基于key-value的NoSQL数据存储系统的设计与实现

软件工程 罗晨 指导教师 穆斌

【摘要】 数字家庭将改变人们的生活方式，为人们提供方便、安全、智能化、舒适的数字化生活。数字家庭产业的发展是关系到国民生活信息化能否实现的关键问题。数字家庭建设是推动国民经济发展的一个重要着力点，也是一项关系信息化建设前途的战略任务。本设计所指的数字家庭的解决方案立足于近二、三年内可实现的终端产品。主要包括四件产品：家庭网关（数字家庭的控制中心）、与之相配的蓝牙手柄（可打蓝牙电话）、机顶盒、机顶盒遥控器（可用作游戏手柄），同时还有个附属的无限摄像头。除了对终端本身的功能和家电功能方面的考虑，还因涉及到进入普通家庭的产品能否和原先的家庭环境有着一定的融合性，而不是以一个孤立在外的高科技产品的形象存在。这样就要求产品的平民化和生活化，也就是说，最终体现在终端产品上的数字家庭概念能更快地融入普通消费者的生活，它和家庭的环境也能更快地融合。

【关键词】 环境 终端 平民化

【Abstract】 （略）

【Keywords】 （略）

# 引言

## 概述

## 课题背景

## 研究目标

## 论文结构

# 相关概念

## 数据存储系统

数据存储系统也可被称作数据库管理系统，由一个互相关联的数据的集合和一组用以访问这些数据的程序组成[1] 。数据库技术最初产生于20世纪60年代中期，随着计算机管理数据的规模越来越大，应用越来越广泛。数据库技术也在不断地发展和提高。尤其是在1970年IBM的研究员发表”A Relational Model of Data for Large Shared Data Banks”[2] 奠定了现代关系数据库的基础。关系数据库是创建在[关系模型](https://zh.wikipedia.org/wiki/%E5%85%B3%E7%B3%BB%E6%A8%A1%E5%9E%8B)基础上的[数据库](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%BA%93)，借助于[集合代数](https://zh.wikipedia.org/wiki/%E9%9B%86%E5%90%88%E4%BB%A3%E6%95%B0)等[数学](https://zh.wikipedia.org/wiki/%E6%95%B0%E5%AD%A6)概念和方法来处理数据库中的数据。关系数据库提供完整的关系操作（增、删、改、查等），完整性约束（实体完整性、参照完整性和用户定义完整性）以及关系数据语言SQL。关系数据库中的数据通过关系（表）来组成，结构清晰，并且功能强大，因此被广为使用。目前比较流行的关系数据库有Oracle，MySQL，微软的SQLSERVER等。

除了关系数据库之外，目前业界比较流行的被称为数据库有对象数据库。对象数据库是人们为了适配当前面向对象语言（如C++，Java）而尝试开发的一种数据库。面向对象语言可以直接操作数据库中的对象，而无需进行繁琐的转换。比较有代表性的对象数据库有Versant公司的Versant以及db4o数据库，主要运用在电信、金融等领域。但由于对象数据库的诞生时间较晚，以及缺少形式化的数学基础而导致的查询上的弱势，对象数据库并没有广泛的流行起来。

由于关系数据库在业界所占非常大的比重，因此本论文中主要考虑关系数据库，除了特殊的说明之外，下文中的“数据库”也代指的是关系数据库。

## 传统数据库瓶颈

由于SQL数据库对数据的存储与操作提供了强大的支持，人们也因此普遍采用关系数据库作为应用的数据存储系统。但随着互联网的发展和互联网用户的增多导致了数据集的急剧膨胀以及高并发的数据访问，例如Google、Facebook此类网站。传统的关系数据库在这种环境下开始成为应用的瓶颈之一：

