摘 要

随着因特网应用和计算机技术的飞速发展，数据库逐渐成为信息系统的核心部分并广泛应用于企业、金融机构、政府及国防等各个领域。其中，分布式关系型数据库以其低成本、高可靠性等特点成为当前数据库理论与应用领域的研究热点。

通过对分布式理论、关系型数据库理论及相关技术的学习和研究，本文基于Java语言实现了一个分布式的关系型数据库JSQL。JSQL主要包含五大模块，网络模块接受前端应用程序的连接请求，对客服端进行认证和授权，并对连接进行管理，可实现基于Mysql的通信协议，这样便于Mysql的用户方便的迁移到本系统上来；Sql的解析和执行模块接受客户端的SQL请求，然后解析和执行数据库存储的调用，返回执行结果；审计模块是存储和分析所有对数据库的更改情况，以可视化的方式向用户展示；数据库引擎模块利用了orientdb开源的数据库引擎，实现可靠的分布式存储；分布式模块利用hazlcast实现了数据库集群。基于以上功能模块，ＪＳＱＬ具有高可用性，可扩展性，负载均衡等特性，同时从数据库底层考虑了数据库安全审计需求，加入了数据审计图形化界面显示审计结果。

论文对系统的进行了功能和性能测试。功能测试结果表明，系统在功能上符合分布式数据库的基本要求，审计系统的功能也达到本论文的要求。论文通过对性能测试结果进行分析，认为系统的性能基本达到本论文的要求。但是本系统对复制SQL语句的支持还不是很完善，最后提出了改进的方案。

关键词：分布式数据库，mysql，安全审计，OLTP，NOSQL

ABSTRACT

Database, operating system and compiler and called the three systems, can be said that the cornerstone of the entire computer software. Which is closer to the application layer database, is a lot of business support. This field after decades of development, There are new developments. From the beginning of the hierarchy database and relational database, to the recent hot Nosql database, and then to the recent Google Spanner and F1 as the representative of the NewSql database.

In the Internet age, the storage and access of massive data becomes the bottleneck of system design and use. For mass data processing, King, divided into two types: online transaction processing (OLTP) and online analytical processing (OLAP).

Relational database is based on the relational model of the database, which by means of aggregation algebra and other mathematical concepts and methods to deal with the database data. Which is the most popular mysql, mysql is an open source relational database, the advantage lies in the open source code, any business and individuals can according to their own needs to modify the source code mysql.

NoSQL database, called Not Only SQL, meaning that when the relational database is used when the relational database, not applicable There is no need to use relational database is not necessary, you can consider the use of more appropriate data storage.

Oracle, mysql and other traditional relational database is very mature and has been large-scale commercial, why use NoSQL database? mainly With the development of the Internet, the amount of data is growing, the performance requirements are getting higher and higher, the traditional database of congenital defects, namely stand-alone (single Library) performance bottlenecks, and difficult to expand. This is a stand-alone single library bottleneck, but difficult to expand, naturally unable to meet the growing mass of data storage And its performance requirements, so there will be a variety of different NoSQL products.

Although in the cloud computing era, the traditional database there are congenital defects, but NoSQL database can not be replaced, NoSQL can only For the traditional data supplement can not be replaced, so to avoid the shortcomings of traditional databases is the current era of large data must be resolved.

In order to solve these problems, such as mysql and other relational database, this article describes how to design and implement a compatible mysql protocol distributed database. He can automatically find the distributed database cluster nodes, Automatically allocate data to support massive data storage. Taking into account the increasingly important security of the database, from time to time the occurrence of database administrators or other attackers malicious changes in the database data, So the database developed by the database from the bottom of the database to join the audit function.

The main work of this paper is as follows:

Based on the realization of java language compatible mysql communication protocol database;

The realization of the database cluster, to achieve the database of high availability, scalability, load balancing and other characteristics;

From the bottom of the database to consider the database security, joined the data audit and other features.

Keywords: mysql, java, mysql, java, mysql

目 录

第一章 绪论

在互联网时代，海量数据的存储与访问成为系统设计与使用的瓶颈问题，对于海量数据处理，按照使用场景，主要分为两种类型：联机事务处理（OLTP）和联机分析处理（OLAP）。

联机事务处理（OLTP）也称为面向交易的处理系统，其基本特征是原始数据可以立即传送到计算中心进行处理，并在很短的时间内给出处理结果。

联机分析处理（OLAP）是指通过多维的方式对数据进行分析、查询和报表，可以同数据挖掘工具、统计分析工具配合使用，增强决策分析功能。

1.1 研究工作的背景与意义

web2.0的到来，存储海量数据的需求越来越大。

在互联网时代，海量数据的存储与访问成为系统设计与使用的瓶颈问题，对于海量数据处理，按照使用场景，主要分为两种类型：联机事务处理（OLTP）和联机分析处理（OLAP）。

联机事务处理（OLTP）也称为面向交易的处理系统，其基本特征是原始数据可以立即传送到计算中心进行处理，并在很短的时间内给出处理结果。

联机分析处理（OLAP）是指通过多维的方式对数据进行分析、查询和报表，可以同数据挖掘工具、统计分析工具配合使用，增强决策分析功能。

1.2 国内外研究历史与现状

google spanner和t1等系统。在互联网时代，海量数据的存储与访问成为系统设计与使用的瓶颈问题，对于海量数据处理，按照使用场景，主要分为两种类型：联机事务处理（OLTP）和联机分析处理（OLAP）。

联机事务处理（OLTP）也称为面向交易的处理系统，其基本特征是原始数据可以立即传送到计算中心进行处理，并在很短的时间内给出处理结果。

联机分析处理（OLAP）是指通过多维的方式对数据进行分析、查询和报表，可以同数据挖掘工具、统计分析工具配合使用，增强决策分析功能。

1.3 论文的主要工作

结合nosql的关系型数据库的特点，同时从数据库自身考虑数据库安全。 ...

1.4 本论文的结构安排

本论文后续各章结构如下：

第二章介绍了

第三章介绍了

第四章描述了审计子系统是如何实现，同时将涉及的技术方法作了介绍和分析。

第五章对审计管理器模块进行了分析和设计。首先提出该模块的概述，并结合这些目标对系统总体结构和重要的功能进行了详细的分析和设计，从而保证设计目标的实现。

第六章主要通过介绍审计子系统的具体应用，说明了审计子系统的实用价值和地位。

结论部分详细总结了本论文的研究工作，并针对存在的主要不足指出了改进的重点和今后的方向。

1.5 本章要求

...

% !Mode:: ”TeX:UTF-8”

第二章 理论基础和相关的技术

本章介绍分布式数据库方面的相关理论和技术，还有mysql协议的研究和学习。

2.1 分布式数据库概念和相关技术

2.1.1 数据库的分类

数据库通常分为层次式数据库、网络式数据库和关系式数据库三种。而不同的数据库是按不同的数据结构来联系和组织的。安装机器数量的多少，数据库系统也可以分为单机版数据库和分布式数据库。

而在当今的互联网中，最常见的数据库模型主要是两种，即关系型数据库和非关系型数据库。

2.1.2 分布式数据库的介绍

分布式数据库是用计算机网络将物理上分散的多个数据库单元连接起来组成的一个逻辑上统一的数据库。每个被连接起来的数据库单元称为站点或节点。分布式数据库有一个统一的数据库管理系统来进行管理，称为分布式数据库管理系统。

分布式数据库的基本特点包括：物理分布性、逻辑整体性和站点自治性。从这三个基本特点还可以导出的其它特点有：数据分布透明性、按既定协议达成共识的机制、适当的数据冗余度和事务管理的分布性。

分布式数据库按照各站点中数据库管理系统的数据模型的异同分为异构型分布式数据库和同构型分布式数据库，按照控制系统的类型分为全局控制集中性、全局控制分散型和全局控制可变型。

优点：

●随时能针对各区域的用户做调整

●数据共用和分布式控制

●增加处理绩效，可作平行处理

●系统管理费用较低

●质量维持容易

缺点：

●重复存储数据很花时间

●数据处理与管理上具复杂度

●数据的保密性与安全性受到威胁

2.1.3 分布式数据库相关技术

2.1.3.1 负载均衡技术

面对大量用户访问、高并发请求，海量数据，可以使用高性能的服务器、大型数据库，存储设备，高性能Web服务器，采用高效率的编程语言，当单机容量达到极限时，我们需要考虑业务拆分和分布式部署，来解决大型网站访问量大，并发量高，海量数据的问题。

从单机网站到分布式网站，很重要的区别是业务拆分和分布式部署，将应用拆分后，部署到不同的机器上，实现大规模分布式系统。分布式和业务拆分解决了，从集中到分布的问题，但是每个部署的独立业务还存在单点的问题和访问统一入口问题，为解决单点故障，我们可以采取冗余的方式。将相同的应用部署到多台机器上。解决访问统一入口问题，我们可以在集群前面增加负载均衡设备，实现流量分发。

负载均衡，意思是将负载进行平衡、分摊到多个操作单元上进行执行。是解决高性能，单点故障,高可用，扩展性的终极解决方案。

系统的扩展可分为纵向垂直扩展和横向水平扩展。纵向扩展，是从单机的角度通过增加硬件处理能力，比如CPU处理能力，内存容量，磁盘等方面，实现服务器处理能力的提升，不能满足大型分布式系统大流量，高并发，海量数据的问题。因此需要采用横向扩展的方式，通过添加机器来满足大型网站服务的处理能力。比如：一台机器不能满足，则增加两台或者多台机器，共同承担访问压力。

负载均衡的作用：

●解决并发压力，提高应用处理性能增加吞吐量，加强网络处理能力

●提供故障转移，实现高可用

●通过添加或减少服务器数量，提供网站伸缩性1扩展性

●安全防护，负载均衡设备上做一些过滤，黑白名单等处理

根据实现技术不同，可分为DNS负载均衡，HTTP负载均衡，IP负载均衡，链路层负载均衡等。

DNS负载均衡 DNS负载均衡是最早的负载均衡技术，利用域名解析实现负载均衡，在DNS 服务器，配置多个A记录，这些A记录对应的服务器构成集群。大型网站总是部分使用DNS解析，作为第一级负载均衡。

优点：

- 使用简单：负载均衡工作，交给DNS服务器处理，省掉了负载均衡服务器维护的麻烦
- 提高性能：可以支持基于地址的域名解析，解析成距离用户最近的服务器地址，可以加快访问速度，改善性能；

缺点

- 可用性差：DNS解析是多级解析，新增修改DNS后，解析时间较长；解析过程中，用户访问网站将失败；
- 扩展性低：DNS负载均衡的控制权在域名商那里，无法对其做更多的改善和扩展；
- 维护性差：也不能反映服务器的当前运行状态；支持的算法少；不能区分服务器的差异,不能根据系统与服务的状态来判断负载.

