上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

# 学士学位论文

THESIS OF BACHELOR


论文题目：云环境下高性能数据库构建的研究与实践

学生姓名：万志程

学生学号：5090379104

专　　业：软件工程

指导教师：邹恒明

学院(系)：软件学院

# 上海交通大学

# 本科生毕业设计（论文）任务书

课题名称：云环境下高性能数据库构建的研究与实践

执行时间：2012 年 11 月 至 2013 年 06 月

教师姓名：邹恒明 职称：教授

学生姓名：万志程 学号：5090379104

专业名称：软件工程

学院(系)：软件学院

**毕业设计（论文）基本内容和要求：**

云数据库是在 SaaS（software-as-a-service：软件即服务）成为应用趋势的大背景下发展起来的云计算技术，它极大地增强了数据库的存储能力，消除了人员、硬件、软件的重复配置，让软、硬件升级变得更加容易，同时也虚拟化了许多后端功能.云数据库具有高可扩展性、高可用性、采用多租形式和支持资源有效分发等特点。可以说，云数据库是数据库技术的未来发展方向。但是对于学术界而言，要想在云数据库中提供类似于现有 DBMS 的丰富功能，比如查询、索引和事务处理，仍然有许多亟待解决的问题。云数据库领域中的研究问题主要包括：云数据库中数据模型设计、编程模型、服务器体系架构设计、事务一致性和基于云数据库的容灾等。本设计旨在改进或重新设计当前的数据模型、体系架构以及与之相关的若干种云数据库高级特性。具体内容及要求如下：

1. 调研当前存储系统所使用的数据模型，进行改进或重新设计一种适用于的云计算环境的数据模型；

2. 利用该数据模型设计并开发一种高性能的数据库原型；

3. 调研当前云数据库系统架构，进行改进或设计一种新的系统架构，并利用上述原型构建云数据库，使之能保证可扩展性；

4. 测试，改进，性能调优如资源合理分配和负载均衡等；

5. 探究高级特性如高可用性、事务一致性、备份恢复、数据迁移、编程模型等并选择性实现。

**毕业设计（论文）进度安排：**

| 序号 | 毕业设计（论文）各阶段内容 | 时间安排 | 备 注 |
|---|---|---|---|
| 1 | 调研，改进或设计新数据模型 | 2012.11-2013.01 | |
| 2 | 开发数据库原型 | 2013.01-2013.02 | |
| 3 | 调研，改进或设计新架构，构建云库 | 2013.02-2013.03 | |
| 4 | 测试，改进、性能优化 | 2013.03-2013.04 | |
| 5 | 高级特性探究及选择性实现 | 2013.04-2013.05 | |
| 6 | 论文撰写，准备答辩 | 2013,05-2013.06 | |

**课题信息：**

课题性质：设计☐ 论文☐

课题来源：国家级☐ 省部级☐ 校级☐ 横向☐ 预研☐

项目编号 _____

其他_____

指导教师签名：_____

年 月 日

**学院（系）意见：**

院长（系主任）签名：_____

年 月 日

学生签名：_____

年 月 日

# 上海交通大学
# 毕业设计（论文）学术诚信声明

本人郑重声明：所呈交的毕业设计（论文），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：

日期：　　年　　月　　日

# 上海交通大学
# 毕业设计（论文）版权使用授权书

本毕业设计（论文）作者同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本毕业设计（论文）。

**保密**□，在___年解密后适用本授权书。

本论文属于

**不保密**□。

（请在以上方框内打“√”）

作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

# 云环境下高性能数据库构建的研究与实践

## 摘要

随着云计算的发展, 云数据库的重要性和使用价值日益得到体现。本文首先介绍了云数据库的特性、影响和相关产品，详细讨论了云数据库领域的研究问题, 包括数据模型、系统体系架构、事务一致性、编程模型、数据安全、性能优化和测试基准等，并针对现有的应用环境提出了构建云数据库的从数据模型到系统架构再到高级特性的基本解决方案，主要研究并开发一个云数据库系统原型。该系统原型结合了 NoSQL 数据模型中的图模型、去中心化分布式存储等现有存储技术，并做了适当改进，使得本系统能够提供可扩展性、高性能和负载均衡等云数据库的基本特性。同时，在大量调研和实证的基础上又逐个研究了高级特性的研究现状和本系统能采用的实现方式，包括事务一致性、高可用性、数据安全、文件组织、性能优化、测试基准、备份/恢复、数据迁移、信息检索。其中，高可用性、数据安全、文件组织、备份/恢复和信息检索均已实现。然后通过完整一致的性能测试验证了其可行性。最后讨论了本研究的局限和未来的研究方向。

**关键词：** 云计算, 云数据库, 数据模型, 分布式系统

# ON CONSTRUCTION OF HIGH-PERFORMANCE CLOUD DATABASE

## ABSTRACT

With the development of Cloud Computing, it is increasingly shown that the importance and utility of Cloud Database. This research at first introduces the characteristics, influence and related products of Cloud Database. Then research questions in the field are discussed in details, including the data model, system architecture, transactional consistency, data security, performance optimization and benchmark. According to existing application environment we put up out solution for constructing Cloud Database from data model, system architecture to advanced features. The main product of this research is a prototype of Cloud Database, which combined graphic model of NoSQL data model and existing storage technology such as decentralized distributed storage and has made some improvements, which enabled the system to display basic characteristics of Cloud Database such as extensibility, high performance, load balancing and so on. Meanwhile, this research looked into current research of advanced features and possible implementation ways for this system, including consistency of tasks, high usability, data security, file organization, performance optimization, testing benchmark, backup and recovery, data migration, and information retrieval, among which high usability, data security, file organization, backup and storage, and information retrieval have already been realized. Then, the correctness of this system was proved by complete and consistent performance test. Finally we discussed the limitations of this study and future research direction.

**Key words:** Cloud Computing, Database Management System, Data Model, Distributed System

# 目 录

# 第一章 绪论

云计算（Cloud Computing）是一种通过可扩展的虚拟资源来提供服务的技术。云只是一种抽象的说法，主要指代万维网的下层基础设施。因为云计算的潜在市场很大，许多 IT 巨头都已涉足云计算。数据库领域也因此受到一些影响。不少已有的数据库提供商，如 MYSQL，DB2，SQL Server 等都相继开发了时候现有环境的新数据库产品。一些原本不涉足数据库领域的大企业也开始加入了新市场的争夺，如谷歌，亚马逊[11]等。云数据库发展的速度之快超出大多数人的想象，甚至有要把传统数据库取而代之之势。

本章将结合我们已经进行的调研成果对云数据库及其相关研究进行综合阐述。本章第 1 节是对云数据库的简介, 包括云数据库的几种特性和与传统分布式数据库的区别。第 2 节介绍当前市面上存在的几种云数据库产品。第 3 节阐述本项目的研究目的和意义。第 4 节阐述根据上述信息得到的构建云数据库的研究思路及方法。

## 1.1 云数据库简介

云数据库是在 SaaS（software-as-a-service，软件即服务）大行其道而衍生出来的云计算技术，它使得许多后台功能透明化，省却了用户重复配置软硬件的麻烦，还容易进行升级。云数据库具有可扩展性、高可用性、多租户和资源合理分配等特点。如图 1-1 所示，云数据库对用户隐藏了所有底层实现细节，即所谓的透明化。对于用户来说，使用云数据库的感觉与传统单点数据库无异，但是这个感觉没什么不同的数据库却几乎能存储无限的数据。

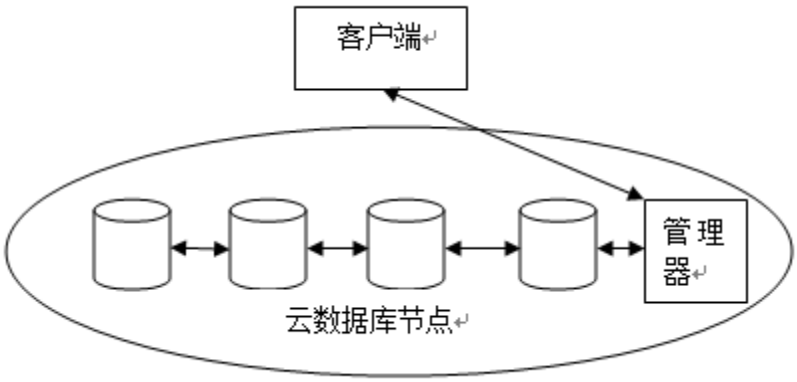



**图 1-1 云数据库示意图**

### 1.1.1 云数据库的特性

云数据库具有以下特性:

（1） 动态可扩展性：云数据库理论上可以无限扩展以满足不断增加的数据存储需求。例如, 对于一个 B2C（Business-to-Customer，商家对顾客）电子商务公司,会存在节假日突发性的产品需求变化。在这种时候就可以通过临时增加存储或计算节点来处理额外任务的方式来迅速解决额外需求。

（2） 高可用性：不会出现不可用的情况。因为云数据库中的数据通常是复制的分布的，所以单一节点失效并不会影响到整个系统的功能，因为其它节点可以代替实效节点完成剩

余的任务。如谷歌，亚马逊等大型云数据库产品供应商都在世界不同角落分散有数据中心，所以能够提供高水平的容错能力。

（3） 易用性或者透明性：使用云数据库的用户不必了解系统的运行细节和地理位置，只需通过身份验证即可使用。

（4） 应用开销较低：云数据库的多租户模式能够充分利用资源，避免了软硬件资源的浪费。

（5） 大规模并行处理：云数据库的分布性决定了这一特性，并行处理对于科学计算、实时分析都有重要意义。

### 1.1.2 云数据库与传统分布式数据库的对比

分布式数据库系统通常使用较小的计算机系统，每台计算机可单独放在一个地方，每台计算机中都有 DBMS（Database Management System，数据库管理系统）的一份完整拷贝副本，并具有自己局部的数据库，位于不同地点的许多计算机通过网络互相连接，共同组成一个完整的、全局的大型数据库。分布式数据库存在时间很长，通常使用不共享的结构来进行存储。

分布式数据库和云数据库的相似之处在于数据都存储在不同节点。同时，两者又有以下几点不同：

（1） 相比分布式数据库云数据库的扩展性更好。因为在设计时就考虑到可扩展性的要求，云数据库的数据模型、分区模式和一致性要求就尽量简化、降低，这样对性能的损耗也就最小。

（2） 比起分布式数据库“一人一个”的使用方式，云数据库多租的模式具有更高的使用效率。用多少租多少能够极大地利用已有资源并节省开销。

## 1.2 云数据库产品

当前市场上的云数据库供应商一共有传统的数据库企业、涉足数据库市场的互联网企业和一些创业公司（如 Vertica 和 EnterpriseDB）三种。

即使现在有一些云数据库产品已经实现了大数据的处理，如谷歌的 BigTable 等。但是这些产品距离我们理想的云数据库的完整概念还有很大的距离。现在的产品最多只能算是实现了基本功能，还有很多高级的特性有待研究。表 1-1 给出了目前市场上常见的云数据库产品。

**表 1-1 常见的云数据库产品**

| 数据模型 | 部署在虚拟机上 | 作为数据库服务 |
|---|---|---|
| 关系模型 | Oracle Database | Amazon Relational Database |
| | IBM DB2 | Microsoft SQL Azure |
| | Ingres (database) | Heroku PostgreSQL |
| | PostgreSQL | Clustrix Database as a Service |
| | MySQL | Xeround Cloud Database |
| | NuoDB | EnterpriseDB Postgres Plus |
| | GaianDB | GaianDB |
| | | ClearDB ACID-compliant |
| 非关系模型 | CouchDB | |
| | Hadoop | |
| | Apache Cassandra | |
| | Neo4J | |

续表 1-1

| 数据模型 | 部署在虚拟机上 | 作为数据库服务 |
| --- | --- | --- |
| 非关系模型 | MongoDB | Amazon DynamoDB<br>Amazon SimpleDB<br>Cloudant Data Layer<br>Database.com by SalesForce<br>Google App Engine Datastore<br>MongoDB Database as a Service<br>Cloudbase.io Cloud Database |

## 1.3 研究目的和意义

如上一节所述，要在云数据库中实现传统数据库中一些诸如索引，事务处理等高级特性仍待未来的研究。这里小列几项当前该领域研究的热点：数据模型设计、编程模型、体系架构设计、数据迁移、事务一致性、数据访问控制和权限管理、性能优化、测试基准、数据安全等。本设计旨在改进或重新设计当前的数据模型、体系架构以及与之相关的若干种云数据库高级特性，以提高当前云数据库的服务水平，同时也实现一种新的云数据库原型。

## 1.4 研究思路和方法

（1） 调研当前存储系统所使用的数据模型，进行改进或重新设计一种适用于的云计算环境的数据模型；

（2） 利用该数据模型设计并开发一种高性能的数据库原型；

（3） 调研当前云数据库系统架构，进行改进或设计一种新的系统架构，并利用上述原型构建云数据库，使之能保证可扩展性；

（4） 测试，改进，性能调优如资源合理分配和负载均衡等；

（5） 探究高级特性如高可用性、事务一致性、备份恢复、数据迁移、编程模型等并选择性实现。

## 1.5 本文组织结构

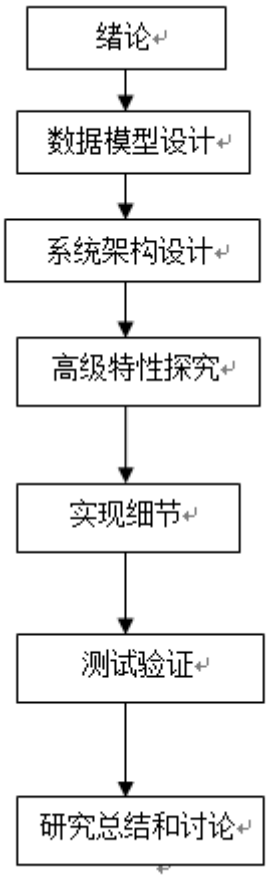



**图 1-2 本文的组织结构**

如图 1-1 所示，本文的组织结构如下：

第二章将分析本数据库基本的数据模型的设计。

第三章将对首先整个数据库的架构与设计进行详细阐述，然后将对架构中的其他问题进行说明。

第四章主要介绍各种数据库高级特性在本数据库上的可行性和设计方法。

第五章介绍本数据库的具体实现细节和接口说明。

第六章将对本数据库进行详细的性能测试与实验分析。实验将对框架的可扩展性、高可用性、高性能等指标进行测试。

第七章是对本课题的总结与展望。第七章首先系统性的总结了本课题的研究内容以及研究结果，接下来对本课题中的缺陷进行了分析，提出了对未来研究工作的参考意见。

## 1.6 本章小结

本章首先介绍了课题的研究背景，包括云计算和云数据库的基本概念、特性和产品，然后提出了当前领域内的研究问题现状、本文的研究目的和意义，接下来介绍了本课题的主要研究思路和方法，最后对论文的组织结构进行了展示。

# 第二章 数据模型设计

数据模型（Data Model）是数据特征的抽象，是数据库管理的教学形式框架。数据库系统中用以提供信息表示和操作手段的形式构架。数据模型包括数据库数据的结构部分、数据库数据的操作部分和数据库数据的约束条件。本章将介绍当前云数据库数据模型研究现状，本文所设计并采用的模型及其特点。

## 2.1 数据模型研究现状

数据模型因应用场景和使用需求而异，下面就介绍几种常见模型。

### 2.1.1 键/值模型

有不少产品使用了键/值模型，如 BigTable，HBase 等等，以下我们就以这两种为例介绍一下该模型。

BigTable 存放行列信息和时间戳到单元格里。BigTable 的数据簇中包含很多表，每个表又包含很多小表（一定数量的行组，可设定大小），每个小表存储在一个服务器节点上。它通过三个值来进行索引，分别是行键、列键和时间戳。数据都是以字符串的形式存储 [15]：

