以在数千部服务器上同时展开不同的业务。MapReduce 算法模型是一种简化并行计算的编程模型[21]，它向上层用户提供接口，屏蔽了并行计算特别是分布式处理的诸多细节问题，从而使并行计算变的简单。MapReduce 算法模型[22]包括两部分，数据首先通过 Map 程序切割成各类型的小数据块，然后这些数据块被分配给大量的服务器处理，最后通过 Reduce 程序将处理后的结果汇整输出给客户端。在 MapReduce 架构中，技术人员通过 Map 快速选定数据后，在许多服务器上划分数据，并使同步处理划分后的每个部分，再集合这些中间数据，借由 Reduce 简化步骤，输出最终的简化结果。图 2-2 是 MapReduce 运作架构。

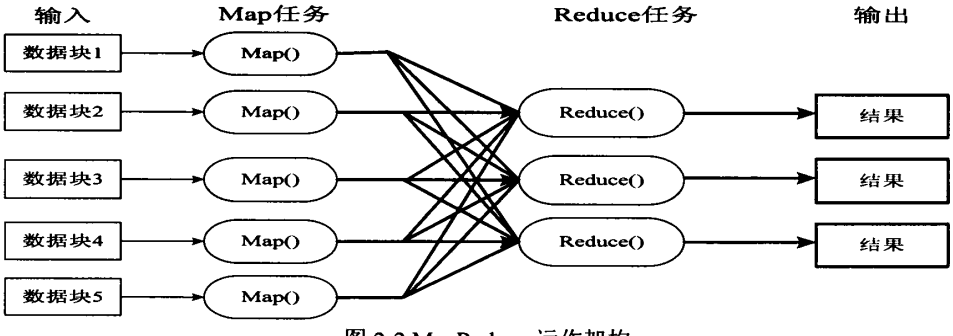

图 2-2 MapReduce 运作架构

Fig.2-2 The Operational Architecture of MapReduce

MapReduce 算法模型包括两个函数，Map 函数和 Reduce 函数。Map 函数负责从原始数据中整理分类出中间数据，Reduce 函数简化这些中间数据。MapReduce 的作用是对计算集群上的大型数据进行分布式计算，替大量数据做并行计算处理。

Map 函数和 Reduce 函数需要使用 MapReduce 程序模型的开发者自己编写实现程序，Map 函数是一个从(key,value)对应到 list(key',value')的函数[23]，Reduce 函数负责将 Map 函数输出的相关性 intermediate key 合并，然后输出最后的 key/value 键值对统计结果。

Map 函数原型如下：

map(key,value)->list(intermediate_key,intermediate_value) (2.1)

Reduce 函数原型如下：

reduce(intermediate_key,list(intermediate_value))->list(out_key,out_value) (2.2)

Map 函数输入的是一个 key/value 键值对，输出则为另一组 intermediate key/value 中间键值对组。简单地说，对 Map 函数输入一串数据后，函数会将数据做分类处理，然后把每一个 key 与每一 value 数据配对好。Reduce 函数将许多相同的 intermediate key 的 value 汇整后，输出最后的结果值 out_key 与 out_value。Map 函数是一个映射函数，而 Reduce 函数是一个化简规约函数。对于 Map 映射函数，比如对[2,4,6,8]进行平方的映射就变成了[4,16,36,64]。Reduce 是对生成的临时中间键值对进行规约，这个规约的规则由一个用户自定义 Reduce 函数操作指定，输出最终结果。比如对[2,4,6,8]进行乘积规约最终输出 384。

2.1.4 Google 的 BigTable 数据库

BigTable 数据库是 Google 为了搜索需求所开发的技术[24]。Google 用 GFS 文档系统存储数据，用 MapReduce 分布式简化数据，用 BigTable 数据库放置。各种分布式资料分布式系统的一个基本功能就是能够对海量数据在物理上进行分布式的存储，而在逻辑上提供一个透明、一致的文件系统操作接口，存储算法提出的目的就是为了实现这个分布式存储的目的。不同的分布式系统在存储算法的设计和实现形式上各有差异。

BigTable 分布式数据库专门用来处理 Petabyte(1PB=1000TB)等级的结构化数据，是设计来分布存储大规模结构化数据的[25]，分布存储在成千上万的各式各样的应用服务器集群中。如搜索功能的网页索引、Google Earth、Google Finance、Google Analytics 等，这些应用对 BT 的要求各不相同，有的需要高吞吐量的批处理，有的需要快速反应给用户数据。它们使用的 BT 集群也各不相同，有的只有几台机器，有的有上千台，能够存

储 240 字节(terabytes)数据。

BigTable 分布式数据库查询数据所用的语言为 GQL(Google Query Language)，GQL 是一种 SQL-like 数据库语言[26]，且 Bigtable 可以配合 MapReduce 技术架构运行复杂的分析与查询功能，特别适合应用于 100TB 以上的集成数据库表格访问。BigTable 数据存储方式是 column-oriented，而不是传统关联式数据库的 row-oriented 方式，column-oriented 的好处是每一个数据可以存放不固定字段的对象。byte[]是 BigTable 唯一的数据形态，每个数据都以下列格式保存：

(Row, Column, Timestamp) -> byte[] data。

BigTable 的第一个字段是 Row key，第二个字段是 Column key，第三个字段则是以 Timestamp 时间戳作为数据索引。Row key 与 Column key 的数据型态都是 string 字符串，而 Timestamp 的数据型态则是 64 位整数。这三个参数所对应的 data 数据也是以字符串的方式存储的。

2.2 Hadoop 平台框架

Hadoop 是 Apache 开源组织的一个分布式计算开源框架[27]，它可以将数据和处理程序分散到上百台甚至上千台计算机上同时运行，每一台负责运行任务的计算机，被称为一个节点。本质上 Hadoop 是 Google Map/Reduce 算法模型的一个 Java 实现，它让程序自动分布到一个由普通机器组成的超大集群上并发执行。Hadoop 分散处理数据文档，可以让文档的处理速度变得非常快速。它可以将计算的程序和放置的数据在每一个节点间进行自动复制和备份操作，来避免运行中的程序或存放的数据因为某一个节点计算机的硬件或操作系统损坏而使文档消失或损毁。Hadoop 所使用的分布式文档系统与数据库是 HDFS（Hadoop Distributed File System 与 Hbase，与 Google 的 GFS 和 BigTable 并不相同，因此 Hadoop 是真正运作 MapReduce 云计算技术的开放源代码云计算系统，图 2-3 是 hadoopd 云计算架构。

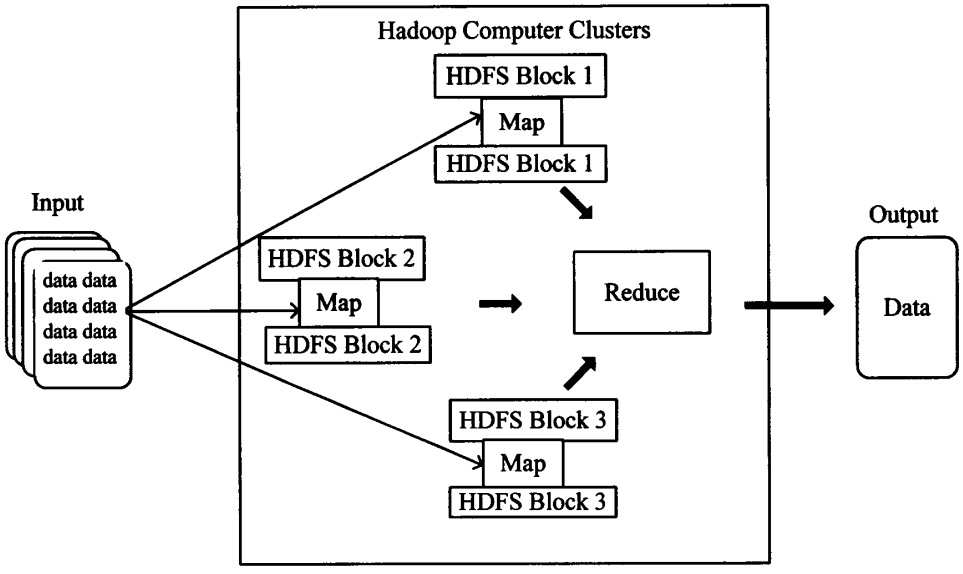

图 2-3 Hadoop 的云计算架构

Fig.2-3 The Could Computing Architecture of Hadoop

Hadoop 开源云计算平台[28]包括 HDFS 分布式文档系统、MapReduce 分布式平行计算框架以及 Hbase 分布式数据库，这正好与 Google 的 GFS、MapReduce 与 BigTable 三大技术类似。Hadoop 除了 HDFS、MapReduce 与 Hbase 以外，还包括 Avro(处理动态数据的 Script language)、Chukwa(大型分布式数据管理系统)、Hive(分布式文档基础架构)、Pig(应用与平行计算的程序语言)等子技术架构。

当前 Hadoop 云计算系统可运行于 Linux、Mac OS/X、Windows 和 Solaris 等一般操作系统上，通过通用的 java 程序语言来开发程序，在客户端提供诸如 C++、Java、Shell Command 等语言的开发访问界面。在程序开发工具上，可以使用开放源代码的集成开发环境 Eclipse 搭配 Hadoop 提供的大量插件快速开发程序。在 Hadoop 这样的基于集群的分布式并行系统中[29]，计算节点可以很方便地扩充，而因它所能够提供的计算能力近乎是无限的，但是由是数据需要在不同的计算机之间流动，故网络带宽变成了瓶颈，是非常宝贵的。数据存在哪一台计算机上，就由这台计算机进行这部分数据的计算，这样就可以减少数据在网络上的传输，降低对网络带宽的需求。

2.3 HDFS 分布式文档系统

HDFS 是根据 Google 的 GFS 文档系统开发的开放源代码分布式文档系统[30]，HDFS 的设计理念是在超大型的分布式存储环境里，提供单一的目录系统，它可以处理高达 1 万个节点、1 亿个文档与 10PBs 的数据量。HDFS 数据访问特性是采用

Write-once-read-many 访问模式，即一次写入，多次读出的访问模式。文档一旦创建或写入，就不允许更改，而是以附加的方式加在原文档的后面，这一点和 GFS 的 record append 记录追加操作模式类似。在 HDFS 文档系统中，每个文档被切割成许多区块，每个区块的大小通常 128MB，不同于 GFS 系统的 64MB，而且 HDFS 文档系统会将每个区块复制成许多副本并分散到不同的数据节点主机上，默认是每个文档存储 3 份副本。因为移动计算到数据端比移动数据到计算端的成本来的低，所以通过大量复制数据以避免某台节点主机因硬件故障而造成数据损坏或丢失。

HDFS 由一个名叫 Namenode 的主节点和多个叫 Datanode 的子节点组成[31]，Namenode 主节点存储着文件系统的元数据，这些元数据包括文件系统的名字空间等，向用户映射文件系统并负责管理文件的存储等服务，但实际的数据并不是存放在 Namenode 里。图 2-4 是 HDFS 文档系统的运作架构。

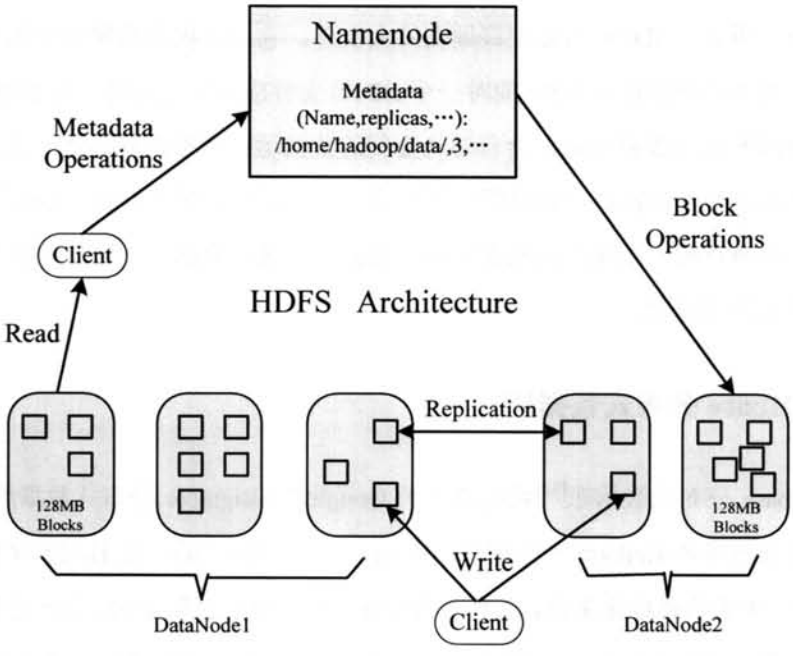

图 2-4 HDFS 文档系统运作架构

Fig.2-4 The Operational Architecture of HDFS File System

HDFS 在使用上同我们熟悉的单机上的文件系统非常类似，一样可以建目录，创建，复制，删除文件，查看文件内容等。在 HDFS 中，Datanode 用于实际对数据的存放，对 Datanode 上数据的访问并不通过 Namenode，而是与用户直接建立数据通信。HDFS 工作原理是这样的，用户向 Namenode 请求创建文件的指令，Namenode 在接到请求访问后，将存储数据的 Datanode 的信息返回给用户，并通知其他接收副本的 Datanode，由用户直

接与 Datanode 进行数据传送。

HDFS 的底层实现上是把文件切割成 Block[32]，然后这些 Block 分散地存储于不同的 Datanode 上，每个 Block 还可以复制数份存储于不同的 Datanode 上，达到容错容灾之目的。Namenode 则是整个 HDFS 的核心，它通过维护一些数据结构，记录了每一个文件被切割成了多少个 Block，这些 Block 可以从哪些 Datanode 中获得，每个 Datanode 的状态等重要信息。图 2-5 是存储于 Datanode 的 Block 块信息。

Block Replication
Namenode(Filename,numReplicas,block-ids,…) /users/sameerp/data/part-0,r:2,{1,3},… /users/sameerp/data/part-1,r:3,{2,4,5},…

图 2-5 Block 块信息

Fig.2-5 The Information of Block

运行在 HDFS 之上的程序一般都要处理海量数据。一般的 HDFS 文件都是 GB 到 TB 的大小，所以，HDFS 可以很好地支持大文件。它应该提供高聚合数据带宽并且可以在一个集群中支持数百个节点，同时一个 HDFS 文件应该可以支持一个集群中千万的文件。而当 HDFS 检测到错误时，会自动从复制的数据运行数据恢复动作。这是因为名字节点控制所有的块复制操作。它周期性地接受来自集群中数据节点的“心跳”回应和块报告。收到一个节点的“心跳”回应表示这个数据节点是正常的。一个块报告包括该数据节点上的所有的块列表。

2.4 Hbase 分布式数据库

Hbase 分布式数据库[33]系统类似于 Google 的 Bigtable 分布式数据库的功能，Hbase 分布式数据库和 Hadoop 一样都是用 Java 语言开发而成的，以 HDFS 文档系统为存储基础，将一个表格拆成很多份，由不同的服务器负责该部分的访问，借此达到高性能。Hbase 不是一般常见的 MySQL、SQLite、Oracle 等关联式数据库系统，其不提供 join 功能与 SQL 语法，但是 Hbase 拥有类似表格的数据结构，其表格只有一个主要索引即 Row key，Hbase 表格依 Row key 来自动排序，并提供 Java 函数库、REST 与 Thrift(Thrift 是一套由 Facebook 所开发的跨语言数据交换平台)等数据访问界面，可通过 getRow()或 Scan()等函数访问数据。

Hbase 是一个稀疏的，排序的，长期存储在硬盘上的，多维度的，映射表[34]。这张表的索引是行关键字，列关键字和时间戳。每个值是一个不解释的字符数组，数据都是

字符串。在创建 Hbase 表格时，其采用类似于 Bigtable 的形式，只要定义 Column Family，每个 Column Family 可拥有无限数量的 Columns，而且每个 Column 的值可有无限数量的时间戳。另外，Column 可以动态新增，每个 Row 可以有不同数量的 Columns。而 Hbase 数据库在写入数据时和 Bigtable 数据库一样，都是先写到节点主机的动态内存中，并且有 Write-Ahead Log 以防放生意外时可以进行恢复，每隔一段时间就会把数据写入 HDFS 文档系统中。在 Hbase 系统中，getRow()函数可以取得一笔 Row range 的数据，同时也可以通过时间戳指定不同时间存储的数据版本，而 Scan()函数可以取得整个表格的数据。

