

南开大学博士毕业（学位）论文

（匿名评阅论文封面）

论文题目（含英文题目）：

基于区块链的云存储审计关键技术研究

Key Technology Research on the Blockchain-Based Data Auditing
for Cloud Storage

一级学科：计算机科学与技术

二级学科：

论文编号：

摘要

近年来，云存储服务在全球范围内广泛普及。在我国随着医疗云、政务云等行业的快速发展，云存储服务已经跟每一位国民生活的方方面面息息相关。与此同时，多云存储服务因其高安全性、高灵活性、去厂商锁定等独特优势逐渐得到越来越多企业和机构青睐。云上存储的很多数据涉及个人隐私、商业机密或是组织机构的重要信息。这些数据一旦丢失或破坏，将造成巨大的经济损失和不良的社会影响。因此，提高云存储服务的安全性、可靠性和隐私性成为十分重要的研究问题。

针对大规模云存储系统，远程数据审计是十分有效的保护数据完整性和提升存储可靠性的方法。审计过程无需下载原文件便可快速验证指定数据的完整性，并且可以反复执行。这使得我们能够尽早发现已损坏的数据并执行数据修复措施。多年以来，国内外学者在这一领域进行多方面研究，提出了很多方案，但在审计的可信性和实用性方面一些难点始终尚未解决，具体表现在如下三个方面。（1）审计可信性。目前采用的基于第三方的中心化审计模型导致审计规则不公开、审计过程不透明、审计结果无法验证，因而无法从根本上保证审计的安全可信。（2）审计可行性。目前提出的基于区块链的分布式审计方案虽然可以部分实现审计可信性，但区块链本身存储可扩展性差的问题成为阻碍审计区块链落地的一大难点。（3）审计机制评价。可靠性评价方法可以帮助我们量化分析审计机制对提升存储系统可靠性的贡献以及不同审计策略的实际效果，从而为审计机制设计者提供理论指导。然而目前很少有学者关注对于云存储审计系统的可靠性评价。

本文面向大规模云存储系统，基于区块链技术研究安全可信高效的远程数据审计方案，主要工作包括：

第一，设计规则公开、过程透明、结果可验证的分布式审计方案。为彻底解决传统中心化审计的单点故障问题，基于区块链技术设计去中心化审计模型，构建分布式审计网络。依靠算法、协议而不是第三方来保证审计的可靠可信。针对传统审计过程不公开、结果不可验证的问题，基于智能合约技术实现审计算法和协议，使得审计规则以及审计运行过程全部记录上链，向所有用户公开。用

摘要

户既是审计的执行者也是监管者，由此实现透明可监管的审计。为避免审计记录上链引发的隐私泄露风险，基于零知识证明技术设计具有隐私保护性质的审计协议，从而在去中心性、公开性和隐私保护之间取得平衡。

第二，设计安全、高效、高可扩展的审计区块链系统。审计记录上链是保证可信审计的关键，而如何维护链上数据成为新的难点。我们采用审计网络与区块链网络合一的思想，由云存储用户（即审计者）作为区块链节点维护链上数据，避免使用外部区块链网络引发的安全隐患，保证审计记录可靠存储。针对不同云存储用户拥有资源参差不齐的现状，提出具有分层结构的审计联盟链系统模型。在网络中区分出三类区块链节点，为每类节点分配不同等级的任务和权利，以此提升系统可扩展性。为解决节点存储开销过大的问题，设计动态弹性存储分片方案以实现区块链存储模式优化。每个节点只需存储所有区块头以及一部分区块，并通过节点间高效协作来访问本地未存储的区块。根据实时访问热度动态调整区块副本个数、存储位置和存储模式，从而在节点存储开销、数据访问性能和数据可靠性之间取得平衡。

第三，设计适用于大规模多云审计系统的可靠性评价方法。考虑到多云存储系统结构的复杂性以及故障发生原因的多样性，从系统整体可靠性入手，着重分析单个云服务商的可靠性机制失效导致多云存储系统发生数据丢失的情况。为客观细致评价远程数据审计机制的运行效果，建立马尔可夫模型模拟系统故障发生和修复的动态过程，对比分析不同参数下无审计机制的多云系统与运行审计机制的多云系统可靠性，并对比分析不同审计策略对系统可靠性的提升效果。设计蒙特卡洛仿真程序以验证和辅助模型分析结果。结合网络流量开销、网络规模等多角度分析，给出恰当的审计方案设计建议，从而在系统可靠性、数据安全性和审计机制开销之间取得平衡。

总之，本文提出了实现审计可信性的分布式审计框架以及实现隐私保护的分布式审计协议，提出了审计联盟链系统模型以及区块链存储优化方案，提出了适用于多云存储系统的审计机制评价方法。上述研究成果从不同角度加强了远程数据审计机制的安全性和可行性，能够有效促进区块链审计技术在大规模云存储系统的实际应用。

关键词：远程数据审计；区块链；云存储；数据完整性；可靠性评价

Abstract

We have entered an era where cloud storage is widely used around the world. In China, cloud platforms for government services and health care grew rapidly in recent years, making cloud storage service closely relevant to our everyday life. Meanwhile, multi-cloud storage is gaining popularity among corporations and organizations for its high-level security, flexibility and avoiding vendor lock-in. With more and more data stored on the cloud, most of which are sensitive, data protection becomes very important, for that data loss or data broken may result in huge financial loss and negative social impacts.

Remote data auditing is a trending way for large-scale cloud storage systems to improve data integrity and reliability. The integrity checking is executed without downloading the original data and can be performed repeatedly, which allows us to detect damaged data as soon as possible and to perform the repair process on time. Many researchers have concentrated on this research field and presented numerous proposals. However, there are still some problems waiting to be solved, as follows. (a) Auditing dependability. Currently the audit is performed by a third-party auditor, which leads to single-point failures, opaque audit processes and undetected mistakes. (b) Auditing feasibility. The blockchain-based decentralized auditing framework is promising to solve the above problem, but the poor scalability of blockchain may hinder the feasibility. (c) Auditing assessment. There has been few proposals on assessing the reliability of cloud storage systems running remote-data-auditing schemes, although the reliability assessment plays an important role on analyzing the effects of different auditing strategies.

This thesis focuses on the blockchain-based remote data auditing for large-scale cloud storage systems. The main work is as follows.

Firstly, we propose a decentralized auditing scheme in which the auditing rules and auditing processes are public to cloud users and the auditing results are verifiable. Decentralized auditing network and decentralized auditing protocols are proposed to solve single-point failures. The audit is performed by multiple auditors and the audit result

Abstract

is reached in a collaborative and transparent way. Auditing-related procedures are implemented using a smart contract, and every system participant is allowed to call the functions. Auditing records are published on a blockchain, which are evidence for arbitration. To avoid the leakage of personal privacy, a zero-knowledge auditing protocol is proposed, which balances the decentralization, transparency, and privacy protection.

Secondly, we propose a safe, efficient, and scalable auditing-blockchain system. Publishing auditing records on the blockchain is a key step to achieve dependable auditing, but the fast-growing data size and the append-only nature of blockchain pose a severe challenge to system scalability. We design a permissioned blockchain used to store auditing records, where cloud users (also as auditors and blockchain nodes) are entitled to maintain the blockchain. To improve system scalability, cloud users are divided into three categories according to the amount of resources they have. Those with sufficient resources are responsible for storing blockchain data as full nodes or sharding nodes, while those with limited resources are light nodes. Sharding nodes are formed into groups and each group maintains a blockchain copy collaboratively. A dynamic block allocation mechanism is applied to adjust the placement and replication ratio of blocks adaptively, thus the block request latency is minimized.

Thirdly, we propose a reliability assessment method for large-scale multi-cloud auditing systems. Considering that multi-cloud systems are complex and the causes of failures are multiple, we analyze from the system level and neglect inner-cloud failures which won't be sensed by the multi-cloud system. We focus on the situation when the reliability-protection mechanisms on the single cloud is invalid and the failure may cause data loss in the multi-cloud system. A continuous-time Markov chain is used to depict the process of failure and recovery. The effect of different auditing strategies is analyzed using the Markov model. A Monte-Carlo simulation is applied to confirm the analysis results derived from the Markov model. Combing the reliability analysis and network-cost analysis, we give suggestions about how to design an appropriate auditing scheme for multi-cloud systems.

For summary, this thesis proposes: a) a decentralized auditing framework and two decentralized auditing protocols which achieve dependability and security, b) the blockchain-based auditing system which applies the dynamic storage sharding for the

Abstract

blockchain, c) reliability assessment methods for multi-cloud auditing systems and advises on suitable auditing strategies.

Key Words: remote data auditing; blockchain; cloud storage; data integrity; reliability assessment

Abstract

目录

第一章 绪论	1
第一节 研究背景与挑战	1
第二节 研究内容和创新点	6
第三节 文章组织结构	10
第二章 背景知识和相关工作	11
第一节 背景知识	11
2.1.1 云存储系统数据散布技术	11
2.1.2 点对点存储系统数据散布技术	14
2.1.3 零知识证明技术	16
第二节 远程数据审计技术	18
2.2.1 可证明数据持有	18
2.2.2 可检索性证明	20
2.2.3 隐私保护公开审计	21
第三节 基于区块链的数据审计技术	23
2.3.1 区块链技术概述	23
2.3.2 针对云存储的区块链数据审计	24
2.3.3 针对 P2P 存储的区块链数据审计	26
第四节 区块链存储优化技术	28
第五节 云存储可靠性评价方法	30
2.5.1 针对磁盘故障的可靠性分析	31
2.5.2 针对外部攻击的生存性分析	33
第六节 本章小结	34
第三章 隐私友好的分布式公开审计方案	35
第一节 引言	35
第二节 基于区块链的公开审计模型	38
3.2.1 系统模型	39
3.2.2 安全模型	41
3.2.3 设计目标	42

目录

第三节 审计者选举	43
3.3.1 选举	44
3.3.2 身份验证	45
3.3.3 分析	45
第四节 PoR [*] 协议设计	46
3.4.1 准备阶段	47
3.4.2 挑战阶段	47
3.4.3 回复阶段	48
3.4.4 验证阶段	48
3.4.5 分析	49
第五节 zk-PoR [*] 协议设计	50
3.5.1 背景知识简介	50
3.5.2 zk-PoR [*] 工作流程	52
3.5.3 分析	54
第六节 激励机制	54
3.6.1 针对用户的激励机制	54
3.6.2 针对云服务商的激励机制	57
第七节 实验测试	58
3.7.1 实验设置	58
3.7.2 财富值与获胜概率	59
3.7.3 审计的均匀性	59
3.7.4 NIST 随机性测试	61
3.7.5 验证阶段时延	62
第八节 本章小结	65
第四章 实现存储优化的审计联盟链系统	67
第一节 引言	67
第二节 审计联盟链系统设计	70
4.2.1 系统模型	70
4.2.2 系统工作流程	71
4.2.3 存储分片模块	73
第三节 存储分片模块设计	75
4.3.1 初始散布	75
4.3.2 动态复制	76

目录

4.3.3 存储优化	78
第四节 服务节点数据请求	80
4.4.1 无节点故障时的数据请求	80
4.4.2 节点故障及数据恢复	81
4.4.3 节点加入及退出	82
第五节 存储分片模块实验测试	83
4.5.1 实验设置	83
4.5.2 测试结果	85
4.5.3 CUB 性能测试结果	89
第六节 审计区块链系统实验测试	91
4.6.1 审计区块链系统构建	91
4.6.2 测试结果	92
第七节 本章小结	96
第五章 多云审计系统存储可靠性评价	97
第一节 引言	97
第二节 背景知识	99
5.2.1 多云散布系统	99
5.2.2 云存储可靠性评价	100
第三节 多云审计系统介绍	101
5.3.1 多云数据散布	101
5.3.2 数据审计和修复	102
第四节 数学模型	103
5.4.1 随机审计策略	104
5.4.2 批量审计策略	107
5.4.3 更新审计策略	111
5.4.4 无审计的多云存储系统可靠性模型	112
5.4.5 模型求解	112
第五节 实验验证	114
5.5.1 蒙特卡洛实验设置	114
5.5.2 实验结果	116
5.5.3 实验结论	123
第六节 本章小结	125
第六章 总结与展望	127

目录

第一节	本文工作总结	127
第二节	未来工作展望	129
参考文献		131
个人简历		155

第一章 绪论

第一节 研究背景与挑战

随着大数据时代的到来以及云计算技术的日益成熟，云服务在全球范围内广泛普及。云服务能够快速部署，使得企业和机构可以高效地将业务迁移上云，节省大量成本。云服务有专门的运维团队负责后台维护，使得用户无需为系统运维、资源配置等花费过多精力。用户还可以根据自身实时需求对资源进行动态弹性配置，无需为闲置资源付费。由于以上优势，云服务在软件开发、网站建立以及服务平台搭建等方面表现突出，受到越来越多公司、组织和机构青睐。据统计，2014 至 2018 年期间我国医疗云行业的复合增长率高达 34.7%^[1]。同时，政务云服务已在我国 90% 以上的地市乃至县域得到普及，且预计 2023 年整体市场规模将达到 1114.4 亿元^[2]。可以说，云服务已然跟每一位国民生活的方方面面息息相关。然而一个不容忽视的现状是，使用云服务也带来了数据安全方面的隐患。过去几年，全球知名云服务商如 Google Docs^[3]、Amazon S3^[4]、Microsoft Azure^[5] 等都出现过突发性的服务故障，给用户造成恐慌。云上存储的数据涉及到个人及组织的重要信息，一旦丢失或损坏将造成巨大的经济损失，造成不良社会影响。因此，对云存储数据的安全性保护显得尤为重要。

远程数据审计是专为大规模云存储系统提出的数据完整性保护技术，主要包括可证明数据持有（PDP, Provable Data Possession）和可检索性证明（PoR, Proof of Retrievability）两类方案^[6,7]。其主要设计思想如下。在数据上传阶段，用户针对每个待上传文件块生成验证子并要求云服务商保存数据块及其验证子；在审计开始后，审计者采用抽查（spot-checking）的方式对云上存储的数据发起挑战；云服务商根据挑战内容，使用指定文件块及其验证子计算证明并发送给审计者；审计者执行验证并得出本轮审计结果。大部分审计方案具有如下特点：（1）无需下载原始数据便可执行验证；（2）验证过程计算复杂度较低且耗费的带宽较低；（3）验证可反复多次进行。合理的审计机制配合有效的数据修复机制，可以较好地保护数据完整性和存储可靠性。

根据审计执行主体的不同，审计方案可划分为私有审计和公开审计两类。

其中，私有审计由数据所有者（即云存储用户）本地执行。如图1.1(a)所示，私有审计过程一般包含三个步骤：(1) 用户生成“挑战”发送给云服务商，该挑战会随机选中一些文件块并要求云服务商证明其完整性；(2) 云服务商检索被选中的块，计算“回复”并发送给用户；(3) 用户验证“回复”的内容是否正确，并得出审计结果。已经提出的私有审计方案包括2007年提出的PoR^[7]、2008年提出的Scalable PDP、2013年提出的RDPC^[8]、2017年提出的DPoR^[9]等。不过，私有审计方案普遍存在一些弊端。首先，用户本地得出的审计结果无法达到不可抵赖性。当诚实用户通过审计发现云上存储的数据出错时，云服务商有可能为了避免赔偿而否认审计结果，这将损害用户利益。反之，当不诚实的用户谎称从云上下载到的数据有问题时，云服务商无法证明自身，这将损害云服务商利益。此外，当用户上传到云端的数据量较大，或是对于数据完整性要求较高的情况下，必须频繁执行私有审计以确保数据安全，而这将耗费用户大量的时间和资源。因此，私有审计更适合那些在云端存储数据量较小的用户或是部署了本地服务器的用户。

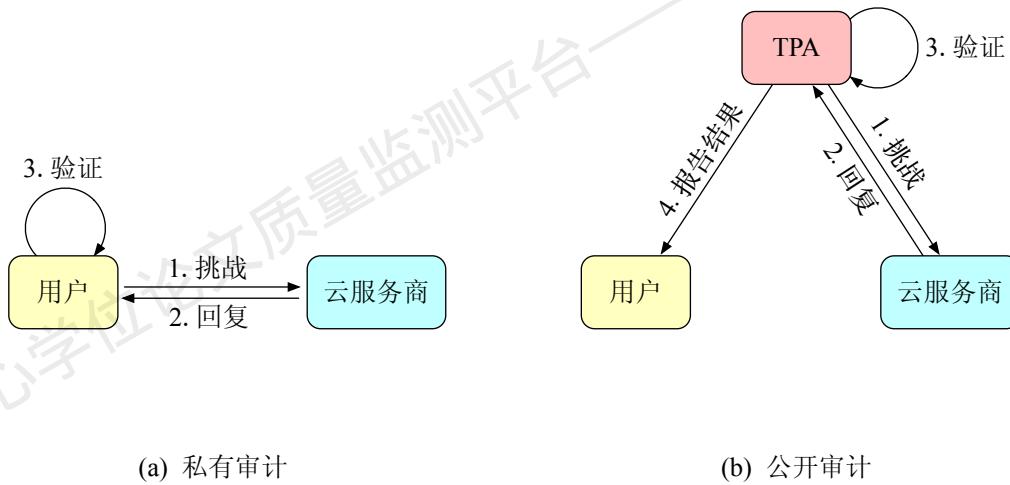


图 1.1 远程数据审计模型

由于私有审计存在的种种局限，公开审计成为目前更为主流且研究更多的审计范式。如图1.1(b)所示，公开审计允许用户将审计任务委托给一个第三方审计者（TPA, Third-Party Auditor）。公开审计过程一般包含四个步骤。(1) TPA生成“挑战”发送给云服务商，该挑战会选中一些文件块要求云服务商证明其完整性；(2) 云服务商检索被选中的块，计算“回复”并发送给TPA；(3) TPA验证“回复”的内容是否正确；(4) TPA将审计结果报告给数据所有者。已经提

出的公开审计方案包括 2007 年提出的 PDP^[6]、2009 年提出的 DPDP^[10]、2011 年提出的 FD-PoR^[11]、2013 年提出的 CPoR^[12] 等。总的来看，公开审计通过审计任务外包的方法解除了用户负担，并试图通过引入第三方来保证审计的不可抵赖性。然而，公开审计方案的安全性依赖于“可信第三方”假设。这种中心化的审计模型存在如下三方面问题。

1. 审计规则不公开。审计规则由 TPA 单方面制定，而用户对这些规则并不知情，这可能导致用户利益受损。举个例子，假设 TPA 使用的随机挑战生成算法无法达到很好的随机性，造成一部分用户的数据很少被审计到。这可能导致一些已经发生的数据损坏无法被及时检测，错过数据修复的最佳时机，使得相关用户的权益受损。
2. 审计过程不透明。根据审计规则所实现的审计算法和程序在 TPA 的服务器中运行，而用户对运行状态以及中间结果并不了解。一旦 TPA 的服务器发生意外故障或是遭遇恶意攻击，审计服务便会被迫中止或者发生错误。更有甚者，由于审计过程不公开，TPA 完全有能力捏造审计结果而不被发现。
3. 审计结果不可验证。用户无法验证审计结果是否可靠，只能完全信任 TPA。当云服务商与用户因数据审计结果发生纠纷时，也只能交由 TPA 进行裁决，无法通过其它手段维护自身权益。

区块链技术有望解决上述中心化审计模型存在的问题。区块链是一种分布式账本技术，最初被应用于点对点加密电子货币系统^[13]。区块链账本由区块前后链接而成，其中每个区块包含一段时间内所产生的交易。这一账本会被复制给系统内每一个节点进行保存。所有节点遵守共识机制以确保各个区块链副本之间保持一致性。区块链具有（拜占庭）容错属性，在节点之间互不信任的前提下，即使一小部分节点发生故障或故意作恶，全网仍旧能够成功对新区块达成共识。由于其独特的去中心化、不可篡改、可追溯的特点，区块链十分适合于以下场景：1) 保存数据量较小的重要数据，以保证数据不可篡改；2) 在互不信任的多方之间分享信息、相互合作。近几年，许多学者提出了基于区块链的远程数据审计方案，以弥补传统私有审计方案和公开审计方案各自内生的不足^[14-19]。例如，文献 [14] 提出一种适用于物联网的私有审计方案。用户将数据块的验证子存储在区块链上作为完整性验证凭证。数据审计不仅可以由数据所有者执行，还可以由获得数据所有者授权的数据访问者执行。再如，CPVPA^[18]

提出一种能够对 TPA 实行监督的公开审计方案。在一次审计过程中，TPA 根据链上最新几个块的哈希值生成挑战信息，并将审计结果发布在区块链上。用户通过查看链上信息来监督 TPA 行为，不仅可以发现 TPA 的恶意行为，而且可以发现不及时执行验证的拖延行为。

根据所采用的系统模型的不同，已提出的基于区块链的审计方案可分为两类。第一类方案采用审计网络与区块链网络分离的架构。云存储用户和云服务提供商作为区块链轻节点只发布数据和访问数据，而区块链系统的维护以及区块链数据的存储依赖于外部节点。这一架构省去了部署区块链全节点以及维护区块链网络的开销，但是将审计记录存储在外部节点的做法削弱了系统安全性。另一类方案将审计网络与区块链网络合一，使得用户既能够参与审计又能够参与区块链维护。我们将这类方案统称为“审计区块链”方案。全部审计记录由审计网络中的节点也就是云存储用户来维护，无需再依赖外部网络，这使得审计结果更加可靠可信。目前已经有一些针对审计区块链的研究工作。例如，在 Permacoin^[20] 和 Retricoin^[21] 中提出了这样一种审计区块链，所有想要参与挖矿的区块链节点必须提供存储空间来存储系统中的一部分数据。每个挖矿节点需要负责三项工作，即数据存储、数据审计，以及区块链维护。一次共识过程如下。(1) 挖矿节点收集新产生的交易并打包新区块；(2) 挖矿节点随机选中本地存储的若干数据块并对其生成完整性证明，这一过程即改进版的“挖矿”；(3) 挖矿节点将生成的证明写入新区块作为挖矿凭证，并将新区块广播给全网；(4) 其余节点验证新区块，若验证成功则新区块被接收、挖矿节点获得挖矿奖励。再如，Filecoin^[22] 是一个基于区块链的点对点存储系统，其存储节点必须生成时空证明（proof-of-spacetime）和复制证明（proof-of-replication）来证明自己按照要求存储了数据^[23]。

借助区块链技术的公开性、去中心性和不可篡改性，我们有望解决中心化审计存在的弊端。不过，这一研究领域仍有待进一步探索。目前已提出的基于区块链的审计方案仍旧存在如下几方面问题。

1. 已提出的审计区块链方案^[20-22] 大多针对点对点（P2P, peer-to-peer）存储系统，并不适用于云存储系统，而后者才是目前普及率更高、受众更多的存储服务。
2. 少数针对云存储系统设计的方案并没有采用分布式的审计模型，这不仅没能充分利用区块链技术的优势，也无法很好地保证审计的安全性和去中心

性。去中心化是区块链技术的核心特性，也是我们得以构建分布式审计机制的关键。如何充分利用这一特点，构建规则公开、过程可追溯、结果可验证的审计方案，是目前的一个研究难点。

3. 将审计规则和审计记录公开在审计区块链上，可以实现审计的公开透明以及可追溯、可验证。然而，区块链采用全副本存储模型，每个共识节点需要保存完整的区块链副本。这导致节点存储开销较大，存储可扩展性较弱。大部分云存储用户不具备足够的存储资源，或者没有意愿贡献过多存储资源。过大的存储开销可能导致用户对审计系统的使用意愿降低，而节点数量过少将影响分布式系统的去中心性和安全性。假如存储可扩展性的问题不解决，审计区块链系统将难以构建、难以实现、难以运行。在目前提出的基于区块链的审计方案中，尚且没有关于可扩展性优化的讨论。

此外，最近几年多云存储已经逐渐取代单云存储或者私有云存储，成为众多企业的首要选择。借助多云可以将数据分散存储到不同的云上，使得每个云服务商只掌握部分用户数据，降低了用户数据泄露的风险。多云存储架构还可以规避单个云服务商发生故障或是内部人员恶意操作^[24]等行为导致的数据损坏、服务故障等问题。在这一背景下，如何在多云散布系统中实现对数据完整性的保护成为一个十分重要的研究问题。为多云系统设计适合的审计机制，需要一套有效的分析和评价方法作为指导，从而切实提高云存储数据可靠性。但是目前在远程数据审计领域仍旧缺乏这样一套理论，指导我们为多云存储系统制定有效的审计策略和修复策略。具体地，如下三方面问题有待深入探索。

1. 衡量远程数据审计机制对多云存储系统数据可靠性的提升效果。虽然远程数据审计技术已提出多年，中心化的公开审计方案也得到了较为充分的研究，但是很少有研究工作分析引入审计机制的云存储系统的可靠性。这种分析十分重要，可以证明审计机制在数据保护方面的实际效果。
2. 对比不同审计策略和修复策略在可靠性和资源消耗等方面的差异，从而判断什么样的策略是适合多云系统的。例如，提高审计频率可以让我们更及时地发现数据损坏从而进行修复，但同时也使得系统的网络流量消耗增大，加重系统负载。不同的审计策略影响了数据被选中的概率，不同的修复策略影响了数据修复成功率。我们需要量化分析各种影响因素，并进行综合判断。
3. 为云存储系统设计有效且高效的审计方案，以较低的成本保证较高的数据

可靠性。分析的最终目标是指导数据审计方案的制定。我们希望能够以较低的网络流量消耗，达到期望的系统整体可靠性水平。

综上，在如今云存储服务广泛普及的大时代背景下，为了从根本上提升数据完整性保护措施的可信性和可行性，迫切需要研究安全、可信、高效的远程数据审计方案，这主要涉及三方面的挑战：

1. 为保证远程数据审计的安全可信，需采用去中心化审计模型，构建规则公开、过程透明、结果可验证的审计协议，并由此构建分布式审计系统。
2. 为保证审计区块链系统的可扩展性和可行性，需针对区块链存储模式进行优化，在降低节点存储开销的同时保证链上数据的访问性能。
3. 为研究引入审计机制的多云存储系统的可靠性，需要构建一套适用于多云审计系统的可靠性分析方法，并根据分析结果制定安全、高效的审计策略。

第二节 研究内容和创新点

本文的核心研究主题是面向大规模云存储系统，借助区块链技术设计安全可信且高效的远程数据审计方案，保证公开审计的可监管性和可验证性以及审计区块链系统的可扩展性和安全性，从而加强对云存储系统数据完整性的保护。基于这一研究目标，提出如下三个研究问题（RQ, research question）。各研究问题之间的关系如图1.2所示。

- RQ1：如何设计具有零知识属性的分布式公开审计协议，以实现透明、可监管的公开审计，同时不泄露用户隐私？
- RQ2：如何设计适用于云存储的审计区块链系统，并进行存储模式优化？
- RQ3：如何制定多云可靠性评价方法，以量化分析审计策略对于云存储系统的提升效果，从而设计合适的审计方案？

具体地，本文的研究内容及相关创新点可以归纳为如下三点。

（一）隐私友好的分布式公开审计方案

我们将研究问题 RQ1 拆分成两个子问题，一是如何设计透明、可追溯、可监管的公开审计模型，二是如何实现具有零知识隐私保护性质的分布式审计协议。由此提出以下研究内容。

1. 为彻底解决传统公开审计的单点故障问题，采用去中心化审计模型、构建分布式审计网络。在分布式审计网络中，每个用户既作为审计者贡献计算资源，又作为审计服务使用者享受高质量服务。设计审计者选举算法以确

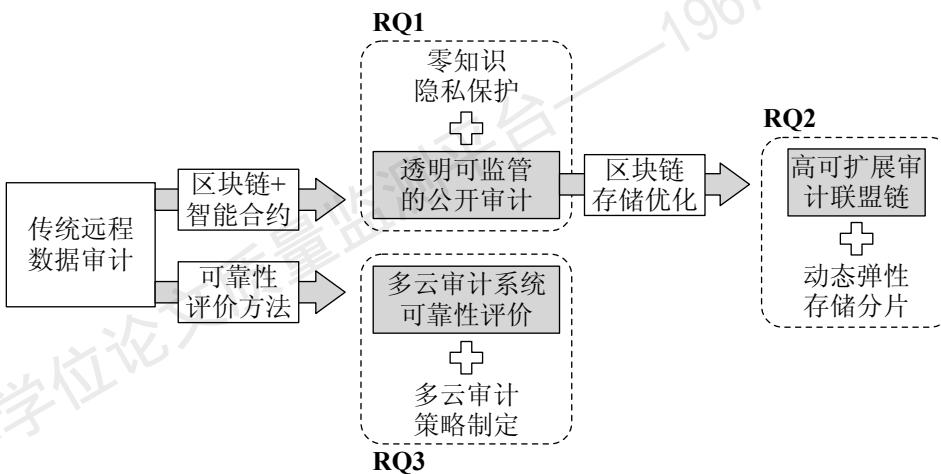


图 1.2 各研究问题之间的关系

保审计者中大部分为诚实用户，避免恶意用户操控审计结果。我们认为在云上存储了更多文件的用户有更大的可能性积极维护系统安全，因此设计选举算法以使得这类用户有更大概率被选中。审计者选举不仅能够保证审计结果的正确性，还能够保证审计频次的稳定性。考虑到云存储用户持有资源和在线时长参差不齐的现实，设计激励机制以促使更多用户参与到审计活动，维持网络活性。资源充裕的节点可以多参与审计，同时获得更多奖励，而资源不充裕的节点可以少参与甚至不参与审计。提出“为存储可靠性付费”的付费模式以规范云服务商行为。激励机制由智能合约自动执行。利用上述方法，无需依赖可信第三方审计者实现了分布式审计，很好地规避了基于 TPA 的中心化审计的固有弱点。