（1） 行键：行数据通过字典序来维护，同时由此组成的行组也是数据分区和负载均衡的基本单位；

（2） 列键：这是用作访问控制的键，即同一列的数据属于同一列家族，通常具有同样的数据类型，便于控制和压缩；

（3） 时间戳：用来保证数据的一致性，即每次修改时存储时间戳作为版本号，读取时读最新的版本。

因为 BigTable 的表之间没有连接关系，数据分区只需根据行区间即可。

HBase[19]是一个使用 JAVA 编写、开源、分布式、面向列、高可用、高性能的数据库，。HBase 既可以提供结构化数据的可靠存储，又利用 Mapreduce 建立多级索引实现了高性能。同时，它还可以设置存储在内存当中以提高性能。

还有一些云数据库也采用了键/值模型，如 SimpleDB。SimpleDB 最大的不同就是采用了静态数据（哈希函数）划分的方法。这种方法虽然容易实现，但是会造成数据的离散存储，因此 SimpleDB 做了单表数据量上限的限制。

此外，因为单键访问对于一些网络应用比如在线游戏、网络社区等合作性比较强的系统显得比较麻烦。由此，文献[23]设计了通过键组来对数据进行访问的系统，名为 G-Store。

总之，键/值模型与关系型模式的区别就是：第一，不存在表间关系操作；第二，整个模式只有一个索引——行键。

### 2.1.2 关系模型

关系型模型除了有用于分区的行组，还有表组的概念。表组是分区键相同的表的集合。行组与键/值模型一样。表组则通常用于加快查询效率。如把两张有关联的表放到一个数据节点上构成一个表组。这样链接操作会快很多。

### 2.1.3 其它模型

除了这两种模型还有其他的种类。比如一种名为 CloudDB 的系统[24]就可以提供最多三种的数据模型以供存储，它甚至能够提供列式存储[25]。当然，具体选择哪一种则根据应用的使用环境，用户需求等等因素来决定。

## 2.2 图数据模型

对象关系的不匹配使得把面向对象的“圆的对象”挤到面向关系的“方的表”中是那么的困难和费劲，而这一切是可以避免的。关系模型静态、刚性、不灵活的本质使得改变模式以满足不断变化的业务需求是非常困难的。所以，关系模型很不适合表达半结构化的数据。

而键/值模型虽然简单高效,但是要存储一些具有复杂结构的数据仍然需要很多处理。

除上述两种常见的云数据库数据模型以外，还有一些不常见的类型。网络（图）就是其中一种。网络是一种非常高效的数据存储结构。人脑是一个巨大的网络，万维网也同样构造成网状，这些都不是巧合。关系模型可以表达面向网络的数据，但是在遍历网络并抽取信息的能力上关系模型是非常弱的。网络（从数学角度叫做图）是一个灵活的数据结构，可以应用更加敏捷和快速的开发模式。

图数据库是一种基于图数据结构的数据存储库，能够对一些复杂的关系数据进行存储。受到图数据库的启发，本文提出了一种通用的数据模型——图模型。

### 2.2.1 基本图数据模型

一个图数据库是一个数据库，使用图形结构表示和存储节点、边和属性数据。根据定义，图形数据库是任何提供免费指数邻接的存储系统，。这意味着，每一个元素都包含一个直接指到其相邻元素的指针，而索引查找是不必要的。

图形数据库基于图论。图形数据库包括节点，属性和边缘。节点和面向对象编程的对象具有非常相似的性质。

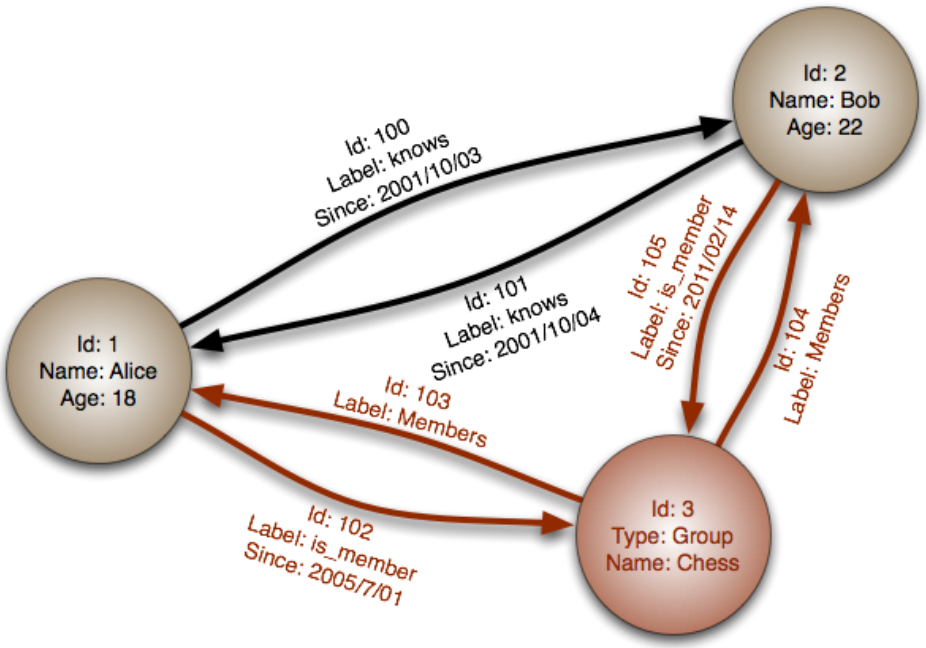



**图 2-1 图数据模型**

图 2-1 中的节点代表实体，如个人，企业账户。属性是节点相关的信息。例如，如果“用户”是一个节点，可能它就有属性如“名字”,“密码”或“邮箱”。边线连接节点和节点或节点的属性，代表了两者之间的关系。边缘也可以存储属性信息。一个具有节点互联和属性完整的图就构成了一种有意义的模式。

### 2.2.2 改进的图模型

由于节点和关系属性的复杂性过于影响性能且不够易用,在当前快速高效的云数据库实际存储中很少自己定义具有过多属性的对象。所以本文对其做了如下改进:

(1) 取消关系 id、名称和属性。

(2) 取消节点 id 并以 key 值取代，取消属性以唯一值 Data 取代。

如图 2-2 所示，每个节点仅具有一个属性 Data，节点间的边无属性。这样我们就能同时支持高关联度数据和良好查询性能的需求。

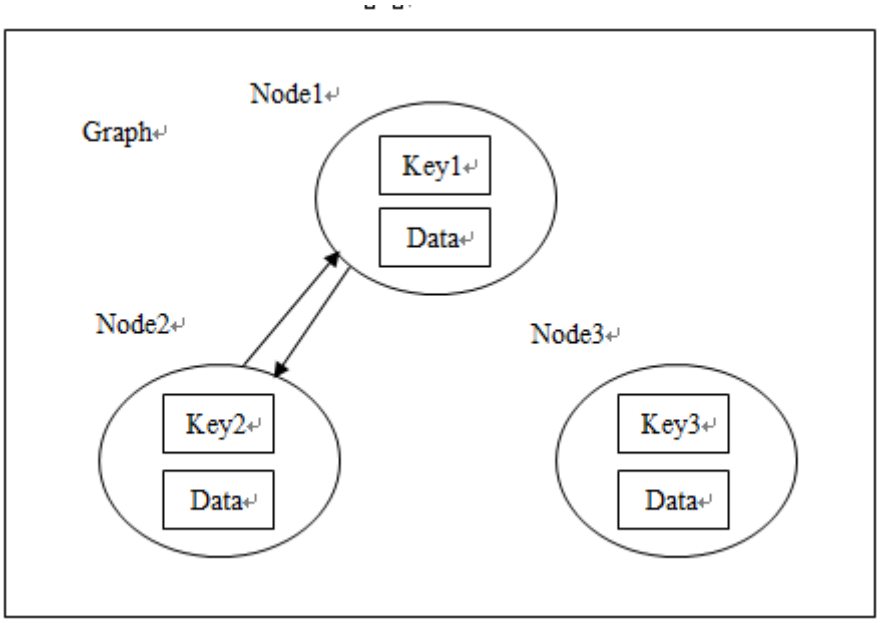



**图 2-2 改进的图数据模型**

### 2.2.3 序列化表示

对于任意一张所描述的图，都可以用一串序列化后的字符串来表示，以作为实际查询的结果。为方便转化和处理，我们采用了基于 JSON 的序列化格式，如表 2-1 所示：

**表2-1 图数据的序列化表示**

| 结构 | 序列化（JSON） |
|---|---|
| 节点 | {“key”:“data”} |
| 关系 | {“key1”:[“key2”，“key3”，..... ]} |
| 图 | {“node1 key”：“node1 data”, “node1 key”:[“node2 key”,…]} |

如此，用户便可使用本序列号表示方法输入，当然，同时也有更友好的方式（见第五章）。同时在执行图查询时返回序列化后的字符串作为结果。

### 2.2.4 与其他模型对比

图形数据库与关系型数据库相比，关联的数据集往往更快，更直接的面向对象应用程序的结构，可以更自然地扩展到大型数据集。因为通常并不需要繁重的加入操作。图数据库较少依赖于一个僵化的模式，更适合管理特设的和不断变化的数据与不断变化的架构。相反，关系型数据库通常是大量的数据元素执行相同的操作更快。

图形数据库与键/值型数据库对比,数据之间的关联度更高。

图形数据库是一个功能强大的工具，特别是高级查询和检索，根据图论的许多理论可以实现许多关系数据库中无法实现的查询方法。如两节点数据的距离（最短路径），某一节

点树的深度等等。

但最重要的还是图的通用性。

### 2.2.5 通用性

图模型具有很高的通用性，从理论上来说可以在应用层面转换为任意一种模型。以上一节两种模型为例,转换示例方法如下：

（1） 转换为键/值模型:因为数据访问方式可以直接通过键匹配,所以只要在用户层面避免使用图节点之间的关系功能,即拓扑为散点图的数据图结构就是键/值模型。如图 2-3 所示,每个节点之间都没有关系,节点的数据通过其键去访问。

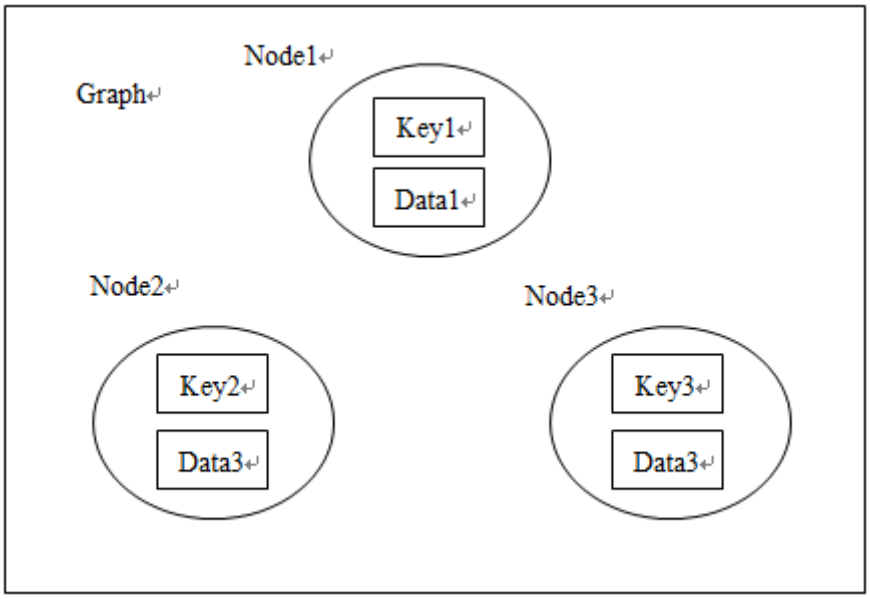



**图 2-3 图模型转化为键/值模型**

（2） 转换为关系模型：从图模型转换为关系模型需要较多用户层面的处理。此处举一例说明，假设有关系模式 People（id，Name, Age），该关系表中存有两条记录，分别是（1，Jensen, 22），（2，Micki， 22）则若使用本图模型，可采用以下存储方式。如图 2-4 所示，一颗以关系模式中的主键为顶点的树即可代表关系模式中的一条记录。只是这里的模式并不是强制性的，要保证记录模式的一致性需要在应用层面做约束。

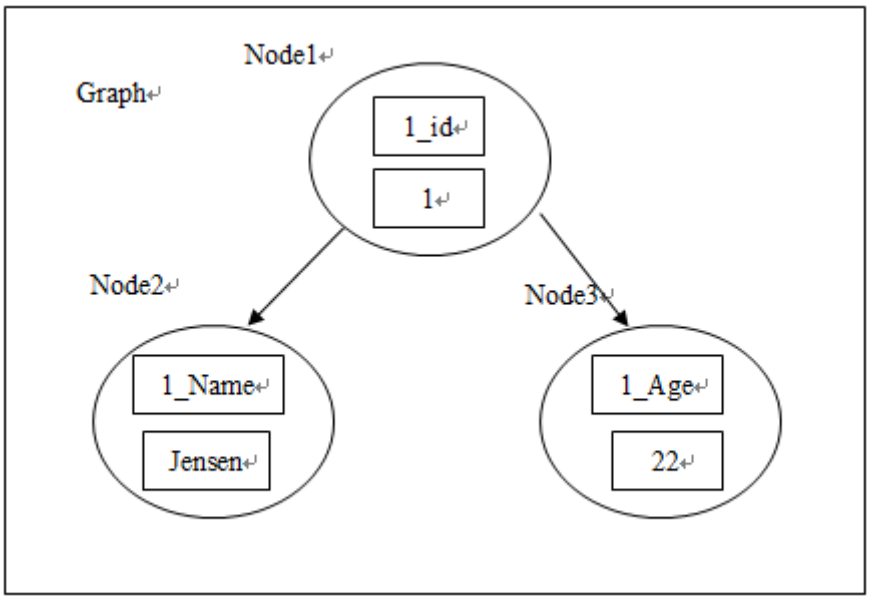



**图 2-3 图模型转化为关系模型**

## 2.3 本章小结

本章介绍了几种常见的数据模型，然后引出了本文采用的改进的图模型，然后介绍了的该模型的存储方式、基本特点、序列化表示、与其他模型的对比以及通过实例介绍的广泛的通用性。

# 第三章 系统架构设计

本章主要讨论本数据库系统的架构设计，首先分析当前采用非共享架构的几个云数据库，然后再引出本系统的体系架构、系统实例、数据访问方法、数据分区等等具体问题。本系统的主要架构受启发与点对点（Peer-to-Peer）以及自动发现的技术。本架构主要的目标是能够提供大规模存储服务并能够快速响应和扩展以适应快速变化的系统。接下来就先分析已有的体系架构和本文的设计思路。

## 3.1 体系架构

采用不同数据模型的数据库通常采用不同的体系架构，下面就着重讲一下键/值模型的关系模型的架构，并为分别以 HBase 和 SQL Azure 为例。

### 3.1.1 采用键/值数据模型的云数据库体系架构

HBase[19]的架构和 BigTable 类似。其体系架构如图 3-1 所示，包括客户端、Zookeeper 模块、Hmaster、HRegionServer 和存储模块，具体功能如下：

（1） 客户端：负责用户接入 HBase；

（2） Zookeeper 模块：HRegion 的寻址入口，管理数据库数据模式，并监控 HRegionServer 的状态；

（3） HMaster：管理用户操作和负载均衡；

（4） HRegionServer：根据用户的操作响应文件 I/O，通过 HDFS 读写数据；

（5） 存储模块（Store）：存储的关键部分，Memstore 用作写入数据的缓冲区，满了会写入 StoreFile。

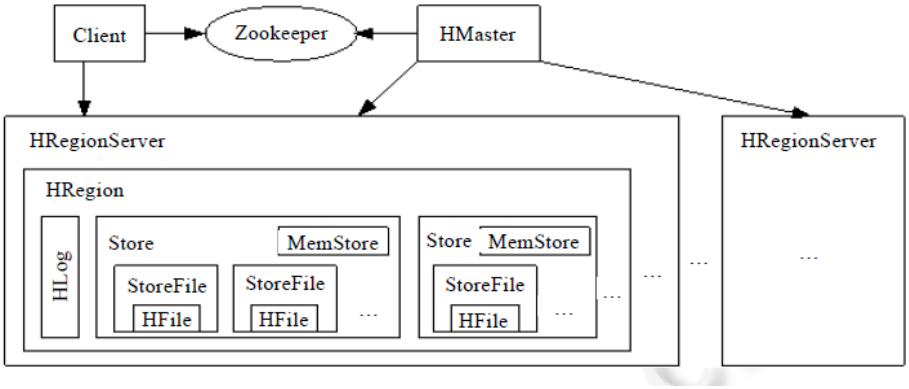



**图 3-1 HBase 的体系架构**

### 3.1.2 采用关系数据模型的云数据库体系架构

此处以 SQL Azure 为例，SQL Azure 的体系架构[24]中通过管理一个虚拟机簇来响应需求地变化，如图 3-2 所示。每台虚拟机 SQL Server VM 不仅安装了 DBMS（这里是 SQL Server），还安装了 SQL Azure Fabric 和 SQL Azure 管理服务，以管理和保障 SQL Azure 的高可用性要求。同时，每个节点内部的两种服务还会实时互相交互监控信息，以保障整个系统的可监控性。