#### 随着数据集的增大和高并发的访问，单机数据库难以处理大规模的数据存储与访问；

#### 关系数据库通常为了提供数据的强一致性约束和ACID属性的事务，会在并发的数据访问时对数据进行加锁，因此限制了并发能力；

#### 同样由于强一致性的保证，关系数据库难以实现分布式。虽然可以通过two-phase commit[3] 技术来实现分布式事务，但通常会涉及集群中的所有机器，同样限制了集群的规模；

#### 关系数据库提供存储过程以实现数据的业务逻辑，但随着SOA架构的流行，人们逐渐开始将数据操作与数据存储解耦，即数据操作放在业务层以web服务的方式提供，而数据库中只负责存储数据。

虽然关系数据库比较难以实现分布式，但目前仍有一些公司通过放松数据一致性的要求以及关系数据库中的某些特性（如表连接查询）实现了关系数据库的分布式集群，并得到了较好的应用。以此为代表的是eBay所采用的以分库分表的方式[4] 构建数据库集群。通过对数据的主键或某些业务属性进行分组，可以将数据分配到不同的表或不同的数据库中。但需要额外的路由程序或查询解释器将数据分配到对应的表或数据库中，同时也失去了数据的强一致性保证。

## NoSQL数据库

NoSQL通常也称为Not Only SQL的缩写，是对不同于传统的关系数据库的数据库管理系统的统称。NoSQL数据库与传统的关系数据库在很多方面上都有区别，但最重要的特点是NoSQL数据库不使用SQL作为查询语言，并且数据存储不采用固定的表格模式，因而也更加灵活，并具有良好的水平扩展性。

目前NoSQL数据库通常包括以下几类[5] ：

#### 键-值存储：系统根据用户定义的键来存储并索引对应的值。键-值存储就像一个哈希表，但是通常提供了额外的功能，例如复制，版本控制，加锁，事务，排序等。键-值存储系统的代表有Redis(<http://redis.io>)，Amazon公司研发的Dynamo[6] 等。

#### 文档存储：系统存储定义好的文档，并提供索引和简单的检索机制。与键值存储不同的是，这类系统通常提供二级索引，多种类型的文档以及嵌套文档。文档存储系统的代表有MongoDB（<http://www.mongodb.org/>）。

#### 可扩展的记录存储：系统存储可被垂直分区以及水平分布在不同节点上的可扩展的记录。基本的数据模型通常为行和列，行通过唯一的键指定，而列中通常包含若干预定义好的组以及任意多的属性。可扩展的记录存储系统的代表有Google公司研发的BigTable[7] ，Cassandra等。

以上几种NoSQL数据库都包含一些相似的特征，例如数据的无结构或弱结构性，提供弱一致性或最终一致性[8] 的保证，易于水平扩展，很少提供事务支持等。而通常的互联网应用如社交网站、论坛、邮箱等，业务逻辑较为简单，对数据的一致性要求不是太高，并且通常具有大数据、高并发等特性，因此比较适合采用NoSQL数据库作为数据存储系统。而与此同时，NoSQL数据库也在这些大型的互联网公司的推动下以及大数据的背景下不断发展，与关系数据库进行更好的互补。

## 系统扩展性

随着大数据时代的来临，传统的单机数据库模式早已不能适应日益增长的数据规模和高并发的访问。对于数据库系统的扩展方式，通常有如下两种不同的方案。

### 垂直扩展

垂直扩展即对一个物理实体增加资源而提高计算实体的性能。例如使用更多核的CPU，添加更多的内存和硬盘等。垂直扩展的优点是简单方便，而且无需对软件系统的代码进行修改。目前市场上许多服务器厂商如Dell、IBM、联想等都提供有超高性能的服务器。SAP的HANA数据库可支持TB级的内存计算。然而垂直扩展的缺点也是显而易见，首先超高性能服务器的价格远远高于普通机器，其次服务器很难无限制的扩展，容易达到瓶颈，并且当前性能越高，进一步扩展的开销也就越大。因此垂直扩展的架构方案并没有被广泛的采用。

### 水平扩展

水平扩展也被称作横向扩展。与垂直扩展相反，水平扩展将多台计算实体通过特定的软硬件连接成为一个逻辑的实体以提供更强大的计算能力。水平扩展通常采用性能普通的服务器甚至PC机，通过特定的分布式算法将任务分配到不同的机器上，以获得强大的计算能力。例如Google的BigTable[7] 存储系统就已经实现了上千台机器组成的集群中。水平扩展的优点显而易见，即价格低廉，使用普通的机器便可获得强大的集群，而且集群的总体性能也不受单台机器性能的限制，因而也受到许多IT公司的青睐。