因此，Hbase 具备分布式、高可用性、高性能与容易伸缩容量等特性，更适用于在数据量庞大的服务器集群上运行，用来存储 Petabytes 等级的大量数据，而且 Hbase 同时搭配 Hadoop 的 MapReduce 功能，更有利于数据的快速计算。

2.5 Hbase 体系结构

Hbase 是 Hadoop Database 的简写，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统[35]，利用 Hbase 技术可在廉价 PC Server 上搭建起大规模结构化存储集群。Hbase 是 Google Bigtable 的开源实现，类似 Google Bigtable 利用 GFS 作为其文件存储系统，Hbase 利用 Hadoop HDFS 作为其文件存储系统；Google 运行 MapReduce 来处理 Bigtable 中的海量数据，Hbase 同样利用 Hadoop MapReduce 来处理 Hbase 中的海量数据；Google Bigtable 利用 Chubby 作为协同服务，Hbase 利用 Zookeeper 作为对应，Hbase 的体系结构如图 2-6 所示：

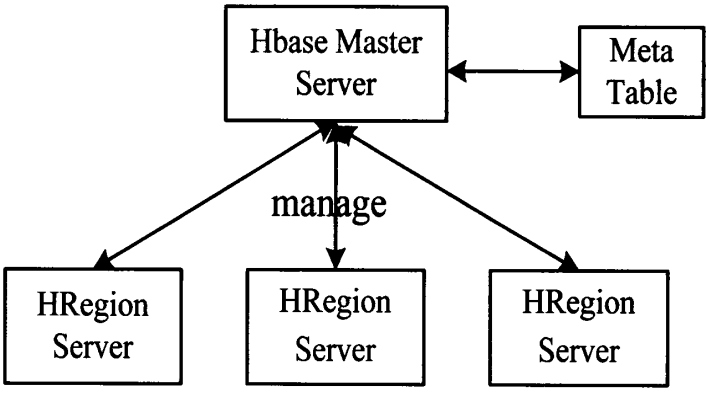

图 2-6 Hbase 体系结构

Figure.2-6 The System Structure of Hbase

如图 2-6 所示，Hbase 数据库有 HMaster Server 与 HRegion Server 组成，HMaster Server 上面还存放元数据信息。Hbase 依托于 Hadoop 的 HDFS 作为存储基础，因此结构也类似

与 Hadoop 的 Master-Slave 模式，由 Hbase Master 主服务器和 Hregion 服务器群构成[36]。

2.5.1 Hbase 访问接口

1、Native Java API，最常规和高效的访问方式，适合 Hadoop MapReduce Job 并行批处理 Hbase 表数据。

2、Hbase Shell，Hbase 的命令行工具，最简单的接口，适合 Hbase 管理使用。

3、Thrift Gateway，利用 Thrift 序列化技术，支持 C++，PHP，Python 等多种语言，适合其他异构系统在线访问 Hbase 表数据。

4、REST Gateway，支持 REST 风格的 Http API 访问 Hbase，解除了语言限制。

5、Pig，可以使用 Pig Latin 流式编程语言来操作 Hbase 中的数据，和 Hive 类似，本质最终也是编译成 MapReduce Job 来处理 Hbase 表数据，适合做数据统计。

6、Hive，当前 Hive 的 Release 版本尚没有加入对 Hbase 的支持，但在下一个版本 Hive 0.7.0 中将会支持 Hbase，可以使用类似 SQL 语言来访问 Hbase。

2.5.2 Hregion Server

Hregion Server 主要负责响应用户 I/O 请求，向 HDFS 文件系统中读写数据[37]，是 Hbase 中的最核心模块。Hregion Server 内部管理了一系列 Hregion 对象，每个 Hregion 对应了 Table 中的一个 region。Hregion 由多个 Hstore 组成，每个 Hstore 对应了 Table 中的一个 Column Family 存储，可以看出每个 Column Family 其实就是一个集中的存储单元，因此最好将具备共同 IO 特性的 column 放在一个 Column Family 中。Hregion 的结构组成如图 2-7 所示：

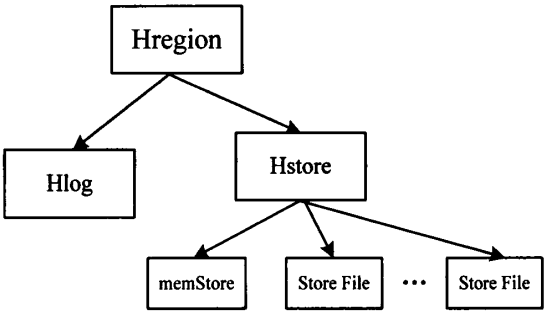

图 2-7 HRegion 的结构组成

Fig.2-7 The Structure of HRegion

如图 2-7 所示，Hregion 又由 Hlog，Hstore 组成。其中，Hstore 是 Hbase 存储的核心，而 Hstore 由两部分组成[38]，一部分是 MemStore(Sorted Memory Buffer)，一部分是

StoreFiles。用户写入的数据首先会放入 MemStore，当 MemStore 满了以后会 Flush 成一个 StoreFile(底层实现是 Hfile)，当 StoreFile 文件数量增长到一定阈值，会触发 Compact 合并操作，将多个 StoreFiles 合并成一个 StoreFile，合并过程中会进行版本合并和数据删除，但对 Hbase 来说其实只有增加数据，因为所有的更新和删除操作都在后续的 compact 过程中进行，这使得用户的写操作只要进入内存中就可以立即返回，保证了 Hbase I/O 的高效性。当 StoreFiles Compact 后，会逐步形成越来越大的 StoreFile，当单个 StoreFile 大小超过一定阈值后，会触发 Split 操作，同时把当前 Region Split 成 2 个 Region，父 Region 会下线，新 Split 出的 2 个孩子 Region 会被 Hmaster 分配到相应 HregionServer 上，使得原来 1 个 Region 的压力得以分流到 2 个 Region 上。Compact 和 Split 的过程[39]如图 2-8 所示：

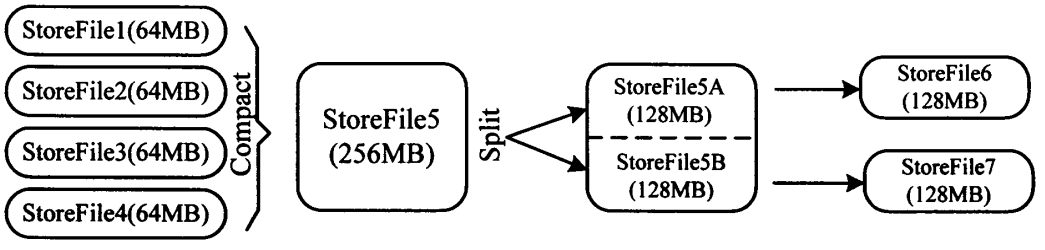

图 2-8 Compact 和 Split 过程

Fig.2-8 The Process of Compact and Split

上述描述式基于一个假设，那就是系统在正常工作。但是在分布式系统中，无法避免系统出错或宕机，因此一旦 Hregion Server 意外退出，MemStore 中的内存数据将会丢失，这就需要 Hlog 了。每个 Hregion Server 中都有一个 Hlog 对象，Hlog 是一个实现 Write Ahead Log 的类，在每次用户操作写入 MemStore 的同时，也会写一份数据到 Hlog 文件中，Hlog 文件会定期滚动出新的，并删除旧的文件(已持久化到 StoreFile 中的数据)。当 Hregion Server 意外终止后，Hmaster 会通过 Zookeeper 感知到，Hmaster 首先会处理遗留的 Hlog 文件，将其中不同 Region 的 Log 数据进行拆分，分别放到相应 region 的目录下，然后再将失效的 region 重新分配，领取到这些 region 的 Hregion Server 在 load Region 的过程中，会发现有历史 Hlog 需要处理，因此会 Replay Hlog 中的数据到 MemStore 中，然后 flush 到 StoreFiles，完成数据恢复。

2.5.3 Hmaster Server

Hmaster 没有单点问题[40]，Hbase 中可以启动多个 Hmaster，通过 Zookeeper 的 Master Election 机制保证总有一个 Master 运行，Hmaster 在功能上主要负责 Table 和 Region 的

管理工作。Hmaster 服务器并不存储数据，每个 Hregion 服务器都会和 Hmaster 服务器通讯，Hmaster 的主要任务就是要告诉每个 Hregion 服务器它要维护哪些 Hregion。Hmaster 服务器会和每个 Hregion 服务器保持一个长连接。如果这个连接超时或者断开，会导致：

1、Hregion 服务器自动重启

2、Hmaster 认为 Hregion 已经死机，同时把它负责的 Hregion 分配到其它 Hregion 服务器。

当一个新的 Hregion 服务器登陆到 Hmaster 服务器，Hmaster 会告诉它先等待分配数据。而当一个 Hregion 死机的时候，Hmaster 会把它负责的 Hregion 标记为未分配，然后把它们分配到其它 Hregion 服务器。

2.6 Hbase 存储格式

Hbase 中的所有数据文件都存储在 Hadoop HDFS 文件系统上，主要包括两种文件类型：

1、HFile，Hbase 中 key/value 数据的存储格式，HFile 是 Hadoop 中的二进制格式文件，实际上 StoreFile 就是对 HFile 做了轻量级包装，即 StoreFile 底层就是 HFile。

2、Hlog File，Hbase 中 WAL(Write Ahead Log)的存储格式，物理上是 Hadoop 的 Sequence File。

HFile 的存储格式[41]如图 2-9 所示：

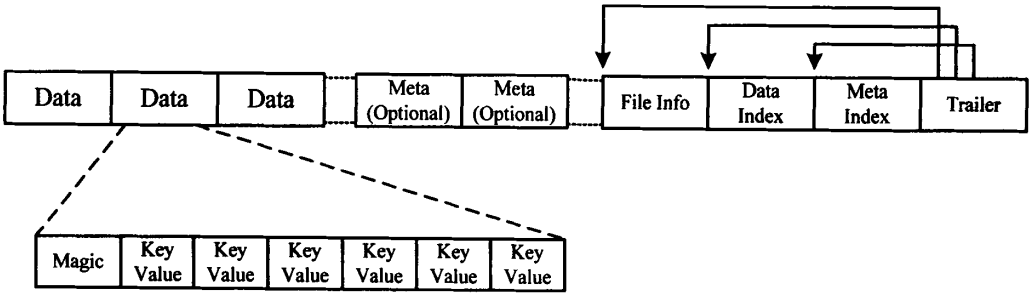

图 2-9 HFile 的存储格式

Fig.2-9 The Storage Pattern of HFile

如图 2-9 所示，HFile 是不定长的，长度固定的只有其中的两块，Trailer 和 FileInfo。Trailer 中有指针指向其他数据块的起始点，File Info 中记录了文件的一些 Meta 信息，例如：AVG_KEY_LEN, AVG_VALUE_LEN, LAST_KEY, COMPARATOR, MAX_SEQ_ID_KEYD 等。Data Index 和 Meta Index 块记录了每个 Data 块和 Meta 块的起始点。

Data Block 是 Hbase I/O 的基本单元，为了提高效率，HregionServer 中有基于 LRU 的 Block Cache 机制。每个 Data 块的大小可以在创建一个 Table 的时候通过参数指定，大号的 Block 有利于顺序的 Scan，小号 Block 利于随机查询。每个 Data 块除了开头的 Magic 以外就是一个个 Key/Value 对拼接而成，Magic 内容就是一些随机数字，目的是防止数据损坏。

HFile 里面的每个 Key/Value 对就是一个简单的 byte 数组[42]，但是这个 byte 数组里面包含了很多项，并且有固定的结构，Key/Value 对的具体结构如图 2-10 所示：

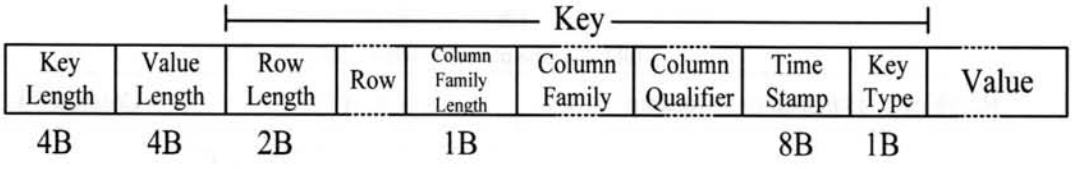

图 2-10 Key/Value 的数组结构

Fig.2-10 The Array Structure of Key/Value

如图 2-10 所示，结构开始是两个固定长度的数值，分别表示 Key 的长度和 Value 的长度。紧接着是 Key，开始是固定长度的数值，表示 RowKey 的长度，然后是 RowKey。接下来是固定长度的数值，表示 Family 的长度，然后是 Family，再然后是 Qualifier，在之后是两个固定长度的数值，表示 Time Stamp 和 Key Type(Put/Delete)。Value 部分没有非常复杂的结构，是纯粹的二进制数据。

HlogFile 负责备份数据，以防数据丢失，HlogFile 的结构[43]如图 2-11 所示：

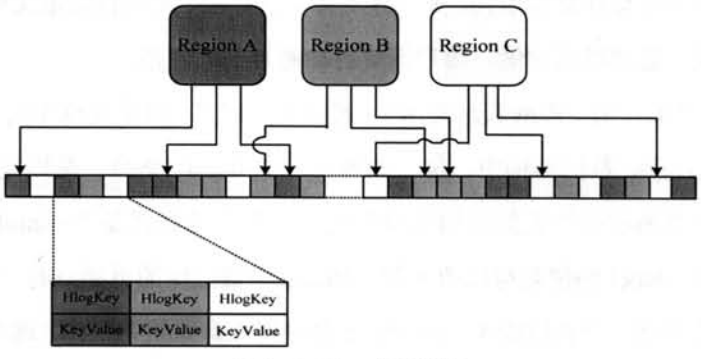

图 2-11 Hlog 文件结构

Fig.2-11 Hlog File System

图 2-11 给出了 Hlog 文件的结构，其实 Hlog 文件就是一个普通的 Hadoop Sequence File，Sequence File 的 Key 是 HlogKey 对象 HlogKey 中记录了写入数据的归属信息，除了 table 和 region 名字外，同时还包括了 sequence number 和 time stamp，timestamp 是“写入时间”，sequence number 的起始值为 0，或者是最近一次存入文件系统中 sequence

number。Hlog Sequence File 的 Value 是 Hbase 的 Key/Value 对象，即对应 HFile 中的 Key/Value。

2.7 Hbase 数据模型

在 Hbase 数据库中，数据都是存储在表格[44]中，表格中的每一行由一个可排序的主键和任意多的列组成。表格存储方式是稀疏存储，在同一张表里的每一行都可以有不同的列。Hbase 的逻辑数据模型如表 2-1 所示：

表 2-1 Hbase 数据模型

Table2-1 The model of Hbase data

RowKey	TimeStamp	Column “C1”	Column “C2”		Column “C3”
	T5		“C2:L2”	“L2-content”	
	T4		“C2:L1”	“L1-content”	
“rk1”	T3	“C1-content3”			“C3-content1”
	T2	“C1-content2”			
	T1	“C1-content1”			

如表 2-1 所示，表格有三部分组成，分别是：Row key 行键、TimeStamp 时间戳和 Column Family 列簇。Row key 是表的主键，表中的记录按照 Row Key 排序。TimeStamp 对应每次数据的操作，可以看做是数据的标记。表在水平方向上由一个或多个列簇组成，一个列簇中可以有任意多个列，及列簇支持动态扩展，无需预先定义列的数量以及类型，所有列均以二进制格式存储，用户需要自动进行类型转换。

表 2-1 共有一行，Row Key 是 rk1。有 4 个列定义，分别是<C1:>, <C2:L1>, <C2:L2>, <C3:>。在 Hbase 数据结构中，每一张表由一个 family 集合，是固定不变的，相当于表的结构，只能通过改变表的结构来改变它。列名字的格式是“<family>:<label>”，是由字符串组成，label 的值是可以改变的。Hbase 中数据库的更新只有一个时间戳标记，每次更新都会变成一个新的版本，而 Hbase 会保留一定数量的版本，这个值是可以设定的。客户端可以一次获取所有版本，也可以选择获取距离某个时间最近的版本。

Hbase 数据库中的表格虽然从概念视图来看每个表格是由很多行[45]组成，但是在物理存储上面，它是按照列来保存的，Hbase 物理存储数据模型如表 2-2、2-3、2-4 所示：

表 2-2 Hbase 物理模型 1

Table2-2 The Physical Model 1 of Hbase

RowKey	TimeStamp	Column “C1”
“rk1”	T3	“C1-content3”
	T2	“C1-content2”
	T1	“C1-content1”