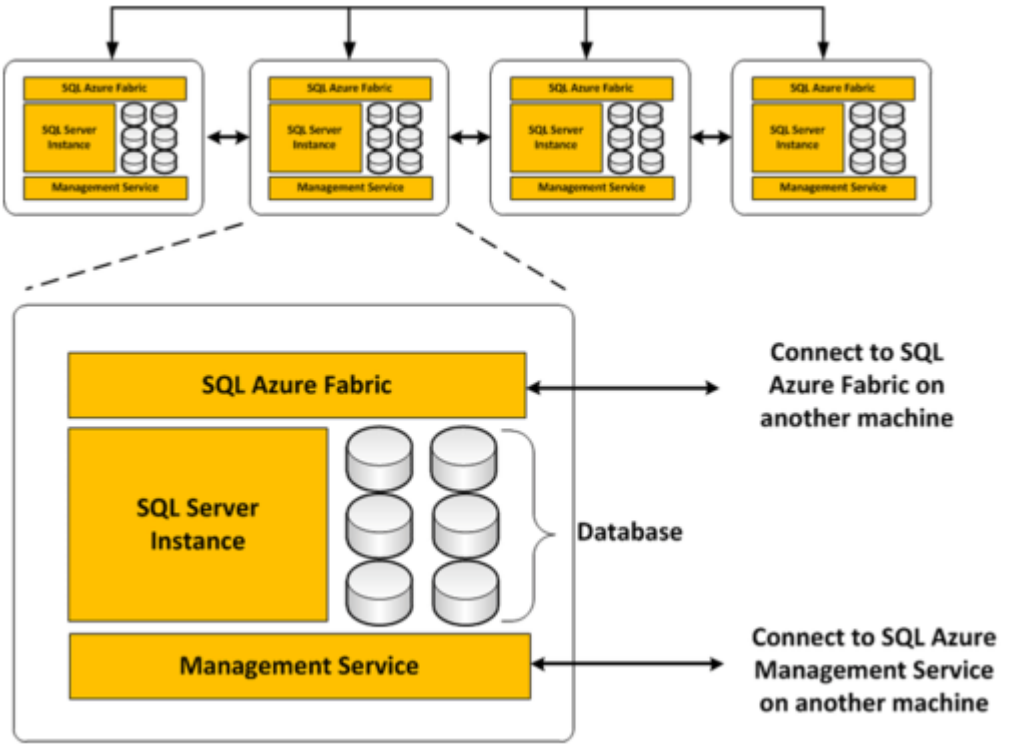



**图 3-2　SQL Azure 的体系架构**

### 3.1.3 采用图数据模型的云数据库体系架构

这两种体系架构各有各的特点，但共同的是都有全局管理的服务节点或模块。这样的结构对管理服务依赖过高且会造成一定的性能瓶颈。所以本文提出了服务端点对点（Peer-to-Peer）的系统架构。

点对点数据库系统定义为一系列通过逻辑映射关系连接在一起的数据库系统。与集成系统不同，点对点的系统节点都是独立的，他们之间是通过预定的某种映射规则来进行连接。其中每个节点都能自主选择数据交换的其它节点而无需其它节点帮助。因此，与传统分布式数据库系统不一样的是这种数据库系统没有全局控制、全局服务以及全局资源管理等节点，因此而获得了更大的灵活性。但是点对点数据库系统的主要设计难点就在于数据管理，每个节点都或多或少的与其它节点共享数据，如何区分和管理是关键。本系统的体系架构如图 3-3 所示。

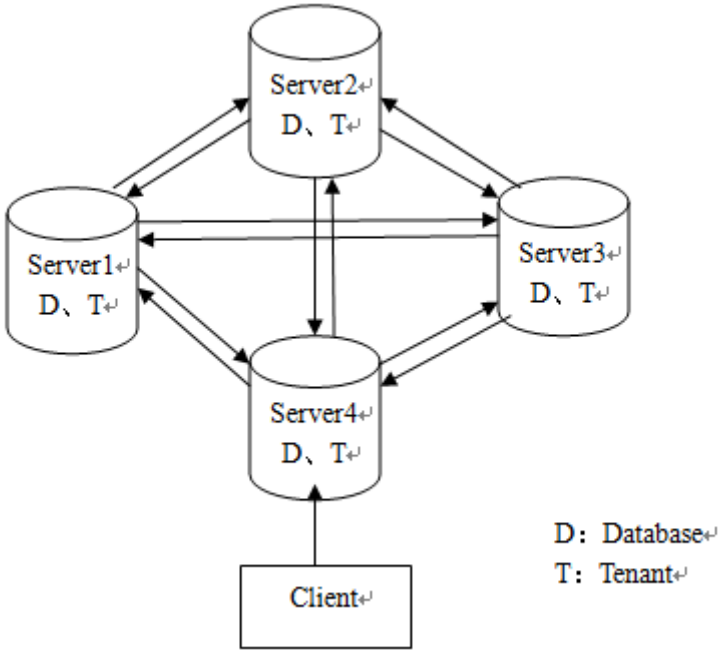



**图 3-3　wanDB 的体系架构(4 节点,数据库级分区)**

## 3.2 数据访问方法

首先阐释通常云数据库的数据访问方法。如图 3-4 所示, 当客户端需要进行查询操作时，先向管理器发起请求，得到映射图后再向对应的数据节点发起请求。当然也可以通过客户端缓存来提高性能。

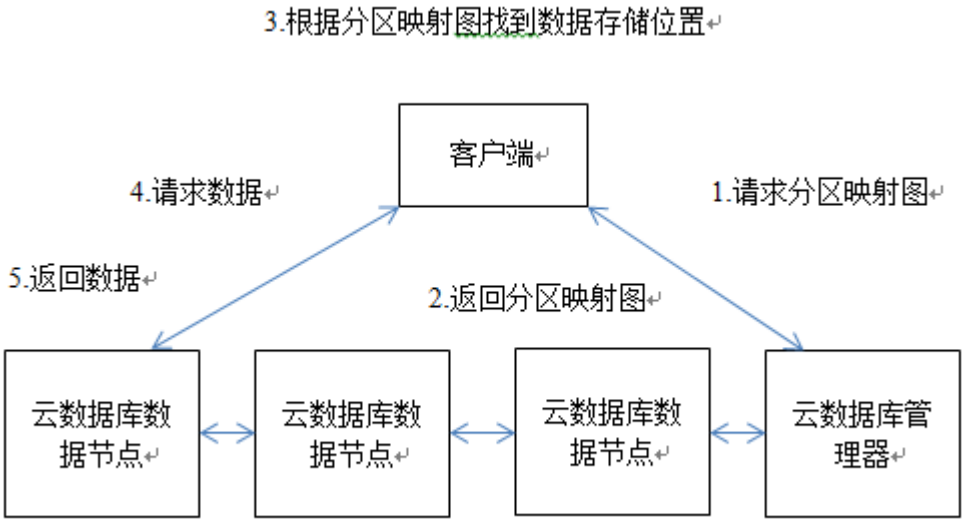



**图 3-4 数据访问常用方法**

而在本系统架构中，因为没有单独管理器的存在，所以分区查询工作需要由每一个节点自身完成。举例来说，就是客户端发出请求至任意一个节点，该节点先查找自身存储的分区映射图，得到分区节点位置后直接查询返回结果。

## 3.3 系统实例

如上一节所说，当前的云数据库大都采用以下的节点配置。

BigTable 中有 Tablet 服务器和主服务器两种节点实例。主服务器负责各种全局管理工作,如数据分配，负载均衡，垃圾收集和模式变化（如表和列家族的创建）等等，而 Tablet 服务器节点主要负责存储主服务器分配的数据。

HBase[19]大部分与 BigTable 相似。只是它的数据分区存储在 HDFS 中。同时它还依赖 Zookeeper 来保证随时都至少有一个主服务器来负责节点数据的分配和元数据信息保存。

为彻底避免主服务器单点失效的问题，本系统只采用一种实例节点。每个节点的功能描述如下：接受来自前端的请求，负责底层存储；与其它节点进行通讯，更新自身数据以及定期汇报自身状态；监控各个节点的状态，对添加新节点、删除死亡节点进行处理等。而数据在节点中主要采用多级自定义大小的方式进行分区（Partition）以及冗余副本管理（Replication）。因为存储节点是去中心化（Decentralized）的组织架构，节点的增加或者删除都是自动完成，不需要额外的用户操作。

## 3.4 数据分区

为了更适应变化的需求，本系统采用了多级分区策略，即可以根据用户需求在不同存储层面进行数据分区，主要包括数据库层、图层和结点层。

对于数据库层的分区策略来说，就是以数据库为最小分区粒度进行分区。具体来说，就是在创建数据库的时候进行存储节点选取，选取策略需满足负载均衡条件（见 3.6 节）。然后将选取节点的地址保存到本节点元数据中。最后发送创建数据库命令到该节点。同样的，当修改或读取该数据库时，也是先从元数据中读取其存储位置，再发送读取或修改命令，接收反馈。

图层的分区策略与数据库层类似，以图为最小分区粒度，在创建、修改和读取时采取相同的选取策略。不过需要注意的是因为在创建图时需要依赖于已有的数据库，所以为了确保能够执行需要先发送创建当前节点使用数据库的命令，以防止所选取节点是新加入节点等特殊情况。同时，在创建数据库时也需要广播该命令，即所有存储节点共享数据库层结构。

结点层的分区策略最为复杂，也最类似键/值模型的分区方法。首先，同图层一样，所有节点需要共享分区层的上层结构，而这次不仅是数据库，还包括图。即在创建数据库和图时都需要广播命令。然后，在创建、修改和查询结点时需要进行选取节点来执行操作。不仅如此，结点层分区最难的是关系的处理。对于同一节点上图结点之间的关系可以通过简单的指针存储方式实现。但是不同节点上的结点之间关系就需要在本地存储一份“影子”副本，即一个存储有实际结点地址的结点，再通过指针连接两个本地结点。这是通过牺牲空间来换取时间的一种措施。这样在对关系本身进行修改时就能有较高性能。

## 3.5 启动

首先需要配置好任一节点，配置参数有存储模式、哈希方法、客户端口等。接下来，所有的其它节点可以按照第一个节点的配置方法启动，启动的时候需要指定一个端口作为监听端口，接受来自其他节点的消息。最后，在任一节点上启动客户端，即可作为使用入口。任一存储节点启动的时候，都将与其它节点互发广播消息并将其它节点的地址、大小等信息记录下来。同时，存储节点建立相关的服务线程以及消息处理线程等，建立底层存储文件（如果是文件模式）。存储节点完成初始化工作之后，自身状态为初始化，将进行等待，直到客户端发出指令。至此，所有节点初始化完成。

## 3.6 负载均衡

介绍读写操作之前，需要先介绍一下负载均衡的策略。负载均衡是在创建新数据时根据当前节点及其他节点的数据量大小来进行合理分配的策略，也是保持数据库整体性能的重要手段。很明显，用户有权选择他们自己当前使用的节点可以存储多少数据，也就是说每一个节点存储的数据量上限可以自定义。但如果每个节点数据量上限相差不大，调整负载的方式就显得格外重要。在本系统中，在创建新数据结构选取存储节点的时候，我们也采取分级的模式，共有最小最优、历史最优两种模式。

（1） 最小最优：顾名思义，就是选取当前所有节点中已存储数据量最小的节点；

（2） 历史最优：就是根据历史数据，首先计算出要创建数据结构的平均数据量，然后根据所有节点中能存储这个数据结构后剩余空间最小的节点，即刚好能存储的节点。如公式（3-1）（3-2）所示，i 是节点编号，$M_i$ 代表第 i 个节点的存储量上限，$C_i$ 代表当前数据量,$L_i$ 代表该节点的剩余存储空间，H 是该数据结构的历史数据量, n 是节点总数。这样做的目的是为了尽量“填满”已有的数据节点，以给潜在的大数据结构留出更多空间。

$$L_i = M_i - C_i, i = 1,2,\ldots,n \quad (3\text{-}1)$$

$$i = Min(L_i - Avg(H)), i = 1,2,\ldots,n \quad (3\text{-}2)$$

显而易见，最小最优的性能消耗最少，几乎不需要任何计算，所有作为默认策略。用户可以根据需要进行配置。第二种策略对于分区结构数据量差异较大的应用比较有用，能够充分利用存储资源。

## 3.7 读写操作

每次的读写操作都需要先查询元数据 Metadata，也就是地址映射区。然后根据得到的地址采用负载均衡协议完成读写操作。这里为降低一致性要求以提升性能未引入锁机制（详见 4.1 节）。所以读写操作不需要更多额外处理。

## 3.8 弹性扩展

可扩展性是云数据库最重要也最基本的功能之一。由于本系统尽力避免用户的过多参与，所以增加节点的工作基本上都是自动完成，步骤如下：

(1) 假定新加入的存储节点为 A，首先初始化的过程建立本节点数据结构、服务线程池等；

(2) 节点 A 通过自身广播监听和发送获得其它节点信息；

(3) 读取其它节点的配置参数（同一集群自动配置的参数应一致），包括存储模式、永久存储、分区级别等等；

(4) 根据分区级别开始共享分区上层数据结构，如数据库、租户、图等；

(5) 复制完共享数据后即完成加入工作，可以接受客户端请求。

以上步骤均由系统自动完成，无需用户做过多配置。同时，如果数据量很小，平均每个节点的数据量都占数据量上限的 20%以下，则客户端可以通过命令来关闭（shutdown）一些节点，关闭节点的工作也是自动完成，具体步骤如下：

（1） 假定要关闭的节点为 A，则首先节点 A 将不再接受客户端请求；

（2） 然后 A 将根据负载均衡协议向其它节点发送自身数据，即发送所有创建自身数据的命令到选取的其它节点；

（3） 最后 A 再结束运行。

配置这样的弹性机制是为了满足快速变化的应用需求。

## 3.9 本章小结

本章介绍了几种常见云数据库的体系架构，然后提出了本文使用的体系架构，服务端为点对点架构，客户端可以随机访问。之后根据这个体系架构提出了相关的数据访问方法、系统实例、数据分区算法、启动、读写操作、负载均衡和弹性扩展策略。

# 第四章 高级特性探究

本章将针对云数据库中的各种高级特性做逐一介绍，包括当前研究现状、产品概述和本系统的对应解决方案等等。特性共有事务一致性、高可用性、数据安全、性能优化、测试基准、备份恢复、数据迁移、数据检索八种。

## 4.1 事务一致性

加州大学伯克利分校的 Brewer 教授在 2000 年提出了著名的 CAP 理论[28]。所谓的 CAP 指的是：

（1） Consistency(一致性)：任意的读操作总是能读到之前写入的数据；

（2） Availability(可用性)：任意的操作总是能在确定的时间内返回结果；

（3） Tolerance of Network Partition(分布性)：在分布式存储的条件下，仍然能够满足一致性和可用性。

该理论的核心论调是一个分布式系统不可能同时满足这 3 个需求，最多只能同时满足 2 个。因此，根据 CAP 理论，我们在设计一个系统时可以有几种选择：

（1） 放弃分布性。采用单例实现，显然，这种方法可扩展性很差。

（2） 放弃可用性。这对于用户和系统设计来说都是不可接受的。因为这样的系统是不确定的，很容易因单点性能瓶颈而崩溃。

（3） 放弃一致性。这样的系统对于某些应用来说（如社交网站）可以接受，因为用户不会在乎暂时读不到刚刚写入的状态或留言信息等等。而对于某些应用来说，暂时的不一致就会导致巨大的损失，如银行和证券行业，放弃一致性是绝对不可行的。