然而水平扩展在实施上通常也比垂直扩展更加复杂：

#### 数据存储系统的水平扩展需要软件层面的支持。例如eBay公司的分库分表方案[4] ，即需要专门的路由程序的支持，以将数据根据主键或特定的业务属性分配到对应的机器中

#### 集群需要较高的维护难度和成本。随着机器数量的增加，集群的维护成本也相应的增加，并且需要自动化工具的支持以实现日常的维护工作。

#### 分布式集群为系统提出了额外的问题和复杂性，例如网络分区问题，节点失效问题，节点间协调问题等等，都需要专门的算法进行解决。

由于垂直扩展与水平扩展各自的优缺点，二者适应的场景也有所不同。垂直扩展的简单方便比较受非IT企业的青睐，如电信、金融、能源等领域的公司，并且垂直扩展带来的性能也可基本满足其业务需求。而水平扩展由于比较复杂，目前只有较大的IT公司如Google、Facebook、Amazon等才有能力自主研发数据存储系统，并搭建大规模的集群。本论文中所实现的系统基于横向扩展，将多服务器搭建成统一的集群，以提供数据存储服务。

## CAP理论

2000年，加州大学伯克利分校的Eric Brewer教授在2000年的分布式原则研讨会上提出了著名的CAP理论，即对于一个分布式系统来说，不可能同时满足以下三点：

#### 一致性（Consistency），即所有节点在同一时间有相同的数据；

#### 可用性（Availability），保证每个请求不管成功或者失败都有响应；

#### 分区容忍性（Partition Tolerance），系统中任意信息的丢失或失败不会影响系统的继续运作。[10]

而CAP理论的正确性也在2002年麻省理工学院的Seth Gilbert等人的” Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services”[11] 论文中被证明。

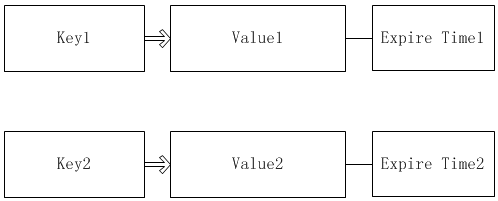
由于分区容忍性是分布式系统的基础，因此人们通常会在一致性和可用性之间进行取舍，或对二者进行平衡。具有一致性与分区容忍性（CP）的代表系统有Google公司的BigTable[12] [7] 系统，其通过Google File System[12] 实现了较高的数据一致性。而Amazon的Dynamo[6] 系统则通过版本号机制和数据多备份等实现了较高的一致性，然而应用程序则需要解决有可能读取不到最新数据的问题。而另一类系统如Cassandra（<http://cassandra.apache.org/>）则允许用户配置成功写操作的最小节点数（W）、成功读操作的最小节点数（R）和副本节点的数量（N）来调整系统的一致性和可用性。当W+R越大，则一致性越高，可用性越低，反之亦然。

# 系统编程接口

## 数据模型

### 键值结构

Kvstore采用键值模型作为数据模型结构，类似于一个分布在集群中的哈希表。每一个键都唯一确定了一个值，并且不允许有重复的键以及空值出现。另外键和值在系统中都存储为字节数组，而由应用程序开发者来负责解释键值的含义，例如可以是字符串、序列化的对象或者其他任意的结构。

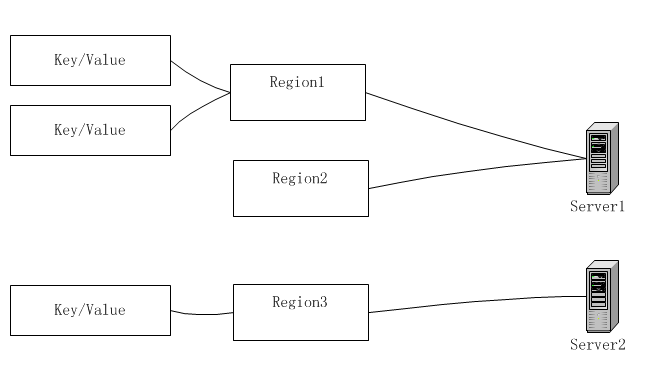


图表 3‑1

如**错误!未找到引用源。**所示，每个键分别唯一对应了一个值和失效时间。失效时间为long型的时间戳，如果服务器的当前时间超过了失效时间，那么该键值对将会被自动删除，即用户再也无法通过键取得对应值。

