Google 云计算平台的分析与测试

摘要

当今，云计算作为一种成功的商业模式正在快速的发展，得到了商业界和学术界的大力支持。因为能够提供强大的计算和存储能力，云计算能够以较低的成本完成海量的任务，受到了许多 IT 公司的青睐，纷纷推出了云计算发展战略。

云计算包含两个方面的含义：一方面是底层构建的云计算平台的基础设施，是构建上层应用程序的基础；另一方面是构建在这个基础平台之上的云计算应用程序。本课题正是是从这两个方面分析 Google 云计算平台的原理和使用的。

在基础设施分析方面，首先根据现有的云计算平台，总结出通用的云计算平台体系结构，介绍云计算平台的层次划分以及各个层次使用的技术，认识 Google 云计算平台的体系结构，对 Google 在云计算平台各个层次所使用的技术有一个总体的了解。之后把研究重点放在对 Google 核心技术 GFS、MapReduce、BigTable 的分析上，了解这些技术如何帮助 Google 云计算平台实现按需分配资源、充分利用资源、容易扩展和高可用性的特性。

在云计算应用程序研究方面，基于 Google App Engine 提供的服务编写一个云计算应用程序来测试 Google 云计算平台的各项功能。在理论的基础上结合实践，更加全面了解 Google 云计算平台的原理和使用。

关键字： 云计算；GFS；MapReduce；BigTable；Google App Engin

THE ANALYSIS AND TESTING OF GOOGLE CLOUD COMPUTING PLATFORM

ABSTRACT

Today, the cloud computing as a successful business model is developing rapidly，and has got the commercial and academic support.Because it can provide powerful computing and storage capacity, cloud computing can complete massive task at low cost, many IT companies attache great importance to it , and launch their cloud computing strategy one after another.

Cloud computing consists of two aspects: one is to build the underlying infrastructure for cloud computing platform that is used to build the basis for the upper application; the other hand, is cloud computing applications built on this foundation platform.The research of the Google cloud computing platform in this issuse is also from these two aspects to analysize the principles and use of this platform.

On the hand of analysizing infrastucture ,this issuse summed up a common architecture for cloud computing describing the level of division and the technology used at all levels based on the existing cloud computing platform firstly,then have a look at the Google cloud computing platform architecture to have a general understanding of the technologies used at all levels in the platform. Then the issuse put the focus of the study on the understanding and analysis of these core technologies GFS、BigTable of Google, understanding the principle of how these technologies can help Google's cloud computing platform realize assigning resources based on needs,making full use of resources, easy expansion and high availability features of cloud computing.

On the hand of studying cloud applications ,the work is to write a Google cloud computing platform based on the services provided by Google App Engine to test various functions of platform. On the basis of the theory with practice, I can have a more comprehensive understanding of the principles and user of Google's cloud computing platform.

Keywords : Cloud Computing;GFS;MapReduce;BigTable;Google App Engine

目录

1 绪论

1.1 课题背景与意义

1.1.1 课题背景

云计算无疑是 IT 技术界当前最热点的关键词之一。在云计算提出后的短短几年间，Google、Amazon、Yahoo 等互联网服务商，IBM 、Microsoft 等 IT 厂商都纷纷提出了自己的云计算战略，各电信运行商也对云计算投入了极大的关注，这对云计算的商业价值给予了充分的肯定。同时学术界也对云计算进行深层次的研究，例如 Google 和 IBM 联合宣布推广“云计算”的学术合作计划，包括卡内基梅隆大学、斯坦福、伯克利、华盛顿大学、MIT、清华大学在内的许多高校都参加了这项计划[1,2]，推动云计算的普及，加紧对云计算的研究。

云计算并不是一门新技术，而是融合了分布式计算，网格计算和并行计算，创造性的提出动态管理资源并按需分配的一种新商业模式。云计算利用规模化管理和按需分配的优势，大大提高了资源的利用率，降低了计算、存储、网络等资源的使用成本。Google 宣称，由于使用了云计算技术，其计算成本是竞争对手的 1/100，而存储成本是竞争对手的 1/30[3]。正是由于云计算能够显著降低企业信息化成本，其蕴藏着巨大的潜力。Google、Amazon、Microsoft、IBM 等 IT 厂商都推出了自己的云计算平台，提供了 IAAS、PAAS、SAAS 各个类别的不同服务。云计算很有可能改变当今 IT 界的格局，使得分散化应用向集中化演变，以桌面为核心使用各项应用转移到以 Web 为核心进行各种活动。

Google 可以说是云计算鼻祖，Google 的发展就是依据云计算模式进行的。Google 的各项应用如 Google Search、Google Gmail、Google Maps、Google Earth 等都是建立在 Google 的上百万台廉价的 X86 服务器上的[4]，而能够利用这些廉价服务器资源的正是 Google 的云计算平台。Google 以论文的形式展示了其云计算平台的核心技术原理，为云计算的研究提供了理论依据。Google App Engine 这一平台的发布使用户通过编程的方式使用同 Google Search、Google Map 一样的基础设施，让用户能够结合理论，切实的参加到云计算的实践之中。

1.1.2 课题意义

云计算已经成为一种趋势，不论是对企业还是个人，各种应用系统将运行于云计算平台之上。与传统的虚拟空间只能提供固定的计算能力、存储空间和网络带宽相比，Google 云计算平台作为公有云能够为应用提供强大的按需的运算和存储能力，能够提高资源利用率并降低成本。

本课题通过研究与分析 Google 云计算平台的原理、性能以及 Google App Engine 提供的编程接口，能够有利于应用系统在 google 云计算平台上的部署和管理。并且能够帮助企业借鉴 google 云计算平台先进的思想和原理搭建私有云。

1.2 研究现状

与建立在高性能 UNIX 服务器集群的基础上的大型 IT 系统相比，Google 的云计算平台建立在大量的 X86 廉价服务器集群之上，那么两者在技术架构上也存在着明显的差异。Google 采用了许多独特的技术，如数据中心节能技术、节点互联技术、可用性技术、容错性技术、数据存储技术、数据管理技术、数据切分技术、任务调度技术、编程模型、负载均衡技术、并行计算技术和系统监控技术等[3]。正是这些技术的互相协作，使得 Google 的云计算平台成为一个能够自动管理资源、自动容错的一个有机整体，成为承载 Google 应用的强有力支撑。

Google 云计算平台中，Node 是最基本的处理单元。在 Google 云计算平台的技术架构中，除了少量负责特定管理功能的节点如（GFS Master、Chubby 和 Scheduler 等），所有的节点都是同构的，即同时运行 BigTable Server、GFS Chunk Server 和 MapReduce Job 等核心功能模块，与之相对应的则是数据存储、数据管理和编程模型等 3 项关键技术。

Google 相继发表了三大核心技术 GFS[5]、MapReduce[6]、BigTable[7]的论文，详细的介绍了其实现原理。文献[5]是 GFS 的论文，介绍了 GFS 这一分布式文件系统的设计思想，体系结构以及在数据完整性、一致性的问题上所采用的策略。文献[6]是 MapReduce 的论文，介绍了 MapReduce 这一并行计算的编程模型以及实现。文献[7]是 BigTable 的论文，介绍了 BigTable 这一分布式数据库的存储原理。此外，Google 也发表了许多辅助技术的论文，例如 Chubby[8]锁服务的论文。

做为使用 Google 云计算平台接口的 Google App Engine 服务目前已经可以支持 Python、Java 和 Go 三种语言，提供了存储、邮件、图像处理、Google 账户、内存缓存等多项服务。但是还是存在着很多限制，例如不能使用 Google 的文件系统。随着平台的不断发展，更多得服务将会提供，而编程限制也会逐渐减少。

1.3 本文主要工作

Google 云计算平台是一个由应用服务器群、Bigtable 数据库及 GFS 数据存储服务组成的平台，Google App Engine 可以让开发者在 Google 的基础架构之上运行网络应用程序，能为开发者提供一体化的、可自动升级的在线应用服务。

本文首先对 Google 云计算的 GFS、MapReduce、Chubby 以及 Bigtable 等核心技术进行深入的分析，然后分析 Google App Engine 的应用程序环境、沙盒、数据库、App Engine 服务、开发流程、配额和限制等。最后以所分析的理论知识为基础，设计并实现一个简单的信息系统来测试 Google App Engine 的各项服务。

2 Google 云计算平台概述

2.1 Google 云计算平台的体系结构

2.1.1 云计算体系结构

云计算充分利用网络和计算机技术实现资源的共享和服务，解决云进化、云控制、云推理和软计算等复杂问题，其基础架构可以用云计算体系结构来描述，而云计算的服务层次则从提供服务类型角度描述云计算对应提供的功能或服务，云计算技术层次从云计算软硬件结合角度说明云计算平台的构成。

在云计算中，根据其服务集合所提供的服务类型，整个云计算服务集合被划分成 4 个层次：应用层、平台层、基础设施层和虚拟化层[9-10]。这 4 个层次每一层都对应着一个子服务集合。云计算的服务层次是根据服务类型即服务集合来划分，层次是可以分割的，即某一层次可以单独完成一项用户的请求而不需要其他层次为其提供必要的服务和支持。应用层对应 SaaS 软件即服务如：Google APPS；平台层对应 PaaS 平台即服务如：IBM IT Factory、Google APP Engine；基础设施层对应 IaaS 基础设施即服务如：Amazon EC2、IBM Blue Cloud、Sun Grid；虚拟化层对应硬件即服务结合 Paas 提供硬件服务，包括服务器集群及硬件检测等服务。

云计算技术层次和云计算服务层次不是一个概念，后者从服务的角度来划分云的层次，主要突出了云服务能给我带来什么。而云计算的技术层次主要从系统属性和设计思想角度来说明云，是对软硬件资源在云计算技术中所充当角色的说明。从云计算技术角度来分，云计算大约有 4 部分构成：物理资源层、资源池层、管理中间件层和 SOA 构建层，如图 2-1 所示[11-12]。 其中资源池层和管理中间件层是云计算技术的最关键部分。

SOA 构建层：将云计算能力封装成标准的 web services 服务，并纳入到 SOA 体系进行管理和使用，时用户端与云端交互操作的入口，可以完成用户或服务注册，对服务的定制和使用。

管理中间件层：在云计算技术中，中间件位于服务和服务器集群之间，提供管理和服务即云计算体系结构中的管理系统。对标识、认证、授权、目录、安全性等服务进行标准化和操作，为应用提供统一的标准化程序接口和协议，隐藏底层硬件、操作系统和网络的异构性，统一管理网络资源。其用户管理包括用户身份验证、用户许可、用户定制管理；资源管理包括负载均衡、资源监控、故障检测等；安全管理包括身份验证、访问授权、安全审计、综合防护等；映像管理包括映像创建、部署、管理等。

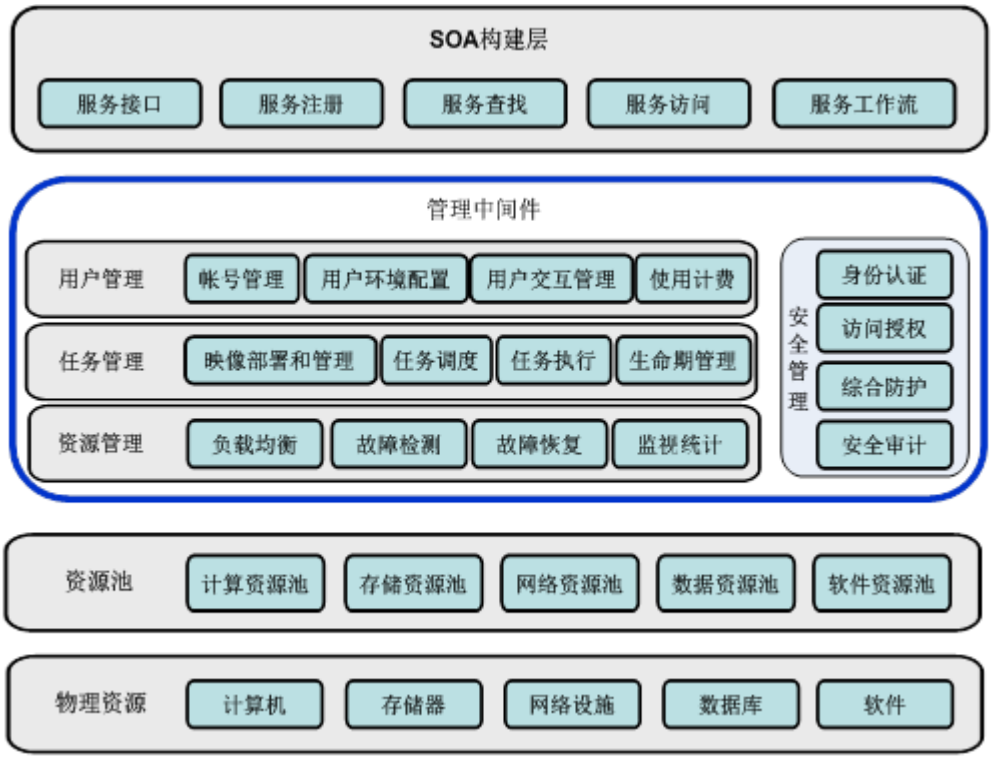

图 2-1 云计算技术层次模型

资源池层：将大量相同类型的资源构成同构或接近同构的资源池，如计算池，存储池和网络池、数据库资源等，构建资源池更多的是物理资源的集成和管理工作，通过软件技术来实现相关的虚拟化功能包括虚拟环境、虚拟系统、虚拟平台。