传统的数据库提供强一致性保证，但是在分布式环境下就需要花费很大代价。很多现有的云数据库产品在实现时，都放松了对事务 ACID(原子性，一致性，隔离性，持续性)四性的要求，以下试举几例：

（1）BigTable 只支持单行事务，不支持跨行事务；

（2）Amazon SimpleDB 采用最终一致性，使得所有副本不必都获得数据副本的当前最新值；

（3）Amazon Dynamo[14]也采用了最终一致性；

（4）Yahoo! PNUTS 采用时间线一致性以追求更高的可用性、容错能力和更快的响应等[21]。

对于本系统来说，要实现强一致性不是不可行，但是需要消耗很多的资源和时间。这里只是假设性的探讨一下实现的可能性，实践当中本系统并没有实现真正意义上的事务以及保持强一致性，而只是能保证最终一致性。

如果要实现强一致性，那么在读写操作之前都需要对存储有当前操作数据对象的节点进行上锁。而对于点对点的分布式系统来说，要保证同时上锁相当困难，因为当前节点在选取节点并发送上锁命令的过程中其它节点有可能正在修改数据，这种时间差造成的不一致无法避免。所以如果要保证读出的数据必定是上一次写入的，必须引入独立于数据存储模块的另一个控制系统，由此来控制整个系统的读写操作。比如利用 Zookeeper 服务来对所有存储有当前操作数据的节点并发的进行原子更新。但是这样做无异于又回到了传统的云数据库主服务器架构，又会存在单点性能瓶颈、单点失效等问题。

实际上, 分布式环境下的事务一致性就应该是灵活可变的。如果将数据库中的事务服务

从数据管理组件中单独分离出来，让这个组件来管理并发控制、恢复和访问控制。当对一致性的需求较低时，该组件就可以采取一些简单的策略如加入版本号，时间戳等来保持最终一致性。而对一致性有较高要求时就应该对整个系统的读写操作进行锁控制。甚至该系统还可以具有一定的智能来判断当前应用所需要满足的一致性要求，从而更方便的调整策略。

## 4.2 高可用性

高可用性（High Availability）通常来描述一个系统经过专门的设计，从而减少停工时间，而保持其服务的高度可用性。对于分布式系统来说，保证高可用性的解决方案有很多，下面就列举几个并分析它们各自的优劣：

（1） 数据库镜像：数据库镜像就是每一个数据库都有几个实时同步的副本，可能在不同的节点上。当主数据库（首先操作的数据库）所在节点实效时就自动切换到从数据库（其它同步数据库）来进行操作。这种方案的优点是可以提供几乎是瞬时的故障转移。不过这个解决方案也有一定的不足，最主要是其限制条件比较多, 较难实现；

（2） 日志传送：这种方式稍微比镜像慢一些。它利用日志作为中间文件来在主服务器和辅助服务器之间复制数据库。用户可以修改主服务器复制和辅助服务器还原之间的延迟。日志传送的优点在于可以手动设置同步的时间，同时也可以在故障恢复时手动还原，更加可靠；

（3） 发布-订阅模式：在数据库中通常都采用复制的方法来提高可用性，而发布-订阅模式就是一种常用的复制方法。发布-订阅模式是通过主服务器发送数据给订阅了该数据库的辅助服务器的方式实现复制。使用这种方法，优点主要是两点。一是订阅服务器可以有选择性的接收数据，提高同步传送效率。另一个优点就是数据滞后的时间短，数据同步及时。

此外还有基于硬件的高可用性方案这里不再赘述。本系统为实现高性能的设计需求，采用了日志和复制结合的方案，即在创建数据结构时可以设置需要复制的份数，从而分发到不同的节点上，同时，主节点也发送日志到辅助节点上。当主节点更新时，逐一发送更新数据和更新日志到辅助节点。当任意节点出现故障时都可以通过对比日志的新旧程度来选取新的主节点或者根据日志来恢复故障节点。这样的措施既保证了高可用性，也不至于损失过多效率。

## 4.3 数据安全

云数据库是否安全也是当前研究领域最关心的几个问题之一。云数据库的海量存储能力是一把双刃剑，这么多数据的保密性和访问控制可能是复杂而脆弱的。一是敏感数据比须加密以后再存入数据库。二是必须对不同的用户设立不同的访问级别，以此来实现访问控制。当前来看，在云数据库中对数据使用加密和应用各种级别的访问控制规则主要有以下几个研究方向：

（1） 基于元数据和特权访问控制的方法：为每个虚拟机存储器中的授权策略和云数据库中的资源都构建成特权链图，以此来进行访问控制；

（2） 采用 KP-ABE(key-policy attribute based encryption，基于关键策略属性加密)加密策略：用户的访问策略与密钥关联，通过登录即可进行访问控制；

（3） 同态加密：同态加密是基于数学难题的计算复杂性理论的密码学技术。它是一种不需要解密算法就可以对数据进行计算的方法。

本数据库系统根据当前的条件对不同的操作类型进行了访问控制，同时也采用了文件加密和数据加密两种方式来保证数据安全。

（1） 访问控制：对于普通租户，其可操作的范围只有自身信息，拥有用户权限的数据库。而对于数据库管理员，还可以对租户本身进行操作，对节点进行配置。在执行管理员专属的操作之前都需要身份验证，非管理员用户将被禁止和警告。

（2） 加密方式：本系统采用了两种加密模式，一种是数据加密，即在增加、删除、修改和查询数据时，都需要经过加密和解密的过程。而文件加密的模式只需要在启动和关闭时对所有文件进行加/解密即可。

很明显第二种方式是为了运行过程中并不采取任何安全措施以保证性能，也是默认采取的策略。加密具体方法如下：

（1） 生成一个随机数种子；

（2） 根据该种子设定随机偏移量；

（3） 对要加密的数据的每一个字符应用该偏移量得到加密字符。

（4） 根据种子生产随机顺序，将加密字符按该顺序排序，加密结束。

## 4.4 文件组织

对于永久存储模式来说，大部分数据库的操作就是文件的操作。当前，不同数据库提供商所采用的文件组织方式都不相同，例如 MYSQL 数据库的数据文件只有一个，而 ORACLE 的就有很多。本系统所设计和实现的文件组织方式主要是本着分离、直观的思想，将不同的数据结构都保存在不同的文件当中，以便于调试和观测存储结果。当然，这样做的坏处就是数据库的安全性和一致性会稍微受到影响，因为文件是分散存储，必然导致管理上的困难以及文件之间数据一致性维护的问题。接下来就简要介绍数据结构的组织方式。

如表 4-1 所示，永久储存模式下每种数据结构都有对应的维护文件。此处以实例为例，其运行共需要维护 metadata、tenants、databases、seed、logs 几个文件和 tenant、database、graph 三个文件夹。其中，metadata 复杂维护冗余备份的路由信息，tenants 存储用户的名称列表，databases 存储数据库名称列表，seed 存储加密方式的密码种子，logs 存储日志数据。这里采用了一种“文件指针”的存储方式，即一个文件中只储存能够定位另一个文件的信息，而不存储多余的信息。详细的数据操作方式见第五章。

**表 4-1 主要数据结构对应文件列表**

| 数据结构 | 文件列表 |
|---|---|
| 实例（Instance） | metadata、tenants、databases、seed、logs、tenant 目录、database 目录、graph 目录 |
| 数据库（Database） | databases、database 目录、tenant 目录 |
| 图（Graph） | database 目录、graph 目录 |
| 租户（Tenant） | tenants、tenant 目录 |

## 4.5 性能优化

随着云计算和虚拟化技术不断发展，原来的软件产品逐渐服务化。数据库也不例外，虚拟环境下的数据库器具成为以后数据存储服务的提供者。但是由于在这样的环境下资源为了快速响应需求是动态分配的，如何在这样的环境下获得最好的操作性能，也是一个研究的热点问题。这其中，有的研究通过动态分配 CPU 的计算资源来进行系统调优。而有的系统是通过人工智能（机器学习）的算法来进行用户间资源的合理分配。还有的研究是在数据库实例之间进行合理分配。

本系统的设计思路从一开始就围绕着高性能展开。包括数据模型设计中键的引入、架

构设计中的去中心化思想、负载均衡策略、可扩展性、弱一致性和高可用性策略等等。除此之外，在实现过程中，也尽量避免使用可能引起性能问题的解决方式，如尽量少使用标准库数据结构、多使用更接近硬件的 C 函数调用等等。此外，还有一个重要的性能优化手段是结点操作性能优化。

对于永久储存模式，每个图结点的文件储存偏移量都由内存中的哈希表来记录以提高性能。本系统提供了两种哈希表的结构，分别是哈希链表和哈希数组。哈希链表在添加或删除结点操作时性能较高，哈希数组在查询时性能较高，本系统的用户可以根据需要选择。通过这种手段能保证图内结点的匹配算法时间复杂度在 O(1)级别，如此达到高性能的要求。

但是不可否认，对于当前所设计的系统仍然可能存在性能问题，主要有可能发生在文件 I/O 冲突，偶然的单点客户请求阻塞等等。这些问题有待进一步研究解决。

## 4.6 测试基准

由于目前云数据库市场还处于发展上升的阶段,产品种类名目繁多,如何用一个统一的标准去衡量这些产品的好坏就成为一个棘手的问题。通常，性价比是一个不错的指标，但是其中性能的部分又比较不好衡量。因而我们需要一个测试基准(benchmark)来对不同的云数据库产品的性能进行评估。当前已经有几种可用户性能衡量的方法，如 BigTable 采用的性能测试方法[15]就已经可以对不支持结构化查询的系统进行性能评估，Hadoop 则反过来可以对一些支持结构化查询的系统进行性能评估。

但是,这些性能测试方法都没有体现当采用不同的系统实现方法时对系统整体性能的影响。实际上,系统实现方法多种多样,比如存储体系架构、数据模型、一致性和可用性水平等等,都不尽相同。因此,文献[45]对多个系统采用不同的实现方法进行了详尽的性能指标测试,可以作为评估不同系统性能的参考。

限于实验室条件，本文采用了几种常规的性能测试方法进行系统性能测试，包括运行效率、响应时间、故障恢复时间等等，详见第六章。

## 4.7 备份/恢复

数据库备份是一个长期过程，而恢复通常发生在事故后，恢复的步骤和备份相反，所以备份的如何直接影响到能不能恢复。而且恢复时的步骤是否正确也会影响的还原的数据。备份的方式有很多种，按照数据库的大小可以分为四种类型，下面简要介绍一下：

（1） 完全备份：这是最常用的方式，即备份整个数据库，但需要花费更多的时间和空间；

（2） 基于日志备份：事务日志是一个记录自上次备份以后数据库产生变化的文件，所以这种方法只需要很少的时间就可以完成，即写入文件；

（3） 差异备份：也叫增量备份。它使用整个数据库的一种更新包。和日志一样，它也只保存自上次备份以来的数据，这种方式的优点是存储和恢复速度快；

（4） 文件备份：如果数据库非常大，并且一次不能将其备份完，则可以使用文件备份每次备份数据库的一部分。

另外，按照数据库的状态有可分为三种：冷备份，即数据库处于关闭状态时进行备份，优点是能够较好的保证数据库的完整性；热备份，即数据库不关闭时依赖于日志文件进行备份；逻辑备份，使用第三方软件从数据库中抽取数据并将结果写到另外的文件上。

本数据库系统可通过三种方式进行备份和恢复，一种是功能性的热备份恢复，一种是基于文件的手动备份恢复，还有一种是基于日志的备份恢复。最后一种更适用于灾难恢复。以下逐一说明：

（1） 热备份/恢复：首先在接收到用户备份命令之后将停止继续接受客户端请求。然

后开始从顶层向下逐一保存内存中的数据结构或者复制已有文件到一个以时间和数据库名命名的文件位置下。全部保存工作完成后重新接收客户端请求。如果过程中出现任何问题，如数据不全或不一致等等，则清空已保存数据。恢复时则与备份过程相逆，根据数据库名和指定时间来从对应文件位置逐一读取所有数据。这种方式比较简单，容易操作，对用户比较友好。可用于日常功能性备份；

（2） 冷备份/恢复：这是一种冷备份的方法。通常在某一节点无法正常工作的时候用户手动执行。备份的过程就是直接将数据库所在文件位置（wanData 目录下）的所有文件拷贝到另一个数据节点，然后将该文件覆盖该新节点上的实例文件重新启动即可；

（3） 基于日志的备份/恢复：严格来说，这种方式没有备份过程，因为日志是每一次操作完成后产生的，也就是所谓的实时备份。所以在节点发生意外故障以后，只要在重启后利用故障时间以前的日志进行恢复即可。

## 4.8 数据迁移

云数据库虽然已经成为了各个机构数据存储的重要举措和发展方向。但是我们如何把数据从单一阵列式的存储框架中转移到虚拟、动态的云环境下还面临着很多问题。数据迁移可以将历史数据进行清洗、转换，并装载到新系统。这对云计算和云存储发展和推动都有着重要的意义。下面我们就对数据迁移进行探讨。

数据迁移有很多种理解方式。如不同服务器之间的迁移，不同数据库之间的迁移，不同数据库提供商之间的迁移等等。这里将逐个分析这些迁移的定义和方法。

（1） 服务器间迁移：这是最容易理解的数据迁移，当前使用服务器因某些原因不再使用的时候，就需要将该服务器中的数据迁移到另一个服务器中去。而这种迁移的解决方法也很简单，就是将数据备份好（如 Mysql 的 dump 命令，产生一个文件）到另一个服务器上进行恢复就行了；

（2） 数据库之间的迁移：这是在同一数据库内的数据迁移。对于传统数据库，这种类型的迁移很容易实现，再创建一个相同模式的数据库，然后不断读取插入新数据库就可以了。但是如果放到分布式的条件下，就会增加很多难度。因为这两个数据库可能不在一个节点上。这就需要主服务器来进行协调操作。而对于本系统这种去中心化的情况，则不需要额外的操作，因为各节点之间可以直接复制数据库副本；

（3） 不同供应商数据库之间的迁移：这种类型是本文要重点讨论的类型。对于传统的关系型数据库来说，即使是不同的数据供应商之间的数据迁移也可以通过导出 SQL 语句这种两者都有的共同标准数据形式来简单实现（当然不同供应商之间不同的高级特性另当别论）。但是在云环境下，没有任何标准是普遍适用的。云数据库产品的种类、模型、架构各不相同。要在不同供应商的云数据库之间传输数据就显得格外困难。

所以，一种通用的数据模型就是云数据迁移的关键。根据第二章中图数据模型的通用性介绍，通过对其它模式的模仿，图模型理论上可以接收来自其它任意数据库的数据，从而进行数据迁移，只是数据库产品相关的特性则各不相同，需根据具体情况分析。当然，这里只是狭义的迁移了数据（即一条一条的记录），而实际上数据库的内容远不止这些最基本的记录。以 Mysql 为例，它在保存拥有一张表的一个数据库时，同时还保存了可以对表数据进行的分析处理的表元数据、存储过程和函数等等内置结构。这些结构的迁移就没有那么通用的方法了。就目前的数据迁移研究趋势来说，都是尽量放松模式要求，放弃过于深化的内置结构工具。

## 4.9 信息检索

信息检索（Information Retrieval）是指根据用户需要查找和分析信息的技术。对于传统

的数据库来说，信息检索没有任何新意。只是通过结构化查询语言（通常是 SQL）构造所要搜索的信息，然后数据库查询引擎进行解析和优化后进行满足条件的记录匹配，最后返回结果。而在云环境下，特别是 NoSQL 的出现以后，信息检索就变得活力无限。挣脱了关系模式的缰绳，数据库信息检索不再是枯燥无味的机器翻译，而是充满各种可能性。这其中又以图模型等半结构化的模式为代表，以下举例说明这种查询方法的优势。