### Region结构

Region是从开始键到结束键的所有键值对的集合，也是系统负载均衡的基本单位。如图表 3‑2所示，Region会被分配到某个唯一的服务器中，之后所有对该区域内的键值对的请求都会由该服务器所负责。



图表 3‑2

Region中包含如下的属性：开始键，结束键，以及由主服务器分配的唯一的Id。另外每个Region还对应了统计信息，统计的数据项包括键值对数量、大小、读次数和写次数。这些统计信息会作为负载均衡的依据供主服务器使用。 键的比较会按照字节顺序，首先按照数组的长度，长度短者小；如果长度相等，这依次按无符号字节的格式比较数组的每一位，首先出现小字节的字节数组小。

## 键值操作

Kvstore作为一个键值存储系统，支持如下几种常见的键值操作。下文中如未进行特殊说明，将按照operation param1 [param2=value]…的格式进行说明各个操作语义，其中operation是操作名称，param1是必须参数，param2是可选参数，并且默认值为value。

### set操作

Set操作的语义为设置某个键值对，即将一个键值对存储到系统当中。Set操作的指令为：set key value [ttl=0]。其中key、value都为二进制数组，ttl为整形，表明该键值对应存活的毫秒值，默认为0，即永久保存。

### get操作

Get操作的语义为获取某个键对应的值以及存活时间的毫秒值，如果键不存在，则返回空。Get操作的指令为：get key。其中key为二进制数组。

### incr操作

Incr是increment的缩写，即增加某个键对应的整数计数器的值。Incr操作的指令为incr key [incremental = 1] [initValue=0] [ttl=0]。其中key为二进制数组，incremental为要为计数器增加的整数值，默认为1；initValue为计数器的初始值，默认为0；ttl为计数器存活的时间毫秒值，默认为0，即永久保存。另外计数器实际上在系统中对应了一个长度为4的二进制数组，如果尝试为一个值长度不为4的键执行incr操作，那么系统将返回错误信息。

### delete操作

Delete操作是在系统中删除某个键指定的键值对。Delete操作的指令为delete key。其中key为二进制数组。

### stat操作

Stat是statistics的简写，用户将获得数据结构体的数组，其中每个数组结构体包含了当前服务器的地址，权值，使用情况，以及其中所包含的Region情况。服务器的权值为用户指定的整数值，会在主服务器进行负载均衡时使用。服务器的使用情况包括服务器总的内存、当前剩余的内存以及CPU的使用率。由于kvstore基于分布式的文件系统，因此这里并未统计真实文件系统的使用情况。用户可以通过HDFS的管理页面进行查看。

# 系统依赖

Kvstore作为分布式数据存储系统，需要数据的持久化存储、副本、分布式协调、网络通信等功能。以下系统或框架作为kvstore的基础设施，为kvstore的实现提供了基础。

## 分布式文件系统

分布式文件系统是建立在网络之上的文件系统，即通过软件的方式，将多台服务器的文件系统集中成统一的文件系统，为用户提供文件的存储与访问服务，具有高度的透明性。

本系统采用HDFS（[Hadoop Distributed File System](http://hadoop.apache.org/docs/stable/hdfs_user_guide.html)）作为文件系统，以存储操作日志文件和数据文件。HDFS是hadoop项目的一部分，也是Google File System[9] 的开源Java版本实现。HDFS具有如下几大特性：