物理资源层：主要指能支持计算机正常运行的一些硬件设备及技术，可以是价格低廉的 PC，也可以是价格昂贵的服务器及磁盘阵列等设备，可以通过现有网络技术和并行技术、分布式技术将分散的计算机组成一个能提供超强功能的集群用于计算和存储等云计算操作。在云计算时代，本地计算机可能不再像传统计算机那样需要空间足够的硬盘、大功率的处理器和大容量的内存，只需要一些必要的硬件设备如网络设备和基本的输入输出设备等。

2.1.2 Google 云计算体系结构

上一节从服务和技术的层次分别提出了云计算的通用体系结构，各大 IT 厂商的解决方案只实现了其中的部分功能，接下来介绍一下 Google 云计算平台的体系结构。Google 最大的 IT 优势在于它能够建造出一套既富于性价比而又能承载极高负载的高性能系统。因此 Google 认为与竞争对手相比,具有更大的成本优势,同时 Google 程序员的效率比其他 web 同行么高出 50%~100%，原因是 Google 应经开发出了一套专用于大规模并行系统的软件库。从整体看来，Google 的云计算平台包含了如下结构层次[13]：

网络系统：包括内部网络和外部网络。内部网络是用于连接 Google 自建的各个数据中心的网络系统，这一高速的网络系统使得 Google 的每一台服务器连接在一起成为一个负载平衡的集群。外部网络是指在 Google 数据中心之外，有 Google 自己搭建的用于不同国家/

地区，不同应用之间的数据交换网络。

硬件系统：从层次上来看，包括单个服务器、整合了多服务器的机架以及存放、连接各服务器机架的数据中心（IDC）。

软件系统：包括每个服务器上安装的单机操作系统和 Google 云计算底层软件系统（文件系统 GFS、并行计算处理算法 MapReduce、并行数据库 BigTable、并行所服务 Chubby 和云计算消息队列 GWQ）。

Google 内部使用的软件开发工具，包括 C++、Java、Python 等。

Google 发布的可以使用 Python、Java 等编程语言调用云计算底层软件系统的 PAAS 平台—Google App Engine。

Google 自己开发的 SAAS 类型的各项服务，例如 Google Search、Google Gmail、Google Map、Google Earth 等。

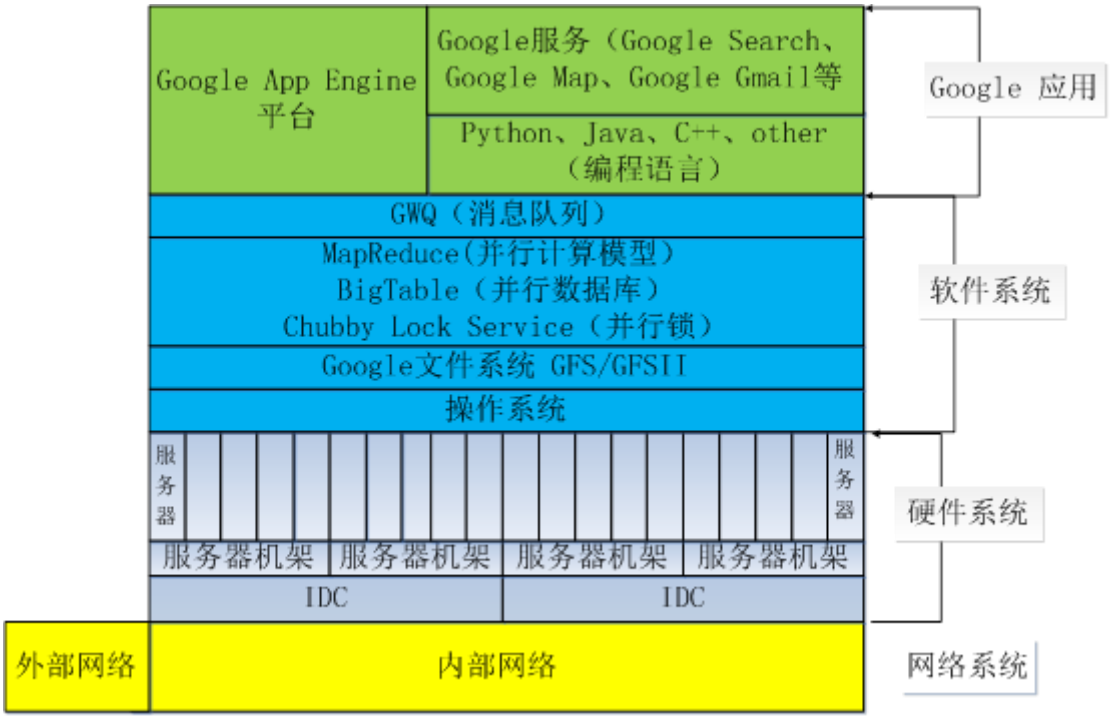

图 2-2 Google 云计算平台体系结构

2.2 Google 云计算平台的应用领域

Google 提出了应用向互联网迁移、数据向互联网迁移、计算能力向互联网迁移、存储空间向互联网迁移这一云计算构想，甚至在未来浏览器可能代替操作系统的作用，直接使用来自互联网的各种软件。从 Google 提供的基于浏览器使用的各项服务，到推出 Chrome OS 网络操作系统，可以看到其在云计算战略中坚定的步伐。然而，以桌面操作系统为中心向以服务器为中心的过渡必然会增加服务器端的计算和存储压力，服务器端必须有处理海量计算和存储的能力才能够支撑。

而 Google 的云计算平台就具有这种能力，它最直接的应用领域便作为基础设施来承载 Google 提供的搜索、邮件、文档、地图等各项互联网应用。随着 Google App Engine 的发

布，不论是简单的个人应用还是企业级应用，都可以构建在 Google 的云计算平台之上。这不仅能让广大用户体验到云计算带来的低成本优势，而且有利于基于互联网的应用的繁荣。

总的来说，Google 提供了 SAAS 和 PAAS 两种类型的服务，不仅可以提供给用户与桌面软件相同体验的互联网应用，而且提供了一个用来创建互联网应用的平台。在 Google 云计算平台的强有力支持下，Google 的服务具有广阔的应用空间。

3 Google 云计算平台核心技术

Google 拥有全球最强大的搜索引擎。除了搜索业务以外，Google 还有 Google Maps、Google Earth、Gmail、YouTube 等各种业务，包括刚诞生的 Google Wave[14]。这些应用的共性在于数据量巨大，而且要面向全球用户提供实时服务，因此 Google 必须解决海量数据存储和快速处理问题。Google 的诀窍在于它发展出简单而又高效的技术，让多达百万台的廉价计算机协同工作，共同完成这些前所未有的任务，这些技术是在诞生几年之后才被命名为 Google 云计算技术。Google 云计算技术具体包括：Google 文件系统 GFS、分布式计算编程模型 MapReduce、分布式锁服务 Chubby 和分布式结构化数据存储系统 Bigtable 等。其中，GFS 提供了海量数据的存储和访问的能力，MapReduce 使得海量信息的并行处理变得简单易行，Chubby 保证了分布式环境下并发操作的同步问题，Bigtable 使得海量数据的管理和组织十分方便。本章将对这四种核心技术进行详细介绍。

3.1 Google 文件系统 GFS

Google 文件系统是一个大型的分布式文件系统，它为 Google 云计算提供海量存储，并且与 Chubby、MapReduce 以及 Bigtable 等技术结合十分紧密，处于所有核心技术的底层。下面先介绍一下分布式文件系统的概念以及 GFS 与其他已有分布式文件系统的区别。

3.1.1 分布式文件系统概述

文件系统是操作系统的一个重要组成部分，通过对操作系统所管理的存储空间的抽象，向用户提供统一的、对象化的访问接口，屏蔽对物理设备的直接操作和资源管理。根据计算环境和所提供功能的不同，文件系统可划分为四个层次，从低到高依次是：单处理器单用户的本地文件系统，如 DOS 的文件系统；多处理器单用户的本地文件系统，如 OS/2 的文件系统；多处理器多用户的文件系统，如 Unix 的本地文件系统；多处理器多用户的分布式文件系统。

本地文件系统（Local File System）是指文件系统管理的物理存储资源直接连接在本地节点上，处理器通过系统总线可以直接访问。分布式文件系统（Distributed File System）是指文件系统管理的物理存储资源不一定直接连接在本地节点上，而是通过计算机网络与节点相连。分布式文件系统是目前最高级的文件系统，它将由网络连接的各个存储节点抽象成为一个统一的存储系统，内部各个存储节点的管理和协作等复杂问题均由系统实现，提供了与最基本的本地文件系统几乎相同的访问接口和对象模型，大大简化了分布式文件系统的使用。正是由于对用户透明地改变了结构，满足用户的需求，以掩盖分布式文件操作的复杂性，才大大增加了分布式文件系统的实现难度。

3.1.2 GFS 的设计思想和目标

GFS 与过去的分布式文件系统有很多相同的目标，但 GFS 的设计受到了当前及预期的应用方面的工作量及技术环境的驱动，这反映了它与早期的文件系统明显不同的设想[5]：

1) 硬件出错是正常而非异常。因为文件系统由成百上千个用于存储的机器构成，而这些机器是由廉价的普通部件组成并被大量的客户机访问。部件的数量和质量使得一些机器随时都有可能无法工作并且有一部分还可能无法恢复。所以实时地监控、错误检测、容错、自动恢复对系统来说必不可少。

2) 需要存储大尺寸的文件。长度达几个 GB 的文件是很平常的。每个文件通常包含很多应用对象。当经常要处理快速增长的、包含数以万计的对象、长度达 TB 的数据集时，我们很难管理成千上万的 KB 规模的文件块，即使底层文件系统提供支持。因此，设计中操作的参数、块的大小必须要重新考虑。对大型的文件的管理一定要能做到高效，对小型的文件也必须支持，但不必优化。

3) 大部分文件的更新是通过添加新数据完成的，而不是改变已存在的数据。在一个文件中随机的操作在实践中几乎不存在。一旦写完，文件就只可读，很多数据都有这些特性。一些数据可能组成一个大仓库以供数据分析程序扫描。有些是运行中的程序连续产生的数据流。有些是档案性质的数据，有些是在某个机器上产生、在另外一个机器上处理的中间数据。由于这些对大型文件的访问方式，添加操作成为性能优化和原子性保证的焦点。而在客户机中缓存数据块则失去了吸引力。

4) 工作量主要由两种读操作构成：对大量数据的流方式的读操作和对少量数据的随机方式的读操作。在前一种读操作中，可能要读几百 KB，通常达 1MB 和更多。来自同一个客户的连续操作通常会读文件的一个连续的区域。随机的读操作通常在一个随机的偏移处读几个 KB。

5) 工作量还包含许多对大量数据进行的、连续的、向文件添加数据的写操作。所写的数据的规模和读相似。一旦写完，文件很少改动。在随机位置对少量数据的写操作也支持，但不必非常高效。

6) 系统必须高效地实现定义完好的大量客户同时向同一个文件的添加操作的语义。

3.1.3 GFS 的特点

1) 单 Master 模式[5]：只有一个 Master 极大的简化了设计并使得 master 可以根据全局情况作出先进的块放置和复制决定。但是我们必须要将 master 对读和写的参与减至最少，这样它才不会成为系统的瓶颈。 Client 只从 Master 读取文件块的元数据信息，然后知道了要和哪个 Chunk Server 联系，Client 在一段限定的时间内将这些信息缓存，在后续的操作中 Client 直接和 Chunk Server 交互。这样的设计降低了 Master 的压力，平衡了负载。