关系型数据库实际上数据之间没有关系，至少在同一种模式（同一张表下）没有。而图则不然，由于数据之间的关联性，图数据的查询不需要在构造查询语句时就构思如何精确定位，而是可以通过和目标数据相邻的已知数据来进行查找，这是关系数据库所不具备的特性。例如，假设有一个社交网络图的数据，想要找到其中某个人（假设叫“Jensen”）的所有朋友，则只需遍历一遍该节点的关系便足以找出他的朋友。不仅如此，基于图模型的数据库还可以完成非常复杂的遍历描述和过滤器。如可以指定遍历深度和关系类型等等可以实现诸如“朋友的朋友”“好朋友”“朋友的爱好”等等。

但是规范的针对图模型进行查询的语言还没有形成统一的标准。仅在几个特定的研究领域，如在语义网/RDF 领域就有 SPARQL 这种受 SQL 启发的查询语言，专门用来描述本体图等。除了这个，其它种类的查询语言大都与 JSON 类似，但是这些语言只适用于自定义的数据模型。至于针对各种图数据模型（如 RDF）的更复杂有趣的查询也不是没有。如 Gremlin 就就能支持通过 XPath 来表示的不同图上的路径描述。

当然，这只是图形模式所独有的特性。像关系模型或者键/值模型一样的精确查找也是可以实现的, 本系统中所采用的改进图模型就可以通过键来直接定位数据结点。同时，也实现了关系遍历的一些基本查询方法和深度控制器。

## 4.10 本章小结

本章介绍了几种云数据库的高级功能，针对每一种功能首先总结当前研究现状及产品概况等内容，然后提出基于本数据库的解决方案或思路。其中，共有包括事务一致性、高可用性、数据安全、性能优化、测试基准、备份/恢复、数据迁移和信息检索在内的九种功能。

# 第五章 实现方法与接口说明

本章将介绍基于本项目所设计数据库的具体实现方法，以验证设计方法的可行性。首先对数据库的功能模块进行分析。

## 5.1 功能模块

本数据库包括的主要功能模块有以下几个方面：

（1） 权限验证：用户在用客户端连接服务器之一的时候，需要首先输入该用户的密码，经过验证后才能进行操作。另外，每次执行用户命令之前也会进行数据库操作权限验证，查看该用户是否具有当前数据库的操作权限；

（2） 数据库的增删改查：具有当前数据库操作权限的租户可以在数据库层面进行相应的增加、删除、修改和查看操作，查看操作会显示数据库大小，拥有的图列表等等相关信息；

（3） 图的增删改查：具有所操作图的数据库操作权限的用户可以对该图进行增加、删除、修改和查看操作。查看操作会显示图大小、所有结点信息和关系信息等等；

（4） 节点的增删改查：具有所操作图的数据库操作权限的用户可以对该图中的结点和结点关系进行增加、删除、修改和查看操作；

（5） 租户的增删改查：具有管理员权限的租户可以对其它租户进行增加、删除、修改和查看操作，非管理员租户只可以对自己的信息进行修改操作；

（6） 查看实例数据大小：具有管理员权限的租户可以查看实例数据大小信息；

（7） 查看版本信息：可以查看数据库版本和修订日志等信息；

（8） 查看本机信息：可以查看本机型号和配置等信息；

（9） 切换租户：任何人都可以注销并切换用户重新登录；

（10）切换数据库：具有要切换数据库权限的租户可以切换到该数据库；

（11）存储模式：在实例启动时可以配置其存储模式，有内存和永久两种模式；

（12）永久存储模式：在实例启动时可以配置其永久存储模式，有哈希链表和哈希数组两种；

（13）信息加密模式：在实例启动时可以配置其加密模式，有不加密、数据加密和文件加密三种模式；

（14）备份/恢复：具有当前数据库权限的租户可以进行备份和恢复操作；

（15）高级检索：具有当前数据库权限的租户可以进行数据检索操作；

（16）节点自动发现：当有新节点启动能够自动发现；

（17）负载均衡：每个节点之间的数据量不应太大。

## 5.2 用户用例

如图 5-1 所示，普通租户可以进行登录、设置存储模式、查看实例大小、修改个人信息、切换数据库、数据库操作、图操作、结点操作、查看版本信息、查看本机信息、切换用户、备份恢复和高级检索的功能，而管理员租户不仅可以执行普通租户能够执行的所有操作，同时还可以对租户进行管理和关闭实例。

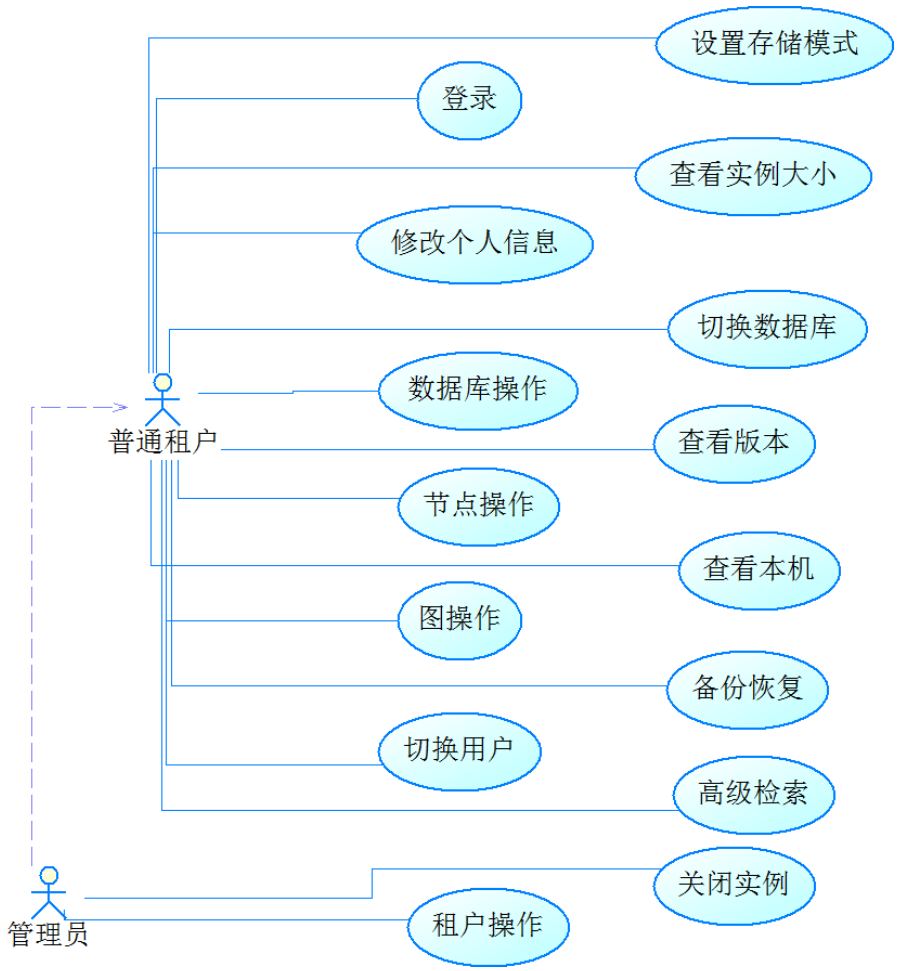



**图 5-1　wanDB 的体系架构(4 节点,数据库级分区)**

## 5.3 数据结构

以下是几个重要数据结构的属性和方法：

**表5-1 主要数据结构**

| 结构 | 属性 | 方法 |
|---|---|---|
| 实例（Instance） | ip_address 、 port 、 router 、 metadata 、 databaseList 、 currentDatabase 、 databaseAmount 、 maxDatabaseAmount 、 tenantList 、 currentTenant 、 tenantAmount 、 maxTenantAmount 、 queryResult 、 isPersistent 、 switchOn、granularity、file、log、 crypto | Instance、~Instance、show、execute、 addTenant 、 getTenantByName 、 getTenant、 delTenant 、 addDatabase 、 showTenants、getDatabaseByTitle、 getDatabase 、 getIndex 、 delDatabase 、 getMetadata 、 getRouter、getSize getTenantAmount 、 getDatabaseAmount 、 getQueryResult、setCurrentTenant、 setCurrentDatabase、run |

续表 5-1

| 结构 | 属性 | 方法 |
| --- | --- | --- |
| 数据库（Database） | graphList、graphAmount、maxGraphAmount、title | getIndex、Database、~Database、copy(Database* db)、showGraphs、getGraph、setTitle、getTitle、getSize、getGraphAmount、getGraphList、addGraph、deleteGraph、clear |
| 图（Graph） | nodeList、title、nodeAmount、maxNodeAmount | Graph、~Graph、show、clear、getNodeList、insert、insertToFile、deleteData、updateData、updateDataToFile、getSize、getNodeData、getNodeAmount、getTitle、getIndex、getGraphData、setTitle、showNode、deleteNode、updateNode、addNode、createConnection、destroyConnection、hasConnectionBetween |
| 图结点（GraphNode） | Key、data、adjacentNodeAmount、maxAdjacentNodeAmount、adjacentNodeList | GraphNode、~GraphNode、Show、setKey、getKey、setData、getData、getInfo、getAdjacentNodeAmount、addAdjacentNode、deleteAdjacentNode、hasAdjacentNode、getAdjacentNodeList |
| 租户（Tenant） | Username、password、databases、databaseAmount、maxDatabaseAmount、isAdmin、account | Tenant、~Tenant、Show、getSize、copy、setUsername、getUsername、updatePassword、checkPassword、getPassword、grantDatabase、revokeDatabase、hasPermissionOn、addAccount、setAccount、 upLevel、downLevel、clear、hasPermissionOnAll、getAccount、 getDatabaseAmount、getDatabaseTitle |

## 5.4 算法与数据操作

本节下将逐一阐述每个功能涉及的算法和数据操作：

（1） 权限验证：根据当前系统用户名称和所输入密码与所存储的对应数据进行对比，如果匹配，则设置当前用户指针指向该租户，并返回成功信息，客户端进入 shell 模式；

（2） 数据库的增删改查：增加、删除数据库根据当前模式（存储、分区）进行对应数据结构（实例、租户或其文件）的数据库名称的添加，如内存模式下，则添加的 Instance 实例的数据库列表，同时当前租户数据库指针列表中新增一个指针指向该新增数据库；文件模式下，则修改 databases 文件和当前租户文件添加新增数据库名称。修改操作则直接在该数据库上进行操作，不影响其它数据结构。查看数据库信息则返回所包含的所有图和大小信息；

（3） 图的增删改查：与数据库操作类似，增加、删除图根据当前模式（存储、分区）进行对应数据库的图名称的添加，如内存模式下，则添加图名称到当前数据库的图列表。这里需要注意的是，添加图时可以附加初始化数据；文件模式下则在数据库信息文件中加入该图名称。修改操作则直接在该图上进行操作，不影响其它数据结构。查看图信息则返回所包含的所有结点和结点关系的序列化信息；

（4） 节点的增删改查：与数据库和图操作类似，增加、删除结点及关系根据当前模式（存储、分区）进行对应节点或关系的添加或删除，不同的是，在内存模式下是在图实例中进行添加或删除结点操作，而文件模式下只对图文件进行修改，并没有结点文件单独存在。修改操作也是直接在该图上进行操作，不影响其它数据结构。查看结点信息则返回所包含的数据和所有指向其它结点的关系的序列化信息；

（5） 租户的增删改查：与上述数据操作类似，只是在执行操作之前需要验证用户的权限是否是管理员，并且查看操作返回信息中除了该租户基本信息外还包括拥有权限的数据库列表；

