分类号_____ 学号 M201376145

学校代码 10487 密级_____

华中科技大学

硕士学位论文

去中心化的网络社区系统的设计与实现

学位申请人 ： 龙小康

学 科 专 业 ： 软件工程

指 导 教 师 ： 陆永忠副教授

答 辩 日 期 ： 2016.1.14

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree for the Master of Engineering

Design and Implementation of a Decentralized Online Community System

Candidate : Long Xiaokang

Major : Software Engineering

Supervisor : Assoc. Prof. Lu Yongzhong

Huazhong University of Science & Technology

Wuhan 430074, P.R.China

January 14, 2016

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，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。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权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。本人授权华中科技大学可以将本学位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学位论文。

本论文属　保密□，　　在_____年解密后适用本授权书。
　　不保密□。

（请在以上方框内打“ √ ”）

学位论文作者签名：　　指导教师签名：

日期：　　年　月　日　　日期：　　年　月　日

摘要

网络社区系统在当前的互联网上有着非常广泛的应用，为互联网用户的在线交流提供了便利，丰富了互联网在线内容的多样性，在互联网上扮演着无可替代的角色。然而，传统的网络社区系统所依赖的服务器-客户端的基本架构模式存在着较多的弊端。首先，中心服务器极易成为整个系统的性能瓶颈，进而拖累整个系统。其次，中心服务器是一个非常明显的攻击目标，很容易遭受攻击导致系统不可用，甚至用户数据泄露。再次，中心服务器通常由单一一方所掌控，整个系统的可信任度取决于这单一一方的可信任度。最后，掌控系统中心服务器的商业公司出于商业利益，可能会利用用户数据做出损害用户利益的行为。

本论文提出一种完全去中心化的分布式网络社区系统的设计，该网络社区系统与传统网络社区系统的不同之处在于，该系统将不再需要任何中心服务器来提供和存储数据，而是由参与进系统的所有计算机节点共同维护整个系统。系统使用区块链技术和分布式共识技术构建出灵活而可靠的去中心化的分布式结构，并结合现代加密技术中的加密散列算法和数字签名算法来保障系统的安全性。

系统的运行不依赖于任何单一特定节点，其中的信息既不会被审查，也不会被篡改，同时整个系统也非常难以被关闭。该系统不需要用户提供详细资料，也不会记录和分析用户行为，给用户极大的隐私保护。并且，结合现代加密技术和分布式理论，使得系统具有安全可靠的匿名性和不可追溯性，同时又能保证匿名身份的不可伪造性。

关键词：去中心化　分布式　网络社区系统　区块链

ABSTRACT

Ordinary online community systems have been wildly adopted on internet. They provide internet users with convenience and enriched internet content. They are still playing a very important role on internet. But ordinary online community systems rely on server-client architecture and have several drawbacks. First, central servers can easily become the bottle-neck of the whole system and slow down the system. Second, central servers are very obvious targets for attackers, attacks against central servers may result in unavailable system, or even worse, user data leaks. Third, central servers are usually controlled by a single party, the system's trustworthiness depends on this single party. Finally, companies which control these central servers may use user data to do things that are against the interests of users.

This paper introduces a fully decentralized distributed online community system design. It differs from ordinary online community system in that it does not need any central servers, and the whole system is maintained by the computer nodes involved together. This system utilizes blockchain and distributed consensus to build a flexible and reliable decentralize distribution structure, and combines with cryptographic hash algorithm and digital signature algorithm to secure the system.

The system's functionality does not depend on any single node, the system's data will not be censored, or be altered, and the system is nearly impossible to shutdown. This kind of system can provide users great privacy protection, secure and reliable non-traceability and anonymity, and keep user identity unforgeable.

Keywords: Decentralization Distributed Online-Community-System Blockchain

目录

1 绪论

1.1 研究背景与意义

网络社区系统在互联网发展的初期便得到广泛应用，互联网用户使用这类系统来进行各种在线交流。在互联网技术特别是移动互联网技术高度发展的今天，网络社区系统仍然在互联网用户的生活中扮演着无可替代的角色。

尽管计算机技术和互联网技术的发展已经今非昔比，但是网络社区系统的本质和基本运行模式并没有多大变化。这类系统依然起着承载用户之间相互交流的信息的作用，并且大多采用服务器-客户端的架构模型。即这类系统需要至少一个中心服务器，用来存储数据、维持数据完整和处理各种数据之间的关系，并且需要客户端来与用户进行交互，客户端可以是专用的客户端，也可以是运行在网页浏览器中的通用客户端。

在用户数量较多的情况下，租用或者购买运行此类系统服务器程序的硬件设备的开销将会非常不菲，并非普通个人用户可以承担。因此，此类系统的多数由商业公司运营，商业公司对其运营的社区系统拥有最高控制权，在商业利益的驱使下，用户在其系统上发布的信息可能会遭到篡改、删除，或者被用来投放基于“兴趣跟踪”的商业广告，这些都是用户所不乐见的，甚至是侵犯用户利益的。

本论文将提出一种不需要中心化服务器，完全依赖所有最终用户共同维持和运行的社区系统的设计。在去中心化设计的基础之上，中心化的服务器程序将不再是必要的，因此既可以摆脱商业公司的控制，又可以承载非常大的用户量，也就可以避免前面所提到的当前由商业公司所运营的社区系统的各种弊病。并且事实上，该系统的最终用户量越大，整个系统的运行也更加快速、更加安全。

1.2 国内外研究概况

传统的中心化网络社区系统在国内外互联网上都有大量的应用，比较知名的有美国的twitter、reddit、日本的2ch以及中国的百度贴吧和天涯论坛等等，这些网络社区均服务了大量的用户，并且用户活跃度高，在其中产生了大量内容。为了应对大量用户的同时访问，这些社区系统的服务端普遍采用分布式的架构模型，以达到系统处理容量可高效伸缩的目的，即在访问量增加的情况下，可以简单地增加硬件处理能力来应对，比如增加一定数量的后台服务器，整个系统的处理容量随即增加，反之亦然。为了高效地存储和查询用户数据，大多采用了高性能的关系型数据库管理系统，近年来非关系型数据库管理系统也获得了较多采用。由于，传统的数据库管理系统的可伸缩性不佳，分布式的数据库管理系统也在不断涌现出来。为了保证数据的安全，避免磁盘损坏造成的数据丢失，多种类型的磁盘冗余阵列技术被开发出来，几乎成为本地备份的标准配置。

本论文所研究的去中心化的网络社区系统尚未有成功的大范围应用先例，但是与之相关的技术和理论则有较为丰富的研究结果。其中分布式网络架构协议，有Paxos协议和Gossip协议等等，数字签名算法有DSA和ECDSA算法等等，安全的加密散列算法，如SHA-2等。与之相似的应用有目前正在兴起的数字加密货币系统，如比特币等。这类系统同样不依赖中心化服务器，但是能够承载数字货币的流通，并保证数字货币系统的安全。

1.3 论文的主要研究内容

本论文主要的研究工作包括以下几个方面：

（1）分析和研究数字加密货币系统中的相关原理和技术，包括网络架构协议、数字签名算法和分布式共识算法。

（2）在相关分析和研究的基础上提出一种去中心化社区系统的设计。

（3）依据本文提出的去中心化社区系统的设计，采用 Python 程序设计语言和

SQLite 数据库构建出一个初步可用的实现。

1.4 论文的组织结构

第一章绪论，简要介绍本文所研究的主题的背景、研究意义、国内外研究现状以及本轮文的主要研究工作。

第二章介绍去中心化社区系统相关的技术，并对其进行一定的分析。

第三章简要分析了去中心化社区系统，并提出具体设计。

第四章在设计方案的基础上实现了一个具体的系统程序。

第五章对论文进行了总结，并提出后续优化的方向。

2 去中心化网络社区系统相关技术的分析

2.1 加密散列算法

加密散列算法是一种事实上几乎无法逆向推导的散列算法，逆向推倒是指从散列算法的输出值推导得到输入值。一个良好的加密散列算法需要符合以下四个特征[1]：

（1） 对于任何输入数据都可以简单地计算得到散列值

（2） 从散列值推导得出原输入数据从计算理论上来说是非常困难的，即单向性

（3） 改变输入数据而不改变散列值从计算理论上来说是非常困难的，即弱抗碰撞性

（4） 找出两组有着相同散列值的输入数据从计算理论上来说是非常困难的，即强抗碰撞性

这里的非常困难是指该问题理论上是可以解的，但是解决该问题所需要耗费的时间和资源将大到无法接受的程度，使得该问题事实上不可解。

特征（1）是显而易见的，如果一种加密散列算法需要耗费大量的计算资源，那么将极大地限制其应用场景，因为将不具有实用价值。

如果不符合特征（2），那么输出值的空间将和输入值的空间至少一样大，结合加密散列算法输入值任意的特性，不具有单向性的加密散列算法无论从计算还是从存储的角度来看都是不可接受的。另外，特征（2）所说的单向性并不能代表找到碰撞是困难的。简单地以取模运算为例，很显然取模运算是单向的，但是要找到碰撞却极其简单。如 17 对 10 取模，得到 7，显然不可能从 7 推导得到 17，但是任何 N×10＋7（N≥0）的整数都能导致碰撞。

特征（2）（3）（4）是保证加密散列算法的基石，特征（4）可认为是包含特征（3）的，因为如果找到两组有着相同散列值的数据是困难的，那么给定其中一组

数据和散列值，而找出另一组数据只会更加困难，因此可统称为抗碰撞性。对于特征（3），如果加密散列算法没有明显漏洞，而只能使用穷举的方式暴力破解的话，那么破解难度将随着输出空间的增长而呈指数级增长，假设其输出为 N 比特位的长度，那么找到一个碰撞将平均花费 2^{N-1} 轮操作，即 2^{N-1} 轮操作内发现碰撞的概率为 50%。对于特征（4），同样在没有明显漏洞而只能以穷举的方式暴力破解的情况下，其破解难度将远小于特征（3），假设其输出为 N 比特位的长度，那么找到有着相同散列值的两组输入数据将平均只花费 $2^{N/2}$ 轮操作。因为加密散列算法的输出可以认为是伪随机的，那么对于操作轮数 t（$t<2^N$），其发现碰撞的概率为：

$$P(t;2^N) = 1 - \prod_{k=1}^{t-1}\left(1 - \frac{k}{2^N}\right) \quad （2\text{-}1）$$

使概率 P=0.5，得到 t 约为 $1.172^{N/2}$，可看作为 $2^{N/2}$ 轮操作[2]。

加密散列算法在信息安全领域有着非常广泛的应用，如数字签名、消息鉴别码以及其他类型的鉴别手段。同样的，加密散列算法也可以作为普通的散列算法来使用，在散列表中作为数据的索引；也可以作为数据的指纹，以检测数据的唯一性；或者作为数据校验和，以检测数据损坏。

信息安全领域发展到今天，存在着非常多种类的加密散列算法，但是其中很多都被发现是有漏洞的，或者没有明显漏洞，但是可在较短的时间内被暴力破解，这类加密散列算法都被认为是不安全的，而不应该继续使用，如 MD5[3][4]和 SHA-0[5]。还有一些加密散列算法虽然在当前没有遭到任何破解，但是被发现其破解难度要小于设计者生成的难度，如 SHA-1，其在 2005 年被发现可在 2^{63} 轮操作之内被破解[6]，低于设计者声称的 2^{80} 轮操作，这类加密散列算法也是不建议使用的，但是仍然被广泛使用至今。因为虽然实际破解难度低于设计者所声称的难度，但是尚没有任何组织或者个人宣称已经破解 SHA-1。

表 2-1 常见加密散列算法的比较

算法名		最大输入（比特）	输出大小（比特）	块大小（比特）	操作轮数	安全级别（比特）
MD5		$2^{64}-1$	128	512	64	小于 64（已有破解）
SHA-0		$2^{64}-1$	160	512	80	小于 80（已有破解）
SHA-1		$2^{64}-1$	160	512	80	小于 80（理论破解）
SHA-2	SHA-224	$2^{64}-1$	224	512	64	112
	SHA-256		256			128
	SHA-384	$2^{128}-1$	384	1024	80	192
	SHA-512		512			256
SHA-3	SHA3-224	无限制	224	1152	24	112
	SHA3-256		256	1088		128
	SHA3-384		384	832		192
	SHA3-512		512	576		256