2) 块规模为 64MB[5]： 块规模是设计中的一个关键参数。64MB 的存储块比一般的文件系统的块规模要大的多。每个块的副本作为一个普通的 Linux 文件存储，在需要的时候可以扩展。 较大的块规模能够减少 Client 和 Master 之间的交互 ，使 Client 在一个给定的块上很可能执行多个操作，同时能够减少 Master 上保存的元数据（metadata）的规模 。

3) 不缓存文件数据，缓存元数据[5]：对于存储在 Chunk Server 上的文件数据，由于其本地文件系统能够提供缓存机制，而在 GFS 中实现缓存受制于 Chunk Server 不稳定所带来

的复杂的数据一致性问题，并没有实现缓存机制。对于存储在 Master 中的元数据，GFS 采取了缓存策略，GFS 中 Client 发起的所有操作都需要先经过 Master。Master 需要对其元数据进行频繁操作，为了提高操作的效率，Master 的元数据都是直接保存在内存中进行操作；同时采用相应的压缩机制降低元数据占用空间的大小，提高内存的利用率。

3.2 并行数据处理 MapReduce

MapReduce 是一个编程模型,是用来处理和产生大数据集的相关实现。用户指定一个 map 函数处理一个 key/value 对,从而产生中间的 key/value 对集.然后再指定一个 reduce 函数合并所有的具有相同中间 key 的中间 value 集合[6]。MapReduce 也是 Google 开发的一个并行计算框架，把分布式的业务逻辑从复杂的细节中抽象出来，提供了自动的并行化与分布式、容错、I/O 调度以及状态监控等功能，为并行开发经验不足的程序员进行并行编程提供了简单的接口。

3.2.1 MapReduce 的产生背景

Google 的使命是整合全球信息，使人人皆可访问并从中受益。所以 Google 比一般网站更早遭遇了只有分布才能存储的数据，这导致了 Google File System 的诞生。紧接着他们遇到的问题是怎么才能让公司所有的程序员都学会些分布计算的程序，因为他们用 Google File System 存储的海量数据分析起来需要的运算量也是惊人的。这就催生了 MapReduce 技术，通过把海量数据集的常见操作抽象为 Map 和 Reduce 两种集合操作，大大简化了程序员编写分布计算程序的难度。

与传统的分布式程序设计相比，MapReduce 封装了并行处理、容错处理、本地化计算、负载均衡等细节，还提供了一个简单而强大的接口。通过这个接口，可以把大尺度的计算自动地并发和分布执行，从而使编程变得非常容易。还可以通过由普通 PC 构成的巨大集群来达到极高的性能。另外，MapReduce 也具有较好的通用性，大量不同的问题都可以简单地通过 MapReduce 来解决。

MapReduce 把对数据集的大规模操作，分发给一个主节点管理下的各分节点共同完成，通过这种方式实现任务的可靠执行与容错机制。在每个时间周期，主节点都会对分节点的工作状态进行标记，一旦分节点状态标记为死亡状态，则这个节点的所有任务都将分配给其他分节点重新执行。通过使用 MapReduce 这种编程模式，保持了服务器之间的均衡，提高了整体效率。

3.2.2 MapReduce 的编程模型

MapReduce 模型中，输入和输出都是一个键值对 key/value 集合。用户自定义 Map 函数,接受一个输入对,然后产生一个中间 key/value 对集，所有的 Map 函数都可以高度的并行运行。随后 MapReduce 库把所有具有相同中间 key I 的中间 value 聚合在一起，然后把它们传递给 reduce 函数。用户自定义的 Reduce 函数，接受一个中间 key I 和相关的一个 value 集。它合并这些 value,形成一个比较小的 value 集。每个 Reduce 所处理的 Map 中间结果是

互不交叉的，所有 Reduce 产生的最终结果经过简单连接就形成了完整的结果集，因此 Reduce 也可以在并行环境下执行。以下用是 MapReduce 的执行过程[6]：

Programmer specifies two functions:

1.map (in_key, in_value) -> list(out_key, intermediate_value)

Processes input key/value pair

Produces set of intermediate pairs

2.reduce (out_key, list(intermediate_value)) -> list(out_value)

Combines all intermediate values for a particular key

Produces a set of merged output values (usually just one)

Map 的输入参数是 in_key 和 in_value，它指明了 Map 需要处理的原始数据是哪些。Map 的输出结果是一组<key,value>对，这是经过 Map 操作后所产生的中间结果。在进行 Reduce 操作之前，系统已经将所有 Map 产生的中间结果进行了归类处理，使得相同 key 对应的一系列 value 能够集结在一起提供给一个 Reduce 进行归并处理，也就是说，Reduce 的输入参数是（key, [value1,…,valuem]）。Reduce 的工作是需要对这些对应相同 key 的 value 值进行归并处理，最终形成（key, final_value）的结果。这样，一个 Reduce 处理了一个 key，所有 Reduce 的结果并在一起就是最终结果。

例如，假设我们想用 MapReduce 来计算一个大型文本文件中各个单词出现的次数，Map 的输入参数指明了需要处理哪部分数据，以<在文本中的起始位置，需要处理的数据长度>表示，经过 Map 处理，形成一批中间结果<单词，出现次数>。而 Reduce 函数则是把中间结果进行处理，将相同单词出现的次数进行累加，得到每个单词总的出现次数。

3.3 分布式数据库 BigTable

3.3.1 BigTable 概述

Bigtable 是 Google 开发的基于 GFS、MapReduce 和 Chubby 的分布式存储数据库系统，它被设计用来处理海量数据：通常是分布在数千台普通服务器上的 PB 级的数据，并且能够部署到上千台机器上。Bigtable 和数据库很类似：它使用了很多数据库的实现策略，但是它有不是一个完全的关系型数据库，它不支持完整的关系数据模型，而是提供了一个简单的数据模型接口，使得数据的存储更加灵活。 Google 的很多数据，包括 Web 索引、卫星图像数据等在内的海量结构化和半结构化数据，都是存储在 Bigtable 中的。Bigtable 已经实现了下面的几个目标：适用性广泛、可扩展、高性能和高可用性[7]。Bigtable 已经在超过 60 个 Google 的产品和项目上得到了应用，包括 Google Analytics、Google Finance、Orkut、Personalized Search、Writely 和 Google Earth。这些产品对 Bigtable 提出了迥异的需求，有的需要高吞吐量的批处理，有的则需要及时响应，快速返回数据给最终用户。它们使用的 Bigtable 集群的配置也有很大的差异，有的集群只有几台服务器，而有的则需要上千台服务器、存储几百 TB 的数据。

3.3.2 BigTable 体系结构

Bigtable 是建立在其它的几个 Google 基础构件上的。BigTable 集群通常运行在一个共享的机器池中，池中的机器还会运行其它的各种各样的分布式应用程序，BigTable 的进程经常要和其它应用的进程共享机器。

BigTable 使用 Google 的分布式文件系统(GFS)存储日志文件和数据文件。WorkQueue 是一个分布式的任务调度器，它主要被用来处理分布式系统队列分组和任务调度。Bigtable 的实际执行过程中，Google 的 MapReduce 也被使用来改善其性能，不过需要注意的是这不是实现 Bigtable 所必需的[7]。另外，BigTable 还依赖一个高可用的、序列化的分布式锁服务组件—Chubby。Chubby 主要有以下几个作用[8]：

1).选取并保证同一时间内只有一个主服务器（Master Server）。

2).获取子表的位置信息。

3).保存 Bigtable 的模式信息及访问控制列表。

Bigtable 包括了三个主要的组件：链接到客户程序中的库、一个主服务器和多个子表服务器。针对系统工作负载的变化情况，BigTable 可以动态的向集群中添加（或者删除）子表服务器，其基本结构如图 3-1 所示[7]。

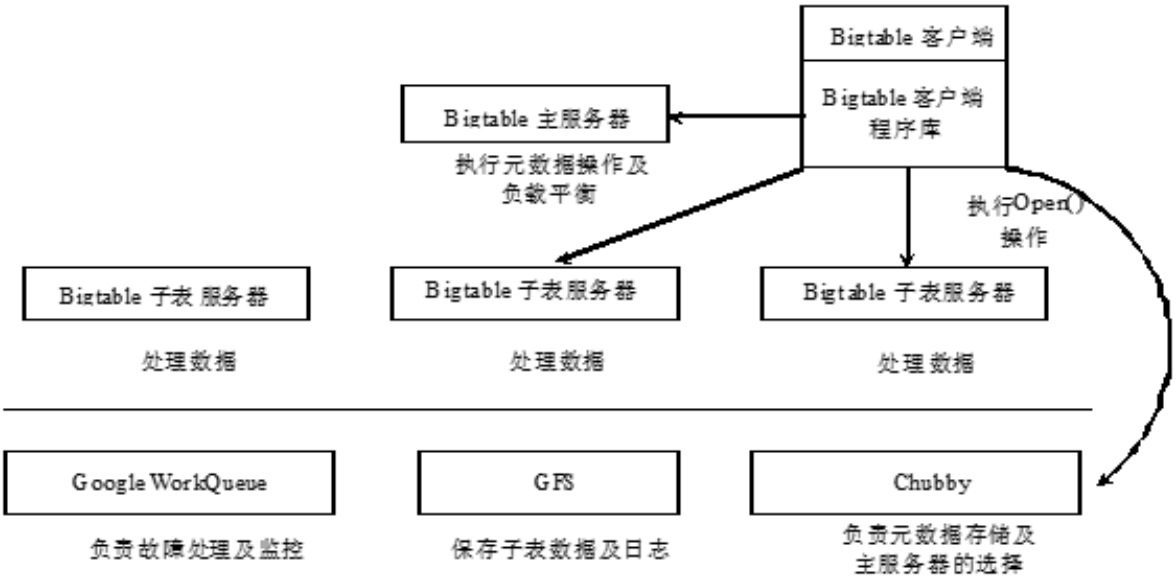

图 3-1 Bigtable 体系结构

一个 BigTable 集群存储了很多表，每个表包含了一个子表的集合，而每个子表包含了某个范围内的行的所有相关数据。Bigtable 中存储数据都是以子表的形式保存在子表服务器上的，客户一般也只和 Tablet 服务器进行通信。

主服务器主要负责以下工作：为子表服务器分配子表、检测新加入的或者过期失效的子表服务器、对子表服务器进行负载均衡、以及对保存在 GFS 上的文件进行垃圾收集。除此之外，它还处理对模式的相关修改操作，例如建立表和列族。Bigtable 中主服务器对子表服务器的监控是通过 Chubby 来完成的，子表服务器在初始化时都会从 Chubby 中得到一个独占锁。通过这种方式所有的子表服务器基本信息被保存在 Chubby 中一个称为服务器

目录（Server Directory）的特殊目录之中。主服务器通过检测这个目录就可以随时获取最新的子表服务器信息，包括目前活跃的子表服务器，以及每个子表服务器上现已分配的子表。

每个子表服务器都管理一个子表的集合（通常每个服务器有大约数十个至上千个子表）。每个子表服务器负责处理它所加载的子表的定位、分配和读写操作，以及在子表过大时对其进行分割。子表实际上存储的是包含了某个范围内的行的所有相关数据，而子表又是以 SStable 作为基本的存储单元，由多个 SStable 共同组成的，SSTable 是 Google 为 Bigtable 设计的内部数据存储格式。所有的 SSTable 文件都是存储在 GFS 上的，用户可以通过键来查询相应的值。一个子表服务器上所有子表都共享一个日志文件。

由于 BigTable 中的数据都是以子表的形式存储在各个子表服务器上的，所以子表的寻址成为了 BigTable 系统的关键问题。Google 使用了一个三层的、类似 B +树[10]的结构存储 Tablet 的位置信息。如图 3-2 所示：