2. 为保证审计规则公开、过程公开、可追溯，使用智能合约自动执行分布式审计协议，实现了审计的不可抵赖性和纠纷定责。我们将审计相关算法实现在智能合约中并部署上链，使得所有用户皆可查看审计规则。智能合约在执行过程中将审计记录全部上链，向所有用户及云服务商公开，从而实现了审计过程的公开性和可监管性。整个审计过程处于监督之下，可追溯可审计。我们将链上公开信息作为伪随机种子生成挑战信息，实现了审计的不可预测性和不可操控性，很好地规避了审计者错误行为或恶意行为导致的问题。
3. 为解决公开全部审计记录可能引发的隐私泄露风险，基于零知识证明技术提出具有隐私保护性质的分布式审计协议。我们经过研究和对比发现，传

统的隐私保护方法¹普遍存在验证时间长且证明长度长的问题，因此并不适合于区块链审计方案。因此，我们提出满足如下两点要求的零知识审计协议。（1）验证时间快。验证过程由云存储用户或是智能合约执行，因此其计算复杂度必须足够低。相对地，证明生成过程由云服务商执行且云服务商具备较强计算能力，因此我们能够容忍该过程计算复杂度较高。（2）证明长度短。证明需要存储到链上，而链上的存储空间有限，因此证明长度必须足够短。

（二）实现存储优化的审计联盟链系统

我们将研究问题 RQ2 拆分成两个子问题，一是如何构建适用于云存储系统的审计联盟链，二是如何设计适用于审计联盟链的存储优化方案。由此提出以下研究内容。

1. 针对第一个子问题，我们考虑以下事实。（1）云存储用户分为个人用户和企业、机构用户，他们自身硬件设施不同，对文件完整性保护的需求也不同。企业用户一般拥有企业级服务器，计算能力和带宽资源较为充足，而个人用户往往不是；企业用户存储在云端的数据大多关乎企业经济利益或用户个人利益，因此对文件完整性保护的要求较高，对审计可靠性的要求也较高。（2）企业用户一般服务于很多个人用户，他们之间存在层级关系。根据以上事实，我们提出具有分层结构的适用于云存储场景的审计联盟链系统模型。云服务商（作为被审计者）和所有用户（作为审计者和监督者）都是联盟链节点。根据持有资源多少的不同，云存储用户被划分为联盟链服务节点和轻节点两类。服务节点参与共识并负责维护区块链、存储一部分区块链数据；轻节点只使用区块链服务，不维护数据。云服务商有充足的存储和计算资源，因此作为联盟链全节点维护完整的区块链数据副本。
2. 基于所提出的系统模型构建审计联盟链系统。该系统包括三个主要工作模块。云存储模块包含与文件上传下载相关的功能，主要涉及用户与云服务商之间的交互；远程审计模块包含与数据审计相关的功能，涉及到用户与区块链、云服务商与区块链之间的交互；存储分片模块实现联盟链存储模式优化，将所有用户划分成若干存储组并规定同一存储组内用户之间的交互规则。在一次审计过程中，轻节点或服务节点通过调用智能合约对云服

¹在传统的隐私保护审计方案中，一般采用对完整性证明中的聚合文件块进行盲化处理的方法，以使得审计者无法获取到关于原文件块的信息。

务商发起挑战或是验证云服务商提交的回复；云服务商通过调用智能合约查看挑战内容并进行回复，以证明文件完整性。文件上传记录、数据审计记录以及审计历史访问记录全部作为监管凭证记录在区块链上。

3. 针对审计联盟链系统设计一种动态弹性存储分片方案。每个服务节点只需存储一部分区块链数据，并通过节点间协作访问本地未存储的数据。该方案的主要特点之一是能够根据实时区块访问热度动态调整区块的副本个数、存储位置和存储模式，从而在数据可靠性、数据访问性能和节点存储开销之间取得平衡。当一个区块上链后，服务节点运行初始散布机制决定是否保存这一新区块。每个新区块在存储组内有固定数量的初始存储节点，从而保证数据可靠性。若某历史区块在近期访问频次较高，动态弹性复制机制会及时增加该区块在存储组内的副本个数，从而降低数据访问时延。若某历史区块在一段时间内未被访问，存储优化机制将这类冷区块成批编码，从而降低存储开销。特别地，存储优化过程不会影响数据可靠性。

（三）多云审计系统存储可靠性评价

我们将研究问题 RQ3 拆分成两个子问题，一是如何为引入审计机制的多云存储系统建立可靠性评价方法，二是如何对比分析不同审计策略从而制定适用于多云存储系统的审计方案。由此提出以下研究内容。

1. 传统的云存储可靠性分析多聚焦于磁盘故障，如磁盘老化、崩溃、出现扇区错误等等。然而相比较于单云存储系统，多云存储系统的复杂度更高，引发故障的因素也更多。因此，仅仅考虑单云内局部组件层面的故障是远远不够的，必须从系统整体的可靠性入手进行分析，才能更好地判断远程数据审计机制的整体效果。因此，本文着重分析在大规模多云存储系统中，单个云服务商的可靠性机制失效导致多云存储系统发生数据丢失的情况。
2. 本文使用指数分布函数模拟单个云存储系统上发生的故障事件。指数分布可以帮助我们直观分析一些重要参数对于系统可靠性的提升效果，例如调整审计频率对可靠性的影响以及不同审计策略的表现差异等等。虽然有学者提出韦布分布能够更好地刻画存储系统中的磁盘老化现象，但是云存储系统内运行的磁盘故障预警机制会削弱磁盘老化的作用，并且电力故障、系统漏洞、自然灾害、人为失误等复杂因素均会影响系统可靠性。因此，使用指数分布刻画系统整体的平均故障率是更为合适的。
3. 本文使用蒙特卡洛算法模拟多云存储系统动态运行和修复的过程，并统计

发生数据丢失事件的期望时间。我们用仿真程序的运行结果验证马尔可夫的计算结果，并进行辅助分析。分析结论证明了审计机制对于提升多云系统可靠性有非常好的效果。结合多角度的可靠性分析和网络流量分析，我们为多云存储系统制定了恰当的区块链审计策略。

第三节 文章组织结构

本文立足大规模云存储环境，研究基于区块链的远程数据审计方法。第二章介绍了云存储的研究背景以及相关工作；第三章到第五章分别介绍三个主要研究内容；第六章对本文的主要研究进行总结并给出下一阶段的研究方向。各章具体内容安排如下。

第一章为绪论。首先总览目前云存储的发展现状以及研究背景，对远程数据审计方案中的私有审计和公开审计分别进行介绍和分析；然后，总结出目前有待解决的问题并简要给出研究思路；最后，介绍本文的主要研究内容及创新点，并给出各章节安排。

第二章介绍背景知识和相关工作。首先介绍适用于云存储及P2P存储的数据散布技术以及零知识证明技术；然后重点介绍了远程数据审计技术，包括传统审计技术以及近几年研究较多的基于区块链的审计技术，还介绍了区块链存储可扩展性优化技术；最后介绍了云存储可靠性评价方法。

第三章提出了基于区块链的分布式审计框架和相应的公开审计协议。首先介绍审计框架；基于该框架给出两个审计协议，其中一个协议基于零知识证明技术实现了隐私保护；最后，针对系统的安全性进行了理论分析，针对系统的性能和可行性进行了实验测试。

第四章提出了适用于云存储的审计联盟链系统，并基于该系统设计区块链存储分片方案。首先介绍审计联盟链系统的系统模型和工作流程，然后介绍存储分片模块的设计细节，最后针对存储分片模块和审计联盟链系统分别进行实验测试。

第五章从系统整体层面分析了引入远程数据审计机制的多云存储系统的可靠性。借助多角度的可靠性分析和网络流量分析，给出了设计区块链审计策略的综合性建议。

第六章总结本文的研究工作，并进一步给出未来研究方向。

第二章 背景知识和相关工作

本章在后续小节中对如下内容进行了介绍。首先，第一节约回顾了传统的数据散布技术以及目前广泛应用于区块链系统中的零知识证明技术。第二节介绍了远程数据审计技术的基本原理和研究脉络。接着，第三节详细介绍了近几年提出的基于区块链技术的分布式审计方案。第四节分析了区块链系统目前在存储可扩展性方面遇到的问题，并给出了工业界和学术界已提出的几类解决方案。第五节约回顾了云存储系统可靠性评价方法的发展脉络。

第一节 背景知识

2.1.1 云存储系统数据散布技术

分布式云存储系统（下文简称“云散布系统”）将多个服务器或多个云看作一个存储集群，将文件切分成若干个文件块后分别散布到集群中的不同位置。当用户想要读取文件时，系统检索该文件所有文件块的一个子集即可成功还原出文件。上述数据散布和还原过程由一个或多个节点执行中心化调度。相较于将整个文件直接存储在某个服务器的存储方式，分布式存储提供更高的可用性和可靠性，且能够抵御单点故障。近年来提出的云散布系统包括 Oceanstore^[25,26], Pergamum^[27], POTSHARDS^[28], PASIS^[29], Gridsharing^[30], Tahoe-LAFS^[31] 等等。

云散布系统采用的散布策略一般基于副本技术或者编码技术，图2.1对比了这两种策略。在三副本散布系统中，每个文件块拥有3个副本，每个副本被散布到不同的存储位置；在(4,2)编码散布系统中，由4个文件块生成2个校验块，并将这6个块散布到不同的存储位置。可以看到，这两种散布策略都可以使系统具有容忍双节点故障的能力，但是二者的存储开销差异较大。记 d 为原文件的大小，则三副本散布系统的存储开销是 $3d$ ，而 $(k,k+2)$ 编码散布系统的存储开销是 $(1+\frac{2}{k})d$ 。另外，编码散布系统还能够提供检错、纠错等能力。从节省存储开销等角度，编码散布方式相较于副本散布方式更优。

一般地，我们说编码散布系统的门限为 (r,k,n) 表示该散布系统将原数据散布到 n 个存储位置，使得利用 k 个数据分片即可恢复原数据，同时使用少于 r 个

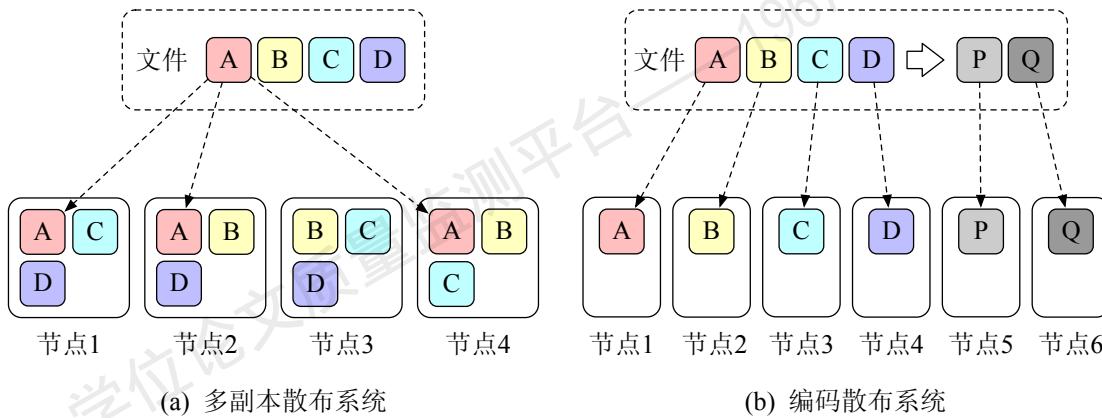


图 2.1 散布策略对比。

分片无法推断出关于原数据的任何信息¹。散布过程可以抽象成图2.2所示。散布矩阵 $G = \{g_{i,j}\}$ 包含 n 行 k 列元素，其中每个元素包含 w 位数据。由原数据生成包含 k 个数据块的向量 $\vec{D} = (d_0, d_1, \dots, d_{k-1})$ ，其中每个数据块包含 w 位数据。将矩阵 G 乘以向量 \vec{D} 我们得到长度为 n 的向量 $\vec{C} = (c_0, c_1, \dots, c_{n-1})$ ，此即为将要被散布给 n 个参与者的数据块集合。构建散布矩阵时要保证 G 中任意 k 行元素构成的子矩阵都是可逆矩阵，因此使用向量 C 中的任意 k 个元素都可以恢复出原数据 D 。

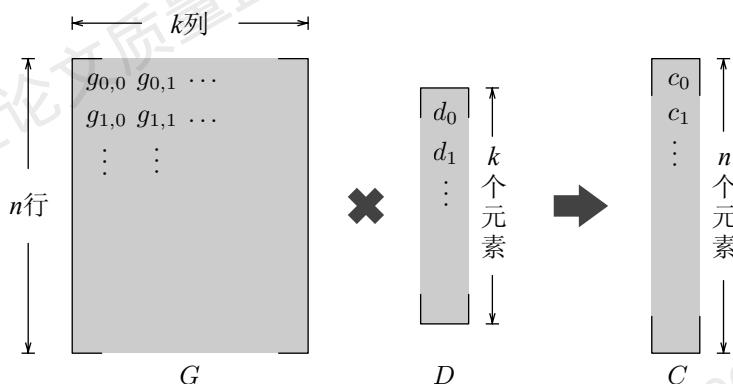


图 2.2 信息散布过程示意图。

在散布存储系统中常常使用到秘密共享、信息散布、加密等技术，或根据不同的使用需求对这些技术进行组合。秘密共享技术^[32,33]的特点是高安全性，高计算开销，高存储开销。其门限一般为 (k, k, n) 。例如，Shamir 秘密共享算法

¹本文中讨论的编码散布系统是具有数据隐私保护能力的。

(SSSS, shamir's secret sharing scheme) 在生成向量 \vec{D} 时将原数据进行编码得到 d_0 , 而 d_1, \dots, d_{k-1} 都是随机生成的值, 且长度与 d_0 相等。散布矩阵 G 是范德蒙矩阵, 即有 $g_{i,j} = i^j$ 。当 $n \leq 2^w$ 时矩阵 G 中任意 k 行元素组成的子矩阵都是可逆矩阵。记原始数据的长度为 b , 则算法的存储开销为 nb , 且计算过程需要执行 $O(knb)$ 次异或及乘法操作²。尽管存储开销及计算开销较大, 该算法达到了信息论安全性 (information-theoretic security): 即使攻击者拥有无限算力, 在持有少于 k 个秘密分片的情况下也无法获取关于原始数据的任何信息。秘密共享技术一般不适合于存储较大的数据, 而更适合于存储数据量小且安全性需求较高的数据。信息散布算法 (IDA, information dispersal algorithm) 由 Rabin 于 1989 年首次提出 [34], 其特点是低安全性, 低计算开销, 低存储开销。Rabin IDA 基于纠删码技术 [35], 向量 \vec{D} 中每个元素 d_i 都包含一部分原始数据的信息, 所生成的向量 \vec{D} 的数据长度等于 b , 因此算法存储开销仅为 $\frac{nb}{k}$, 接近最优。不过, Rabin IDA 的安全性较低, 门限为 $(1, k, n)$ 。即使攻击者只获取到 1 个分片, 也有可能推断出关于原数据的部分信息。因此, IDA 主要用于保证分布式存储系统的数据可靠性。

基于秘密共享和 IDA 各自的特点, 一些研究指出可将两者结合以构建安全、高效的数据散布方案。例如, 在 SSMS [36] 中提出如下散布方案: 文件 F 经过加密得到密文 E ; 对密文使用信息散布算法得到 n 个密文分片 E_1, \dots, E_n ; 对密钥使用秘密共享技术得到 n 个密钥分片 K_1, \dots, K_n 。将组合分片 $S_i = (E_i, K_i)$ 散布给 n 个参与者。SSMS 方案能够达到计算安全性 (computational secuerity): (1) 攻击者想要获取到原数据必须先尝试解密, 而加密算法是计算上难解的; (2) SSMS 使用 Shamir 秘密共享保护密钥安全性, 因此持有少于 k 个密钥分片无法破解出加密密钥。AONT-RS [37] 结合加密技术 [38] 以及纠删码 [39] 构建了安全、高效、可验证完整性的散布方案。该方案将文件与加密密钥拼接成 AONT 包 (AONT, All-or-nothing Transform), 然后使用 RS 码 (RS, Reed Solomon) 将 AONT 包编码成若干数据块。AONT-RS 实现了比 SSMS 更低的开销。近些年, 也有学者提出了一些无需额外生成随机密钥的散布方案 [40-43]。表格2.1给出了一些方案的对比。其中 RS 码指的是 Reed-Solomon 码, 变量 b^{key} 和 b 分别代表密钥和文件的长度。

²秘密共享算法生成的秘密分片的长度一般不小于原秘密长度, 因而存储开销不小于 nb 。

表 2.1 散布方案对比。

无密钥	是否加密	机密性 & 可靠性	安全性	存储开销
SSSS [33]	×	完备秘密共享	$(n, k, k-1)$	n
Rabin IDA [34]	×	纠删码	$(n, k, 0)$	$\frac{n}{k}$
基于密钥	密钥类型	密钥安全 文件安全	安全性	存储开销
SSMS [36]	随机密钥	SSSS Rabin IDA	$(n, k, k-1)$	$\frac{n}{k} + n \cdot \frac{b^{\text{key}}}{b}$
AONT-RS [37]	随机密钥	AONT RS 码	$(n, k, k-1)$	$\frac{n}{k} + \frac{n}{k} \cdot \frac{b^{\text{key}}}{b}$
CAONT-RS [40]	数据哈希	AONT + RS 码	$(n, k, k-1)$	$\frac{n}{k} + \frac{n}{k} \cdot \frac{b^{\text{key}}}{b}$

2.1.2 点对点存储系统数据散布技术

点对点 (P2P, peer-to-peer) 存储与云存储相对。P2P 存储系统中不存在中心化的管理节点，而是由对等节点 (peers) 构成自组织的覆盖网络。每个节点都贡献一部分本地存储空间用以保存系统内的数据，并为其它节点提供数据路由和数据访问服务。P2P 存储系统具有许多优点，如鲁棒的广域路由，高效的数据检索，高可扩展性，高容错性等等。自 2000 年之后，有许多 P2P 文件共享及存储系统被提出，例如 Napster [44]，LOCKSS [45]，Mnemosyne [46]，GFS [47]，Elephant [48]，PAST [49]，BitTorrent [50] 等等。

P2P 存储系统一般使用副本策略进行数据散布。给定若干节点和若干数据对象，P2P 散布算法确定每个数据对象的副本个数及每个副本的存储位置。由于系统往往包含成百上千个节点且节点流动性很大，P2P 散布算法一般采用高冗余度的散布策略，以在较大概率下保障数据可用性。P2P 散布方案的主要研究目标是在节点流动性较高的情况下保证数据高可用性、高访问性能以及负载均衡 [51]。根据适用系统类型的不同，P2P 散布算法可分为三类。

- 适用于结构化 P2P 系统。结构化 P2P 系统的网络拓扑具有确定的结构，且数据存储位置遵循一定的规则，因此数据检索更为高效 [52,53]。结构化 P2P 散布算法 [54-58] 一般基于分布式哈希表 (DHT, distributed hash table)。给定数据对象的标识符，DHT 将其映射到存储该数据的节点标识符。系统中主要操作为 $\text{Put}(key, value)$ 和 $value \leftarrow \text{Get}(key)$ ，其中 key 为数据对象标识符而 $value$ 为数据对象。为了支持数据存储和检索，每个节点需要维护一个部分路由表，记录其邻居节点的 nodeID 及 IP 地址。数据检索性能指标为成功找到某一数据对象平均所需最小 DHT 跳数。一般地，当网络中节点个

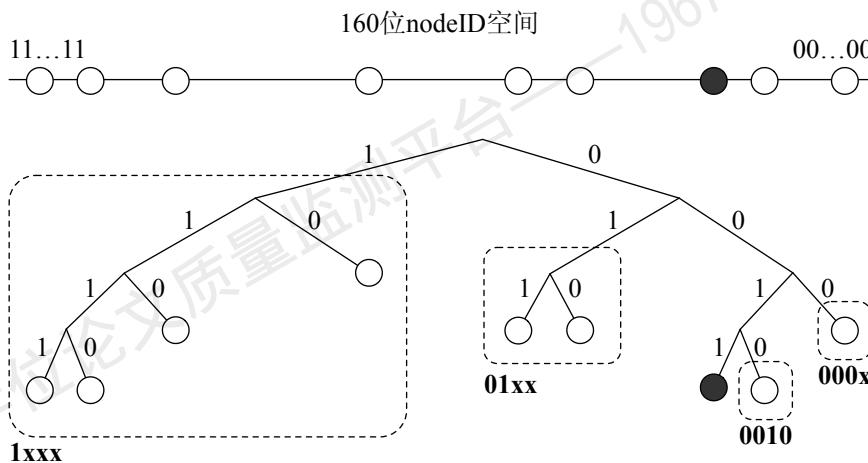
数为 N 时，任意数据对象都能够在 $O(\log N)$ 跳数内检索到。

2. 适用于非结构化 P2P 系统。非结构化 P2P 系统没有确定的网络拓扑结构，且数据散布位置没有特定规则^[59,60]。非结构化 P2P 散布算法^[61,62]的数据查找过程一般基于洪泛技术（flooding），随机游走（random walk）或扩展环搜索（expanding ring）。以洪泛查找为例，每个节点在接收到请求后会转发给符合条件的若干个邻居节点，因此这一请求将在一定范围内大量传播。每条请求设置有存活时间（TTL, time to live），用以限制一条消息的最大转发次数。数据检索性能指标为成功检索某一数据对象平均所需问询的节点数。基于洪泛的 P2P 散布算法在检索热数据时性能表现良好，并且能够很好地应对节点进出系统的情况，但是检索冷数据的性能表现不佳。
3. 适用于 P2P 社区网络。假设 P2P 社区网络与一个外部的更大的网络（如广域网）相连接，当数据对象在社区内部检索失败时可以在外部网络中继续检索^[63-65]。数据检索性能指标为数据对象在 P2P 网络内的命中率。

Kademlia^[66] 是一个经典的 DHT 算法。系统中每个节点被分配一个 160 位的标识符 nodeID，节点所发送的每条消息都会包含其 nodeID。每个数据对象也被分配一个 160 位的标识符 key。每个数据对象表示为 $\langle key, value \rangle$ 。Kademlia 将每个对等节点视作一个二叉树中的叶子节点，如图2.3给出了一个示例。对于前缀为 0011 的节点，节点二叉树被划分为 4 个子树，称为“K 桶”。每个 K 桶的节点前缀依次为 1、01、000、0010，分别包含了 $1/2$ 、 $1/4$ 、 \dots 比例的节点。每个节点维护一个部分节点列表，包含每个 K 桶内至少一个节点的路由信息³。每当节点接收到一个来自其它节点的信息，便会更新本地节点列表。Kademlia 数据散布基于节点与数据对象之间的逻辑距离。给定两个标识符 x 和 y ，定义二者之间的距离为两标识符按位异或的结果，即 $d(x, y) = x \oplus y$ 。每个数据对象都会被分配给距离它最近的节点，因此在 Kademlia 数据存储和检索过程中一个最重要的操作就是寻找最近节点。该过程递归进行，每个接收到消息的节点会从节点列表中距离目标 ID 最近的 K 桶里选出 α 个节点转发消息，直至找到目标节点或者目标数据。

设计适当的数据复制策略可以提升 P2P 存储系统中数据访问性能和数据可用性，并能使负载更为均衡。例如，路径复制将被请求的数据对象复制在搜索路径上的全部或一部分节点^[67,68]。邻居复制指将被请求的数据对象（或数据对

³路由信息指 nodeID、IP 地址和 UDP 端口

图 2.3 Kademlia 节点二叉树^[66]。

象的索引) 复制在数据持有节点的邻居节点^[69]; 请求者复制指将被请求的数据对象复制在请求节点或由请求节点指定的若干节点; 持有者复制指由数据持有节点指定被请求数据对象将被复制到的位置。

2.1.3 零知识证明技术

零知识证明 (ZKP, zero-knowledge proof) 是一类隐私保护协议, 可以使证明者在不泄露任何有关被证明信息的前提下向验证者证明自己知道该信息。最早开始提出的 ZKP 协议是交互式的^[70]。后来有学者提出非交互式零知识证明框架 (NIZK, non-interactive zero-knowledge proof), 其证明过程无需证明者与验证者进行交互^[71]。这一改进使得零知识证明协议的适用范围变得更广且可操作性更强。这之后, NIZK 的一个研究难点在于如何缩小证明长度^[72,73], 以及如何将证明长度压缩至常量级别^[74-76]。

零知识简洁非交互式知识论证 (zk-SNARK, zero-knowledge succinct non-interactive argument of knowledge) 是目前在区块链领域使用较为广泛的一种非交互式零知识证明框架^[77]。证明者可以在不泄露任何有关被证明信息的前提下向验证者证明自己知道该信息, 因此非常适合货币系统、金融系统等有隐私保护需求的场景。证明过程是非交互式的, 所以证明者可以在链下生成证明并发布到区块链上, 而验证者可以在链上执行验证。zk-SNARK 验证速度快 (时间复杂度为次线性), 因此将验证过程实现到智能合约中可以达到高效自动化验证。同时, zk-SNARK 证明长度短且为常数级别, 因此非常适合存储到区块链上作为

存证。zk-SNARK 的诸多优点使得它在近些年获得广泛关注，并在提升区块链系统隐私性^[78-80]、区块链系统扩容^[81,82]、隐私保护^[83]、数据审计^[22,84]、电子投票^[85]等领域有了很多应用。

根据可信设置的不同，零知识证明算法可以分为两类。（1）无需可信设置，代表性算法如 zk-STARKs^[86]、Bulletproofs^[87] 等。这类方案安全性最高，但是验证性能低（时间复杂度为近线性）且证明长度大。因此方案的适用性较为有限。（2）基于可信设置。这类方案需要通过可信设置生成公共参考字符串（CRS, common reference string），方案的安全性很大程度依赖于 CRS 的安全性。以 plonk^[88] 为代表的算法只需执行一次可信设置便可应用于不同证明问题，称为通用 zk-SNARK 算法。以 Pinocchio^[89]、Groth16^[90] 为代表的算法需要针对每个电路都执行一次可信设置、生成不同的 CRS，称为非通用 zk-SNARK 算法。虽然后者的安全性较之前者稍弱，但是验证性能更高（验证算法的时间复杂度与算术电路大小无关）且证明长度固定。例如，Pinocchio 协议可以做到证明生成复杂度与算术电路规模呈线性关系且验证时间少于 10ms，而 Groth16 协议在证明长度和验证复杂度方面比 Pinocchio 协议更优。目前，Groth16 算法在区块链系统中应用十分广泛。

在 zk-SNARK 中，根据要证明的问题和待证明的声明（statement），我们可以构建一个算术电路。算术电路中每一个门（gate）代表一个约束，即模加或者模乘操作。每一根电线（wire）代表一个输入信号或输出信号，信号的值为域 \mathbb{F}_p 上的元素。一个算术电路对应一组约束条件，称为 Rank-1 约束系统，其中每一个约束条件必须是常量表达式、线性表达式或二次表达式。给定算术电路以及一组输入信号，我们可以得到输出信号。定义见证者 $w = (s_1, s_2, \dots, s_n)$ 为能够满足算术电路中所有约束的一组信号的值。由此，原证明问题被转化为证明我们知道正确的 w 。如图2.4给出一个示例。假设我们想证明我们知道整数 c 的因数分解结果，则该问题将被转化为证明我们知道满足约束条件 $s_1 * s_2 = s_3$ 的 s_1, s_2, s_3 的值。证明过程一般包含三个步骤。（1）设置。根据所构建的算术电路生成两个公开密钥 pk 和 vk ，前者用于证明者生成证明而后者用于验证者验证证明。（2）证明。根据证明密钥 pk 、公开信息、以及见证者 w 生成零知识证明 π 。（3）验证。任何人都可以使用公开的验证密钥 vk 、公开信息、以及零知识证明 π 执行验证。验证过程也可以通过智能合约来执行。