表 2-3 Hbase 物理模型 2

Table2-3 The Physical Model 2 of Hbase

RowKey	TimeStamp	Column “C3”
“rk1”	T3	“C3-content1”

表 2-4 Hbase 物理模型 3

Table2-4 The Physical Model 3 of Hbase

RowKey	TimeStamp	Column “C2”	
“rk1”	T5	“C2:L2”	“L2-content”
	T4	“C2:L1”	“L1-content”

从表 2-2~2-4 可以看出，Hbase 物理数据模型实际上是将逻辑数据模型中的一行分成按列簇存储，把同列的数据存储在一个目录底下。Hbase 存储格式虽然是稀疏存储，但是在物理存储时，逻辑模型中 NULL 值是不存储的，并没有占据空间。

2.8 Hadoop 平台 MapReduce 模型

MapReduce 是云计算的关键技术，MapReduce 将要运行的大量数据处理任务拆分成 Map 和 Reduce 的方式来运行，以达到分散计算的效果。有了 Map 程序之后，将可以使用大量机器来运行 Map 程序，分别同步处理分析每一段数据，再将每个 Map 程序分析出来的结果，传送给 Reduce 程序合并，最后汇整出完整的结果。

Map 操作是并行的，在文件划分时并不考虑输入文件的内部逻辑结构，划分后的文件片段会对应的创建一个新的 Map 任务。当一个单独的 Map 任务开始时，对应的都会按照每个 Reduce 任务配置输入文件写操作者。然后 Map 任务会使用从指定的 InputFormat 类获得的 RecordReader 类来读取它的 FileSplit 属性。InputFormat 类负责解析输入和生成 key/value 对。InputFormat 也需要处理达到 FileSplit 边界值的记录。N 个输入文件会产生 M 个待运行的 Map 任务，每个 Map 任务都会产生由系统配置好的规约任务数量相同的

输出文件。每个输出文件对应一个规约任务，所有 Map 对象的输出键值对都会被路由以保证每一个给定的索引键的所有键值对会最终出现在指定的一个 Reduce 任务中。

当一个 Reduce 任务开始时，它的输入来源于分散在多个节点上的 Map 任务所产生的许多文件。如果 Reduce 过程是运行在分布式模式下的话，需要在拷贝阶段先将这些文件拷贝到 Reduce 任务所在节点的本地文件系统。

图 2-12 是 MapReduce 的执行流程[46]，具体从数据流的角度展示了 MapReduce 算法模型。

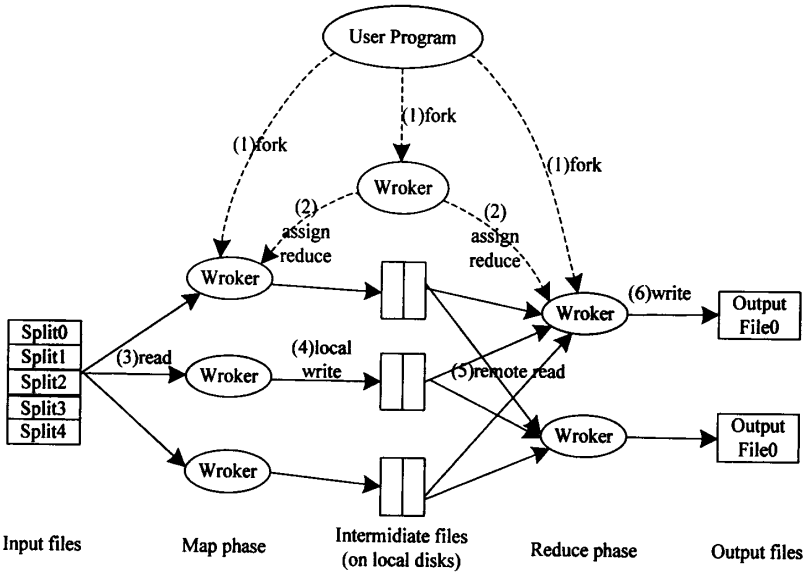

图 2-12 MapReduce 执行流程

Figure.2-12 Implimentation process of MapReduce

如图 2-12 所示，MapReduce 执行流程大体分为五部分，具体有六个步骤。五部分分别是输入文件、Map 环节、本地存储、Reduce 环节和输出文件，而六个步骤分别是：

1、fork 分割文件

在 MapReduce 执行流程之初，将输入文件分成大小不等的若干小数据块，数据块大小由用户给定参数控制，然后启动机器集群中的程序拷贝。

2、指派 MapReduce 任务

在 MapReduce 执行流程中，有一台专门负责管理的主服务器(master)，其他均为工作站(worker)，工作站程序由主服务器指派任务，主服务器指派空闲的工作站执行 Map 任务或是 Reduce 任务。

3、read 读取

被指派执行 Map 任务的工作站读取相关的数据块，从原始数据中解析出 Key/Value

对，经过 Map 函数处理，得到中间键值时，存入内存缓冲区。

4、本地存储

内存中的数据组被划分函数周期性的划分到 R 个区域写入本地磁盘。这些在本地磁盘的数据数列的存放位置信息被送回主服务器，主服务器器负责将这些位置信息传送到执行 Reduce 任务的工作站。

5、remote read 远程读取

当执行 Reduce 任务的工作站获悉存储具体数据的位置信息，通过远程方式读取执行 Map 任务的工作站中的本地缓冲数据。Reduce 工作站读取完所有中间的数据后，通过中间关键字对数据进行排列，把具有相同关键字的数据分为一类。排序操作是必须的，因为具有不同的关键字 Map 后会进行相同的 Reduce 操作。如果中间数据的数量太大不适合存入内存，就启用外部存储。

6、write 写到输出文件

Reduce 工作站对每一个由唯一的中间关键字对应的中间数据进行排列，并且发送关键字和相对应的中间值给用户的 Reduce 函数。Reduce 函数的输出结果将被添加到最后的输出文件中。

在所有的 Map 任务和 Reduce 任务完成了之后，主服务器 master 激活用户程序，MapReduce 返回用户程序的调用点，这就是一个完整的 MapReduce 执行流程[47]。

在 MapReduce 执行流程中，需要注意几个问题：

1、任务粒度

把原始大数据集切割成小数据集时，通常让小数据集小于或等于 HDFS 中一个 Block 的大小(缺省是 64M)，这样能够保证一个小数据集位于一台计算机上，便于本地计算。有 M 个小数据集待处理，就启动 M 个 Map 任务，注意这 M 个 Map 任务分布于 N 台计算机上并行运行，Reduce 任务的数量 R 则可由用户指定

2、Partition

把 Map 任务输出的中间结果按 key 的范围划分成 R 份(R 是预先定义的 Reduce 任务的个数)，划分时通常使用 hash 函数如: hash(key) mod R，这样可以保证某一段范围内的 key，一定是由一个 Reduce 任务来处理，可以简化 Reduce 的过程。

3、Combine

在 partition 之前，还可以对中间结果先做 combine，即将中间结果中有相同 key 的 <key, value>对合并成一对。combine 的过程与 Reduce 的过程类似，很多情况下就可以直

接使用 Reduce 函数，但 combine 是作为 Map 任务的一部分，在执行完 Map 函数后紧接着执行的。Combine 能够减少中间结果中<key, value>对的数目，从而减少网络流量。

4、中间结果

Map 任务的中间结果在做完 Combine 和 Partition 之后，以文件形式存于本地磁盘。中间结果文件的位置会通知 master，master 再通知 Reduce 任务到哪一个工作站上去取中间结果。注意所有的 Map 任务产生中间结果均按其 Key 用同一个 Hash 函数划分成了 R 份，R 个 Reduce 任务各自负责一段 Key 区间。每个 Reduce 需要向许多个 Map 任务结点取得落在其负责的 Key 区间内的中间结果，然后执行 Reduce 函数，形成一个最终的结果文件

5、任务管道

有 R 个 Reduce 任务，就会有 R 个最终结果，很多情况下这 R 个最终结果并不需要合并成一个最终结果。因为这 R 个最终结果又可以做为另一个计算任务的输入，开始另一个并行计算任务。

论文主要有两部分工作组成：一是关于海量数据的存储，二是关联规则的挖掘。如图 2-13 所示：

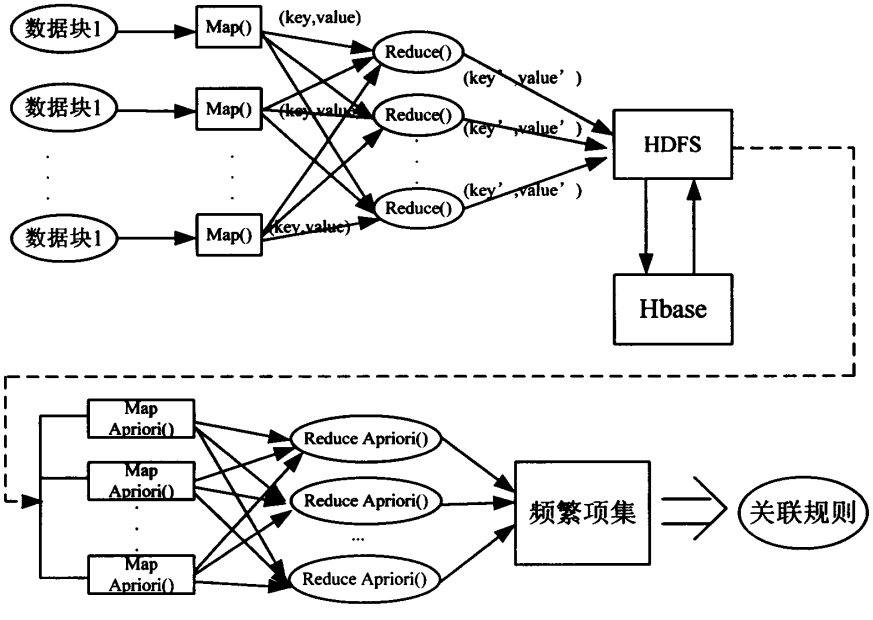

图 2-13 系统流程图

Fig.2-13 The Flow of System

如图 2-13 所示，在本文的第三章主要工作是基于 Hadoop 平台的电子健康档案的海量数据存储模型设计，第四章主要工作是电子健康档案的关联规则挖掘的相关实现，第五章主要工作是系统软硬件的具体实施和配置。

第 3 章　基于 Hadoop 的电子健康档案的数据存储模型设计

为了保证数据的高可用性和高可靠性，云计算的数据一般采用分布式的方式来存储和管理。类似于一般的数据存储安全保证办法，云计算也采用冗余存储的方式来保证存储数据的可靠性。由于云计算系统需要同时满足大量用户的需求，并行地为大量用户提供服务，因此云计算的数据存储技术必须具有高吞吐率，分布式存储正好满足了这一需求特点。

3.1　数据定义

医疗信息化数据定义规范包括数据元定义文件规范、数据集模板文件规范、数据集实例文件规范。三种规范的意义在于抽象数据集结构，数据元定义文件抽象每种数据类型，数据集模板文件定义各种数据集模板，数据集实例文件则定义具体数据集实例。

以上定义用 XML(Extensible Markup Language)即可扩展标记语言定义，它与 HTML 一样，都是 SGML(Standard Generalized Markup Language，标准通用标记语言)。XML 是 Internet 环境中跨平台，依赖于内容的技术，是当前处理结构化文档信息的有力工具。扩展标记语言 XML 是一种简单的数据存储语言，使用一系列简单的标记描述数据，而这些标记可以用方便的方式建立，在本文中，XML 语言被用来定义数据规范。

一个关于病人病历定义的实例如下如所示，在数据元定义文件中所有元数据定义在<fields>---</fields>标签内，每个字段的定义定义在<field>---</field>内，其 name 属性作为标识符被各种数据集模板文件引用，type 属性对应计算机开发语言内部数据结构类型，其他属性则可选，针对具体内部类型有关。此文件由系统管理员维护，在定义数据集模板文件时被载入并被解析，用于定义数据集模板文件。

数据集模板文件中所有数据集模板定义在<templates>---</templates>标签内，每个数据集模板定义在<template>---</template>内，其 name 属性作为标识符被各种数据集实例文件引用，数据集每列则定义在<field>---</field>内，其具体数据结构通过 type 属性引用之前定义的数据元定义文件中的数据元定义，而 column 属性则描述此列所在的物理位置，其中 ALL 预定义为表示全部，OTHER 预定义为其他未被定义的数据列。

数据集实例文件总所有数据集实例定义在<instances>---</instances>标签内，每个数据集实例定义在<instance>---</instance>内，其 name 属性作为标识符用于被相关用户所

查询，具体数据集实例所引用的模板通过 template 标签描述，通过其 name 属性指向数据集模板文件。而具体数据集类型与所在位置以及访问引擎等信息则由 datasource 标签定义，其 type 属性表示其数据访问引擎类型，location 属性则描述具体的数据集实例所在的物理位置，其他属性则与具体的数据集类型有关，例如 database 属性则是在数据集类型为 SQL Server 与 Oracle 等关系数据库时才有用的属性，用于指向具体的数据库。下面列出三个 XML 文件。

数据元定义文件：

```
<?xml version="1.0" encoding="UTF-8"?>
 <fields>
   <field name="int" type="INTEGAR" lower="-65535" upper="65535"/>
   <field name="float" type="FLOAT"/>
   <field name="string" type="STRING" maxlength="256"/>
   <field name="remark" type="REMARK"/>
 ...
 </fields>
```

数据集模板文件：

```
<?xml version="1.0" encoding="UTF-8"?>
 <templates>
  <template name="全浮点数">
   <field type="float" column="ALL"/>
  </template>
  <template name="病历模板 1">
   <field name="编号" type="int" column="1"/>
   <field name="名称" type="string" column="2"/>
   <field name="年龄" type="int" column="3"/>
   <field name="病种" type="int" column="4"/>
   <field name="性别" type="sex" column="5"/>
   <field type="remark" column="OTHER"/>
  </template>
  ...
```

```
</template>
```

数据集实例文件

```
<?xml version="1.0" encoding="UTF-8"?>
<instances>
  <instance name="某图像 PACS 分类用特征库">
    <template name="全浮点数"/>
    <datasource type="TEXT" location="../dataset" database="***.dat" seperator=" ">
  </instance>
  <instance name="某病历数据库">
    <template name="病历模板 1"/>
    <datasource type="MSSQL" location="http://mssql***" database="db***"
username="***" passwork="***"/>
  </instance>
  ...
</instances>
```

3.2 文件分块策略分析

Hadoop 的 HDFS 文件系统在提供高效的海量数据存储上具有相当的优势，然而 Hadoop 以 64MB 为单位对文件进行分块[48]，从块的大小来看 Hadoop 确实是面向海量数据的处理和存储，但对于小文件以及频繁存取的数据 Hadoop 并没有太大的优势，这大大限制 HDFS 的应用领域及应用的灵活性。本文提出的文件分块策略既能满足海量文件的高效存储和处理，又能满足小文件的频繁访问及处理。

我们采用一个简单的模型进行分析：

1、对文件分块作以下的假设：

(1) 只有一个机群系统。

(2) 公共网络的数据传输速度慢于机群内部间通信的速度。

(3) 各节点和主服务器的网速相同。

(4) 下载到客户端后文件整合方式相同。

2、相关符号符号约定如下

rs：客户端发送请求的时间。

cont：客户端与字节点连接一次的时间。

n：文件分块的块数。

fs：文件的空间。

cs：客户端的带宽。

ns：节点服务器的带宽。

a：计算机性能的系数。

cn=1：客户端 CPU 的数量。

3、文件分块模型的建立与求解

文件分块存储的原因是利用各硬盘的读取速度，总体提高读取数据的速度，减少读取数据的时间。文件分块运算的地点是集群主服务器，由主服务器来维护文件的分块策略，在 Hadoop 等并行系统上文件分块存储是实现计算向存储迁移的重要一步。