将DNS作为第一级负载均衡，A记录对应着内部负载均衡的IP地址，通过内部负载均衡将请求分发到真实的Web服务器上。一般用于互联网公司，复杂的业务系统不合适使用。

IP负载均衡 IP负载均衡是在网络层通过修改请求目标地址进行负载均衡。用户请求数据包，到达负载均衡服务器后，负载均衡服务器在操作系统内核进程获取网络数据包，根据负载均衡算法得到一台真实服务器地址，然后将请求目的地址修改为，获得的真实ip地址，不需要经过用户进程处理。真实服务器处理完成后，响应数据包回到负载均衡服务器，负载均衡服务器，再将数据包源地址修改为自身的ip地址，发送给用户浏览器。 IP负载均衡，真实物理服务器返回给负载均衡服务器，存在两种方式：负载均衡服务器在修改目的ip地址的同时修改源地址。将数据包源地址设为自身盘，即源地址转换1snat。将负载均衡服务器同时作为真实物理服务器集群的网关服务器。

优点：

- 在内核进程完成数据分发，比在应用层分发性能更好；

缺点：

- 所有请求响应都需要经过负载均衡服务器，集群最大吞吐量受限于负载均衡服务器网卡带宽；

链路层负载均衡 链路层负载均衡是在通信协议的数据链路层修改mac地址，进行负载均衡。数据分发时，不修改ip地址，指修改目标mac地址，配置真实物理服务器集群所有机器虚拟ip和负载均衡服务器ip地址一致，达到不修改数据包的源地址和目标地址，进行数据分发的目的。实际处理服务器ip和数据请求目的ip一致，不需要经过负载均衡服务器进行地址转换，可将响应数据包直接返回给用户浏览器，避免负载均衡服务器网卡带宽成为瓶颈。也称为直接路由模式1DR模式.

优点：

●性能好

●配置复杂；

由于多个服务器群内硬件设备、各自的规模、提供的服务等的差异，可以考虑给每个服务器群采用最合适的负载均衡方式，然后又在这多个服务器群间再一次负载均衡或群集起来以一个整体向外界提供服务即把这多个服务器群当做一个新的服务器群1，从而达到最佳的性能。将这种方式称之为混合型负载均衡。此种方式有时也用于单台均衡设备的性能不能满足大量连接请求的情况下。是目前大型互联网公司，普遍使用的方式。

2.1.3.2 数据分片技术

在分布式存储系统中，数据需要分散存储在多台设备上，数据分片就是用来确定数据在多台存储设备上分布的技术。数据分片要达到三个目的：

●分布均匀，即每台设备上的数据量要尽可能相近；

●负载均衡，即每台设备上的请求量要尽可能相近；

●扩缩容时产生的数据迁移尽可能少。

数据分片一般都是使用Key或Key的哈希值来计算Key的分布，常见的几种数据分片的方法如下：

●划分号段。这种一般适用于Key为整型的情况，每台设备上存放相同大小的号段区间，如把Key为[1, 10000]的数据放在第一台设备上，把Key为[10001, 20000]的数据放在第二台设备上，依次类推。这种方法实现很简单，扩容也比较方便，成倍增加设备即可，如原来有N台设备，再新增N台设备来扩容，把每台老设备上一半的数据迁移到新设备上，原来号段为[1, 10000]的设备，扩容后只保留号段[1, 5000]的数据，把号段为[5001, 10000]的数据迁移到一台新增的设备上。此方法的缺点是数据可能分布不均匀，如小号段数据量可能比大号段的数据量要大，同样的各个号段的热度也可能不一样，导致各个设备的负载不均衡；并且扩容也不够灵活，只能成倍地增加设备。

- 取模。这种方法先计算Key的哈希值，再对设备数量取模1整型的Key也可直接用Key 取模1，假设有N台设备，编号为0 N-1，通过Hash1Key1N就可以确定数据所在的设备编号。这种方法实现也非常简单，数据分布和负载也会比较均匀，可以新增任何数量的设备来扩容。主要的问题是扩容的时候，会产生大量的数据迁移，比如从N台设备扩容到N+1台，绝大部分的数据都要在设备间进行迁移。检索表。在检索表中存储Key和设备的映射关系，通过查找检索表就可以确定数据分布，这里的检索表也可以比较灵活，可以对每个Key都存储映射关系，也可结合号段划分等方法来减小检索表的容量。这样可以做到数据均匀分布、负载均衡和扩缩容数据迁移量少。缺点是需要存储检索表的空间可能比较大，并且为了保证扩缩容引起的数据迁移量比较少，确定映射关系的算法也比较复杂。
- 一致性哈希。一致性哈希算法在1997年由麻省理工学院提出的一种分布式哈希实现算法，设计目标是为了解决因特网中的热点问题，一致性哈希的算法简单而巧妙，很容易做到数据均分布，其单调性也保证了扩缩容的数据迁移是比较少的。通过上面的对比，在这个系统选择一致性哈希的方法来进行数据分片。

2.1.3.3 数据库高可用技术

高可用性指的是通过尽量缩短因日常维护操作和突发的系统崩溃所导致的停机时间，以提高系统和应用的可用性。HA系统是目前企业防止核心计算机系统因故障停机的最有效手段。

随着IT信息系统的不断发展，数据在企业的应用越来越广，如何提高IT系统的高可用性成为建设稳健的计算机系统的首要任务之一。构成计算机网络系统的三大要素是：网络系统，服务器系统，存储系统。网络系统包括防火墙，路由器等网络设备，服务器系统主要指用户使用的各种服务器系统，存储系统，则是用户最主要的数据存储放的地点。因此IT系统的高可用建设应包括网络设备高可用性，服务器设备高可用性，及存储设备的高可用性三个方面。

- 网络高可用,由于网络存储的快速发展，网络冗余技术被不断提升，提高IT系统的高可用性的关键应用就是网络高可用性，网络高可用性与网络高可靠性是有区别的，网络高可用性是通过匹配冗余的网络设备实现网络设备的冗余，达到高可用的目的。比如冗余的交换机，冗余的路由器等
- 服务器高可用, 服务器高可用主要使用的是服务器集群软件或高可用软件来实现。

- 存储高可用, 使用软件或硬件技术实现存储的高度可用性。其主要技术指标是存储切换功能，数据复制功能，数据快照功能等。当一台存储出现故障时，另一台备用的存储可以快速切换，达一存储不停机的目的。

负载均衡

负载均衡的概念

负载均衡在多节点之间按照一定的策略分发网络或计算处理负载, 提供了一种廉价而有效的方法来扩展服务器带宽, 增加吞吐量, 提高数据处理能力, 同时又可以避免单点故障。负载均衡包括静态负载平衡和动态负载平衡。只是利用系统负载的平均信息, 而忽视系统当前的负载状况的方法被称为静态负载均衡; 根据系统当前的负载状况来调整任务划分的方法被称为动态负载均衡。目前, 静态负载均衡已越来越不能满足需要, 动态负载均衡技术有取而代之的趋势。但是, 如何降低负载信息获取、交互所占用的系统资源将成为动态负载均衡技术所必须解决的问题。

负载均衡算法

常用的负载均衡算法有，轮询，随机，最少链接，源地址散列，加权等方式。

轮询 将所有请求，依次分发到每台服务器上，适合服务器硬件同相同的场景。优点：服务器请求数目相同；缺点：服务器压力不一样，不适合服务器配置不同的情况；

随机 请求随机分配到各个服务器。优点：使用简单；缺点：不适合机器配置不同的场景；

最少链接 将请求分配到连接数最少的服务器1目前处理请求最少的服务器1。优点：根据服务器当前的请求处理情况，动态分配；缺点：算法实现相对复杂，需要监控服务器请求连接数；

Hash1源地址散列1 根据IP地址进行Hash计算，得到IP地址。优点：将来自同一IP地址的请求，同一会话期内，转发到相同的服务器；实现会话粘滞。缺点：目标服务器宕机后，会话会丢失；

加权 在轮询，随机，最少链接，Hash’等算法的基础上，通过加权的方式，进行负载服务器分配。优点：根据权重，调节转发服务器的请求数目；缺点：使用相对复杂；

硬件负载均衡 采用硬件的方式实现负载均衡，一般是单独的负载均衡服务器，价格昂贵，一般土豪级公司可以考虑，业界领先的有两款，F5和A10。使用硬件负载均衡，主要考虑一下几个方面： 111功能考虑：功能全面支持各层级的负载均衡，支持全面的负载均衡算法，支持全局负载均衡； 121性能考虑：一般软件负载均衡支持到5万级并发已经很困难了，硬件负载均衡可以支持 131稳定性：商用硬件负载均衡，经过了良好的严格的测试，从经过大规模使用，在稳定性方面高；141安全防护：硬件均衡设备除具备负载均衡功能外，还具备防火墙，防DDOS攻击等安全功能； 151维护角度：提供良好的维护管理界面，售后服务和技术支持；161土豪公司：F5 Big Ip 价格：15w 55w不等；A10 价格：55w-100w不等；缺点111价格昂贵； 121扩展能力差； 4.4小结 111一般硬件的负载均衡也要做双机高可用，因此成本会比较高。 121互联网公司一般使用开源软件，因此大部分应用采用软件负载均衡；部分采用硬件负载均衡。比如某互联网公司，目前是使用几台F5做全局负载均衡，内部使用Nginx等软件负载均衡。

一致性hash算法

一致哈希是一种特殊的哈希算法。在使用一致哈希算法后，哈希表槽位数1大小1的改变平均只需要对个关键字重新映射，其中K是关键字的数量， n是槽位数量。然而在传统的哈希表中，添加或删除一个槽位的几乎需要对所有关键字进行重新映射。

一致哈希由MIT的Karger及其合作者提出，现在这一思想已经扩展到其它领域。在这篇1997年发表的学术论文中介绍了“一致哈希”如何应用于用户易变的分布式Web服务中。哈希表中的每一个代表分布式系统中一个节点，在系统添加或删除节点只需要移动项。

一致哈希也可用于实现健壮缓存来减少大型Web应用中系统部分失效带来的负面影响。

一致哈希的概念还被应用于分布式散列表1DHT1的设计。DHT使用一致哈希来划分分布式系统的节点。所有关键字都可以通过一个连接所有节点的覆盖网络高效地定位到某个节点。

在使用n台缓存服务器时，一种常用的负载均衡方式是，对资源的请求使用n来映射到某一台缓存服务器。当增加或减少一台缓存服务器时这种方式可能会

改变所有资源对应的hash值，也就是所有的缓存都失效了，这会使得缓存服务器大量集中地向原始内容服务器更新缓存。因此需要一致哈希算法来避免这样的问题。一致哈希尽可能使同一个资源映射到同一台缓存服务器。这种方式要求增加一台缓存服务器时，新的服务器尽量分担存储其他所有服务器的缓存资源。减少一台缓存服务器时，其他所有服务器也可以尽量分担存储它的缓存资源。一致哈希算法的主要思想是将每个缓存服务器与一个或多个哈希值域区间关联起来，其中区间边界通过计算缓存服务器对应的哈希值来决定。1定义区间的哈希函数不一定和计算缓存服务器哈希值的函数相同，但是两个函数的返回值的范围需要匹配。1如果一个缓存服务器被移除，则它所对应的区间会被并入到邻近的区间，其他的缓存服务器不需要任何改变。

一致哈希将每个对象映射到圆环边上的一个点，系统再将可用的节点机器映射到圆环的不同位置。查找某个对象对应的机器时，需要用一致哈希算法计算得到对象对应圆环边上位置，沿着圆环边上查找直到遇到某个节点机器，这台机器即为对象应该保存的位置。当删除一台节点机器时，这台机器上保存的所有对象都要移动到下一台机器。添加一台机器到圆环边上某个点时，这个点的下一台机器需要将这个节点前对应的对象移动到新机器上。更改对象在节点机器上的分布可以通过调整节点机器的位置来实现。