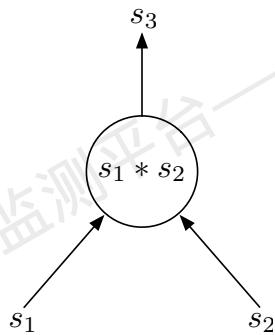


图 2.4 算术电路示例 [91]。

第二节 远程数据审计技术

由于云上存储的数据量巨大且多为重要、敏感数据，云存储用户的一个主要担心是云上数据可能会丢失或损坏^[92]。大型云服务商曾经出现的一些突发故障^[93, 94]更是加重了云存储用户的担心。为了解决这一问题，Ateniese 等人^[6]于 2007 年提出了可证明数据持有（PDP, provable data possession）的概念，并基于同态可验证标签构建了无需下载原始文件的完整性验证方案。同年，Juels 等人^[7]提出了可检索性证明（PoR, proof of retrievability）的概念，并基于纠错码构建了可检索性证明方案。这类 PDP 和 PoR 方案统称为远程数据审计（RDA, remote data auditing），其设计初衷是用一种高效（无需下载原数据）的方式可靠地证明用户在云上的数据是完好的。一次典型的 RDA 审计遵循“挑战-回复”范式，其基本过程如下。1) 挑战：审计者随机选中云上的一些文件（块）并要求云服务商证明其完整性；2) 回复：云服务商检索被选中的数据并计算完整性证明；3) 验证：审计者验证完整性证明并得出审计结果。近些年，学者们提出了很多 RDA 方案。我们可以根据不同的标准对现有方案进行分类。例如，现有方案可分为针对静态场景^[8, 95, 96]或是动态场景^[10, 97–99]（“动态”指支持数据增删改操作）。再如，我们可以按照方案是否实现了隐私保护功能、是针对单服务器还是多服务器^[100]进行分类。特别地，根据审计执行的主体，现有方案可分为私有审计和公开审计这两类。私有审计中的审计者为数据所有者；公开审计中的审计者一般为可信第三方机构。表格 2.2 给出了对一些主要方案的对比。

2.2.1 可证明数据持有

静态场景。Ateniese 等人于 2007 年首次提出了可证明数据持有（PDP, provable data possession）模型^[6]。该模型使得用户能够以高效、安全的方式验证

表 2.2 远程数据审计方案对比

PDP 类	协议名	主要技术	静态	动态	隐私保护
私有 审计	RDPC [8]	对称加密, 代数签名	✓		
	PPDP [101]	双线性对	✓		
	Scalable PDP [97]	对称加密		✓	
	E-DPDP [102]	同态区块标签		✓	
公开 审计	PDP [6]	基于 RSA 的 HVT	✓		
	ES-PDP [103]	椭圆曲线加密, HVT	✓		
	DPDP [10]	同态区块标签		✓	
	Public PDP [98]	同态线性验证子		✓	
	PP-PDP [104]	带随机掩码的 HLA			✓
PoR 类	Sec-PDP [105]	基于承诺的采样			✓
	PoR [7]	哨兵, RS 码	✓		
	CPoR [12]	HLA, 伪随机方程	✓		
	DPoR [9]	哨兵		✓	
公开 审计	CPoR [12]	HLA, BLS 签名	✓		
	Public PoR [106]	多项式承诺	✓		
	FD-PoR [11]	哈希压缩并签名		✓	

云端保存的文件块是否正确，且无需下载存储在云端的文件块。审计过程中选中云端数据的一个随机子集以生成概率性的数据持有证明。审计协议基于 RSA 同态可验证标签 (HVT, homomorphic verifiable tag)。用户端需要保存的元数据以及审计过程中需要传输的数据量都为常数级别，因此 PDP 模型非常适合于使用分布式系统存储大量数据的场景。文中提出了两个 PDP 方案，支持公开审计，不过所提出的方案只适用于静态场景。2013 年提出的 ES-PDP [103] 基于椭圆曲线密码系统构造了既支持私有审计又支持公开审计的方案。用户和第三方都可以执行审计协议。该方案对私有审计和公开审计使用相同的初始化流程，生成相同的元数据集，且在随机预言机模型下被证明安全⁴。2013 年提出的 PPDP [101] 是一种使用了代理 (proxy) 的 RDA 方案，可以将远程数据审计的任务授权给代理来执行。审计协议基于双线性对技术，共包含四个步骤：(a) 由第三方生成系

⁴证明过程假设椭圆曲线离散对数问题是难解的。

统参数和公钥；(b) 用户对代理进行授权和签名，代理验证授权及相应签名；(c) 用户将文件块及相应标签上传到云端，云端对文件块及标签执行验证；(d) 代理针对云上数据执行审计，包含挑战、回复和验证。PPDP 只支持静态场景。

动态场景。 2008 年，Ateniese 等人首次提出了适用于动态场景的 RDA 方案 Scalable PDP^[97]。该方案利用对称加密技术提升了审计系统的可扩展性和性能，并能够支持部分动态操作（文件块追加、删除、修改操作）。Scalable PDP 只支持私有审计。用户在上传文件前需要预先计算出一些用于执行验证的标签，其中每个标签由一个随机挑战和一个随机数据块集合生成。因此，文件在上传至云端后只能执行有限次审计。2009 年，Erway 等人提出了两个支持全部动态操作（文件块插入、删除、修改操作）的 DPDP 方案^[10]。方案一使用了基于排名信息的验证字典^[107] 和跳表^[108]，实现了对动态操作的高效验证。方案二使用基于排名的 RSA 树，提高了检测到错误区块的概率。不过，在执行动态操作时的计算复杂度较高。2012 年提出的 E-DPDP 方案^[102] 致力于降低动态审计方案的开销。作者提出了一种新的二叉树结构，称为块更新树。树中每个节点包含四类数据：1) 操作类型（删除对应-1，修改对应 0，插入对应 1）；2) 数据块索引范围；3) 数据块索引偏移量；4) 版本号。块更新树的一个特点是无论进行多少次动态操作，树总是平衡的；另一个特点是树的大小与其所对应的文件大小无关。为支持对动态操作进行验证的功能，数据所有者和云服务器都需要保存块更新树。默克尔哈希树（MHT, merkle hash tree）是一种简单且高效的支撑验证的二叉哈希树^[109]，可用于检测数据篡改。Public PDP^[98] 便使用了 MHT 技术，并使用双线性聚合签名^[110] 解决 MHT 的平衡问题。当需要更新某个数据块时执行如下操作。数据所有者将新的数据块及相应验证子发送给云端；云端在接受到数据后执行更新操作，生成新的 MHT 树根并更新聚合块标签，将签名后的 MHT 树根返回给数据所有者；最后，数据所有者验证签名的树根以确认更新操作是否正确执行。

2.2.2 可检索性证明

可检索性证明（PoR, proof of retrievability）除了能够提供数据完整性验证之外，还能够在发现错误数据后进行数据恢复，恢复过程一般基于前向纠错码（FEC, forward error-correcting code）。PoR 与 PDP 的主要不同在于前者会存储一些冗余文件块以保证数据可靠性、支持数据恢复等功能^[111]。

静态场景。 Juels 等人于 2007 年首次提出了 PoR 模型^[7]。在该方案中，文

件在上传到云端之前被嵌入一些由单向函数计算得出的哨兵块。若文件遭到篡改，有一定的概率会使得哨兵块被篡改。PoR 的验证过程即随机选中一部分哨兵块并验证其完整性。该方案的局限在于执行审计的次数受限。2008 年，Shacham 和 Waters 提出了更为安全高效的 CPoR 方案，并设计了一个公开审计协议和一个私有审计协议。私有审计基于同态线性验证（HLA, homomorphic linear authentication）和伪随机方程；公开审计基于 BLS 签名^[112] 将多个 PoR 证明聚合成一个 PoR 证明，降低了网络传输开销。该方案还使用了 RS 编码（RS, reed solomon）实现错误数据恢复功能^[113]。基于同态承诺技术构建的 PoR 方案普遍存在通信开销过大的问题，因为 PoR 证明的大小与文件大小线性相关。针对该问题，Public PoR^[106] 使用常数级多项式承诺技术实现了常数级的通信开销，同时打开承诺的计算复杂度也是常数级别的。为了降低方案所生成的挑战的数据长度，作者借鉴了文献[114] 所使用的碰撞采样器（hitting sampler）算法^[115] 来高效地生成挑战。

动态场景。大部分 PoR 方案无法支持动态场景的一个主要局限在于，云上存储的数据包含一些计算出的冗余块，因此即使只是改动原文件中很少量的数据也需要重新上传大量数据，这将导致方案十分低效。DPoR^[9] 实现了支持任意数据读写操作的 PoR 协议，并使得客户端和云端的计算开销和传输开销保持在多重对数级别。DPoR 将文件划分成若干文件块，并对每一个文件块进行 RS 编码，因此对某个文件块的写操作不会影响到其它块。为了防止云服务器识别出哪些块是原始文件块而哪些是冗余块（进而删除掉冗余块以使得文件无法恢复），DPoR 借助不经意随机访问机（ORAM, oblivious read access machine）技术^[116] 来隐藏用户的访问规律。ORAM 是一种层级式数据结构，包含多层哈希表。顶层表存储最近被访问过的数据，底层表存储最少访问的数据。当用户访问某个数据块时，系统首先通过检索哈希表得到目标数据的存储位置，然后再随机产生几个对其它数据块的访问行为，以此来隐藏用户的访问规律。FD-PoR 方案^[11] 借助基于范围的 2-3 树和哈希压缩并签名（HCS, hash-compress-and-sign）构建了支持动态场景的 PoR 方案，并能够确保不诚实的用户无法诬陷云服务器篡改了数据。

2.2.3 隐私保护公开审计

从隐私保护的角度，用户不希望公开审计的执行过程引发任何信息泄露的风险。然而，很多 RDA 方案并没有考虑到对于用户隐私的保护，因而可能会产

生两方面问题，即用户数据泄露和用户身份泄露。具体内容阐述如下。

针对数据泄露的隐私保护。在大部分公开审计方案的设计中，云服务商提交的回复包含对多个被审计文件块的线性组合。如果 TPA 掌握了同一组文件块的不同线性组合，便有可能还原出文件块的内容。在很多应用场景下（尤其是涉及医疗、金融数据等隐私敏感型数据的场景），这类问题是必须严格避免的。因此，一些研究者提出对回复中的敏感内容进行盲化的方法。例如，文献 [104] 使用随机掩码技术（random masking）对文件块线性组合进行盲化，从而避免 TPA 破解出原数据块的内容。另外，文中还提出了一种实现可证明零知识泄露（provably zero knowledge leakage）的公开审计方案，进一步提升了隐私保护强度。文献 [99] 提出适用于动态场景的隐私保护审计框架，其提出的公开审计协议在随机预言机（random oracle）模型下证明安全。文献 [117] 基于整数同态加密技术构造了具有隐私保护功能的审计方案，且能够同时保证数据机密性（该方案允许对云上密文进行操作）。文献 [118] 利用密钥同态密码原语构造了基于身份（identity-based）的隐私保护审计协议，能够避免使用公钥加密体系（PKI, public key infrastructure）所引发的密钥管理等问题。该方案实现了零知识隐私性。文献 [119] 针对电子健康记录大数据系统提出能够进行聚合验证的审计协议，同时支持隐私保护、动态操作以及批量审计。

针对身份泄露的隐私保护。TPA 在审计过程中会了解到用户与文件之间的关系，而这可能会泄露用户隐私。因此，一些研究者提出对用户身份进行隐藏的方法。这类方案一般假设用户数据在共享组内进行共享。例如，Knox^[120] 使用群签名来构造审计验证子，从而保证 TPA 无法得知被审计文件块归属于共享组内的哪个用户。基于 Knox 的工作，文献 [121] 中提出了支持动态场景的隐私保护 RDA 方案。Panda^[122] 考虑了用户撤销（user revocation）问题：若组内某用户退出，则其负责签名的文件块必须由留在组内的某用户进行重签名。Panda 使用了代理重签名技术使得用户无需下载数据便可完成重签名过程。文献 [123] 针对一个文件可以由多个用户进行修改的场景，考虑了用户撤销和合谋攻击（collusion attacks），即多个用户联合执行恶意操作的攻击。该方案使用基于多项式的审计验证予以允许不同文件块的验证子进行累加，并能够将用户撤销操作安全地委托给云服务商执行。文献 [124] 考虑了用户撤销期间用户与云服务商合谋进行攻击的场景，使用矢量承诺（vector commitment）技术以及局部验证者撤销（verifier-local revocation）的群签名技术来对抗该攻击。文献 [125] 使用了基

于身份 (identity-based) 的审计协议, 避免了使用 PKI 引发的密钥管理等问题, 并且实现了对文件上传者身份的隐藏以及高效的审计过程。NPP^[126] 针对于文件共享场景, 提出基于同态可验证群签名技术的隐私感知 (privacy aware) 审计方案。共享群内的用户可以追溯数据修改历史并修复被损坏的数据块。

总结上述方案, 我们发现传统远程数据审计方法存在一些内生的弊端。首先, 由数据所有者执行的私有审计缺乏纠纷裁决机制, 无法实现不可抵赖性。再者, 依赖 TPA 实现不可抵赖性的公开审计在实际使用中会遇到单点故障、TPA 作恶等问题, 无法达到理想中的安全性。一系列研究工作表明, 基于区块链的分布式数据审计可以很好地规避传统远程数据审计的弊端, 并带来新的优势。因此, 本论文提出分布式审计模型, 基于区块链技术构建分布式审计网络并设计分布式审计协议。详细内容请见第三章。

第三节 基于区块链的数据审计技术

2.3.1 区块链技术概述

2008 年, 区块链作为比特币系统^[13] 中的分布式公开账本技术被首次提出。区块链由含有时间戳的区块依次链接而成, 其中每个区块包含系统在一段时间内产生的交易, 并通过哈希指针指向下一个区块。每个新产生的区块在经过节点验证并被认为有效之后被追加到区块链中, 且数据一旦上链则不能删除、不能修改。区块链账本被复制给系统内每一个节点进行保存, 这称为“全副本”存储模式。共识机制可以保证所有节点保存的账本内容一致。以上设计使得区块链在无需可信第三方的前提下实现了分布式系统拜占庭容错、数据不可篡改和可追溯等性质。因此区块链特别适合于互不信任的合作者之间需要共享信息的场景。近几年, 区块链已经被应用到各个领域, 如数据完整性保护^[14], 数据溯源^[127], 数据管理^[128] 等等。

以太坊^[129] 是一种可编程的区块链系统。通过在区块链上部署智能合约^[130], 所有区块链节点都可以调用合约并运行合约中所包含的代码⁵, 且运行过程和运行结果是公开的、安全可信的。一般地, 调用并运行合约中某个函数包含四个步骤。(1) 调用者创建一笔交易, 给出被调用函数及相关参数; (2) 以太坊矿工将该交易打包进某个新区块, 运行被调用函数并得到函数输出; (3) 以太坊矿工发布新区块并将其广播至全网; (4) 全网节点接收到新区块后运行区块

⁵一般地, 合约调用者需要支付一定费用。

内包含的所有交易并验证，若验证成功则接收该区块，否则丢弃该区块。智能合约可以增强多方合作的系统的安全性，因此该技术已被应用于各个领域，例如加强公钥体系的安全性^[131]，在数据使用过程中保护用户隐私^[132]等等。

根据网络架构的不同对现有的区块链系统进行划分，则比特币和以太坊都属于公有链。在公有链系统中，节点之间不存在层级关系，任何节点都可以参与共识。而在 Hyperledger^[133]、Tendermint^[134] 和 Multichain^[135] 等许可链(permissioned blockchain) 系统中，只有一部分节点有权参与共识并验证区块。由于共识节点较少，许可链系统一般会采用 PBFT^[136] 等更为高效的共识协议，交易吞吐率也更大。不过，公有链中没有任何中心化机构也不需要信任前提，而许可链中需要一个可信第三方对节点身份进行授权。

2.3.2 针对云存储的区块链数据审计

在文献[16]中，提出了一种基于智能合约的私有审计方案。在其审计模型中，用户作为审计者，通过调用智能合约执行私有审计。云服务商和用户都是区块链轻节点，而共识节点由外部节点负责。在一次审计过程中，数据所有者(挑战者)生成挑战并发送给合约，云服务商(证明者)读取链上公开的挑战信息并向智能合约提交回复，而数据所有者(验证者)读取链上提交的回复信息并执行验证。共识节点只负责执行合约并出块，但并不验证回复、也不负责给出审计结果。链上记录私有审计过程的所有历史信息。特别地，该方案实现了纠纷定责功能。当出现纠纷时，通过调用智能合约特定接口进行纠纷定责。DIRA^[15]提出了一种适用于多云存储架构的公开审计方案。在其审计模型中，由所有云服务商作为审计者以及区块链的共识节点，用户不参与审计。在一次审计过程中，任意云服务商(挑战者)生成挑战并发起一笔交易将该挑战包含在交易内，而被审计的云服务商(证明者)计算出回复并发起一笔交易将该回复包含在交易内。记账节点会将包含着挑战或回复的交易打包进新区块。共识节点(验证者)在验证新区块时执行对回复的验证从而得到审计结果。将信誉值作为出块奖励以鼓励云服务商积极出块。链上存储审计过程的历史信息以及每个云服务的信誉值。CAB^[17]提出一种适用于多云存储架构的公开审计方案。在其审计模型中，所有用户被划分成若干组，每个组由一个资源较充裕的用户担任管理节点。共识节点由所有云服务商和一部分被选中的管理节点组成。在每轮共识中，从所有共识节点中选出一个记账节点。在一次审计过程中，任意用户(挑战者)生成一个挑战并发送给本组的管理节点，管理节点对该挑战签名后广播至全网。

云服务商（证明者）在收到挑战后计算出回复，并将该回复广播至全网。记账节点（验证者）收集所有挑战和回复并执行，然后将验证通过的审计记录打包到新区块中。最后，记账节点将新区块广播至全网。特别地，该方案支持批量审计（batch auditing）。CPVPA^[18]提出了一种借助区块链技术对第三方审计者实行监督的公开审计方案，不仅可以发现审计者的恶意行为，而且可以发现不及时执行验证的拖延行为。在一次审计过程中，TPA 根据区块链上最新的 φ 个块的哈希值生成挑战信息并发送给云服务商。云服务商将计算出的回复发送给 TPA 后，TPA 执行验证并将本次审计写入一笔新交易，发布在区块链上。用户可以通过审计链上记录来监督 TPA 的行为。文献 [19] 提出一种基于区块链的公开审计方案。区块链用于可靠存储数据块的标签，用以作为审计时的验证子。为了在审计过程中快速找到所需验证子，使用 T-Merkle 哈希树优化区块链数据结构，以快速执行二分查找。在一次审计过程中，审计者生成挑战选中某些文件块，并将挑战分别发送给区块链及云服务商。区块链和云服务商分别生成对数据标签的证明和对数据块的证明，发送给审计者。审计者执行验证并得出审计结果。上述“审计者”既可以是数据所有者，也可以是云服务商、第三方等等。共识节点只负责执行合约并出块，但并不验证回复、也不负责给出审计结果。

特别地，文献 [137] 提出了一种基于区块链且实现隐私保护的公开审计方案。在其审计模型中仍旧保留了 TPA。为了防止 TPA 作弊，将区块链上的公开信息作为输入以生成随机的挑战信息，同时要求 TPA 将所有审计记录公开在区块链上以备查验和审计。所有用户都可以通过访问链上信息来验证审计执行过程。为了防止链上公开的审计记录泄露用户隐私，该方案基于 CPoR 方案^[12] 进行改进并构造零知识证明。云服务商在计算出文件块证明 μ 和文件签名证明 S 后对二者进行盲化，从而使得云服务商提交的恢复不泄露关于原文件的任何信息。不过，相比较于 CPoR，该方案在计算回复和验证回复时的计算复杂度大大增加。表格2.3给出了二者在证明长度和验证复杂度方面的对比。同时我们还给出了2.1.3小节介绍的 Groth16 零知识证明算法的证明长度和验证复杂度。表格中 $|G|, |G_T|, |\mathbb{Z}_p|$ 表示群 G, G_T , 域 \mathbb{Z}_p 的元素长度。 m, n 分别指 Groth16 使用的算术电路中电线（wire）和门（gate）的个数。 ℓ 指 Groth16 证明者要证明的声明（statement）的长度。 E 指群 G 上的求幂运算， P 指配对（pairing）运算。 M, M_z 指群 G , 域 \mathbb{Z}_p 上的标量乘法。 c 指挑战中包含数据块的个数。从表格中可以看出，与 CPoR 相比，zk-CPoR 的证明长度以及验证复杂度都大大增加。而与

zk-CPoR 相比, Groth16 在同样保证零知识的前提下, 证明长度更短且为定值, 且验证复杂度也更低。

表 2.3 审计方案性能对比。

	证明长度	验证复杂度
CPoR ^[12]	$ \mathbb{G} + \mathbb{Z}_p $	$2P + 2cE + (2c - 1)M$
zk-CPoR ^[137]	$11 \mathbb{G} + \mathbb{G}_T + 5 \mathbb{Z}_p $	$6P + (3c + 7)E + (3c - 3)M + 2cM_z$
Groth16 ^[90]	$3 \mathbb{G} $	$\ell E + 3P$

2.3.3 针对 P2P 存储的区块链数据审计

随着区块链技术的发展, 许多学者提出了适用于 P2P 存储系统的分布式审计方案。对等节点运行基于区块链的审计协议来相互检验数据完整性, 在无需第三方的前提下保证了审计的可靠可信。

在比特币的挖矿过程中, 全网矿工持续尝试破解哈希算力难题直到某个矿工算出正确结果并获得记账权。这一过程耗费大量算力(电力)且计算结果不具有现实用处。因此, 2014 年, Permacoin^[138] 的作者提出使用有意义的存储证明(PoS, proof of storage) 来替代比特币中无意义的工作量证明(PoW, proof of work)的思想。参与挖矿的 Permacoin 节点不仅需要贡献算力, 还需要贡献存储空间来存储一部分文件块。在文件存储过程中, 文件所有者将文件默克尔根作为公开信息发布在区块链上; 在挖矿过程中, 节点提交自己保存的一部分数据块的完整性证明(包含文件块及其默克尔路径)。这一完整性证明将被全网节点进行验证, 若验证成功则矿工成功赚取到挖矿酬劳。Permacoin 存在的不足是完整性证明的长度过长。假设证明者选中了 k 个文件块, 则证明包括 k 个文件块及 k 个默克尔路径。2016 年提出的 Retricoin^[21] 针对这一问题进行了优化。Retricoin 借鉴了 CPoR 协议^[12], 将多个完整性证明聚合成一个完整性证明, 使得矿工提交的证明长度更短, 带宽消耗更低。2016 年提出的 KopperCoin^[139] 利用区块链为分布式存储系统提供有效的激励机制, 解决用户参与度不足(导致数据可靠性不佳)的问题。KopperCoin 将比特币网络中的 PoW 替换为可检索性证明(PoR, proof of retrievability)。在一次审计过程中, 矿工根据给定规则选择一个数据块并计算其完整性证明, 然后进行挖矿。矿工存储的数据量越多则在挖矿中获胜的概率越大。

Sia^[140]、Filecoin^[22] 和 BlockStore^[141] 是基于区块链的分布式存储系统。它们都将区块链作为用户和存储节点之间发布信息的公开平台。例如，Sia 使用智能合约来管理存储节点和文件所有者之间的行为。当某个用户想要存储文件时，其发布一个存储约定，包含文件默克尔根、存储截止日、审计频率和每次审计的酬劳。在文件发送给存储节点进行存储后，节点需要定期执行数据审计。审计过程与 Permacoin 类似，节点随机选中一些文件块并计算存储证明，然后将存储证明（包含文件块及其默克尔路径）提交给区块链网络。若证明被其它节点验证有效，则存储节点能够获得审计酬劳。Filecoin 使用区块链构建了两个市场，其中存储市场用于用户发布待存储文件的信息，而检索市场用于用户发布文件检索请求。Filecoin 使用时空证明（proof of spacetime）来证明文件在一段时间内被相应节点完好保存。Blockstore 也将区块链作为分布式的交易市场，但与 Sia 和 Filecoin 不同的是，Blockstore 中的审计过程由用户执行。Storj^[142] 是一种存在弱中心节点的分布式存储系统。卫星节点（satellites）负责 Storj 网络中的协调和管理工作。分布式审计既可以由用户执行也可以由 satellites 执行。与 Permacoin 和 Sia 类似的是，Storj 节点提交的完整性证明也是数据块及其默克尔路径，因此 Storj 也存在证明长度较长、带宽消耗较大的问题。2020 年提出的 Audita^[143] 提出了链上链下结合的分布式存储方案。由链外节点负责数据存储，由区块链节点对链外节点执行数据审计。该方案适用于任何区块链系统，并且能够保证分布式数据审计的安全可靠。

2017 年，文献 [14] 提出一种适用于物联网（IoT, internet of things）环境的私有审计方案。数据审计不仅可以由数据所有者执行，还可以由获得数据所有者授权的数据访问者执行。文献 [144] 基于智能合约设计适用于边缘计算环境的公开审计方案。在其存储模型中，边缘节点负责数据存储，服务商负责协调和管理边缘节点。在其审计模型中，从所有系统节点中选出审计者构成审计委员会。选举过程中要保证边缘节点不能自己审计自己、数据所有者不能自己审计自己的数据。在一次审计过程中，任意审计委员会成员（挑战者）生成一个挑战并发送给智能合约。被审计的边缘节点（证明者）计算回复并将其提交给智能合约。最后，审计委员会所有成员对提交的回复执行验证并将验证结果发送给智能合约。链上记录公开审计过程的所有历史信息。文献 [145] 基于智能合约设计适用于 IoT 环境的公开审计方案。其审计模型中仍旧保留了第三方审计者（TPA, third-party auditor），审计过程与传统基于 TPA 的公开审计类似，只不过用

户、云服务商和TPA之间的每一步交互都是通过调用智能合约来完成的。历史审计记录全部上链，实现了审计的可追溯可审计。特别地，智能合约维护所有审计者的信誉值并定期除掉信誉值过低的审计者。表格2.4给出了基于区块链的数据审计方案的对比。

表 2.4 基于区块链的数据审计方案对比

审计类型	架构	方案名	挑战者	验证者	共识节点
私有审计	单云	文献 [16]	数据主	数据主	外部节点
	IoT	文献 [14]	数据主	数据主	外部节点
公开审计	单云	CPVPA ^[18]	TPA	TPA	外部节点
		文献 [19]	数据主	数据主	/
	多云	DIRA ^[15]	数据主	CSP	CSP
		CAB ^[17]	数据主	记账节点	CSP、数据主
	P2P	Audita ^[143]	共识节点	共识节点	外部节点
		Permacoin ^[138]	存储节点	记账节点	存储节点
		Retricoin ^[21]	存储节点	记账节点	存储节点
	IoT	文献 [144]	审计委员会	审计委员会	/
		文献 [145]	第三方审计	第三方审计	外部节点

总结上述方案我们看到，基于区块链的远程数据审计技术还有待进一步研究和拓展。首先，现有方案多针对P2P系统或者IoT系统设计，针对云存储系统的方案较少，而后者才是目前普及率更高、受众更多的云存储服务。另外，在现有方案中区块链数据的维护依赖于外部节点或少量节点，这一做法削弱了系统安全性。因此，本论文采用将审计网络与区块链网络合一的设计思想，使得所有用户既能够参与审计又参与区块链维护。设计区块链存储优化方案从而在节点开销、数据访问性能和数据可靠性之间取得平衡。详细内容请见第四章。

第四节 区块链存储优化技术

近年来，区块链技术在各领域得到广泛应用，尤其联盟链技术凭借高吞吐率、隐私性及多场景支持性应用最广^[146,147]。区块链系统要求全节点存储完整数据副本，而这导致全节点存储开销过大，并逐渐引发新的问题。据统计，2019

年以太坊数据总量超过 2TB，2021 年比特币数据总量超过 350GB^[148]。联盟链交易吞吐率高于公有链，因此数据增长更快。假设某联盟链平均区块大小为 2MB，出块间隔为 2 秒，则一个月将产生 2.5TB 数据。过大的存储开销引发了如下两方面问题。

- 全节点数量减少，系统安全性降低。由于使用普通硬件设备难以胜任全节点所负责的数据存储任务，全网范围内全节点数量减少。这将削弱系统去中心性，并影响分布式系统安全。
- 新加入节点同步数据困难。对新节点来说，同步数据的时间开销和带宽开销过大，且扫描区块建立世界状态时的计算开销过大。

因此，优化区块链尤其是联盟链存储模式，降低全节点存储开销成为一个研究方向^[149]。许多研究工作致力于提升区块链系统的存储可扩展性。已提出的方案可分为如下三类。