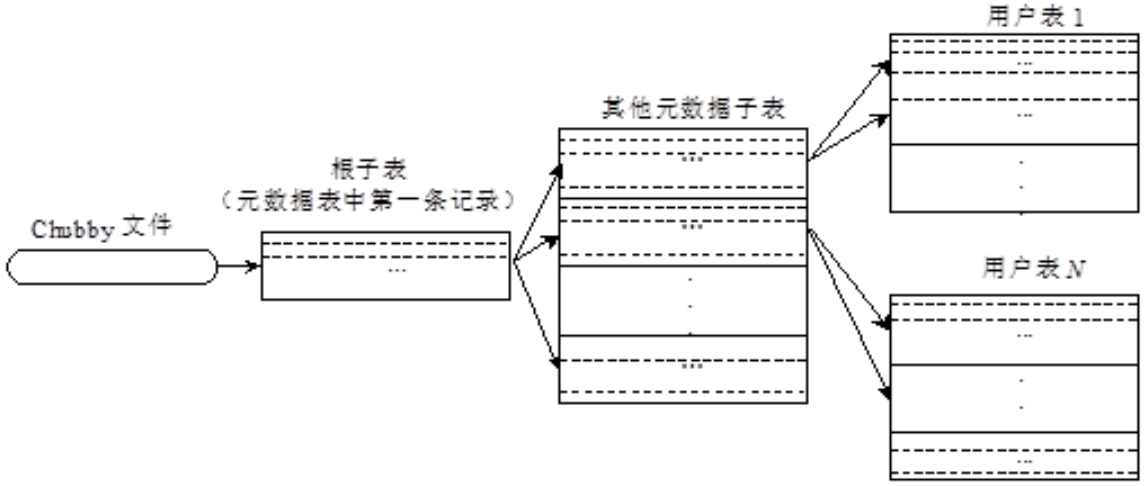

图 3-2 子表地址结构

所有的子表地址都被记录在元数据表中，元数据表也是由一个个的元数据子表（Metadata tablet）组成的。第一层是存储在 Chubby 中的根子表，它是元数据表中的第一个元数据子表，存储了其他元数据子表的地址；第二层是根子表指向的其他元数据子表，表中每个子表的位置信息都存放在一个行关键字下面，而这个行关键字是由子表所在的表的标识符和子表的最后一行编码而成的。第三层便是所有用户子表。

为了减少访问开销，提高客户访问效率，Bigtable 使用了缓存（Cache）和预取（Prefetch）技术，这两种技术手段在体系结构设计中是很常用的。子表的地址信息被缓存在客户端，客户在寻址时直接根据缓存信息进行查找。一旦出现缓存为空或缓存信息过时的情况，客户端就需要按照图 3-5 所示方式进行网络的来回通信（Network Round-trips）进行寻址，在缓存为空的情况下需要三个网络来回通信。如果缓存的信息是过时的，则需要六个网络来回通信。其中三个用来确定信息是过时的，另外三个获取新的地址。预取则是在每次访问元数据表时不仅仅读取所需的子表元数据，而是读取多个子表的元数据，这样下次需要时就不用再次访问元数据表。

4 Google App Engine 测试程序分析与设计

4.1 测试程序需求分析

为了测试 Google 云计算平台提供的各项服务的使用以及性能，我编写了一个简单的博客系统。如图 4-1 的系统功能结构图所示，本系统主要由三个模块构成：

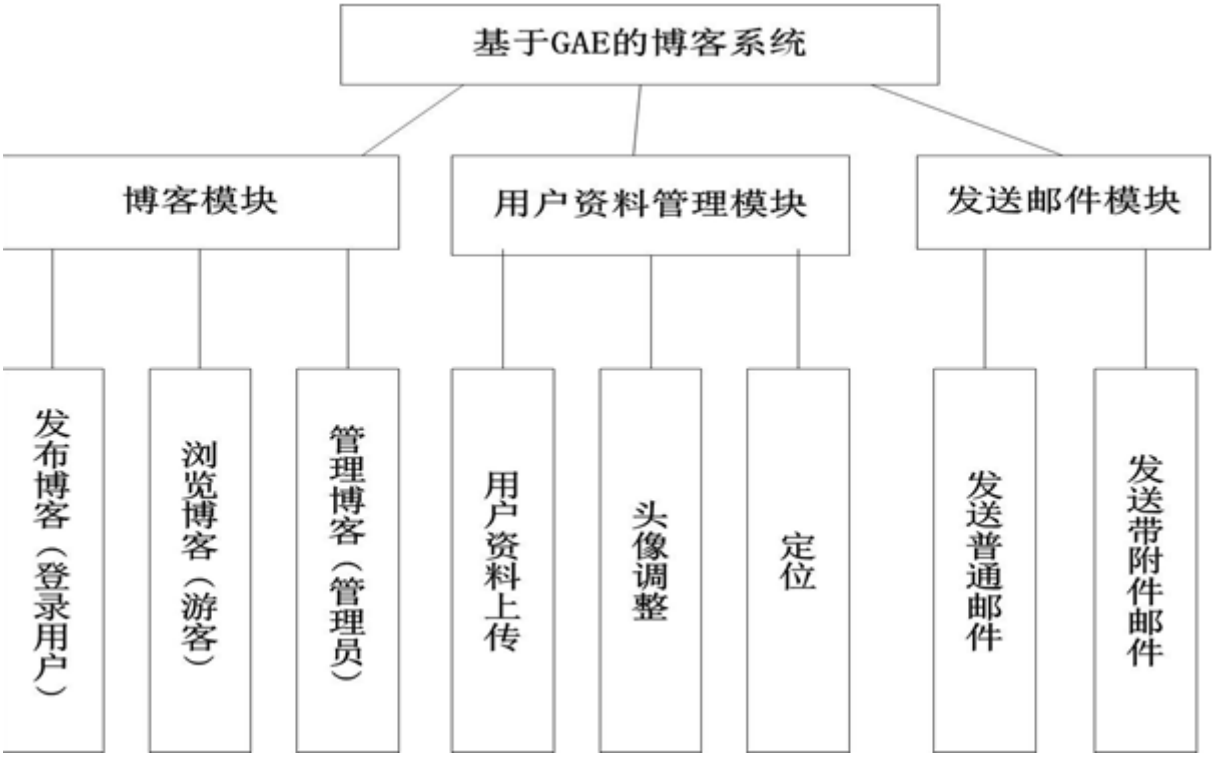

图 4-1 系统功能结构图

博客的发布和浏览模块。该模块测试了 Google 的数据存储服务，使用了 JPA 来完成对博客的增加和查询。

用户的登录和资料管理模块：该模块测试 Google 的账户服务、用于大文件的 BlobStore 服务、图片处理服务和 Google Map 服务。实现了用户资料增加和删除、头像的调整以及用户的定位。

电子邮件发送模块：该模块测试了 Google 的电子邮件服务，实现了能够通过登录用户的电子邮件地址发送邮件的功能。

本系统还实现了能够将博客的查询时间统计到日志的功能，能够简单的分析一下 GAE 存储服务的性能。此外，程序运行过程中产生的错误也会被记录到日志中。

4.2 测试程序运行环境分析

4.2.1 Google App Engine 概述

App Engine 是在 Google 的基础架构上构建和运行网络应用程序的平台，提供了使

用 Google 基础服务的编程结构。App Engine 应用程序易于构建和维护，开始使用 App Engine 是免费的。所有应用程序都可以使用 500 MB 的存储空间，以及可支持每月约 500 万页面浏览量的足够的 CPU 和带宽，并可根据您的访问量和数据存储需要的增长轻松扩展。为您的应用程序启用付费后，您的免费配额将提高，您只需为使用的超过免费水平的资源付费[15]。 App Engine 平台实现了资源的按需分配，App Engine 平台会监控资源的使用情况，根据资源的使用量来收取费用。

4.2.2 Google 软件基础设施分析

App Engine 构建在 Google 云计算平台基础设施之上，这个基础设施由硬件和软件组成。硬件就是 Google 全球的数据中心里上百万台的 X86 服务器和连接它们的网络系统，软件是 Google 云计算平台的核心，用来管理这庞大的硬件设施提供的计算能力和存储能力，使它们成为一个整体来提供强大的计算和海量的存储。

4.2.2.1 文件系统分析

文件系统是一个操作系统和平台必不可少的组件，是用来存储数据的工具，Google 云计算平台的文件系统是 GFS，与一般操作系统的本地文件系统（Local File System）将物理存储资源直接连接在本地节点上的方式不同，GFS 管理的物理存储资源不一定直接连接在本地节点上，而是通过计算机网络与节点相连。分布式文件系统的是目前最高级的文件系统，它将由网络连接的各个存储节点抽象成为一个统一的存储系统，内部各个存储节点的管理和协作等复杂问题均由系统实现，提供了与最基本的本地文件系统几乎相同的访问接口和对象模型，大大简化了分布式文件系统的使用。GFS 是一个分布式文件系统，但是与已有的其他分布式文件系统采用高性能磁盘阵列来实现节点容错不同[16]，GFS 完全采用软件的方法来实现容错，这大大节省了存储的成本，但是在功能上完全可以满足需求。下面就来介绍一下 GFS 的系统结构，了解数据在 Google 云计算平台中的存储方式。

GFS 的系统架构如图 4-2 所示[5]。GFS 将整个系统的节点分为三类角色：Client（客户端）、Master（主服务器）和 Chunk Server（数据块服务器）。Master 和 chunkserver 通常是运行用户层服务进程的 Linux 机器。 Chunk Server 将块（块默认大小为 64M）当作 Linux 文件存储在本地磁盘并可以读和写由 chunk－handle 和位区间指定的数据。出于可靠性考虑，每一个块被复制到多个 Chunk Server 上。默认情况下，保存 3 个副本，但这可以由用户指定。Master 维护文件系统所以的元数据（metadata），包括名字空间、访问控制信息、从文件到块的映射以及块的当前位置。它也控制系统范围的活动，如块租约（lease）管理，孤儿块的垃圾收集，Chunk Server 间的块迁移。Master 定期通过心跳消息与每一个 Chunk Server 通信，给 Chunk Server 传递指令并收集它的状态。 Client 是 GFS 提供给应用程序的访问接口，它是一组专用接口，不遵守 POSIX 规范，以库文件的形式提供。Client 与 master 和 Chunk Server 通信以代表应用程序读和写数据。客户与 Master 的交换只限于对元数据（metadata）的操作，所有数据方面的通信都直接和 chunkserver 联系。

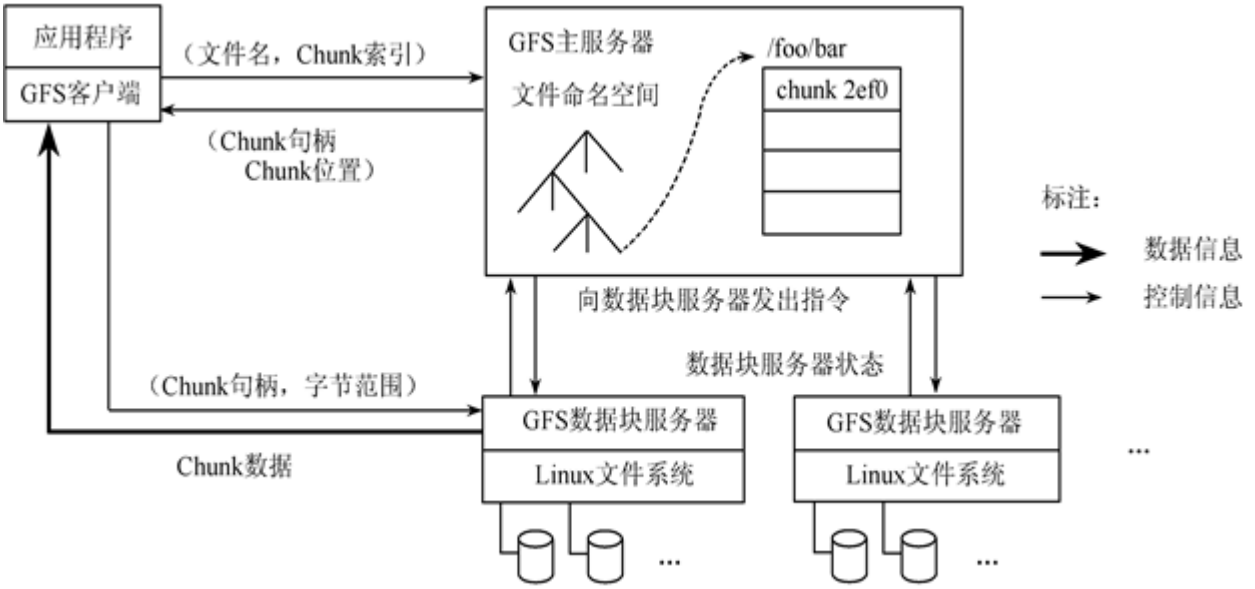

图 4-2　GFS 体系结构

客户端在访问 GFS 时，首先访问 Master 节点，获取将要与之进行交互的 Chunk Server 信息，然后直接访问这些 Chunk Server 完成数据存取。GFS 的这种设计方法实现了控制流和数据流的分离。Client 与 Master 之间只有控制流，而无数据流，这样就极大地降低了 Master 的负载，使之不成为系统性能的一个瓶颈。Client 与 Chunk Server 之间直接传输数据流，同时由于文件被分成多个 Chunk 进行分布式存储，Client 可以同时访问多个 Chunk Server，从而使得整个系统 I/O 高度并行，系统整体性能得到提高。