#### 适于利用普通服务器搭建分布式存储集群，并且具有容错性和易于扩展性。

#### 具有高度配置性，可通过配置系统参数如文件块大小、副本数量等适应不同的需求。

#### 由Java编写，可支持所有主流平台。

#### 内置了web服务器，可以方便检查集群的状态。

#### 支持文件的负载均衡，尽可能的将文件均匀分布到每个节点上。

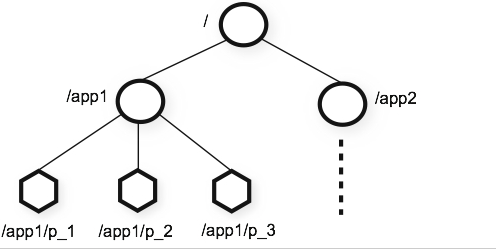
#### 提供类似Java IO的流式IO接口，简单易用。

由于HDFS通过文件的副本机制提供了数据的容错性，因此本系统中将通过在HDFS中存储操作日志和数据文件，即利用HDFS自身提供的容错性机制来保障数据的可靠性。

另外由于Apache官方提供的发行版本中并未提供HDFS单独的包，而是作为Hadoop版本的一部分进行发行。因此为减少系统的依赖，作者基于Hadoop r1.0.4版本的源码，将其中的HDFS部分单独打包以供本系统使用。

## 分布式协调系统

本系统选用ZooKeeper作为分布式协调系统。ZooKeeper(<http://zookeeper.apache.org/>)是Hadoop的子项目，也是Google Chubby[13] 服务的Java开源实现。作为分布式协调系统，ZooKeeper提供了配置信息管理、命名服务、分布式同步、分组等服务。



图表 4‑1

如图表 4‑1所示，ZooKeeper中采用了类似文件系统的树状结构，用户可以在某个路径上创建节点或者读写节点上附加的数据，也可以以回调的方式监听某个节点。ZooKeeper中的节点分为暂时性和永久性，暂时性节点当客户端失去连接时，会被ZooKeeper删除，而永久性节点则会被永久保存。此外ZooKeeper为用户的更新操作提供了如下保证：

#### 顺序一致性，即客户端的更新操作会按照发送的顺序执行。

#### 原子性，更新操作要么成功要么失败。

#### 单系统镜像，客户端不管连接到哪台ZooKeeper服务器，看到的都是统一的视图。

#### 可靠性，一旦更新生效，它将一直持续到某个客户端覆盖了此次更新。

#### 时效性，客户端可以在固定的时间延迟内，看到系统的最新状态。

通过ZooKeeper提供的操作以及以上几点保证，用户可以方便的实现各种分布式协调算法，如通过监控暂时性节点可监控集群中节点的状态，通过顺序创建多个暂时性节点来实现Paxos领导选举算法[14] ，以及实现分布式队列、分布式锁服务等等。

## 异步网络通信框架

作为分布式的数据存储系统，网络通信是系统中必不可少也是影响性能的一部分。Kvstore系统采用Mina作为网络通信框架。Mina([http://mina.apache.org](http://mina.apache.org/))是一款开源的基于Java NIO技术的网络通信框架，它具有异步、高性能、模型简单、易于使用等诸多优点：