David Karger及其合作者列出了使得一致哈希在互联网分布式缓存中非常有用的几个特性：

- 冗余少
- 负载均衡
- 过渡平滑
- 存储均衡
- 关键词单调

数据分片技术

数据分片概述

在分布式存储系统中，数据需要分散存储在多台设备上，数据分片1Sharding1就是用来确定数据在多台存储设备上分布的技术。数据分片要达到三个目的：

分布均匀，即每台设备上的数据量要尽可能相近；负载均衡，即每台设备上的请求量要尽可能相近；扩缩容时产生的数据迁移尽可能少。

数据库分片策略

数据分片一般都是使用Key或Key的哈希值来计算K ey的分布，常见的几种数据分片的方法如下：

划分号段 。这种一般适用于Key为整型的情况，每台设备上存放相同大小的号段区间，如把Key为[1, 10000]的数据放在第一台设备上，把Key为[10001, 20000]的数据放在第二台设备上，依次类推。这种方法实现很简单，扩容也比较方便，成倍增加设备即可，如原来有N台设备，再新增N台设备来扩容，

把每台老设备上一半的数据迁移到新设备上，原来号段为[1, 10000]的设备，扩容后只保留号段[1, 5000]的数据，把号段为[5001, 10000]的数据迁移到一台新增的设备上。此方法的缺点是数据可能分布不均匀，如小号段数据量可能比大号段的数据量要大，同样的各个号段的热度也可能不一样，导致各个设备的负载不均衡；并且扩容也不够灵活，只能成倍地增加设备。

取模 。这种方法先计算Key的哈希值，再对设备数量取模1整型的Key也可直接用Key取模1，假设有N台设备，编号为0 N-1，通过Hash1Key1就可以确定数据所在的设备编号。这种方法实现也非常简单，数据分布和负载也会比较均匀，可以新增任何数量的设备来扩容。主要的问题是扩容的时候，会产生大量的数据迁移，比如从N台设备扩容到N+1台，绝大部分的数据都要在设备间进行迁移。检索表。在检索表中存储Key和设备的映射关系，通过查找检索表就可以确定数据分布，这里的检索表也可以比较灵活，可以对每个Key都存储映射关系，

也可结合号段划分等方法来减小检索表的容量。这样可以做到数据均匀分布、负载均衡和扩缩容数据迁移量少。缺点是需要存储检索表的空间可能比较大，并且为了保证扩缩容引起的数据迁移量比较少，确定映射关系的算法也比较复杂。一致性哈希。一致性哈希算法1Consistent Hashing1 在1997年由麻省理工学院提出的一种分布式哈希1DHT1实现算法，

设计目标是为了解决因特网中的热点1Hot Spot1问题，一致性哈希的算法简单而巧妙，很容易做到数据均分布，其单调性也保证了扩缩容的数据迁移是比较少的。通过上面的对比，在这个系统选择一致性哈希的方法来进行数据分片。

数据库高可用技术

集群可靠性技术

集群1cluster1技术是一种较新的技术，通过集群技术，可以在付出较低成本的情况下获得在性能、可靠性、灵活性方面的相对较高的收益，其任务调度则是集群系统中的核心技术。

集群是一组相互独立的、通过高速网络互联的计算机，它们构成了一个组，并以单一系统的模式加以管理。一个客户与集群相互作用时，集群像是一个独立的服务器。集群配置是用于提高可用性和可缩放性。

高可用性集群的主要目的是为了使集群的整体服务尽可能可用。如果高可用性集群中的主节点发生了故障，那么这段时间内将由次节点代替它。次节点通常是主节点的镜像。当它代替主节点时，它可以完全接管其身份，因此使系统环境对于用户是一致的。高可用性集群使服务器系统的运行速度和响应速度尽可能快。它们经常利用在多台机器上运行的冗余节点和服务，用来相互跟踪。如果某个节点失败，它的替补者将在几秒钟或更短时间内接管它的职责。因此，对于用户而言,集群永远不会停机。在实际的使用中，集群的这三种类型相互交融，如高可用性集群也可以在其节点之间均衡用户负载。同样，也可以从要编写应用程序的集群中找到一个并行集群，它可以在节点之间执行负载均衡。从这个意义上讲，这种集群类别的划分是一个相对的概念，不是绝对的。

MYSQL复制

MySQL复制就是一台MySQL服务器从另一台MySQL服务器进行日志的复制然后再解析日志并应用到自身，类似Oracle中的Data Guard。

MySQL复制有那些好处：

第一是解决宕机带来的数据不一致，因为MySQL复制可以实时备份数据；第二点是减轻数据库服务器的压力，多台服务器的性能一般比单台要好。但是MySQL复制不适合大数据量，大数据量推荐使用集群。 MySQL复制过程分成三步：

master将改变记录到二进制日志。这些记录过程叫做二进制日志事件，binary log events； slave将master的binary log events拷贝到它的中继日志； slave重做中继日志中的事件，将改变应用到自己的数据库中。 MySQL复制是异步的且串行化的其他复制方式主主复制

master-slave只能进行单向操作，像网络中的半双工。 master-master可以实现服务器之间互相同步。主主复制最大的问题就是数据插入或更新冲突。配置方

法同主从复制，反过来让slave同步master。注意masterlogfile和masterlogpos参数要与master上对应。具体可以参考这里

单一master和多slave

由一个master和多个slave组成的复制系统比较简单。 slave之间并不互相通信，只能与master通信。如果写操作较少，读操作很多，可以采用。可以将读操作分布到其他slave，从而减轻master的压力。但slave增加到一定数量时，slave对master的负载以及网络带宽都会成为问题。

主从多级复制

读操作很多可以采用单一maste和多slave，但增大到一定slave后连到master的slaveIO线程太多会造成master压力增大，从而造成数据复制延时。多级复制就是为了解决这个问题。

当然，增加复制的级联层次，同一个变更传到最底层的Slave所需要经过的MySQL也会更多，同样可能造成延时较长的风险。如果条件允许，倾向于通过拆分成多个Replication集群来解决

数据一致性

数据一致性介绍

在数据有多分副本的情况下，如果网络、服务器或者软件出现故障，会导致部分副本写入成功，部分副本写入失败。这就造成各个副本之间的数据不一致，数据内容冲突。实践中，导致数据不一致的情况有很多种，表现样式也多种多样，比如数据更新返回操作失败，事实上数据在存储服务器已经更新成功。

CAP定理

CAP定理是2000年，由 Eric Brewer 提出来的。Brewer认为在分布式的环境下设计和部署系统时，有3 个核心的需求，以一种特殊的关系存在。

这里的分布式系统说的是在物理上分布的系统，比如我们常见的web系统。这3个核心的需求是：Consistency，Availability和P artition Tolerance，赋予了该理论另外一个名字－ CAP。 Consistency：一致性，这个和数据库ACID的一致性类似，但这里关注的所有数据节点上的数据一致性和正确性，而数据库的ACID关注的是在在一个事务内，对数据的一些约束。系统在执行过某项操作后仍然处于一致的状态。在分布式系统中，更新操作执行成功后所有的用户都应该读取到最新值。 Availability：可用性，每一个操作总是能够在一定时间内返回结果。需要注意“一定时间”和“返回结果”。“一定时间”是指，系统结果必须在给定时间内

返回。“返回结果”是指系统返回操作成功或失败的结果。 Partition Tolerance：分区容忍性，

是否可以对数据进行分区。这是考虑到性能和可伸缩性。 CAP定理认为，一个提供数据服务的存储系统无法同事满足数据一致性、数据可用性、分区容忍性。为什么不能完全保证这个三点了，个人觉得主要是因为一旦进行分区了，就说明了必须节点之间必须进行通信，涉及到通信，就无法确保在有限的时间内完成指定的行文，如果要求两个操作之间要完整的进行，因为涉及到通信，肯定存在某一个时刻只完成一部分的业务操作，在通信完成的这一段时间内，数据就是不一致性的。如果要求保证一致性，那么就必须在通信完成这一段时间内保护数据，使得任何访问这些数据的操作不可用。如果想保证一致性和可用性，

那么数据就不能够分区。一个简单的理解就是所有的数据就必须存放在一个数据库里面，不能进行数据库拆分。这个对于大数据量，高并发的互联网应用来说，是不可接受的。在大型网站应用中，数据规模总是快速扩张的，因此可伸缩性即分区容忍性必不可少，规模变大以后，机器数量也会变得庞大

，这是网络和服务器故障会频繁出现，要想保证应用可用，就必须保证分布式处理系统的高可用性。所以在大型网站中，通常会选择强化分布式存储系统的可用性1A1和伸缩性1P1，在某种程度上放弃一致性1C1。一般来说，

数据不一致通常出现在系统高并发写操作或者集群状态不稳1故障恢复、集群扩容等1的情况下，应用系统需要对分布式数据处理系统的数据不一致性有所了解并进行某种意义上的补偿和纠错，以避免出现应用系统数据不正确。

数据一致性模型

一些分布式系统通过复制数据来提高系统的可靠性和容错性，并且将数据的不同的副本存放在不同的机器，由于维护数据副本的一致性代价高，因此许多系统采用弱一致性来提高性能，一些不同的一致性模型也相继被提出。

强一致性：要求无论更新操作实在哪一个副本执行，之后所有的读操作都要能获得最新的数据。弱一致性：用户读到某一操作对系统特定数据的更新需要一段时间，我们称这段时间为“不一致性窗口”。最终一致性：是弱一致性的一种特例，保证用户最终能够读取到某操作对系统特定数据的更新。

数据一致性实现技术

两阶段提交算法

在两阶段提交协议中，系统一般包含两类机器1或节点1：一类为协调者1coordinator1，通常一个系统中只有一个；另一类为事务参与者，一般包含多个，在数据存储系统中可以理解为数据副本的个数。两阶段提交协议由两个阶段组成，在正常的执行下，这两个阶段的执行过程如下所述：

阶段1：请求阶段1commit-request pha se，或称表决阶段，voting phase1。在请求阶段，协调者将通知事务参与者准备提交或取消事务，然后进入表决过程。在表决过程中，参与者将告知协调者自己的决策：同意1事务参与者本地作业执行成功1或取消1本地作业执行故障1。阶段2：提交阶段1commit phase1。

在该阶段，协调者将基于第一个阶段的投票结果进行决策：提交或取消。当且仅当所有的参与者同意提交事务协调者才通知所有的参与者提交事务，否则协调者将通知所有的参与者取消事务。参与者在接收到协调者发来的消息后将执行响应的操作。举个例子：A组织B、C和D三个人去爬长城：如果所有人都同意去爬长城，那么活动将举行；如果有一人不同意去爬长城，那么活动将取消。用2PC算法解决该问题的过程如下：

首先A将成为该活动的协调者，B、C和D将成为该活动的参与者。阶段1：A发邮件给B、C和D，提出下周三去爬山，问是否同意。那么此时A需要等待B、C和D的邮件。B 、C和D分别查看自己的日程安排表。 B、C发现自己在当日没有活动安排，则发邮件告诉A它们同意下周三去爬长城。由于某种原因，D白天没有查看邮件。