4.2.2.2 计算模型分析

单机操作系统的计算是以线程为单位进行的，多个线程轮流的占用 cpu 时间来处理数据，在同一个进程中的线程可以共享数据。而 Google 云计算平台需要利用的是大量的服务器，其提出的 MapReduce 计算模型可以让多个单机操作系统协同计算，将并行化程度从线程提升到了操作系统，一个计算是通过多个操作系统的多个线程来完成的，其计算能力得到了显著的提升。同时，MapReduce 也需要 GFS 的配合来存储输入数据、中间数据和输出数据。下面就来介绍一下 MapReduce 在 Google 云计算平台中的运行过程。

Google 内部广泛使用的运算环境的实现：用以太网交换机连接、由普通 PC 机组成的大型集群。在我们的环境里包括：

1) x86 架构、运行 Linux 操作系统、双处理器、2-4GB 内存的机器。

2) 普通的网络硬件设备，每个机器的带宽为百兆或者千兆，但是远小于网络的平均带宽的一半。

3) 集群中包含成百上千的机器，因此，机器故障是常态。

4) 存储为廉价的内置 IDE 硬盘。一个内部分布式文件系统用来管理存储在这些磁盘上的数据。文件系统通过数据复制来在不可靠的硬件上保证数据的可靠性和有效性。

5) 用户提交工作（job）给调度系统。每个工作（job）都包含一系列的任务（task），调度系统将这些任务调度到集群中多台可用的机器上。

执行过程：通过将 Map 调用的输入数据自动分割为 M 个数据片段的集合，Map 调用被分布到多台机器上执行。输入的数据片段能够在不同的机器上并行处理。使用分区函数将 Map 调用产生的中间 key 值分成 R 个不同分区（例如，hash(key) mod R），Reduce 调用也被分布到多台机器上执行。分区数量（R）和分区函数由用户来指定。

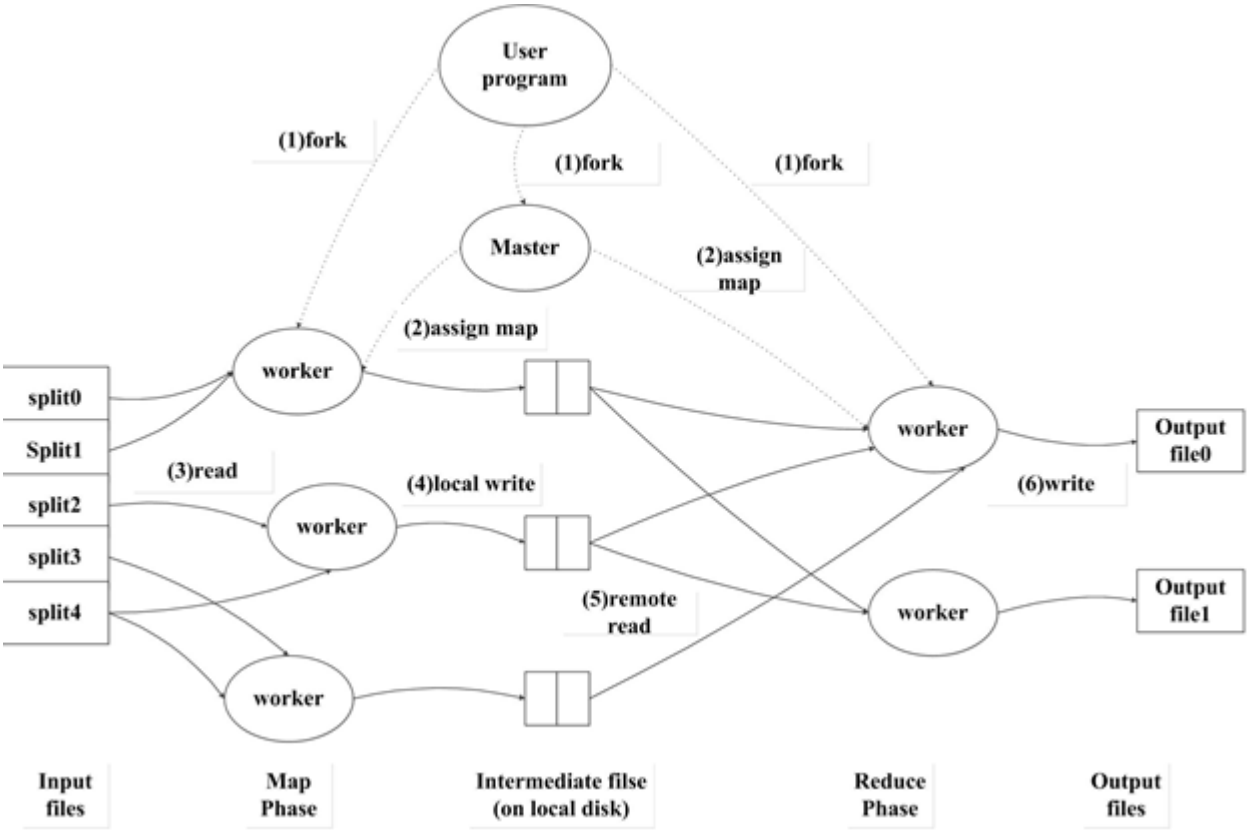

图 4-3 MapReduce 计算执行过程

图 4-3 展示了我们的 MapReduce 实现中操作的全部流程[6]。当用户调用 MapReduce 函数时，将发生下面的一系列动作（下面的序号和图中的序号一一对应）：

1) 用户程序首先调用的 MapReduce 库将输入文件分成 M 个数据片度，每个数据片段的大小一般从 16MB 到 64MB(可以通过可选的参数来控制每个数据片段的大小)。然后用户程序在集群中创建大量的程序副本。

2) 这些程序副本中的有一个特殊的程序－Master。副本中其它的程序都是 Worker 程序，由 Master 分配任务。有 M 个 Map 任务和 R 个 Reduce 任务将被分配，Master 将一个 Map 任务或 Reduce 任务分配给一个空闲的 Worker。

3) 被分配了 map 任务的 worker 程序读取相关的输入数据片段，从输入的数据片段中解析出 key/value pair，然后把 key/value pair 传递给用户自定义的 Map 函数，由 Map 函数生成并输出的中间 key/value pair，并缓存在内存中。

4) 缓存中的 key/value pair 通过分区函数分成 R 个区域，之后周期性的写入到本地磁盘上。缓存的 key/value pair 在本地磁盘上的存储位置将被回传给 Master，由 Master 负责把这些存储位置再传送给 Reduce Worker。

5) 当 Reduce Worker 程序接收到 Master 程序发来的数据存储位置信息后，使用 RPC

从 Map Worker 所在主机的磁盘上读取这些缓存数据。当 Reduce Worker 读取了所有的中间数据后，通过对 key 进行排序后使得具有相同 key 值的数据聚合在一起。由于许多不同的 key 值会映射到相同的 Reduce 任务上，因此必须进行排序。如果中间数据太大无法在内存中完成排序，那么就要在外部进行排序。

6) Reduce Worker 程序遍历排序后的中间数据，对于每一个唯一的中间 key 值，Reduce Worker 程序将这个 key 值和它相关的中间 value 值的集合传递给用户自定义的 Reduce 函数。Reduce 函数的输出被追加到所属分区的输出文件。

7) 当所有的 Map 和 Reduce 任务都完成之后，Master 唤醒用户程序。在这个时候，在用户程序里的对 MapReduce 调用才返回。

在成功完成任务之后，MapReduce 的输出存放在 R 个输出文件中（对应每个 Reduce 任务产生一个输出文件，文件名由用户指定）。一般情况下，用户不需要将这 R 个输出文件合并成一个文件－他们经常把这些文件作为另外一个 MapReduce 的输入，或者在另外一个可以处理多个分割文件的分布式应用中使用。

GFS 和 MapReduce 配合使用，再加上 Chubby 锁服务、GWQ 消息队列等技术的辅助，Google 云计算平台才能够实现海量数据的存储和处理，为上层应用程序提供有力的支撑。

4.2.3 App Engine 应用程序环境和沙盒

通过 Google App Engine，即使在负载很重和数据量极大的情况下，也可以轻松构建能安全运行的应用程序。App Engine 包括以下功能：

1).动态网络服务，提供对常用网络技术的完全支持。

2).持久存储空间，支持查询、分类和事务。

3).自动扩展和负载平衡。

4).用于对用户进行身份验证和使用 Google 帐户发送电子邮件的 API。

5).一种功能完整的本地开发环境，可以在您的计算机上模拟 Google App Engine。

6).用于在指定时间和定期触发事件的计划任务。

本次测试以应用程序使用 java 语言编写，下面介绍一下 java 语言的运行时环境和沙盒限制。App Engine 使用 Java 6 虚拟机 (JVM) 来运行 Java 应用程序。App Engine SDK 支持 Java 5 及更高版本，该环境包括 Java SE 运行时环境 (JRE) 6 平台和库。出于服务和安全原因，JVM 在安全的“沙盒”环境中运行以隔离不同的应用程序。沙盒确保了应用程序仅执行不影响其他应用程序的性能和可伸缩性的操作。沙盒限制了应用程序不得进行一下操作：1.向文件系统写入；2.打开套接字或直接访问另一主机；3.产生子进程或线程。可以在 JVM 使用的更多类可以参考 App Engine JRE 白名单。

App Engine 对网络应用程序使用 Java Servlet 标准。您以标准 WAR 目录结构提供应用程序的 servlet 类、JavaServer Pages (JSP)、静态文件和数据文件以及部署描述符（web.xml 文件）和其他配置文件。我们也可以在应用程序中使用 Java Web 编程中的常用框架，但是像 Hibernate 这样面向关系型数据库的持久层框架不可以使用。其他框架中在沙

沙盒限制范围的功能将不能使用。

4.3 App Engine 测试程序设计

4.3.1 测试程序概要设计

本系统采用了分层的架构来设计，整个系统可以分为模型层、服务层和 web 展示层。分层的架构能够降低系统各个组件之间的耦合度，提高了程序的可读性和可测试性，图 4-4 展示了本系统的架构设计。

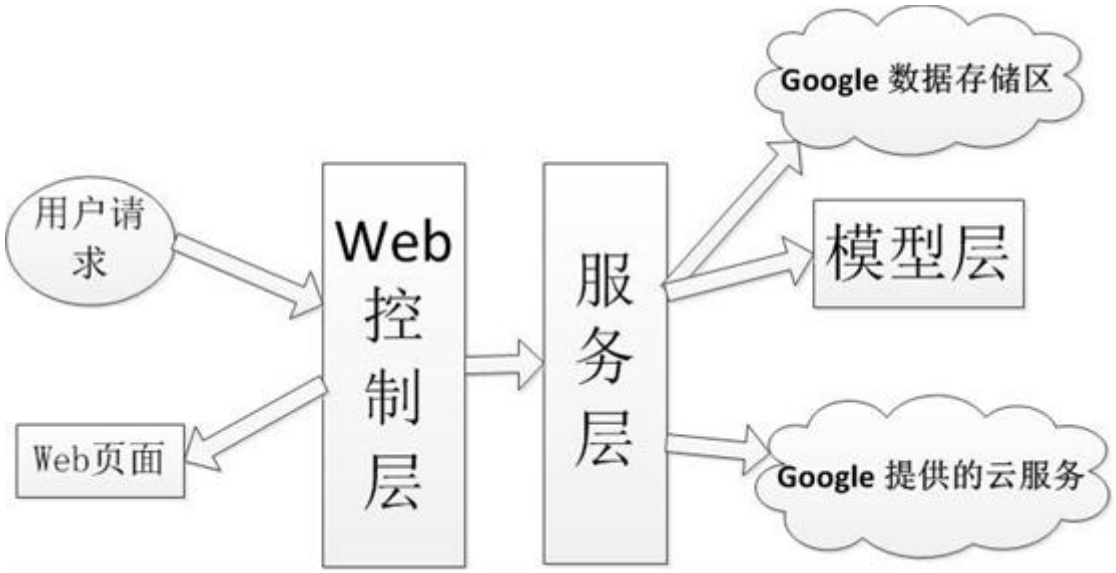

图 4-4 系统架构设计图

4.3.3.1 模型层设计

模型层是由可持久化到 Google 数据存储区的实体类组成。这些实体类最终存储在 Google 的 BigTable 中，BigTable 是构建在 GFS 和 MapReduce 上的一个多维分布式数据库，与关系模型相比，BigTable 的数据模型维度更多，能够存储的信息更多，组织方式更加灵活。Bigtable 是一个多维度排序 Map。Map 的索引是行关键字、列关键字以及时间戳；Map 中的每个 value 都是一个未经解析的 byte 数组[7]。与关系型数据库采用二维模型不同，BigTable 是具有行关键字、列族、列关键字和时间戳的多维模型，提高了存储的灵活性，同时能够存储数据的多个版本。