（6） 查看实例数据大小：通过计算字节数返回实例大小信息；

（7） 查看版本信息：返回发布时指定的版本号；

（8） 查看本机信息：通过系统调用返回系统信息；

（9） 切换租户：客户端断开连接，并以切换用户名重新发起连接；

（10）切换数据库：先进行权限判断，然后将当前数据库指针指向所要切换数据库；

（11）存储模式：通过实例启动时指定参数来配置；

（12）永久存储模式：通过实例启动时指定参数来配置；

（12）分区模式：通过实例启动时指定参数来配置；

（13）信息加密模式：通过实例启动时指定参数来配置；

（14）备份/恢复：停止接收新操作，不论当前模式，将所有数据信息按照永久存储的模式保存到备份文件夹中，恢复则从制定文件夹中读取数据；

（15）高级检索:；

（16）节点自动发现：通过线程实现让每个节点定时广播自身数据大小，同时接受广播信息，更新当前可用的节点信息；

（17）负载均衡：租户每次新增数据库或表时都需要选取当前数据量最小的节点，然后将该数据库或表建立在该节点下。

## 5.5 用户界面和命令列表

### 5.5.1 服务端运行截图

如图 5-2 所示，服务端启动时的初始化及发现节点的信息会打印在屏幕上。

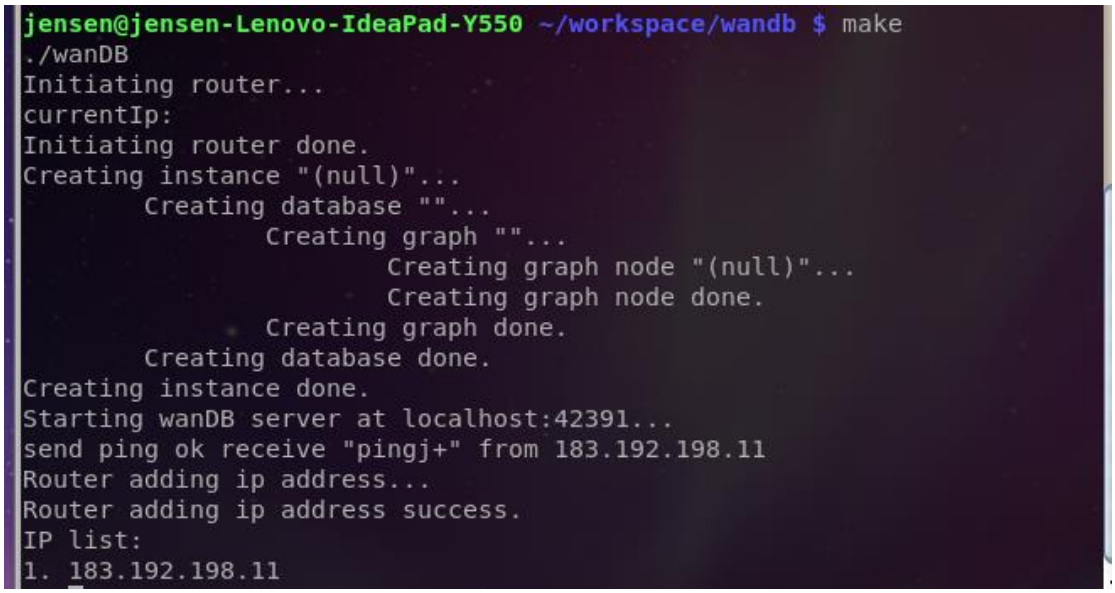



**图 5-2　服务端运行截图**

### 5.5.2 客户端运行截图

如图 5-3 所示，客户端在登录后输入“help”命令后进入到这个界面。

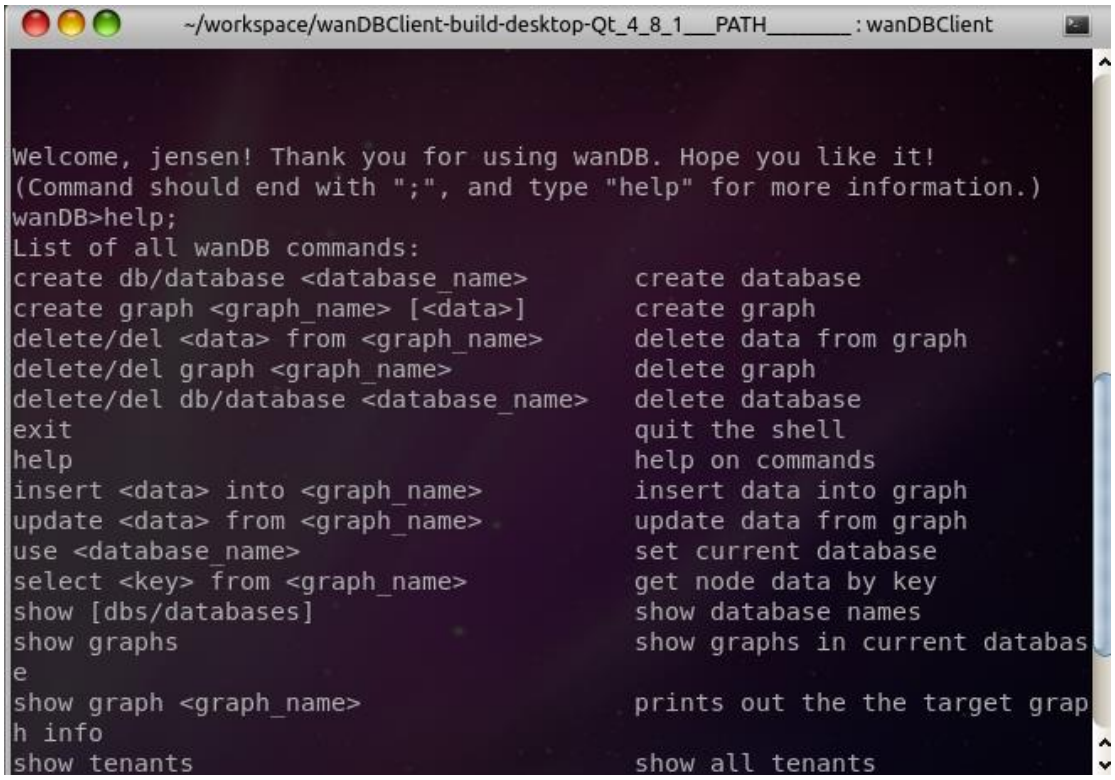



**图 5-3　客户端运行截图**

### 5.5.3 命令列表

表5-2中列举了所有的已实现功能命令及其使用说明。

表5-1 已实现命令列表

| 命令格式 | 使用说明 |
|---|---|
| create db/database <database_name> | 创建新数据库，需要附加数据库名 |
| create graph <graph_name> [<data>] | 创建新图，需要附加名称，初始数据可选 |
| delete/del <data> from <graph_name> | 从指定名称的图中删除JSON格式的数据 |
| delete/del graph <graph_name> | 删除指定名称的图 |
| delete/del db/database <database_name> | 删除指定名称的数据库 |
| exit | 退出 |
| help | 帮助 |
| insert <data> into <graph_name> | 向指定图中插入JSON格式的数据 |
| update <data> from <graph_name> | 从指定图中更新JSON的数据 |
| use <database_name> | 切换到指定数据库 |
| select <key> from <graph_name> | 查询指定结点的数据 |
| show [dbs/databases] | 查询数据库的名称、大小和所包含的图 |
| show graphs [from <number> to <number>] | 查询当前数据库中指定序号的图名称 |
| show graph <graph_name> | 查询指定名称的图中的数据（JSON格式） |
| show tenants [from <number> to <number>] | 查询指定序号的租户名称 |
| show tenant <tenant_name> | 查询指定名称的租户的信息 |
| show profile | 查询当前实例所在系统概况 |
| show logs [from <timestamp> to <timestamp>] | 查询指定时间段内的命令日志，默认最近5条 |
| show size | 查询当前实例数据量大小 |
| dump <database_name> [to <filename>] | 备份指定名称的数据库，可选指定文件，默认为数据库名加dump |
| restore <database_name> [from <filename>] | 恢复指定名称的数据库，可选指定文件，默认为数据库名加dump |
| flashback to <timestamp> | 根据日志恢复到指定时间点 |
| search <key> by <deep_degree> | 查找指定结点周围指定深度的结点信息 |

## 5.6 开发环境

整个云数据库管理系统采用 Linux 环境下的 C++语言编写，用 Makefile 工具进行自动编译。

## 5.7 本章小结

本章介绍了数据库具体的实现方法、功能模块、接口说明、数据操作、用户界面、命令列表和开发环境，详细的说明了整个数据库从设计到实现的过程。

# 第六章 测试与验证

## 6.1 测试环境

实验使用 1 台机器上的 3 台虚拟机作为测试环境，其中宿主机器配置两个 64 位 CPU，配置如表 6-1 所示。各虚拟机各配置一个，配置如表 6-2。

表6-1 host主机配置

| 属性 | 值 |
|---|---|
| CPU型号 | P7450 |
| 核心数量 | 2 |
| 主频 | 2130MHz |
| 主存 | 4GB |

宿主机安装有 linux 3.8 内核系统。

表6-2 虚拟主机配置

| 属性 | 值 |
|---|---|
| 核心数量 | 1 |
| 主频 | 1050MHz |
| 主存 | 1GB |

虚拟机安装有 linux 3.8 内核系统。在实验中，通过 VMWARE 对集群中的 3 台机器进行控制。集群之间的节点通过局域网进行连接，虚拟机与宿主机之间通过 NAT 模式进行连接。各节点属于同等关系。

## 6.2 常规性能测试

### 6.2.1 内存模式

首先，我们先测试了内存模式下读操作的性能。测试时，我们模拟了同一用户对任选（1 号）节点进行读操作请求(以查询图结点数据命令作为基准，即“select “i” form graph”，i=1,2,…,n)，观测数据库执行完这些请求所消耗的时间长度。

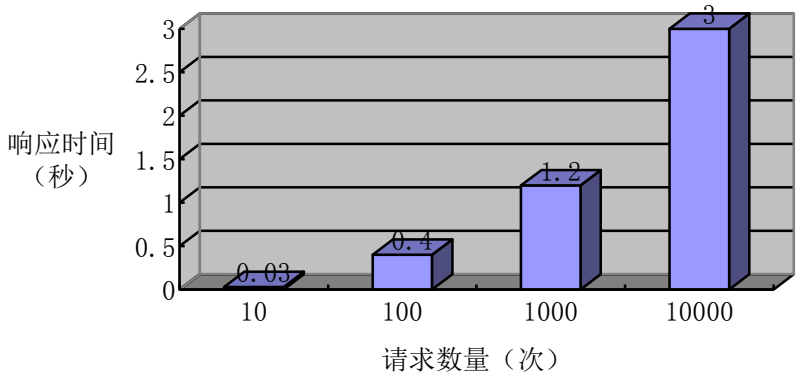



**图 6-1 内存模式读（查询）请求响应时间统计图**

图 6-1 反映了本数据库系统不同请求数量下的响应时间。其中横坐标为模拟用户的请求

数量，数量从 10 个请求一直变化到 10000 个；纵坐标反映了响应读操作的时间，以秒为单位。从该图中可以看到，响应读操作的时间随着用户请求个数的增加而不断增大。从测试结果来看，基本上请求数每增加 3000 个，响应时间就增加 1 秒。在测试的过程中，我们还可以观测平均一个读操作需要的响应时间，结果如图 6-2 所示。

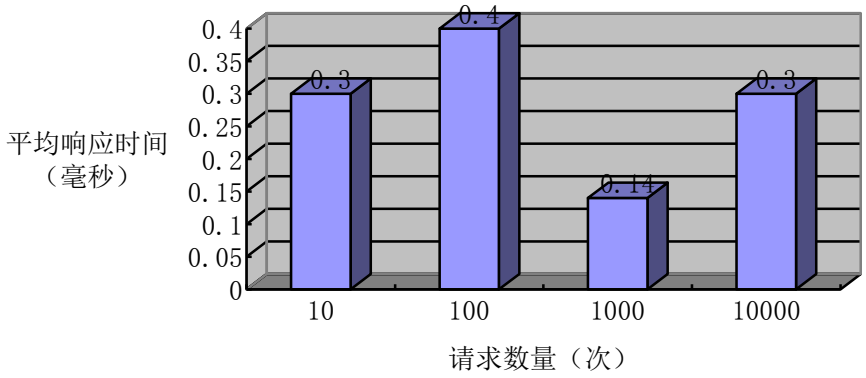



**图 6-2　内存模式读（查询）请求平均响应时间统计图**

从结果上来看，不论用户的请求数量，读操作的平均响应时间能够稳定在 0.3ms 左右。响应时间和请求的数量并没有太大的关系。读操作的平均响应时间较快，能够很好的满足读操作的需求。

接下来，我们按照同样的方法测试后台写操作（包括增加、删除和修改等，以插入图结点命令为基准，即“insert {‘i’:’i’} into graph”，i=1,2,…,n）的性能。图 6-3 反映了响应写操作请求的时间与请求数量的关系。

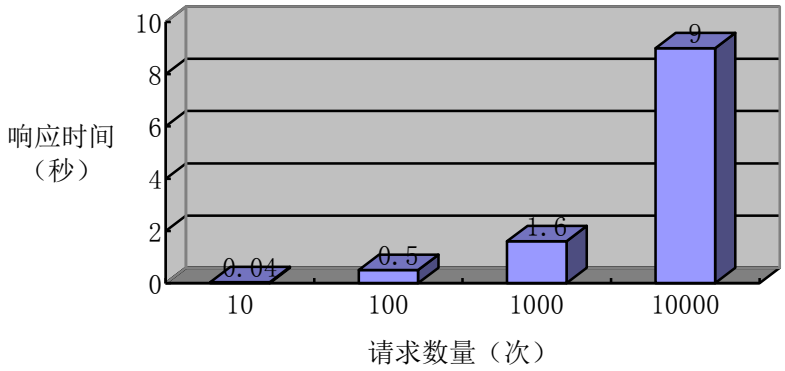



**图 6-3　内存模式写（修改）请求响应时间统计图**

与前面分析响应读操作的时间一样，图中横坐标为用户的请求数量，数量从 10 次到 10000 次；纵坐标反映了系统响应写操作的时间。从该图中可以看到，系统响应写操作请求的时间随着用户请求个数的增加而不断增大。从测试结果来看，基本上用户中每增加 1000 个请求，响应写操作的时间可以上升大约 1 秒。从结果中我们也可以看出，后台写操作的效率要低于读操作的效率。这个与数据库系统架构的设计是相关的。写操作需要进行内存修改操作，而读操作只有打印操作。实验结果和预期是相符的，证明了该框架对读操作的支持更好。实际上，对于一个数据库系统来说，读操作是用户使用频率最高的操作，读操作占了所有操作中的大多数。接下来，我们分析写操作请求的平均响应时间。

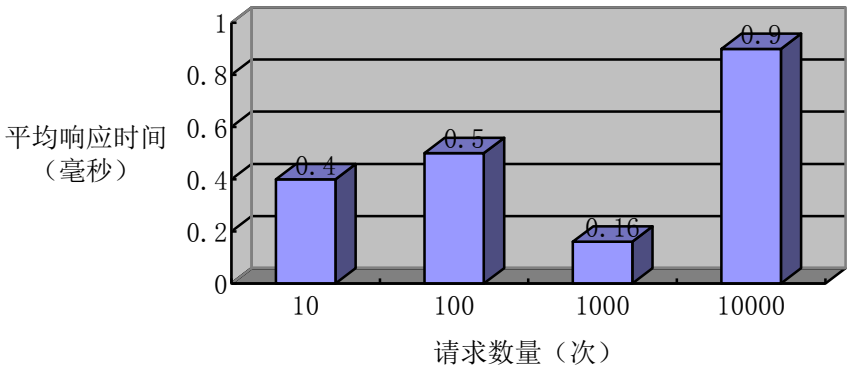



**图 6-4　内存模式写（修改）请求平均响应时间统计图**

从图 6-4 中可以发现，写操作请求的平均响应时间大概为 0.5ms 左右。平均响应时间和请求数量的关系不大。这个趋势基本上和读操作请求的平均响应时间保持一致。接下来，清空数据库中数据，并配置到永久（文件）存储模式重新进行读写测试。

### 6.2.2 文件模式

首先，我们先测试文件模式下读操作的性能。同内存模式一样模拟同一用户进行不同数量的读操作请求，记录数据库响应时间。

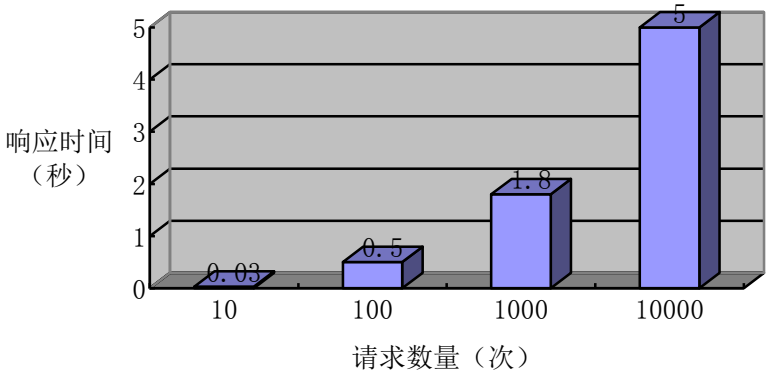



**图 6-5　文件模式读（查询）请求响应时间统计图**

从图 6-5 中可以看到，基本的变化趋势与内存模式一致，每增长 3000 个请求响应时间增加 1 秒。但是，就绝对值来说，文件模式的读操作响应时间要比内存模式慢一些。这也是文件需要从磁盘读取数据的反映。磁盘的读取速度要比内存慢很多。接着，平均一个读操作需要的响应时间如图 6-6 所示。

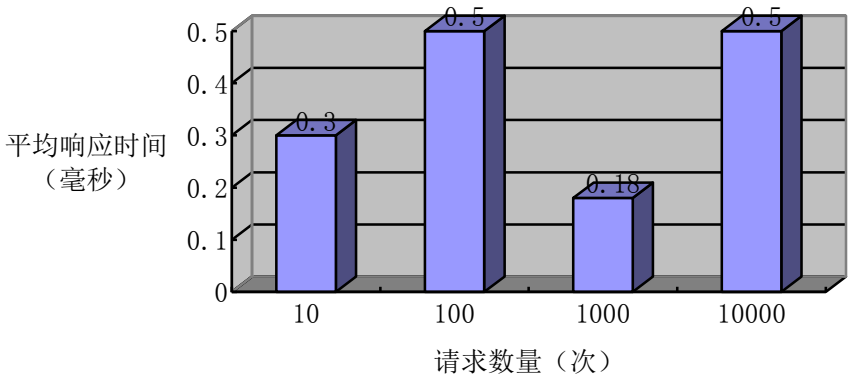



**图 6-6　文件模式读（查询）请求平均响应时间统计图**

不论用户的请求数量，文件模式读操作的平均响应时间能够稳定在 0.4ms 左右，比内存模式要慢差不多 0.1ms。接下来，我们按照同样的方法测试文件模式写操作的性能。图 6-7 显示了结果。

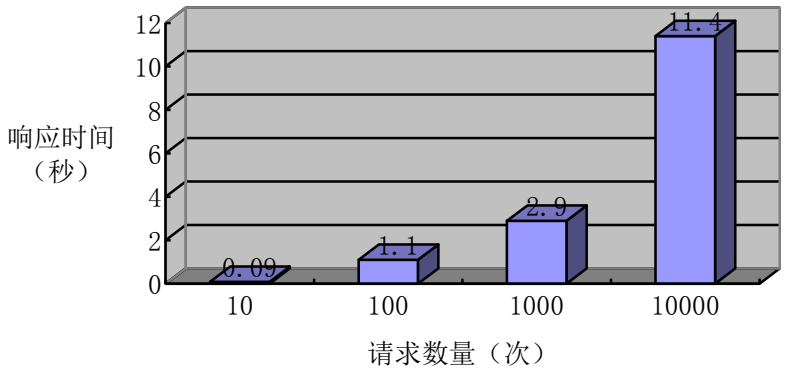



**图 6-7　文件模式写（修改）请求响应时间统计图**

由图 6-7 可知变化趋势与前面读操作请求响应时间趋势保持一致。从结果中我们也可以看出，后台写操作的效率要低于读操作的效率。这个与数据库系统架构的设计也是相关的，因为写操作需要不止一步的磁盘修改（特别是删除）。实验结果符合预期。接下来，我们分析写操作请求的平均响应时间。