SHA-2 加密散列算法是目前应用比较广泛的一种加密散列算法，由美国国家安全局（NSA）设计，并由美国国家标准与技术研究院（NIST）作为美国联邦信息处理标准的一部分发布，它包含了一系列的加密散列函数，依据摘要（digest）的长度分类为：SHA-224、SHA-256、SHA-384、SHA-512，通过这些加密散列函数计算输入数据，最后生成一个与输入数据对应的固定长度的摘要[7]。

到目前为止，针对 SHA-2 加密散列算法还没有出现比穷举更有效的攻击手段，理论上需要平均进行 $2^{(L/2)}$次穷举计算才会得到一次碰撞，其中 L 代表输出摘要的长度。以目前人类文明所具有计算能力来看，$2^{(L/2)}$次穷举计算所需要花费的时间

是不可接受的，因此可以认为 SHA-2 是安全的。

SHA-3 加密散列算法属于 Keccak 加密算法的一个子集，由 Guido Bertoni，Joan Daemen，Michaël Peeters 和 Gilles Van Assche，在 2012 年成为美国国家标准与技术研究院发起的加密散列算法的胜出者。但是作为 SHA-2 的下一代算法，其目的并非在于完全取代 SHA-2，因为 SHA-2 仍然被认识是非常安全的，并没有明显的弱点，而是在于提出一种在体系结构上非常不同的加密散列算法，以提供更多选择。SHA-3 加密散列算法在 2015 年 8 月被美国国家标准与技术研究院批准作为标准发布[8]。

2.2 数字签名算法

数字签名是一种使用数学方法来鉴别数字信息的方法，一个有效的数字签名可以让接收者确信数字信息确实来自该发送者，并且没有被篡改过，类似于在纸面上手写的物理签名可用于鉴别签署人的身份。在中国，数字签名是具法律效力的，正在被普遍使用。2005 年 4 月 1 日起，中国首部《电子签名法》正式实施[9]。

一套数字签名方案需要定义两种互补的算法，一个用于签名，另一个用于验证。另外还需要一个密钥生成算法，用于生成一对密钥，包含一个私钥和对应的公钥。与手写物理签名的鉴别需要进行笔迹鉴定不同的是，数字签名的签名和验证是通过非对称加密算法中的数学运算来进行的，应用了非对称加密技术领域所使用的单向函数原理，单向函数指的是正向操作非常简单，而逆向操作非常困难的函数。这种函数往往提供一种难解或怀疑难解的数学问题。目前，非对称加密技术领域具备实用性的三个怀疑难解问题为：大数的素因数分解、离散对数和椭圆曲线问题。事实上，可靠的非对称加密算法均能作为可靠的数字签名算法的基础。

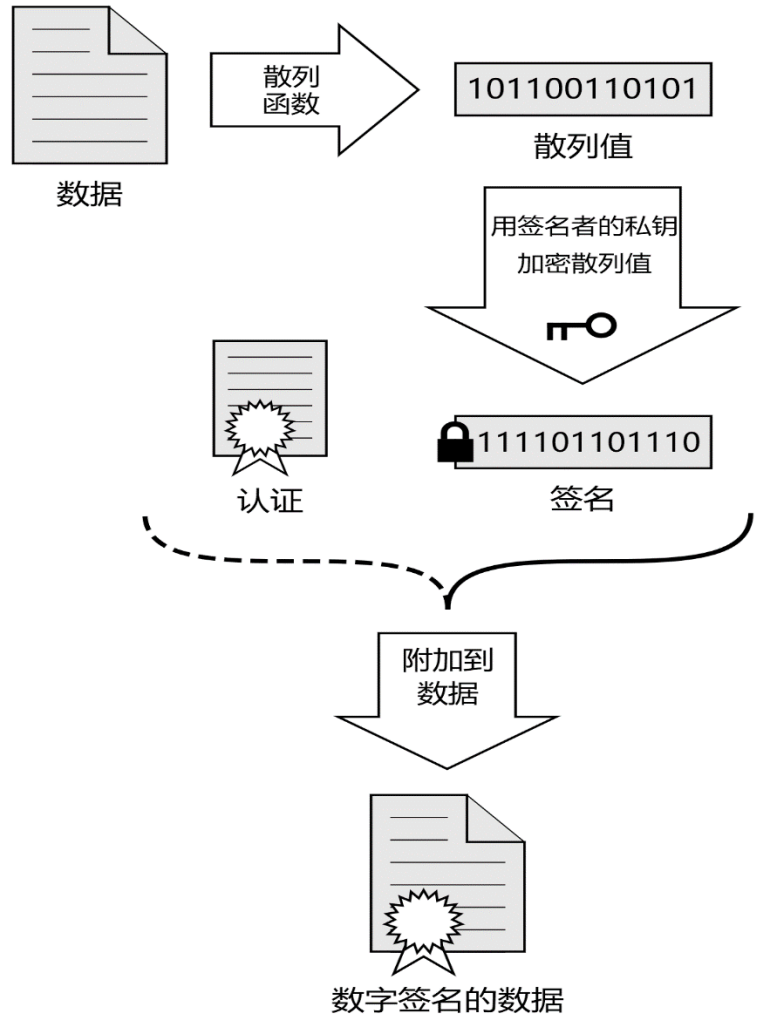


图 2-1 数字签名生成过程

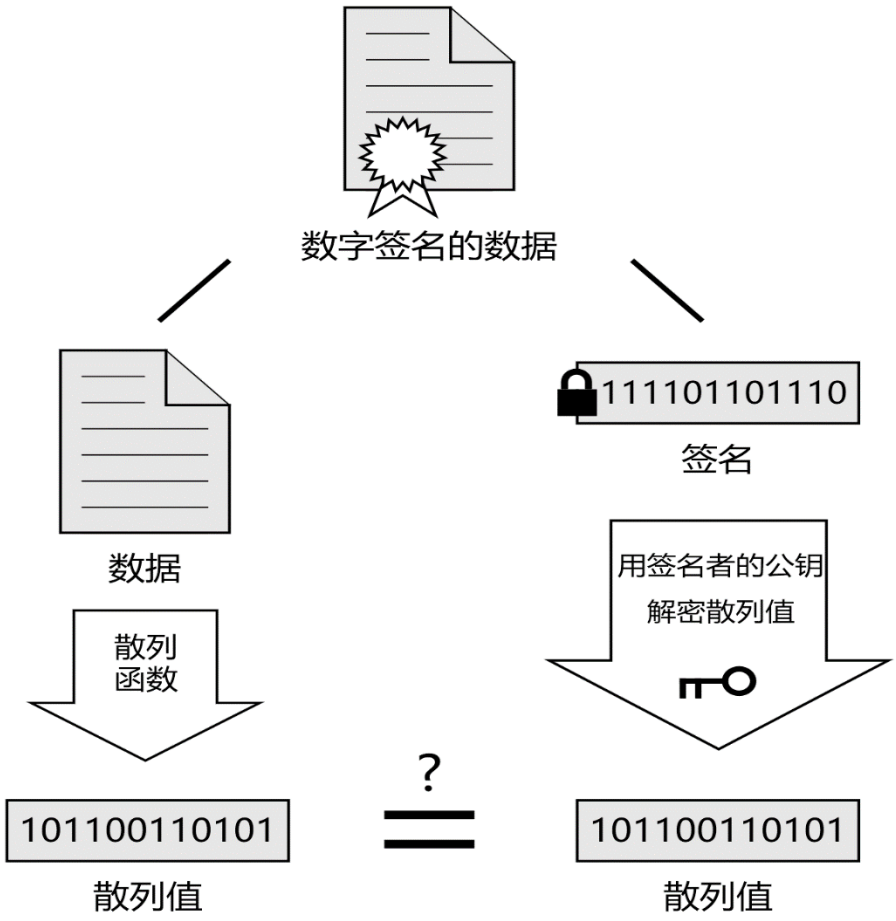


图 2-2 数字签名验证过程

签名者将需要签名的信息结合私钥通过事先定义的签名算法进行加密，通常是将需要签名的信息或者信息的散列值，通过非对称加密算法和私钥加密得到签名，并将信息连同公钥、签名一起分发。当需要进行签名鉴别以证明信息确实出自公钥持有人的时候，鉴别人使用事先定义的验证运算结合公钥解密签名，通常是将签名通过非对称加密算法和公钥解密得到信息或者信息的散列值，通过与信息或者在信息上计算得到的散列值对比，则可证明该信息是出自该公钥持有人，并且是未经篡改的。选择信息的散列值作为签名的主体，相比直接使用原信息，有如下好处：

- 散列值的长度通常要远小于原信息，进行签名或者验证运算时更为高效
- 原信息可能不适合作为签名运算的输入，通过散列可以作适当转换
- 某些签名运算对于输入长度有限制，对于超长的信息，则不得不生成多个签名，而散列值的长度通常远小于原信息，并且是固定的

Whitfield Diffie 和 Martin Hellman 在 1976 年首先描述了数字签名的概念[10]，但是并没有提出具体方案。随后出现了用于非对称加密的 RSA 算法[11]，同样可用作数字签名，因此可以算作第一个数字签名算法[12]，其他早期的数字签名算法还有 Lamport 算法[13]和 Rabin 算法[14]。

数字签名算法主要有如下三个应用方向：

（1） 保证数字信息的真实性。即数字签名可以确保数字信息确实是出自签名人，不过这只能验证数字信息与对应公钥的关系，如要验证数字信息与对应社会实体的关系，还要一个中心化的证书服务，将公钥与社会实体对应起来

（2） 保证数字信息的完整性。对数字信息进行加密只隐藏了原信息，但是无法保证信息的完整性，很多加密算法可以在原信息未知的情况下，修改加密后的信息，并且使得修改后的加密信息仍然有效。而在有数字签名验证的情况下，任何对信息的删改都会导致签名校验失败

（3） 保证数字信息的不可否认性。数字签名有助于向公钥持有方证明其所签名的数字信息的有效性。“否认”指签名方否认任何与签名的数字信息有关系的行为。

目前在数字签名中应用较多的算法有 RSA 算法、数字签名算法（Digital Signature Algorithm，DSA）[15]和椭圆曲线数字签名算法（Elliptic Curve Digital Signature Algorithm，ECDSA）[16]。

其中 RSA 算法，起初是作为非对称加密算法而设计的，但是也可以作为有效的数字签名算法，由 Ron Rivest、Adi Shamir 和 Leonard Adleman 在 1977 年提出，RSA 算法是建立在大数的素因数分解的难题之上的，即 p 和 q 是两个非常大的素

数，它们相乘得到 n，可以非常快速地计算得到，但是要将 n 分解为 p 和 p 则非常困难，在 n 达到一定长度后，可以认为这个问题不可解。

DSA 算法则是特指由美国国家安全局前雇员 David W. Kravitz 于 1991 年提出的数字签名算法，该算法起初是有专利覆盖的，但是美国国家标准与技术研究院（NIST）被 David W. Kravitz 授权免费发布，并成为美国的数字签名标准（Digital Signature Standard , DSS）的一部分[17]。DSA 算法的数学基础是离散对数问题的难解性，即给定一个素数 p，和有限域 Z_p 上的一个元根 a，对 Z_p 上整数 b，寻找唯一的整数 c，使得 $a^c \equiv b \pmod p$。一般来说，如果选用了合适的 p，则可以认为该问题是难解的，且目前还没有找到计算离散对数问题的多项式时间算法。从目前已知的攻击方式来看，p 选用 150 位以上的十进制整数是比较合适的，并且 p-1 至少有一个大的素数因子。

ECDSA 算法属于 DSA 算法的一个变种，基于椭圆曲线加密算法（Elliptic curve cryptography，ECC），椭圆曲线加密算法由 Neal Koblitz 于 1987 年提出[18]。这种加密算法相对于其他非对称加密算法的主要优点在于：在达到相同等级的安全性的情况下，椭圆加密算法所需要的密钥长度更短，有利于存储和传输。比如，一个 256 位长度的椭圆加密算法公钥，可以提供等同于一个 3072 位长度的 RSA 加密算法公钥的安全性[19][20]。ECDSA 算法与 DSA 算法基本一致，主要不同点在于 ECDSA 算法使用的是椭圆曲线上的域，而 DSA 算法使用的是普通乘法域。