行：表中的行关键字可以是任意的字符串。对同一个行关键字的读或者写操作都是原子的（不管读或者写这一行里多少个不同列），这个设计决策能够使用户很容易的理解程序在对同一个行进行并发更新操作时的行为。Bigtable 通过行关键字的字典顺序来组织数据。表中的每个行都可以动态分区。每个分区叫做一个”Tablet”，Tablet 是数据分布和负载均衡调整的最小单位。这样做的结果是，当操作只读取行中很少几列的数据时效率很高，通常只需要很少几次机器间的通信即可完成。用户可以通过选择合适的行关键字，在数据访问时有效利用数据的位置相关性，从而更好的利用这个特性。

列族：列关键字组成的集合叫做“列族“，列族是访问控制的基本单位。存放在同一

列族下的所有数据通常都属于同一个类型（我们可以把同一个列族下的数据压缩在一起）。列族在使用之前必须先创建，然后才能在列族中任何的列关键字下存放数据；列族创建后，其中的任何一个列关键字下都可以存放数据。根据我们的设计意图，一张表中的列族不能太多（最多几百个），并且列族在运行期间很少改变。与之相对应的，一张表可以有无限多个列。访问控制、磁盘和内存的使用统计都是在列族层面进行的。

时间戳：在 Bigtable 中，表的每一个数据项都可以包含同一份数据的不同版本；不同版本的数据通过时间戳来索引。Bigtable 时间戳的类型是 64 位整型。Bigtable 可以给时间戳赋值，用来表示精确到毫秒的“实时”时间；用户程序也可以给时间戳赋值。如果应用程序需要避免数据版本冲突，那么它必须自己生成具有唯一性的时间戳。数据项中，不同版本的数据按照时间戳倒序排序，即最新的数据排在最前面。

Google 数据存储区以 BigTable 作为基础，封装了 BigTable 与关系型数据库的区别，为用户提供了一个类似关系数据库的简单接口来使用 Google 的 BigTable，但是在实体类的主键、实体类间关系的配置问题上还是以 Google 的数据模型为依据来考虑。

4.3.3.2 服务层设计

服务层依赖于 GAE 提供的各种服务，依据设计模式中的门面模式，根据服务的类别设计了适合本测试程序使用的服务类，例如博客服务类使用 JPA 接口操作 Google 数据存储区，具有增删改查博客的功能，图像服务类使用 Google 的图像处理服务 ImagesService，具有调整、翻转、旋转、裁剪图像的功能。

4.3.3.3 web 控制层设计

Web 控制层是由多个 Servlet 构成，根据请求中携带的控制参数判断执行流程，调用服务层的服务类进行数据处理，根据数据处理结果返回相应页面。web 控制层是用户界面和服务层的桥梁，是测试程序的控制器。

4.3.2 测试程序详细设计

概要设计在横向上对测试程序的层次有了明确的划分，并且确定了各个层次间的依赖关系，下面在纵向上以模块为单位进行类的详细设计。模块是跨越了应用程序的各个层次，组合了各个层次的相关类从而能提供独立功能的组件，根据需求分析的结果，测试程序分为三个模块，一下就是这三个模块的详细类设计。

4.3.2.1 博客的发布和浏览模块

该模块主要包括：存储博客信息的 Blog 实体类，它存储了 Google 提供的 User 对象来与用户建立无主的关系；用户获取 Google 数据存储区连接的 EMF 工具类；对博客实体类进行增删改查的 BlogService 类；控制博客处理流程的 BlogServlet 类。

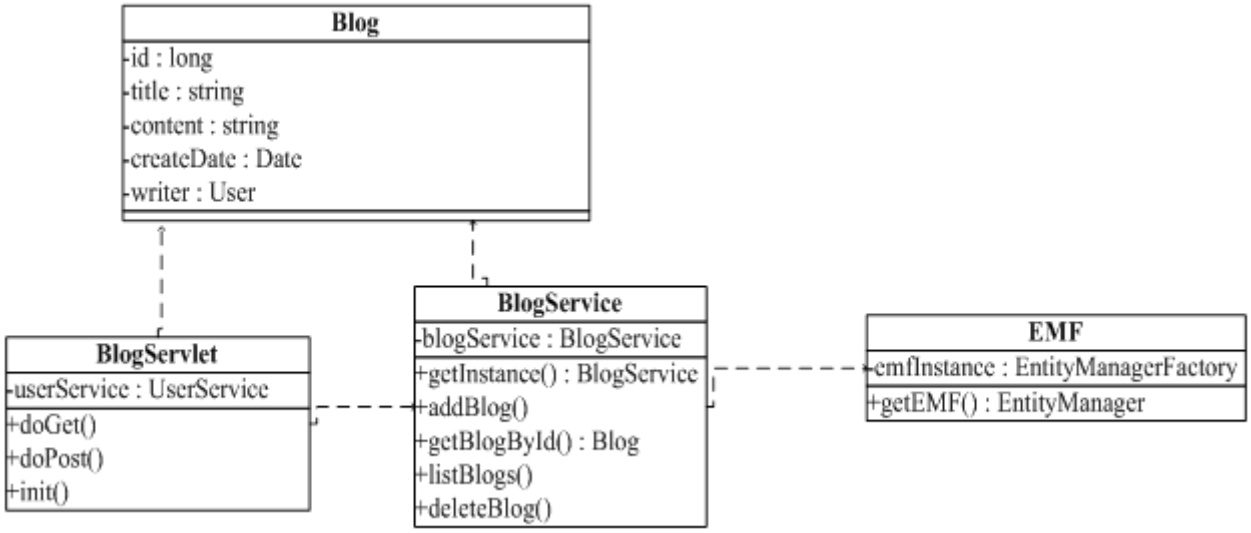

图 4-5 博客模块类图

4.3.2.2 用户资料管理模块

该模块包括：用来存储用户资料的实体类 AccountInfo，它以用户的邮件地址作为主键，依次与用户建立关联；对用户资料实体进行增删改查的 AccountService 类；控制用户资料处理流程的 AccountServlet 类，它使用了 Google 提供的 blobStore 服务来完成头像的上传；操作用户头像变化的 ImageService 类；控制用户的头像显示和头像转变流程的 logoServlet 类。

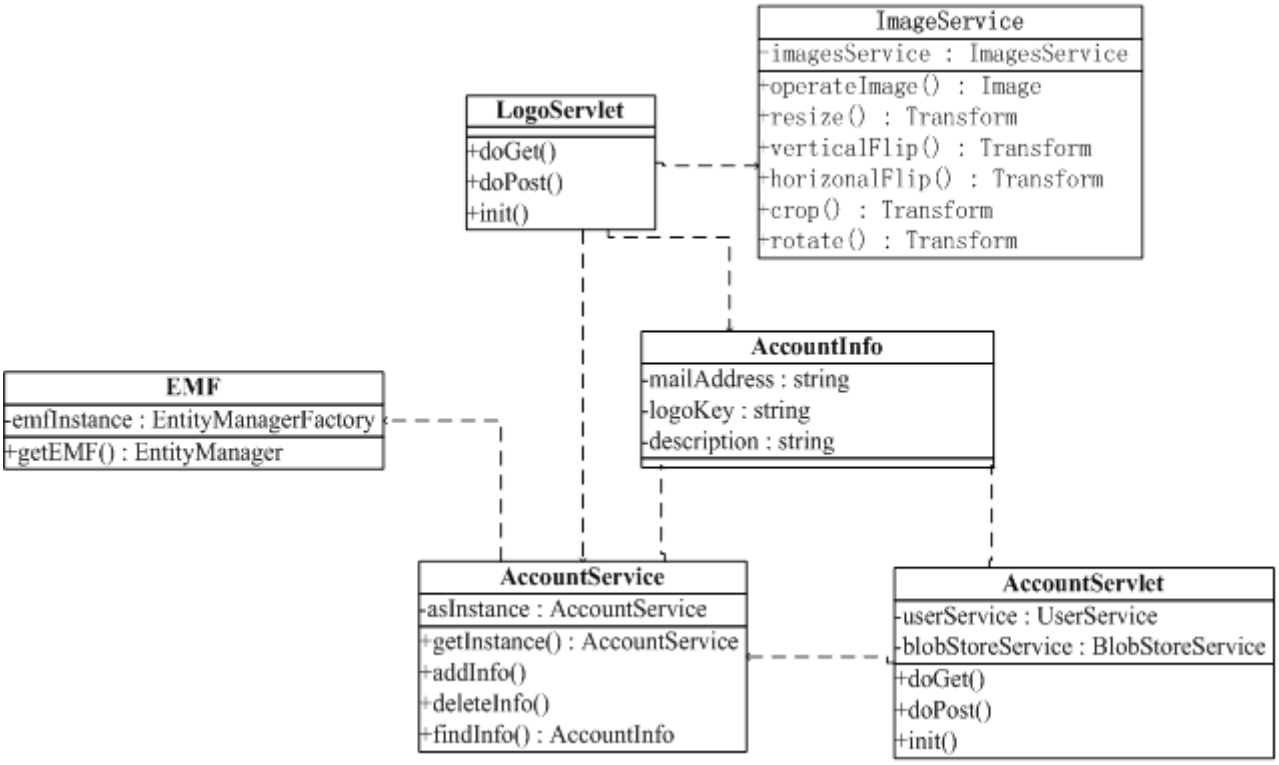

图 4-6 用户资料管理模块类图

4.3.2.3 邮件模块

该模块包括：能够发送普通文本邮件和多部分邮件的 MyMailService 类；处理发送邮件表单并返回相应结果的 MailServlet 类。

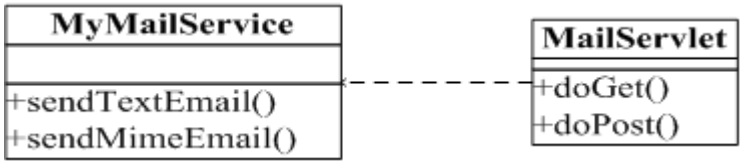

图 4-7 邮件发送模块类图

对测试程序进行概要设计和详细设计之后，对测试程序的结构和各模块的组成有了总体的认识，接下来的工作就是参照测试程序的设计来实现功能。

5 Google App Engine 测试程序实现

5.1 开发环境

Google 提供了 App Engine Sdk 开发包，能够在本地计算机上模拟 GAE 提供的各项服务，其中包括：用于调试程序的一个内嵌的 Jetty 服务器，用于上传应用程序到 GAE 的工具，本地开发依赖的 jar 包及其 doc 文档，还有一些实例供参考。测试程序的开发和调试是在 Eclipse 集成环境中完成，通过使用 Google 的 Eclipse 插件，可以方便的建立、调试以及上传 GAE 项目。

5.2 应用程序配置

App Engine 使用若干配置文件来控制网络服务器运行应用程序的方式、应用程序的上传方式、构建数据存储区索引的方式以及执行计划任务的方式。

Java 网络应用程序使用部署描述符文件来配置应用程序的行为。此文件名为 web.xml，且位于应用程序的 WAR 中的 WEB-INF/ 目录中。web.xml 是网络应用程序的 servlet 标准的一部分。在测试程序中，配置的内容为：Servlet 和 url 的映射，字符编码过滤器和需要过滤 url 的映射，网址的安装约束配置（既配置只有登录用户能够访问的页面）。

App Engine Java 应用程序使用名为 appengine-web.xml 的配置文件来指定应用程序的注册应用程序 ID 以及最新代码的版本标识符，并确定在应用程序的 WAR 中哪些文件是静态文件，哪些是应用程序使用的资源文件。当您上传应用程序时，AppCfg 命令将使用这些信息。测试程序中配置了应用程序的 ID 以及最新代码版本。

App Engine 数据存储区对您的应用程序进行的每个查询都将使用索引。每当实体更改时，这些索引将更新，以便在应用程序进行查询时快速返回结果。为此，数据存储区必须预先了解应用程序将进行哪些查询。可在配置文件中指定应用程序所需的查询。测试程序中没有配置存储区索引，所以开发服务器会使用自动索引配置。

5.3 博客的发布和浏览模块实现

该模块的核心功能就是博客的新增和查询，使用的是 GAE 的数据存储服务。GAE 数据存储区是一个无架构对象数据存储区，具有查询引擎和原子事务。该存储区是出于网络应用程序的考虑而设计的，着重于读取和查询性能。它存储带属性的数据实体，通过由应用程序定义的类型进行组织。通过对属性值和键设置过滤条件和排序顺序，该数据存储区可以对同一类型的实体执行查询。所有的查询都预先编入索引，便于在处理很大的数据集时快速获得结果。该数据存储区支持事务更新，使用由应用程序定义的实体分组作为分布式数据网络中事务性的单位。