文件下载的流程如图 3-1 所示：

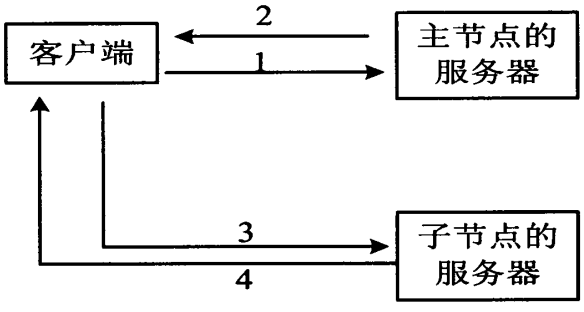

图 3-1 文件下载流程

Fig.3-1 The Download Flow of Files

如图 3-1 所示，文件下载流程总体分为四个步骤：

(1) 客户端发送请求下载的信息。

(2) 服务端收到信息，返回下载位置的信息。

(3) 客户端通过下载位置的信息连接到子节点的服务器。

(4) 客户端与子节点的服务器端互联，传输数据。

(5) 客户端下载完成所有文件块后，将其整合为一个完整的文件，并删除文件块。

文件上传的流程如图 3-2 所示：

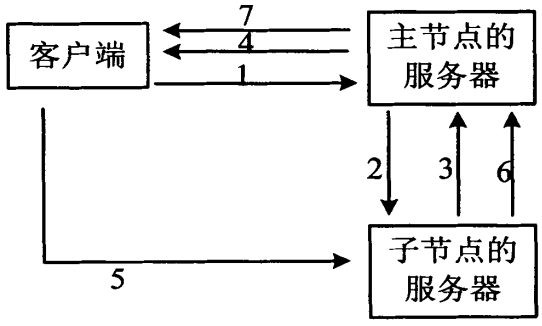

图 3-2 文件上传流程

Fig.3-2 The Upload Flow of Files

如图 3-2 所示，文件上传流程总体分为 7 个步骤：

(1) 客户端发送请求上传的信息。

(2) 服务器端收到信息，连接子节点的服务器，发送建立存储的空间。

(3) 子节点建立空间，返回建立的空间信息给主节点服务器。

(4) 主服务器保存空间信息，发送空间信息给客户端。

(5) 客户端通过空间信息连接子节点服务器，上传数据。

(6) 客户端上传完毕，子节点的服务器通知主节点的服务器。

(7) 服务器确定上传完毕，保存空间信息，发送到客户端。

分块的过程在上传的流程的第 5 个步骤。此过程的时间是连接的时间加上上传的时间。

数据传输的流程：内存从外存读入数据，从内存传送数据给目的计算机。内存的速度大于外存的速度，外存的速度大于网络的速度。

由图 3-1 和 3-2 可知；

当 cs≤ns 时，即客户端的下载速度小于节点服务器的上传速度时，客户端上传数据时，服务器建立文件的分块信息，从第一个节点依次上传数据。

分块后下载所需的时间：$ct = rt + \left(cont \times \frac{n}{cn} + \frac{fs}{cs} \right) + \frac{fs}{n} \times a \times (n-1)$。

未分块下载所需的时间：$uct = rt + cont + \frac{fs}{cs}$。

而：$ct - uct = cont \times (n-1) + \frac{fs}{n} \times a \times (n-1) > 0$。

所以此时不用分块。

当 cs>ns 时，即客户端的下载速度大于节点服务器的上传速度时，客户端的带宽无法充分利用。解决的方法是不够再添。不够再添的方法是再次使用负载均衡的方法，直到满足客户端的最大带宽。

分块后下载的时间：$ct = rt + \left(cont \times \frac{n}{cn} + \frac{fs}{cs} \right) + \frac{fs}{n} \times a \times (n-1)$。

未分块下载的时间：$uct = rt + cont + \frac{fs}{cs}$。

而：$ct - uct = cont \times (n-1) + fs \times \left(\frac{1}{cs} - \frac{1}{ns} \right) + \frac{fs}{n} \times a \times (n-1)$

此时可以采用分块并行上传。

上述文件分块策略只是一个定性判断，具体的决定是否分块还需要与实测数据相比对。一般来说为了提高单个文件的传输速度需要文件分块，也可以对文件直接进行多副本存储，用于向不同的用户提供不同的服务器连接。

3.3 电子健康档案数据模型

电子健康档案是医院信息系统(HIS，Hospital Information System)的重要组成部分，它客观完整地记录了病人信息。Hbase 数据存储具有一定的哈希性质，非常适合处理 Key/Value 类型的数据以及结构稀疏的大规模数据，因为 Hbase 并不是完全的关系型数据库，所以不适处理关系型很强的结构化数据。Hbase 在大规模数据的存储和处理上有独特的优势，其最大的特色是数据的一次写入，多次读取，所以 Hbase 数据修改频率不高。

医疗信息化产生的数据量是超大规模的，海量的，依据 Hbase 表结构设计电子健康档案的存储模式，用病人编号作为表的 RowKey，用病人的基本信息名字、年龄、性别、病的种类来作为 Column Family,虽然可以通过关系型数据库来设计关系表实现节点之间的关联，但是对于医疗信息化产生的海量数据来说，这样的效率并不理想。表 3.5 是在搭建好的 Hadoop 平台上，通过 Hbase 分布式数据库来设计电子健康档案的表格式。基于 Hbase 的电子健康档案数据模型如表 3-1 所示；

表 3-1 电子健康档案数据存储表

Table3-1 The Storage Table of Electronic Health Record Data

RowKey	Time Stamp	Column Info	Column Name	Column Age	Column Sex	Column illType
"0001"	T4	Info: Name	Name：某某			
	T3	Info: Age		Age：40		
	T2	Info: Sex			Sex：Male	
	T1	Info: illType				illType：Cardiopathy

电子健康档案逻辑表如 3-1 所示，病人编号作为表的 RowKey，表一共有 4 个列族，Info 列存储 Key 的基本信息，包括病人姓名、年龄、性别和病种。

表 3-1 在逻辑结构上是一张稀疏的半结构化的表，描述一个病人的基本的信息，与 Google 的 BigTable 表结构相对应，对于一个 Rowkey 来说，你只需要指定相关的列族名就可以获取相关病人的全部记录。表 3-2~3-6 是其物理存储结构形式。

表 3-2 病人信息的物理存储片段

Table3-2 The physical Storage Segment of The Patient's Information

RowKey	TimeStamp	Column Info
"0001"	T4	"Info:Name"
	T3	"Info:Age"
	T2	"Info:Sex"
	T1	"Info:illType"

表 3-3 病人名字的物理存储片段

Table3-3 The physical Storage Segment of The Patient's Name

RowKey	TimeStamp	Column Name
"0001"	T4	"Name:某某"

表 3-4 病人年龄的物理存储片段

Table3-4 The physical Storage Segment of The Patient's Age

RowKey	TimeStamp	Column Age
"0001"	T3	"Age:40"

表 3-5 病人性别的物理存储片段

Table3-5 The physical Storage Segment of The Patient's Sex

RowKey	TimeStamp	Column Sex
"0001"	T2	"Sex:Male"

表 3-6 病人病种的物理存储片段

Table3-6 The physical Storage Segment of The Patient's Symptom

RowKey	TimeStamp	Column illType
"0001"	T1	"illType: Cardiopathy"

从上述物理表结构我们可以看出，在 Hbase 数据库里面存储的相关的数据是连续存储的，从而可以大大节约查询的时间。

Hbase 数据库是一个基于列模式的映射数据库，只能表示简单的 key/value 的映射关系，这样设计的可视化系统模型在存储模式、数据维护和可伸缩性等方面相对于传统的 Oracle、SQL Server 等关系型数据模型有很大的优势。

3.4 元数据表模型

在 Hbase 数据库中，Hregion 是按照表名和主键范围区分的[50]，因为主键范围是连续的，比如说电子健康档案数据表的主键就是连续存储的病人编号，所以一般用开始主键就可以表达出来需要查找的结果集所在的 Hregion 块。但是有时候只需要主键还是不够的，因为当执行 Map 和 Reduce 操作的节点恰好出错或者死机，这种情况下可能会出现有多份表明和开始主键一样的数据，这就需要通过 Hbase 的 Metadata 来区分哪一份才是我们真正需要的数据文件，这时每个 Hregion 都有一个"regionID"来标识其唯一性。

这样就可以通过表名、主键和 regionID 来找到我们需要的那个表，而这些信息都是作为元数据被保存在 Hregion 里的，也是通过表结构存储，称为元数据表。里面保存的就是 Hregion 标识符和实际 Hregion 服务器的映射关系。通过元数据表，可以查询和控制各个 Hregion 的数据信息以及状态。元数据表是存放在 Master 服务器上集中管理的，

这样就非常有利于数据的维护。但是需要注意的是由于 Master 服务器通常只有一个，这就要在设计系统的时候做好元数据表的备份。元数据的备份和容错与 Hadoop 平台的容错机制是一致的。元数据表结构如表 3.7 所示：

元数据表的 Info 列簇存储预案数据的基本信息，分别是：

1）info:regioninfo 包含了一个串行化的 HregionInfo 对象。

2）info:server 保存了一个字符串，是服务器地址 HserverAddress.toString()。

3）info:startcode 一个长整形的数字的字符串，是 Hregion 服务器启动的时候传给主服务器的，让主服务器决定这个 Hregion 服务器的信息有没有更改。

表 3-7 元数据表

Table3-7 The Table of Metadata

RowKey	TimeStamp	Column family:Info	Column family:Hregion
“0001”	T4	Info: regioninfo	
	T3	Info:server	
	T2	Info:startcode	
	T1		Hregion:regionid
			.....

Hregion 列簇则是用来存储 Hregion 的信息，列标签就是 Hregion 的标识符。当元数据表随着记录数不断增加而变大后，会逐渐分裂成多份 splits，成为 regions，一个 region 由[startkey,endkey]表示，不同的 region 会被 Master 分配给相应的 RegionServer 进行管理。为了定位这些，设计一个根数据表 ROOT_TABLE，Zookeeper 记录了 ROOT_TABLE 的具体位置，而根数据表是不能被分割的，永远只存在一个 Hregion。由于这个表只存元数据表所在 Hregion 的信息，所以空间是足够的。ROOT_TABLE 也包含一个 Info 列簇，它与 META_TABLE 的结构和含义是一样的。用户在访问之前首先访问 zookeeper，找到 ROOT_TABLE，接着在访问元数据表，最后才能找到用户数据去访问，中间需要多次网络操作。

3.5 海量数据存储模型设计

我们使用 MapReduce 模型在 Hbase 系统上运行批处理运算，模型如图 3-3 所示：

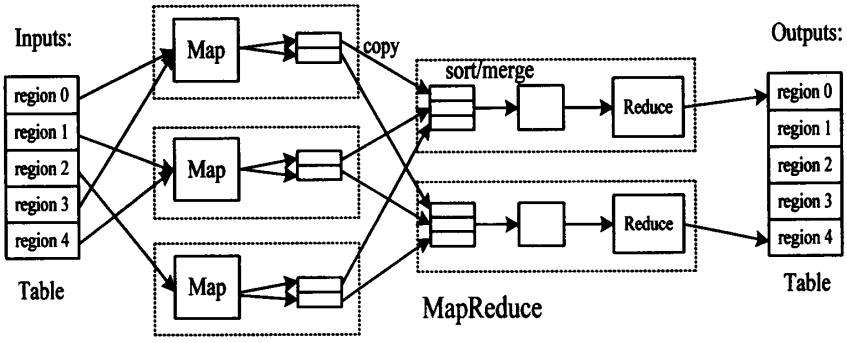

图 3-3 Hbase 的 MapReduce 模型

Fig.3-3 The MapReduce Model on Habse

在 Hbase 启动的时候，我们设定主服务器首先扫描根数据表，在这里根数据表只有一个 Hregion，Hregion 是被写死的。把根数据表分配到一个 Hregion 服务器需要一定的时间，当根数据表被分配好之后，主服务器就会去扫描根数据表，获取元数据表的名字和位置，然后把元数据表分配到不同的 Hregion 服务器。最后就是扫描元数据表，找到所有 Hregion 区域的信息，然后把它们分配给不同的 Hregion 服务器。

根据数据的海量特性，结合云计算技术，特提出基于云计算的海量数据存储模型，如图 3-4 所示：

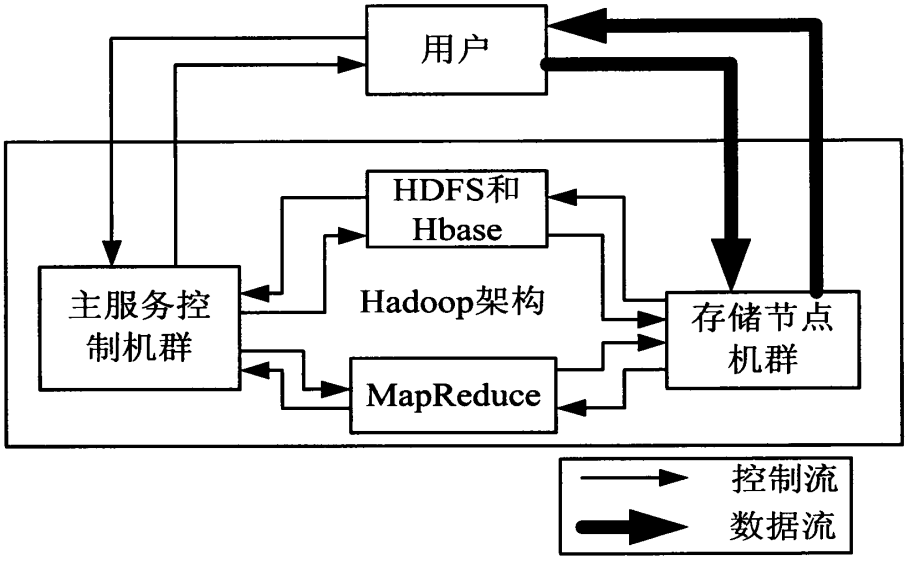

图 3-4 基于 Hadoop 的海量数据存储模型

Fig.3-4 The Storage Model of Massive Data Based On Hadoop

如图 3-4 所示，主服务控制机群相当于控制器部分，主要负责接收应用请求并且根据请求类型进行应答。存储节点机群相当于存储器部分，是由庞大的磁盘阵列系统或是具有海量数据存储能力的机群系统，主要功能是处理数据资源的存取。

Hadoop 有许多元素构成，最底部是 Hadoop Distributed File System(HDFS)，它存储 Hadoop 集群中所有存储节点上的文件。HDFS 的上一层是 MapReduce 引擎，该引擎由 JobTracker 和 TaskTracker 组成。MapReduce 算法模型、HDFS 和 Hbase 数据库是 Hadoop 的三大核心。HDFS 和 Hbase 用来将数据存储或部署到各个计算节点上。Hadoop 中有一个作为主控的 JobTracker，用于调度和管理其它的计算机(将其称之为 TaskTracker)，JobTracker 可以运行于机群中任一台计算机上。TaskTracker 负责执行任务，必须运行于 DataNode 上，DataNode 既是数据存储节点，也是计算节点。JobTracker 将 Map 任务和 Reduce 任务分发给空闲的 TaskTracker，让这些任务并行运行，并负责监控任务的运行情况。如果其中任意一个 TaskTracker 出故障了，JobTracker 会将其负责的任务转交给另一个空闲的 TaskTracker 重新运行。用户不直接通过 Hadoop 架构读取及 HDFS 和 Hbase 存取数据，从而避免了大量读取操作可能造成的系统拥塞。用户从 Hadoop 架构传给主服务控制机群的信息后，直接和存储节点进行交互进行读取操作。

3.6 存储模型的实现

本文设计了一个基于 MapReduce 的海量数据存储模型，模型的实现方法为：

存储数据基本方法为：

1 存储数据时，将存储数据的信息及其附加信息(如用户 ID)发送给主服务控制机群。

2 主服务控制机群接收到数据的信息。

3 将接收到的数据信息传送给 Hadoop 架构。

4 MapReduce 利用其 Map 函数对数据进行切块计算。

5 HDFS 和 Hbase 根据节点状态将数据均衡分配到各存储节点。

6 将数据块信息及存储节点地址返回主服务控制机群，并由主服务控制机群反馈给用户。

7 用户为每个存储节点建立一个数据块队列，将数据块并行上传到对应的存储节点。

下载数据基本方法为：

1 下载文件时，将要下载的文件信息传送给主服务控制机群。

2 主服务控制机群接收到要下载的文件信息。

3 HDFS 和 Hbase 查找该文件的块信息，并且将查找到的信息反馈给主服务控制机群。

4 主服务控制机群然后把信息传回给用户。

5 用户根据接收到的主服务控制机群传回的信息，为每个存储节点创建一个下载线程，将文件块并行下载到本地计算机临时文件夹中。

6 用户在下载完所有文件块以后，根据 MapReduce 的 Reduce 函数整合成一个完整的文件，并删除文件块。

此外，Hadoop 具有高容错性，能自动处理失效节点是通过 MapReduce 来实现的。MapReduce 通过把对数据集的大规模操作分发给网络上的每个节点实现可靠性，每个节点会周期性的把完成的工作和状态的更新报告回来。如果一个节点保持沉默超过一个预设的时间间隔，主节点记录下这个节点状态为死亡，并把分配给这个节点的数据发到别的节点。此外每个操作要保证不会发生并行线程间的冲突。