2.3 分布式系统通讯协议

在分布式系统中，节点之间按照约定的规则依托网络进行信息交换，以在节点之间实现状态同步，这套规则就是分布式系统的通讯协议。

在一个中心化管理的分布式系统中，不同节点之间的状态同步通常很容易实现，一个简单有效的方法就是使得每个节点都通过同一个中心节点来获取状态。但是在系统规模非常大的情况下，设置这样一个中心节点也可能现实或者是不经济的，并且这一节点的任何波动或者错误都可能影响整个系统，极易变成整个系统的

瓶颈。

对于去中心化的分布式系统，目前应用较多的解决方案是 Paxos 协议[21]和 Gossip 协议[22]，其中 Paxos 协议较为复杂，用于在节点之间保持强一致性，维护这种强一致性需要很大的系统开销，伸缩性较差，通常适用于节点数量较少的系统。Gossip 协议较为常用也易于实现，可适用于非常大型的分布式系统，符合本论文所描述的系统的需求。该协议传播模式类似于人类之间传播流言的行为，并因此而得名，是一种被用在分布式的非强一致性系统中用来同步各节点状态的方法，最终所有节点的状态将会是一致的，Gossip 协议并不要求当前节点知道所有其他节点，节点之间完全对等，具有去中心化的特点。

Gossip 协议于 1987 年由 Alan Demers 等提出，以用于解决复制型数据库的一致性问题。近年来随着大型分布式系统架构的兴起，以及 Gossip 协议的简洁性和强健性，其受到越来越多的关注，很多现代的分布式系统都选择使用 Gossip 协议来解决节点之间的通信问题，如著名的分布式数据库管理系统 Cassandra 和点对点文件共享软件 Tribler。

分布式系统中的 Gossip 协议有两种较为常见的实现形式，这两种形式通常都被结合起来同时使用，分别为：

- 反熵（Anti-Entropy）：节点每隔固定时间段与其他节点交换信息，发现不同即开始同步
- 扩散（Rumor-Mongering）：节点一旦收到更新，便通知其他节点

反熵模式用伪代码表示如下：

```
while true do:
   wait(time);
   p = selectPeer();
   prepare(m);
   sendTo(m, p);
   pm = receive(p);
   process(m, pm);
```

扩散模式用伪代码表示如下：

```
while true do:
   pm = receive();
   p = sender(pm);
   prepare(m);
   sendTo(m, p);
   process(m, pm);
```

以上的操作都是在一对节点之间完成，每个节点 P 每隔一段时间选取一个其他节点 Q，节点 P 从本地选取信息分享给节点 Q，同时节点 Q 也回应其本地的信息给节点 P，双方各自收到信息后做相应的处理。不停的重复这一过程，在没有进一步更新的情况下，这一信息对于整个系统的所有节点的覆盖率是随着时间指数级增长的，也就是说可以保证系统最终达到一致性[23]。

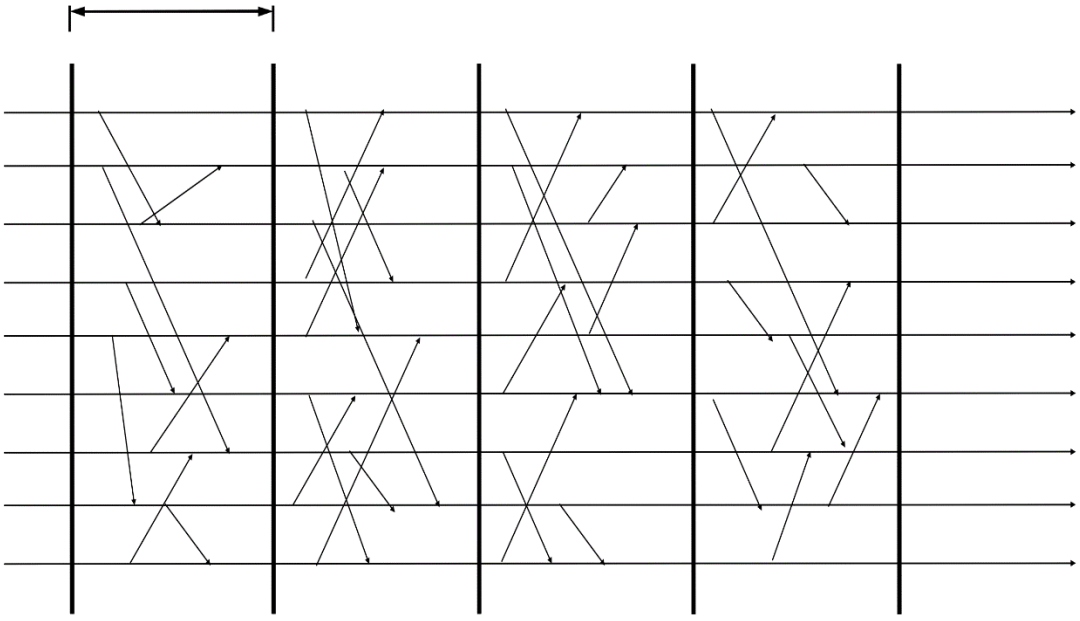


图 2-3 Gossip 传播示意图

信息的选取和信息的处理的问题都基本可以归类到应用层面，最为关键的是如

何选取节点，这将关系到最终的传播效果。Gossip 协议要求在选择一个节点进行信息交换的时候，这一节点必须是从整个系统当前所有节点中随机抽取出来的。一个可能的方案是每个节点都维护一张整个系统的成员表[24]，其维护的难度也将随着系统节点数量的增长而增长，这对一个任意组织的去中心化系统显然是不现实的。

针对这个问题，论文[25]和论文[26]分别提出了两种方案 Newscast 和 Lpbcast，这两个方案均是在只已知一部分系统节点的情况下，达到伪随机抽取节点的目的。论文[27]把这种伪随机化抽取节点的方案过程进行了提炼，将其分为三个关键的子过程，分别为节点选取（Peer Selection）、视图选取（View Selection）和视图传播（View Propagation），前面提到的两个方案均可以通过这三个关键子过程来描述。

节点选取可分为三种形式：

rand	从当前节点的已知节点列表中随机选取
head	从当前节点的已知节点列表中选取第一个节点
tail	从当前节点的已知节点列表中选取最后一个节点

视图选取可分为三种形式：

rand	当前节点从自己的节点列表中随机选取一定数量的列表
head	当前节点从自己的节点列表中选取最前的一定数量的列表
tail	当前节点从自己的节点列表中选取最后的一定数量的列表

视图传播可分为三种形式：

push	当前节点向选取节点发送自己的节点列表
pull	当前节点向选取节点获取节点列表
pushpull	双方节点各自合并对方节点列表

那么，Newscast 可以描述为（rand, rand, pushpull），Lpbcast 可以描述为（rand, rand, push）。

并且论文[27]针对 Newscast 和 Lpbcast，以及三个关键子过程的其他四种组合，进行了模拟测试，以研究各种伪随机的节点选取方案所形成的网络的特点。模

拟测试包含三种完全不同的初始网络结构：第一种模拟现实中的不断增长的分布式网络，网络初始为 1 个节点，每轮传播加入 100 个新节点，新加入的节点只有初始节点的信息，第二种初始为一个环形结构，第三种则是一个完全随机组成的网络结构，并且分析平均路径长度、聚集系数和平均节点度，分别如图 2-4、图 2-5 和图 2-6 所示。将图 2-4 和图 2-5 分别与图 2-6 对比，可以看到，（*, rand, pushpull）组合的三个值的分布最为接近随机组成的网络。

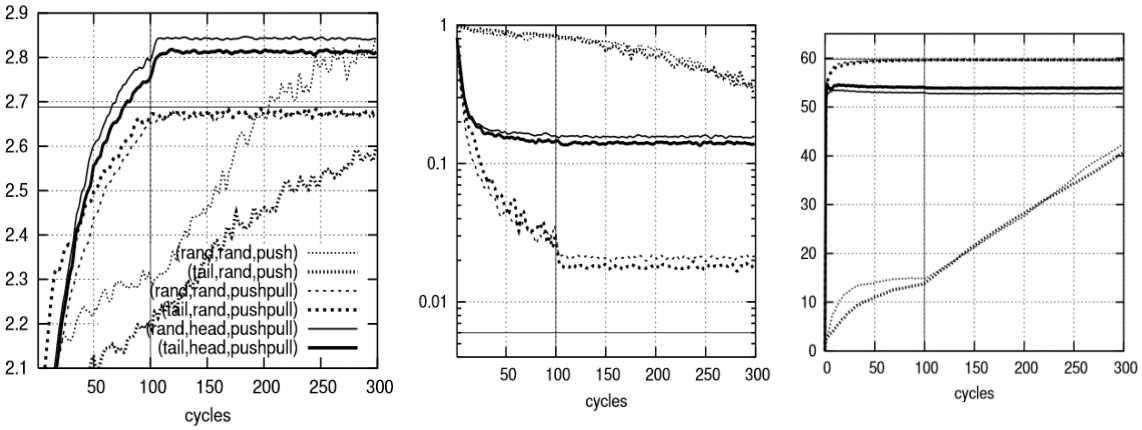


图 2-4　模拟现实中的网络

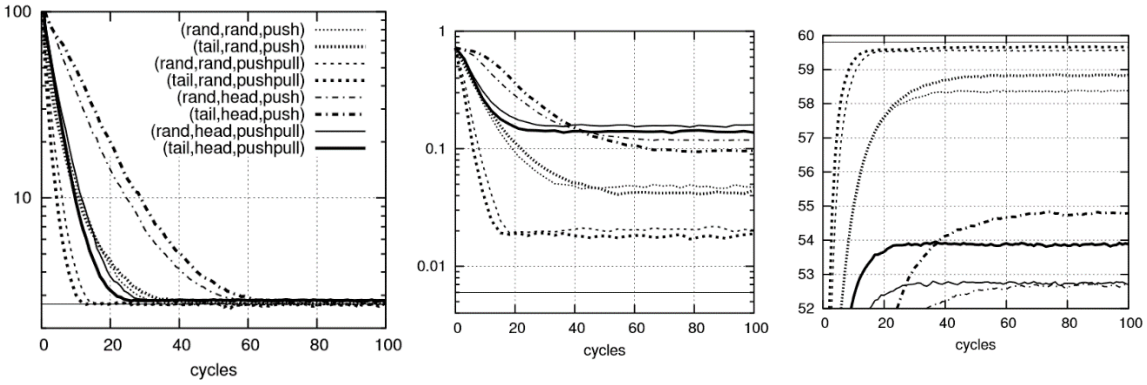


图 2-5 初始为环形结构的网络

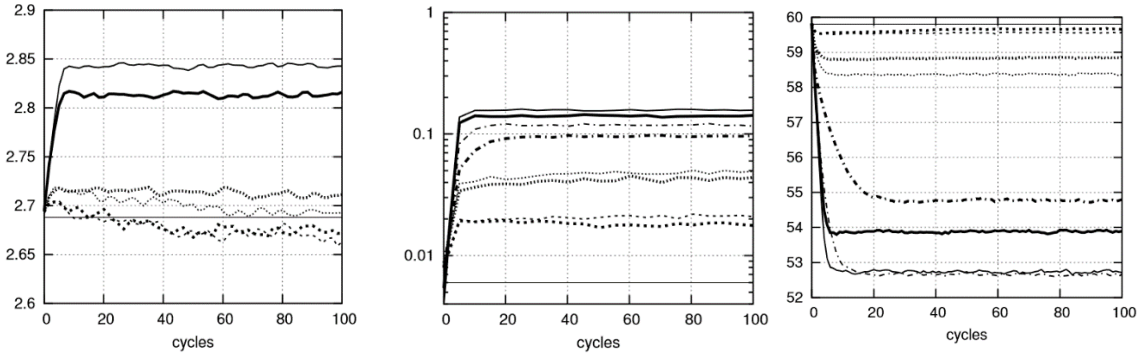


图 2-6 随机组成的网络

2.4 本章小结

本章对系统设计中应用的关键技术进行了分析，包括加密散列算法、数字签名算法和分布式系统通讯协议，这些关键技术是本系统重要的组成部分，在后面的系统具体设计以及实现中都会用到。