#### 基于异步的事件回调模型，减少应用程序的等待时间，因而具有高并发性。

#### Mina支持多种传输协议，如TCP/IP、UDP/IP、RS232等，封装了底层实现并提供统一的编程接口。

#### 提供类似于Java Servlet Filter的过滤器，过滤器可作为扩展点，方便开发人员对消息进行预处理以及后处理。

#### 提供底层及高级的API，开发人员既可以直接操作二进制的消息流，也可以提供消息的编码、解码器以面向对象的方式操作消息。

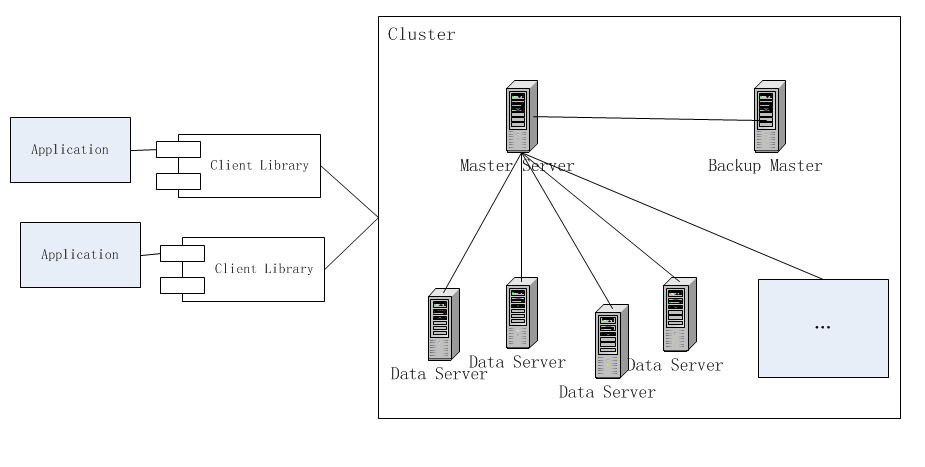
#### 高度订制的线程模型，包括单线程、单线程池和多线程池。

系统中的主服务器、数据服务以及Java客户端之间的网络通信完全基于Mina框架以及TCP/IP协议。具体的消息格式将在5.5通信协议进行详细介绍。另外系统同样提供了基于C#的客户端程序，该客户端则直接采用了Socket直接连接到主服务器和数据服务器。

# 系统架构

## 架构概览

### 总体架构



图表 5‑1

图表 5‑1为系统的总体架构图。系统集群采用Master-Slave模式，并提供Backup Master以避免单点失效问题。另外应用程序通过系统提供的相应语言版本的客户端库访问集群，使用系统提供的数据存储服务。

#### 主服务器（Master Server）

主服务器是系统中的“管理者”，负责监听集群状态（例如数据服务器的加入与移除），定期收集各个数据服务器的状态信息，并定期的对集群数据服务器的负载进行负载均衡操作。此外，主服务还会响应客户端的Region表请求，并返回目前集群中的Region表。集群只能同时允许一个主服务器存在。主服务器将在章节5.2主服务器中详细介绍。

#### 备份主服务器

为了避免主服务器的单点失效问题，集群中通常还会有备份主服务器。备份主服务器在功能上与主服务器完全相同，但是所有的主服务器在启动时都会参与到领导选举算法中，被选作领导的服务器会成为主服务器，而其他的服务器都会成为备份主服务器，等待下一次领导选举算法的执行。

#### 数据服务器（Data Server）

数据服务器负责系统中键值对数据的存储工作。每个数据服务器会被主服务器分配若干个Region，数据服务器会负责这些Region范围内的键值对的操作。数据服务器提供内存和持久化两种存储引擎，内存存储引擎完全使用内存存储，但在内存容量满之后会置换出某些键值对，因而适合用作缓存系统；而持久化存储引擎则使用内存+文件的方式保证数据的持久性存储。数据服务器将在章节5.3数据服务器中详细介绍。

#### 客户端库程序（Client Library）

客户端库程序是应用程序以访问本系统数据存储服务的桥梁。客户端库程序主要封装了键路由、网络连接管理等工作。开发人员只需指定主服务器的地址，便可通过get、set等操作访问本系统，而无需繁琐的初始化、连接建立与释放等工作。理论上系统支持任意语言的客户端访问，只要客户端按照网络消息协议的规范与主服务器以及数据服务器交互即可。目前系统提供了Java和C#两个版本的客户端库程序。客户端库程序将在章节5.4客户端中详细介绍。

#### 应用程序（Application）

应用程序是系统的最终用户，通过系统提供的客户端库程序访问集群以使用系统的数据存储服务。目前Java和C#的应用程序可直接使用系统提供的客户端库，而其他语言的应用则可能需要按照系统的网络消息协议，开发对应的客户端库程序。应用程序可自己解释键值的含义，例如序列化的对象、文档等，同时也可根据不同的需求定制系统的功能，如用作缓存系统、持久存储系统等。

### 设计风格

Kvstore作为系统级应用和框架级应用，应具有较高的性能、较好的可扩展性以及可维护性以及提供简单易用的API。为达到这些目标，本系统的设计与编码遵循如下几点原则和风格：