SPV 方案。简单支付验证 (SPV, simplified payment verification) 使得区块链轻节点仅存储少量数据便可验证交易合法性。例如，Bitcoin 轻节点仅需存储区块头^[13]。当验证一笔交易时，轻节点向全节点索要默克尔证明，以证明该笔交易已被打包到最长链。NIPoPoWs 轻节点无需存储任何数据^[150]。当验证一笔交易时，轻节点向全节点索要后缀证明以证明该全节点持有当前最长链，索要前缀证明以证明指定交易已被打包到最长链。不过，NIPoPoWs 方案只在挖矿难度保持不变时有效。FlyClient^[151] 方案在挖矿难度变化的情况下仍旧有效，且证明长度比 NIPoPoWs 更短。

修剪方案。区块链修剪技术 (pruning) 允许节点删除一部分历史数据并且不影响节点参与共识。已提出方案可归纳为如下两类。(1) 区块修剪。比特币允许节点只存储最近新产生区块^[152] 或某个高度之后的区块^[153]。CoinPrune^[154] 允许节点定期创建区块链状态快照并删除快照之前所有区块。不过，若系统中大部分节点执行了区块修剪，则历史数据的可访问性和可靠性大大降低，给数据审计带来困难。(2) 交易修剪。Jidar^[155] 允许节点只保存与自己相关的交易，即自己是发送方或接收方的交易。文献 [156] 提出联盟链节点可以删除不影响区块链状态且预计未来一段时间不会被访问的交易。不过，交易修剪的弊端在于，节点在尝试恢复完整区块的过程中需要与多节点进行交互，导致数据恢复效率低下。

存储分片。存储分片可视为特殊的修剪策略，因为节点之间的行为是协作

式的：将全网划分若干存储组，在组内使用区块散布机制保证每个节点存储的区块集合各不相同，从而所有节点可以“拼凑”出完整的区块链副本。已提出的存储分片方案可归纳为如下几类。（1）优化问题求解。以 CUB^[157] 为代表，这类方案对区块散布问题进行形式化定义并求近似最优解。优化目标是降低区块访问时延^[158, 159] 或保证区块链数据可用性^[160]。不过，上述方案主要针对静态场景（即区块链数据不增长、访问概率分布已知），导致方案适用性受限。另外，所提出的区块散布算法需要中心化管理节点，影响系统去中心性和安全性。（2）基于编码。这类方案针对区块链增长的动态场景，将新区块先编码再散布。GCBLOCK^[161] 提出基于纠删码的区块散布方案，但散布过程需要中心化管理节点。BFT-Store^[162] 结合了纠删码和副本两种存储技术，且实现拜占庭容错，但并未考虑对不同热度的区块弹性调整散布策略。（3）复制率调度算法。ElasticChain^[163] 和 SE-Chain^[164] 提出动态调整不同区块的复制率以保证系统安全性，但并未考虑节点间数据请求的效率问题。较之于修剪方案，存储分片方案能够更好地在系统安全性、数据可靠性和数据检索性能间取得平衡。已提出的存储分片方案各有所长，但针对动态场景的分布式区块散布算法还有待进一步研究和探索，尤其应结合区块访问热度进行性能和存储方面的优化。

总的来看，SPV 方案增强了系统可扩展性但削弱了系统去中心性，且存储节点开销未得到改善。修剪方案能够降低存储节点开销，但数据可靠性和可访问性大大降低，在区块链审计系统中这是无法容忍的。存储分片方案具有很多优点，但是针对动态场景的区块散布算法仍有待进一步研究。本论文提出适用于云存储审计联盟链的动态弹性存储分片方案。根据实时区块访问热度动态调整区块的副本个数、存储位置和存储模式，从而在数据可靠性、数据访问性能和节点存储开销间取得平衡。详细内容见第四章。

第五节 云存储可靠性评价方法

可靠性分析（reliability analysis）一直是云存储数据保护领域的研究热点。其中，“可靠性”指云存储系统在给定时间内能够正常运转、不发生故障或数据丢失的概率^[165, 166]。可靠性评价指标包括平均失效时间（MTTF, mean time to failure）、平均数据丢失时间（MTTDL, mean time to data loss）等等。其中，MTTDL 指标衡量存储系统平均发生一次数据丢失事件的期望时间，也可表示为存储系统在给定运行时间内（一般为 3 到 5 年）发生数据丢失事件的期望个数

或概率 [167]。

2.5.1 针对磁盘故障的可靠性分析

在大规模云存储系统中可能会发生服务中断、数据丢失或者损坏等故障，影响系统可靠性和可用性。诱发原因包括磁盘故障、设备故障、软件故障、网络故障、电力故障、人为失误等等 [168-173]。这其中，研究得最多的是磁盘故障导致的可靠性降低问题。数据中心大多使用成本较为低廉的磁盘作为存储设备。在拥有众多磁盘的大型数据中心发生磁盘故障的概率是较高的，这类故障多由偶然因素或是磁盘老化导致。影响云存储数据可靠性的磁盘故障类型主要有如下两种。

- 块损坏故障，即磁盘内某个数据块无法读取或者数据内容发生错误。发生块损坏故障的主要原因有两种，一是硬盘介质损坏或是被磁头划伤导致一至多个二进制位永久损坏，二是软件缺陷或固件缺陷导致数据块内容发生错误。云存储系统一般会定期执行磁盘清洗过程以及时修复发生损坏的块 [174]。
- 磁盘运行故障，即整个磁盘永久性无法访问。在拥有上万块磁盘的大型数据中心，磁盘运行故障的发生频率是比较高的。故障磁盘一旦被检测到会立即后台执行重构过程，该过程一般需要十几至几十小时。

云存储系统一般采用冗余存储方式以提高数据可用性和可靠性，因此多个磁盘发生运行故障或者块损坏时才有可能导致数据丢失。下面我们就四种广泛使用的存储方式分析其发生数据丢失的可能性。

RAID 阵列存储。每个 RAID-5 阵列包括一个校验盘，至少两个存储盘。给定阵列内磁盘个数为 g ，则每个校验组包含 $g - 1$ 个数据块以及根据这几个数据块生成的 1 个校验块 (parity block)。如图2.5(a) 所示，在存储校验组的时候进行混合散布，即每个磁盘既保存数据块、也保存校验块。当有一个磁盘发生运行故障或块损坏故障时，系统进入降级模式。此时再有另一个运行故障发生，则校验组发生数据丢失，无法修复 [175]。如图2.5(b) 所示，RAID-6 阵列的一个校验组中包含两个校验块，并且采用混合散布策略。当有两个磁盘发生运行故障或块损坏故障时（可能是两个运行故障，也可能是一个运行故障和一个块损坏故障），系统进入降级模式。此时再有另一个运行故障发生，则校验组发生数据丢失，无法修复 [176]。

副本存储。副本存储系统为每个数据块生成若干副本，并按照两个基本规

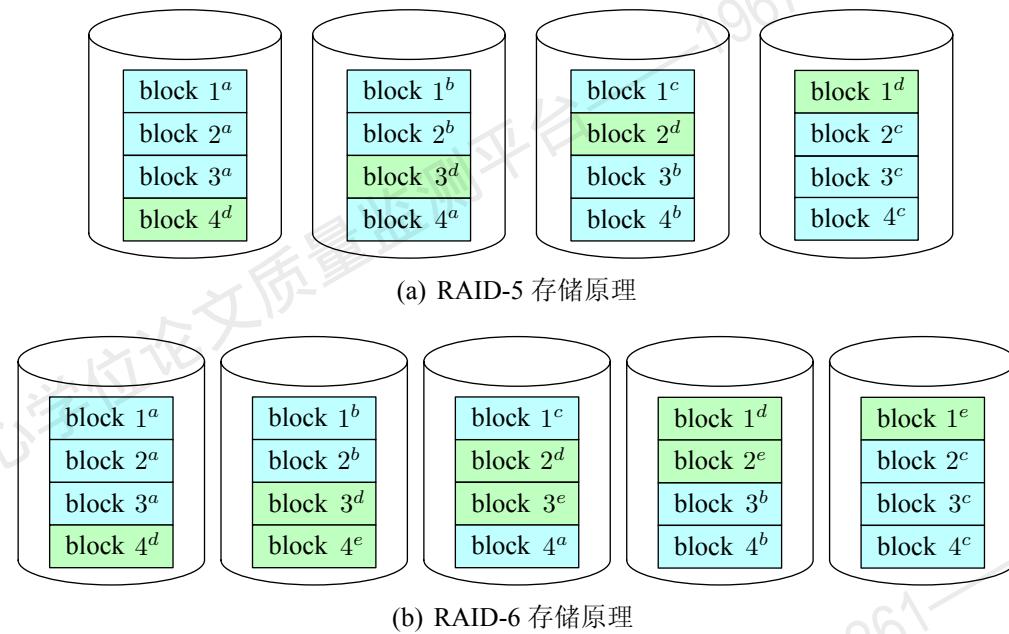


图 2.5 RAID 存储原理示例 [176]

则将数据块散布到不同的磁盘上：（1）同一数据块的不同副本必须散布到不同的存储节点；（2）同一数据块的所有副本必须散布到至少两个机架。对于二副本存储系统，一般假定系统内存储数据块数量足够多，使得任意两个机架上的硬盘都存有相同的数据块。因此，假如两个机架上同时发生了磁盘运行故障或是其中一个机架发生运行故障后另一个机架发生块损坏故障，数据丢失事件便会发生。三副本系统将同一数据块的三个副本散布到三个节点，其中两个节点在同一个机架上。因此，假如两个不同机架上的三个磁盘发生运行故障，或是其中两个发生运行故障、一个发生块损坏故障时，数据丢失事件便会发生。

马尔可夫模型是评价云存储可靠性的常用分析方法。该模型使用状态表示系统内当前发生故障的个数，使用状态转移事件表示系统由 i 个故障转变为 j 个故障的过程。马尔科夫模型假设故障发生的时间服从指数分布，并且假设故障的发生彼此独立。不过，一些工作指出使用指数分布模拟硬盘故障率并不符合海量存储场景的实际数据 [177]，而韦布分布能够更为准确地刻画因磁盘老化导致磁盘故障的现象 [178-180]。因此一些学者提出使用韦布分布结合组合概率分析 [181] 的方法来计算云存储系统可靠性。综合来看，马尔可夫模型的劣势在于不能准确描述磁盘老化现象，而其优势在于能够直观刻画系统内的故障发生和修复过程，比较适合用于对比不同策略的效果或是分析某些参数对于系统可靠

性的影响。组合概率分析的劣势在于无法刻画故障修复过程，而其优势在于计算速度快且适用于多种故障分布函数。

2.5.2 针对外部攻击的生存性分析

传统可靠性分析方法只考虑系统内部的偶发性故障，并未考虑到云存储系统可能遭遇的恶意攻击。然而，随着云存储服务广泛普及，云上数据受到的针对性的、恶意的攻击也越来越普遍^[182]。例如，攻击者出于损害其它公司声誉等目的，会发动数据修改攻击^[183]以试图擦除、篡改存储在云上的用户数据。再如，攻击者锁定某个目标文件并持续性地尝试损坏该文件所在校验组的分片，直至文件无法重构。因此，除了评价云存储自身能够提供的可靠性外，评价云存储系统在可能遭遇外部攻击的情况下正常提供服务的能力也是十分必要的。生存性（survivability）指系统在同时面对外部攻击和内部故障的情况下仍旧能够完成既定任务的能力^[184-186]。使用生存性分析模型可以更完整、更真实地反映云存储系统在实际运行时的表现。

例如，文献[166]考虑了存储系统可能遭遇拜占庭故障的情况，并给出了一般性的可靠性评价公式。定义可靠性方程 $R(t)$ 为系统直到时间 t 仍旧正常运转的概率，并假设 $R(0) = 1$ 。故障率 $\lambda(t)$ 表示单位时间发生故障的个数⁶，将故障率视为常数可以计算出可靠性方程：

$$R(t) = e^{-\int_0^t \lambda(x)dx} = e^{-\lambda t}$$

给定系统内共有 n 个服务器，每个服务器的可靠性方程均为 $R(t)$ 。定义整体可靠性方程 $R_s(t)$ 为 n 个服务器中有至少 k 个未发生故障的概率，于是有：

$$R_s(k, n, t) = \sum_{i=k}^n \binom{n}{i} R^i(t) (1 - R(t))^{n-i}$$

定义攻击者方程 $C(t)$ 为系统直到时间 t 被攻击的服务器的个数。对于攻击者进行如下假设：(1) 不考虑对服务器的部分控制，认为攻击者要么攻击失败、要么完全掌握某个服务器的控制权；(2) 不考虑系统修复，一旦攻击者掌握了某个服务器则服务器无法恢复。由此定义出如下四类攻击：

1. 固定时间：攻击者需要 p 时间来成功操控系统内所有服务器。例如，所有服务器运行同样的操作系统而攻击者发现了某个系统漏洞，因此可以一次性操控所有服务器。

⁶经验主义故障率 $\hat{\lambda}(t)$ 通常使用观察到的发生一次故障的时间间隔（的倒数）来计算。

2. 线性时间：攻击者每攻击一个服务器都需要 p 时间。有 $C(t) = \min\{\lfloor t \cdot p \rfloor, n\}$ 。

例如，每个服务器运行不同的操作系统因此攻击者每次都需要重新发现系统漏洞。

3. 对数时间：攻击者攻破第一个服务器时需要固定时间，后续攻击所花时间遵循对数曲线（即攻击速度越来越慢）。有 $C(t) = \min\{\lfloor \log_p t \rfloor, n\}$ 。例如，系统一旦发现某个服务器遭受攻击便会加强防御系统，因此使得攻击难度越来越大。

4. 多项式时间：攻击者攻破服务器的速度越来越快。有 $C(t) = \min\{\lfloor t^p \rfloor, n\}$ 。

例如，攻击者在攻击过程中持续发现一些系统秘密（如秘密共享相关参数）因而攻击得越来越快。

给定系统拜占庭容错节点数为 f ，则系统整体可靠性评价公式为：

$$\begin{aligned} R_C(f, n, t) &= R_s(n - f, n - C(t), t) \\ &= \sum_{i=n-f}^{n-C(t)} \binom{n - C(t)}{i} R^i(t) (1 - R(t))^{n - C(t) - i} \end{aligned}$$

当系统内攻击者成功攻击 $f + 1$ 个服务器时，系统崩溃。

总结上述方案我们发现，现有可靠性评价多针对单个云服务商发生磁盘故障或是遭遇攻击的情况，少有对于多云存储系统的可靠性评价，而多云架构目前正受到越来越多的关注和使用。另外，目前缺少对引入审计机制的云存储系统的可靠性分析。这种分析十分重要，可以证明审计机制在云存储数据保护方面的实际效果，并为审计方案的设计提供理论指导。本论文第五章针对这一问题进行研究，并建立了一套针对多云审计系统的可靠性评价方法。

第六节 本章小结

本文的研究工作主要关注于基于区块链的数据可靠存储和数据完整性审计。我们将基于本章中所介绍的部分前人工作，提出适用于云存储系统的分布式公开审计方案，并研究如何借助零知识证明框架实现隐私友好的分布式公开审计协议。针对区块链系统所存在的存储可扩展性问题，我们将提出一套全新的存储优化方案，从而提升审计联盟链系统的安全性和可扩展性。我们将为多云存储系统提出一套可靠性评价方法，用于分析审计机制对于系统可靠性的提升作用，以及不同审计策略的实际效果。

第三章 隐私友好的分布式公开审计方案

第一节 引言

如今，越来越多企业、机构和个人使用云计算服务构建面向社会大众的数字平台。云存储作为云计算的基础模块，在其中扮演着十分重要的角色。通过使用云存储服务，用户将繁杂的数据存储、数据管理、数据维护等任务外包给云服务商，并享受方便高效的远端数据访问^[187]。不过，数据外包也引发了一定的安全风险，尤其是数据完整性遭破坏的风险^[188]。存储在云上的数据可能会因为物理故障、人为失误或者恶意攻击而出现数据损坏甚至丢失的情况^[189, 190]，而这将导致云计算平台的故障或错误，造成不良影响。公开可验证的远程数据审计技术（以下简称“公开审计”）致力于解决上述问题并为云存储用户提供安全高效的数据完整性保护方案。在公开审计中，用户委托一个可信第三方审计者（TPA, third-party auditor）代表用户对云上数据执行完整性验证，审计规则一般遵循数据持有证明^[6] 或可检索性证明^[7] 等审计协议。审计过程一般包含三个步骤：首先，TPA 选择用户数据的一个随机子集生成挑战，并要求云服务商提供被选中数据的完整性证明；然后，云服务商按照要求计算完整性证明并回复给 TPA；最后，TPA 检验接收到的完整性证明并报告本次审计的结果（成功或失败）。随机的、频繁的审计能够及时发现数据损坏或数据删除，并采取适当修复措施。

传统公开审计方案一般假设 TPA 是可信的，然而这一假设并不符合实际^[18]。因此，传统公开审计方案在实际使用中会遇到三类问题。1) 单点故障。一旦 TPA 的服务器发生物理故障或者人为失误，审计将被迫中止，或者产生错误的审计结果。2) 审计不透明导致审计过程中发生的错误无法检测到。例如，TPA 运行的数据选中算法可能有偏倚而使得一部分数据很少被审计到。这将使得一部分用户文件的完整性保护水平下降，损害用户利益。但是，由于审计协议以及审计过程并不对用户公开，用户无法发现已发生的问题。3) 审计结果无法验证导致错误的甚至是捏造的审计结果无法被发现。比如，为了减少执行验证的开销，TPA 有可能不执行验证而总是给出审计通过的结果。再如，TPA 可

能被外部攻击者操控或是与云服务商合谋而捏造审计结果。但是，用户无法判断 TPA 报告的审计结果是否可靠。鉴于上述问题，近些年提出的许多公开审计方案已经开始尝试更改信任假设，认为 TPA 是半可信或不可信的^[191-194]。

针对 TPA 不可信的问题，很多学者提出基于区块链技术的公开审计方案。根据审计模型的不同，这些方案大体可分为两类。第一类基于区块链构建针对 TPA 的监督机制，从而及时发现 TPA 的错误行为。例如，在 CPVPA^[18] 中，TPA 必须根据连续若干个区块的公开信息来生成挑战，以此来保证文件选择的随机性。另外，TPA 必须将审计信息公开在区块链上，因此用户不仅可以及时发现 TPA 的恶意行为，还可以发现不及时执行验证的拖延行为。然而，这类方案的审计过程仍旧是中心化的，并没有从根本上解决中心化审计内生的问题。第二类方法试图借助区块链技术构建去中心化的审计协议。例如，CAB^[17] 提出一种适用于多云存储架构的公开审计方案。审计者由所有云服务商和一部分资源充足的用户节点组成。在每轮区块链共识中，从所有审计者中随机选出一个记账节点验证云服务商提交的完整性证明。全部审计信息公开在区块链上，以便追溯和查验。

采用分布式审计模型可以很好地规避基于 TPA 的中心化审计的固有弱点。同时，向用户和云服务商公开所有审计相关的信息可以起到监督的作用，方便后续追溯和审计。然而，目前提出的分布式审计模型仍旧存在一些可改进之处。例如，已提出的方案并没有充分实现去中心性，只使用少量用户节点^[17] 或是云服务商^[15] 担任审计者，而其余大量用户并不能参与审计。而对于分布式审计，足够多的参与者才能保证足够的安全性和可信性。另外，目前提出的基于区块链的审计方案并没能很好地利用智能合约实现自动化且安全的审计过程。最后，目前提出的分布式审计方案缺乏适当的激励机制来促进积极行为、避免恶意行为。

作为总结，目前已提出的公开审计模型存在如下问题：

1. 基于 TPA 的中心化审计模型存在单点故障问题，且审计规则及过程不公开导致错误无法检测，更可能引发来自内部外部的恶意行为。
2. 基于区块链技术构建针对 TPA 的监督机制，虽然能一定程度避免 TPA 的错误行为，但仍旧无法解决中心化审计模型内生的安全问题。
3. 目前提出的基于区块链的分布式审计模型没有充分实现去中心化，没有很好地利用智能合约实现自动化审计，且由于缺乏激励机制无法促进积极行

为、避免恶意行为。

另一方面，基于区块链的公开审计还可能引发隐私泄露的风险。云服务商提交的回复中包含对多个被审计文件块的线性组合。由于全部审计记录都公开在区块链上，攻击者可以持续收集云服务商提交的完整性证明从而获取同一组文件块的不同线性组合，进而有可能还原出文件块的内容。因此，必须对云服务商提交的完整性证明进行隐私保护处理。在目前已经提出的隐私保护公开审计方案中^[99,104,117,118]，一般采用的方法是对完整性证明中的聚合文件块进行盲化处理，以达到零知识属性（“零知识”指能够证明所声称的结论但不会泄露相关隐私数据）。然而，这类传统方法并不能直接应用于审计区块链系统，因为这类方案普遍存在验证时间长且证明长度长的问题。如果想要构建基于区块链的隐私保护审计方案，必须满足两点要求：（1）验证时间快。验证操作由云存储用户执行或是在链上执行，那么其计算复杂度必须足够低。相对地，证明生成过程由云服务商执行，而云服务商具备较强计算能力，因此我们能够容忍证明生成操作的计算复杂度较高。（2）证明长度短。由于证明需要存储到链上，而链上的存储空间有限，因此我们希望证明长度足够短。

在本章中，我们提出基于区块链的分布式公开审计模型，弥补现有审计模型的不足。每隔固定时间从所有用户中随机选出一批审计者执行审计，从而在去中心性和高效性之间取得平衡；审计过程通过智能合约自动化执行，保证了审计规则以及全部审计记录在区块链上公开；设计激励机制以规避审计者和云服务商的恶意行为并促进其积极行为。基于所提出的分布式审计模型，我们设计了分布式公开审计系统 Audinet。该系统包含三个主要部分。1) 审计者选举。该算法从所有用户中随机选择一部分用户成为审计者。选举过程中，在云上存储了更多数据的用户有更大的概率被选中。2) 分布式审计协议。基于传统中心化公开审计协议 CPoR^[195]，我们提出了分布式审计协议 PoR* 以及具有零知识隐私保护性质的分布式审计协议 zk-PoR*。3) 智能合约自动执行的激励机制。该机制提供自动执行的奖励和惩罚，用以规范系统参与者的行。总的来看，Audinet 系统具有如下优势。

- 容错：即使一部分审计者出现故障或者被攻击者操控，审计仍旧能正常、可靠地执行。
- 透明：审计规则以及审计过程在区块链上进行公开，所有参与者都可以进行访问。

- 可追溯：任何参与者都可以通过查询链上数据追溯历史审计记录，从而实现监管。

本章的后续小节以如下方式组织：第二节提出了一个基于区块链的公开审计模型；第三节介绍了审计者选举的设计思想和执行流程，这是 Audinet 系统中十分重要的一个模块；第四节和第五节分别介绍了 PoR^{*} 协议和 zk-PoR^{*} 协议；第六节从理论层面分析了 Audinet 的激励机制如何能够促进参与者的积极行为；第七节通过在原型系统中的实验验证了 Audinet 审计的公平性、均匀性以及性能。第八节总结了本章的研究工作。

第二节 基于区块链的公开审计模型

在本小节，我们的目标是给出一种安全、可靠、高效的公开审计模型，并使得该模型适配于大部分现有的公开审计协议。在介绍正式内容之前，我们先来讨论一个问题：谁更适合作为审计者？一个天然的答案就是云存储用户。最早提出的审计方案普遍采用了由文件所有者作为审计者的模式，称为私有审计 [6,7,196]。但是私有审计采用的一对一的审计模式会引发两点问题。

- (1) 如果想要很好地保护文件完整性，那么审计必须是频繁进行的、能够均匀扫描并完整覆盖所有文件的。但云存储用户的资源持有情况参差不齐，不是每个用户都具备能力执行高质量的文件审计。那么，资源受限的用户便无法享受高水平的文件完整性保护。
- (2) 由用户单方面给出的审计结果可能会遭到云服务商否认。也就是说，私有审计结果并不具备不可抵赖性，无法实现纠纷定责。

这也就是为什么在公开审计中会引入“可信第三方审计者”这一角色。可信第三方的设置看似规避掉了私有审计存在的两个重大问题，然而实际上却引发了新的风险。在实际使用中，第三方审计者并不总是可信的。基于 TPA 的公开审计方案不仅无法抵御单点故障，且 TPA 还可能存在恶意行为 [18,197,198]。所以说，引入 TPA 并不能很好地解决原本的问题。

那么，是否存在更好的解决方案？首先我们认为，审计权仍应交到云存储用户手中，因为他们是最有意愿监督和审计云服务商行为、保护自己文件完整性不受破坏的群体。从文件保护角度，云存储用户有共同的目标，利益一致，因而可以合作。而对于上文提到的两点问题，我们可以采取不同于现有解决思路的全新方法。

1. 云存储用户资源能力参差不齐，怎么办？我们可以借鉴如 BitTorrent^[50] 等点对点（P2P, peer-to-peer）存储系统的思路。在 P2P 存储系统中，每个参与者既作为存储节点贡献本地存储空间，又作为存储服务使用者访问其它节点上的数据。资源充裕的节点可以多贡献（同时，这些节点享受到更多系统奖励），而资源不充裕的节点可以少贡献甚至不贡献。类似地，我们可以构建由所有云存储用户组成的 P2P 审计网络。在这个网络中，每个用户既作为审计者贡献计算资源，帮助其它用户执行审计，同时又作为审计服务使用者享受高质量的文件审计服务。资源充裕的节点可以多参与执行审计（同时获得更多奖励），而资源不充裕的节点可以少参与甚至不参与执行审计。
2. 审计如何实现不可抵赖性和纠纷定责？通过区块链实现不可抵赖性和纠纷定责已经是一个较为成熟的实践^[85, 127, 131]。例如，如果我们将所有审计记录公开在区块链上，那么整个审计过程便处于监督之下，可追溯可审计。再如，将重要审计功能迁移至链上执行，可以很好地规避审计者错误行为或恶意行为导致错误审计结果的问题。另外，以链上公开信息作为伪随机种子生成挑战信息，可以实现审计的不可预测性和不可操控性^[199]。

基于以上分析，本小节提出一种基于区块链的分布式公开审计模型，在无需第三方审计者的前提下实现公开审计的不可抵赖性、公开透明性、可追溯可审计性。

3.2.1 系统模型

如图3.1所示，审计过程中涉及 4 类角色，下面分别进行介绍。

智能合约。 智能合约提供了一个分布式的可信执行环境。我们将审计相关程序实现在智能合约中并向系统节点提供调用接口，这不仅使得特定程序能够在链上可靠执行，而且使得每一次接口调用的输入输出信息都被记录在区块链上作为存证。另外，智能合约还可以自动化执行激励机制，通过向用户和云服务商支付酬劳或者实施惩罚以促进系统参与者的积极行为、避免破坏性行为。

用户（审计者）。 每一个云存储用户都可以参与审计者选举并有机会成为审计者。审计者分为挑战者和验证者两种。其中，挑战者负责在一次审计中生成一个或者多个挑战；验证者负责对云服务商所提交的回复进行验证并促进最终审计结果的形成。审计者在完成既定审计任务后会获得酬劳。审计者选举定期执行，本轮的审计者到下一轮会重新恢复普通用户身份。（第三节详细介绍了审计者选举的过程。）

云服务商。在一次审计过程中，云服务商作为被审计者必须证明被选中数据的完整性。并且，只有在规定时间内成功证明数据完整性并通过审计后，云服务商才能获得相应的存储费用。这一创新性的支付模式称为“为存储可靠性付费”（pay for reliability），其可以激励云服务商采取有效的数据保护方案，并尽可能快地对审计进行响应。（针对云服务商的激励机制分析在第六节给出。）