3 去中心化网络社区系统的分析和设计

3.1 系统需求分析

对于一个社区系统来说，用户以及用户发布的内容是最为重要的，随之而来的问题就是：

（1）如何鉴别用户，即如何正确可靠地将用户的身份区分开来，使得用户身份无法被冒用

（2）如何安全地存储用户发布的内容，即保证用户发布的内容不会出现丢失损坏的情况

在中心化模式的社区系统里，用户数据和用户发布的内容都存储在中心服务器中。用户创建身份时需要提前将身份信息提交保存到服务器，一般为用户名/密码的形式，系统首先验证这些信息，保证其不与现有的用户信息相冲突，然后，将用户信息保存在服务器上。通常，为了能可靠地验证用户身份，需要完整地将用户信息保留，包括密码原文，但是出于安全考虑，防止在系统出现漏洞被非法侵入后原文密码泄露对用户利益造成损害，一般只保存密码的散列摘要，通过散列摘要同样可以有效地对用户名/密码进行鉴别。在系统完成对用户身份的鉴别后，该用户方可在系统上发布内容，所发布的内容，同样需要集中存储到中心服务器上，并标注所属用户。也就是说用户身份的鉴别以及用户内容的存储和分发，都是透过中心服务器完成，用户并不能确切地鉴别其他用户的身份，也无法确切地得知用户内容的真正所属，为了保证数据的安全，这些过程无法做到对用户透明。

在本文所要讨论的去中心化模式的社区系统里，所有节点角色都是平等的，无法像中心化社区系统那样选择信任固定的中心节点，并由它们来负责用户身份鉴别和用户内容存储与分发。因此，首先需要考虑的是节点之间信息同步的问题，保持节点之间的状态尽可能地一致，使得用户发布的新内容可以以尽可能快的速度通知到网络中的所有其他节点，各个节点也可以获取到较为完成的历史数据，以便查

询。其次，我们需要考虑如何独立地完成用户身份地鉴别，即不要其他节点的参与，只通过已有数据便能确认用户身份。最后，我们需要考虑实现在各个节点之间保存历史数据，即所有用户所发布的内容的历史存档，这一份历史数据需要在各个节点上保持基本一致，并且非常难以篡改。

另外需要强调的是，本系统将网络社区系统简化为其最基本的形式，并主要着眼于网络社区系统的最基本要素，即内容发布和内容分发，重点描述如何实现“去中心化”，而不去赘述目前已经很成熟的网络社区系统业务形态。

3.2 系统架构设计

一个节点表示一个运行系统程序的计算机设备，系统由一定数量的这种节点构成。节点之间通过网络相互连接和交换信息，节点的连接和相互之间的信息交换遵循事先定义在系统程序中的协议，这里我们采用 Gossip 协议，以实现网络中的所有节点能够快速同步状态。

系统上运行的每一个节点都可以发布文章，同时也负责文章消息的转发，以及对文章的存储。节点发布的每一篇文章都使用 SHA-256 加密散列算法生成散列摘要，并使用 ECDSA 数字签名算法结合节点的私钥进行签名，以表明文章所属，任何节点都可以对签名进行验证。发布文章时选择一定数量的其他节点将构造的消息发送过去，收到文章消息的其他节点在对齐进行验证后，再将文章消息转发出去，以实现文章消息在系统中的快速和广泛传播。

系统上的节点共同维护一份相同的数据集，称之为区块链。系统上的需要持久保存的数据，主要为文章数据，最终都会被节点存储到区块链中。区块链由分块存储的区块构成，每个区块中存储一定数量的数据，并单向链接前一个区块，形成链式结构，类似于向前单向链接的链表。

系统架构如图 3-1 所示，下一节中系统架构设计将被细分为系统功能进一步详细阐述。

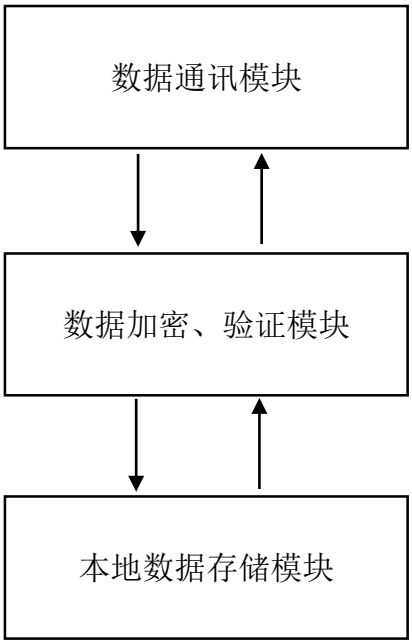


图 3-1 系统架构图

3.3 系统功能设计

节点间的通讯

1. 节点发现

在去中心化的分布式网络中，当一个全新的节点启动时，它没有关于其他节点的任何信息，因此也就无法该网络中，处理这种情况需要对该节点进行初始引导（Bootstrap），以使得其可以获得该分布式网络的入口，通常为至少一个已经连接到该分布式网络的其他节点的信息，只要连接上至少一个已经在该分布式网络中的其他的节点，即可视为该节点加入到了该分布式网络中，后续该节点可以通过与网络中的其他节点的信息交换，获取更多的其他节点信息。

初始引导一般如下有几种种方式[28]：

- 节点不间断地对不同的网络地址的对应端口号进行扫描，如该特定端口开放，并且能够正常连接，则尝试握手
- 在节点所使用的程序中嵌入一定数量的已知节点信息，这些节点长期在

线，并且相对安全

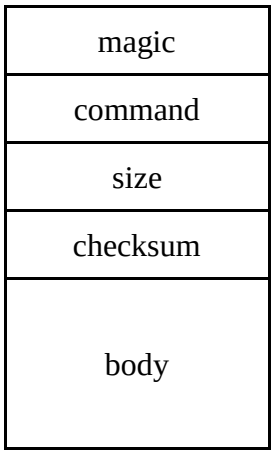
- 由用户自行通过其他途径，向其他信任用户获取节点信息，如电子邮件或互联网聊天室，并加入到节点所使用的程序中

每当节点成功建立一个连接后，都将对方节点的地址信息保存到本地，以便下次启动时使用。并且向对方节点获取其所持有的成功连接上的地址信息，然后依次尝试连接，连接成功后同样保存到本地。这样，在该节点下次启动时便可以利用这些节点地址信息快速地加入网络，而不必再次进行初始引导。

2. 节点通讯

节点与节点之间的通讯，均以消息为单位，节点上发送消息，既可以单独发送给某个节点，也可以广播发送到当前所有已连接的节点。

消息的结构如下：



其中，消息头固定为 24 个字节，分别为：

magic：固定的 4 字节整型，用来在消息传输过程中标识消息的开头

command：12 字节字符串，用来表示消息的类

size：4 字节整型，用来表示消息体的长度

checksum：4 字节长度，用来校验消息体，防止传输错误，为消息体 SHA-256 散列摘要的前 4 个字节

消息体为不定长度，由消息头中的 size 字段确定，为标准的 JSON（JavaScript Object Notation）格式[29]的字符串，并且保持为紧凑格式，关键字保持为字母顺序排列，以防止出现相同内容的 JSON 字符串所得的散列摘要或者签名不同。

用户身份的鉴别

本系统采用椭圆曲线数字签名算法（ECDSA）来实现用户身份的鉴别，对于加入到系统中的每一个用户（或者说节点，一个节点只对应一个用户），都需要一对椭圆曲线数字签名算法的密钥，这一对密钥由系统随机生成，使用其中的公钥作为用户的唯一公共标识，同时也用来验证签名，私钥由用户保管，不能对外公开，用来对其发布的内容作签名，以证明该内容确实由其发布，并且没有被篡改。

椭圆曲线加密算法中的关键部分是选择一条椭圆曲线，及其相关参数，这里我们选择使用美国国家标准与技术研究院（NIST）推荐的十五条椭圆曲线之一：NIST Curve P-256，其私钥长度为 256 位，公钥长度为 512 位，将公钥进行 Base64 编码后长度为 88 个字节。

由于私钥是从一定区间内随机选取的一个整数，因此存在可能性两位不同用户所得到的自动生成密钥相同，这个情况出现的概率虽然不为零，但是低到足以忽略不计。因为，美国国家标准与技术研究院（NIST）所推荐椭圆曲线：NIST Curve P-256 所给定的私钥范围为[30]：

115792089210356248762697446949407573529996955224135760342422259061068512044436

这是一个十分巨大的数字，约等于 2 的 252 次方，在随机选取值均匀分布的情况下，这种可能性可以认为是 0。

用户内容的发布

在网络社区系统里面，用户发布的内容通常为一篇文章或者针对某篇文章的评论，评论也可以有子评论。为了简化设计，在本系统里面将所有内容统一为文章的形式，如某文章指向另一篇文章，则视为对另一篇文章的评论，如果没有指向，则

视其为主题文章。

当用户发出一篇文章，系统程序将其包装为一则文章类型的消息，广播给所有已连接的节点，其他节点在收到该消息后，同样将其广播给其当前所有已连接的节点。为了防止引起“广播风暴”，即重复的消息不停被广播，导致占用大量网络带宽，节点间的正常通讯受阻，节点在收到广播类型的消息后，需要首先判断其是否已经是一条已经存在的文章类型消息，如是则将其忽略，否则将其保存到本地，再广播给所有已连接的节点，以尽可能广地传播该消息。

所有文章类型的消息都必须包含用户的公钥和对应的签名，以及一个 SHA-256 散列摘要，用来唯一标识一篇文章，节点收到该类型的消息后，首先需要检查其散列摘要和签名是否相符，否则将其忽略。

文章类型消息的消息体结构如下：

```
{
   "hash": "",
   "signature": "",
   "post": {
      "category": "",
      "title": "",
      "content": "",
      "timestamp": 12345,
      "username": "",
      "userkey": "",
      "parent": ""
   }
}
```

其中，“post”键值代表文章主体的数据，“hash”和“signature”分别为“post”键值中的子键按字母顺序排列紧凑排列后的 SHA-256 散列摘要和 ECDSA

椭圆曲线签名。“post”键值中包含“userkey”子键，对应为该文章作者的公钥，使用此公钥来验证签名。

用户内容的存储

如何在一个非信任的分布式网络上安全可靠地存储用户内容，是本系统所需要解决的最为关键的问题。由于节点之间的非信任关系，因此需要一种机制来使得各个节点对于这份数据可以达成一个共识，即大多数诚实节点都可以在同一份数据上进行操作，而恶意节点又非常难以对这份数据进行篡改。为了解决这个问题，本系统采用了在现代数字加密货币系统中得到普遍应用的区块链技术[31][32]。

区块链可以认为是一个简单的分布式数据库，其中维护的数据可以在各个节点上保持基本一致，数据按时间先后顺序累计，每达到一定的量就形成区块保存，除第一个块之外，每一个块都包含其前一个区块的索引，形成一个单向链条，因此称之为“区块链”。

为了形成一个区块，节点需要通过使用其计算能力，计算解决一个非常耗费时间的问题，比如在区块数据之上计算 SHA-256 散列摘要，并不断更改其中某个值，使得这个区块数据的散列摘要值小于给定的值。在区块数据被部分更改后，再次计算出来的 SHA-256 散列摘要值可以认为是随机的，因此搜寻一个小于给定值的散列摘要值也就类似于彩票摇奖过程，摇奖的次数越多，给定的彩票号码中奖的概率就越大。一旦节点成功得到这样一个散列摘要，便形成了一个有效的区块，于是通知其他节点，其他节点在收到并验证该区块后，将其加入到自己的本地数据，放弃其正在运算的区块，重新开始以这个新的区块为指向，计算下一个区块。这一过程称为工作量证明（Proof-of-Work）[33]，另外，我们将工作量证明最终得到散列摘要所要小于的给定值称为目标（Target）。

```
{
   "hash": "",
   "nonce": 0,
   "block": {
      "height": 0,
      "prevblock": "",
      "merkleroot": "",
      "timestamp": 123456,
      "target": 123456
   },
   "posts": [
      "",
      ""
      ...
   ]
}
```