1、实现接口：

Hadoop 文件系统是 HDFS，由一个 namenode 节点和多个 datanode 节点组成，每个结点均是一台普通的计算机。

要实现对 Hadoop 文件系统的读写，必须要配置目标文件的绝对路径，来连接 Hadoop 文件系统，在第五章会介绍本项目关于 hdfs 的配置，然后通过 FileSystem 的 get 方法获得对目标文件的抽象引用。

对于 Hadoop 文件系统中的文件的访问是基于 InputStream 和 OutputStream 的流式访问，其访问方法如下所示。

```
String hdfsFileName = _hdfsFileName;
String hdfsFullPathFile = hdfsSrvAddr + hdfsDownloadPath + hdfsFileName;
InputStream hdfsInStream = null;
FileSystem fs = FileSystem.get(URI.create(hdfsFullPathFile), conf);
hdfsInStream = fs.open(new Path(hdfsFullPathFile));
```

2、存储数据的部分代码：

打开本地上传文件的输入流，同时以创建方式打开 Hadoop 文件的输出流，将输入流按字节读取出来，写入输入流即可。

代码如下：

```
InputStream in = new BufferedInputStream(new FileInputStream(localSrcFile));
FileSystem fs = FileSystem.get(URI.create(hdfsDstFile), conf);
OutputStream out = fs.create(new Path(hdfsDstFile));
int readLen = in.read(ioBuffer);
```

```
while(-1 != readLen){
out.write(ioBuffer, 0, readLen);
uploadBytes += readLen;
readLen = in.read(ioBuffer);
}
```

3、下载数据的部分代码：

打开 Hadoop 文件的输入流，同时以创建方式打开本地下载文件的输出流，将输入流按字节读取出来，写入输入流即可。

代码如下：

```
FileSystem fs = FileSystem.get(URI.create(hdfsFullPathFile), conf);
hdfsInStream = fs.open(new Path(hdfsFullPathFile));
OutputStream out = new FileOutputStream(localDstFile);
int readLen = hdfsInStream.read(ioBuffer);
while(-1 != readLen){
out.write(ioBuffer, 0, readLen);
downloadBytes += readLen;
readLen = hdfsInStream.read(ioBuffer);
}
```

第 4 章 基于 MapReduce 的电子健康档案关联规则挖掘

医疗信息化是贯穿医疗改革全过程的重要内容，通过对医院病人档案管理的精确评价，利用先进的评价分析方法从不同的角度分析和反馈医疗档案情况，是引导医生以及医院工作人员改进医疗方法、提高行医治病的本领，也是卫生行政部门、医院着力研究的重要课题。

目前许多医疗档案管理系统只是实现数据的输入、查询、统计等功能，难以发现数据中存在的关联、关系和规则，无法从大量数据中提取有用信息以预测病人全面健康发展趋势。如果能将这些隐含的规则和知识从海量数据中提取出来，将有望为医疗档案管理系统提供支持。因此，如何高效、准确地提取数据中隐含的规则和知识，提高决策的科学性和规范性，需要科学的工具和方法。数据挖掘技术正是满足这种需要而产生的一种综合技术，包含了统计学、机器学习、人工智能、数据库、知识获取、模式识别、分布式多媒体环境的只能代理等多方面的技术。

4.1 医疗信息化数据特点

医疗信息化数据具有以下特点：

1、数据的多样性和复杂性。由于医疗档案管理系统是一项复杂的系统工程，涉及到医疗行业整体和局部的各个方面，因此涵盖的数据量是极大的，其中既包括表示地理位置的空间数据，还用一些由文本、图像、音频等组成的非空间数据。这些数据的来源非常广泛，数据类型多样化。从系统内部的数据源角度看，面向医疗档案管理系统的数据挖掘系统要处理的数据类型更多，数据模型更复杂。

2、数据库的分布性和异构性。医疗档案管理系统是基于计算机和网络技术发展的，医院内各级单位用户的数据资源不是存在于一个数据库中，而是存在于地理分布的各个数据库中，其中的数据有可能是结构化的，也有可能是非结构化的。所以面向医疗档案管理系统的数据挖掘系统所处的计算机环境是分布式的，所处理的数据是异构的。

3、服务对象的多样性和层次性。所设计的数据挖掘系统并不仅仅只是为某一个层次服务的。它所服务的群体可能是面向普通大众。因此，用户类型的多样性也成为面向医疗档案管理系统数据挖掘系统的一个特点。

4.2 关联规则挖掘的定义

关联规则挖掘[51]发现大量数据项中项集之间有趣的关联或相关联系，从大量的数据记录中发现有趣的关联关系，可以更方便的管理数据，制定决策。本文中的关联规则挖掘通过分析病人的电子病历和电子健康档案，发现不同年龄的病人的常见病症，相同病症之间的不同人群，遗传病症等的常见发病趋势。通过分析可以找出相关症状的发病人群，一个人从出生到死亡的不同阶段的常发病症，可以更好的预防病症的发生，医院甚至通过这些分析提醒人们生活中可以注意的细节，医生可以提醒病人可能发生其身上的病种，从而做到相应的预防和应对策略，从而做到真正惠民，以最小的代价达到最好的治疗效果等。下面介绍如何从大量数据中找出关联规则的方法。

设 I={i1,i2,...,im}是项的集合。设任务相关的数据 D 是数据库事务的集合，其中每个事务 T 是项的集合，使得 $T \subseteq I$。每一个事务有一个标识符，称作 TID。设 A 是一个项集，事务 T 包含 A 当且仅当 $A \subseteq T$。关联规则是形如 $A \Rightarrow B$ 的蕴涵式，其中 $A \subset I$，$B \subset I$，并且 $A \cap B = \Phi$。规则 $A \Rightarrow B$ 在事务集 D 中成立，具有支持度 s，其中 s 是 D 中事务包含 $A \cup B$(即 A 和 B 二者)的百分比。它是概率 $P(A \cup B)$。规则 $A \Rightarrow B$ 在事务集 D 中具有置信度 c，如果 D 中包含 A 的事务同时也包含 B 的百分比 c。这是条件概率 P(B|A)。即：

$$support(A \Rightarrow B) = P(A \cup B) \tag{4.1}$$

$$confidence(A \Rightarrow B) = P(B \mid A) \tag{4.2}$$

同时满足最小支持度阈值(min_sup)和最小置信度阈值(min_conf)的规则称作强规则，用 0%和 100%之间的值表示支持度和置信度。

项的集合称为项集，包含 k 个项的集合成为 k-项集。例如集合{age，illtype}是一个 2-项集。项集的出现频率是包含项集的事务数，简称为项集的频率、支持计数或计数。项集满足最小支持度 min_sup，如果项集的出现频率大于或等于 min_sup 与 D 中事务总数的乘积。如果项集满足最小支持度，则称它为频繁项集，频繁 k-项集的集合记作 Lk。

关联规则的挖掘一般分为两个步骤：

1、找出所有频繁项集：根据定义，这些项集出现的频率性至少和预定义的最小支持计数一样。

2、由频繁项集产生强关联规则：根据定义，这些规则必须满足最小支持度和最小置信度。

在这两步中，第二步最容易，挖掘关联规则的总体性能由第一步决定。

设医疗信息活动中产生一个事务数据库(Transaction Database)，记为 D， 记 D 中的事务数为|D|；设 D 包含 n 个不同的项，记为 I={i1,i2,...,in}。

定义 1 关联规则：关联规则是形如 $A \Rightarrow B$ 的蕴涵式，其中 $A \subset I$，$B \subset I$，并且 $A \cap B = \Phi$。

定义 2 支持度：支持度用来描述项集在数据库里出现的频率，项集 X 的支持度是包含 X 的事务在数据库 D 中所占的百分比，可用公式(4.3)计算：

$$support(X) = \frac{D(X)}{|D|} \tag{4.3}$$

其中， D(X)是数据库 D 包含 X 的事务数。

定义 3 置信度：置信度描述了关联规则中条件和结果的确定程度，关联规则 $A \Rightarrow B$ 的置信度可用公式(4.4)计算：

$$confidence(A \Rightarrow B) = \frac{support(A \cup B)}{support(A)} \tag{4.4}$$

关联规则挖掘首先要找出所有满足支持度的项集，从而得到置信度大于阈值的所有关联规则。

4.3 Apriori 算法

Apriori 算法是一种找频繁项集的基本算法[52]，通过产生的频繁项集查找所需的关联规则。Apriori 算法是一种最有影响的挖掘布尔关联规则频繁项集的算法，Apriori 算法的基本算法是使用候选项集找频繁项集，算法的名字基于这样的事实：算法使用频繁项集性质的先验知识。Apriori 使用一种称作逐层搜索的迭代方法，k-项集用于搜索(k+1)-项集。首先，找出频繁 1-项集的集合，该集合记作 L1。L1 用于找频繁 2-项集的集合 L2，而 L2 用于找 L3，如此下去，知道不能找到频繁 k-项集。找每个 Lk 需要一次数据库扫描。为提高频繁项集逐层产生的效率，一种称作 Apriori 性质的重要性质用于压缩搜索空间，所谓的 Apriori 性质是频繁项集的所有非空子集都必须也是频繁的。Apriori 性质基于如下考察：根据定义，如果项集 I 不满足最小支持度 s，则 I 不是频繁的，即 P(I)<s。如果项 A 添加到 I，则结果项集(即，$I \cup A$)不可能比 I 更频繁出现。因此 $I \cup A$ 也不是频繁的，及 $P(I \cup A)$<s。

该性质属于一种特殊的分类，称作反单调，意指如果一个集合不能通过测试，则它的所有超集也都不能通过测试。所谓的反单调，是因为在通不过测试的意义下，该性质

是单调的。

设有某事务数据库 D，数据库中有 9 个事务，即|D|=9，Apriori 假定事务中的项按字典次序排放，事务数据库如表 4-1 所示：

表 4-1 事务数据库

Table4-1 Transactions Database

TID	List of item_ID's
T100	I1,I2,I5
T200	I2,I4
T300	I2,I3
T400	I1,I2,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I1,I2,I3,I5
T900	I1,I2,I3

对表 4.1 所示的事务数据库 D 进行 Apriori 分析：

1、在算法的第一次迭代，每个项都是候选 1-项集的集合 C1 的成员。算法简单地扫描所有的事务，对每个项的出现次数技术。

2、假定最小事务支持技术为 2(即，*min_sup*=2/9=22%)，可以确定频繁 1-项集的集合 L1，它由最小支持度的候选 1-项集组成。

3、为发现频繁 2-项集的集合 L2，算法使用 L1，L1 产生候选 2-项集的集合 C2。C2 由$\binom{|L1|}{2}$个 2-项集组成。

4、下一步，扫描 D 中事务，计算 C2 中每个候选项集的支持计数。

5、确定频繁 2-项集的集合 L2，它由最小支持度的 C2 中的候选 2-项集组成。

6、候选 3-项集的的集合 C3 产生的过程：首先令 C3=L2，L2={{I1，I2,I3}，{I1,I2,I5}，{I1,I3,I5}，{I2,I3,I4}，{I2,I3,I5}，{I2,I4,I5}}。根据 Apriori 性质，频繁项集的所有子集必须是频繁的，我们可以确定后 4 个候选不可能是频繁的。因此，我们把它们有 C3 删除，这样，在此后扫描 D 确定 L3 时就不必再求它们的计数值。Apriori 算法使用逐层搜索技术，给定 k-项集，只需检查它们的(k-1)-子集是否频繁。

7、扫描 D 中事务，以确定 L3，它由具有最小支持度的 C3 中的候选 3-项集组成。

8、算法使用 L3，L3 产生候选 4-项集的集合 C4。尽管连接产生结果{{I1,I2,I3,I5}}，这个项集被剪枝去，因为它的子集{I1,I3,I5}不是频繁的。这样，C4=∅，因此算法终止，找出了所有的频繁项集。图 4-1 描述候选项集的产生，假定最小支持技术为 2。

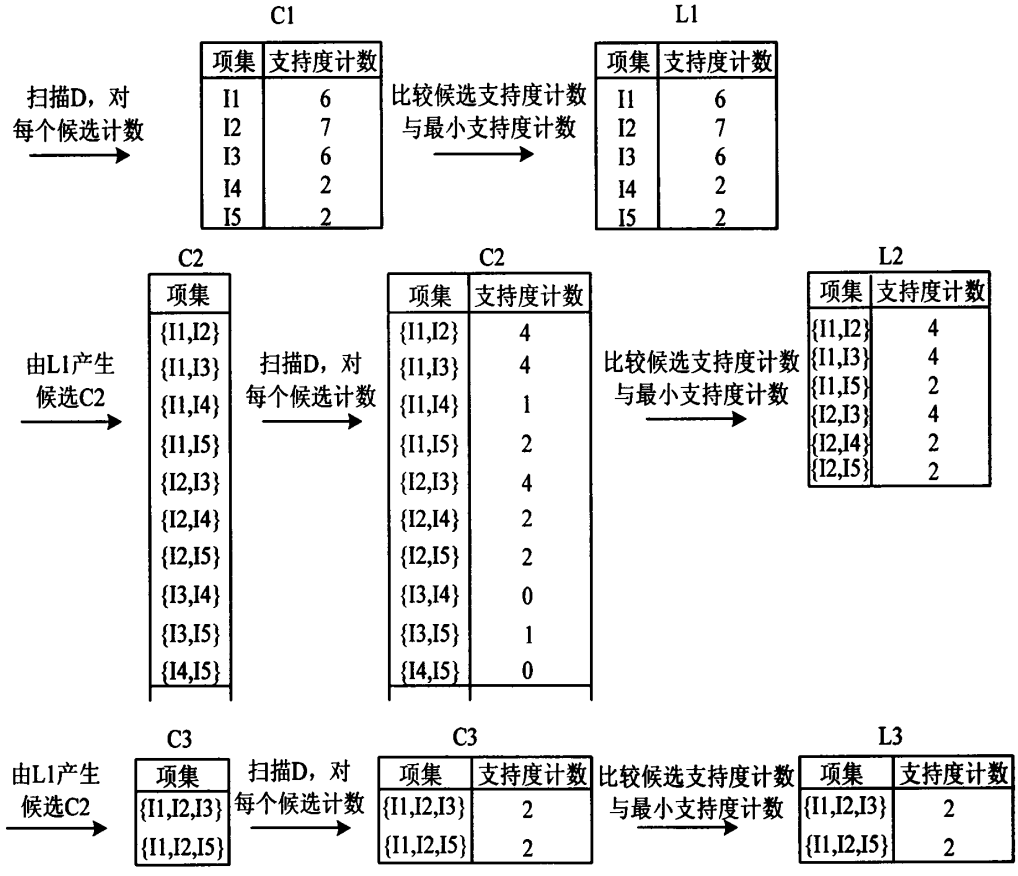

图 4-1 挖掘频繁项集流程图

Fig.4-1 The Flow of Mining Frequent Itemsets

使用逐层迭代找出频繁项集，给定事务数据库 D，最小支持度阈值，要求输出 D 中的频繁项集 L，在 Apriori 的第一步找出频繁 1-项集的集合 L1，Lk-1 用于产生候选 Ck，以找出 Lk。

4.4 MapReduce 下的 Apriori 算法

Apriori 使用的一种称作逐层搜索的迭代方法，k-项集用于搜索(k+1)-项集。首先，找出频繁 1-项集的集合。该集合记作 L1。L1 用于找频繁 2-项集的集合 L2，而 L2 用于找 L3，如此下去，直到不能找到频繁 k-项集。找每个 Lk 需要一次数据库扫描。

MapReduce 模式的主要算法是将自动分割要执行的问题，拆解成 Map 和 Reduce 的方式在数据被分割后通过 Map 函数的程序将数据映射成不同的区块，分配给计算机集群系统处理达到分布式运算的效果，在通过 Reduce 函数的程序将结果汇整，输出需要的结果。假设事务数据库水平分割成不相交的数据块，分散存储在集群系统或 PC 服务器上，我们将一段实现 Apriori 算法的 Map 函数的 Java 程序部署到各分块的结点。在每个结点基于自己的数据分块可以获得项集的局部支持度，生成<key,value>对，其中 key 是项集的名称，value 是该项集对应的支持度。然后将生成的<key,value>对发送到一个或多个部署 Reduce 程序的结点，根据 key 值分组，并计算出所有项集的全局支持度，根据阈值找出频繁项集。然后重复执行以上步骤，直到找出全局频繁项集。

设一电子健康档案由 n 个病人记录{T1，T2,...，Tn}组成，每个记录 Ti（i∈1,2，...，n）由若干项组成（识别号，年龄，姓名，病种...）组成，记作 I1,I2,...，In。项集 I 在 Ti 中出现是指 I 包含的所有元素均属于 Ti。MapReduce 框架下的 Apriori 算法如图 4-2 所示：