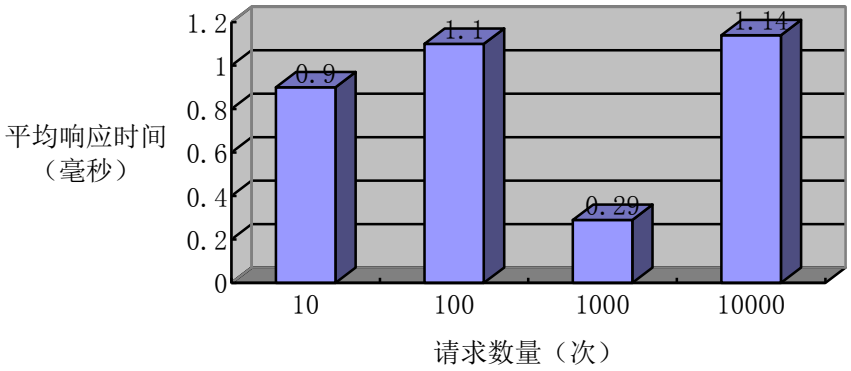



**图 6-8　文件模式写（修改）请求平均响应时间统计图**

从图 6-8 中可以发现，文件模式下写操作请求的平均响应时间大概为 0.7ms 左右，比文件模式的读操作慢了 0.3ms，比内存模式的写操作慢了近 0.2ms。这跟预期相符，如上一节所说，用户的查询操作占大部分的日常操作，这样的测试结果符合使用需求。然后，作为对照组，我们要测试 Mysql 数据库在同样测试环境下的性能指数。

### 6.2.3 参照测试

为了进行横向对比，我们将测试 MYSQL 数据库在当前环境下读操作的性能。为了控制变量，我们建立了同样数据量大小的数据库。然后以查询语句(“select data from graph where key=i”,i=1,2,…,n)作为测试基准来观测数据库的响应能力。

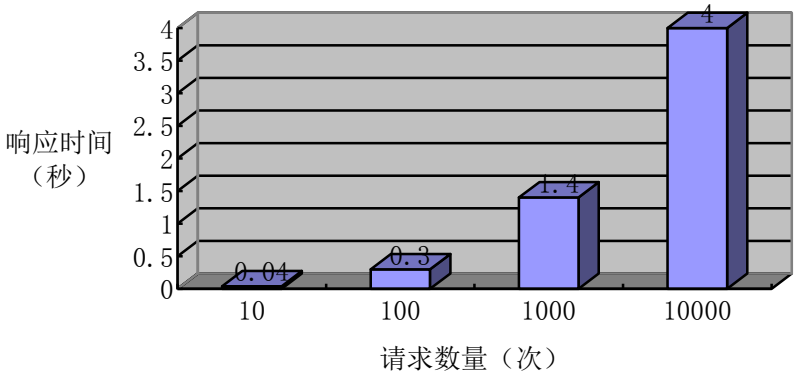



**图 6-9　MYSQL 读（查看）请求响应时间统计图**

图 6-9 反映了 MYSQL 的查询请求响应时间情况，从测试结果来看，基本上响应时间与本系统的文件模式类似，稍稍优于本系统，但是效率低于内存模式。我们再观测平均一个读操作需要的响应时间，结果如图 6-10 所示。

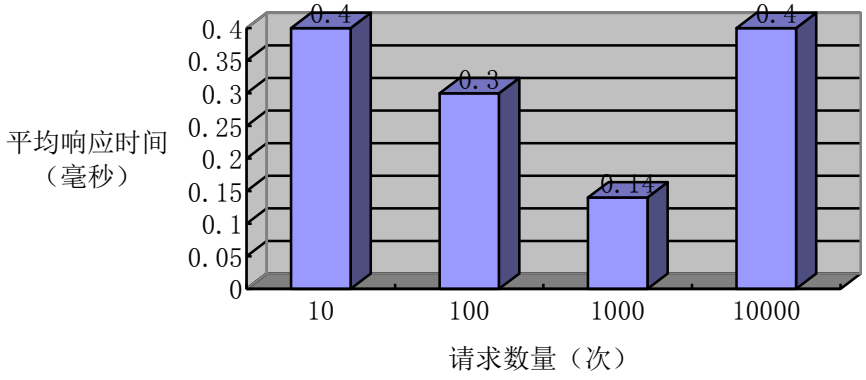



**图 6-10　MYSQL 读（查看）请求平均响应时间统计图**

从结果上来看，读操作的平均响应时间大概稳定在 0.3ms 左右。

接下来，我们按照同样的方法测试 MYSQL 写操作（增加、删除和修改）的性能。图 6-11 反映了响应写操作请求的时间与请求数量的关系。

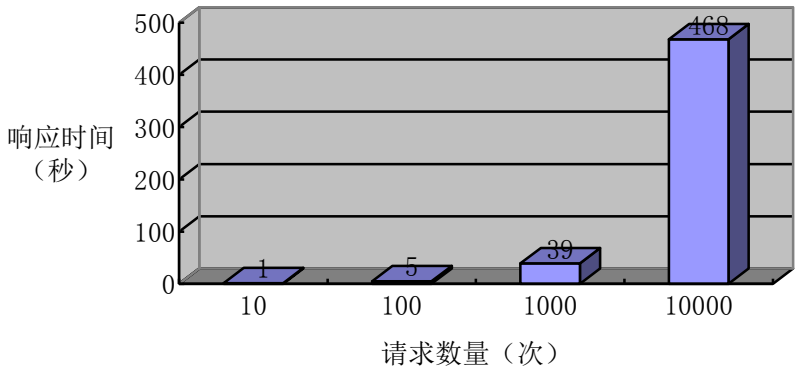



**图 6-11　MYSQL 写（查看）请求响应时间统计图**

从该图中可以看到，系统响应写操作请求的时间随着用户请求个数的增加而急剧增大。这个趋势与前面读操作请求响应时间趋势非常不同。从测试结果来看，MYSQL 写操作的效率要远远低于读操作的效率。这个原因可能与其内部的复杂储存模式有关，我们不得而知。接下来，我们分析写操作请求的平均响应时间。

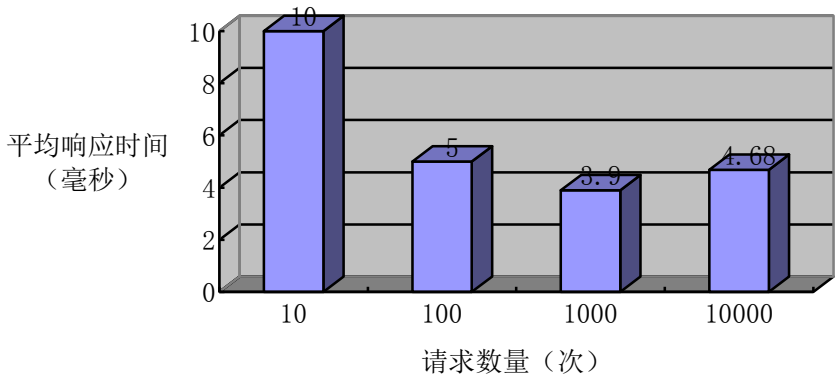
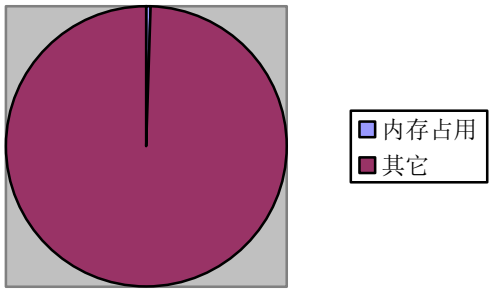



**图 6-12 MYSQL 写（查看）请求平均响应时间统计图**

从图 6-4 中可以发现，MYSQL 写操作请求的平均响应时间大概为 6ms 左右。平均响应时间和请求数量的关系不大。很明显，本系统在写操作上对比 MYSQL 有较大性能优势。接下来，我们也看一下本数据库和 MYSQL 在运行过程中消耗的本地资源。

### 6.2.4 运行时效率

如图 6-13 所示，在数据量为 1000 个结点时，本系统大概占用了 1 号虚拟机当中 4.5M 内存和平均 7.5%CPU。



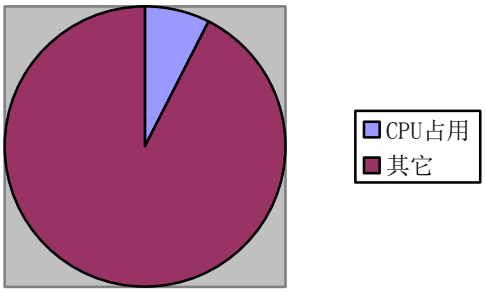




**图 6-13 wanDB 运行效率图**

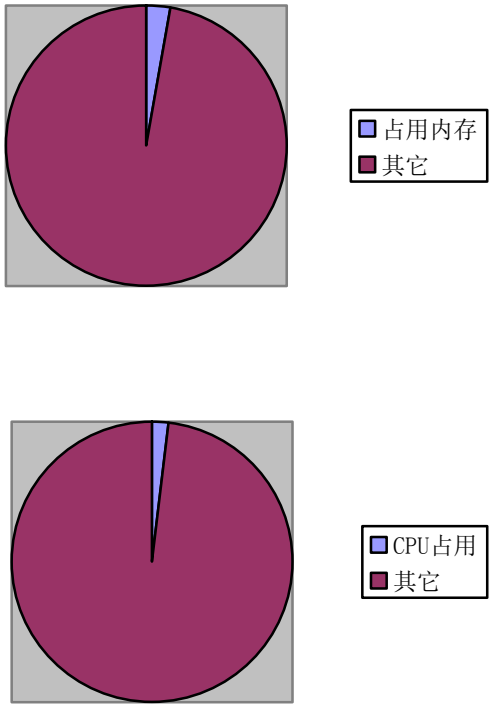



**图 6-14 MYSQL 运行效率图**

而如图 6-14 所示，MYSQL 服务器运行时占用 CPU 计算资源约为 2%，内存资源 24.6M。从运行效率上来看，本系统内存占用较少，而计算资源占用较高，这可能与 MYSQL 内部的复杂存储结构有关。

## 6.3 特殊情况测试

接下来我们继续分析当集群中出现死亡节点或者有节点加入集群时候后台性能的变化情况。

### 6.3.1 节点失效

假设这个节点数量为 3 的集群中出现了一个节点失效。在这个过程中我们依旧不断每隔 1 秒发送一条读或写数据请求，并记录响应的时间。测试结果如图 6-15、6-16 所示。

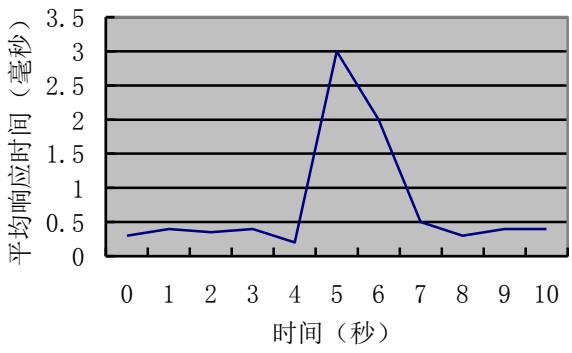



**图 6-15 节点失效时读操作平均响应时间**

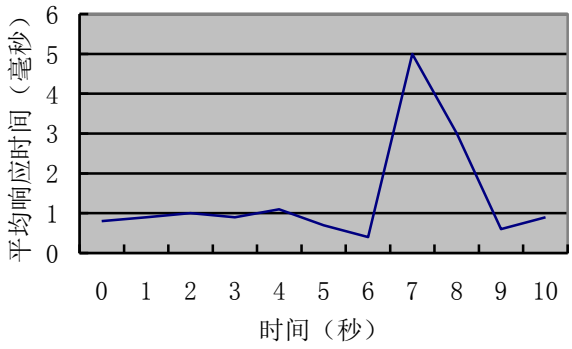



**图 6-16 节点失效时写操作平均响应时间**

图6-15和图6-16分别反映了在节点死亡的整个过程中，系统的读操作响应时间以及写操作响应时间的变化。一开始，所有的节点都运行正常，读写操作的响应时间维持在稳定水平；当节点死亡的时候，读操作和写操作的响应时间会立刻升高；而在过了大约3s之后又开始恢复。 当节点死亡的时候，集群内部节点并不知道某个节点已经死亡，所有在该节点上的读写操作仍将继续进行。但是这时候，所有的读写操作最终会出现超时异常，在该节点上的读写失败，原读写操作将转到其他节点上继续进行。这个过程会耗费大量的时间和资源，这就是造成节点刚死亡时候读写响应时间升高的原因。当所有节点都获得此节点失效的信息以后，所有在死亡节点上的读写操作将被转给其他节点进行操作。读写操作不会再出现超时的异常。这样读写的响应时间又开始回升。这与预期的结果一致。

### 6.3.2 节点加入

与上一节同样的初始状态下，我们再启动一个新的节点。此时的系统性能测试结果如图6-17、6-18 所示。

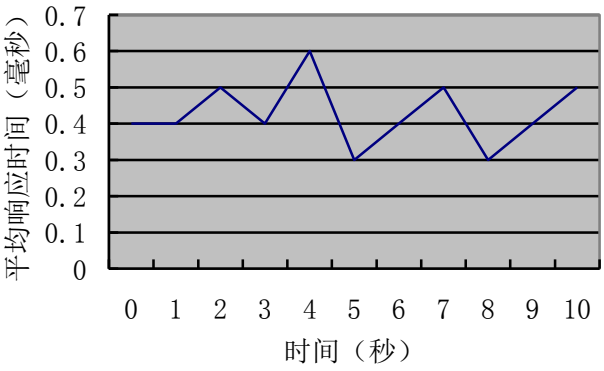



**图 6-17 节点加入时读操作平均响应时间**

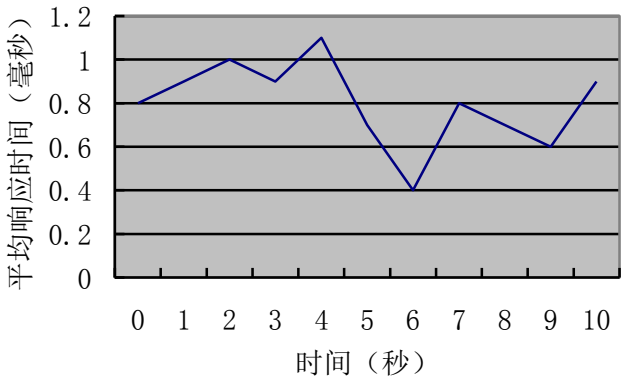



**图 6-18 节点加入时读操作平均响应时间**

从这组数据中可以看出，当有节点加入的时候，系统的性能基本不会受到影响。这是因为去中心化的系统架构，在处理新节点加入的请求的时候，需要将其他节点上的数量拷贝到新加入的节点上，这个拷贝过程并不影响当前正在进行的读写操作。但是当节点加入完成之后，整个系统的数据存储容量会上升。

## 6.4 本章小结

本章对整个数据库系统进行了详细的测试，首先着重分析了系统的性能，从实验结果来看，数据库的性能良好，能够满足开发应用的需要。接下来，本章分析了性能与 Mysql 为例的主流数据库的性能对比，最后通过实际测试分析了本数据库的高可用性。总体来说，本系统满足了预期的设计要求。

# 第七章 研究结论与讨论

## 7.1 本文总结

本课题主要研究并开发一个云数据库系统原型。本系统原型结合了NoSQL数据模型中的图模型、去中心化分布式存储等现有存储技术，并做了适当改进，使得本系统能够提供可扩展性、高性能和负载均衡等云数据库的基本特性。同时，在大量调研和实证的基础上又逐个研究了高级特性的研究现状和本系统能采用的实现方式，包括事务一致性、高可用性、数据安全、文件组织、性能优化、测试基准、备份/恢复、数据迁移、信息检索。其中，高可用性、数据安全、文件组织、备份/恢复和信息检索均已实现。最后通过完整一致的性能测试证实了本系统设计的正确性。具体工作总结如下：