具体到本系统中，区块的基本结构如上图所示，其中：

hash：表示区块数据的散列摘要

nonce：为任意数字，使得将其追加到区块数据后计算出的区块散列摘要小于目标值

posts：该区块所包含的所有文章的数据

block：包含代表一个区块的基本数据

height：表示该区块在区块链中的位置

prevblock：表示该区块所链接的前一个区块的散列摘要

merkleroot：posts 键值中散列摘要的 Merkle 树的根值

timestamp：表示组成这个区块的时间，为 UNIX 时间戳

target：表示当前区块的散列摘要是以小于这个值为目标来计算的

节点在启动一次工作量证明运算之前，尽可能地搜集目前的未加入到任何区块的文章，并结合其来源消息，对其进行验证。然后，将所有这些文章数据的散列摘要提取出来，按照散列摘要的字母顺序进行排列，计算得出其 Merkle 根[34][35]。最后，将这个计算得出的 Merkle 根包含进一个待进行工作量证明运算的区块数据中，开始进行工作量证明运算。

为了计算出当前区块所包含的文章的 Merkle 根，首先将表示文章的散列摘要全部提取出来，并按照字母先后顺序排列，将这些散列摘要作为一颗二叉树的叶子节点。为了得到两个节点的父节点，将两个节点的散列摘要值前后拼接，然后计算 SHA-256 散列摘要，作为其父节点。如此反复，直到最终得到一个散列摘要值，这个就是代表所有文章的 Merkle 根。当某一个高度的节点数量为奇数时，最后一个节点使用本身的两个相同散列摘要值前后拼接。计算 Merkle 根的过程大致如图 3-2 所示。

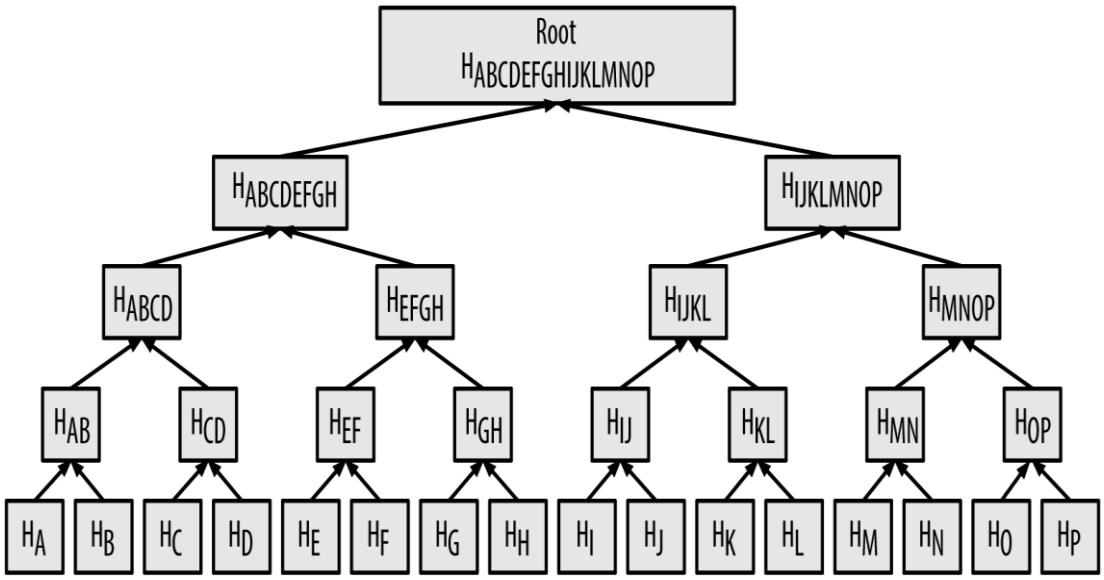


图 3-2 计算 Merkle 根的过程

使用散列摘要来表示文章，可以使得区块数据和文章数据有一定分离度，方便网络传输。同时，该散列摘要又能对最终完整的文章数据进行验证。采用 Merkle 树的方式提取散列摘要，而不是直接将所有文章的散列摘要值拼接在一起来计算，有利于在文章数据有部分变化的情况下，提高重新计算代表所有文章数据的散列摘要值的效率。一个节点在工作量证明运算中能够成功得到一个满足预先给定目标值的散列摘要值的概率是相对较低的，因此，需要多次对其所运算的文章数据进行调整，如剔除已被包含进区块的文章，增加文章数据等。对于这种情况，使用 Merkle 树根散列摘要值来代表所有文章数据，并在运算过程中保留这个 Merkle 树的数据结构的意义便凸显出来，只需要重新计算变更过的叶子节点上升到根节点的路径上的散列摘要，而不需要重新计算所有文章数据就能再次得到代表所有文章数据的 Merkle 根。

如当前共有 N 个文章数据，当其中一个发生变化时，只需要重新进行 $\log_2 N$ 次计算，便能再次得到 Merkle 根，如图 3-3 所示，假设 H_k 发生变化，那么只需要重新计算虚线框中的几个值。

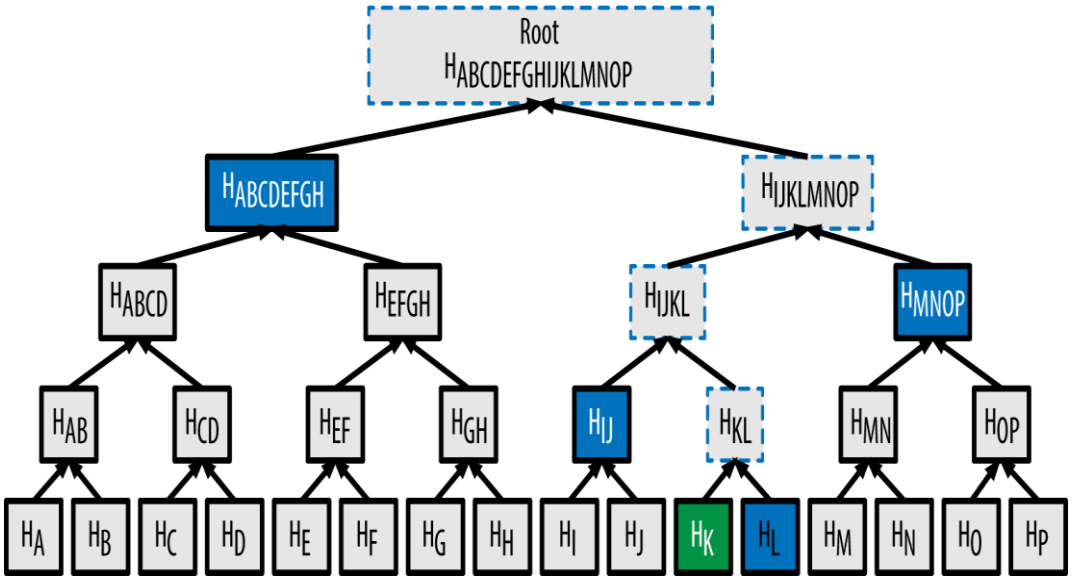


图 3-3 重新计算 Merkle 根的过程

在计算得出 Merkle 根之后，系统程序查询本节点上已知的最新的区块，并以其为基准设置当前区块的高度和及其链接指向，然后根据本机时间设置当前区块的 UNIX 时间戳，并加上工作量证明的目标值。这样，便组成得到一个区块，但是要使得这个区块被其他节点所认可，还需要一个小于目标值的代表整个区块数据的散列摘要。为了计算得到这个散列摘要，首先将 block 键值中的子键按字母顺序、以紧凑格式输出为标准的 JSON 字符串，而后在这个字符串后面拼接一个整数，称之为 nonce，这样，每次改变这个整数值便会得到一段不同的数据，也就会计算得出一个完全不同的散列摘要值。持续不停的改变 nonce 值，直到得到符合目标值的散列摘要为止。

在得到符合目标值的散列摘要后，节点首先去掉文章数据，将这个不完整的区块数据向其所有已连接的节点广播，使得这个广播消息的长度较短并且固定，以便快速的传播。其他节点在收到这一消息后，首先在不依赖具体的文章数据的情况下，对其进行初步验证，如验证通过，再向对方节点发送消息，请求完整的区块数据。在收到完全的区块数据后，再次对其进行验证，这次的验证需要包括具体的文章数据。

节点对于从其他节点接收到的新区块的验证包括多个方面：

- 该区块的格式符合要求
- 该区块本身声明的散列摘要值与实际计算出的区块数据的散列摘要值相符
- 该区块本身的散列摘要值在本节点上不存在
- 该区块所指向的父区块在本节点上存在
- 该区块的散列摘要值小于目标值
- 该区块的时间戳在可接受范围内
- 该区块所包含的文章数据，均符合预先约定的规则

在对新区块完成验证和存储后，该节点也向其已连接的其他节点广播新区块的消息。节点在进行工作量证明计算的过程中收到新区块的消息，如验证通过，则放

弃正在进行的工作量证明计算，并检查是否有与新区块中相同的文章，如有则将其剔除，然后以新的区块为基准重新开始工作量证明计算。

目标值的自动调整

设定一个工作量证明计算的目标值的目的在于使得区块的发现需要耗费一定的时间，那么相应的改写区块链同样需要耗费对应的时间，但是随着科技的进步，计算机硬件的计算能力逐步增长，这个时间会变得越来越短，所以，一个恒定的目标值显然是不合适的，它需要能够根据系统中的计算能力来自动调整。

每个区块中都记录有区块形成的 UNIX 时间戳，那么区块 B_n 到区块 B_m 之间所花费的时间可大致认为是 T_m-T_n（T 为 UNIX 时间戳），如果 T_m-T_n 大于预期的时间，那么就认为当前目标值相对于系统中的计算能力来说难度较高，于是将目标值调高，如果 T_m-T_n 小于预期的时间，那么就认为当前目标值相对于系统中的计算能力来说难度较低，于是将目标值调低。

假设预期每 1 分钟产生一个区块，并且以一个星期为周期调整目标值，一个星期将大致产生 60×24×7=10080 个区块，那么系统便每隔 10080 个区块重新计算目标值。假设 B_n 区块的 UNIX 时间戳为 T_n，$B_{n+10079}$ 区块的 UNIX 时间戳为 $T_{n+10079}$，区块 B_n 至区块 $B_{n+10079}$ 使用的目标值为 T，那么区块 $B_{n+10080}$ 的目标值 $T'=(T_{n+10080}-T_n)\div(60\times60\times24\times7)\times T$。

目标值自动调整的计算完全由节点根据已有区块数据独立进行，不依赖其他节点，但是区块数据在节点间是统一的，所以最终得到的新目标值可以做到节点间相同。

3.4 系统数据库设计

每个节点需要在其本地保存一份区块链数据，理论上这部分数据可以采用任何类型的数据，但是考虑到系统程序的应用弹性，采用数据库是比较合适的选择。系统设计上只有区块数据需要存储在本地，但是因为其中包含的文章数据需要向用户

进行展示，因而可能需要在文章数据之上进行较为复杂的查询。如果文章数据包含在区块数据之中，那么这些查询将会变得非常繁琐，并且非常耗费系统资源。

基于以上考虑，将区块数据分为两张表来存储，分别对应区块数据和文章数据。Block 表用来存储区块的数据，Post 表用来存储文章数据。为了避免产生数据冗余，节省存储空间，在插入区块数据之前，将其中的文章数据移除，只留下散列值。获取区块数据的时候，再通过其中的散列值，从 Post 表中提取相应的文章数据，将散列值填充为完整的文章数据。具体结构如表 3-1 和表 3-2 所示。

表 3-1 Block 表结构

Block 表			
列名	类型	主键	描述
hash	text	是	区块数据的散列摘要，用来唯一标识区块
height	integer	否	区块在区块链中的位置
parent	text	否	所指向的前一个区块的散列摘要
timestamp	integer	否	区块创建时间，UNIX 时间戳
data	blob	否	区块整体数据，紧凑格式的 JSON 字符串

表 3-2 Post 表结构

Post 表			
列名	类型	主键	描述
hash	text	是	文章数据的散列摘要，用来唯一标识文章
timestamp	integer	否	文章发布时间，UNIX 时间戳
userkey	text	否	文章作者的公钥
category	text	否	文章所属分类
parent	text	否	文章上级，空表示没有上级，为主题文章
data	blob	否	文章整体数据，紧凑格式的 JSON 字符串