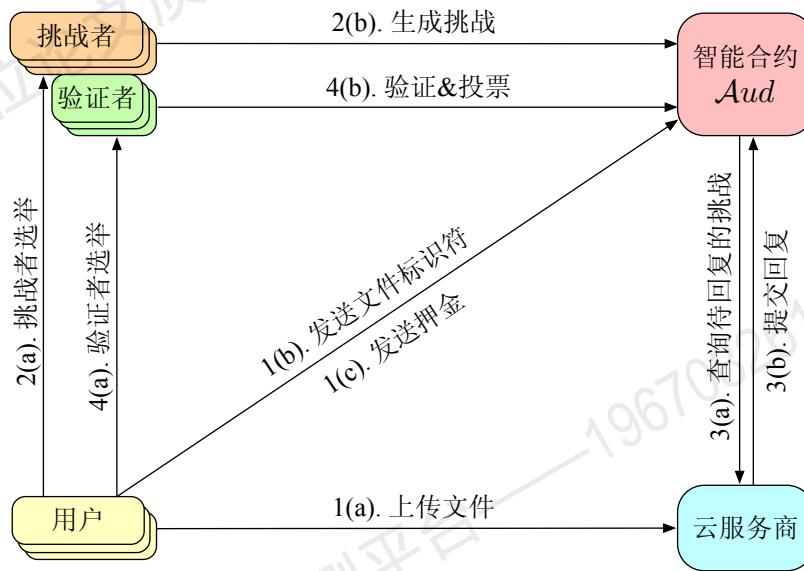


图 3.1 基于区块链的分布式公开审计模型。

一次审计过程包含四个阶段，执行流程如下。

1. 准备阶段。首次加入系统的用户需要执行初始化，生成用于文件上传和审计者选举的系统参数和密钥对。之后，用户可以上传文件到云端，这使得用户拥有了“财富”因而有资格参与审计者选举。每上传一个文件，用户需要将文件标识符等参数发送给智能合约 Aud 。用户还需发送一些押金给智能合约，用于支付后续可能发生的审计费用。
2. 挑战阶段。若某用户成功通过审计者选举成为挑战者，则可以调用合约对应接口以生成新的挑战。挑战生成后，智能合约发放挑战者酬劳，方法是将文件所有者的一部分押金转发给挑战者。
3. 回复阶段。云服务商需要调用合约以获取一个待回复的挑战。根据该挑战，云服务商检索相应数据并计算完整性证明，亦即针对该挑战的“回复”。云服务商调用智能合约接口以提交该回复。
4. 验证阶段。验证方式分为两种。第一种是由用户执行验证。若某用户成功

通过审计者选举成为验证者，则可以调用合约并获取到一个待验证的回复。验证者本地执行验证，并将验证结果发送给合约进行投票。验证者完成投票后，智能合约发放验证者酬劳，方法是将文件所有者的一部分押金转发给验证者。在验证阶段将有许多验证者参与投票，这些投票决定了最终的审计结果：若规定时间内统计的支持票数超过某一阈值则云服务商通过本次审计，否则云服务商未通过本次审计。第二种方式是由智能合约自动化执行验证。若审计通过，智能合约将向云服务商发放酬劳，方法是将文件所有者的一部分押金转发给云服务商。

从审计系统实施的角度，我们需要确定审计网络和区块链网络的关系。参考现有方案的做法，我们有两个选择。一是使用外部区块链网络，即区块链由外部节点维护而用户和云服务商作为区块链轻节点使用区块链服务^[16,18]。另一种选择是将用户或云服务商部署为区块链全节点^[17,21,138]。在本章中，为了使后续描述更为简洁，并突出本章重点（即如何执行分布式审计），我们暂且使用第一种实施方法。在第四章中我们将会具体讨论第二种实施方法。

3.2.2 安全模型

我们假设云服务商是不可信的，其倾向于在未按要求存储数据的情况下成功通过审计（从而获取报酬）。云上存储的数据可能会因为意外故障、外部攻击或者是某些恶意行为而遭到篡改或者删除。在这种情况下，云服务商为了仍旧能够通过审计可能会采取一些恶意行为，包括：1) 猜测哪些数据或文件将会被审计，从而提前进行准备；2) 操控挑战生成过程，从而避免有问题的文件被审计；3) 故意不对某些已经生成的挑战进行回复（可能是因为该文件已经损坏、被删除等等）；4) 在系统中部署恶意用户或者操控一部分用户，从而干扰最终的审计结果。我们假设在任意时刻云服务商可以操控的用户比例不超过 50%。

假设审计者是不可靠的。审计者可能会发生崩溃、故障、运行失误，可能会被云服务商控制等等。我们假设在任意时刻系统内所有审计者中发生故障的比例不超过 50%。假设所有的用户都是理性的，即他们想要最大化自身获得的利益或者最小化自身损失。另外，我们不考虑针对区块链网络的攻击，并假设区块链网络是安全的、可靠的。

3.2.3 设计目标

结合安全模型中给出的云服务商可能进行的四类攻击，我们认为一个安全可靠的分布式审计系统必须满足如下设计要求。

在挑战阶段，分布式审计系统需要抵御云服务商的第1、2类攻击，因此必须保证：（1）挑战频次足够高且保持稳定。假如审计频次太低，那么有可能无法及时发现已经出现的错误，更无法及时采取数据恢复措施。因此，必须设计适当的激励机制，使得有足够多的用户积极参与审计。另外，假如审计频次忽上忽下，那么云服务商无法分配一个合适的资源量去计算回复，不仅给云服务商造成困难，而且影响云服务商的回复速度。因此，必须设计合理的审计者选举机制，使得参与生成挑战的用户数量保持稳定，从而保证生成挑战的频次保持稳定。（2）挑战生成算法具有良好的随机性。在挑战阶段，所有文件必须均匀被选中，否则对用户来说是不公平的，同时也可能导致已经发生的文件错误无法及时被检测到。另外，必须保证挑战的内容无法预测且无法操控，以避免云服务商提前猜中即将被审计的文件而事先准备，或是云服务商试图恶意操控挑战生成结果。

在回复阶段，分布式审计系统需要抵御云服务商的第3类攻击，因此必须保证云服务商提交回复的时延在一个合理范围内。如果云服务商回复的过慢，我们完全有理由怀疑云服务商进行了数据外包^[138,200]，或者被选中的数据已经遭到了损坏或者删除。因此，必须要求云服务商在规定时间内提交回复。假如云服务商提交回复的时延超过某一上限，应对其施以经济上的惩罚措施。

在验证阶段，分布式审计系统需要抵御云服务商的第4类攻击，因此必须保证：（1）验证者足够多。验证者越多则审计结果越可靠。因此，必须设计适当的激励机制以使得有足够多用户积极参与验证。（2）验证者中有足够多诚实用户，这是为了防止云服务商在系统内布置大量恶意节点从而操控审计结果。因此，审计者选举必须能够保证可信度更高的用户有更大概率被选中，且任何人无法提前预测出选举结果。（3）验证时延低。若验证阶段耗费时间过长，则数据错误无法被及时发现，以致于无法及时执行数据修复措施。

将以上设计目标进行归纳，我们得出审计系统必须满足的要求如下。后文中，我们将会分析 Audinet 系统如何通过审计者选举机制、分布式审计协议以及激励机制很好地满足了下述要求。

1. 审计者选举保证审计频次稳定，保证审计者中有足够多的诚实用户，且审

计者身份无法提前预测。

2. 挑战阶段能够随机地选中文件，选中过程满足均匀性、不可预测性、不可操控性。
3. 回复阶段和验证阶段的时延被控制在一个合理且安全的范围内。
4. 激励机制保证系统中有足够多用户参与审计，并促使云服务商尽快提交回复。
5. 审计相关的所有信息向所有用户公开。

第三节 审计者选举

审计者选举的作用是从所有用户中选择一个随机子集，使得被选中的用户作为审计者参与到一轮分布式审计中。之所以在分布式审计系统中加入审计者选举机制，是出于如下考量。

- 如果不加限制地允许所有用户自由参与审计，可能会导致某些时间段挑战生成频次过高。假设所有用户都可以调用合约以生成新的挑战，则挑战生成可被视为一个随机事件。由于云存储服务本身存在使用的波峰和波谷，可以预料，挑战生成的频次也会存在波峰和波谷。如前文所述，这将导致云服务商无法分配合适的资源用于计算完整性证明，影响回复时延。设计审计者选举机制正是为了保证稳定的审计频次。
- 为了抵御审计者可能出现的各种故障，以及云服务商的恶意攻击，我们必须保证审计者中有足够多的诚实用户。为了保证这点，我们采取了两个对策。(1) 我们认为，一个用户在云上存储的文件越多便越有可能维护系统的稳定和安全。因此，审计者选举的设计理念是，一个用户上传到云上的文件越多则被选为审计者的概率应该越大。(2) 我们设计了针对审计者身份验证机制。合约在接收到来自审计者的接口调用请求后，会首先执行身份验证。这是为了避免云服务商部署大量节点假扮审计者从而操控审计结果。

在本节中，我们先介绍如何执行审计者选举以及如何验证审计者身份。图3.2给出了工作流程示意图。然后，我们分析审计者选举的设计如何保障审计的安全性、可靠性和可操作性。

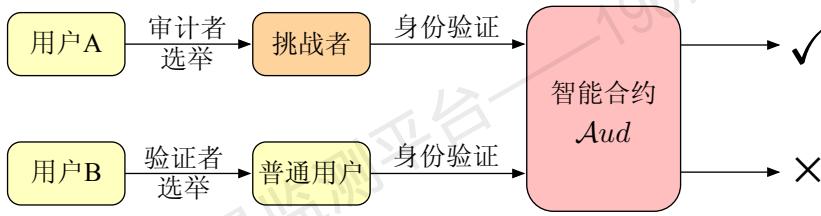


图 3.2 审计者选举的工作流程。

3.3.1 选举

定义一个区块链纪元（epoch）为上一个区块生成结束到下一个新区块生成的这段时间。在每个 epoch 内，每个用户有一次机会参与审计者选举，且过程中无需与其它用户进行交互。下面给出选举挑战者的详细流程。

第 1 步：计算随机值。用户根据可验证随机方程^[201]（VRF, verifiable random function）计算一个哈希值 hash：

$$\langle \text{hash}, \pi \rangle \leftarrow \text{VRF}_{sk}(\text{epoch} || \text{role} || \text{seed}), \quad (3.1)$$

其中 sk 是用户的 VRF 私钥， epoch 是当前 epoch 的序号， role 等于 0（在验证者选举中 $\text{role} = 1$ ）， seed 是最新一个区块的 blockID， π 是用于验证 hash 的证明。

第 2 步：计算每单位财富值被选中的概率。假设所有文件在上传到云端之前都被切分成若干个等大的数据块。那么，假如一个用户总共上传了 b 个文件块，我们说该用户在系统内拥有 $w = b/\theta$ 单位的财富，或者说该用户的财富值是 w 。（ θ 是一个预定义的系统常量。）系统内总共的财富值记为 B 。在每个 epoch 内，我们想要选中 α 单位的财富，那么每单位财富被选中的概率是 $p_1 = \alpha/B$ 。由于每单位财富值被选中的概率相等，那么拥有财富值多的用户被选中的概率越大。具体地，对于一个拥有 w 财富值的用户，其在审计者选举中有 k 单位财富被选中的概率等于

$$P(k) = \binom{w}{k} p_1^k (1 - p_1)^{w-k}$$

第 3 步：计算选举结果。用户首先计算 $w+1$ 个数值区间，如下：

$$I_i = \begin{cases} [0, P(0)], & i = 0, \\ \left[\sum_{k=0}^{i-1} P(k), \sum_{k=0}^i P(k) \right), & i = 1, \dots, w. \end{cases} \quad (3.2)$$

$$(3.3)$$

然后，用户求得满足条件 $\text{hash} \in I_j$ 的 j 值，即为挑战者选举的结果。若求得的 j 值为 0，则该用户未被选中。假如求得 $j > 0$ ，则该用户有 j 单位财富被选中，

或者说该用户在本轮选举中被选中 j 次。这意味着该用户有权在当前 epoch 内生成 j 个挑战。

选举验证者的过程与选举挑战者大体一致，不同之处在于每单位财富被选中的概率等于 $p_2 = \alpha\beta/B$ ，这里 β 是验证一个回复平均所需要的验证者数量。在一轮验证者选举中被选中 $j > 0$ 次的用户有权验证 j 个回复。

3.3.2 身份验证

针对审计者身份的验证由智能合约执行，用以检查某个用户是否有权调用某个与审计相关的接口。下面介绍身份验证的工作流程。

第 1 步：验证 VRF 输出。当某个用户想要调用与审计相关的合约接口时，该用户必须提供审计者选举第 1 步得到的 VRF 输出 $\langle hash, \pi \rangle$ ，以证明自己是一个有效的审计者。合约运行函数 VerifyVRF() 来验证用户提交的 VRF 输出。该函数使用 π 和用户的 VRF 公钥 pk 来验证 $hash$ ：

$$0/1 \leftarrow \text{VerifyVRF}_{pk}(hash, \pi, epoch || role || seed),$$

其中 $epoch$ 和 $seed$ 是从区块链上读取到的参数， $role$ 的值是 0（适用于挑战者身份验证）或者 1（适用于验证者身份验证）。如果 VerifyVRF() 输出 0 则验证失败，若 VerifyVRF() 输出 1 则验证成功。

第 2 步：验证 j 的值。当 VerifyVRF() 函数输出 1 时，合约继续验证选举结果，即验证 j 的值。首先，根据表达式 (3.2) 和 (3.3) 计算 $w+1$ 个数值区间 I_0, \dots, I_w 。然后，计算满足 $hash \in I_j$ 的 j 值。若算得的 j 值大于 0 则身份验证成功，若算得的 j 值等于 0 则身份验证失败。

3.3.3 分析

审计者选举能够保证选举结果无法预测且无法操控。首先，审计者选举的结果是无法预测的，这是因为选举算法的输入包含区块哈希值，而这一哈希值是无法预测的。再者，审计者选举的结果是无法操控的，原因有两点：(1) 选举算法的输入是链上公开信息、是确定性的，因此用户无法通过捏造算法输入来提高选中概率；(2) 即使用户通过捏造算法输入伪造了身份，由于有身份验证机制，伪造身份的用户会被发现而其请求会被拒绝。

挑战者选举保证了在挑战阶段有稳定的挑战生成频率。在挑战者选举过程中，一个单位的财富被选中的概率是 $p_1 = \alpha/B$ 。理想情况下，在每个 epoch 内都有大约 α 个财富值被选中。Audinet 系统规定，当一个用户在挑战者选举中有

j 单位财富被选中时，该用户可以发起 j 个挑战。因此，在每个 epoch 内会平均生成 α 个挑战。通过设置一个合理的 α 值，同时配合激励机制以使得有足够的用户参与审计者选举，我们可以控制系统内的挑战生成频次并使其保持在预期值。另外，由于审计者选举的身份验证机制，只有合法的挑战者才有权利调用合约接口生成新的挑战，攻击者无法假扮成挑战者去生成挑战。

验证者选举保证了在验证阶段有足够的诚实验证者。所谓诚实验证者，指他们有意愿维护 Audinet 系统的安全可靠、不会从事恶意行为，同时节点未发生意外故障。在验证阶段，云服务商可能会试图控制一部分用户从而操控审计结果。但是这样的攻击行为是不可能成功的，原因如下。首先，云服务商无法通过事先操控大量用户节点来操控审计结果，这是因为用户无法事先知道自己是否能在下一次审计者选举中被选中。只有在新一轮 epoch 开始之后，用户才可以运行审计者选举算法并确认自己是否被选中为验证者。第二点，云服务商即使事先部署大量节点参与审计者选举，也很难保证有足够的恶意节点成功被选为验证者。这是因为，一个用户被选中的概率正比于该用户在云上存储的文件数量。一个用户想要增加自己被选中的概率，那么必须要上传大量文件到云上，而这些文件会产生不小的审计费用，大大增加了云服务商的攻击成本。总的来说，这种方法成本高昂但回报低微。

第四节 PoR^{*} 协议设计

在本节中，我们介绍分布式公开审计协议 PoR^{*}，该协议是3.2.1小节介绍的分布式审计模型的一个实例，同时也可看作是基于 TPA 的审计协议 CPoR^[195] 的分布式版本。表格3.1给出了协议运行过程中使用到的合约接口。

表 3.1 智能合约 *Aud* 提供的调用接口。

接口	调用者	功能描述
Upload()	用户	向智能合约发送一个新上传到云上的文件的相关信息
GenChall()	挑战者	生成一个挑战
RtrChall()	云服务商	查询当前待回复的挑战
SbmResp()	云服务商	提交一个回复
RtrResp()	验证者	查询当前待验证的恢复
Vote()	验证者	对某个回复进行投票

3.4.1 准备阶段

参数和密钥生成。给定安全参数 $\lambda = 80$, 每个用户需要初始化与 BLS 签名相关的四个参数, 如下。

- 大素数 p , 长度为 2λ 位;
- 群 G 的生成元 g , 其中群 G 的支撑集为 \mathbb{Z}_p ;
- 双线性映射 $e := G \times G \rightarrow G_T$;
- BLS 哈希函数 $H := \{0,1\}^* \rightarrow G$ 。

除此之外, 还需要初始化 BLS 密钥对 (x,y) , 用以进行文件上传时的签名操作。生成方法如下。

- 生成私钥 $x \xleftarrow{R} \mathbb{Z}_p$, 用于对数据块进行签名;
- 生成公钥 $y \leftarrow g^x$, 用于进行签名验证。

对于想要成为审计者的用户, 还需要初始化 VRF 密钥对 (sk,pk) , 其中私钥 sk 用于生成随机哈希值而公钥 pk 用于验证该随机值。

文件上传。当向云端上传文件 F 时, 用户首先使用纠删码对文件进行编码并得到 n 个文件块 $F_1, F_2, \dots, F_n \in \mathbb{Z}_p$ 。用户从群 G 中随机选择一个生成元 u 。对于每个文件块 F_i (其中 $i \in \{1, \dots, n\}$), 用户使用生成元 u 、BLS 私钥 x 和 BLS 哈希函数 $H()$ 得到签名:

$$\sigma_i = (H(i) \cdot u^{F_i})^x.$$

最后, 用户定义文件标识符 $I^F = (\text{filename}, \text{username})$, 并将文件标识符 I^F 、文件块及其签名 $\{F_i, \sigma_i\}_{i \in \{1, \dots, n\}}$ 一并上传给云服务商。同时, 用户调用合约函数 $\text{Upload}()$, 将文件标识符 I^F 、文件块个数 n 和生成元 u 上传给智能合约 $\mathcal{A}ud$ 。

3.4.2 挑战阶段

身份验证。当有用户调用合约函数 $\text{GenChall}()$ 时, 合约首先运行身份验证来确认该用户是一名合法的挑战者。如果身份验证成功, 且该挑战者的选举结果为 j , 那么合约将运行函数 $\text{GenChall}()$ 并循环 j 次, 从而生成 j 个挑战。

生成挑战。算法3.1给出了函数 $\text{GenChall}()$ 的执行过程。首先, 伪随机函数生成一个随机的文件索引 f , 式子中 $time$ 指函数 $\text{GenChall}()$ 被调用时的时间, $addr$ 指调用者的区块链地址, $nonce$ 是一个计数变量 (每次函数 $\text{GenChall}()$ 被调用则 $nonce$ 的值加 1), N 是云上存储的总文件个数。通过文件索引 f 便可得知被选中的文件。对于被选中的文件 $F = \{F_1, \dots, F_n\}$, 随机选择其中 m 个文件块

算法 3.1 GenChall($time, addr, nonce, N, w_c$)

```

1  $f \leftarrow \text{keccak256}(time, addr, nonce) \bmod N$ 
2  $nonce \leftarrow nonce + 1$ 
3  $F \leftarrow \text{FindFile}(f)$ 
4  $u_1, \dots, u_m \xleftarrow{R} \{1, \dots, n\}$ 
5  $v_1, \dots, v_m \xleftarrow{R} \mathbb{Z}_p$ 
6  $\text{SendPayment}(addr, w_c)$ 
7  $\text{AppendList}(C = (I^F, \{u_i, v_i\}_{\forall i \in \{1, \dots, m\}}))$ 

```

F_{u_1}, \dots, F_{u_m} 。再从集合 \mathbb{Z}_p 中随机选择 m 个值 v_1, \dots, v_m , 作为被选中文件块的权重值。至此, 生成了新的挑战 $C = (I^F, \{u_i, v_i\}_{\forall i \in \{1, \dots, m\}})$ 。最后, 将 w_c 个区块链代币发放给挑战者作为酬劳。在这一阶段新生成的挑战会被追加到合约内所维护的审计表中, 向所有用户公开。如表格3.2(a)所示, 新生成的挑战 C_3 被添加到审计表中。

3.4.3 回复阶段

云服务商调用合约函数 RtrChall() 来查询当前待回复的挑战。当接收到合约返回的挑战 $C = (I^F, \{u_i, v_i\}_{\forall i \in \{1, \dots, l\}})$ 后, 云服务商检索对应的文件块以及 BLS 签名 $\{F_{u_i}, \sigma_{u_i}\}_{\forall i \in \{1, \dots, m\}}$, 并计算回复 $R = (\mu, \sigma)$:

$$\begin{aligned}\mu &= \sum_{(u_i, v_i) \in C} v_i \cdot F_{u_i}, \\ \sigma &= \prod_{(u_i, v_i) \in C} \sigma_{u_i}^{v_i}.\end{aligned}$$

云服务商调用合约函数 SbmResp() 提交计算出的回复 R 。如表格3.2(b)所示, 合约将挑战 C_3 返回给云服务商后, 云服务商计算回复 R_3 并提交给合约, 然后合约将回复 R_3 增加到审计表中。

3.4.4 验证阶段

对于云服务商提交给合约的每个回复, 平均有 β 个验证者对其进行验证。每个验证者本地运行验证算法, 并根据算法输出的结果进行投票。合约对投票进行统计而得出最终的审计结果。在表格3.2(b)给出的例子中, 回复 R_1 总共需要 50 个验证者参与投票, 而目前已已有 20 个验证者投票通过。

身份验证。与挑战阶段类似, 当用户调用合约函数 RtrResp() 时, 合约首先运行身份验证来确认该用户是一名合法的验证者。如果身份验证成功, 且该验

表 3.2 在审计过程中，审计表内容的变化情况。

(a) 新生成的挑战 C_3 被追加到审计表中。

挑战	回复	投票
C_1	R_1	20/50
C_2	R_2	null
C_3	null	null

(b) 云服务商提交针对挑战 C_3 的回复 R_3 。

挑战	回复	投票
C_1	R_1	20/50
C_2	R_2	null
C_3	R_3	null

(c) 针对回复 R_1 的验证阶段结束，相应条目被删除。

挑战	回复	投票
C_2	R_2	null
C_3	R_3	null

证者的选举结果为 j ，那么合约将分配 j 个待验证回复给该验证者。

验证并投票。验证者接收到回复 $R = (\mu, \sigma)$ 及相应的挑战 $C = (I^F, \{u_i, v_i\}_{i \in \{1, \dots, l\}})$ 后，验证是否满足如下条件：

$$e(\sigma, g) = e\left(\prod_{(u_i, v_i) \in C} H(i)^{v_i} \cdot u^\mu, y\right), \quad (3.4)$$

上式中 u 是该文件上传时的生成元， y 是数据所有者的 BLS 公钥（用以验证 BLS 签名）。如果表达式 3.4 满足，那么验证者将投票结果 $V = 1$ 发送给合约，意味着验证通过；反之，验证者将投票结果 $V = 0$ 发送给合约，意味着验证不通过。接收到投票结果后，合约将 w_v 个区块链代币发放给验证者作为酬劳。验证阶段结束，则审计表中相应的条目会被删除，如表格 3.2(c) 所示。

统计投票。合约负责统计验证者的投票。假设对于每个回复 R 需要有 β 个验证者进行投票。合约根据每个验证者的投票结果 V_i 统计总票数 $\hat{V} = \sum V_i$ 。假如在达到限定时间之前总票数 \hat{V} 超过 $\beta/2$ ，那么对于回复 R 的验证通过，本轮审计成功，云服务商可以获取到相应的报酬。假如在达到限定时间时候总票数 \hat{V} 仍旧没达到 $\beta/2$ ，那么本轮审计失败，云服务商无法获取到相应的报酬。

3.4.5 分析

在挑战阶段，云服务商可能进行的恶意行为包括：(1) 提前猜测哪个文件会被选中，(2) 试图操控挑战中被选中的文件。不过，在 Audinet 系统中这两类行为是不会成功的，原因如下。一方面，如 3.3.3 小节所述，云服务商无法事先得知哪些用户会被选为挑战者，也无法操控挑战者选举的结果。另一方面，挑战生成算法的输入是有效挑战者的地址以及算法调用的时间，云服务商只有提前得知这两个变量的值或者操控这两个变量的值才有可能预测或者操控挑战内容。因此我们得出结论，Audinet 系统能够保证挑战内容无法预测也无法操控。

在验证阶段，采用链上链下结合计算的目的是提高效率。合约的运行效率

比较低，直接在链上执行验证算法会导致验证阶段时间开销过大^[85]。一个比较好的办法是将验证算法迁移至链下执行^[202]。在 PoR^{*} 协议中，我们采用了先在链下执行验证再在链上统计验证结果的方法，可以大大提升验证阶段的效率。在验证阶段，只要多于一半的验证者是诚实则审计结果是可靠的。如3.3.3小节所述，审计者选举保证了系统内会有足够多的诚实验证者。因此，PoR^{*} 可以抵御云服务商部署大量恶意节点以控制验证结果的攻击。

在审计过程中，每一步都需要用户或者云服务商与智能合约交互，因此每一步审计信息都记录在链上，实现了可追溯、可监督、可审计。用户可以查询自己的文件的历史审计记录，或者重新验证某一个审计结果的有效性。外部审计者可以通过追溯链上信息执行第三方审计。

第五节 zk-PoR^{*} 协议设计

考虑到云存储用户之间可能存在利益关系，云存储系统必须保证不同用户的数据彼此隔离，而把所有完整性证明以明文形式公开在区块链上可能会违背这一原则。攻击者可以持续收集云服务商提交的完整性证明从而获取同一组文件块的不同线性组合，进而有可能还原出文件块的内容。因此，我们提出具有零知识性隐私保护性质的审计协议 zk-PoR^{*}，该协议能够保证在审计过程中云服务商所提交的证明不会泄露关于被审计数据的任何信息。

3.5.1 背景知识简介

Groth16 算法。在 zk-PoR^{*} 审计协议中，我们使用 Groth16 算法^[90] 来生成被选中文件块的零知识证明，以确保该证明不会泄露关于原文件块的任何信息。在这里，我们简要介绍关于 Groth16 的基本概念和原理。

- 线性表达式：指变量之间只进行加法操作的表达式。在下面的例子中，表达式3.5和3.6是线性表达式而表达式3.7不是。

$$x + x + 2y, \quad (3.5)$$

$$2x + 3y, \quad (3.6)$$

$$x^2 + 3y, \quad (3.7)$$

- 二次表达式：指两个线性表达式相乘或相加得到的表达式。若 A, B, C 都是线性表达式，则二次表达式形如 $A * B + C$ 。表达式3.8给出了一个示例。

$$(3x + 5y + 7) * (2x + 3y) + (x + 2y - 1) \quad (3.8)$$

- 算术电路 (*Arithmetic Circuit*): 给定一个待证明的问题, 或者一个程序, 我们可以将其转化为一个算术电路。一个 \mathbb{F}_p 算术电路由一组电线 (wires) 和加法/乘法门 (gates) 构成, 其中电线连接起不同的门并传递信号 (signals), 信号的值为域 \mathbb{F}_p 的元素。
- 约束 (*constraints*): 给定一个算术电路, 我们可以将其描述为一组约束。所有的约束必须使用二次表达式 $A * B + C$ 的形式表达, 其中 A, B, C 是信号的线性表达式。描述某个算术电路的约束集合称为秩 1 约束系统 (R1CS, rank-1 constraint system)。
- 见证者 (*witness*): 在进行零知识证明的过程中, 证明者需要证明自己知道使得 R1CS 中所有表达式成立的一组解, 或者说证明自己知道能够使算术电路输出正确结果的一组信号值。而这组信号值称为见证者。

针对一个待证明问题, 我们将其转化为一个或多个算术电路。证明过程包含三个步骤:

- $(pk, vk) \leftarrow Setup(1^\lambda, C)$: 给定算术电路 C 及安全参数 λ , 执行可信设置生成证明密钥 pk 及验证密钥 vk (二者均公开);
- $\pi \leftarrow Prove(pk, \vec{s}, \vec{w})$: 证明者使用证明密钥 pk 、见证者 \vec{w} (秘密) 及公开声明 \vec{s} 生成零知识证明 π ;
- $0/1 \leftarrow Verify(vk, \vec{s}, \pi)$: 验证者使用验证密钥 vk 、零知识证明 π 及公开声明 \vec{s} 验证零知识证明; 输出 1 表明验证成功。

由上述过程可以看出, 验证者在验证过程中不会获取到任何非公开信息。图3.3展示了使用 circom^[203] 编译算术电路并生成零知识证明的过程。

默克尔树。默克尔树 (merkle tree) 是一种树状结构的哈希表^[204]。如图3.4所示, 默克尔树中叶子节点为数据块的哈希值, 由叶子结点向上每一个节点都是下一层两个哈希值拼接而成的字符串的哈希值。由此层层向上计算得到一个树根节点, 称为默克尔根 (merkle root)。在区块链系统中, 默克尔树的一个主要应用是进行数据完整性验证。假设证明者想要证明自己持有默克尔树中某个叶子节点所对应的原数据块, 且假设默克尔树根为公开值。那么, 证明者只需提供该叶子节点及其对应的默克尔路径 (merkle path) 即可。图3.4给出了一个示例。假设证明者想要证明自己持有数据块 3, 则证明者需要提供块 3 及其默克尔路径 h_3 、 h_{01} 。在验证时, 验证者利用块 3 计算出 h_2 , 利用 h_2 和 h_3 计算出 h_{23} , 再利用 h_{01} 和 h_{23} 计算出默克尔根 root。验证者将计算出的 root 与公开的默克尔树根