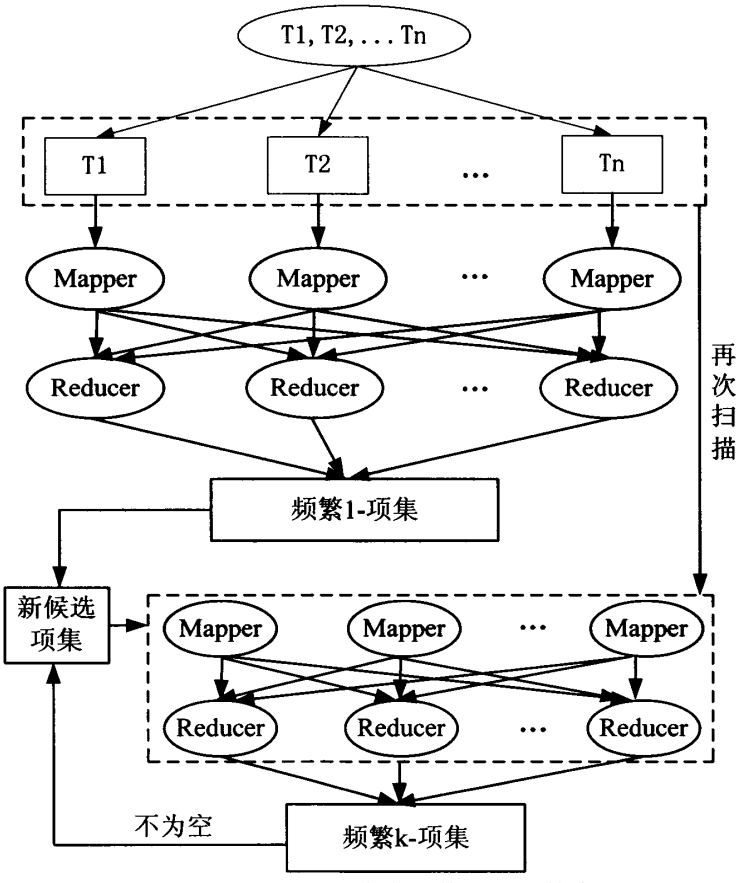

图 4-2 MapReduce 框架下的 Apriori 算法

Fig.4-2 The Apriori Algorithm of MapReduce Architecture

如图 4-2 所示，首先统计 1-频繁项集，1-频繁项集是初始频繁项集。在寻找 1-频繁项集时，所有的元素都是候补频繁项集，对所有的项集进行计数统计，可以在一次 MapReduce 计算中完成，在此过程中设定支持度计数 Fmin。

在 Mapper 过程中，以 key-value 的方式读入数据的块，其具体形式为<key,value =T>，其中 T 是一个病人的记录。每个 Mapper 分别以 T 中的元素 ti 作为 key，每个 key 对应的 value 为 1，若出现一次，则记作<key=ti,value=1>。在 Reducer 过程中，每一个 reducer 获得一个<key=ti,S(ti)>,统计 ti 的出现频率，与 Fmin 比较可得 1-频繁项集。算法描述如下：

（1）Mapper(key,value=T)

foreach T 中的每个元素 ti do

Emit(<key=ti,1>)

（2）Reducer(ti,S(ti))

Sum←0

foreach S(ti)中每个元素 do

Sum←Sum+1

end;

If Sum>Fmin

Emit(<ti,Sum>)

统计 k-频繁项集，由（k-1）-频繁生成 k-候选项集，然后再次扫描数据，统计 k-候选项集中项集的频率。大于 Fmin 的候选项集，形成 k-频繁项集。

在 Mapper 过程中，再次以 key-value 的方式遍历数据，根据（k-1）-频繁项集生成的 k-候选项集。判断 k-候选项集中的每个项集 ci 是否在 T 中出现，如果在 T 中出现 Mapper 产生一个<key=ci,value=1>。在 Reducer 过程中，每一个 reducer 获得一个<key=ci,S(ci)>，统计 ci 出现频率，与 Fmin 比较可得 k-频繁项集。算法描述如下：

（1）Mapper(key,value=T)

Load Candidatae file //载入 k-候选项集文件

Foreach candidate i in Candidate

If candidate i∈T

Emit(<candidate i,1>);

（2）Reducer(key=ci,S(ci))

```
Sum←0
foreach S(ci)中每个元素 do
Sum←Sum+1
end;
If Sum>Fmin
Emit(<Ci,Sum>)
```

当 Apriori 挖掘出的最大频繁项集是 Cn 时，Apriori 最多扫描数据库 n+1 次，因此，部署 Map 程序的结点需要最多执行 n+1 次。而由于 Apriori 的 Reduce 函数仅仅是统计项集的支持度，启动单个部署 Reduce 结点就足以满足 Apriori 的需要。

4.5 性能分析

设 N 表示事务数据库的交易数，T 表示最大交易长度，则：

第一次数据库扫描时间为 $T_I = O(T \times N)$，在接下来每一步中，连接时间开销是 $T_L = \sum_{k\geq 2}(O(|L_{k-1}| \times |L_{k-1}|) + O(|C_k|) + O(N \times |C_k|))$，剪枝时间是 $T_J = O(|C_k|)$，扫描计数时间是 $T_S = O(N \times |C_k|)$。

Apriori 算法总的时间开销为：

$$T_{总} = O(T \times N) + \sum_{k\geq 2}(O(|L_{k-1}| \times |L_{k-1}|) + O(|C_k|) + O(N \times |C_k|))$$

Apriori 算法的空间复杂度为：

$$S = O(|L_1|) + \sum_{k\geq 2}(O(|C_k|) + O(|L_k|))$$

由以上分析可知，Apriori 的复杂性在于每轮迭代挖掘 Ck 时，都需要扫描一次事务数据库 D，当 D 的记录数量急剧增多时，Apriori 的时间复杂度会升高，而且当前的很多应用都将庞大的数据库分散存储在不同结点上。Apirori 算法的时间开销主要集中在频繁 2-项集和频繁 3-项集的生成及候选频繁项目集 C2，C3 的计数连接过程中。本文提出的 MapReduce 框架能减少连接次数进而减少数据库扫描时间，对候选频繁项目集不存储，从而减少内存空间，改善算法的性能。

因为基于 MapReduce 的 Apriori 算法是由局部频繁项集产生全局频繁项集，事务数据库被分成不重叠的 n 部分，时间复杂度是线性的，所以时间复杂度不变，但由于连接次数减少了，从而提高了算法收敛的速度，使得算法的时间开销大大减少。

改进后的算法只需保留频繁项集 Lk 的空间，无需保留候选频繁候选项集 Ck，所以改进的算法的空间复杂度为：$s=O(|L_1|)+\sum_{k\geq 2} O(|L_k|)$

基于 MapReduce 框架的 Apriori 算法对关联规则进行挖掘，实验数据来源于由南京市卫生局牵头的，涵盖市县二甲、三甲医院，涉及的数据包括医院电子健康档案系统，HIS，PACS 系统的医疗数据。利用各个医院的硬件资源搭建一个 Hadoop 的平台，整个平台由各个医院的服务器系统和汇聚到卫生局信息中心的交换机，使用的操作系统为 linux redhat fedora，Java 环境为 jdk-1_5_0-linux，Hadoop 软件版本为 hadoop-0.20.1。

实验创建了一个包含 93,956 条记录和 635 个项的事务数据库，事务数据库有两个属性，事务标识符和项序列，实验结果如图 4-3 所示：

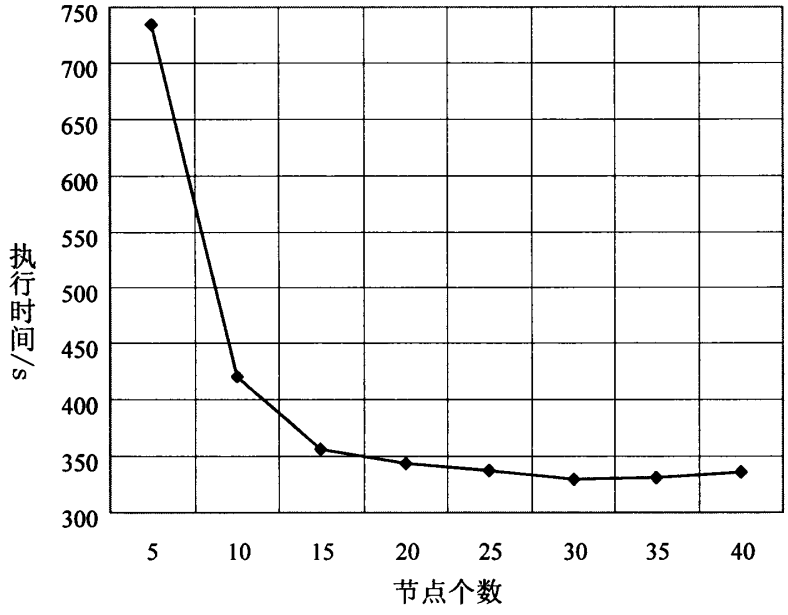

图 4-3 算法执行时间随节点个数的变化

Fig.4-3 The Change of Executing Time Along With The Number of Nodes

在实验开始时，Apriori 算法执行时间随着节点个数增多快速下降。但随着节点个数的持续增多，算法执行时间变化缓慢，当节点个数达到一定数量时，执行时间趋于平稳。因为当结点个数增多时，存储到每个节点的数据记录会相应的减少，Apriori 算法扫描数据库的时间大大减少，从而减少 Apriori 算法的执行时间。当节点数量增加到一定数量时，Apriori 算法扫描数据库的时间会略有上升，这是因为当节点个数到达一定数量后，节点个数对执行时间的影响不能忽略。

4.6 关联规则的产生

从数据库 D 中的事务找出频繁项集后，下一步就是由此产生关联规则(即满足最小支持度和最小置信度)[56]。

对于置信度，可以用公式(4-5)表示，其中条件概率用项集支持度计数表示：

$$confidence(A \Rightarrow B) = P(A \mid B) = \frac{\sup port\_count(A \cup B)}{\sup port\_count(A)} \tag{4.5}$$

在公式 4.5 中 $\sup port\_count(A \cup B)$ 是包含项集 $A \cup B$ 的事务数，support_count(A)是包含项集 A 的事务数。根据 4.5，关联规则可以产生如下：

1、对于每个频繁项集 l，产生 l 的所有非空子集。

2、对于 l 的每个非空子集 s，如果 $\frac{\sup port\_count(l)}{\sup port\_count(s)} \geq \min\_conf$，则输出规则“$s \Rightarrow (l-s)$”，其中 min_conf 是最小置信度阈值。

如表 4.1 的事务数据库 D，假设数据包含频繁项集 l={I1,I2,I5},则 l 的非空子集有{I1,I2},{I2,I5},{I2,I5},{I1},{I2}和{I5}，结果关联规则如下，每个都列出置信度。

confidence(I1 ∧ I2 ⇒ I5)=2/4=50%

confidence(I1 ∧ I5 ⇒ I2)=2/2=100%

confidence(I2 ∧ I5 ⇒ I1)=2/2=100%

confidence(I1 ⇒ I2 ∧ I5)=2/6=33%

confidence(I2 ⇒ I1 ∧ I5)=2/7=29%

confidence(I5 ⇒ I1 ∧ I2)=2/2=100%

设最小置信度阈值为 70%，那么第二、第三和最后一个关联规则是符合要求的，则输出的关联规则是：I1 ∧ I5 ⇒ I2，I2 ∧ I5 ⇒ I1 和 I5 ⇒ I1 ∧ I2。

第 5 章 系统实施

本章对南京市卫生局信息中心的 Hadoop 集群进行、软硬件设置，通过这些设置，来满足我们对海量数据存储的性能和关联规则挖掘的要求。实施架构如图 5-1 所示：

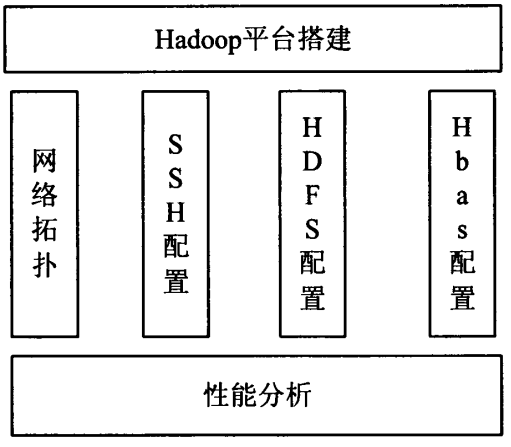

图 5-1 系统实施框架

Fig.5-1 The Implementary Frame of System

5.1 介绍集群的拓扑结构，5.2 介绍集群的软件配置，包括 hadoop 平台搭建，hdfs 配置，hbase 配置。

5.1 硬件配置

Hadoop 被设计运行在商业硬件上，这意味着硬件不再受限于昂贵的硬件，甚至可以从任意大范围的卖家中选择标准的的且普通可用的硬件来创建自己的集群系统。“一般”并不意味着“低档”，低档机器往往是便宜的原件，错误率高。在操作十台甚至成百上千台机器时，更高的错误率会导致更高的维护费用。Hadoop 主体是用 Java 设计的，因此它能运行在任意一个由 JVM 的平台上，在所使用的机器上需要安装 Java 6 或更新版本，在电子健康档案系统中 Hadoop 使用的是 Sun 公司的 JDK。

南京市卫生局的电子健康档案系统的 Hadoop 集群结构由一个两阶网络拓扑构成，如图 5-1 所示：

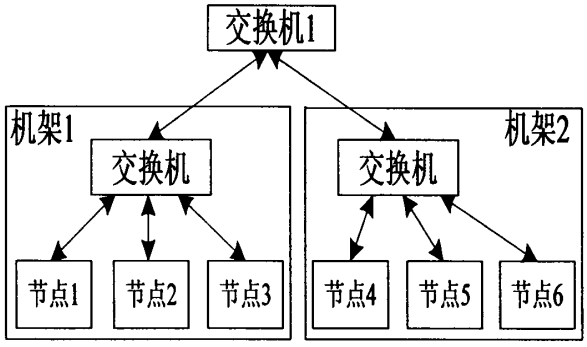

图 5-2 网络拓扑结构图

Fig.5-2 The Topologic Structure of Network

如图 5-2 所示，每个机架有 30~40 个服务器，带有一个 1GB 的交换器(图中只展示了三个)，并向上传输到一个核心交换器或者路由器(正常是 1GB 或更多)，由图可知，在相同机架中的节点间的带宽总和比不同机架中的节点间的要更大。在多机架的集群，配置 Hadoop 时需要映射节点到机架上。通过映射，放置 MapReduce 任务在节点中时，Hadoop 将优先做机架内传输而不是机架外传输(有更多的带宽可用)，HDFS 能更智能地放置副本。

网络位置(如节点和机架)可以表示成一棵树，它反映了网络中位置之间的“距离”。名称节点在决定哪里存放块的副本时，会用到网络位置；当一个 map 任务被分配到一个 tasktracker 上运行时，jobtracker 节点会使用网络位置来确定作为 map 任务输入最近副本的位置。在图 5-2 中，机架拓扑是由两个网络位置来描述的，即/交换机/机架 1 和/交换机 1/机架 2。因为在集群中只有一个最高级交换器，位置可以简化成/交换机 1/机架 2。

Hadoop 配置指定了一个在节点地址和网络位置间的映射，这个映射由一个 Java 接口描述，DNSToSwitchMapping 的接口名如下：

```
public interface DNSToSwitchMapping{
  public List<String> resolve(List<String> names);
}
```

参数 names 是 IP 地址的列表，返回值是一个对应网络位置的字符串列表。topology.node.switch.mapping.impl 配置属性定义一个 DNSToSwitchMapping 接口的实现，名称节点和 jobtracker 用它处理工作者节点网络位置。

对于图 5-1 所示的网络，我们将映射节点 1、节点 2 和节点 3 到/rack1，映射节点 4、节点 5 和节点 6 到/rack2。

大多数设置不需要自己实现接口，因为默认实现是 ScriptBasedMapping，它运行一

个用户定义脚本去决定映射。这个脚本的位置由 topology.script.file.name 的属性控制。此脚本必须传入一个可变数量的参数即被映射的主机名或 IP 地址，并且它必须生成相应的网络位置给标准输出，用空格分开。