该模块的主要工作是用来存储博客的实体类的及跟它相关的实体类的设计以及通过 JPA 接口来操作实体类。Google 官方给出的是使用 JDO 配置的实例，采用了简单的 java 标注技术将一个普通的 java 类配置为一个可以被持久化到 GAE 存储区的实体类。JPA 下

的配置与 JDO 基本原理相同，在实体类的设计有如下几个关键性问题。

5.3.1 实体健的配置

每个实体都具有一个在 App Engine 的所有实体中唯一的键，一个完整的键包含若干条信息，其中包括应用程序 ID、类型和实体 ID（键也包含有关实体组的信息）。对象的键存储在实例的某一个字段中，GAE 存取区提供了 4 种配置主键字段的方法：

1) 由数据存储区自动生成的长整形实体 ID。对于没有父实体组、其 ID 应由数据存储区自动生成的对象，使用此类型合适。实例的长整型键字段在该实例保存时填充。

```
@Id
private Long id
```

2) 对象创建时由应用程序提供的字符串类型实体 ID（“键名”）。对于没有父实体组、其 ID 应由应用程序提供的对象，使用此类型合适。应用程序在保存前将此域设置为所需的 ID。

```
@Id
Private String id
```

3) Key 实例 (com.google.appengine.api.datastore.Key)。键值包括父实体组（如果有）的键以及应用程序分配的字符串 ID 或系统生成的数字 ID。要创建带应用程序分配的字符串 ID 的对象，请创建带有该 ID 的键值并将字段设置为该值。要创建带系统分配的数字 ID 的对象，请将键字段留为 null。