3.5 系统安全性设计

算法安全

本系统的安全性主要取决于 SHA-256 加密散列算法和 ECDSA 数字签名算法的安全性，因为这两套算法在本系统中扮演了重要角色，本系统的设计都是在假定这两套算法安全的基础上展开的。到目前为止，SHA-256 加密散列算法和 ECDSA 数字签名算法都没有被报告有明显的漏洞，并且本系统所选区的 SHA-256 加密散列算法的散列空间大小为 256 比特位，ECDSA 数字签名算法也为 256 比特位长度，均被认为暴力破解难度是相当高的，是事实上安全的。

伪造身份

前面提到了本系统采用的 ECDSA 数字签名算法来保证用户身份不被伪造，这一种应用了非对称加密算法的数字签名技术，每个用户拥有一对密钥，包括私钥和公钥，公钥作为用户标识对外发布内容，私钥对用户内容进行签名，其他用户使用公钥对签名进行验证，数字签名的特性决定了用户内容是无法被增加、删减或者修改的，签名验证失败的数据既无法经由诚实节点传播，也无法进入到区块链中。更有可能的情况是用户丢失了密钥，这种情况本系统将是无能为力的，而只能是由用户本人发现其公钥所产生的异常行为，也就是发布了用户本人并不知情的内容，并重新生成新的密钥对。

修改区块链

在一个由非信任的节点自发组成的分布式网络中，存在恶意节点是可能的。对于本系统来说，恶意节点的目的可能是希望通过改写区块链来删除用户内容，理论上这是可行的，并且一旦成功也将被系统中的其他节点所接受，但是难度视恶意节点所持有的计算能力和整个系统中诚实节点所持有的计算能力总和而定。发现新的区块需要消耗大量的计算能力，那么相应的更改这一区块，也需要消耗同样的计算能力。在这一基础上，如果某个恶意节点想要篡改某个区块，它必须将重做包括这

一区块及之后所有区块的工作量证明。那么，这一恶意节点必须拥有整个系统计算能力超过 50%的份额，才有可能在未来某个时间点使得其篡改后的区块链被整个系统认可，所需要的时间长短视其所有持有的计算能力份额而定。

区块链分支

同一时间点，理论上系统中的大部分节点都在做工作量证明计算，一个节点做工作量证明计算并最终使得散列值小于目标值实际上可以认为是一个随机的过程。那么就存在可能性，两个节点或者更多的节点在非常相近的时间内都发现一个新的区块，这些区块都指向同一个父区块，或者说这些区块位于区块链中的同一高度，并且各自节点都能成功将发现的区块广播出去。这些区块实际上都是有效的，符合节点对新区块进行检查的所有规则，那么其他节点也都将接受这些区块，并将它们插入到自己的本地存储中。

这种情况便称作区块链的分支，可能会造成系统的不稳定，但是不足以对系统造成破坏。因为，其他节点可以始终以最新、最长的那条区块链分支为准，也就是说节点在进行后续的工作量证明计算，或者进行本地数据查询的时候，如果存储在本地的区块链中有分支，则首先选择长度最长的那条区块链，如果长度最长的区块链仍然超过 1 条，则选取最先接收到的，或者最先发现的区块所属的那条链。这个时候分支之间将会形成一种竞赛，但是最终会回归到同一条分支上去，另外分支则被抛弃。

另外一种造成区块链分支的情况则是系统程序的更新，可能出于某些原因，比如安全性问题或者重大的程序改进，而不得不更改区块的格式，并且无法做到向前兼容，那么就不得不不再接受由运行旧版程序节点所产生的区块，运行旧版程序的节点很可能也无法接受由新版程序所生成的区块。这种情况下，系统将无法自动协调解决这些分支，而只能寄希望于所有节点都升级到新版程序，这个时候区块链最终将统一到由新版系统程序所产生的那条分支，而另外一条由旧版程序所生成的分支将被废弃。

网络嗅探和中间人攻击

本系统中所有节点间通讯的网络传输均是明文的，没有经过任何安全层加密，因此任何网络路径上的第三方都能查看到系统中的节点之间传输的内容，但是这样做没有实际意义。因为本系统实质上是开放的，任意运行系统程序的节点都可以加入进来，并且最终生成的所有内容也是对所有人公开的，因此并不存在任何安全性问题需要进行加密来保证。

因为节点间的通讯是明文，所以也存在中间人攻击的可能性，经过数字签名的消息，可以很容易地通过签名而检测到改动，而其他未经数字签名的消息，均为最终不会进入到区块链的内容，对这些消息的篡改，最坏的情况是导致节点无法正常工作，而无法破坏整个系统。节点如果担心遭到中间人攻击，启用加密传输也是非常容易的，因为加密传输并不涉及系统协议的改动，而只需改动系统程序。

3.6 系统伸缩性设计

本系统中区块链数据是不断累积的，假设每个区块包含 1024 篇文章，区块按照预定每分钟 1 个的速度生成，那么每年的将会生成 60×24×365=525600 个区块，包含的文章总数将为 525600×1024=538214400 篇，约等于 5.4 亿篇，这是一个非常巨大的数字，以目前世界上用户最多、最为活跃的网络社区平台 reddit 参照，根据其在 2015 年 6 月 23 日公布的最新统计数据显示，该社区平台从 2005 年至 2015 年 6 月所累积的所有文章和评论数量约为 19 亿，所以 5.4 亿篇是一个在短时间内非常难以达到的数字，是非常有富余的估计。再假设每篇文章为 1K 字节大小，那么一年时间内所产生的区块链的数据大小约为 513GB，这是一个不小的数字，但是对于当前的普通个人电脑所配备的硬盘空间来说仍然是可以接受的，并且随着计算机技术的不断进步，普通消费者所能购买的磁盘空间也在不断增长。尽管如此，对于区块链的数据大小仍可以从以下几点做简单优化。

首先，区块链中的数据实际上是文本数据，在本地存储区块链的时候，可以对其进行适当压缩，以目前流行的 GZIP 快速压缩解压缩算法来看，对于文本数据的

平均压缩率可以达到 30%[36]。

其次，对于一定时间之外的文章数据，节点可以选择不存储到本地，而只需要存储区块的数据，那么对于不包含文章数据的区块链，一年时间所需要的存储空间将减少到约为 513M，如果再启用 GZIP 压缩，存储空间将进一步减少到约为 153M。

3.7 本章小结

本章首先分析去中心话网络社区系统的基本需求，将系统分为两大基本需求，即系统安全和系统数据存储。从这两个基本问题出发，提出了使用 SHA-2 加密散列算法和 ECDSA 椭圆曲线数字签名算法的安全性设计方案，和使用类 Gossip 分布式通讯协议的系统节点间的通讯设计方案，以及安全的分布式数据存储方案。接下来进一步说明了可能面临的安全性问题和伸缩性问题，并针对这些问题进行了分析说明。

4 去中心化网络社区系统的实现

本系统主要采用 Python 编程语言实现，版本为 3.5.1，为 Python 编程语言的下一代版本，相比 Python 2，其语法更加完善，并且支持更多现代语言的特性。Python 是一种面向对象、解释型的计算机编程语言，具有近二十年的发展历史。它的语法简单，标准库比较完备，有利于提高开发效率。另外，由 Python 编程语言写成的程序无需编译就能在当前主流的计算机软硬件平台上运行，免去了跨平台移植的麻烦，这一特性非常适合编写面向普通用户的程序。

网络连接部分是基于 TCP/IP 协议的，但是为了简化开发，采用了 Tornado 网络编程框架。Tornado 基于 Python 编程语言的，并且提供了基于 Python3 的版本，最初有 FriendFeed 所开发，FriendFeed 被 Facebook 收购后，Tornado 以开源软件的形式提供给大众使用。得益于非阻塞式的 IO 模型和对 epoll 的运用，Tornado 每秒可以处理数以千即计的网络连接，非常适合开发实时应用。

本地数据存储部分使用数据库管理系统实现，但是传统的数据库管理系统显然是不合适的，首先，它们都需要进行独立安装、独立运行，并且对计算机资源的消耗较大，很难被普通用户所接受。因此，本系统决定采用 SQLite 数据库管理系统，SQLite 是遵守 ACID 的关系数据库管理系统，实现了大多数 SQL 标准，它包含在一个相对小的 C 程序库中，与许多其它数据库管理系统不同，SQLite 不是一个客户端/服务器结构的数据库引擎，通常是直接被嵌入到程序中，因此其应用非常广泛，如著名的 Firefox 浏览器就采用了 SQLite。

对于用户界面部分，本系统采用 B/S 模式，即 Browser（网页浏览器）/Server（服务器）模式，但是并不是依赖远程服务器，而是只在本机上运行一个简单的 HTTP（超文本协议）服务器，通过网页浏览器向用户提供交互界面。这样做的优点在于，只要遵守 Web 开发的标准即可保证用户界面可以成功地运行在各种主流的操作系统之上，包括移动操作系统（Android，iOS 等等），保持用户体验基本一致，因为这些操作系统通常都带有实现了常见 Web 标准的网页浏览器。如果采用通

常的操作系统原生用户交互界面，那么将不得不针对不同的操作系统进行特定的适配，这是十分耗时耗力的，并且难以在各个操作系统上保持统一。

4.1 系统开发环境

本系统主要在 Arch Linux 操作系统之上开发和运行，使用了 Python 集成开发环境 PyCharm，版本为 Community Edition EAP, build 143.1453.1。PyCharm 由捷克公司 JetBrains 开发，能够跨平台运行在 Windows、Mac OS X 和 Linux 操作系统上，是一款非常强大的 Python 集成开发工具，除了提供基本的 Python 代码编辑和运行功能之外，还包含了代码分析、图形化调试器、单元测试工具、版本控制工具集成和 Web 开发支持等等功能，对提高开发效率很有帮助。

4.2 SHA-2 散列算法的实现

SHA-2 是一系列加密散列函数的统称，它包括 SHA-224、SHA-256、SHA-384、SHA-512，后缀分别对应不同的比特位长度的散列摘要，224 比特位和 384 比特位分别由 256 比特位和 512 比特位截取而来。

SHA-256 函数将数据分为 512 比特位大小的块，每个块进行 64 轮操作，使用了 64 个 32 比特位长度的常量 $K_0 \cdots K_{63}$，和 8 个带固定初始值的临时变量 $H_0 \cdots H_7$，它们的具体值可以在[7]中查到。在进行处理之前需要将数据对齐到 512 比特位长度：

（1）设数据比特位长度为 L

（2）在数据末尾追加一个比特位“1”

（3）末尾追加 k 个比特位“0”，使得 $L+1+k \bmod 512 = 448$

（4）在数据末尾追加一个 64 比特位长度的整数 L，大端序排列

SHA-256 的伪代码定义如下：

```
SHA-256(M):
   For each 512-bit block B in M do
      W = f_exp(B);
      a = H0;
      b = H1;
      c = H2;
      d = H3;
      e = H4;
      f = H5;
      g = H6;
      h = H7;
      for i = 0 to 63 do
         T1 = h + Σ1(e) + f_if(e,f,g) + Ki + Wi;
         T2 = Σ0(a) + f_maj(a,b,c);
         h = g;
         g = f;
         f = e;
         e = d + T1;
         d = c;
         c = b;
         b = a;
         a = T1 + T2;
      H0 = a + H0;
      H1 = b + H1;
      H2 = c + H2;
      H3 = d + H3;
      H4 = e + H4;
      H5 = e + H5;
      H6 = e + H6;
      H7 = e + H7;
   return concat(H0, H1, H2, H3, H4, H5, H6, H7);
```

其中：

$$W_i = \begin{cases} M_i, & \text{if } 0 \le i \le 15 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16}, & \text{if } 16 \le i \le 63 \end{cases}$$

$$\Sigma_0(x) = x \ggg 2 \oplus x \ggg 13 \oplus x \ggg 22$$
$$\Sigma_1(x) = x \ggg 6 \oplus x \ggg 11 \oplus x \ggg 25$$
$$\sigma_0(x) = x \ggg 7 \oplus x \ggg 18 \oplus x \gg 3$$
$$\sigma_1(x) = x \ggg 17 \oplus x \ggg 19 \oplus x \gg 20$$

$$f_{if}(b,c,d) = b \wedge c \oplus \neg b \wedge d$$
$$f_{maj}(b,c,d) = b \wedge c \oplus b \wedge d \oplus c \wedge d$$