(1) 研究并分析了现有的数据模型。重点研究并分析了键/值模型、关系模型和图模型。然后改进了图模型并应用在本框架中；

(2) 设计并实现了本云数据库的分布式底层架构。在服务端采用了点对点的存储模式，降低了单点性能瓶颈和故障风险，并能够自动发现和删除其它节点；

(3) 为了实现高性能的初衷，弱化了一致性要求，强调最终一致性，即容许短暂时间的数据不一致；

(4) 通过采用日志传送和冗余复制的方法实现了高可用性，能够在节点故障时保持一定的整体可用性；

(5) 提出了一种新的数据加密方式来加密数据，同时设计了两种加密模式，数据加密和文件加密；

(6) 提出了一种“文件指针”的底层存储文件组织方式，提供了底层数据直观的可读性；

(7) 通过节点数据量负载均衡、开发时性能调优等手段实现性能优化；

(8) 设计并实现了三种方式的备份/恢复功能，包括热备份/恢复、冷备份/恢复和基于日志的备份/恢复；

(9) 提出了利用几种本数据库进行信息检索的解决方案，实现了其中一种，即按深度查找指定节点的临近目标节点；

(10) 通过对运行效率、响应时间、扩展影响和故障恢复能力的测试实验结果表明，本系统实现了高性能、可扩展和高可用的目标。

总之，本系统的设计与实现基本满足了预期目标。实验结果也证明了本系统的可行性。

## 7.2 局限与展望

本课题提出了一种高性能云数据库构建模式。但是由于时间仓促和实验条件的限制，目前研究还存在如下问题：

(1) 本数据库的数据模型采用了图模型。该模型尚处于实验研究阶段，并未在实际市场中广泛使用。可靠性尚未验证，可能存在潜在的缺陷；

(2) 本系统并不保证事务的一致性，仅仅保证最终一致性。所以对数据一致性要求比较强的应用系统不太适用，如银行、证券交易系统；

(3) 由于实验条件所限，系统的集群测试并没有运行在大规模集群上。由于缺乏必要设备，无法继续测试。同时，对许多高级功能的测试也有限。在以后的工作中，我们需要继续测试框架运行在大规模集群上的性能变化以及其它特性的实用性，以验证框架的可行性，并且不断改善框架。

(4) 由于时间关系，暂没有开发具有图形化可视界面的客户端，对于图模型来说，直观的图形化界面可能更有使用价值，未来需要尽力实现。

## 7.3 本章小结

本章主要对整个论文进行了总结。本章首先系统性的总结了本课题的研究内容，同时对研究结果进行了总结。本章对接下来本课题中的缺陷进行了分析，提出了未来研究工作的方向。

# 参考文献

[1] Chen K, Zheng WM. Cloud Computing: System instances and current research[J]. Journal of Software, 2009,20(5):1337−1348

[2] Dash D, Kantere V, Ailamaki A. An economic model for self-tuned cloud caching[C]. The 25th Int'l Conf. on Data Engineering (ICDE 2009). New York: IEEE Computer Society Press, 2009. 1687−1693.

[3] Feng DG, Zhang M, Zhang Y, Xu Z. Study on cloud computing security[J]. Journal of Software, 2011,22(1):71−83

[4] Xu M, Gao D, Deng C, Luo ZG, Sun SL. Cloud computing boosts business intelligence of telecommunication industry[C]. Proc. of the 1st Int'l Conf. on Cloud Computing (CloudCom 2009). Berlin: Springer-Verlag, 2009.224−231.

[5] Qi J, Qian L, Luo ZG. Distributed structured database system HugeTable[C]. Proc. of the 1st Int'l Conf. on Cloud Computing (CloudCom 2009). Berlin: Springer-Verlag, 2009. 338−346.

[6] Abouzeid A, Bajda-Pawlikowski K, Abadi DJ, Silberschatz A, Rasin A. HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads[C]. PVLDB, 2009,2(1):922−933.

[7] Ahrens M, Alonso G. Relational databases, virtualization, and the cloud[C]. Proc. Of the 27th Int'l Conf. on Data Engineering (ICDE 2011). New York: IEEE Computer Society Press, 2011. 1254.

[8] Agrawal D, Abbadi AE, Das S, Elmore AJ. Database scalability, elasticity, and autonomy in the cloud—(extended abstract)[C]. Proc. of the 16th Int'l Conf. on Database Systems for Advanced Applications (DASFAA 2011). Berlin: Springer-Verlag, 2011. 2−15.

[9] Soares L, Pereira J. Improving the scalability of cloud-based resilient database servers[C]. Proc. of the 11th IFIP WG 6.1 Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS 2011). Berlin: Springer-Verlag, 2011.136−149.

[10] Ion M, Russello G, Crispo B. Enforcing multi-user access policies to encrypted cloud databases[C]. Proc. of the IEEE Int'l Symp. on Policies for Distributed Systems and Networks (POLICY 2011). New York: IEEE Computer Society Press, 2011. 175−177.

[11] Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE. Bigtable: A distributed storage system for structured data[J]. ACM Trans. on Computer Systems, 2008,26(2):1−26.

[12] Yoon JP. Access control and trustiness for resource management in cloud databases[C]. Proc. of the Int'l Conf. on Grid and Cloud Database Management. Berlin: Springer-Verlag, 2011. 109−131.

[13] Chen C, Chen G, Jiang DW, Ooi BC, Vo HT, Wu S, Xu QQ. Providing scalable database services on the cloud[C]. Proc. of the 11th Int'l Conf. on Web Information Systems Engineering (WISE 2010). Berlin: Springer-Verlag, 2010. 1−19.

[14] Indu Arora1 and Dr. Anu Gupta2, Cloud Databases: A Paradigm Shift in Databases[J], IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 4, No 3, July 2012 ISSN (Online): 1694-0814

[15] Daniel J. Abadi, New Haven, Data Management in the Cloud: Limitations and Opportunities[M]
[16] Yvette E. Gelogo and Sunguk Lee, Database Management System as a Cloud Service as a Cloud Service[J], International Journal of Future Generation Communication and Networking Vol. 5, No. 2, June, 2012
[17] Sudipto Das Shashank Agarwal Divyakant Agrawal Amr El Abbadi, ElasTraS: An Elastic, Scalable, and Self Managing Transactional Database for the Cloud[J], UCSB Computer Science Technical Report 2010-04.
[18] Carlo Curino , Evan Jones , Yang Zhang , Eugene Wu, and Samuel Madden, Relational Cloud: The Case for a Database Service[J/OL], Computer Science and Artificial Intelligence Laboratory Technical Report massachuse t t s i n stitute of technology, cambridge, ma 02139 usa — www. c s a il.mit.edu MIT-CSAIL-TR-2010-014 March 14, 2010
[19] Shyam Kumar DODDAVULA, Bangalor (IN); Abhishek Pratap SINGH, Ajmer (IN), SYSTEM AND METHOD FOR IMPLEMENTING ON DEMAND CLOUD DATABASE[P], US Patent Application Publication, Pub.No US 2012/02966866 A1, Nov.22, 2012
[20] 林子雨,赖永炫,林琛,谢怡,邹权. 云数据库研究[J], 软件学报 ISSN 1000-9825, Journal of Software,2012,23(5):1148−1166
[21] Cooper BF, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen HA, Puz N, Weaver D, Yerneni R. Pnuts: Yahoo!’s hosted data serving platform[C]. Proc. of the VLDB Endowment, 2008,1(2):1277−1288.
[22] 史恒亮,任崇广,白光,普杰信. 自适应蚁群优化的云数据库动态路径查询[J].计算机工程与应用. 2010,46（9）10-12 48
[23] Das S, Agrawal D, Abbadi AE. G-Store: A scalable data store for transactional multi key access in the cloud[C]. In: Hellerstein JM, Chaudhuri S, Rosenblum M, eds. Proc. of the 1st ACM Symp. on Cloud Computing (SoCC 2010). New York: ACM Press, 2010. 163−174.
[24] Hacigümüs H, Tatemura J, Hsiung WP, Moon HJ, Po O, Sawires A, Chi Y, Jafarpour H. CloudDB: One size fits all revived[C]. In: Proc. of the 6th World Congress on Services (SERVICES 2010). New York: IEEE Computer Society, 2010. 148−149.
[25] Stonebraker M. Technical perspective—One size fits all: An idea whose time has come and gone. Communications of the ACM, 2008,51(12):76.
[26] 李超零,陈越,谭鹏许,杨刚,李文俊. 基于分解与加密的云数据库隐私保护机制研究[J]. 信息工程大学学报. 第 13 卷第 3 期 2012 年 6 月 1671-0673
[27] 陈平平,谭定英,刘秀峰. 可扩展的云关系型数据库的研究[J]. 第 33 卷,第 7 期,1000-7024（2012）

# 谢辞

四年的本科学习生涯即将结束，回顾过去，取得的成绩已成为过眼云烟，未来还有很多未知的挑战在等待着我。目前项目已经接近尾声，但是整个项目带给我的思考与感悟一直让我不能释怀。

在这里，首先要感谢我的指导老师，邹恒明老师，他不论是在立项的过程中，还是在项目的开发过程中，都给予了我极大的帮助。在整个项目过程中，邹老师给予了我极大的关心与鼓励，定时的到公司跟我们开会交流，认真负责地监督我的工作。每当我遇到困难的时候，邹老师总能在第一时间提出自己的意见与建议。

其次，要感谢一路陪我一起实习的同学们。如果没有他们的帮助，这个项目也不可能达到预期的效果。我从每个同学身上都学到了很多，他们每个人都有自己特别擅长的地方，而我，正是在与他们交流合作的过程中渐渐发现了自己的不足，并且逐步改进。

再次，感谢所有帮助我的朋友们和我的父母。每当我失意的时候，你们的支持与鼓励能给我最大的动力和勇气；在我得意之时，你们有不断提醒我戒骄戒躁，踏踏实实走好每一步。

最后，我要感谢软件学院和上海交通大学为我提供了优秀的硬件环境和杰出的师资队伍，让我在整个四年中不断成长。

# ON CONSTRUCTION OF HIGH-PERFORMANCE CLOUD DATABASE

Based on the Internet, Cloud Computing is a mode concerning the increase, usage, and delivery mode of Internet-related services, usually involves providing dynamic, scalable, and often virtualized resources via the Internet. With its increasing popularity, Cloud Computing technology began to exert more and more influence on other technological fields, among which a typical one is database. Cloud Database, releasing users from repeating the configuration of software and hardware by making many backstage services transparent, was derived from SaaS (software-as-a-service, software as a service) at the time when SaaS was enjoying great popularity. Besides, it is also convenient to upgrade. Cloud Database possesses many advantages, such as high scalability, high usability, multi-rent forms, and the support of efficiency in the distribution of resources and so on.

For the academic community, many problems are remained to be solved before providing rich functions as that of existing DBMS in database, such as searching, index, and transaction processing. Research issues in the field of Cloud Database mainly include— the design of data model in Cloud Database, programming model, the design of system structure of the server, consistency of tasks, disaster recovery and SLA (Service Level Agreement, services, etc. Level agreement) monitoring based on Cloud Database, access control and authorization management of Cloud Data, system tuning of cloud application data, lifecycle management of Cloud Data, collaboration of cloud and local database, testing benchmarks and so on. This research aims to improving or re-design the current data model, system architecture and related advanced features of different kinds of Cloud Database, so that improving the service level of the current Cloud Database field, and implementing a new Cloud Database prototype at the same time.

Here are our research ideas and methods:

(1) Researching the current storage systems used by the data model, modified or redesigned a data model is suitable for the Cloud Computing environment;

(2) Utilizing the data model designing to develop a high-performance singleton database prototype;

(3) Researching on the current Cloud Database system architecture to improve or design a new system architecture, and use it to implement the Cloud Database prototype;

(4) Test and improve performance by methods such as resources distribution and load balancing, etc.

(5) Exploring the advanced features such as high availability, transactional consistency, backup/ recovery, data migration and so on, and then put up specified solutions for this case.

The main result of this research is a prototype of Cloud Database, which combined graphic model of NoSQL data model and existing storage technology such as decentralized distributed storage and has made some improvements, which enabled the system to display basic characteristics of Cloud Database such as extensibility, high performance, load balance and so on.

Meanwhile, this research looked into current research of advanced features and possible implementation ways for this system as well, including consistency of tasks, high usability, data security, file organization, performance optimization, testing benchmark, backup and recovery, data migration, and information retrieval, among which high usability, data security, file organization, backup and storage, and information retrieval have already been realized. In the end, the correctness of this system was proved by complete and consistent performance test. Detailed work are summarized as following:

(1) Studied and analyzed existing data model, in which key / value model, relational model and graph model are concentrations. Improved the graphical model and applied it to the framework of the discussed system;

(2) Designed and implemented the distributed underlying architecture of the Cloud Database discussed in this essay. Used a peer to peer storage mode on the service side, reducing performance bottlenecks of single points and the risk of failure as well as possessing the ability to automatically discover and remove other nodes;

(3) Weakened consistency requirement and emphasized eventual consistency (allows a short time data inconsistencies) in order to achieve the original goal of high performance;

(4) Realized high availability by implementing log shipping and redundant replication methods, which could maintain certain overall usability when encountering node failure;

(5) Proposed a new data encryption to encrypt data; meanwhile, designed two encryption modes—data encryption and file encryption;

(6) Proposed "file pointer"— a kind of underlying storage file organization, which provided intuitive readability of underlying data;

(7) Realized performance optimization by balancing data amount of nodes, tuning performance when developing and other means;

(8) Designed and implemented three ways to backup/restore, including hot backup/restore, cold backup/restore and journal-based backup/restore;

(9) Proposed several solutions to information retrieval using the database discussed in this essay and realized on the them— searching target nodes that are close to specific nodes based on depth;

(10) The testing results of operating efficiency, response time, expanded impact, and recovery capabilities revealed the system discussed in this essay had realized goal of high performance, scalable, and high available.

In short, the design and implementation of this system had met expected goals on the whole. The results of the experiments had also demonstrated the feasibility of this system.

This topic presents the construction mode of a high performance Cloud Database. However, due to time constraints and limited experimental conditions, the present study has the following problems:

(1) This database used a graph model, which is still in experimental research stage and has not been widely used in the market. Therefore, its reliability has not been verified and it may have potential pitfalls;

(2) This system guarantees eventual consistency instead of transactional consistency. So, if does not apply to systems which require relatively strong data consistency, such as banking and securities trading system;

(3) Due to limited experimental conditions, the cluster test did not cover a large scale; and

further testing was impossible because of the lack of necessary equipment. Meanwhile, testing of many advanced features was limited as well. In the future, we need to continue cluster testing on a large scale to find its performance variations and other features of practicality, in order to verify the feasibility of the framework, and to continuously improve the framework.

(4) Because of time limitation, a graphical visual interface of the client had not been developed temporarily. And to a graph model, the intuitive graphical interface is of more use value, which needs to be implemented in the future.

The following is the papers' structure.

Chapter one at first introduced the research background, including the basic concepts of Cloud Computing, Cloud Database and its features, products and then put forward the current problems in the field of research, research purpose and its significance, then introduced topic of main research ideas and methods. Finally, it shows its organizational structure.

The second chapter introduced several common data models, and then raised the improvement approach of graph model and its features including the storage mode, the basic characteristics of the model, the serialized representation and comparison to other models by a wide range of versatility.

Chapter three brought up several common cloud database architectures, and then put forward the architecture—server side as peer-to-peer and client random access. According to this architecture related data access method, system instance, data partitioning algorithm, starting method, read and write operations, the complex equilibrium and elastic extension strategy were proposed.

Chapter four firstly summarizes the current research status and product profiles of several advanced capabilities of Cloud Database, and then based on this database solutions or ideas were put forward. Among them in total nine, including transactional consistency, high availability, security, performance optimization, benchmark, backup/restore, data migration and including information retrieval function.

Chapter five introduced specific implementation methods, function module, database interface, data manipulation, user interface, the command list and development environment, detailed description of the entire database from the design to the implementation process.

The sixth chapter on the database of the whole system has carried on the detailed test, first emphatically analyzes the performance of the system. From the experimental results, the performance of the database is good, which can meet the needs of the development and application. Next, this chapter analyzes the performance compared with the performance of mainstream database (taking MySQL as the example). At last, it is through analysis of actual test the high availability of this database. In general, this system can meet the expected design requirements.

The last chapter was mainly summary of the whole thesis. In this chapter, first of all, it systematically summarized the research contents of this topic, and the research results were summarized at the same time. Then defects were analyzed in this project and it put forward the direction of future research work.