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
 private Key key;
实体设置主键字段
Key key=KeyFactory.createKey(Object.class.getSimplename,"id");
Object o=new Object();
o.setKey(key);
```

4) 编码字符串形式的键。与键类似，但值是键的编码字符串形式。编码字符串键允许您以便携方式编写应用程序且仍可以利用 App Engine 数据存储区实体组。

由于考虑到使用自动生成的长整形主键能够避免手工维护主键的复杂并且使用更加简单，博客实体类采用了第一种主键策略。

5.3.2 实体间关系

使用对象类型的字段，可以在持久对象之间建模关系。JPA 接口的 App Engine 实现可以建模有主的一对一关系和有主的一对多关系，既一个对象无法脱离另一个而存在，这些关系既单向又双向。有主关系的两个实体会被放在同一个实体组中，处于同一个实体组中的实体在持久化，更新和删除时会产生级联。例如当拥有有主关系的对象保存到数据存

储区中时，会自动保存能够沿关系到达且需要保存的所有其他对象（都为新建对象的时候需要使用事务）。例如删除父对象，则也会删除所有子对象。而且通过对父对象的从属字段赋新的值来打破有主关系也会删除旧的子对象。

无主的关系（两个对象都可存在，而不管其彼此的关系）尚未为自然语法所支持，但可以通过直接在字段中存储数据存储区键来自己管理这些关系 。本博客系统中的用户资料实体和博客实体具有一对多的关系，用户资料实体存储的是使用 Google 账户登录本系统的用户上传的资料，使用登录用户的邮件地址作为主键。博客实体需要有作者信息，而博客可以独立于用户资料而存在，只要存储了 Google 账户的邮件即可找到作者信息。所以通过用户的邮件地址即可建立两个实体的无主关系。

5.3.3 事务

事务是数据库中最重要的问题，事务是一项或一系列数据存储区操作，这些操作要么全部成功，要么全部失败。如果事务成功完成，则会对数据存储区产生所有预期的作用。如果事务失败，则不会起任何作用。 要想在一个事务中控制多个实体，必须保证这些实体在同一个组中。当应用程序创建一个实体时，它将另一个实体分配为新实体的父实体。向新实体分配父实体会将新实体放置在与父实体相同的实体组。在博客模块中使用的都是对单个实体的操作，所以没有涉及到事务的使用。

在实体类设计完毕之后，接下来就是编写 Service 类实现对实体的增删改查操作。通过使用 JPA 接口，可以以对象的方式来管理实体，操作十分方便。Web 层使用了 Java 的 Servlet 和 JSP 技术，以 Servlet 作为控制器控制请求和响应的流程，以 JSP 页面向作者展示请求结果。

5.4 用户的登录和资料管理模块实现

该模块在使用 GAE 数据存储区的基础上，还要使用 Google 账户服务验证用户登录；使用 BlobStore 服务存储用户头像；使用 Google 的图像服务来处理用户头像；使用 Google Map 服务来定位用户位置。

5.4.1 用户验证

在本博客系统中，只有登录的用户才可以发布博客，游客只能够浏览博客，而对博客的管理则需要管理员权限。用户登录和权限可以通过在实体中存储相关的信息来实现，但是 Google 为我们提供了可以直接使用 Google 账户登录和权限验证的服务。App Engine 应用程序可以使用 Google 帐户验证用户。应用程序可以检测到当前用户是否以 Google 帐户登录，并且可以将用户重定向到 Google 帐户登录页面，以便登录或新建一个帐户。用户登录应用程序后，应用程序可以访问用户的电子邮件地址。应用程序也可以检测当前用户是否为管理员，这样便可在应用程序中轻松实现仅管理员区域 。该服务中最核心的两个类为 UserServiceFactory 和 UserService：

userService.createLoginURL(String url)用于重定向到 Google 的登录界面；

userService.isUserLoggedIn()用于检测用户是否登录；

userService.getCurrentUser()用于获取登录用户的信息；

userService.isUserAdmin()用于检测用户是否为管理员；

userService.createLogoutURL(String url)用于返回当前用户退出此应用程序的网址。

5.4.2 资料管理

登录的用户可以上传自己的头像和描述。头像和描述作为用户资料实体的属性被存储在数据存储区中，在最初的设计中，头像是以 Blob（任意数量的字节）存储的，通过使用 Commons-Fileupload 组件将用户上传的头像解析为字节数组存储到 Blob 中，但是经过测试上传的字节数组长度限制在 3000 左右。GAE 的 BlobStore 提供了存储图片，视频等大文件的服务，用户只要将带有文件的表单提交到 BlobStore 指定的 url 下，便可以通过 BlobStore 存储数据，然后通过 BlobKey 来指向存储的数据。BlobStore 中的核心类为 BlobstoreService、BlobKey 和 BlobInfo。

blobstoreService.createUploadUrl(String url)用于提交带有文件的表单到 BlobStore 服务将上传文件存储后再转到处理程序的 url。

blobstoreService.getUploadedBlobs(HttpRequest request)用户获取含有上传文件的 BlobKey 的 Map，Map 的键为上传组件的名字。

blobstoreService.serve(BlobKey blobKey, HttpResponse response)用于将 BlobKey 标示的文件返回给用户。

BlobKey 是存储在 BlobStore 中文件的唯一标示，本模块将 BlobKey 以 String 的形式存储为用户资料实体的一个属性，用来关联到用户上传的头像。

BlobInfo 类存储了 blob 的信息，例如大小，类型等等，可以通过 BlobKey 来获取一个 blob 的 BlobInfo。

5.4.3 头像处理

GAE 提供了使用专用图像服务来操作图像数据的功能。图像服务可以调整图像大小，旋转、翻转和裁剪图像；可以将多个图像合并为单个图像；可以在多种格式之间转换图像数据。它还能够使用预先定义的算法提高照片质量，在该模块中，实现了用户头像的大小调整、翻转、旋转和裁剪图像。在图像服务中最核心的类为 Image、ImagesServiceFactory、ImagesService 和 Transform。Image 类表示一个图像实体，它是通过字节数据来产生图像的。Transform 类表示对图像的转换。ImagesServiceFactory 是一个工具类，它可以利用字节数据来产生 Image，可以产生多种 Transform。ImageService 类将 Transform 应用于 Image 来产生新的图像。

5.4.4 用户定位

通过此功能，登录用户能够在 Google 地图中定位到所在位置。地理定位指通过各种数据收集机制识别用户或计算设备的地理位置。通常而言，大多数地理定位服务使用网络路

由地址或内部 GPS 设备来确定该位置。某些浏览器/设备支持地理定位，但某些则不支持（或无法支持），因此您不能始终假定网络应用程序具备该功能。目前浏览器支持 3 种定位方式，它们不是 Google Map 的特性，而是公共的标准。根据浏览器支持的程度，我分三个层次依次调用 3 种定位服务，确保用户能够正确定位。

最先使用的是 W3C Geolocation 标准，此标准是 HTML5 的一部分，只被一些最新版本的浏览器所支持，使用旧版浏览器会定位失败。接下来使用 Google Gears Geolocation API 实现定位。前提是浏览器带有 Google Gears， 由于其对 W3C 标准的广泛支持，所以用这种方式定位的成功率比较高。最后使用 Geoip 公司提供的 Ip 查询服务来定位，此服务能够查询 ip 地址所在的经纬度、国家、省市等信息。这种方式是最稳妥和浏览器支持最广泛的定位方式，很多旧版的浏览器也能够成功定位，由于 ip 地址的问题，会出现一定的偏差，但是通过测试基本可以成功定位。

5.5 电子邮件发送模块实现

该模块实现了登录用户通过 GAE 邮件服务向合法的电子邮件地址发送邮件的功能。GAE 应用程序可以代表应用程序管理员和拥有 Google 帐户的用户发送电子邮件。GAE 应用程序使用 JavaMail 标准来发送邮件，但是与普通的应用程序不同，它无需设置 SMTP 服务器的信息，JavaMail 始终调用 GAE 邮件服务来发送电子邮件,邮件服务支持普通邮件和多部分的邮件。JavaMail 的核心类是 Session、MimeMessage、MimeMultipart、MimeBodyPart 和 Transport。Session 用来建立和邮件服务器的链接，MimeMessage 表示邮件实体，包含了发件人、收件人、主题、内容等邮件信息，对于多部分的邮件需要在 MimeMessage 中添加 MimeMultipart 对象，MimeMultipart 对象则由多个 MimeBodyPart 对象构成，它们可以是普通文本、Html 和邮件服务允许的 Mime 类型的附件。Transport 用来发送设置好的邮件。

5.6 App Engine 测试程序结果

博客的发布和浏览模块：

图 5-1 博客概览页面

图 5-2 博客详解页面

用户资料管理模块：

图 5-3 用户资料页面

图 5-4 头像处理页面

图 5-5 用户定位页面

电子邮件发送模块：

图 5-6 电子邮件发送页面

5.7 App Engine 测试程序部署

在本地开发服务器调试过的测试程序要上传到 GAE。可以使用 GAE 提供的管理控制台是基于网络的接口，用于管理在 App Engine 上运行的应用程序。可以使用 appspot.com 域上的免费域名来为您的应用程序提供服务，也可以绑定自己的顶级域名。

5.7.1 测试程序上传

App Engine Java SDK 含有用于和 App Engine 交互的命令。可使用该命令将您的应用程序的新版本代码、配置和静态文件上传到 App Engine。还可使用该命令来管理数据存储区索引以及下载日志数据。测试程序使用了 Google 的 Eclipse 插件来上传应用程序，要求登录 Google 账户才能够上传。

5.7.2 测试程序管理

谷歌应用程序引擎的管理控制台提供了完整的访问应用程序的公开版本的网络接口。

可以使用它创建新应用程序、配置域名、更改您的应用程序当前的版本、检查访问权限和错误日志以及浏览应用程序数据存储区。每个免费使用的用户可以最多创建十个应用程序，每个应用程序可以有多个版本。

图 5-7 应用程序管理

通过管理控制台，可以监视正在运行的实例的各项数据包括请求数、错误数、QPS（每秒查询率）、Latency（延迟时间）和 Average Memory（平均内存占用）。

图 5-8 应用程序实例检测

管理控制台的日志界面记录了应用程序运行过程中产生的日志，可以通过选择日志级别来筛选。

图 5-9 应用程序日志

通过管理控制台还可以查看应用程序的存储数据以及索引，还可以通过控制台来新增和删除实体。

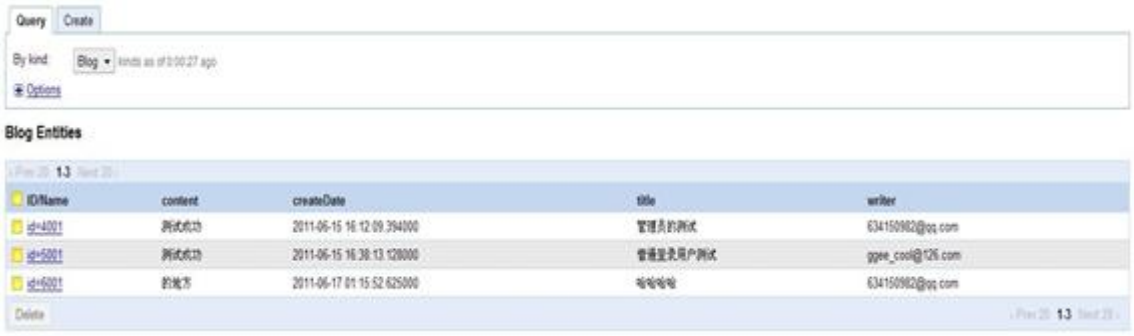

图 5-10 应用程序数据

管理控制台还提供了许多其它的管理功能，例如任务队列管理可以产看和删除正在运行的后台任务，但是本测试程序中并没有使用。BlobStore 查看界面用来管理存储在 GAE 中的大文件，但是需要开启计费模式才能够使用。通过管理控制台，可以管理和监控运行在 GAE 上的网络应用程序，了解应用程序的负载，及时的做出调整。

5.7.3 GAE 应用程序配额和限制

App Engine 应用程序可以消耗的资源上限为特定的最大值或配额，以日为单位结算。利用配额，App Engine 可确保您的应用程序不会超出预算，并且在 App Engine 上运行的其他应用程序不会影响您的应用程序的性能。收费配额是由您（应用程序管理员）设置的资源最大值，用来避免应用程序的成本超出您的预算。每个应用程序可免费获得一定数量的各种收费配额。您可以通过启用付费、设置每日预算，然后将预算分配给配额来为应用程序增加收费配额。您只需对应用程序实际使用的资源以及使用的超过了免费配额上限的资源量付费。App Engine 会记录应用程序在一个日历日中使用的各资源的量，并在此量达到资源的应用程序配额时认为资源耗尽。当应用程序耗尽分配的全部资源时，如果没有补充配额，则资源会变得不可用。App Engine 为各种服务都制定了配额限制，必须在限制下使用 App Engine 提供的各项资源。

5.7.3.1 用户请求配额

Requests

Quotas are reset every 24 hours. Next reset: 5 hours

Resource	Daily Quota			Rate
CPU Time		0%	0.00 of 6.50 CPU hours	Okay
Requests		0%	19 of Unlimited	Okay
Outgoing Bandwidth		0%	0.00 of 1.00 GBytes	Okay
Incoming Bandwidth		0%	0.00 of 1.00 GBytes	Okay
Secure Requests		0%	0 of Unlimited	Okay
Secure Outgoing Bandwidth		0%	0.00 of 1.00 GBytes	Okay
Secure Incoming Bandwidth		0%	0.00 of 1.00 GBytes	Okay
Backend Usage		0%	$0.00 of $0.72	Okay

图 5-11 用户请求配额

GAE 提供了每天 6.5 cpu 小时来处理用户发起的请求，传入和传出的带宽配额限制在 1GB。

5.7.3.2 数据存储区配额

Storage

Datastore API Calls	0%	16 of Unlimited	Okay
Datastore Queries	0%	8 of Unlimited	Okay
Blobstore API Calls	0%	0 of Unlimited	Okay
Total Stored Data	0%	0.00 of 1.00 GBytes	Okay
Blobstore Stored Data	0%	0.00 of 1.00 GBytes	Okay
Data Sent to Datastore API	0%	0.00 of Unlimited GBytes	Okay
Data Received from Datastore API	0%	0.00 of Unlimited GBytes	Okay
Datastore CPU Time	0%	0.00 of Unlimited CPU hours	Okay
Datastore Entity Fetch Ops	0%	31 of Unlimited	Okay
Datastore Entity Put Ops	0%	1 of Unlimited	Okay
Datastore Entity Delete Ops	0%	0 of Unlimited	Okay
Datastore Index Write Ops	0%	9 of Unlimited	Okay
Datastore Query Ops	0%	8 of Unlimited	Okay
Datastore Key Fetch Ops	0%	0 of Unlimited	Okay
Datastore Id Allocation Ops	0%	0 of Unlimited	Okay
Number of Indexes	0%	0 of 200	Okay
Prospective Search Subscriptions	0%	0 of 10,000	Okay

图 5-12 数据存储配额

数据存储区存储数据的免费默认配额为 1GB，索引数默认配额为 200 个。数据存储区 API 调用、发送至 API 的数据、从 API 接收的数据没有限制，将计入传入传出带宽配额，数据存储区的 Cpu 时间计算计入 CPU 时间配额。BlobStore 中存储的数据计在数据存储区的配额中，对 BlobStore API 的调用没有限制。

5.7.3.3 图片处理配额

Image Manipulation

Image Manipulation API Calls	0%	0 of Unlimited	Okay
Data Sent to API	0%	0.00 of Unlimited GBytes	Okay
Data Received from API	0%	0.00 of Unlimited GBytes	Okay
Transformations executed	0%	0 of Unlimited	Okay

图 5-13 图片处理配额

图片处理 API 调用、发送至 API 的数据、从 API 收到的数据、执行的转换次数都没有限制，将计入传入传出带宽配额中，发送到服务的图像的最大数据大小和从服务接收的图像的最大数据大小的都为 1MB。

5.7.3.4 邮件发送配额

Mail

Mail API Calls	0%	0 of 7,000	Okay
Recipients Emailed	0%	0 of 2,000	Okay
Admins Emailed	0%	0 of 5,000	Okay
Message Body Data Sent	0%	0.00 of 0.06 GBytes	Okay
Attachments Sent	0%	0 of 2,000	Okay
Attachment Data Sent	0%	0.00 of 0.10 GBytes	Okay

图 5-14 邮件配额

每日可以使用 7000 次的邮件 API 调用，每日最多可以使用应用程序给 2000 个收件人地址发送邮件，可以给作为管理员的收件人发送 5000 封邮件。每封邮件的大小限制为 1MB，每日可发送的正文数据量为 60MB，计入到传出带宽配额，每日允许携带的附件数为 2000

个，可发送的附件数据量为 100MB，计入到传出带宽配额之内。

结论

云计算是 Google 于 2006 年提出的概念， 云计算描述了一种基于互联网的新的 IT 服务增加、使用和交付模式，让计算资源的使用像使用水电一样的简单。由于云计算具有十分广阔的应用前景，已引起了许多 IT 厂商和学术界的高度重视，各种云计算技术如雨后春笋般涌现。本课题研究的是 Google 云计算平台的核心技术和云计算应用程序的开发，通过查阅云计算的相关期刊以及 Google 发表的介绍其核心技术的学术论文，对云计算的基本特征、体系结构有了总体的认识，对 Google 云计算平台的组成和运行原理有了更深的认识。之后基于 Google App Engine 编写了应用程序来测试 Google 云计算平台的使用。

在这里对本课题的研究工作总结如下：

1) 根据目前已经存在的云计算平台，总结出了通用的云计算平台体系结构，展示了云计算的各层次组成及使用的关键技术

2) 分析了在云计算体系结构的各个层次中 Google 所使用的技术以及 Google 云计算平台的应用领域。

3) 通过学习 Google 发表的学术论文，分析了 Google 云计算平台中用来构建底层软件设施的核心技术，以及这些核心技术在 Google 云计算平台的工作方式。

4) 通过学习 App Engine 提供的文档，掌握了通过 java 语言使用 Google 云计算服务的方法，并设计和实现了一个应用程序来测试 Google 云计算平台。

本文所研究的课题由于时间的限制及个人能力有限，设计的测试程序只是对 Google 云计算的部分服务进行了简单的使用，还有部分比较高级的服务没有进行测试，而且只是测试了服务的使用，并没有对服务的响应时间、处理速度、可靠性等性能数据进行测试。所以本课题只是对 Google 云计算平台的初步测试，更加全面系统地测试 Google 云计算平台的情况，是我接下来的研究工作

参考文献

[1] 陈全,邓倩妮. 云计算及其关键技术[J].计算机应用,2009,29(9):2562-2567.

[2] Cordeiro T. ,Damalio D,Pereira N. Open Source Cloud Computing Platforms:Grid and Coop-erative Computing(GCC);2010 9th Intenational Conference on,1-5 Nov.2010[C].Fed.Univ.of Pernambuco(UFPE),Recife,Brazil 2010.

[3] 孙健,贾晓菁. Google 云计算平台的技术架构及对其成本的影响研究[J].电信科学,2010,(1):38-44.

[4] Luiz Barroso,Jeffrey Dean,Urs Hoelzle.Web Search for a Planet: The Google Cluster and Architecture[J].IEEE Micro,2003,23(2):22-28.

[5] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System[EB/OL]. http://labs.google.com/papers/gfs.html.

[6] Jeffrey Dean,Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters [EB/OL].http://labs.google.com/papers/mapreduce.html.

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows,Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable:A Distributed Storage System for Structured Data[EB/OL].http://labs.google.com/papers/bigtable.html.

[8] Mike Burrows. The Chubby Lock Service for Loosely-Coupled Distributed Systems[EB/OL]. http://labs.google.com/papers/chubby.html.

[9] 房秉毅,张云勇,程莹,徐雷. 云计算国内外发展现状分析[J]. 电信科学,2010,(s1):1-5.

[10] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, Maik Lindner. A Break in the Clouds: Towards a Cloud Definition[J]. ACM SIGCOMN Computer Communication Review,2009, 39(1):50-55.

[11] 刘鹏. 云计算[M]. 北京:电子工业出版社,2010.

[12] WEISS A. Computing in the clouds [J]. ACM Networker,2007,11 (4):16-25.

[13]蔡键,王树梅.基于 Google 的云计算实例分析[J].电脑知识与技术,2009,25(5):7093-7107.

[14] Anthony T. Velte，Toby J. Velte，Robert Elsenpeter. 云计算实践指南[M].北京-机械工业出版社,2010.

[15] Google. Google App Engine Doc[EB/OL]http://code.google.com/intl/zh-CN/appengine/.

[16] Soltis R.Steven,Erickson M.Grant,Preslan W.Kenneth. The Global File System: A File System for Shared Disk Storage,IEEE Transactions on Parallel and Distributed Systems,1997 [C].

[17] 匡胜徽,李勃. 云计算体系结构及应用实例分析[J].计算机与数字工程, 2010,38(3):60-63.

[18] 陈康,郑纬民. 云计算:系统实例与研究现状[J].软件学报,2009,20 (5):1337～1348.

[19] 陈丹伟,黄秀丽,任勋益. 云计算及安全分析[J].计算机技术与发展,2010,20(2):99-102.

[20] 孙颜珍. 新型网络应用模式-云计算初探[J].电脑知识与技术,2008,S2:127.

[21] Huan Liu, Dan Orban.GridBatch. IEEE International Symposium:CCGRID 2008:Cloud Computing for Large-Scale Data-Intensive Batch Applications, 19-22 May 2008[C].

[22] Buyya R,Chee Shin Yeo,Venugopal S.Market-Oriented Cloud Computing: Vision,Hype,and Reality for Delivering IT Services as Computing Utilities:High Performance Computing and Communications 2008:10th IEEE International Conference on,25-27 Sept. 2008[C].Dept.of Comput. Sci. & Software Eng., Univ. of Melbourne, Melbourne, VIC.

[23] 尹国定,卫红. 云计算—实现概念计算的方法[J].东南大学学报(自然科学版),2003,4(33):503-506.

[24] 蓝伟文,李长虹,李晓飞,溥江. 云计算-未来 IT 技术应用的趋势[J].贵州工业大学学报(自然科学版),2008,5(37):151-152.

致谢

非常感谢我的指导教师王德文教授，本论文是在导师的悉心指导和精心培养下完成的，论文的选题、研究和撰写始终得到导师的指导和帮助。王老师深厚的学术造诣、坚持不懈的探索精神以及对学术前沿的敏锐的洞察力，使学生终身受益。在此，学生谨向导师表示最崇高的敬意和最衷心的感谢！

感谢计算机科学与技术学院所有的领导和老师，他们为我的学习和生活提供了很大的方便和有利条件，是我的学业能够顺利完成。同时也要感谢华北电力大学曾经教授过我的老师们，他们的教侮使我终身难忘。

感谢给我提供过帮助的同学们，是你们的热心帮助和耐心的教导使我的课题研究顺利进行。

感谢我的家人，正是你们无私的帮助、关怀和爱，使我的生活如此幸福温馨，你们给了我人生和事业前进的动力。

最后，再一次向所有曾帮助和关心过我的师长、亲人、朋友表示感谢！