下图为一轮 SHA-256 操作的示意图：

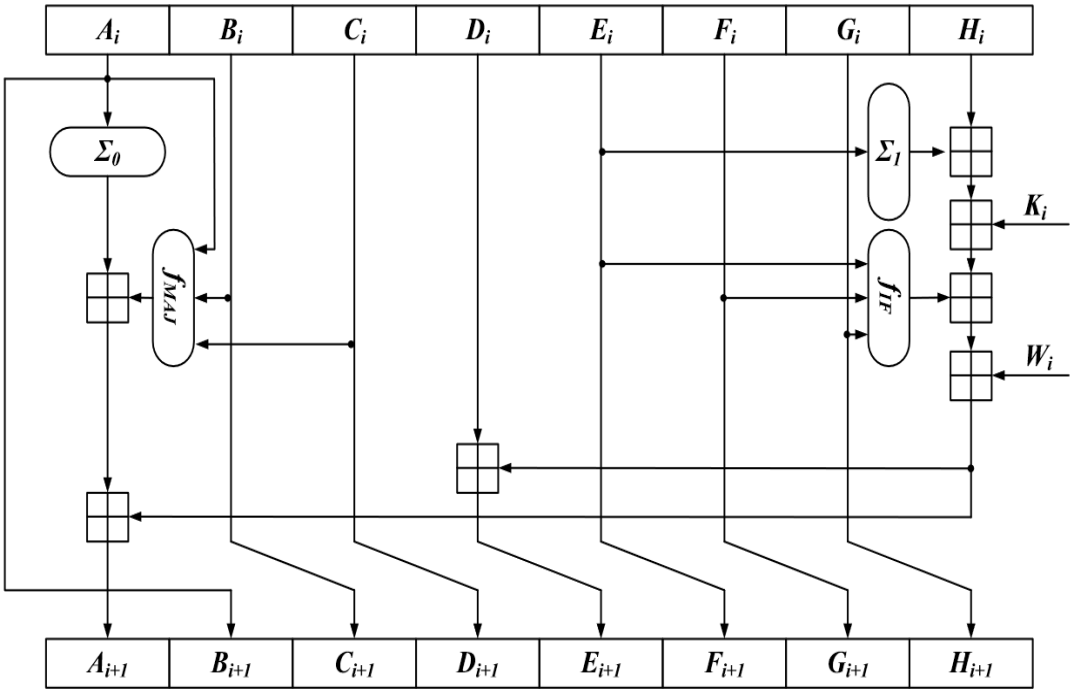


图 4-1 一轮 SHA-256 操作的示意图

另外，SHA-512 函数的过程与 SHA-256 函数基本一致，只需注意有如下几点不同即可，因此不再赘述：

- SHA-512 函数使用 1024 比特位长度的块
- 常量 K 和初始临时变量 H 不同，并且为 64 比特位长度
- 进行 80 轮操作

- 临时数组变量 W 为 80 个 64 比特位长度
- 用于计算的临时变量也为 64 比特位长度
- 对齐所追加的长度为 128 比特位
- 函数 Σ_0、Σ_1、σ_0、σ_1 所使用的比特位右移长度不同

4.3 ECDSA 椭圆曲线数字签名算法的实现

本论文所采用的椭圆曲线为美国国家标准与技术研究院（NIST）所推荐椭圆曲线之一：NIST Curve P-256，可以提供等同于一个 3072 位长度的 RSA 加密算法公钥的安全性。[30]

为了使用椭圆曲线数字签名算法，系统上的各个节点必须约定一个相同的椭圆曲线及其相关参数，公开所选定的椭圆曲线及其参数并不影响安全性，所以采用直接预定义在系统程序中的方式约定。为了提高开发效率，部分算法实现代码引用自 Python 开源类库 python-ecdsa。

密钥的生成

1) 选择一条平面曲线，使之符合如下等式：$y^2 \equiv x^3 + ax + b$，这里我们使用 NIST Curve P-256，表示为 C
2) 选择曲线 C 上的一点表示为 G 和一个大素数表示为 n
3) 随机选取[1, n-1]上的一个整数 d，d 作为私钥。
4) 计算得到点 Q，Q = d×G，其中×操作为椭圆曲线点乘，Q 作为公钥。

签名的生成

1) 计算 e = SHA-256(m)，其中 m 为需要签名的内容，SHA-256 为加密散列函数
2) L_n 代表 n 的位长，取 e 的最左 L_n 位为 z
3) 随机选取区间[1, n-1]上的一个整数 k
4) 计算点(x, y) = k×G，其中×操作为椭圆曲线点乘

5) 计算 r = x mod n，如果 r 等于 0，则重新选取 k，并重复后续步骤

6) 计算 $s = k^{-1} (z + rd) \bmod n$，如果 s 等于 0，则重新选取 k，并重复后续步骤

7) 最终得到点(r, s)作为签名

签名的验证

1) 检查 r, s 是否位于区间[1, n-1]上，如不是则签名无效

2) 计算 e' = SHA-256(m)，其中 m 为需要签名的内容，SHA-256 为加密散列函数

3) L_n 代表 n 的位长，取 e'的最左 L_n 位为 z'

4) 计算 $w = s^{-1} \bmod n$

5) 计算 u1 = z'w mod n，以及 u2 = rw mod n

6) 计算点(x', y') = u1×G + u2×Q，其中+操作为椭圆曲线点加，×操作为椭圆曲线点乘

7) 如果 r 等于 x' mod n，则签名验证为有效

4.4 Gossip 协议的点对点网络的实现

本系统需要同时保持大量的 TCP/IP 网络连接，如果使用通常的阻塞式 I/O 模型连接，那么将需要同时保持大量的线程，这将消耗大量的系统资源，因此本系统采用了 Tornado 网络编程框架的非阻塞式 I/O 模型的网络服务器，它可以使得系统程序在保持大量的 TCP/IP 连接的同时只消耗较少的系统资源。

系统程序既需要主动向其他节点发起连接，也需要被动地等待其他节点的连接，因此网络连接模块主要分为两个部分。

首先，系统程序启动时，在本机上一个特定端口开启监听，以便接收其他节点的连接，这个端口的选择可以是固定的，也可以是随机选择的，但是如果约定一个固定的端口号，那么可以使得处于初始状态的节点通过网络扫描发起连接，而无需通过其他初始化方式。这一部分采用 Tornado 中的 TCPServer 实现，由于连接的监

听也需要长时间保持在后台运行，因此使用一个独立的线程来承载。

其次，系统程序启动时也需要主动向外发起连接，特别是初始启动的时候，当前节点还不为任何节点所知，因此很难依靠监听特定端口来获得连接，这个时候主动发起连接显得尤为重要。对于主动发起的连接，使用的是 UNIX 套接字的方式，但是包装为 Tornado 的 IO 流，然后加入 Tornado 的连接池。

4.5 工作量证明计算的实现

工作量证明计算的本质在于搜寻一个小于目标值的散列摘要，要得到新的散列摘要必须要改变输入数据，但是区块的数据又必须保持不变，因此在区块数据中加入一个可以任意改变的整数域，称之为 Nonce，在进行工作量证明计算的时候只需不停地递增这个整数域，并对比得到的散列摘要值与目标值，最终得到小于目标值的散列摘要。

程序启动之时当前内存区域内必然是没有文章数据，此时如果立即开始工作量证明计算，则将有可能产生一个没有文章数据的空区块，当然，空区块是一种有效的区块形式，因此也有可能被其他节点所接受。但是这样一个空的区块显然浪费了一定的存储空间和计算资源，因为他没有为存储文章数据作出任何贡献。所以，应该尽量避免产生空区块，在程序启动之时，不应该立即启动工作量证明计算，而是应该首先等待一个固定的时间，以接收来自网络的数据。网络中确实没有文章数据也是有可能的，因此，在等待一定时间后，不管有无接收到文章数据，都开始进行工作量证明计算。

区块数据的结构 3.4 中所示，区块的实际内容 block 并不包含 Nonce 值，而是在 block 之外，计算时首先将 block 键值的子键按字母顺序排列并以紧凑格式输出，再将 Nonce 值作为一个整数拼接到末尾，得到需要在其之上做工作量证明计算的数据。由于 Python 的字符串类型对象一旦产生便不能修改，因此进行字符串拼接的时候，实际上会重新生成一个字符串对象，而工作量证明计算将尝试大量不同 Nonce 值，如果使用常用的字符串拼接方法，势必会导致产生大量新的字符串对

象，相应地会大量消耗系统的内存和 CPU 资源。为了提高程序效率，我们将拼接的整数固定为 32 位大端序整型，这样每次修改 Nonce 值的时候，可以直接更新这块 32 位的内存区域，而不用进行字符串拼接，极大地提高了程序效率，如图 4-2 所示。

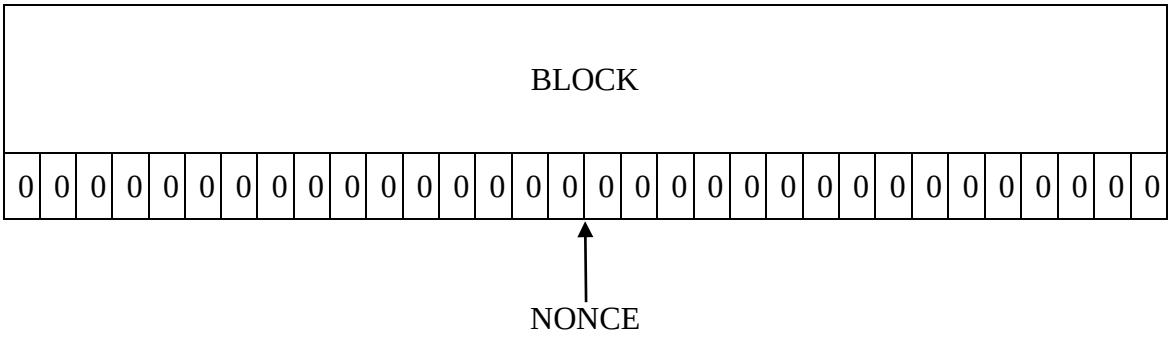


图 4-2 NONCE 值拼接

工作量证明计算需要花费较多的时间，为了不影响程序主体运行，其需要在后台持续运行，因此每一次工作量证明计算都启用一个新的线程，该线程需要首先等待上次工作量证明计算的线程结束，才能开始下一次计算。

4.6 本地数据结构的实现

对于本地数据的存储采用 SQLite 数据库管理系统来实现，Python 标准库中已经带有可直接操作 SQLite 数据库的相关接口，因此无需使用第三方库。

本地数据主要分为两张表，分别对应区块数据和文章数据，数据结构已在表 3-1 和表 3-2 中做了详细描述，具体到 SQLite 数据库中，其实际结构如下文的表创建 SQL 语句所示。

```
create table if not exists blocks(
   hash text primary key,
   height integer,
   parent text,
   timestamp integer,
   data blob);

create table if not exists posts(
   hash text primary key,
   timestamp integer,
   username text,
   userkey text,
   category text,
   parent text,
   data blob);
```

4.7 用户交互界面的实现

用户交互界面，本系统的实现采用了两个分别由 Twitter 和 Facebook 开发的 Web 前端库：Bootstrap 和 React。Bootstrap 是一组用于网站和网络应用程序的开源工具库，包括 HTML、CSS 及 JavaScript 的框架，提供字体、窗体、按钮、导航及其他各种组件，并提供了 Javascript 扩展，旨在使得动态网页和 Web 应用的开发更加容易。React 是一个用来构建动态 Web 前端应用的 Javascript 库。这两者结合起来，Bootstrap 提供样式，React 负责界面渲染，可以极大地提高 Web 前端界面的开发效率。

Web 前端界面与系统程序之间的数据交互通过 REST 形式的 HTTP 请求来实现，即系统程序实现一个简单的 HTTP 服务器，浏览器向这个 HTTP 服务器发起请求。REST 的全称为 Representational State Transfer，表征性状态传输，是 Roy Fielding 博士于 2000 年在他的博士论文中提出来的一种软件架构风格[36][38]。目前在三种主流的 Web 服务实现方案中，因为 REST 模式与复杂的 SOAP[39][40]和 XML-RPC[41][42]相比更加简洁，越来越多的 web 服务开始采用 REST 风格设计和实现。