那么此时A、B和C均需要等待。到晚上的时候，D发现了A的邮件，然后查看日程安排，发现周三当天已经有别的安排，那么D回复A说活动取消吧。阶段2：此时A收到了所有活动参与者的邮件，并且A发现 D下周三不能去爬山。那么A将发邮件通知B、C和D，下周三爬长城活动取消。此时B、C回复A“太可惜了”，D回复A“不好意思”。至此该事务终止。两阶段提交算法在分布式系统结合，可实现单用户对文件 1对象1 多个副本的修改，多副本数据的同步。其结合的原理如下：

客户端1协调者1向所有的数据副本的存储主机1参与者1发送：修改具体的文件名、偏移量、数据和长度信息，请求修改数据，该消息是1阶段的请求消息。存储主机接收到请求后，备份修改前的数据以备回滚，修改文件数据后，向客户端回应修改成功的消息。如果存储主机由于某些原因1磁盘损坏、空间不足等1不能

修改数据，回应修改失败的消息。客户端接收发送出去的每一个消息回应，如果存储主机全部回应都修改成功，向每

存储主机发送确认修改的提交消息；如果存在存储主机回应修改失败，或者超时未回应，客户端向所有存储主机发送取消修改的提交消息。该消息是2阶段的提交消息。存储主机接收到客户端的提交消息，如果是确认修改，则直接回应该提交OK

消息；如果是取消修改，则将修改数据还原为修改前，然后回应取消修改OK的消息。客户端接收全部存储主机的回应，整个操作成功。在该过程中可能存在通信失败，例如网络中断、主机宕机等诸多的原因，对于未在算法中定义的其它异常，都认为是提交失败，都需要回滚，这是该算法基于确定的通信回复实现的

，在参与者的确定回复1无论是回复失败还是回复成功1 之上执行逻辑处理，符合确定性的条件当然能够获得确定性的结果哲学原理。缺点：单个A是个严重问题：没有热备机制，A节点宕机了或者链接它的网络坏了会阻塞该事务；吞吐量不行，没有充分发动更多A的力量，一旦某个A第一阶段投了赞成票就得在它上面加独占锁，其他事务不得接入，直到当前事务提交or回滚。

分布式锁服务

分布式锁是对数据被外界修改持保守态度，在整个数据处理过程中将数据处于锁定状态，在用户修改数据的同时，其它用户不允许修改。采用分布式锁服务实现数据一致性，是在操作目标之前先获取操作许可，然后再执行操作，如果其他用户同时尝试操作该目标将被阻止，直到前一个用户释放许可后，其他用户才能够操作目标。分析这个过程，如果只有一个用户操作目

标，没有多个用户并发冲突，也申请了操作许可，造成了由于申请操作许可所带来的资源使用消耗，浪费网络通信和增加了延时。采用分布式锁实现多副本内容修改的一致性问题，选择控制内容颗粒度实现申请锁服务。例如我们要保证一个文件的多个副本修改一致，可以对整个文件修改设置一

把锁，修改时申请锁，修改这个文件的多个副本，确保多个副本修改的一致，修改完成后释放锁；也可以对文件分段，或者是文件中的单个字节设置锁，实现更细颗粒度的锁操作，减少冲突。常用的锁实现算法有Lamport bakery algorithm 1俗称面包店算法1，还有Paxos算法以及乐观锁。下面对其原理做简单概述。

1. Lamport面包店算法是解决多个线程并发访问一个共享的单用户资源的互斥问题的算法。由Leslie Lamport1英语：Leslie Lamport1发明。这个算法也可以称为时间戳策略，或者叫做Lamport逻辑时钟。这里先陈述一下这个逻辑时钟的内

容：我们用分布式系统中的事件的先后关系，用“1”符号来表示，例如：若事件a发生在事件b之前，那么a1b. 该关系需要满足下列三个条件：

如果a和b是同一进程中的事件，a在b之前发生，则a1b 如果事件a是消息发送方，b是接收方，则a1b 对于事件a、b、c,如果有a1b,b1c,则有a1c 注意，对于任何一个事件a，a 1 a都是不成立的，也就是说，关系1是反自反的。有了上面的定义，我们也可以定义出“并发”1concurrent1的概念了：

对于事件a、b，如果a 1 b，b 1 a两个都不成立，那么a和b就是并发的。

直观上，上面的1关系非常好理解，即“xxx在xxx之前发生”。也就是说，一个系统在输入I1下，如果有a1b，那么对于这个系统的同一个输入I1，无论重复运行多少次，a也始终发生在b之前；如果在输入I1下a和b是并发的，则表示在同一个输入I1下的不同运行中，a可能在b之前，也可能在b之后，也可能恰好同时发生。也就是，并发并不是指一定同时发生，而是表示一种不确定性。1和并发的

概念，就是我们理解一个系统时最基础的概念之一了。有了上面的概念，我们可以给系统引入时钟了。这里的时钟就是lamport逻辑时钟。一个时钟，本质上是一个事件到实数1假设时间是连续的1的函数。这个函数将每个事件映射到一个数字，代表这个事件发生的时间。形式一点来说，对于每个进程Pi，都有一个时钟Ci，这个时钟将该进程中的事件a映射到 i1a1。而整个系统的时钟，对于一个事件b，假设b属于进程Pj，那么C1b1 =Cj1b1。

这里插一句，从这个定义也可以看到大师对分布式系统的理解。分布式系统中不存在一个“全局”的实体。在该系统中，每个进程都是一个相对独立的实体，它们有自己的本地信息1本地Knowledge1。而整个系统的信息则是各个进程的信息的一个聚合。有了时钟的一个“本质定义”还不够，我们需要考虑，什么样的时钟是一个有意义的，或者说正确的时钟。其实，有了前文的1关系的定义，正确的时钟应满足的条件已经十分明显了：时钟条件：对于任意两个事件a，b，如果a 1 b，那么C1a1 ¡ C1b1。注意，反过来讲这个条件可不成立。如果我们要求反过来也成立，即“ 如果a 1 b为假，那么C1a1 ¡ C1b1也为假”，那就等于要求并发事件必须同时发生，这显然是不合理的。结合前文1关系的定义，我们可以把上面的条件细化成如下两条：

如果a和b是进程Pi中的两个事件，并且在Pi中，a在b之前发生，那么Ci1a1 ¡ Ci1b1；如果a是Pi发送消息m，b是Pj接收消息m，那么Ci1a1 ¡ Cj1b1；上面就定义了合理的逻辑时钟。显然，一个系统可以有无数个合理的逻辑时钟。实现逻辑时钟也相对简单，只要遵守两条实现规则就可以了：

每个进程Pi在自己的任何两个连续的事件之间增加Ci值；如果事件a是Pi发送消息m，那么在m中应该带上时间戳Tm=Ci1a1；如果b是进程Pj接收到消息m，那么，进程Pj应该设置Cj为大于max1Tm,Cj1b11。有了上面逻辑时钟的定义，我们现在可以为一个系统中所有的事件排一个全序，就是使用事件发生时的逻辑时钟读数进行排序，读数小的在先。当然，此时可能会存在两个事件同时发生的情况。如果要去除这种情况，方法也非常简单：如果a在进程Pi中，b在进程Pj中，Ci1a1 = Cj1b1且i ¡ j，那么a在b之前。形式化一点，我们可以把系统事件 E上的全序关系“=¿”定义为：假设a是Pi中的事件，b是Pj中的事件，那么：a =¿ b当且仅当以下两个条件之一成立：

Ci1a1 ¡ Cj1b1； Ci1a1 = Cj1b1 且 i ¡ j； Lamport把上面这些数理逻辑时钟的概念以非常直观地类比为顾客去面包店采购。面包店只能接待一位顾客的采购。已知有n位顾客要进入面包店采购，安排他们按照次序在前台登记一个签到号码。该签到号码逐次加 1。根据签到号码的由小到大的顺序依次入店购货。完成购买的顾客在前台把其签到号码归0. 如果完成购买的顾客要再次进店购买，就必须重新排队。这个类比中的顾客就相当于线程，而入店购货就是进入临界区独占访问该共享资源。由于计算机实现的特点，存在两个线程获得相同的签到号码的情况，这是因为两个线程几乎同时申请排队的签到号码，读取已经发出去的签到号码情况，这两个线程读到的数据是完全一样的，然后各自在读到的数据上找到最大值，再加1作为自己的排队签到号码。为此，该算法规定如果两个线程的排队签到号码相等，则线程id号较小的具有优先权。把该算法原理与分布式系统相结合，即可实现分步锁。注意这个系统中需要引入时钟同步，博主的意见是可以采用SNTP实现时钟同步1非权威，仅供参考1。

2.Paxos算法该算法比较热门，类似2pc算法的升级版，在此不做赘述，可以自行搜索相关资料。1博主会在之后整理列出1 需要注意的是这个算法也是Leslie Lamport提出的，由此可见这位大师之牛逼！ Paxos算法解决的问题是一个分布式系统如何就某个值1一致，每个节点都执行相同的操作序列，那么他们最后能得到一个一致的状态。为保证每个节点执行相同的命令序列，需要在每一条指令上执行一个“一致性算法”以保证每个节点看到的指令一致。一个通用的一致性算法可以应用在许多场景中，是分布式计算中的重要问题。节点通信存在两种模型：共享内存1Shared memory1和消息传递1Messag es passing1。Paxos算法就是一种基于消息传递模型的一致性算法。BigTable使用一个分布式数据锁服务Chubby，而Chubby使用Paxos算法来保证备份的一致性。不仅只用在分布式系统，凡是多个过程需要达成某种一致性的都可以用到Paxos 算法。一致性方法可以通过共

享内存1需要锁1或者消息传递实现，Paxos 算法采用的是后者。下面是Paxos 算法适用的几种情况：一台机器中多个进程/线程达成数据一致；分布式文件系统或者分布式数据库中多客户端并发读写数据；分布式存储中多个副本响应读写请求的一致性。

3. 采用乐观锁原理实现的同步我们举个例子说明该算法的实现原理。如一个金融系统，当某个操作员读取用户的数据，并在读出的用户数据的基础上进行修改时1如更改用户帐户余额1 ，如果采用前面的分布式锁服务机制，也就意味着整个操作过程中1从操作员读出数据、开始修改直至提交修改结果的全过程，甚至还包括操作员中途去煮咖啡的时间1 ，数据库记录始终处于加锁状态，可以想见，如果面对几百上千个并发，这样的情况将导致怎样的后果。乐观锁机制在一定程度上解决了这个问题。乐观锁，大多是基于数据版本1 Vers ion1记录机制实现。何谓数据版本？即为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是通过为数据库表增加一个“version” 字段来实现。读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据。对于上面修改用户帐户信息的例子而言，假设数据库中帐户信息表中有一个 versi on 字段，当前值为 1 ；而当前帐户余额字段1 balance 为 100 。

操作员 A 此时将其读出1version=1 1，并从其帐户余额中扣除 501100-1。在操作员 A 操作的过程中，操作员B也读入此用户信息1 versi on=1 1，并从其帐户余额中扣除 201100-1。操作员 A 完成了修改工作，将数据版本号加一1 version=2 1，连同帐户扣除后余额1 balance=1，提交至数据库更新，此时由于提交数据版本大于数据库记录当前版本，数据被更新，数据库记录 version 更新为 2 。操作员 B 完成了操作，也将版本号加一1 version=2 1试图向数据库提交数据1 balance= 801，但此时比对数据库记录版本时发现，操作员 B 提交的数据版本号为 2 ，数据库记录当前版本也为 2 ，不满足“提交作员 B 的提交被驳回。这样，就避免了操作员 B 用基于version=1 的旧数据修改的结果覆盖操作员A 的操作结果的可能。