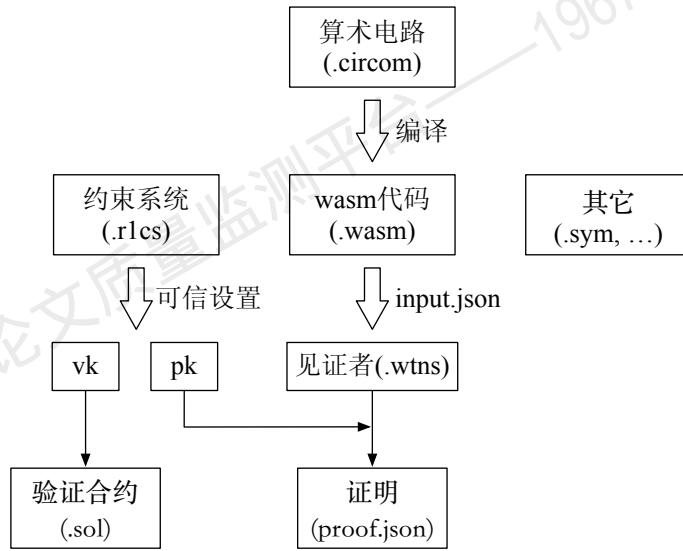


图 3.3 使用 circom 语言编写的算术电路经过编译、可信设置及计算 witness 生成零知识证明，并可以使用智能合约进行验证。

对比，若一致则说明证明者提供的块 3 是正确的。不过，上述验证过程中会暴露原数据块的信息，因此无法实现对数据隐私的保护。使用 zk-SNARKs 框架计算默克尔证明可以很好地解决隐私暴露的问题。

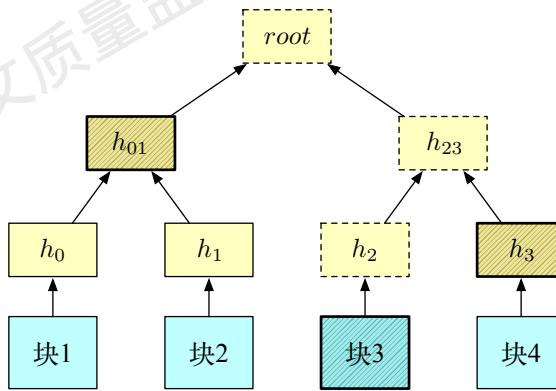


图 3.4 为了验证块 3 的完整性，验证者使用默克尔路径 h_3 、 h_{01} 层层向上计算出默克尔根，并对比计算出的树根与公开值是否一致。

3.5.2 zk-PoR^{*} 工作流程

zk-PoR^{*} 协议包含四个步骤，其中每一步都需要用户或云服务商与智能合约交互。表格3.3给出了协议运行过程中用到的合约接口。

表 3.3 智能合约 *Aud* 提供的调用接口。

接口	调用者	功能描述
zkUpload()	用户	向智能合约发送一个新上传到云上的文件的相关信息
zkGenChall()	挑战者	生成一个挑战
zkRtrChall()	云服务商	查询当前待回复的挑战
zkSbmResp()	云服务商	提交一个回复
zkVerification()	合约	自动化验证云服务商所提交的零知识证明

可信设置及文件上传。在执行审计前需要先构建用于生成零知识证明的算术电路，并根据该电路执行可信设置。可信设置会生成证明密钥 pk 和验证密钥 vk ，二者均为公开信息。用户在上传文件 F 前需要进行预处理。首先，将文件切分成 n 个文件块 F_1, F_2, \dots, F_n ；然后，将文件块哈希值作为叶子结点构建默克尔树；最后，对每个文件块进行签名得到 n 个签名 s_1, s_2, \dots, s_n 。文件预处理完成后，用户将文件块及其签名 $\{F_i, s_i\}_{\forall i \in \{1, \dots, n\}}$ 以及默克尔根 R_F 发送给云服务商，并将文件标识符 $I^F = (filename, username)$ 和默克尔根 R_F 发送给智能合约。云服务商对接收到的文件块执行签名验证无误后，根据文件块构建默克尔树并确认所得默克尔树根是否与接收到的一致。若确认无误，云服务商将文件块及默克尔树保存在云服务器上。

挑战阶段。挑战阶段的工作流程与 PoR^{*} 中类似，唯一不同之处在于 zkGenChall() 在选中 m 个文件块 F_{u_1}, \dots, F_{u_m} 后并不生成权重值。因此，zkGenChall() 生成的挑战内容为 $C^{zk} = (I^F, \{u_i\}_{\forall i \in \{1, \dots, m\}})$ ，其中 u_1, u_2, \dots, u_m 为文件 F 的 m 个文件块的索引值。

回复阶段。云服务商调用合约函数 zkRtrChall() 来查询当前待回复的挑战。当接收到合约返回的挑战 $C^{zk} = (I^F, \{u_i\}_{\forall i \in \{1, \dots, m\}})$ 后，云服务商检索文件 F 被选中的 m 个文件块以及文件 F 对应的默克尔树。对于每个文件块 F_{u_i} ，云服务商根据文件块内容及默克尔路径生成零知识证明 π_i 。所使用的算术电路如算法3.2所示。云服务商生成回复 $R^{zk} = \{\pi_i\}_{\forall i \in \{1, \dots, m\}}$ 并调用合约接口 zkSbmResp() 将该回复提交给智能合约。

验证阶段。一旦智能合约接收到云服务商提交的回复，便会调用合约接口 zkVerification() 对回复中包含的 m 个零知识证明依次执行验证。如果 m 个证明

算法 3.2 Merkle Proof Circuit**Input:** $leaf, path[], levels$ **Output:** $root$

```

1  $hash = []$ 
2  $hash[0] = \text{Hash}(leaf)$ 
3 for  $0 \leq i < levels - 1$ 
4   if  $path[i].position == left$ 
5      $hash[i + 1] = \text{Hash}(path(i) || hash(i))$ 
6   else
7      $hash[i + 1] = \text{Hash}(hash(i) || path(i))$ 
8  $root = hash[levels - 1]$ 

```

都验证通过，则本次审计成功。本阶段为自动化执行，不需要用户参与。

3.5.3 分析

$zk\text{-PoR}^*$ 协议基于 Groth16 算法实现了零知识隐私保护性质。在审计过程中将文件生成的默克尔树根作为公开信息发布在区块链上，其余信息隐藏。验证过程不再需要多个审计者参与，而是由智能合约执行验证函数并得出审计结果，这使得 $zk\text{-PoR}^*$ 的验证更为高效。验证时延基本上取决于对应交易被打包进新区块的等待时间。不过，由于零知识证明算术电路计算复杂度较高，目前阶段 $zk\text{-PoR}^*$ 协议只能支持基于哈希和签名的审计算法，而不能运行 PoR^* 协议中的基于双线性对的审计算法。因此我们建议对隐私保护需求较高的文件使用 $zk\text{-PoR}^*$ 协议，而对其它文件使用 PoR^* 协议。

第六节 激励机制

我们假设 Audinet 系统中所有参与者都是理性经济人，即它们寻求收益最大化，或者说成本最小化。我们分析如何通过在系统中给予适当的激励来规范用户和云服务商的行为。表格3.4给出了本小节的分析中所使用到的系统常量参数。

3.6.1 针对用户的激励机制

我们将用户分成三类：

- (a) 不活跃用户，指很少上线或很少参与审计的用户；
- (b) 贪婪用户，指过于活跃地参与审计者选举以求获得更多审计者报酬的用户；
- (c) 健康用户，指积极（与“不活跃”相对）且有节制（与“贪婪”相对）地参与审计者选举和审计工作的用户。

第三章 隐私友好的分布式公开审计方案

表 3.4 一些系统常数的符号表示和描述。

符号	描述
p_c	一个用户在挑战者选举中获胜的概率
p_v	一个用户在验证者选举中获胜的概率
w_c	每次审计支付给挑战者的报酬
w_v	每次审计支付给验证者的报酬
p_1	一单位财富在挑战者选举中获胜的概率
p_2	一单位财富在验证者选举中获胜的概率
\bar{c}	每个文件块每个月支付给云服务商的存储费用的期望
\bar{a}	每个文件块每个月支付给审计者的审计费用的期望
\hat{c}	每个文件块在每次审计中支付的存储费用的最大值
$\hat{\tau}$	云服务商提交回复的时延上限

在本小节中，我们分析不同的用户行为所带来的经济收益情况。首先，我们会计算一般情况下每个用户每个月能够获得的报酬的期望（或称“平均报酬”），并计算出用户每个月的净花费。然后，我们会分析不活跃用户和贪婪用户各自的月净花费，并论证这两类用户的花费可能会比健康用户的更高。那么，作为理性经纪人，用户会倾向于成为健康用户，而不会选择成为不活跃用户或贪婪用户。（以下分析的报酬和花费均指区块链代币的数额。）

一般用户平均月开销。 经过一次审计者选举（包含一次挑战者选举和一次验证者选举），一个拥有 w 财富值的用户所能获得的报酬的期望是

$$\begin{aligned} E[\text{rewards per round}] &= p_c \cdot w_c + p_v \cdot w_v \\ &= w p_1 \cdot w_c + w p_2 \cdot w_v \\ &= w(p_1 w_c + p_2 w_v) \end{aligned}$$

假设一个用户在一个月之内参与了 ℓ 次审计者选举（其中 $\ell \geq 0$ ），那么该用户在该月所能获得的报酬的期望是

$$E[\text{rewards per month}] = \ell w(p_1 w_c + p_2 w_v) = \ell w r.$$

用 \bar{c} 代表每个文件块每个月所需支付给云服务商的存储费用的期望，用 \bar{a} 代表每个文件块每个月所需支付给审计者的审计费用的期望。那么拥有 w 财富值（即拥有 w 个文件块）的用户每个月所需支付的存储费用的期望是

$$E[\text{cost per month}] = w(\bar{c} + \bar{a}) = w \bar{s}.$$

因此，该用户每个月的净花费是：

$$E[\text{net cost per month}] = w(\bar{s} - \ell r)。$$

不活跃用户。不活跃用户倾向于只享受可靠的云存储服务而不做出额外的费用。Audinet 通过激励机制促使这类用户积极参与对系统的贡献。假设某用户在云上存储的文件量恒定，即 w 值不变。那么，该用户每个月的净花费 $E[\text{net cost per month}]$ 与参与审计者选举的次数 ℓ 成反比例关系。因此，理性的用户会受此驱使而尽量增加 ℓ 以降低 $E[\text{net cost per month}]$ 。也就是说，在 Audinet 中，理性用户会尽可能多地参与审计者选举。

贪婪用户。贪婪用户可能会过分追求 ℓ 最大化以获取更多审计者报酬。但是事实上， ℓ 的值存在一个上限，因此理论上单个用户所能获得的审计者报酬存在上限。前面已经介绍过，每个用户在每个 epoch 只有一次机会可以参与审计者选举，那么我们可以得出

$$0 \leq \ell \leq \hat{\ell} = 3600 \times 24 \times 30 / \bar{e},$$

其中 \bar{e} 是平均一秒钟包含的 epoch 数。对于一个拥有 w 财富值的用户，每个月所能获得的平均报酬的最大值是 $\hat{\ell}wr$ 。贪婪用户可能会想要在云上存储尽可能多的文件以增加自身财富值，从而增加在审计者选举中被选中的概率。但是另一方面，上传到云上的文件会产生审计费用和存储费用。只要我们保证文件所产生的花费大于审计者报酬，便可阻止贪婪用户的行为。也就是说，每个用户每个月的平均净花费的最小值 $w(\bar{s} - \hat{\ell}r)$ 应该恒大于 0。因此，Audinet 系统参数设置应满足 $\bar{s} > \hat{\ell}r$ 。

我们接着分析如何才能满足 $\bar{s} > \hat{\ell}r$ 。首先， \bar{s} 是每个文件块在云上存储一个月所产生的费用，包括支付给云服务商的费用 \bar{c} 和支付给审计者的费用 \bar{a} 。假设一个文件块在一次审计中被选中的概率为 P ，那么每个文件块在一个月内被审计的期望次数是 $P \cdot \alpha \hat{\ell}$ ，其中 α 是每个 epoch 内生成的挑战的个数。假设在每次审计中，云服务商每成功证明一个文件块的完整性便可获取 \hat{c} 的酬劳。那么有

$$\bar{s} = \bar{c} + \bar{a} = P \alpha \hat{\ell} (\hat{c} + w_c + \beta w_v)$$

对于 $\hat{\ell}r$, 有

$$\begin{aligned}\hat{\ell}r &= \hat{\ell}(p_1 w_c + p_2 w_v) \\ &= \hat{\ell}\left(\frac{\alpha}{B} \cdot w_c + \frac{\alpha\beta}{B} \cdot w_v\right) \\ &= \frac{1}{B}\alpha\hat{\ell}(w_c + \beta w_v).\end{aligned}$$

于是有

$$\begin{aligned}\bar{s} - \hat{\ell}r &= P\alpha\hat{\ell}(\hat{c} + w_c + \beta w_v) - \frac{1}{B}\alpha\hat{\ell}(w_c + \beta w_v) \\ &= \alpha\hat{\ell}\left(P\hat{c} + (w_c + \beta w_v)\left(P - \frac{1}{B}\right)\right)\end{aligned}$$

为了保证 $\bar{s} - \hat{\ell}r > 0$, 我们应该使得系统参数满足

$$\begin{aligned}P\hat{c} + (w_c + \beta w_v)\left(P - \frac{1}{B}\right) &> 0 \\ P\hat{c} &> (w_c + \beta w_v)\left(\frac{1}{B} - P\right) \\ \hat{c} &> (w_c + \beta w_v)\left(\frac{1}{BP} - 1\right)\end{aligned}$$

3.6.2 针对云服务商的激励机制

Audinet 采用了为存储可靠性付费的支付模式, 云服务商只有在成功通过审计时才可以收取相关的服务费用。与传统支付模式相比, 云服务商无需再向第三方支付数据审计费用, 因此成本降低、净收益增加。对于在审计过程中回复得较慢的云服务商, 或者是有选择性地不回复一些挑战的云服务商, 由于有惩罚措施的存在, 这些云服务商的收益会降低。

惩罚拖延行为。如果云服务商提交回复的时延较大, 会导致智能合约所维护的审计表越来越大, 并使得审计进程拥塞。我们希望云服务商尽可能快速地针对每个挑战进行回复。因此, 作出如下规定。对于每个已经生成的挑战, 智能合约统计回复时延 τ 为从挑战生成开始到回复提交为止的时间间隔。在每次审计完成后, 智能合约计算每个文件块支付给云服务商的费用是 $c_\tau = \hat{c} - k\tau$, 其中 k 是一个系统常量。也就是说, 回复时延越小则云服务商所能获得的酬劳越高。特别地, 假如回复时延超过了 \hat{c}/k , 那么即使云服务商通过审计, 也不会收

取到酬劳。因此，一个理性的云服务商会倾向于尽快提交回复。作为云服务商，可以考虑运行一个调度算法，来分配合适的资源用于计算回复。

惩罚恶意行为。云服务商可能会故意不回复某些挑战——比如当被选中的数据已经丢失或者损毁时。我们认为这样的行为是恶意的且必须加以杜绝的，并对云服务商的此类行为给予惩罚。例如，假如回复延迟已经超过了 $\hat{\tau} = 2\hat{c}/k$ ，那么智能合约将本次审计标记为失败并从云服务商的押金中扣除 $3\hat{c}$ 作为惩罚。

第七节 实验测试

本节共设置四组实验，简介如下。（1）第一组实验针对审计者选举，验证一个用户在审计者选举中获胜的概率是否与该用户的财富值正相关。（2）第二组实验针对挑战阶段生成的挑战，验证所有文件是否依照均匀分布等概率地被选中。（3）第三组实验为 NIST 测试，验证 Audinet 所使用的两个伪随机函数是否满足随机性。（4）第四组实验针对系统性能，测试 PoR^{*} 和 zk-PoR^{*} 审计执行验证所花费的时间开销。

3.7.1 实验设置

我们基于一个使用 C# 语言实现的多云存储原型系统构建 Audinet 原型系统。审计相关核心代码使用 C++ 和 C# 语言完成，共包括三部分：客户端，服务端，以及智能合约。在客户端实现方面，使用 OpenSSL 1.0.2d 库实现审计者选举中的伪随机方程，使用 PBC 库^[205] 实现 BLS 签名，以上两者均使用 C++ 语言实现并编译成动态链接库（DLL, dynamic link Library）供云存储系统进行调用。在智能合约实现方面，使用 Solidity^[206] 编写智能合约，使用 Ganache^[207] 构建以太坊（Ethereum）本地测试链并将合约部署到测试链上。在服务端实现方面，使用 C++ 语言提供的 PBC 库实现云端计算回复的程序并编译成 DDL 供云存储系统进行调用。云端与智能合约交互的程序使用 Ethereum 提供的 Nethereum 库^[208] 实现。在零知识证明实现方面，使用 circom 库^[203] 编译算术电路，并使用 snarkjs 库^[209] 计算零知识证明以及生成验证零知识证明的合约。我们在原本云存储系统中新加入的 Audinet 核心代码约有 1300 多行，这说明在已有的云存储系统中引入 Audient 相关功能是较容易实现的。在实验过程中，使用一台电脑运行服务端和客户端。电脑配有 8GB 的 RAM，4 核 Intel i7-8565u 处理器，运行 64 位 Windows 10 操作系统。

3.7.2 财富值与获胜概率

在 Audinet 系统中，审计者选举负责从所有用户中筛选出足够多诚实审计者，是保证审计安全性的重要环节。审计者选举算法的设计遵循两个原则：(1) 随机性，即无法预测哪些用户将成为审计者，也无法控制某个用户选举成功与否。这是为了保证攻击者无法通过操控审计者来操控审计结果。对随机性的测试将在3.7.3小节给出。(2) 用户在审计者选举中获胜的概率与其在云上存储的数据量（即财富值）成正比。这一设计基于一个假设，即用户在云上存储的数据越多则越有意愿维护存储系统的安全和稳定。本小节测试审计者选举是否满足这一特性。

实验设置如下使用场景。系统中共有 10 个用户 U_1, \dots, U_{10} ，每个用户拥有的财富值依次为 $1, \dots, 10$ ，即有 $w(U_i) = i$ 。令每个用户参与 100 次审计者选举并统计其被成功选中的次数，记为获胜频次。例如，若用户在一次审计者选举中的选举结果是 $j > 0$ ，那么将该用户的获胜频次加 j 。实验测试过程如下。使用用户 U_1 的账号和密码登陆 Audinet 系统。每隔 5 秒运行一次审计者选举¹直到参与 100 次选举。假如用户在某次选举中获胜，那么选举结果 j 被写入到本地日志文件 `election.txt`。对用户 U_2, \dots, U_{10} 重复上述过程，得到每个用户在 100 次审计者选举中的获胜频次。在 5 种不同的 p_1 值下重复上述实验，得到结果如图3.5所示。图中横轴表示财富值为 $1, \dots, 10$ 的用户，纵轴表示不同用户的获胜频次。我们可以看出，同一个用户的获胜频次随着 p_1 值的增大而增大，这表明了参数 p_1 对用户选举成功概率的影响。系统参数 $p_1 = \alpha/B$ 给出了在挑战者选举中每单位财富被选中的概率，其中变量 α 为每个 epoch 内被选中的财富值，变量 B 为系统内总共财富值。从系统层面，调整 p_1 的值可以改变每个 epoch 内被选出的挑战者数量，从而改变该 epoch 内生成的挑战个数；从用户层面，调整 p_1 的值将影响用户在每次选举中成功的概率。另外我们看到，在相同的 p_1 值下拥有财富值越高的用户在审计者选举中的获胜频次越高。这一实验结果与前文理论分析一致。这说明，审计者选举算法的设计能够实现 Audinet 对审计安全性的需求。

3.7.3 审计的均匀性

为了提高存储可靠性，防止数据丢失或损坏的发生，数据审计必须能够均匀扫描并完整覆盖所有文件。但是假如挑战生成算法无法保证审计的均匀性，可

¹在实验中，设置一个 epoch 的长度小于 5 秒。

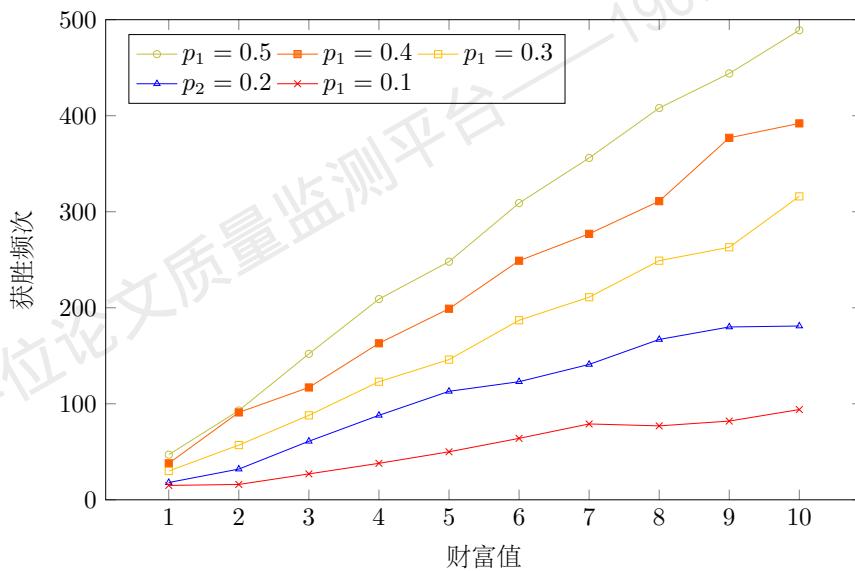


图 3.5 在 100 轮审计者选举中，一个用户的获胜频次正比于其财富值。

能会出现某些文件很长时间无法被审计到的情况。这可能导致一部分已经发生的数据损坏无法被及时发现，错过执行数据修复的最佳时间，并进一步导致数据丢失或损坏事件的发生。本小节测试在一段时间内被选中的文件是否分布均匀。我们设置如下使用场景。系统中共有 10 个用户 U_1, \dots, U_{10} ，每个用户在云上存储的文件数量依次为 $1, \dots, 10$ ，因此云上共有 55 个文件。每个文件被切分成 8 个等大的文件块。根据第三节给出的定义，一个用户的财富值 $w = b/\theta$ ，其中 b 是用户在云上存储的文件块数而 θ 是系统常量。那么，用户 U_i 拥有的文件块数 $b_i = 8i$ 。设置 $\theta = 8$ ，则用户 U_i 拥有的财富值 $w(U_i) = b_i/\theta = 8i/8 = i$ 。

实验测试过程如下。使用用户 U_1 的账号和密码登陆 Audinet 系统。每隔 5 秒运行一次审计者选举直到参与 100 次选举。假如用户在某次挑战者选举中获胜，则调用合约中的挑战生成函数 `GenChall()` 并将选举结果 j 发送给合约。合约运行函数 `GenChall()` 并循环 j 次从而生成 j 个挑战。最后，将被选中文件的信息输出到本地日志文件 `challenge.txt`。每个用户参与 100 次挑战者选举。对用户 U_2, \dots, U_{10} 重复上述过程，统计每个文件被选中的总次数。在 5 种不同的 p_1 值下重复上述实验，实验结果如图3.6所示。定义统计变量 $\delta = \text{最大选中次数} - \text{最小选中次数}$ ，定义变量 S 为一轮实验中所有文件的总选中次数，并计算 δ/S 的值，结果在表格3.5给出。可以看到，虽然随着 p_1 值的增大

表 3.5 审计均匀性测试结果统计。

	$p_1 = 0.1$	$p_1 = 0.2$	$p_1 = 0.3$	$p_1 = 0.4$	$p_1 = 0.5$
δ	5	8	10	14	15
S	552	1088	1622	2187	2711
δ/S	9.06	7.35	6.17	6.40	$(\times 10^{-3})$

δ 也在增大，但是实际上 S/δ 的值在逐渐降低，且始终保持在 10^{-3} 数量级。这表明不同文件的选中次数都被保持在一个跨度较小的数值区间之内。本组实验结果证明在挑战生成阶段不同文件被选中的概率非常接近。

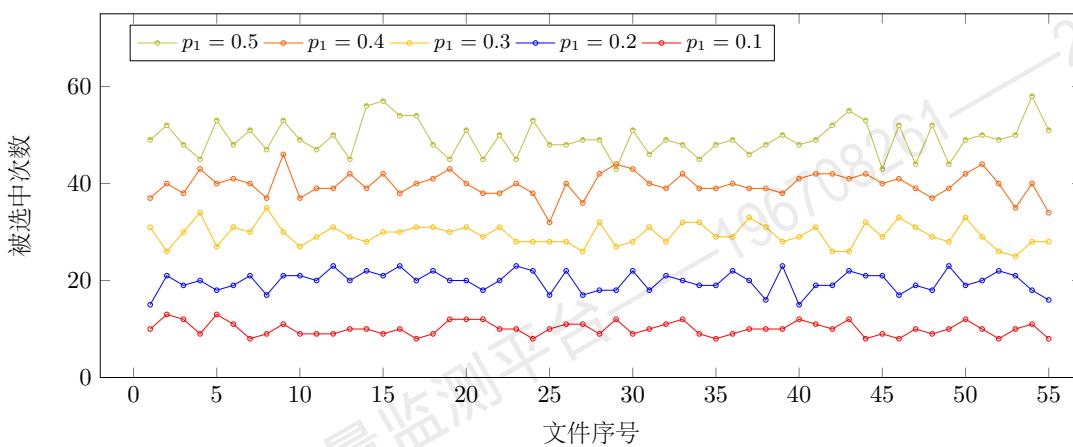


图 3.6 运行 1000 次审计者选举，统计每个文件被选中的次数。

3.7.4 NIST 随机性测试

决定审计随机性的关键在于 Audinet 系统使用的两个伪随机数发生器。其中，审计者选举算法使用可验证随机方程 $\langle \text{hash}, \pi \rangle \leftarrow \text{VRF}_{sk}(\text{epoch} || \text{role} || \text{seed})$ 生成随机哈希值（见表达式3.1），该哈希值决定用户能否被选中为审计者。挑战生成算法使用伪随机数发生器 $f \leftarrow \text{keccak256}(\text{time}, \text{addr}, \text{nonce}) \bmod N$ 生成随机文件索引号（见算法3.1第一行），该索引号决定本次审计要验证的文件。上述两个伪随机方程能够保证审计者选举的不可预测性、不可操控性以及文件选中的均匀性。因此，我们使用 NIST SP 800-22 测试包^[210] 针对这两个方程的随机性进行测试。NIST 是目前较为权威的测试标准，用于判断给定序列是否满足随机序列的若干特性，从而判断该序列是否为随机序列。NIST 测试共包含 15 项

表 3.6 NIST 测试结果：VRF 函数的结果用 * 标记，deccak256 函数的结果用 ** 标记。

测试名称	P 值 *	通过率 *	结果 *	P 值 **	通过率 **	结果 **
Frequency	0.085587	0.987	pass	0.387264	0.993	pass
Block Frequency	0.299736	0.992	pass	0.854708	0.99	pass
Cumulative Sums	0.279033	0.989	pass	0.508406	0.995	pass
Runs	0.075719	0.988	pass	0.753844	0.989	pass
Longest Run	0.192724	0.992	pass	0.299736	0.989	pass
Rank	0.915317	0.987	pass	0.816537	0.993	pass
FFT	0.725829	0.986	pass	0.873987	0.992	pass
Non-overlapping Templates	0.494675	0.99	pass	0.516065	0.99	pass
Overlapping Templates	0.32985	0.992	pass	0.474986	0.986	pass
Universal	0.390721	0.987	pass	0.896345	0.989	pass
Approximate Entropy	0.881662	0.99	pass	0.00085	0.998	pass
Random Excursions	0.634678	0.988	pass	0.454606	0.989	pass
Random Excursions Variant	0.43824	0.99	pass	0.564499	0.989	pass
Serial	0.606242	0.994	pass	0.474556	0.99	pass
Linear Complexity	0.43543	0.99	pass	0.461612	0.992	pass

测试，每项测试输出一个 P 值 (P-value)。若 P 值大于 0.001 则认为给定序列满足随机性，该项测试通过 (pass)。若 15 项测试全部通过，则给定序列满足随机性要求。在实验过程中，我们分别使用上述两个函数生成 1000 个 10^6 位数据样例作为测试程序的输入值，测试结果如表格3.6所示。我们看到，在 15 项测试中通过率都不小于 0.985，并且所有的 P 值都大于 0.01。这说明 Audinet 所使用的 VRF 函数和 keccak256 哈数具有很好的随机性。