5.2 软件配置

在电子健康档案系统中安装 Hadoop，为了缓解安装和维护每一个节点上相同的软件的负担，使用一个自动化安装方法 Red Hat Linux'Kickstart 来完全自动化安装。这些工具通过记录在安装过程(例如磁盘分区)中对于选项的回答来使其自动化操作系统安装，就像安装包一样，同时提供在此过程最后运行脚本的钩子，这对执行那些不含在标准安装中的最终系统调整和自定义是非常重要的。

安装完 JDK 以后，使用以下指令确定 Java 是否正确安装：

```
% java –version
java version "1.6.0_12"
Java(TM) SE Runtime Environment (build 1.6.0_12-b04)
Java HotSpot(TM) 64-Bit Server VM (build 11.2-b01, mixed node)
```

Hadoop 机群包含有无数服务器，两台是主服务器(其中一台作为热备份)，在上面部署 HDFS 的名字节点和 MapReduce 的 JobTracker。其他机器则部署 HDFS 的数据节点与 MapReduce 的 TaskTracker，必须确保每台机器的主机名和 IP 地址之间能正确解析。如果该台机器作名字节点用，则需要在 hosts 文件中加上机群中所有机器的 IP 地址及其对应的主机名；如果该台机器作数据节点用，则只需要在 hosts 文件中加上本机 IP 地址和名字节点机器的 IP 地址。

对于 HDFS 而言，节点分为名字节点和数据节点，其中名字节点只有一个，数据节点可以是很多。对于 MapReduce，节点又分为 JobTracker 和 TaskTracker，其中 JobTracker 只有一个，TaskTracker 可以是很多。在项目中将名字节点和 JobTracker 部署在主服务器上，其他机器作为数据节点和 TaskTracker。

5.2.1 SSH 配置

Hadoop 控制脚本依靠 SSH(Secure Shell)来执行集群范围内的操作，如停止和开始所有集群中的后台程序的脚本，为了操作流畅，SSH 需要设置为允许集群中节点之间执行命令时不需要输入密码的，因此我们需要配置 SSH 使用无密码公钥认证。

这一步非常关键，必须保证 authorized keys 只对其所有者读写权限，其他人不允许有写的权限，否则 SSH 是不会工作的。在系统中设置 ssh 访问控制通过 RSA 方式验证，然后把 authorized_keys 文件复制到其他待访问机器的/snv/home/ssh 下这样就建立了对于当前用户无密码访问其他机器的环境。一旦成功在所有机器上创建 ssh keys 后，就可以开始在从节点上部署 Hadoop 了。

5.2.2 Hadoop 平台搭建

Hadoop 要求所有机器上 Hadoop 的部署目录结构要相同，并且都有一个相同的用户名和账户集群中所有机器都有一个 snv 的账户，主目录是/home/snv。Hadoop 包含了运行命令和控制整个集群中后台程序的开始和停止的脚本，在 hadoop 中有两个文件 masters 和 slaves，每一个都包含机器的主机名或者 IP 地址的列表。masters 决定哪些机器将运行第二名称节点，slaves 文件列出了运行数据节点和 tasktracker 的机器。masters 和 slaves 文件都在配置目录下，通过改变在 hadoo-env.sh 中的 HADOOP_SLAVES 的设置，slaves 可以放在别处。同样，这些文件不需要分布给工作节点，因为只有运行在名称节点或者 jobtracker 上的控制脚本使用它们。Hadoop 部署目录结构如下：

1、/home/snv/hadoopInstall，存放所有 hadoop 版本。

2、/hadoop/conf/，存放所有的配置文件。

3、/home/bin，存放所有的执行目录。

在/home/snv/hadoopInstall/hadoop-conf 目录下的 hadoop_env.sh 中设置 Hadoop 需要的环境变量，其中 JAVA_HOME 是必须设定的变量。HADOOP_HOME 变量的设定是可选的，不设定的话 HADOOP_HOME 默认的是 bin 目录的父目录，即/home/snv/hadoopInstall/hadoop。设置命令如下：

export HADOOP_HOME=/home/snv/HadoopInstall/hadoop

export JAVA_HOME=/usr/java/jdk1.6.0

在 hadoo-conf/目录下，打开 slaves 文件，该文件用来指定所有的名字节点，一行指定一个主机名。

Hadoop 的所有配置项都存放在 conf/目录中的 hadoop-default.xml 文件里，但是不允许直接修改这个文件。在 hadoop-conf/目录下的 hadoop-site.xml 里面可以定义我们需要的项，其值会覆盖 hadoop-default.xml 中的默认值，也可以根据自己的实际需要来进行定制。在项目中的 hadoop-site.xml 设置如下：

```
<configuration>
  <property>
      <name>fs.default.name</name>
      <value>hdfs:// zw-hadoop-master:9000</value> //namenode 的配置，机器名加端口
  </property>
  <property>
    <name>Mapred.job.tracker</name>
    <value> zw-hadoop-master:9001</value>
  </property>
  <property>
    <name>dfs.name.dir</name>
    <value>/home/hadoop/hdfs/name</value>  //namenode 持久存储名字空间及事务日志的本地文件系统路径
  </property>
  <property>
    <name>dfs.data.dir</name>  //Datanode 存放块数据的本地文件系统路径，逗号分隔的列表。
    <value>/home/hadoop/hdfs/data</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/data/hdfs/hdfstmp</value>
    <description> A base for other temporary directories.</description>
  </property>
  <property>
    <name>dfs.replication</name>//数据需要备份的数量，设置为 3
    <value>1</value>
  </property>
</configuration>
```

Hadoop 默认分配 1GB 的内存给它运行的每个后台程序，这是由 hadoop-env.sh 中的环境变量 HADOOP_HEAPSIZE 控制的，而 tasktracker 启动独立的 JAVA 虚拟机来运行 map 和 reduce 任务。在名字节点服务器上 Hadoop 的环境变量的配置好之后，接下来将 Hadoop 部署到其他数据机器上，并且保证目录结构一致。这样我们就搭建好了 Hadoop 平台。启动前要先格式化名字节点，先进入~/hadoopInstall/hadoop 目录，然后执行下面的命令：

[snv@snv-1:hadoop]$bin/hadoop namenode –format

再启动启动 Hadoop 之前，可以参考 bin/下面有众多启动脚本，从而选择我们所需要的启动项。hadoop 主要配置如表 5.1 所示：

表 5-1 Hadoop 配置表

Table 5-1 the configuration table of Hadoop

脚本名	功能
start-all.sh	启动所有的 Hadoop 环境，包括 NameNode，DataNode，JobTracker，TaskTracker
stop-all.sh	停止所有 Hadoop
start-Mapred.sh	启动 MapReduce 环境，包括 JobTracker 和 TaskTracker
stop-Mapred.sh	停止 MapReduce 环境
start-dfs.sh	启动 HDFS 环境，NameNode 和 DataNode
stop-dfs.sh	停止 HDFS 环境
core-site.xml	Hadoop 核心的配置，例如 HDFS 和 MapReduce 中很普通 I/O 设置
hdfs-site.xml	HDFS 后台程序设置的配置：名称节点，第二名称节点和数据节点
mapred-site.xml	MapReduce 后台程序设置的配置：jobtracker 和 tasktracker

5.2.3 HDFS 配置

HDFS 采用 master/slave 架构。一个 HDFS 集群是由一个 Namenode 和一定数目的 Datanodes 组成。Namenode 是一个中心服务器，负责管理文件系统的名字空间(namespace)以及客户端对文件的访问。集群中的 Datanode 一般是一个节点一个，负责管理它所在节点上的存储。HDFS 暴露了文件系统的名字空间，用户能够以文件的形式在上面存储数

据。从内部看，一个文件其实被分成一个或多个数据块，这些块存储在一组 Datanode 上。Namenode 执行文件系统的名字空间操作，比如打开、关闭、重命名文件或目录。它也负责确定数据块到具体 Datanode 节点的映射。Datanode 负责处理文件系统客户端的读写请求。在 Namenode 的统一调度下进行数据块的创建、删除和复制。

HDFS 的具体配置如下所示：

1、配置 slaves，修改 conf/slaves/data-dfs，复制 hadoop 目录以及所有文件到 DataNode 和 Client。

scp-rp/data/soft/hadoop.***/data/soft　//***为 IP 地址

scp-rp/data/soft/hadoop.***/data/soft

2、格式化 Hdfs 文件系统 namenode，进入 hadoop 目录

[master@hadoop]$bin/hadoopnamenode-format 输入 Y，(注意区分大小写.这里一定要输入大写的 Y，否则不会成功 format 文件系统)。如果不成功，就去 hadoop/logs/目录下去查看日志文件。

3、启动 Hdfs 分布式文件系统

Hadoop Hdfs 配置时还要启动 Hdfs 分布式文件系统，进入 hadoop 目录，在 bin/下面有很多启动脚本，如表 5.1 所示，根据需要选择启动项。

[master@hadoop]bin/start-all.sh//会自动找到 conf/slaves 文件里的 IP 或者机器名，启动相应的 slaves 服务端同样，如果要停止 hadoop，则[master@hadoop]$bin/stop-all.sh。

4、HDFS 操作

Hadoop Hdfs 配置过程中对 HDFS 进行的操作，进入 Hadoop 目录,运行 bin/目录的 hadoop 命令，可以查看 Haoop 所有支持的操作及其用法，以几个简单的操作为例。

a、建立目录

[master@hadoop]$bin/hadoopdfs-mkdirtestdir，在 HDFS 中建立一个名为 testdir 的目录。

b、复制文件

[master@hadoop]$bin/hadoopdfs-put/home/dbrg/large.ziptestfile.zip，把本地文件 large.zip 拷贝到 HDFS 的根目录/user/dbrg/下，文件名为 testfile.zip。

c、查看现有文件

[master@hadoop]$bin/hadoopdfs-ls 运行 hadoop 程序时，中途我把它终止了，然后再向 hdfs 加文件或删除文件时，出现 Namenodeisinsafemode 错误：

rmr:org.apache.hadoop.dfs.SafeModeException:Cannotdelete/user/hadoop/input.Namenodeisinsafemode。解决的命令：bin/hadoopdfsadmin-safemodeleave#关闭 safemode。

5.2.4 Hbase 环境搭建

基于现有的 Hadoop 集群，来搭建 Hbase 的环境，Hbase 搭建非常简单，首先下载 Hbase 源码，并解压。在主节点上，编辑 Hbase 目录(指/home/hadoop/Hbase)下的 conf/Hbase-env.sh，设置好 JAVA_HOME 环境变量。

在主节点上，编辑 Hbase 目录(指/home/hadoop/Hbase)下的 conf/Hbase-site.xml 如下：

```
<configuration>
  <property>
    <name>Hbase.rootdir</name>
    <value>hdfs://zw-hadoop-master:9000/Hbase</value>
    <description>The directory shared by region servers.</description>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
    <description>The mode the cluster will be in. Possible values are
      false:standalone and pseudo-distributed setups with managed Zookeeper
      true:fully-distributed with unmanaged Zookeeper Quorum
    </description>
  </property>
  <property>
    <name>hbase.master</name>
    <value>hdfs://zw-hadoop-master:6000</value>
  </property>
</configuration>
```

几个配置的说明：

1、hbase.rootdir 设置 hbase 在 hdfs 上的目录，主机名为 hdfs 的 namenode 节点所在的主机。

2、hbase.cluster.distributed 设置为 true，表明是完全分布式的 hbase 集群

3、hbase.master 设置 hbase 的 master 主机名和端口。

在主节点上，边界 Hbase 目录(指/home/hadoop/Hbase)下的 conf/regionservers，设置 regionservers 的服务器，指定 HRegionServer 机器所在的 ip，每个一行。当然需要把 Hbase 的程序目录分发到被指定为 HRegionServer 所在的机器上，hbase 默认只有一个 master，我们也可以启动多个 master，但是，其他的 master 只有在主 master down 掉后才会工作，到此 Hbase 数据库已经配置完毕。

5.2.5 与客户端系统结合

在客户端编写 JAVA 程序操作 HBase，需要引入的 JAR 如下：hadoop-0.20.1-core.jar，commons-logging-1.0.4.jar，commons-logging-api-1.0.4.jar。

为了与可视化程序结合，目前采用 TCP socket 进行通信的方案，并设计了一套通信协议。服务器监听某个端口，SNV 可视化程序与服务器建立连接后，可以发送各种请求，如建立数据表、查询用户信息等。

在 SNV 程序客户端，用户新建一个 hadoop 项目，项目配置页面中指定 HbaseMaster 服务器所在的 ip 和端口，以及 SNVServer 所在服务器 ip 和端口。在与客户端交互时，使用最频繁的两个操作 columns 与 select。

1、colunms 操作：查询某一表的所有列族名，返回表中所有的列族名。查询格式如下：

columns tableName

返回结果：

columnFamily1

columnFamily2

…

columnFamilyN

通过 columns 操作，可以将一个列簇对应的列明略列出来，对于电子健康档案数据存储表来说，就会将 Age，Name，Sex，illType 四个列簇名展示给用户，帮助用户了解相关的电子健康档案相关内容。

2、select 操作：查询数据。

我们的客户端可视化系统提交到查询主要是针对电子健康档案数据存储表，查询格

式如下：

Select

TableName //要操作的表明

ColumnsFamilyName/./查询的列簇名

Data or Label //取数据或者列标签名

Rowkey//行关键字

Collabel//列标签，可以是空

Rowkey 对应行关键字，在我们的分析系统中，一般客户端传递给 Hadoop 平台与 Hbase 数据库就是一系列病人的编号或者是唯一识别的 ID，这些编号就是对应的 RowKey。

客户端与 Hadoop 平台之间交互最需要注意的一点就是负载平衡，当查询结束后，数据实际上是直接从 DataNode 上返回给了客户端。如果有些节点数据量传输很大，有些则处于闲置状态必然会造成系统的失衡。这样就影响了 Hadoop 平台在分布式处理方面的优势。

5.3 运行结果与分析

5.3.1 不同数据规模的实验

我们的实验都是单层扩展，首先考察数据规模对 Hadoop 平台性能的影响。数据源分别为 10 万、100 万、500 万、1000 万、5000 万。初始节点个数为 100。为了减少单次实验的偶然性，表 5.2 的实验结果是 3 次实验的平均值。

表 5-2 不同数据规模扩展实验用时比较

Table 5-2 The Comparison Table of Time-consuming By Different Sizes Data Set

数据量	Oracle 单机版系统	Hadoop 版本系统	节约时间
10 万	9.8 秒	11.5 秒	-1.7 秒
100 万	69.8 秒	65.4 秒	4.4 秒
500 万	112.9 秒	99.7 秒	13.2 秒
1000 万	205.6 秒	179.5 秒	26.1 秒
5000 万	648.5 秒	567.4 秒	81.1 秒

我们用折线图更能直观的看到两者之间的差别，如图 5-3 所示，其中纵坐标是所需时间，纵坐标表示数据的大小。

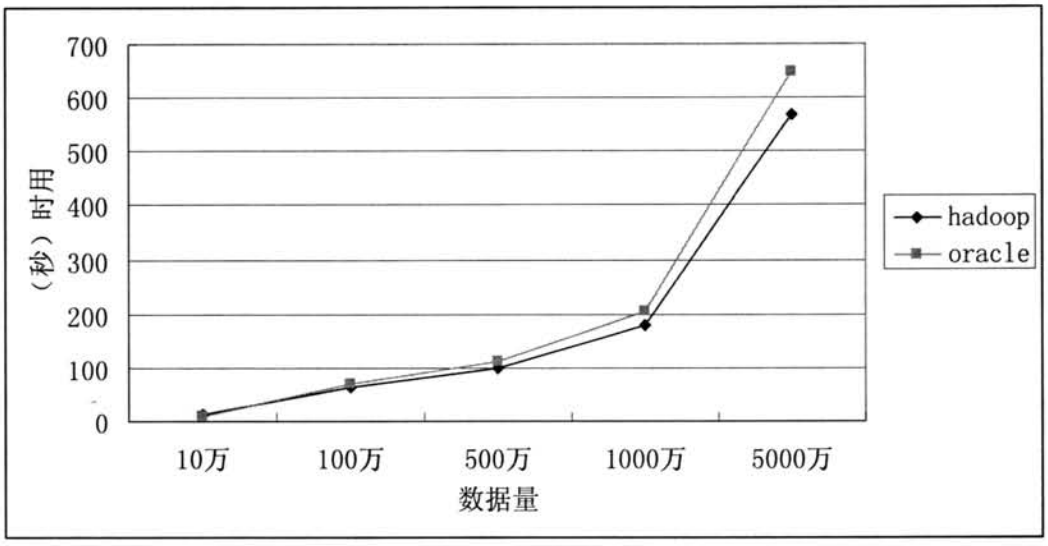

图 5-3 不同数据规模实验结果对比图

Figure.5-3 The Conract of Time-consuming By Different Sizes Data Set

从图 5-3 可以看出随着数据量的增多，Hadoop 系统节约的时间随之增大。与之相对应的是，在数据增加的时候，Oracle 系统用时增加幅度高于 Hadoop。但是数据量减少的时候，Hadoop 系统的扩展效率反而不如 Oracle。这是因为 Hadoop 平台只有在处理超大规模数据的时候才能显示出其优势。而处理小规模的数据集合的时候，反倒还不如传统的关系数据效率高。当处理数据时，节点会随着数据的增加而增加，而新增节点会影响系统的性能。这主要是因为我们将去重操作放在 Combine 函数中处理，影响了处理的效率。但这个影响相对于 Hadoop 平台提高的效率可以忽略。