数据库的分类和架构

非关系型数据库

NoSQL，泛指非关系型的数据库。随着互联网web2.0网站的兴起，传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS类型的web2.0 纯动态网站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则

由于其本身的特点得到了非常迅速的发展。NoSql数据库在特定的场景下可以发挥出难以想象的高效率和高性能，它是作为对传统关系型数据库的一个有效的补充。

NoSQL1NoSQL = Not Only SQL 1，意即“不仅仅是SQL”，是一项全新的数据库革命性运动，早期就有人提出，发展至2009年趋势越发高涨。NoSQL的拥护者们提倡运用非关系型的数据存储，相对于铺天盖地的关系型数据库运用，这一概念无疑是一种全新的思维的注入。

作者：小柑链接：http://www.jianshu.com/p/107c6b045245 来源：简书著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

键值存储数据库

键值数据库就类似传统语言中使用的哈希表。可以通过key来添加、查询或者删除数据库，因为使用key主键访问，所以会获得很高的性能及扩展性。

键值数据库主要使用一个哈希表，这个表中有一个特定的键和一个指针指向特定的数据。Key/value模型对于IT系统来说的优势在于简单、易部署、高并发。

典型产品：Memcached、Redis、MemcacheDB

作者：小柑链接：http://www.jianshu.com/p/107c6b045245 来源：简书著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

列存储数据库

列存储数据库将数据存储在列族中，一个列族存储经常被一起查询的相关数据，比如人类，我们经常会查询某个人的姓名和年龄，而不是薪资。这种情况下姓名和年龄会被放到一个列族中，薪资会被放到另一个列族中。

这种数据库通常用来应对分布式存储海量数据。

典型产品：Cassandra、HBase

作者：小柑链接：http://www.jianshu.com/p/107c6b045245 来源：简书著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

面向文档数据库

文档型数据库的灵感是来自于Lotus Notes办公软件，而且它同第一种键值数据库类似。该类型的数据模型是版本化的文档，半结构化的文档以特定的格式存储，比如JSON。文档型数据库可以看作是键值数据库的升级版，允许之间嵌套键值。而且文档型数据库比键值数据库的查询效率更高。

面向文档数据库会将数据以文档形式存储。每个文档都是自包含的数据单元，是一系列数据项的集合。每个数据项都有一个名词与对应值，值既可以是简单的数据类型，如字符串、数字和日期等；也可以是复杂的类型，如有序列表和关联对象。数据存储的最小单位是文档，同一个表中存储的文档属性可以是不同的，数据可以使用X ML、JSON或JSONB等多种形式存储。

典型产品：MongoDB、CouchDB

作者：小柑链接：http://www.jianshu.com/p/107c6b045245 来源：简书著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

图形数据库

图形数据库允许我们将数据以图的方式存储。实体会被作为顶点，而实体之间的关系则会被作为边。比如我们有三个实体，Steve Jobs、Apple和Next，则会有两个“F ounded by”的边将Apple和Next连接到Steve Jobs。

典型产品：Neo4J、InforGrid

作者：小柑链接：http://www.jianshu.com/p/107c6b045245 来源：简书著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

关系型数据库

1、关系型数据库的由来

虽然网状数据库和层次数据库已经很好的解决了数据的集中和共享问题，但是在数据库独立性和抽象级别上扔有很大欠缺。用户在对这两种数据库进行存取时，仍然需要明确数据的存储结构，指出存取路径。而关系型数据库就可以较好的解决这些问题。

2、关系型数据库介绍

关系型数据库模型是把复杂的数据结构归结为简单的二元关系1即二维表格形式1。在关系型数据库中，对数据的操作几乎全部建立在一个或多个关系表格上，通过对这些关联的表格分类、合并、连接或选取等运算来实现数据库的管理。

关系型数据库诞生40多年了，从理论产生发展到现实产品，例如：Oracle和MySQL，O racle在数据库领域上升到霸主地位，形成每年高达数百亿美元的庞大产业市场。

作者：小柑链接：http://www.jianshu.com/p/107c6b045245 來源：简书著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

2.1.4 Mysql数据库体系结构

因为 MySQL 采用的是客户机/服务器体系结构，所以在使用 MySQL 进行存取数据操作时，必须至少使用两个或者是两类程序： 111一个是位于存放数据的主机上的程序——数据库服务器。数据库服务器在网络上监听来自客户机的请求，然后根据客户机的这些请求访问数据库数据，访问之后再向客户机提供它们想得到的信息。 121连接到数据库服务器的程序——客户机，这些程序是作为用户和服务器之间交互信息的工具，并且告诉服务器需要查询信息的内容。 MySQL 的架构可以描述为层次性子系统组合。MySQL 的源代码不是按照单组件或者模块的方式编写的，但是各个层次的源代码还是能够被分离出来，大部分的子系统依赖于一些通用的底层库。MySQL 包含以下子系统：网络连接和网络通信协议子系统；线程、进程和内存分配子系统；查询解析和查询优化子系统；存储引擎接口子系统；各类存储引擎子系统；安全管理子系统；日志子系统；mysys 核心库文件等。当一个客户端通过网络连接 MySQL 数据库服务时，网络连接子系统执行一系列的与网络协议有关的底层任务。然后网络连接子系统将控制权交给线程子系统，线程子系统提供一个线程来处理这个连接，这个连接称之为连接线程。随后连接线程得到控制权，它首先调用安全管理子系统来验证用户访问的合法性。连接线程将获得的数据传给控制系统，其中一些请求在内核代码中被称作命令。这些命令中的一部分可以由这个控制系统直接完成，对于不可以直接由系统分发来完成查询的，分发系统将调用解析子系统对 SQL 语句进行解析。同时，如果在配置 MySQL 系统时采用了日志功能，那么分发系统还会调用日志系统去记录此次的信息。随后解析子系统将解析结果传给调用优化子系统以优化 SQL 语句。接着进行表操作，并将一系列请求发往存储引南京邮电大学硕士研究生学位第二章数据库系统的立体视图 11 擎接口子系统。存储引擎接口子系统将上述调用自动转化为某个具体的存储子系统方法。上述过程完成后，相应的模块将 SQL 执行结果发往客户端，最后再由服务器将控制权交给连接线程，连接线程完成某些清理工作，并在此等待客户端的连接或者其他查询，直到客户端输入 Quit 命令为止，到此本次通话才会结束。

2.2 Mysql通信协议的研究

在MySQL数据库通信过程开始时，服务器会使用TCP监听一个本地socket 端口或本地socket链接。当一个客户端的连接请求到达，就会执行握手和权限验证。如果验证成功，会话开始。客户端发送消息，服务器会以一个适合该发送命令的数据类型的数据集或一条消息进行回复。当客户端发送完成后，会发送一个特殊的命令，告诉服务器已发送，然后会话结束。通信的基本单位是应用程序包。多个指令可以合成一个包；答复可以包含多个包。

2.2.1 交互过程

MySQL客户端与服务器的交互主要分为两个阶段：握手认证阶段和命令执行阶段。

握手认证阶段

握手认证阶段为客户端与服务器建立连接后进行，交互过程如下：服务器客户端：握手初始化报文客户端服务器：登陆认证报文服务器客户端：认证结果报文

命令执行阶段

客户端认证成功后，会进入命令执行阶段，交互过程如下：客户端服务器：执行命令报文服务器客户端：命令执行结果 MySQL客户端与服务器之间的完整交互过程如图3-1所示

2.2.2 协议基本类型

整型值

MySQL报文中整型值分别有l、2、3、4、8字节长度，使用小字节序传输。

null结尾的字符串

字符串长度不固定，当遇到fNULL’1Ox001字符时结束。

长度编码的字符串

字符串长度不固定，无null结束符，编码方式与上面的Length Coded Binary相同。

二进制数据1长度编码1

数据长度不固定，长度值由数据前的1-9个字节决定，其中长度值所占的字节数不定，字节数由第1个字节决定，如表3．1：表3—1二进制数据编码表第一个字节值后续字节数长度值说明 0—250 0 第一个字节值即为数据的真实长度 251 0 空数据，数据的真实长度为零万方数据华北电力大学硕士学位论文 252 2 后续额外2个字节标识了数据的真实长度一 253 3 后续额外3个字节标识了数据的真实长度 254 8 后续额外8个字节标识了数据的真实长度

2.2.3 mysql报文结构

报文分为报文头和报文数据两部分，其中报文头占用固定的4个字节，报文数据长度由报文头中的长度字段决定，报文结构如图3．2： 3Bytes l Bytes nBytes 图3-2 MySQL协议报文结构

报文头

．3．1．1 报文长度用于标记当前请求报文的实际数据长度值，以字节为单位，占用3个字节，最大值为0xFFFFFF，即接近16 MB大小1比16MB少1个字节1。3．3．1．2 序号在一次完整的请求／响应交互过程中，用于保证报文顺序的正确，每次客户端发起请求时，序号值都会从0开始计算。

报文数据

报文数据用于存放请求的内容及响应的数据，长度由报文头中的长度值决定。

2.2.4 报文类型

登陆认证交互报文

．4．1．1握手初始化报文1服务器．¿客户端1 建立TCP连接后，MySQL服务器向客户端

发送握手初始化包1Handshake Initialization Packet1，对服务器的权能进行设置并等待客户端的验证包。初始化报文格式如图3．3所示：

客服端命令请求报文

客户端命令请求报文的报文头与上述一致。其报文数据分为两个部分，分别是命令和参数，如图3．5所示：

服务器响应报文

JAVA网络和多线程技术

TCP/IP相关协议

TCPIP协议全称Transmission Control ProtocolInternet Protocol，中译名为传输控制协议因特网互联协议，又名网络通讯协议，是Internet最基本的协议、 Internet国际互联网络的基础。

TCP/ip协议概述

TCPIP是一个协议集合，寻址和路由选择以及传输控制是它的核心功能。HTTP协议的基础是TCPIP协议，HTTP实现了服务器与客户端间的请求与响应， TCPIP则实现了两端底层的数据传输‘36。37】。TCPIP协议的体系结构模型如图2．1 所示：

图2．1 TCPIP协议模型物理层和链路层：负责处理操作系统的设备驱动程序或网卡等终端与网络之间的接口细节。

网络层：网络层的协议包括IP协议、IGMP1因特网组管理协议1、ARP1地址解析协议1、ICMP1因特网控制报文协议1、RARP1反向地址解析协议1。其中IP协议向其它高层协议提供了基本的数据传输的功能，是无连接的、不可靠数据报协议。传输层：TCP和UDP协议都使用IP网络层协议，是传输层的两个著名的协议。为客户端和服务器上的应用程序提供端到端的通信。虽然TCP使用了不可靠的IP网络层协议，但它却可以为主机间提供可靠的、面向连接的传输层数据通信
8

工程硕士学位论文服务。与TCP不同的是，UDP为主机间提供无连接的、不可靠的传输层数据通信。由于UDP协议开销很小，因此在特定领域也有着广泛的应用应用层：用于处理应用程序的细节，这一层有许多协议，如Http、FTP、TELNET 世 i手。