#### KISS原则

Kiss是Keep it simple stupid的缩写，即应尽可能的将系统设计的简单易用。Kvstore系统实现过程中尽可能的将复杂的操作拆分并简化，通过简单来提高性能，提高可靠性。同时系统提供的接口也应保持简洁，方便开发人员学习和使用。

#### 面向接口编程

面向接口编程即系统的模块与模块之间通过接口进行依赖，而不是直接依赖于实现类。面向接口编程有益于系统中模块间的解耦，增加系统的灵活性和可扩展性。在kvstore系统中，所有可能发生变化的实现都被抽象为接口，并通过接口进行调用。因而当需要更换某个接口的实现类时，可以尽可能的避免其他模块的修改。

#### 事件驱动

为提高扩展性，系统在许多操作中都提供了事件点，并允许调用者注册监听器进行回调。通过事件驱动机制，可以灵活的向已有实现中添加功能。例如，系统在每个get、set等操作后都提供了事件点，可以以此添加日志监听器、统计监听器等功能。

#### 使用设计模式

设计模式是前人对已有设计问题的总结，也是最佳实践。通过使用设计模式，可以帮助实现低耦合高内聚的目标，提高系统的灵活性、可扩展性和可维护性；同时由于大多数开发人员对设计模式都有所了解，因此通过遵循已有的设计模式，也可让开发人员更快的使用及开发本系统。例如系统中广泛用到了工厂模式、代理模式、策略模式、装饰模式、迭代器模式等。

## 主服务器

### 集群状态监听

### 定时任务

## 数据服务器

### 内存存储引擎

### 持久化存储引擎

### 文件格式

### 索引结构

### 任务线程

## 客户端

### 消息格式

### 同步模型

### 异步模型

## 通信协议

# 功能实现

## 集群启动与关闭

### 启动顺序

### 动态添加Data服务器

## 故障恢复

### Master服务器故障

### Data服务器故障

## 负载均衡

### 自动路由

### Region转移

### Region拆分与合并

## 响应请求

## 系统监控

# 性能数据

## 测试环境

## 测试方法

## 性能结果

# 相关工作

# 结论与展望

主要参考文献

《数据库系统概念-第五版》

Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data banks”. Communications 13 (6): 377–387. [doi](https://en.wikipedia.org/wiki/Digital_object_identifier):10.1145/362384.362685

Bernstein P A, Hadzilacos V, Goodman N. Concurrency control and recovery in database systems[M]. New York: Addison-wesley, 1987.

Pritchett D. Base: An acid alternative[J]. Queue, 2008, 6(3): 48-55.

Cattell R. Scalable SQL and NoSQL data stores[J]. ACM SIGMOD Record, 2011, 39(4): 12-27.

DeCandia G, Hastorun D, Jampani M, et al. Dynamo: amazon's highly available key-value store[C]//ACM Symposium on Operating Systems Principles: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles. 2007, 14(17): 205-220.

Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data[J]. ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 4.

Vogels W. Eventually consistent[J]. Communications of the ACM, 2009, 52(1): 40-44.

Allavena A, Demers A, Hopcroft J E. Correctness of a gossip based membership protocol[C]//Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing. ACM, 2005: 292-301.

Brewer E. CAP twelve years later: How the[J]. Computer, 2012, 45(2): 23-29.

Gilbert S, Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services[J]. ACM SIGACT News, 2002, 33(2): 51-59.

Ghemawat S, Gobioff H, Leung S T. The Google file system[C]//ACM SIGOPS Operating Systems Review. ACM, 2003, 37(5): 29-43.

Burrows M. The Chubby lock service for loosely-coupled distributed systems[C]//Proceedings of the 7th symposium on Operating systems design and implementation. USENIX Association, 2006: 335-350.

Lamport L. The part-time parliament[J]. ACM Transactions on Computer Systems (TOCS), 1998, 16(2): 133-169.

【另注页面设置要求：上、下各设定4.1厘米；左、右各设定3厘米；页眉、页脚各为3.5厘米；文章内容设置行距为“单倍行距”。】

|  |  |  |
| --- | --- | --- |
| 整理后的优秀论文是否经指导老师审核：（是/否） | |  |
| 如果未经指导老师审核请注明原因： |  | |