3.7.5 验证阶段时延

在 PoR^{*} 协议的验证阶段，每个回复会被分配给 β 个验证者进行验证并投票。由于要等待足够多验证者投票之后才能得出最终的审计结果，可能导致验证阶段时间较长。当 β 的数值增大，审计结果会更为可靠，但同时统计投票所需要的时间也会更长。定义 PoR^{*} 协议的验证时延为自云服务商提交回复到最终得出审计结果所花费的时间开销。在本小节中我们测试 β 的值如何影响 PoR^{*} 验证时延的大小。设置如下使用场景。系统中有 100 个用户 U_1, \dots, U_{100} ，每个用户的财富值为 $w(U_i) = i \bmod 10$ ，每个用户都有一个 Ganache 地址。合约所维护的审计表中共有 10 条记录 $[C_i, R_i, null]_{\forall i \in \{1, \dots, 10\}}$ 。设置挑战者选举参数 $p_1 = 0.1$ ，

则 $p_2 = \beta p_1 = 0.1\beta$ 。实验过程中模拟真实系统的用户操作，测试过程如下。每隔 2 秒，随机选中一个用户参与验证者选举。若用户被选中，则调用合约函数 RtrResp() 获取一个待验证的回复。合约将从审计表中随机选中一个待验证回复并分配给验证者。获取到一个回复后，用户验证该回复并调用合约函数 Vote() 进行投票。当有 $\frac{\beta}{2} + 1$ 个验证者投 $V = 1$ 时或者有 $\frac{\beta}{2} + 1$ 个验证者投 $V = 0$ 时，审计结束。合约统计每一个回复的验证时延。直到所有回复 R_1, \dots, R_{10} 全部审计结束，实验终止。在不同的 p_2 值下重复上述测试。我们根据审计结束的先后对所有文件进行排序，得到结果如图3.7所示。我们可以看到，当 p_2 值每增加 0.1 时验证时延会有一个比较明显的增加，这是因为参与投票的验证者变多导致投票等待时间变长。另外，在 p_2 值相同的情况下不同文件的验证时延相差也较大。为了避免这种情况，我们可以在智能合约中加入一个调度算法。例如，若合约发现当前某个回复的验证等待时间较长，便为该回复优先分配更多的验证者。再如，若合约发现当前全部回复的等待时间均较长，则通知系统后台调整参数 β 的值。

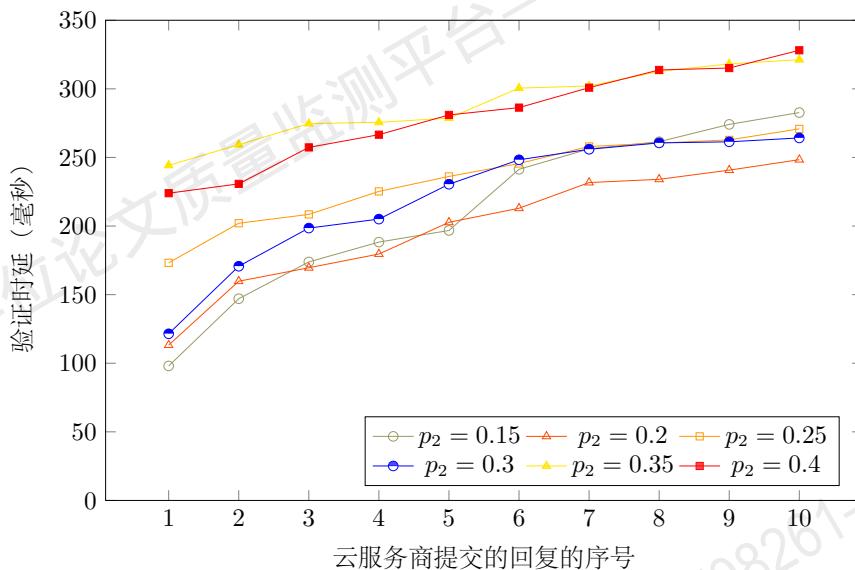
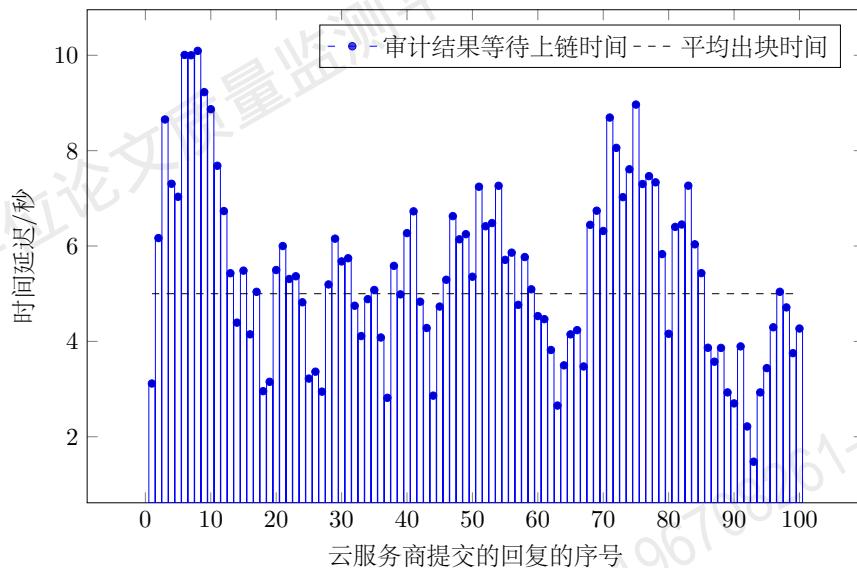


图 3.7 PoR* 验证阶段时间延迟统计。

zk-PoR* 协议在验证阶段不需要审计者的参与，而是由智能合约自动化验证云服务商提交的零知识证明。zk-SNARK 算法的特点是验证速度快^[90,91]，因此当合约中的零知识验证函数 zkVerification() 被调用后几乎立刻就能得到验证结

果。针对 zk-PoR^{*} 协议，我们忽略智能合约执行验证算法的时间开销，而是评估合约得出的审计结果何时能够被打包进新区块。定义 zk-PoR^{*} 协议的验证时延为自云服务商提交回复到验证结果被打包进新区块所花费的时间开销。大部分区块链系统会根据当前出块时间间隔实时调整系统参数以保证出块间隔维持稳定。不过每个 epoch 实际测得的数值可能与系统设置的平均值有所出入。因此，本组实验使用泊松分布模拟区块链系统的出块过程，并设置平均出块时间间隔为 5 秒。同时，使用泊松分布模拟云服务商提交 100 个回复的过程，并设置提交回复的平均时间间隔为 2 秒。假设每个审计结果所对应的交易都会被打包进下一个新区块中。统计从云服务商提交回复到审计结果被打包进新区块的时间间隔，结果如图3.8所示。根据测试结果，尽管出块时间并不稳定，但是约有 43% 的审计结果能够在 5 秒之内上链，剩余审计结果都能够在 10 秒内被打包上链。我们认为这一时延是可接受的。等待审计结果上链的时长主要影响云存储系统启动数据修复的时间。在第五章中我们指出，当审计间隔设置为大约半小时一次时²即可很好地保证存储可靠性。那么，即使在发现数据损坏后需要等待 10 秒钟才能启动数据修复，也并不会加剧数据丢失的可能性。

图 3.8 zk-PoR^{*} 验证时延统计。

²在第五章我们推荐的审计频次为 40–50 次/天，换算为审计间隔约为半小时一次。

第八节 本章小结

本章提出了一个分布式公开审计系统 Audinet，该系统包括如下几个创新点。(1) 采用了基于区块链的分布式审计模型。结合云存储用户持有资源和在线时长参差不齐的现况，我们鼓励一部分资源充足的用户参与到审计活动中，从而构建无需依赖可信第三方的分布式审计网络。另外，基于区块链实现了审计记录的不可篡改性和可追溯性，基于智能合约实现自动化的审计流程，从而实现了审计的不可抵赖性和纠纷定责。(2) Audinet 能够保证稳定的审计频次，这对于审计系统的稳定性十分重要。同时，Audinet 能够保证所有用户的文件被均匀选中，这很好地保证了审计系统的可靠性和公平性。(3) 审计者选举可以保证拥有财富越多的用户有越大的概率被选为审计者。这一方面保证了系统的安全性（我们认为在云上存储了越多数据的用户越有可能是诚实用户），另一方面激励了用户更多地使用系统、参与到审计活动中。(4) Audinet 保证可靠的审计结果，以及审计的可追溯性、可审计性。

本方案在设计过程中的一大难点在于如何保证审计者中有足够多³的诚实用户。为此，我们借鉴了 Algorand^[211] 中所提出的密码抽签算法，设计了基于伪随机方程的审计者选举方法。该方法能够保证用户在云上所存储的数据量与其在审计者选举中获胜的概率成正比。通过激励机制我们能够保证，如果攻击者想要通过上传大量数据到云端从而提高被选中为审计者的概率（进而操纵审计结果），那么这种攻击将会得不偿失，因为存储大量数据将导致较高的存储费用。我们在设计过程中的另一个难点是如何避免存储在区块链上的公开信息泄露用户隐私。我们借助零知识证明框架构造了实现隐私保护的 zk-PoR^{*} 协议。不过，该协议可能存在通信开销较大的问题。一个回复中包含 m 个零知识证明。当 m 的值较大时可能导致传输数据量较大。这个问题可以使用 SnarkPack^[212] 技术解决。SnarkPack 是一种针对 Groth16 证明的聚合技术，可以将 m 个证明的长度降低为 $\log m$ 级别，同时验证时间为 $\log m$ 级别。

³为保证审计结果的可靠性，在所有验证者中诚实用户占比必须大于 50%。

第三章 隐私友好的分布式公开审计方案

第四章 实现存储优化的审计联盟链系统

第一节 引言

区块链技术的出现为远程数据审计带来新的范式。借助区块链去中心化、不可篡改、可追溯的特点，传统审计方案存在的不足能够得到很好弥补。例如，传统私有审计无法实现不可抵赖性以及纠纷定责。而将审计验证子公开发布在区块链上，可以实现不可抵赖、可定责的私有审计，避免云服务商或者用户的恶意行为。传统公开审计依赖第三方审计者（TPA, third-party auditor），导致审计过程不透明且审计结果无法验证。而将审计协议实现在智能合约中，可以保证审计过程公开透明，审计记录可查验可追溯，审计结果可验证。

近年来，许多学者提出了基于区块链的远程数据审计方案。这些方案的设计目标和设计方法各有不同，但从所采用的系统模型的角度看可分为两大类，介绍如下。（1）审计网络与区块链共识网络分离。例如，文献[16]假设审计系统所使用的区块链平台由外部节点负责维护（外部节点将负责区块链出块、区块存储等工作），云服务商和用户作为区块链轻节点，可以调用合约执行特定程序或者读取链上数据。这类方案的弊端有两点。一方面，外部节点缺乏维护区块链系统的内在动机，需要加强激励措施才能保证系统稳定运行。这给审计系统增加了新的不稳定因素。另一方面，由外部节点维护区块链的做法本质上是将执行审计协议以及存储审计记录的任务进行了外包，这使得数据安全性和审计安全性减弱。（2）审计网络与区块链共识网络合一^[15,17,138,200]。将云服务商和用户部署为区块链共识节点，全部审计记录由云存储用户负责维护和存储，无需依赖外部区块链服务网络。我们称之为“审计区块链”系统。从审计安全性和系统稳定性角度看，第二类方法是更为可取的。

在设计审计区块链系统时，我们需要仔细考虑存储可扩展性的问题。现有区块链系统普遍采用全副本存储模式，每个共识节点需要存储完整的区块链数据副本并且不可以删除数据，这导致节点存储开销不断增大。参考针对主流公有链的统计结果，截至2021年比特币数据总量超过350GB^[148]，2019年以太坊数据总量超过2TB。联盟链交易吞吐率高于公有链，因而数据增长更快。假设

某联盟链平均区块大小为 2MB，出块间隔为 2 秒，则一个月将产生 2.5TB 数据。如此巨大的存储开销对大部分云存储用户来说都是难以承担的。大部分云存储用户不具备足够的存储资源，或者没有意愿贡献过多存储资源。加入审计区块链系统引发的存储代价过大可能会影响用户的使用意愿，而审计区块链系统中节点数量过少将影响系统的去中心性和安全性。因此，我们认为全副本存储模式并不适用于审计区块链系统。想要构建安全可信可靠的审计区块链系统，必须优化区块链存储模式、降低节点存储开销。文献 [19] 的作者也提到，其研究工作的一个改进点便是提升区块链的存储可扩展性。

目前工业界和学术界已经提出许多区块链存储优化方案，致力于解决区块链存储开销过大的问题。例如，简单支付验证机制允许轻节点在存储少量数据^[13] 或者不存储额外数据^[150] 的情况下，通过与区块链全节点交互来验证区块链交易的合法性。轻节点不参与系统共识，也不维护区块链。SPV 机制使得资源受限的用户也能够使用区块链服务，提升了系统可扩展性。考虑到云存储用户的存储资源、带宽资源和计算资源差异性较大，我们可以将 SPV 机制应用于资源十分受限且不想参与数据审计的云存储用户。例如，大部分个人用户使用的是台式电脑，存储空间有限，且在线时间不稳定，没有能力参与区块链的维护工作。我们可以允许这些用户作为区块链轻节点享受审计系统提供的服务（同时支付一定的审计费用）。一些企业级用户采用的是公有云加私有云的混合架构。这些企业拥有本地服务器，有能力参与区块链的维护工作。还有些企业级用户使用的是多云架构，因此可以使用 A 云服务器运行区块链节点以审计 B 云上的数据。这些用户可以作为区块链服务节点参与智能合约运行及区块链维护等工作。总的来说，我们需要根据用户的资源持有情况为它们分配不同的系统角色。文献 [17] 也采取了类似的思路，该方案将所有用户区分为负责参与区块链共识的管理节点和任务量较少的轻节点两类。不过在该方案中，每个用户组只分配一个共识节点，容易导致单点故障。

在针对全节点的存储优化方面，主要有修剪和存储分片这两类方法。修剪技术允许节点删除掉一部分历史数据，如只保存少量区块^[152] 或是少量交易^[155]。修剪节点仍可参与区块链共识。不过，这一方法并不适用于审计区块链系统。区块链记录着用于执行审计的验证子或者用于后期追溯和审查的审计记录，因此我们不能允许节点随意删除链上数据。2018 年，文献 [157] 针对联盟链系统提出存储分片思想：在全网划分若干存储组，通过区块散布机制使组内节点以协作

方式维护一份或多份区块链副本。存储分片能够在节点存储开销、存储可靠性和安全性之间取得良好平衡，不过相关研究工作仍有待进一步探索和完善。例如，一些方案提出的区块散布算法主要针对静态场景^[158-160]，并不适合于持续执行审计、区块链数据持续增长的动态场景。一些方案虽适用于动态场景，但散布过程需要中心化节点^[161]，不适合分布式审计的场景。一些方案未能结合区块访问热度进行性能优化^[162-164]，导致查询链上数据的性能低下。总的来看，目前已提出的区块链存储优化方案都不能很好地适用于审计区块链这一场景。

作为总结，目前提出的基于区块链的审计方案存在如下薄弱之处：

1. 审计网络与区块链网络分离，存储着重要审计信息的区块链由外部节点维护，导致审计系统稳定性差、安全性弱；
2. 已提出的审计区块链方案未针对区块链存储可扩展性进行优化，导致共识节点存储开销过大、系统去中心化程度和安全性降低；
3. 未能结合不同用户拥有资源的差异性为用户赋予不同的角色和任务；
4. 已有的区块链存储优化方案无法直接应用于审计系统，必须进行针对性优化。

针对上述问题，本章提出一种适用于云存储场景的审计联盟链系统。云服务商（作为被审计者）和所有用户（作为审计者和监督者）都是联盟链节点。根据持有资源多少的不同，用户被划分为区块链服务节点和区块链轻节点两类。服务节点参与共识并负责维护区块链、存储一部分区块链数据；轻节点只使用区块链服务，不维护数据。云服务商有充足的存储资源和计算资源，所以担任联盟链全节点，存储完整的区块链数据。针对所提出的审计联盟链系统设计一种动态弹性存储分片方案。每个服务节点只需存储一部分区块链数据，并通过节点协作来访问本地未存储的数据。为在数据可靠性、数据访问性能和节点存储开销间取得平衡，根据实时区块访问热度动态调整区块的副本个数、存储位置和存储模式。具体地，初始散布机制为每个新区块分配固定数量的初始存储节点，以保证数据可靠性；动态弹性复制机制增加最近较热区块的副本数，以降低数据访问时延；存储优化机制将最近较少访问的历史区块切换为编码模式，以降低存储开销。存储优化过程不会影响数据可靠性。

本章的后续小节以如下方式组织：第二节给出审计联盟链系统的设计方案；第三节介绍了适用于审计联盟链系统的存储分片模块的设计细节；第四节描述了在采用存储分片的审计联盟链系统中节点间如何进行数据访问；第五节针对

存储分片模块进行实验测试，验证该模块在存储、性能、自适应性方面的表现；第六节针对审计联盟链系统进行实验测试，验证方案可行性及性能；第七节总结了本章的研究工作。

第二节 审计联盟链系统设计

在本节中，我们提出一种审计联盟链系统，并给出该系统的设计概要，包括系统模型、主要工作模块以及系统工作流程。另外，我们着重介绍了存储分片模块的设计思想。

4.2.1 系统模型

如图4.1所示，在审计联盟链系统中，我们可以从三个层面看待云服务商和云存储用户的关系。（1）云存储角度：云服务商是服务提供者，用户是服务使用者。相对应地，云存储模块包含与文件上传下载相关的功能，主要涉及用户与云服务商之间的交互。（2）远程数据审计角度，云服务商是审计对象，用户是审计者。相对应地，远程审计模块包含与数据审计相关的功能，涉及到用户与区块链、云服务商与区块链之间的交互。所有的审计记录都写入到区块链上，以备监督、追溯和审查¹。（3）区块链系统角度：云服务商和用户都是联盟链节点，但由于资源的多寡被区分出如下三类角色。

- **全节点：**特指云服务商。全节点负责参与共识，维护区块链并保存完整区块链副本，且有权调用智能合约和查看审计历史记录²。
- **服务节点：**由云存储用户中存储和计算资源较充足者构成。与全节点的不同之处在于，每个服务节点只存储一部分区块链数据，并在需要时向其它服务节点请求本地未保存的数据。
- **轻节点：**有权调用智能合约、查看审计数据，但不维护区块链。每个轻节点连接到至少一个服务节点。

存储分片模块实现了存储模式的优化。该模块将所有用户划分成若干个存储组，使得每个存储组内部独立执行区块散布机制以降低服务节点的存储开销。各个服务节点只需存储一部分区块，且所存储的区块集合作随实时区块热度动态调整。

¹第三章介绍的分布式审计方案可看作远程审计模块的一个实例。

²全节点需要具有较强的计算能力和存储能力，因此我们假设只有云服务商是全节点。

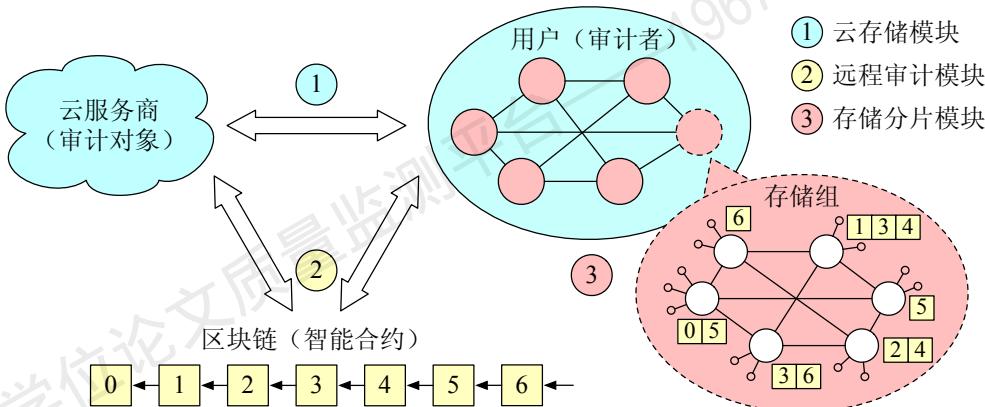


图 4.1 系统模型。

4.2.2 系统工作流程

在一次审计过程中，轻节点或服务节点通过调用智能合约来对云服务商发起挑战或是验证云服务商提交的回复；云服务商通过调用智能合约来对审计者发起的挑战进行回复以证明文件完整性。所有审计记录作为监管凭证记录在区块链上。我们假设审计联盟链系统中执行的是第三章所介绍的 zk-PoR^{*} 审计协议，并介绍一次完整的工作流程示例。图4.2给出了下述过程的一个示意图。

1. 文件上传下载（云存储模块）：

用户 A 上传文件 F ，并在上传之前进行预处理以备后续审计时使用。上传成功后，用户 A 可随时下载该文件。

2. 数据审计（远程审计模块）：

- (a) 文件 F 上传到云端后，某个时刻用户 B 成功通过挑战者选举，作为挑战者触发智能合约并生成一个新的挑战 C 。该挑战要求云服务商证明文件 F 的第 1, 3, 5 个块的完整性。
- (b) 云服务商作为被审计者，计算回复并提交给智能合约。
- (c) 智能合约运行验证函数并得出最终审计结果。

3. 审计记录上链（远程审计模块及存储分片模块）：

- (a) 在上述文件上传和文件审计过程的同时，所产生的所有记录都会以区块链交易的形式被打包到不同的区块中。例如，图中交易 1 记录了“用户 A 上传文件 F ，其中包含 n 个文件块”，交易 2 记录了“用户 B 生成挑战 C ，选中了文件 F 的第 1, 3, 5 个块”。
- (b) 对于每个新产生的区块，每个服务节点运行初始散布算法来确定是否

存储该区块。初始散布机制使得在每个存储组内有固定数量的服务节点存储这一新区块的初始副本。

4. 查看审计记录（存储分片模块）：

- (a) 若某个轻节点想要查看历史审计记录，则轻节点发消息给所连接的服务节点。服务节点确认被请求的数据是否在本地存有副本，若没有则向其它服务节点请求数据。被请求节点会尽快返回数据。
- (b) 在存储组内，伴随节点间数据请求某些区块的副本数量和存储状态会发生改变。对于某些热度较高的区块，动态复制机制增加其副本数量以提升数据访问性能。对于某些热度较低的区块，存储优化机制将其切换为编码模式，以在不影响存储可靠性的前提下降低存储开销。

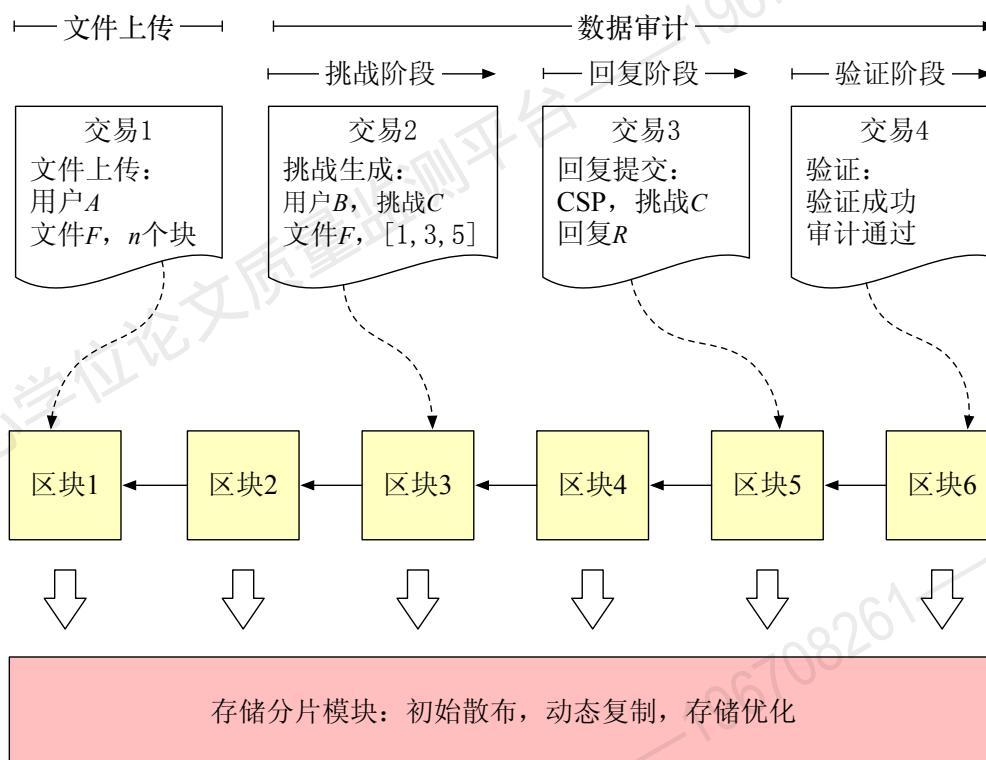


图 4.2 审计记录上链。

4.2.3 存储分片模块

存储分片模块的作用是优化审计区块链系统中服务节点的数据存储模式³。定义一个区块链存储分片（BSS, blockchain storage sharding）系统为由联盟链中的一部分节点构成的相对独立的存储组。全网内包含若干个 BSS 系统。服务节点可以选择加入某个 BSS 系统从而与其它服务节点协作式地维护区块链副本，降低存储开销。这将改变节点的数据存储模式，但并不会影响共识层协议。

在介绍存储分片模块的设计思想之前，我们需要先对区块链数据结构进行定义。如图4.3所示，一个区块包含两部分，其中区块头记录元数据，区块体记录一段时间内产生的交易（按默克尔树结构打包）。每个区块对应唯一标识符 *blockID*，即区块头 SHA256（SHA, secure hash algorithm）哈希值；每个交易对应唯一标识符 *txID*，即交易 SHA256 哈希值。区块头 *prev Hash* 字段记录前一个区块的 *blockID*，由此所有区块前后链接为链式结构。区块头 *root Hash* 字段记录区块体包含所有交易的哈希摘要，由此交易之间、区块体与区块头之间形成链式结构。在链式结构及共识机制的共同作用下，成功篡改区块链数据而不被发现的概率很低。基于上述数据结构，存储分片模块以区块（而不是交易）为单位对区块链账本进行切分。这样设计的好处是保留了区块内部设计精巧的链式结构，从而数据不可篡改性得以最大程度保留，系统设计也较为简洁。作为对比，假设以交易为单位切分账本，即每个节点存储一部分交易集合，那么必须引入新的交易验证协议和区块验证协议来保证数据不可篡改性，系统复杂程度大大增加，且节点需要额外存储对每个交易的存在性证明。

存储分片模块执行的散布规则基于服务节点与数据对象之间的逻辑距离，在此我们对“距离”的概念进行定义。每个服务节点被分配一个 256 比特长度的标识符 *nodeID*，即节点公钥的 SHA256 哈希值（SHA, secure hash algorithm）。每一个区块对应一个唯一标识符 *blockID*，即区块头的 SHA256 哈希值。每一个交易对应一个唯一标识符 *txID*，即交易的 SHA256 哈希值。定义某节点与某区块或交易之间“距离”为相应标识符按位异或的结果。以长度为 4 位的标识符举例，*nodeID* 为 0010 的节点与 *blockID* 为 0101 的区块之间的距离为 $0010 \oplus 0101 = 7$ ，而同一节点与 *txID* 为 0001 的交易之间的距离为 $0010 \oplus 0001 = 3$ 。我们说该交易比该区块距离该节点更近。可以看到，两个标识符的公共前缀越长，那么它们

³该模块与远程审计模块是正交的，因此存储分片既可以应用于执行私有审计的系统也可以应用于执行公开审计的系统。在下文中，我们暂且以执行公开审计的系统来进行举例。

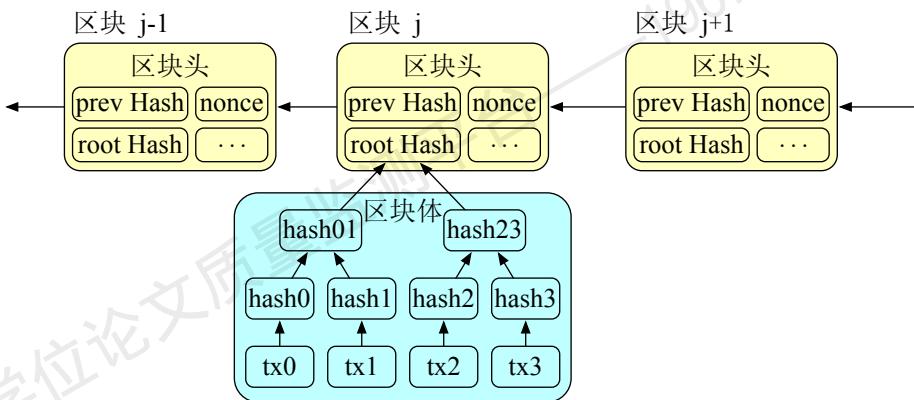


图 4.3 区块链数据结构。

之间的距离越近。定义一个区块项为 $\{\text{blockID}, \text{blockData}\}$ 二元组，其包含一个区块的数据。定义一个交易索引项为 $\{\text{txID}, \text{blockID}\}$ 二元组，其包含一笔交易所在的区块。在执行散布时，存储分片模块总是将区块项或者索引项分配给距离其 ID 最近的服务节点。