TCP通信的连接和终止

连接建立、数据传送和连接终止是TCP连接的三个状态。TCP使用三次握手1three．way handshake1协议来建立连接，终止一个连接要经过4次握手。在TCP建立连接的过程中，将会初始化很多参数，例如对报文序号的初始化来保证连接的强壮性和按序号传输。

111建立连接：在有些情况下会出现一对终端同时初始化一个他们之间的链接的情况，但通常是由服务器端被动打开一个套接字1socket1监听来自客户端的连

接。客户端发送一个SYN报文段指明需要连接的服务器的端口，以及初始顺序号来建立一个主动打开，作为三路握手的一部分。服务器发回包含服务器的初始顺序号的SYN报文段1SY'N为11作为应答。同时，将确认号设置为客户的ISN加1以对客户的SYN报文段进行确认1ACK也为11。服务器端接收到来自客户端的SYN后一般会验证客户端的合法性，为每个合法的客户端发送一个SYN／ACK。最后，客户端接收到服务器的响应后再发送一个ACK。这样三路握手就完成了进入链接建立的状态。

121数据传输：为保证在TCP的数据传送状态下的可靠性和强壮性，主要采取了一下机制，它们包括：对收到的TCP报文采用序号进行排序来检测重复数据；采用校验和对报文段的错误进行检测；利用计时器对丢包或延时进行检测和纠正。在的连接已经建立的状态下，两个主机的TCP层互相交换初始序号1ISN1。初始序号除了能够对字节流中的数据进行标识以外，还可以用来对应用层的数据字节进行记数。通常情况下

，序号和确认号存在于每个TCP报文段中。报文的发送者将自己的字节编号称为序号，将接收者的字节编号称为确认号。为了保证报文的可靠性，接收者在接收到一定数量的连续字节后才发送确认，这种被称为选择确认的机制1SACK1是TCP的一种扩展。当数据以乱序到达接收者端时，如果没有选择确认的机制，报文中的字节就不能按正确的顺序传递给应用层。 131链接终止：TCP用三个分节建立一个连接，终止一个连接则需要四个分节，在标准的拆接过程中，链接的每一端都要提供一个FIN和ACK对。

java流IO技术

在Java的I／O类库中，“流”这一抽象的概念，通常用来表示接收数据的接收端对象或是产生数据的数据源对象。“流”有效地封装了I／O中的实现细节，是 9 NIO高性能框架的研究与应用 Java平台I／O的基础。

输入输出流

就流的运动方向而言，流可以分为输入流1InputStream1和输出流 1OutputStream1，输入流代表从外设流入计算机的数据序列，而输出流则代表从计算机流向外设的数据序列。在Java平台中，所有继承于InputStream的与输入有关的类和所有继承于OutputStream与输出相关联的类，分别称为Java的I／O类库中的输入和输出，是组成JavaI／O类库的两个主要组成部分。从不同数据源产生输入的类用InputStream来表示。这些数据源包括：

111字节数组； 121String对象； 131文件； 141“管道”，工作模式类似于实际的管道，从一端输入，从另一端输出； 151序列； 161其他数据源，例如网络连接等。

abstract int read11是InputStream中的一个最主要的抽象方法方法该方法会返回从输入源中读到的一个字节，当返回结果为一1时，表示已经读到了输入源的末尾。设计具体的输入流类的时候，设计者需覆盖这个方法来提供详细的功能实现。OutputStream决定数据输出的各种不同的目的地，这些目的地包括字节数组、文件或管道等。与InputStream类似，OutputStream也提供了一个抽象方法abstract int write1int b1交由设计者去覆盖实现，这个方法从要输出的数据中的一个字节写到指定的目的地【4们。

阅读器和书写器

Javal．1版本对基本的I／O流类库进行了极大程度的修改。一开始看到Reader1阅读器1和Writer1书写器1时，可能会误认为它们是用来替代InputStrearn 和OutputStream，但实际上不是这样的。InputStream和OutputStream仍然非常有价值，特别是在面向字节形式的I／O中，Reader和Writer则为Unicode和面向字符的I／O功能提供兼容。当需要把面

向字节形式的I／O中的类和向字符形式的I／O 中的类结合起来使用时，“适配器”1adapter1类可以起到中间桥梁的作用。在“适配器”类中InputStreamReader能够将InputStream转换成Reader，而 OutputStreamWriter能够将OutputStream转换成Writer。Reader和Writer继承层次结构的设计是为了满足国际化的需求。

Unicode是国际组织制定的可以容纳世界上所有文字和符号的字符编码方案，由于在I／O流的继承层次结构中只能够支持8位的字节流，当遇到16位的Unicode字符时处理起来很不方便。所以Javal．1 版本添加了Reader和Writer的继承层次结构为了来支持Unicode编码【411。

技术特征

流I／O技术数据的输入输出时面向字节的，输入流每次一个一个字节地从数据源读取数据，输出流也是每次向输出流一个一个字节的写入数据。相对于面向块的数据读取而言，通过这种方式进行的数据读写速度很慢。同时，流I／O采用的是阻塞的调用方式。在真正的写入或读取操作完成以前，工作线程都会被阻塞。这意味着在复杂的网络环境中，由于网络连接繁忙而导致数据不能被流立即读取或写入时，Java就会挂起这个工作线程，一直使其陷入等待的状态，直到流再次可用为止【4引。

Socket编程技术

简介

Socket套接字是网络上服务器端与客户机端之间进行双向通信的一方，可以发送或接受连接请求，Socket将通信双方一端写入的信息发送至另外一端的Socket中，利用Socket套接字可以方便地进行数据的传输。类似于文件传输的输入输出流原理，Socket套接字通信可以通过读写数据实现对网络资源的使用【431。Java中提供了两种套接字方式，分别为流式Socket和数据报式Socket，二者的比较如表2．1：表2．1流式与数据报式Socket类型比较流式Socket通信服务的特点有：面向连接、可靠性高、有序、无差错、无重复、可移植性好，支持TCP协议，可实现不同类型计算机之间的通信。

通信原理

在客户机／服务器模式中，可以把Socket实例看作一个特殊的实例对象，用于描述IP地址和端12，是一个通信链的句柄。客户机端流套接字为Socket对象，服务器端流套接字为ServerSocket对象，这两个对象中都封装了两个方法，分别是输入流getlnputStream0和输出流getOutputStream0，是通信过程中实现Socket 通信连接的关键方法。Socket实现网络通信的流程如图2．2所示：NIO高性能框架的研究与应用客户端调用9Dck烈O创建一个会话调用oonnectO与咖端连接调用recvO／send0进行会话 aiseS：x：ketO关闭套接字服务器调用9xket仓]建监听调用bjndO为监听 9:，cket选择通信对象调用list 调用ao：eptO接收连接同时生成会话

三次握手，建立连接请求连接数据应答调用recv11／sendO进行会话 aosegx：ketO关闭套接字图2．2 Socket通1言机制根据图2．3所示原理，PC机客户监控端与服务器端进行通信传输时，首先服务器端创建Socket，调用bind11为监听Socket选择通信对象，同时调用listen11 和accept11等待客户端请求连接，客户端创建Socket并调用connect11尝试与服务器端连接，然后开始调用read11和write11与服务器端数据进行数据传输，连接建立完成。

技术实现

服务器端程序编写：Java中使用专门建立Sock et服务器的类ServerSocket来创建服务器对象，采用端口号作为传递参数，且端口号是为了唯一标识每台PC机的唯一服务的41。在服务器端建立Socke t通信连接的步骤如下： 111首先在服务器端创建Socket并绑定到端口上； ServerSocket server=new ServerSocket1int port1；121不停监听客户机的连接请求，一旦监听到客户机连接请求后，服务器调用accept11函

数并接受连接请求，完成S ocket通信连接的建立； Socket s=SS．accept11； 131服务器端调用Socket类的输入输出函数以获取输入输出流，并利用输入输出流进行数据的传输。 OutputStream OS=S．getOutputStream11； ／／创建输出流 InputStream is=s．getlnputStreamO； ／／仓JJ建输入流 OS．write1”Hello，welcome you!”．getBytes111； byte[】aa=new byte[100]； ／／建立字节数组 12 工程硕士学位论文 int len=is．read1a1； ／／读取数据到数组中并返回实际读取的字节数 System．out．print1new String1a，0，1en11； 141关闭通信套接字。客户机端和服务器端的类似： 111首先在客户机端创建Socket流套接字，并连接到服务器端； Socket s=new Socket1InetAddress．getByName1null1，port1； 121客户机端调用Socket对象的输入输出函数以获取输入输出流，并利用输入输出流进行数据的传输。 OutputStream OS=s．getOutputStreamO； InputStream is 2s．getInputStreamO； byte[】a=new byte[100]；int len=is．read1a1； ／／从服务器端读取数据 System．out．print1new String1a，0，len11； OS．write1”Hello，this is mengmeng”．getBytes01； 131关闭通信套接字。

NIO技术

在JDKI．4提出的新特性中，NIO技术是最大的亮点。与传统的流I／O技术相比，NIO主要带来了两点改进：1．弥补了原来的I／O的不足，提供了面向块的、非阻塞的I／O处理能力。2．使得Java应用程序能够更加充分地发挥操作系统的性能，进行高性能的I／O操作。通道、缓冲区和选择器是NIO中三个最为关键的内容【46-471。

缓存区和通道

缓冲区是一个数据容器，缓冲区对象由一个用来存储数据的数组和一系列控制数据读写的属性组成。在NIO技术中，所有数据的传输都是通过缓冲区来完成的，这体现了与流I／O技术的一点主要的不同。在流I／O技术中，数据时直接读写至1]Stream对象中的，而在NIO技术中，所有数据都是从缓冲区来读出和写入。所有的缓冲区都具有四个属性来提供关于其所包含的数据元素的信息，它们分别是： 111容量1Capacity1：缓冲区能够容纳的数据元素的最大数量。这一容量在缓冲区创建时被设定，并且永远不能被改变。 121上界1Limit1：缓冲区的第一个不能被读或写的元素。或者说，缓冲区中现存元素的计数。 131位置1Position1：下一个要被读或写的元素的索引。位置会自动由相应的get11和put11方法更新。141标记1Mark1：一个备忘位置。调用mark11来设定mark=postion。调用reset11设

定position=mark。标记在设定前是未定义的1undefined1。这四个属性之间总是遵循以下关系：0¡=mark¡=position¡=limit¡= capacity。缓冲区结构图如图3．1所示：17 NIO高性能框架的研究与应用已经读／写尚未读，写超过限制■。标志位置限制容量图3．I缓冲区结构图通道1Channel1是java．nio的第二个主要创新。它们既不是一个扩展也不是一项增强，而是全新、极好的Java I／O示例，提供与I／O服务的直接连接。Channel 用于在字节缓冲区和位于通道另一侧的实体1通常是一个文件或套接字1之间有效地传输数据。通道与流相比不同之处在于通道是双向的，由于它是双向的，所以通道比流能够更好地反映底层操作系统的真实情况。