将数据接口跟实际页面区分开来，除了数据与视图解耦后开发上更为简便以外，也使得第三方有可能参与进来开发其他形式的 Web 前端界面，甚至各种不基于 Web 的客户端，为用户提供更多选择。

本系统的 HTTP 接口主要分为三类：

1) 静态文件接口：用于获取各种 HTML 页面需要的资源：如图片文件、CSS 文件、Javascript 文件等等
2) 获取文章接口：用于获取文章数据，可单独指定散列值请求，也可指定时间间隔获取
3) 发送文章接口：向系统程序提供文章的基本内容，再由系统程序发布

HTTP 服务器同样使用 Tornado 来实现，并由单独的线程来承载。

4.8 系统测试

因用户交互界面部分对于系统可用性的影响微乎其微，因此只针对核心系统程序进行测试，以验证系统可用性。

测试方法为在本机模拟一定数量的节点运行，各个模拟节点使用不同的端口以作区分。然后在各节点中生成带有特定节点标记的文章消息，检查各个节点上生成的带标记文章消息是否被包含进入区块链。因为条件有限，开发所用的计算机上最多只能支撑约二十个模拟节点的运行，因此测试结果是在一个包含二十个节点的模拟系统的基础上得到的。

测试分别模拟三种不同类型的网络结构，分别为星形结构、环形结构和随机结构。星形结构中一个节点包含其他所有节点的信息，其他节点只包含这一个节点的信息，网络结构呈现单点辐射状。环形结构中每个节点包含额外两个节点的信息，网络结构呈现圆环状。这两种网络结构分别代表网络分布结构的两个极端，分别为高度集中和高度分散。

测试时将系统程序的目标值设定在区块生成速度为 10 秒到 60 秒之间，因为区块生成实际上是一个伪随机的过程，所以在节点数量较少的情况下，无法保证一个基本稳定的区块生成时间。在三种网络结构的情况下，系统内各个节点在一定时间段内各自发出一个文章消息，消息中带有节点的特定标记，最后检查是否所有消息都能被包含进区块链中。测试五次，检查最终包含所有数据的区块个数，测试结果如表 4-3 所示。

表 4-1 测试结果

网络结构	生成多少个区块后包含全部数据
星形结构	平均约为 7 个
环形结构	平均约为 12 个
随机结构	失败率较高，不计算平均值

其中，星形结构和环形结构下，所有测试均成功，但是在随机结构的情况下，因为节点数量较少，所以有较大概率形成相互隔离的网络，这种情况下必然导致测试失败，但是这种情况在现实中运行的节点数量较多的网络中出现的概率非常低。

4.9 本章小结

本章描述了系统实现中的重点部分，首先，描述了系统开发中用到的关键技术和工具，即系统主要采用 Python 编程语言编写，结合了 Tornado 网络编程框架和 SQLite 数据库管理系统。其次，根据上一节对系统中应用的关键算法的分析，描述了其实现，包括 SHA-256 散列加密算法、EADSA 数字签名算法和类 Gossip 的分布式网络通讯协议。第三，描述了工作量证明计算以及用户交互界面的实现。最后，对系统的可用性进行了测试验证。

5 总结与展望

本论文主要提出了一个不需要中心服务器的社区系统的设计，和一个简单的系统实现，但是由于时间和精力有限，无论是系统设计还是系统实现均存在很多不足，这些都是后续研究工作的优化重点。

首先，本系统采用了工作量证明计算（Proof-of-Work）的方式使得一个由非信任节点组成的网络，能够形成共识，即各个节点上的区块链数据基本一致。但是工作量证明需要做大量无意义的计算，消耗大量的计算机系统资源以及电力资源，在节点数量较大的情况下，累加起来的电力资源将是非常可观的。如果达成共识的机制可以摆脱工作量证明计算，那么将会是一个非常有意义的改进。

其次，在系统中存在恶意节点制造大量垃圾内容的情况下，本系统目前还没有提出一个有效的应对机制。大量的垃圾内容将会使得区块链容量在短期内快速增长，消耗节点的存储空间，并使得无论是单独节点还是整个系统的运行效率降低，尽管无法对系统造成致命威胁，但是垃圾内容会伤害用户体验，使得参与到系统中的节点数量降低，那么相应的，整个系统的安全性也会降低。对于垃圾内容，理论上是没有办法从根本上解决，因为评判是否为垃圾内容，并不存在定性、定量的规则，计算机甚至是人类大脑都不能判断某段文字是否为垃圾内容。但是仍然可以从限制节点发送新内容的频率，以及提出某种积分机制来提高发布新内容的门槛，两个方面来尽可能的减少垃圾内容。

再次，本系统为了保证安全，只允许从本机对系统程序的 HTTP 服务器发起访问，如果系统程序运行在本机以外的其他主机上，那么 Web 界面将无法访问到。后续可以给 HTTP 服务器加上某种安全验证机制，如预定义的密码，保证远程访问的安全性，然后开放其他主机访问的权限，这将使用户体验得到一定提升。

致谢

本论文是在导师陆永忠副教授认真而悉心的指导下完成的，陆老师钻研的进取精神和严谨的治学精神给了我十分深刻的影响，研究生学习期间，陆老师严肃的科学态度和精益求精的工作作风为我树立了科研工作的学习楷模，也为我指引着学习和研究的方向。从论文开题到答辩期间，每一步的进展都倾注了陆老师的大量心血。陆老师不仅在学习上指导我，在生活上也十分关心我。在此，对陆老师致以深深的谢意和崇高的敬意！

感谢软件学院陪我一起度过两年多学习时光的同学们，他们无论在学习上还是在生活上都给予了我很多关心和帮助。感谢实验室的同学们，在论文写作期间我们共同讨论论文中的所涉及的问题，并认真地给予了我很多意见和建议。

感谢我的家人，在研究生学习的两年多时间里对我一直以来的支持和鼓励，让我在优越舒适的环境下专注地完成了研究生学业。

最后，感谢评阅论文并给予宝贵评语的老师，感谢答辩中给予批评和指正的老师！

参考文献

[1] P. Rogaway, T. Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. Computer Science, Vol.3017, Springer-Verlag, July 2009: 371-388.

[2] Wouter Penard, Tim van Werkhoven. On the Secure Hash Algorithm family. Cryptography in Context, Utrecht University: 1-17.

[3] R. Rivest. The MD5 message-digest algorithm. IETF Network Working Group, RFC 1321, April 1992.

[4] X. Wang, H. Yu. How to break MD5 and other hash functions. Advances in Cryptology–EUROCRYPT 2005. Springer Berlin Heidelberg, 2005: 19-35.

[5] X. Wang, H. Yu, Y. L. Yin. Efficient collision search attacks on SHA-0. Advances in Cryptology–CRYPTO 2005. Springer Berlin Heidelberg, 2005: 1-16.

[6] X. Wang, H. Yu, Y. L. Yin. Finding Collisions in the Full SHA-1. Advances in Cryptology–CRYPTO 2005. Springer Berlin Heidelberg, 2005: 17-36.

[7] NIST. FIPS Publication 180-2: Secure Hash Standard. Technical Report. National Institute of Standards and Technology (NIST), August 2002.

[8] NIST. FIPS Publication 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, and Revision of the Applicability Clause of FIPS 180-4, Secure Hash Standard. National Institute of Standards and Technology (NIST), August 2015.

[9] 全国人民代表大会常务委员会. 中华人民共和国电子签名法. 2004 年 8 月 28 日

[10] Whitfield Diffie, Martin E. Hellman. New directions in cryptography.

Information Theory, IEEE Transactions on, 1976, 22(6): 644-654.

[11] Rivest, Ronald, Shamir, Adi, Adleman, Leonard. Cryptographic communications system and method. U.S. Patent 4,405,829. Dec 14, 1977.

[12] S. Goldwasser, S. Micali, R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing, 1988, 17(2): 281-308.

[13] Lamport, Leslie. Constructing digital signatures from a one-way function. Vol. 238. Palo Alto: Technical Report CSL-98, SRI International, 1979.

[14] M. O. Rabin. Digitalized signatures and public-key functions as interactable as factorization. Technical Report MIT/LCS/TR-212, Laboratory for Computer Science, MIT, 1979.

[15] David W. Kravitz. Digital signature algorithm. U.S. Patent No. 5,231,668. Jul 27, 1993.

[16] Don Johnson, Alfred Menezes, Scott Vanstone. The elliptic curve digital signature algorithm (ECDSA). International Journal of Information Security 1.1 (2001): 36-63.

[17] NIST. FIPS Publication 186-3: Digital Signature Standard. Technical Report, National Institute of Standards and Technology (NIST), June 2009.

[18] Neal Koblitz. Elliptic curve cryptosystems. Mathematics of computation 48.177 (1987): 203-209.

[19] E. Barker, W. Barker, W. Burr, W. Polk, M. Smid. Recommendation for Key Management - Part 1: General (Revision 3). NIST Special Publication, 800(57), 2012.

[20] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters. Certicom Corp., 2000.

[21] Leslie Lamport. Paxos made simple. ACM Sigact News 32.4 (2001): 18-25.

[22] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, D. Terry. Epidemic Algorithms for Replicated Database Maintenance. Principles of Distributed Computing, ACM, Aug 1987, 1-12.

[23] L. Alvisi, J. Doumen, R. Guerraoui. How robust are gossip-based communication protocols?. ACM SIGOPS Operating Systems Review, 2007, 41(5): 14-18.

[24] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, Y. Minsky. Bimodal multicast. ACM Transactions on Computer Systems, May 1999, 17(2):41–88.

[25] Márk Jelasity, Wojtek Kowalczyk, Maarten Van Steen. Newscast computing. Technical Report IR-CS-006, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, 2003.

[26] P. T. Eugster, R. Guerraoui, S. B. Handurukande. Lightweight probabilistic broadcast. ACM Transactions on Computer Systems (TOCS), 2003, 21(4): 341-374.

[27] M. Jelasity, R. Guerraoui, A. M. Kermarrec, M. van Steen. The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations. Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, Springer-Verlag New York, Inc.: 79-98.

[28] J. Dinger, O. P. Waldhorst. Decentralized bootstrapping of P2P systems: a practical view. NETWORKING 2009. Springer Berlin Heidelberg, 2009: 703-715.

[29] D. Crockford. The application/json media type for javascript object notation (json). 2006.

[30] NIST. Recommended Elliptic Curves for Federal Government Use. July 1999.

[31] Adam Back. Hashcash - a denial of service counter-measure. 2002.

[32] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[33] Debin Liu, L. Jean Camp. Proof of Work can Work. WEIS. 2006.

[34] R. C. Merkle. A digital signature based on a conventional encryption function. Advances in Cryptology—CRYPTO'87. Springer Berlin Heidelberg, 1988: 369-378.

[35] R. C. Merkle. Protocols for public key cryptosystems. Symposium on Security and Privacy, IEEE Computer Society, April 1980, 122-133.

[36] F. Qian, J. Huang, J. Erman, et al. How to reduce smartphone traffic volume by 30%?. Passive and Active Measurement. Springer Berlin Heidelberg, 2013: 42-52.

[37] Roy T. Fielding, Richard N. Taylor. Principled design of the modern Web architecture. ACM Transactions on Internet Technology (TOIT), 2002, 2(2): 115-150.

[38] R. Battle, R. Benson. Bridging the semantic Web and Web 2.0 with representational state transfer (REST). Web Semantics: Science, Services and Agents on the World Wide Web, 2008, 6(1): 61-69.

[39] D. Box, D. Ehnebuske, G. Kakivaya, et al. Simple object access protocol (SOAP) 1.1. 2000.

[40] F. Curbera, M. Duftler, R. Khalaf, et al. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. IEEE Internet computing, 2002 (2): 86-93.

[41] P. Merrick, S. Allen, J. Lapp. XML remote procedure call (XML-RPC): U.S. Patent 7,028,312. 2006-4-11.

[42] M. Allman. An evaluation of XML-RPC. ACM sigmetrics performance evaluation review, 2003, 30(4): 2-11.