因此我们得出的结论是 Hadoop 平台在处理大数据集时是有优势的，但是在处理小规模数据的时候，我们不能忽略任务调度使用的时间，所以在处理小规模数据时，Hadoop 平台并不如传统的关系型数据库更具优势。

5.3.2 不同任务粒度的实验

现在我们来考核一下工作粒度对实验 Hadoop 平台性能的影响。我们保持数据源不变，取 5000 万条记录。初始节点为 100 个。记录大小不等，经过我们的实验总结，我们计一条记录为 1k，因此 5000 万条记录为 50G 的数据。有实验一我们知道，Oracle 扩展时间是 648.5 秒。下面我们看一下对不同任务粒度扩展出的节点情况：

表 5-3 不同任务粒度扩展实验用时比较

Table 5-3 The Comparison Table of Time-consuming By Different Task Granularities

Map/Combine/Reduce 任务个数	耗费时间	节约时间
20/20/15	625.3 秒	23.2 秒
25/25/20	619.7 秒	28.8 秒
30/30/20	609.2 秒	39.3 秒
40/40/35	615.4 秒	33.1 秒
50/50/45	620.6 秒	27.9 秒

我们用折线图来区分两者，如图 5-4 所示，为了方便表示我们用 tn1，tn2，tn3，tn4,tn5 表示任务个数 20/20/15,25/25/20,30/30/20,40/40/35,50/50/45。如图 5-3 所示：

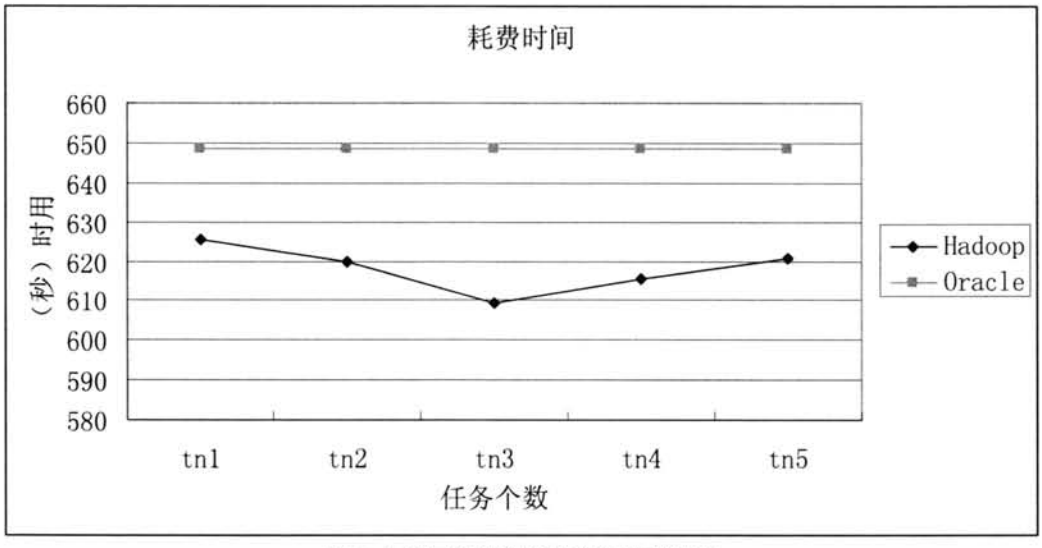

图 5-4 不同任务粒度实验结果对比图

Figure.5-4 The Comparison Chart of Time-consuming By Different Task Granularities

如图 5-4 所示：在实验开始时，随着部署的节点个数的增加，系统用时在减少。但是这里存在着一个阈值，当达到这个阈值时，任务个数继续增加，耗费时间反而随着增加。因此可以推出任务粒度会影响平台性能，Hadoop 节点个数并不是越多越好，当数据量不大但节点个数偏多时，不能忽略任务调配时间对整个系统效率的影响。

5.4 后续展望

目前只是涉及到很少一部分医院的医疗数据，如何扩大到全市所有的医院，还有待进一步的研究。

总体上讲，云计算领域的研究还处于起步阶段，尚缺乏统一明确的研究框架体系，还

存在大量未明晰和有待解决的问题，研究机会、意义和价值非常明显。现有的研究大多集中于云体系结构、云存储、云数据管理、虚拟化、云安全、编程模型等技术，但云计算领域尚存在大量的开放性问题有待进一步研究和探索。

下一步要讨论的工作是关于 Hadoop 中配置信息可以不是单一的、全局的位置，集群中的每一个 Hadoop 节点都有自己的配置文件集，并由管理者确保在整个系统中配置同步，Hadoop 通过使用 rsync 为同步配置提供一个基本的设备。

Hadoop 允许一个配置文件集供所有主控机和工作机所使用，这样最大的优点就是简洁，统一(只是一个配置)和可操作性(Hadoop 脚本足以管理任意单一配置的创建)。

但对于某些集群，一体使用的配置模式就不行了，例如，如果使用与现有机器不同硬盘规格的新的机器扩展集群，就需要为新机器进行不同的配置来使用它们额外的资源。

在这些情况下，我们需要机器类型的概念，并为每种类型维护一个单独的配置。Hadoop 没有提供工具来做这些事情，但是有一些非常好的工具可以精确地做这种类型的配置管理，例如 Puppet，cfengine 和 bcfg2.

对任意大小的集群来说，保持所有机器同步是一个挑战：在应用一个更新时，有机器失效怎么办？等它变得有用时，谁又能确保它能得到更新？这是个严峻的问题，又会引发安装的不统一，所以即使使用 Hadoop 控制脚本管理 Hadoop，为维护集群使用配置管理工具仍不失为一个好的想法。这些工具做一些常规维护也很有效的，例如安全漏洞补丁和更新系统包。

参考文献

[1] 王鹏. 走近云计算[M]. 北京: 人民邮电出版社，2009.

[2] 余顺争, 谢长生. 融合NAS和SAN的存储网络设计与实现[J]. 电子学报, 2006, 34(11): 2012-2017.

[3] Wan Jiguang, Zhan Ling. Research and Implentmentation of a NAS Cluster Based Network Backup System[J]. Mini-Mi -cro Systems, 2005, 26 (6) ,905-908.

[4] 王迪, 舒继武, 薛巍, 沈美明. 基于块级别的 SAN 系统自适应分级存储[J]. 高技术通讯, 2007, 17(2): 111-115.

[5] 张健. 云计算概念和影响力解析[J]. 电信网技术, 2009(1): 15-18.

[6] 吴鹤龄, 崔林. ACM 图灵奖-计算机发展史的缩影[M]. 北京: 科学出版社, 2000.

[7] 王鹏. 云计算的关键技术与应用实例[M]. 北京: 人民邮电出版社, 2010.

[8] 陈国良, 安虹, 陈绫. 并行算法实践[M]. 北京: 高等教育出版社, 2004.

[9] 许俊, 史美林, 李玉顺. 网格计算与 E-Learning Grid[M]. 北京: 科学出版社, 2005.

[10] Grossman R, Gu Y. Data mining using high performance data clouds: experimental studies using sector and sphere[M]. Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, 2008, 920-927.

[11] 王鹏. 移动搜索引擎原理与实践[M]. 北京: 机械工业出版社, 2009.

[12] 张军. 分布式系统技术内幕[M]. 北京: 首都经济贸易大学出版社, 2006.

[13] 何艳辉, 朱珍民. 基于消息传递并行计算环境[J]. 湘潭大学社会科学学报, 2003, 27(5): 233-235.

[14] Pathak V, Iftode L. Byzantine fault tolerant public key authentication in peer-to-peer systems[J]. Computer Networks, 2006, 50(4): 579- 596.

[15] Robert Walker Cooley.Web Usage Mining: Discovery and Application of Interesting Patterns From Web Data[D]. Ph.D. Dissertation, University of Minnesota, 2000.

[16] Stribling J. OverCite: lecture notes in computer science[M]. Berlin: Springer, 2005.

[17] 王鹏. 并行计算应用与实战[M]. 北京: 机械工业出版社, 2009.

[18] 林立宇, 陈云海, 张敏, 等. 云计算技术及运营可行性分析[J]. 广东通信技术, 2008(12): 33-38.

[19] (美)Abraham Silberschatz. 数据库系统概念[M]. 机械工业出版社, 2000.

[20] Szalay A, Bunn A, Gray J, et al. The Importance of Data Locality in Distributed Computing Applications[A]. NSF Workflow Workshop, 2006.

[21] Zhao B, Kubiatowicz J, Joseph A. Tapestry: an infrastructure for fault-tolerant wide-area locat ion and routing[R]. California: Berkeley Computer Science Division, 2001.

[22] FU Xiang-lin, XIE Chang-sheng, LIU Rui-fang. The architecture and performance evaluation of iSCSI-based united storage network merging NAS and SAN[J]. International Journal of Computer Applications in Technology, 2004, 19(2): 84-92.

[23] Calvin Lin and Lawrence Snyder. 并行程序设计原理[M]. 北京: 机械工业出版社, 2009.

[24] 李斌, 谭立湘, 章劲松, 庄镇泉. 面向数据挖掘的时间序列聚类方法研究[J]. 计算机科学, 2000(12): 95-97.

[25] 熊雯琳. 云计算服务是发展趋势[N]. 电脑报, 2009 - 04 - 20 (22).

[26] Morris J H, Satyanarayanan M, Conner M H, et al. Andrew: a distributed personal computing environment [J]. Communications of the ACM, 1986, 29(3):184–201.

[27] 都志辉. 高性能计算之并行编程技术 MPI 并行程序设计[M]. 北京: 清华大学出版社, 2001.

[28] 郭玉东, 尹青. 基于对象的网络存储[M]. 北京: 电子工业出版社, 2007.

[29] Hu J, Marculescu R. Energy and Performance Aware Mapping for Regular NoC Architectures[J]. IEEE Trans on CAD of Integrated Circuit s and Systems, 2005, 24 (4): 551-562.

[30] 卢俊. 高性能计算机并行文件系统[M]. 长沙: 国防科技大学出版社, 2005.

[31] 冯丹. 网络存储关键技术的研究及进展[J]. 移动通信, 2009(11): 34-39.

[32] 毛国君, 段立娟, 王实. 数据挖掘原理与算法[M]. 北京: 清华大学出版社, 2006.

[33] 张建明, 荣冈. 基于关联规则的故障诊断方法及研究[J]. 化工自动化及仪表, 2003(05): 32-34.

[34] Satyanarayanan M, Kistler J J, Kumar P, et al.Coda: a highly available file system for a distributed workstation environment [J]. IEEE Transactions on Computers, 1990, 39(4): 447–459.

[35] 陈艳, 郝丽蕊. 基于 WinPcap 的海量数据接收与保存[J]. 软件导刊, 2008, 7(10): 87-89.

[36] 卢建华, 蒋明, 陈淑芳. 网络数据包捕获及分析[J]. 网络安全技术与应用, 2009(2): 16-18.

[37] 李占波, 李娜. XML 数据在关系数据库中的存储[J]. 微计算机信息, 2007, 23(9-3): 192-194.

[38] 崔清华. XML 文档在关系数据库中的存储研究[J]. 微计算机信息, 2007, 23(4-3): 184-186.

[39] 高明霞, 董英斌, 陈福荣. 一种 XML Schema 到关系数据库模式转换算法及实现[J]. 计算机应用研究, 2003, 6: 154-157.

[40] Miller E L, Katz R H. RAMA: an easy-to-use highperformance parallel file system [J]. Parallel Computing, 1997, 23(4): 419–446.

[41] 王春花, 黄厚宽, 李红莲. 一种快速有效的分布式开采多层关联规则的算法[J]. 计算机研究与发

展, 2001, 38(4): 438-443.

[42] 陈耿, 倪巍伟, 朱玉全. 基于分布数据库的快速关联规则挖掘算法[J]. 计算机工程与应用, 2006(4): 165-167.

[43] 朱明. 数据挖掘[M]. 合肥: 中国科技大学出版社, 2002.

[44] 景永霞, 王治和, 苟和平. 基于分布式数据库的关联规则挖掘算法[J]. 湛江师范学院学报, 2007, 28(6): 74-77.

[45] 黄贤英, 王柯柯, 范伟. 基于星型网络的分布式关联规则挖掘算法研究[J]. 计算机科学, 2007, 34(12): 180-181, 188.

[46] Gu Yunhong, Grossman R L. UDT: UDP-based Data Transfer for High-CCPEed Wide Area Network[J]. Computer Networks, 2007, 51(7): 777-799.

[47] 伊卫国, 卫金茂, 王名扬. 挖掘有效的关联规则[J]. 计算机工程与科学, 2005, 27(7): 91-93.

[48] 窦祥国, 胡学刚. 关联规则的评价方法研究[J]. 安徽技术师范学院学报, 2005, 19(4): 44-47.

[49] 罗可, 吴杰. 关联规则衡量标准的研究[J]. 控制与决策, 2003, 18(3): 277-280.

[50] 马建庆, 钟亦平, 张世永. 基于兴趣度的关联规则挖掘算法[J]. 计算机工程, 2006, 32(17): 121-122.

[51] 赵振, 谢长生, 蒋思宇. 自适应存储海量系统中的一种文件级可调整 RAID[J]. 计算机工程与科学, 2007, 29(9): 150-153.

[52] 王春花, 黄厚宽, 李红莲. 一种快速有效的分布式开采多层关联规则的算法[J]. 计算机研究与发展, 2001, 38(4): 438-443.

[53] 徐宝文, 张卫丰. 搜索引擎与信息获取技术[M]. 北京: 清华大学出版社, 2003.

[54] 胡可云, 陆玉昌, 石纯一. 基于概念格的分类和关联规则的集成挖掘方法[J]. 模型学报, 2000, 11(11): 1478-1483.

[55] 丁卫平, 施佺, 管致锦, 等. 基于频繁概念格电子病历关联规则挖掘研究[J]. 微电子学与计算机, 2008, 25(8): 125-128.

[56] 丁卫平. 关联规则挖掘 Apriori 算法的改进及其应用研究[J]. 南通大学学报: 自然科学版, 2008, 7(1): 50-53.

攻读学位期间科研成果

[1] 侯建，帅仁俊，侯文. 基于云计算的关联规则挖掘算法[J]. 化工自动化及仪表，2011，38(05): 579-581.

[2] 侯建，帅仁俊，侯文. 基于云计算的海量数据存储模型[J]. 通信技术，2011，44(05): 163-165.

致 谢

三年的硕士生活即将结束，回顾过去，有太多值得我永远铭记在心的经历。这是我校园生涯的结束，也是我学习生涯的另一个起点。很快我将带着亲人、老师和无数关心我的人对我的期望，开始我新的生活。

论文完成之际，谨在此向所有关心和帮助过我的人表示诚挚的谢意！

首先，感谢我的导师帅仁俊老师。从毕业设计开始帅老师就在各方面给予了我耐心的教诲和帮助，为我创造和提供了许多参与研究和实践的机会，这对我知识的巩固和能力的提高有着极其重要的意义。在论文进行过程中，帅老师时刻关注着我的论文进展，并给了我悉心的指导和宝贵的意见。帅老师严谨的治学精神，平和的待人态度也令我感动和敬佩。感谢电子商务实验室的王有权老师，王老师知识渊博，给我留下了深刻印象，对我的指导和建议将使我受益终身。

同时，还要感谢祖研、张恒、包力和邓小祥同学，在论文方面给予了我很重要的指导，提出了许多有益的建议，为我硕士期间的学习和生活增添了浓厚的色彩。

感谢我的家人给了我最大的精神支持。

最后，向所有评阅本论文的各位专家和教授致以深深的谢意！感谢所有关心、帮助和支持我的人们。

基于云计算的电子健康档案存储方式与关联规则挖掘的研究

作者： 侯建
学位授予单位： 南京工业大学