多路复用对象

选择器维护着注册过的通道的集合，并且这些注册关系中的任意一个都是封装在SelectionKey对象中的。选择键封装了特定的通道与特定的选择器的注册关系。选择键对象被SelectableChannel．register11返回并提供一个表示这种注册关系的标记。选择键包含了两个比特集1以整数的形式进行编码1，指示了该注册关系所关心的通道操作，以及通道已经准备好的操作。每一个Selector对象维护三个键的集合： 111已注册的键的集合1Registered key set1：与选择器关联的已经注册的键的集合。 121已选择的键的集合1Selected key set1：已注册的键的集合的子集。这个集合的每个成员都是相关的通道被选择器1在前一个选择操作中1判断为已经准备好的，并且包含于键的interest集合中的操作。 131已取消的键的集合1Cancelled key set1：已注册的键的集合的子集，这个集合包含了cancel11方法被调用过的键1这个键已经被无效化1，但它们还没有被注销。选择器提供选择执行已经就绪的任务的能力，这使得多元I／O成为可能。就绪选择和多元执行使得单线程能够有效率地同时管理多个I／O通道1channels1。 C／C++代码的工具箱中，许多年前就已经有select11和poll11这两个POSIX1可移植性操作系统接口1系统调用可供使用了。许过操作系统也提供相似的功能，但对Java程序员来说，就绪选择功能直到JDK 1．4才成为可行的方案。对于主要的 18 工程硕士学位论文工作经验都是基于Java环境的开发的程序员来说，之前可能还没有碰到过这种 I／O模型。

技术特征

数据打包和传输的方式是流I／O技术与NIO技术的一个重要的区别。流I／O技术是面向字节的，而NIO技术通过块的方式来进行数据读写。在流I／O技术中，数据的读写以一次一个字节地进行，这种处理方式I／O字节流必须顺序读取，处

理速度比较慢，经常用于间歇性输入。而在向块的I／O系统中，每一次操作都消费一个数据块，比一个一个字节的读写数据要快得多。 NIO对非阻塞I／O操作的支持是其另外一个重要的特征。在NIO包中的多路复用对象1Selector1提供选择执行已经就绪的任务的能力，这使得非阻塞I／O成为可能。就绪选择使得单线程能够有效率地同时管理多个I／O通道1channels1。 C／C++代码的工具箱中，许多年前就已经有select11和poll11这两个POSIX1可移植性操作系统接口1系统调用可供使用了。许过操作系统也提供相似的功能，但对Java程序员来说，就绪选择功能直到JDK 1．4才成为可行的方案。

线程池技术

计算机技术在经历了几十年的发展历程之后，已步入网络时代，各种分布式应用随处可见，这些应用对系统的响应速度、稳健性和整体性能都提出了较高的要求。线程池技术是满足这些要求而采用的技术之一，它是组织服务器应用的有效工具，可以提高系统的响应速度和整体性能。

传统的线程池模型

主从模型【4引，工作组模型【491，和管道模型[501是三个传统的线程池模型。在工作组模型中，线程池中的线程相互协作地为用户请求提供服务，它们的功能和地位都相同；主从1Master／Slaves1模型的特点在于其采用主从式结构的设计，将线程池中少量的线程设为Master线程，其余的设为Slaves线程。Master线程处于管理者的地位，Slaves线程处于被管理的地位，在接受和处理用户的请求时，由 Master线程来选择Slaver线程进行处理；在管道1Pipeline1模型中，下一步工作线程的输入要依赖上一步线程的输出，线程池中的所有线程的输入和输出相互衔接形成一个“管道”，当用户请求需要经过多个步骤处理时通常会应用到该模型。Master／Slaves模型相对于其它两个常见的模型而言，具有可管理、可移植、易于开发部署的特性，在应用研究和项目开发中使用较多。论文第四章NIO高性能框架详细设计中涉及的线程池模型的设计采用的就是主从模型。

线程池资源调度问题

虽然线程池技术可以有效地避免线程的频繁创建和销毁所带来的性能损耗，但线程池使用的过程中也伴随着诸多的风险。应用线程池技术构建的应用很容易发生同步错误和死锁等并发风险，还会产生一些其它的风险，如线程泄漏和资源不足等【511。当线程池大小调随应用的具体状况整到合适大小时，才能高效的运行。线程占用着大量的系统资源，每个线程都需要独立分配的内存空间和两个很

大的执行调用堆栈，与此同时在线程数很多的情况下，线程间相互切换带来的调度开销也不容忽视。因此当线程池很大时，所有线程消耗的资源将会严重地损害系统的性能。如何实现线程池中的线程的数量和有限的资源之间的平衡，提高系统线程池的性能是至关重要的。论文的第四章的线程池模型设计中致力于解决这个问题。

网络应用中线程池的研究

当前，服务器厂商所开发的Web容器或Web应用服务器中都应用的线程池技术，例如由BEA公司开发的WebLogic服务器【5 2’、Apache组织的tomcat、IBM的WebSphere服务器【53】等，下面对WebLogic和WebSplere的线程池运行机制作出一些简单的介绍。在WebLogic Server9．0前面的版本中，都是使用多个队列来进行请求的处理。根据优先级顺序将不同类型的请求在不同的队列中运行，这样的处理可以很好的降低发生死锁的风险。而在WebLogic Server9．0以后的版本为任务的执行分配单独的线程池来处理。其线程池大小的动态调整建立在默认的队列对吞吐量监控的基础上，默认的队列会记录下吞吐量的历史数据来动态地调整线程池中的线程数量从而使服务器达到最大的吞吐量。例如，当线程池中大量线程的吞吐量都增加时， WebLogic Server会增加线程数；类似的，如果历史记录显示大多数线程的吞吐量都降低时，WebLogic Server会减少线程数量。这个有效地避免了在配置、跟踪和调整自定义执行队列的的复杂性，使管理程序分配处理器资源更加容易。 Webspherefl艮务器的线程池实现具有高性能和高伸缩性的特点。异步Bean的 WorkManager和Commonj都使用此线程池。由于WorkManager实例应用于全局命名空间并且可以在多个应用之间共享，因此需要切换Web应用程序上下文。为完成此任务，WorkManager在提交请求时会获取线程上web应用程序的上下文，产生的对象为WorkWithExecutionContext1WWEC1对象。Websphere Application ServerV4．0 和更新的版本中，线程池定义了反映池大小的限制的最大值和最小值。一般来说，大容量的应用程序有较大的线程池大小，但是有些因素1例如应用程序瓶颈、 synchronized关键字的使用1将阻碍大线程池的高效利用。如果线程池大小没有足够的可用线程用于在合理的时间内完成客户请求，那么线程池的优化就需要对服 20 工程硕士学位论文务器进行调试，以达到堆大小和可用线程数之间一个好的平衡。综上所述，线程池技术能够很好地解决大量线程的资源不足问题和生命周期开销问题，能够极大地提高Web应用服务器的性能。但是也存在着许多有待解决的应用风险，其中有限的资源和线程数量之间的矛盾最为突出。如何分配合适的线程数量来处理用户的请求，最大限度地提高Web应用服

务器的性能是研究的重点。例如，在WebLogic Server9．0后面的版本仅仅使用历史吞吐量的大小动态调整线程池中的线程数量，这种方式忽视了其它影响性能的指标使调整结果不够精确；而WebSphere需手动地更改配置文件来调整线程池的大小，在应用运行的过程中不具备动态调整的能力。

Reactor模式

Reactor模型是一个事件触发模型，当有I／O读写操作准备就绪时，触发操作，然后从线程池中选择线程进行处理。这种机制能够极大地减少线程浪费在等待数据读写操作准备就绪上花费的时间，提高Java网络应用的并发性能。

模型思想

分而治之是Reactor模型的模型思想，在Reactor模型中，客户端的请求事件分为I／O事件和非I／O事件。这两种事件的区别在于I／O事件只有当I／O准备就绪后才能进行，而非I／0事件是可以立即执行的，因此Reactor模型将这两种事件分别进行处理。数据的读写属于I／O事件，编码、计算和解码属于非I／O事件。NIO 技术使得Java具备了非阻塞的读写能力，为这种Reactor模型的实现提供了很好的支持。

模型机制

线程池技术和NIO技术是Reactor模型采用的两个主要技术。NIO类库中提供的选择器Selector具有选择准备就绪的I／O事件的能力，当某条连接的I／0操作准备就绪时，由Selector将任务分发给线程池中的线程来处理。Reactor模型的工作机制图如图3．2所示： 21 NIO高性能框架的研究与应用图3．2 Reactor模型工作机制图 mainReactor对象称为主选择器，主要负责监听客户端的链接。通过调用服务器套接字通道1ServerSockctChannel1的register方法将通道注册到主选择器上，接收客户端链接是服务器套接字通感兴趣的事件，当有连接到达服务器时， Server-SockctChannel调用accept方法，将返回的连接通道注册到subReactor1子选择器1上并绑定通道所感兴趣的数据输入事件。一旦子选择器检测到输入数据准备就绪时，便开始执行read操作，I／O操作执行完毕后subReactor从线程池中选取线程来进行编码、计算、解码等非I／O操作。当非I／O操作执行完成，数据需要写到客户端时候，subReactor将输出注册为该链接通道所感兴的趣的事件，一旦subReactor检测到网络输出缓冲区能够接收数据时，调用通道的write方法将响应发送给客户端。

模型分析

1．Reactor模型的优势首先，与传统的开发模型相比，Reactor模型不会为每一个客户端链接分配一个线程来处理所有操作，而是利用选择器的选择就绪的功能，反应式地进行I／O 操作，当I／O读写完成以后，选择线程池中的线程来处理非I／O事件。这种处理方式能够省去线程等待I／O事件准备就绪花费的时间1图2．4中的t1、t61，增加了线程的利用效率。其次，应用线程池技术构建的Reactor模型能够减少大量线程上下文切换带来的系统开销，同时也减少了线程的生命周期处理以及管理线程的开销的开销。最后。NIO技术能够实现面向块的数据读取，与流式I／O面向字节的特性相比，能够缩短了实际读写所需的时间1图2．4中的t2、t71。 2．Reactor模型的不足按照1Unix网络编程》的划分，I／O模型可以分为：阻塞I／O、非阻塞I／O、工程硕士学位论文 I／O复用、信号驱动I／0和异步I／O，按照POSIX标准来划分只分为两类：同步I／O 和异步I／O。一个I／O操作实际上分成了两个步骤：发起I／O请求和实际的I／0操作，同步I／O和异步I／0的区别就在于第二个步骤是否阻塞，如果实际的I／0读写阻塞请求进程，那么就是同步I／O，因此阻塞I／O、非阻塞I／O、I／0复用、信号驱动I／O都是同步I／O，如果实际的I／O读写操作不阻塞请求进程，而是操作系统帮你完成I／O读写，将数据从内核内存区读取到用户空间再将结果返回给你，那么就是异步I／O。阻塞I／O和非阻塞I／0的区别在于第一步，发起I／O请求是否会被阻塞，如果阻塞直到完成那么就是传统的阻塞I／O，如果不阻塞，那么就是非阻塞I／O。因此，基于Reactor模型的NIO框架做到的仅仅是发起I／O请求时不阻塞，具体的I／O读写是又相应的事件派发处理器交给用户线程来处理的。五个I／O 模型的比较如图3．3所示：阻塞I，O 非阻塞I，O l，O复用信号驱动I／C 异步I／0 发起检查构查发起检查阻笔检查

：成 t苎，成通知了C 7c，兀 1～～“　～一”　5”　”’”　1　’’一”　一“””” -F　；”　、。、。一’、“‘1* 第一阶段处理不同，处理两个第二阶段处理相同阶段图3．3五个I／O模型的比较基于Reactor模型的NIO框架具体的不足表现为：首先，mainReactor对象和 subReactor对象是基于NIO中的选择器实现的，为了实时更新链接通道的读写状态，Selector需要不断调用select11方法来更新注册在这个选择器上通道的 SelectionKey，来确定下一步的操作，两级Select的设置大大地增加了系统的线程开销，拖累系统的并发处理性能和网络吞吐量。其次用户进程处理I／O读写受网络的不确定性影响很大，读到0个字节或者写入0个字节在高负载或者网速慢的情况下经常出现，常用的解决办法是重新注册该Channel的OP READ或 OP WRITE时间等待下一次处理，这样无形中增加的线程的切换开销和

系统调用开销，影响了框架的性能。

Netty简介

一个应用想要支持成千上万并发的客户端，在以前，这样的想法会被认为是荒谬。而在今天，我们认为这是理所当然的。事实上，开发者知道，总是会有这样的需求——以较低的成本交付来换取更大的吞吐量和可用性。我们不要低估最后一点的重要性。我们从漫长的痛苦的经验学习到，低级别的 API 不仅暴露了高级别直接使用的复杂性，而且引入了过分依赖于这项技术所造成的短板。因此，面向对象的一个基本原则：通过抽象来隐藏背后的复杂性。这一原则已见成效，框架的形式封装解决方案成为了常见的编程任务，。现在大多数专业的 Java 开发人员都熟悉一个或多个这些框架1比如 Spring1，并且许多已成为不可或缺的，使他们能够满足他们的技术要求以及他们的计划。

Netty 是一个广泛使用的 Java 网络编程框架1Netty 在 2011 年获得了Duke's Cho ice Award，见https://www.java.net/dukeschoice/20111。它活跃和成长于用户社区，像大型公司 Facebook 和 Instagram 以及流行开源项目如 Infinispan, HornetQ, Vert.x, Apache Cassandra 和 Elasticsearch 等，都利用其强大的对于网络抽象的核心代码。反过来，Netty 也从这些开源项目中获益。随着这些项目的作用，Netty 也不断提高了其应用的范围和灵活性，比如已经实现了的协议就有 FTP, SMTP, H TTP, WebSocket 和 SPDY 以及其他二进制和基于文本的协议。在初创公司中 Firebase 和 Urban Airship 在使用 Netty。前者 Firebase 是使用 long-lived HTTP 连接，后者是使用各种推送通知。当你使用 Twitter,你会使用 Finagle,这个是基于 Netty API 提供给内部系统通讯。Facebook 使用 Netty 来提供于 Nifty 类似的功能 Apache Th rift 服务。这些公司可扩展性和高性能的表现得益于 Netty 的贡献。这些例子的真实案例会在后面几章讲到。 2011 年 Netty 项目从 Red Hat 独立开来从而让广泛的开发者社区贡献者参与进来。Red Hat ， Twitter 继续使用 Netty ,并且成为保持其最活跃的贡献者之一。下面展示了 Netty 技术和方法的特点设计针对多种传输类型的统一接口 - 阻塞和非阻塞简单但更强大的线程模型真正的无连接的数据报套接字支持链接逻辑支持复用易用性大量的 Javadoc 和代码实例除了在 JDK 1.6 + 额外的限制。1一些特征是只支持在Java 1.7 +。可选的功能可能有额外的限制。1 性能比核心 Java API 更好的吞吐量，较低的延时资源消耗更少，这个得益于共享池和重用减少内存拷贝健壮性消除由于慢，快，或重载连接产生的 OutOfMemoryError 消除经常发现在 NIO 在高速网络中的应用中的不公平的读/写比安全完整的 SSL / TLS 和 StartTLS 的支持运行在受限的环境例如 Applet 或 OSGI 社区发布的更早和更频繁社区驱动

所有的网络应用程序需要被设计为可扩展性，可以被界定为“一个系统，网络能力，或过程中能够处理越来越多的工作方式或可扩大到容纳增长的能力”1见Bondi, André B. 120001. ”Ch aracteristics of scalability and their impact on performance”1。我们已经说过，Netty 帮助您利用非阻塞 I/O 完成这一目标，通常称为“异步 I/O” 我们将使用“异步”和其同源词在这本书中大量的使用，所以这是介绍他们的一个很好的时候。异步，即非同步事件，当然是跟你日常生活的类似。例如，您可以发送电子邮件；可能得到或者得不到任何回应，或者当你发送一个您可能会收到一个消息。异步事件也可以有一个有序的关系。例如，你通常不会收到一个问题的答案直到提出一个问题，但是你并没有阻止同时一些其他的东西。在日常生活中异步就这样发生了，所以我们不会经常想到。但让计算机程序的工作方式，来实现我们提出了的特殊的问题，会有一点复杂。在本质上，一个系统是异步和“事件驱动”将会表现出一个特定的，对我们来说，有价值的行为：它可以响应在任何时间以任何顺序发生的事件。这是我们要建立一种制度，正如我们将会看到，这是典范的 Netty 自底向上的支持。

2.3 本章小结

本章主要介绍了本论文相关的关键的理论和技术，包括分布式数据库方面的理论，mysql协议的研究和学习，还有java相关的理论和技术。

第三章 需求分析

本章主要写了又那些数据库，每种数据库的特点。

3.1 系统概述

3.2 需求分析

3.3 通信协议分析

3.4 编程模型分析

..

3.5 本章小结

存储部分的数据库主要分为两大类：关系型数据库与 NoSQL 数据库。

...

第四章 系统设计

4.1 总体架构设计

jsql是一个可以替代mysql的分布式java数据库

...

4.2 系统模块划分

下面是分布式环境下的部署图：

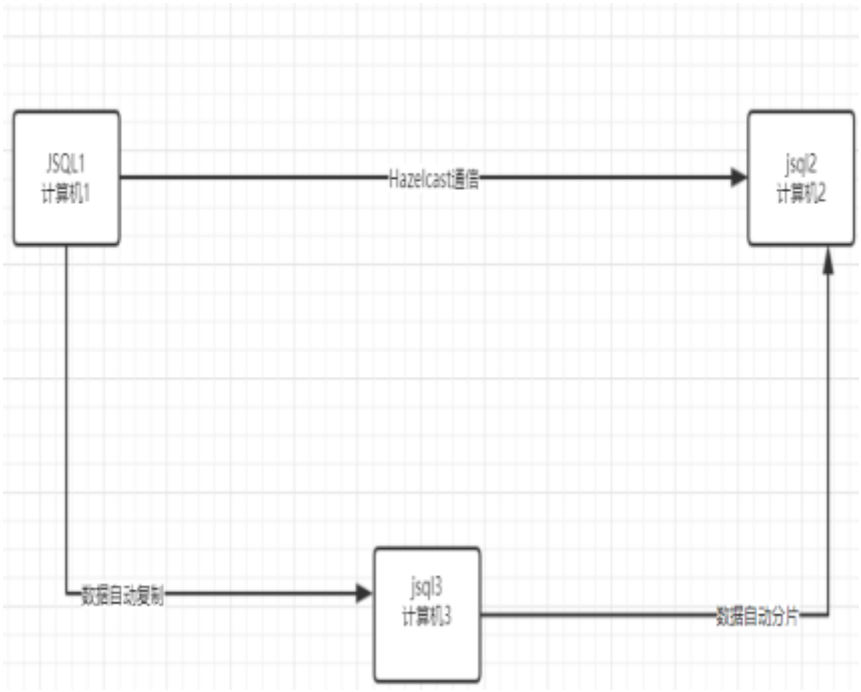

图 4-1 分布式环境下架构

图6-2是jsql的架构图

4.2.1 数据库模块设计

数据库主要是sqk到粗存

...

4.2.2 集群架构设计

主要用hazelcast

...

4.2.3 数据审计模块设计

数据库审计的

...

elasticsearch简介

...

grafana简介

...

审计的架构

...

...

4.3 各模块详细设计

本文主要是

...

4.3.1 数据库模块设计

主要分为3个部分

...

网络模块的设计

数据库主要是sqk到粗存

...

sql解析模块的设计

主要用hazelcast

...

存储引擎模块的设计

数据库审计的

...

4.3.2 集群架构的设计

主要用hazelcast

数据库审计的

...

4.3.3 数据审计架构的设计

...

审计数据库设计

数据库主要是sqk到粗存

...

审计管理器设计

主要用hazelcast

...

审计可视化模块的实现

数据库审计的

4.4 本章小结

分布式数据库

...

第五章 系统实现

本文主要是

...

5.1 数据库模块实现

主要分为3个部分

...

5.1.1 网络模块

数据库主要是sqk到粗存

...

5.1.2 sql解析模块

主要用hazelcast

...

5.1.3 存储引擎模块

数据库审计的

...

5.2 集群架构的实现

主要用hazelcast

...

5.2.1 通信模块

数据库主要是sqk到粗存

...

5.2.2 数据复制

主要用hazelcast

...

5.2.3 负载均衡

数据库审计的

...

5.3 数据审计架构的实现

...

5.3.1 审计数据库

数据库主要是sqk到粗存

...

5.3.2 审计管理器

主要用hazelcast

...

5.3.3 审计可视化模块的实现

数据库审计的

...

5.4 本章小结

分布式数据库

...

第六章 系统测试

...

6.1 测试环境

...

6.2 功能和性能测试

...

6.2.1 基本功能测试

……

6.2.2 数据库集群测试

……

6.3 数据库审计和可视化测试

...

6.4 本章小结

...

致 谢

在攻读硕士学位期间，首先衷心感谢我的导师曹晟老师...

参考文献

[1] 王浩刚, 聂在平. 基于mysql的分布式数据库的研究和实现[J]. 电子学报, 1999, 27(12):68–71

[2] X. F. Liu, B. Z. Wang, W. Shao. A marching-on-in-order scheme for exact attenuation constant extraction of lossy transmission lines[C]. China-Japan Joint Microwave Conference Proceedings, Chengdu, 2006, 527–529

[3] 竺可桢. 物理学[M]. 北京: 科学出版社, 1973, 56–60

[4] 陈念永. 基于mysql的分布式数据库的研究和实现[D]. 成都: 电子科技大学, 2001, 50–60

[5] 顾春. 基于mysql的分布式数据库的研究和实现[N]. 人民日报, 2012年3月31日

[6] 冯西桥. 基于mysql的分布式数据库的研究和实现[R]. 北京: 清华大学核能技术设计研究院, 1997年6月25日

[7] 肖珍新. 基于mysql的分布式数据库的研究和实现[P]. 中国, 实用新型专利, ZL201120085830.0, 2012年4月25日

[8] 中华人民共和国国家技术监督局. GB3100-3102. 中华人民共和国国家标准–量与单位[S]. 北京: 中国标准出版社, 1994年11月1日

[9] M. Clerc. Discrete particle swarm optimization: a fuzzy combinatorial box[EB/OL]. `http://clere.maurice.free.fr/pso/Fuzzy_Discrere_PSO/Fuzzy_DPSO.htm`, July 16, 2010