存储分片模块的引入不会影响共识层协议，但会改变服务节点对链上数据的读写方式。图4.4给出了存储分片模块的主要工作流程。假设用户创建一笔新交易并发送给连接的服务节点。该交易将被广播至全网，并根据共识协议打包到一个新区块。新区块生成后，每个服务节点执行初始散布决定是否保存该区块。如图4.4所示，区块 4 经过初始散布由节点 b 和 d 进行保存，而节点 a 和 c 仅保存区块 4 的区块头。初始散布机制保证每个区块在系统内的副本数不少于某一固定值。假设用户想要获取某个历史区块但所连接的服务节点并未存储该区块。服务节点会在接收到用户请求后向其它服务节点进行请求，并在成功获取目标区块且验证有效后将区块返回给用户。服务节点间区块请求可能会触发对被请求区块的动态复制。如图4.4所示，动态复制使得区块 4 在系统中增加一个副本（由节点 c 存储）。动态副本设置有生命周期，因此一段时间后节点 c 会删除区块 4 的副本。动态复制机制可以提升数据访问性能。系统后台运行的存储优化机制将最近较少访问的历史区块成批编码并删除系统内多余副本，从而在不降低数据冗余度的基础上压缩存储开销^[213]。如图4.4所示，存储优化机制使得节点 a 所保存的区块 1、3 的副本被校验块 x 所替代。节点 a 保存的校验块 x 与节点 c 所保存的区块 1、节点 b 所保存的区块 3 构成一个校验组。存储优化机制可以降低存储开销。

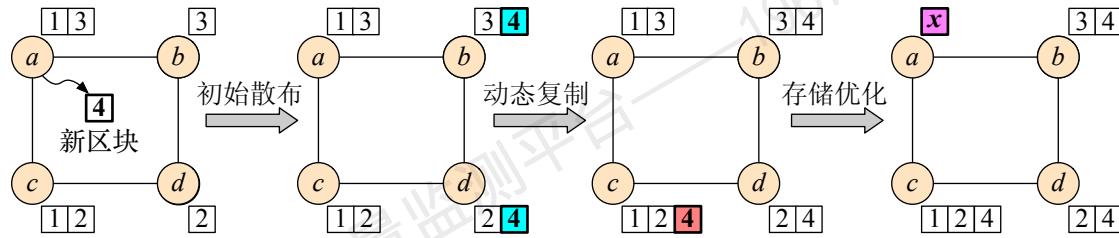


图 4.4 存储分片主要工作流程。

假设联盟链有身份准入机制，全节点和服务节点间彼此互知。以下统一将“服务节点”简称为“节点”。存储分片模块的设计应满足如下要求。(1) 可靠性：能够容忍一定数量的节点故障。(2) 高性能：尽量降低存储分片所引发的节点间数据访问的时间开销。我们使用访问一个区块平均花费的时间开销来衡量系统性能水平。(3) 空间节约：尽量降低节点存储开销，同时不影响可靠性。使用总存储开销与区块链数据规模的比值衡量空间节约程度。(4) 可验证：节点能够验证所请求的数据是否正确，以防数据被意外或有意篡改。

第三节 存储分片模块设计

4.3.1 初始散布

BSS 初始散布机制为每个新区块选择固定数量的节点来存储与该区块相关的数据，节点选择过程遵循一个类 Kademlia^[66] 的分布式协议。每个节点维护一个节点列表来记录系统内所有其它节点的信息。如图4.5所示，节点列表呈前缀树结构。每个叶子节点对应一个服务节点，记录该节点的 *nodeID*、IP 地址和端口。从树根节点到叶子节点的路径即相应服务节点 *nodeID* 的最短唯一前缀。给定 *blockID* 或 *txID*，使用树状结构的节点列表可以快速找到距离相应区块或交易最近的节点。在图4.5的例子中，给定以“010”为前缀的 *blockID*，我们可以找到距离该区块最近的服务节点为叶子结点“011”和“000”对应的节点。

初始散布流程。在一轮共识完成、一个新区块被确认后，节点执行初始散布。完整的工作流程如下。

- (1) 敷布区块。给定新区块的 *blockID*，节点检索本地节点列表找到距离新区块最近的 r 个节点，亦即该区块的“初始宿主”。若节点是初始宿主之一则存储完整区块数据 $\{\text{blockID}, \text{blockData}\}$ 到本地磁盘；若节点不是初始宿主则只存储区块头数据 $\{\text{blockID}, \text{blockHeader}\}$ 。初始宿主不能够删除初

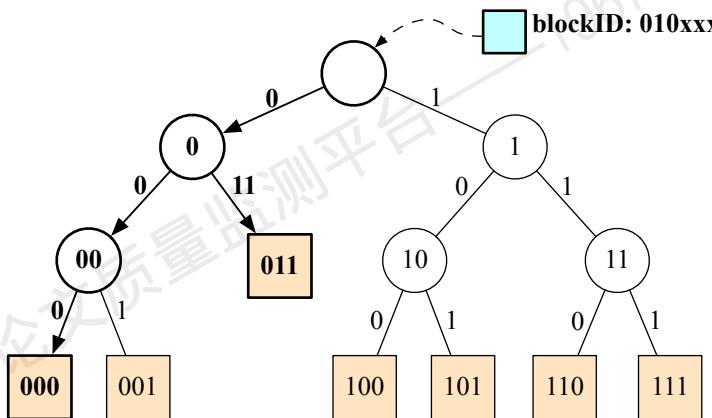


图 4.5 节点列表示意图。

始副本，除非相应区块被切换为编码模式（具体内容见4.3.3小节）。

- (2) 散布交易索引。对于新区块包含的每一笔交易，节点检索本地节点列表找到距离该交易最近的 r 个节点，亦即该交易的“索引宿主”。若节点是索引宿主之一则存储索引项 $\{txID, blockID\}$ 到本地磁盘；否则节点什么都不做。索引宿主不能够删除所存储的索引项。对索引项进行散布是为了支持交易检索功能（具体内容见第四节）。

分析。初始散布机制具有如下优点。(1) 简洁：节点无需记录区块或索引项的存储位置。若需确认初始宿主或索引宿主，只需检索节点列表找到距离该数据最近的 r 个节点。(2) 可靠性：给定初始散布参数 r ，则系统能够容忍 $r-1$ 个节点故障。(3) 存储负载均衡：由于数据被随机散布，各节点存储负载相对均衡。然而，仅仅执行初始散布还不够。由于区块链上不同区块访问热度差异较大，系统内通信负载和数据传输负载不均衡，导致节点间数据访问性能表现不理想。因此，需要根据区块访问热度动态调整不同区块的副本数量，从而在性能方面进行优化。

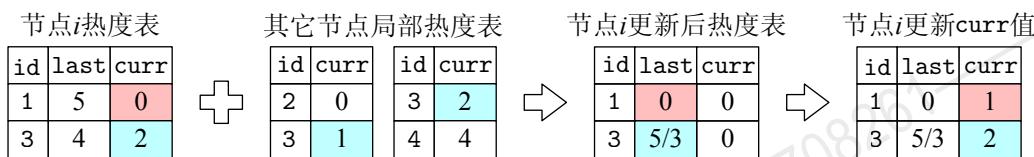
4.3.2 动态复制

BSS 执行动态复制的依据是区块访问热度。每个节点维护一个局部热度表来记录本地保存的所有区块在最近两个纪元(epoch)内的访问频次。图4.6给出热度表更新过程的一个示例(假设存储组中包含3个服务节点)。热度表中 id 指区块序号，变量 $last$ 指区块在上一 epoch 的全局平均访问次数，变量 $curr$ 指区块在当前 epoch 的本地局部访问次数(包含节点对自己的访问)。节点间通过

协作来统一更新 `last` 值，过程如下。

- 1) 在每个 epoch 结束时，每个节点向其它节点广播局部热度表 $\{id, curr\}$ ，即刚刚结束的 epoch 内该节点所统计的一部分区块的局部访问次数。假设共有 n 个节点，则每个节点收到 $n - 1$ 个局部热度表。
- 2) 节点统计本地保存的所有区块的全局总访问次数，并将总访问次数除以节点数得出全局平均访问次数。
- 3) 节点用算出的全局平均访问次数更新 `last` 值，并将 `curr` 置为 0。

自 `last` 值更新完成至下一个 epoch 结束前，节点只更新 `curr` 值（伴随数据访问实时更新）。定义一个区块的实时热度等于 $last + curr$ 。



节点 <i>i</i> 热度表		
<code>id</code>	<code>last</code>	<code>curr</code>
1	5	0
3	4	2

+

其它节点局部热度表	
<code>id</code>	<code>curr</code>
2	0
3	2
3	1

→

节点 <i>i</i> 更新后热度表		
<code>id</code>	<code>last</code>	<code>curr</code>
1	0	0
3	5/3	0

→

节点 <i>i</i> 更新curr值		
<code>id</code>	<code>last</code>	<code>curr</code>
1	0	1
3	5/3	2

图 4.6 热度表更新过程。

请求者复制。每隔固定时间，每个节点从最近向其它节点请求过的区块中选择一部分复制到本地。为执行请求者复制，节点在每次进行数据请求时不仅要求被请求节点返回区块，还要求返回区块当前热度值。以下称正在执行请求者复制的节点为“主节点”。执行过程如下。

- (1) 确定待复制区块。自上一次执行请求者复制后所有被主节点请求过的区块将作为候选区块。对所有候选区块按热度降序排序并选择前 $1/n$ 比例作为待复制区块（假设系统共有 n 个节点）。
- (2) 选择动态宿主。对每个待复制区块，选择尚未保存该区块的节点中距离该区块最近者作为动态宿主。
- (3) 执行复制。主节点通知动态宿主复制对应区块，并向所有节点广播新的复制状态。为追踪全局复制状态，每个节点维护一个复制表 $\{blockID, nodeID\}$ ，即相应区块被动态复制到相应节点。

持有者复制。一旦区块请求聚集在少量节点，系统性能可能会大大降低。因此，若节点发现本地存储的区块访问热度过高，便将该区块复制给其它节点以缓解本地访问负载压力。以下将正在执行持有者复制的节点称为“主节点”。执行过程如下。

- (1) 选择区块。给定系统参数 \hat{p} 和 Δp , 主节点检查热度表选出热度大于 \hat{p} 的区块并将选中区块划分等级: 热度区间在 $[\hat{p}, \hat{p} + \Delta p)$ 的区块级别为 $g = 1$, 热度区间在 $[\hat{p} + \Delta p, \hat{p} + 2\Delta p)$ 的区块级别为 $g = 2$, 热度区间在 $[\hat{p} + 2\Delta p, \infty)$ 的区块级别为 $g = 3$ 。
- (2) 确定副本个数。级别 $g = 1, 2, 3$ 的区块将分别增加 $1, 2, 3$ 个动态副本。
- (3) 选择动态宿主。对于某个待复制区块, 从尚未保存该区块的节点中选择 g 个距离该区块最近者作为动态宿主。
- (4) 执行复制。主节点通知动态宿主复制对应区块, 并向所有节点广播新的复制状态。

每个动态副本的生命周期设置为 T , 即动态副本将在被保存 T 个 epoch 后被删除。删除操作将进行广播以便系统内所有节点更新复制表。

分析。 动态复制机制具有如下优点。(1) 请求者复制使得最近访问过的热区块在系统内副本增加, 提升了热区块的访问性能。(2) 持有者复制使得访问过载节点将通信和数据传输任务及时卸载给其它节点, 避免请求者等待以及网络拥塞。(3) 设置生命周期可以保证系统内动态副本的生成率等于删除率, 使节点存储开销保持稳定。(4) 实验测试结果显示, 热度表的设计带来较大性能提升。相比于节点仅统计本地局部访问次数, 维护全局访问次数可使节点更准确地掌握区块热度变化, 实现精准复制, 从而大幅提升系统性能⁴。不过, 虽然初始散布和动态复制机制保证了数据可靠性和数据访问性能, 节点的存储开销却仍有可优化空间。考虑到在许多区块链系统中不同区块的访问概率差异较大(如较早区块的访问概率较低而较新区块的访问概率较高), 我们可以对较少访问的区块进行数据压缩从而进一步降低节点存储开销。

4.3.3 存储优化

BSS 存储优化将最近较少访问的历史区块由副本态切换为编码态。区块编码由被编码区块的初始宿主合作完成, 过程基于 Reed-Solomon (RS) 编码。编码参数 (k, m) 表示利用 k 个冷区块生成 $k+m$ 个块, 且使用任意 k 个块即可恢复初始的 k 个冷区块^[39]。对某个区块, 规定其编码“主节点”为所有初始宿主中距离该区块最近者。每个节点持续监测其所主导编码的区块, 若存在连续 $2T$ 时

⁴虽然热度表中对全局访问次数 `last` 的更新涉及节点广播, 但这一过程并不会占用过多分网络资源。这是因为: 1) BSS 系统规模较小(服务节点个数为数十个), 所以节点广播是可行的; 2) 热度表的数据规模较小(为 KB 级别), 所以广播时传输的数据量较少。

间内未被访问⁵的区块则将其标记为“冷区块”并放入编码缓存区⁶。每个节点维护一个编码表来追踪被编码区块的信息。

编码过程。若某节点编码缓存区内包含 k 个被存储在不同节点的区块，节点启动编码过程。图4.7(a)给出一个示例。假设初始散布参数 $\hat{r} = 3$ 且 RS 码参数 $(k, m) = (4, 2)$ 。节点 a 编码缓存区内区块 $2, 5, 8, 9$ 刚好被存储在 4 个不同的节点 b, c, d, e ，于是节点 a 启动编码。具体步骤如下。

- (1) 节点 a 对区块 $2, 5, 8, 9$ 使用 RS 码，计算得到校验块 x, y 。
- (2) 节点 a 通知节点 b, c, d, e 执行区块删除，以使得删除完成后每个节点刚好存储一个不同的区块：节点 b 存储区块 2 ，节点 c 存储区块 5 ，节点 d 存储区块 8 ，节点 e 存储区块 9 。
- (3) 节点 a 将校验块 x 存储在本地，然后选择一个新的节点 f 存储校验块 y 。
- (4) 节点 a 广播新的编码状态 $\{\text{block } 2, 5, 8, 9 \rightarrow \text{node } a, f\}$ ，即区块 $2, 5, 8, 9$ 被作为一组进行编码且生成的校验块被存储在节点 a, f 。每个节点接收到该广播信息并更新编码表。特别地，节点 a 无需广播区块 $2, 5, 8, 9$ 的存储位置，因为这四个区块都被存储在距离它们第二近的节点上。

需要强调，编码主节点必须等到编码缓存区内 k 个区块刚好被散布在 k 个不同节点时再启动编码，否则系统的容错能力会降低。图4.7(b)给出一个反例。区块 $2, 5, 8, 9$ 被散布在 3 个节点 c, d, e ，编码主节点执行编码后节点 c 负责存储区块 $2, 5$ 。假设某个时刻节点 c, d 崩溃，则由于可用数据块不足 4 个，区块 $2, 5, 8, 9$ 无法还原。此时系统无法容忍 2 节点故障。一个解决办法是在编码后将区块 2 或区块 5 传输给节点 d 保存，但这就打破了编码态节点存储在第二近区块的规则，因此也不可取。

分析。存储优化机制具有如下优点：(1) 编码结束后，冷区块在系统内只有一个副本，并与其它冷区块共享若干校验块，降低了系统存储开销。(2) 得益于 RS 码的特点，尽管存储开销降低，数据可靠性水平并未降低。(3) 编码过程和删除过程都很简洁。(4) 编码态区块在系统内的唯一副本存储在距离其第二近的节点上，因此无需额外记录其存储位置。编码态区块的访问时延会略大于副本态区块，但这并不会对系统整体性能造成较大影响。

⁵此处“未被访问”不仅包括未被其它节点访问过也包括未被自己访问过。

⁶在第一个 T 时间内，存在一种情况，就是该区块的动态宿主本地访问了该区块，因此无需向初始宿主进行请求。在第二个 T 时间内，我们知道所有动态宿主已经执行了区块删除，只有初始宿主保存有该区块。因此，如果在在第三个 T 时间内仍未发生区块访问，我们可以认定的确没有节点访问该区块。

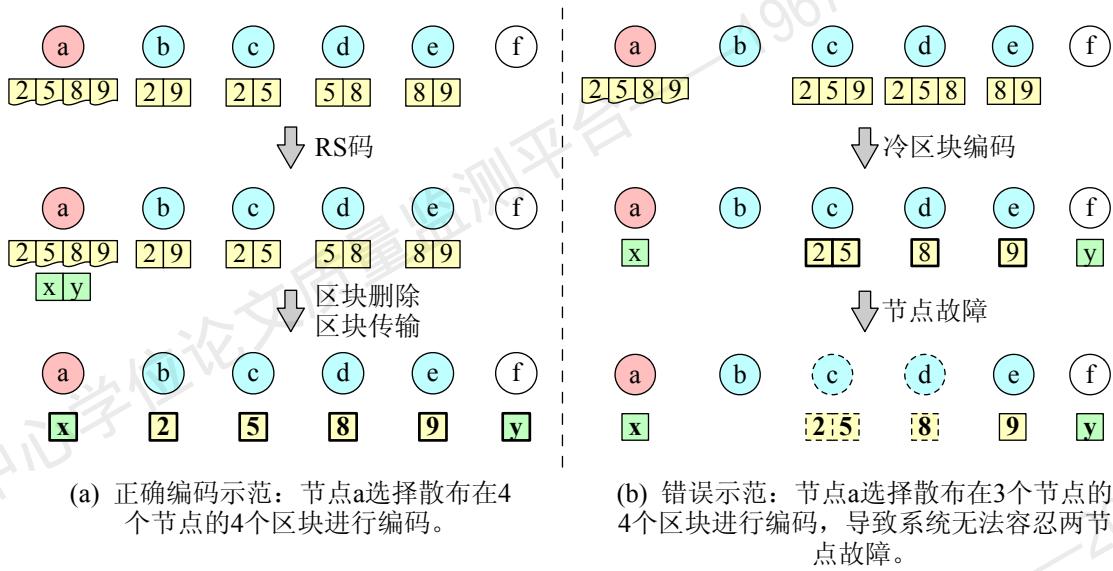


图 4.7 存储优化机制：区块编码的正例与反例。

第四节 服务节点数据请求

本小节介绍服务节点如何向存储组内的其它服务节点请求区块或者交易。4.4.1节给出无节点故障状态下请求区块或交易的过程，4.4.2节考虑节点故障及数据恢复。4.4.3节探讨新节点加入系统或节点退出系统所引发的数据传输问题。

4.4.1 无节点故障时的数据请求

若某节点请求一个区块，第一步需要检查编码表以确认目标区块是否处于编码态。若目标区块处于副本态，节点向目标区块的若干个初始宿主或动态宿主发送消息，请求过程如算法4.1所示。首先，请求节点向目标区块的初始宿主发送消息（第1-5行），并等待 Δw 时间。若等待结束仍未收到初始宿主发来的回复，则请求节点确认目标区块是否有动态宿主，若有则向 $\leq \hat{r}$ 个动态宿主重新发送请求（第14-17行），并再等待 Δw 时间。若等待结束仍未收到回复，则请求节点向更多的动态宿主发送消息直到收到回复。另一种可减少等待的请求模式是，请求节点一次性向所有保存目标区块的节点发送消息，并在接收到目标区块后通知所有节点以免重复发送。这一方法可能会导致系统内消息数量过大，因此比较适合节点数较少的BSS系统。接收到目标区块后，请求节点将执行区块验证过程（第7-12行）。节点首先验证区块头，然后验证区块体。区块验证通

算法 4.1 区块请求算法

```

1 procedure Request(id)
2   H  $\leftarrow$  Host(id)                                 $\triangleright$  目标 blockID: id
3   for host  $\in$  H                             $\triangleright$  查询节点列表
4     ip, port  $\leftarrow$  getInfo(host)
5     sendReq(id, ip, port)                          $\triangleright$  发送区块请求
6
7 procedure Validate(bi)
8   header, body  $\leftarrow$  parBlk(bi)            $\triangleright$  接收到区块: bi
9   if valHead(header)                            $\triangleright$  验证区块头
10  if valBody(body)                             $\triangleright$  验证区块体
11    return success
12  return fail
13
14 procedure Re-Request(id)                    $\triangleright$  等待  $\Delta w$  时间后执行
15   H  $\leftarrow$  adptHost(id)                   $\triangleright$  查询复制表
16   for host  $\in$  H
17     sendReq(id)                             $\triangleright$  向动态宿主发送请求

```

过，则本次数据请求成功。

如果节点请求一笔交易，操作流程会稍稍复杂，具体如下。第一步，节点向索引宿主获取索引项 {txID, blockID}。具体方法与算法4.1所示程序 Request() 类似。接收到索引项 {txID, blockID} 之后，请求节点使用 blockID 再次查询节点列表和复制表，获取该区块的初始宿主和动态宿主，并向宿主发送消息。宿主在返回交易内容的同时还会返回交易所对应的默克尔证明以证明该笔交易的有效性。对默克尔证明的验证方法如下：节点使用接收到的交易和默克尔证明计算得到默克尔根，然后检索本地数据库找到交易对应的区块头中的 root Hash 字段，最后验证本地保存的 root Hash 与计算得到的默克尔根是否一致，若一致则验证通过。交易验证通过，则本次数据请求成功。

4.4.2 节点故障及数据恢复

本小节考虑节点出现瞬时故障的情况。假设发生故障的节点在一段时间后会恢复正常并且其负责存储的数据会再次可用。给定初始散布参数 \hat{r} ，对于副本态区块，若少于 \hat{r} 个节点出现故障则区块请求过程不会受到影响。对于编码态区块，当出现节点故障时需要进行数据恢复。下面主要讨论编码态区块的故障恢复。如图4.8所示，在节点 *a* 对区块 2, 5, 8, 9 执行编码得到校验块 *x, y* 后，某个

时刻节点 b 故障。节点 c 访问区块 2 失败，于是发消息通知其它节点。作为区块 2 的编码主节点，节点 a 在接收到通知后启动对区块 2 的数据恢复。节点 a 向节点 c, d, e, f 请求区块 5, 8, 9, y 。只要四个节点中有三个节点返回对应的区块，则节点 a 可以成功恢复出区块 2。恢复成功之后，节点 a 会在节点 b 再次可用之前代为存储区块 2，并向其它所有节点广播自己暂时存储区块 2。

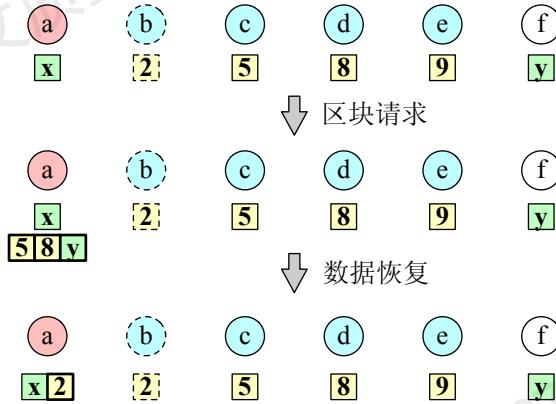


图 4.8 区块链数据结构。

4.4.3 节点加入及退出

当新的服务节点加入存储组或现有服务节点退出存储组时，需要根据初始散布规则重新调整区块副本或者交易索引副本的存储位置。

若有新的服务节点加入系统，需要进行数据同步。作为第一步，新节点需要先获取系统内所有节点的信息并建立节点列表。然后，新节点向多个节点并行请求以获取全部有效区块头。最后，新节点向其它节点请求复制表和编码表。同步结束。新节点的加入会导致一部分区块和交易索引的初始宿主发生改变，但是新节点无需主动同步相关数据，除非遇到以下两种情况：1) 与新节点连接的轻节点想要请求某个区块或者交易，而相关数据理应由新节点保存；2) 有其它节点向新节点请求某个区块或者交易索引，而相关数据理应由新节点保存。这时，新节点执行第四节所述的数据请求过程，将相应数据返回给轻节点或服务节点，并本地保存该数据。

若有服务节点想要退出系统，退出节点必须将负责保存的所有初始副本传输给其它服务节点进行保存。具体地，对于所有需要传输的区块或者交易索引，节点将数据迁移至尚未保存该数据且距离其 ID 最近的节点。退出节点必须在完成全部传输任务之后再退出系统。

第五节 存储分片模块实验测试

在本小节中，我们针对存储分片模块 BSS 进行三组实验测试。（1）存储测试：测试服务节点存储开销的降低程度；（2）性能测试：测试节点之间执行数据访问所需花费的时间开销，尤其是节点发生故障的情况下时间开销的上升幅度；（3）自适应性测试：测试 BSS 动态复制策略是否能够根据实时区块访问热度复制当前较热区块。特别地，我们将 2020 年提出的动态区块散布方案 BFT-Store^[162] 以及 2018 年提出的静态区块散布方案 CUB^[157] 作为测试基准，从性能角度将二者与 BSS 进行对比，相关实验结果在 4.5.2 节和 4.5.3 节给出。

4.5.1 实验设置

区块链设置。参考真实 Bitcoin 数据模拟包含 400 个区块的区块链。具体地，使用 BigQuery^[214] 提供的 Bitcoin 数据库，截取高度 654000 至 654399 的区块大小作为模拟区块链中区块 0 至区块 399 的大小。

节点设置。在 PC 机上模拟 10 个服务节点的运行情况。PC 机的硬件配置为 Intel(R) Core™ i5-8265U 处理器和 8GB RAM，操作系统为 Windows 10。根据正态分布设置节点间通信开销。具体地，对任意两节点，从正态分布 $N(1, 1)$ 中随机选择大于 0 的值作为其通信开销，单位是 s/MB。

访问分布。使用 Zipf 分布来模拟区块链上各个区块的访问概率分布。具体地，对于一次区块访问，一个排名为 i 的区块被访问到的概率为

$$AP_i = \frac{1}{\sum_{j=1}^{\ell} (\frac{1}{j})^s \cdot j^s}, \quad (4.1)$$

上式中 ℓ 表示区块链长度而 s 为预定义的常量，区块排名 $i = 1, 2, \dots, \ell$ 。可以看出，排名越小的区块被访问到的概率越大。伴随区块链增长，同一个区块的排名在不断变化，因此其访问概率也在不断变化。关于如何对区块排名，考虑两种方法。一种方法将较小的 i 值分配给高度更高的区块，使得更新的区块有更大的访问概率。由此得到的访问分布称为“Zipf 分布”。在以 Bitcoin^[13], NIPoPoWs^[150] 和 FlyClient^[151] 为代表的基于 SPV 的区块链中常见 Zipf 分布。在 SPV 机制下，当一个轻节点想要验证一笔交易时，会要求全节点遍历包含该笔交易的区块并计算交易的存在性证明。由于较新的交易有较大的概率被验证，所以较新的区块有较大的概率被访问。另一种排名方法是使高频访问区块分布在区块链上的随机位置。由此得到的访问分布称为“随机 Zipf 分布”。在以

Filecoin^[22] 和 Storj^[142] 为代表的 p2p 存储系统中常见随机 Zipf 分布。在这类存储系统中，文件元数据保存在区块链上。一旦某个文件被访问，则链上相应的元数据也会被检索。由于较早上传到存储系统的文件仍有可能被频繁访问，因此高频访问区块会出现在区块链的随机位置。根据实验结果，BSS 系统在采用 Zipf 分布和随机 Zipf 分布时各方面表现非常接近，因此下文给出的大部分实验结果只展示 Zipf 分布的情况。设置 Zipf 分布系数 $s = 1.2$ 。

随机区块访问。使用泊松过程模拟节点随机访问区块的过程。令 k 表示一个服务节点在一个 epoch 内发起的区块访问个数，则 k 的值服从泊松分布

$$P(k) = \frac{\lambda^k}{k!} e^{-\lambda},$$

其中 $k = 0, 1, \dots, \hat{k}$ 。参数 λ 的值等于 k 的值的期望，因此 λ 决定了各个节点的平均访问负载。在下文所有实验中设置 $\lambda=10$ 或 20 , $k=25$ 。

BSS 实验设置。模拟两个 BSS 系统，其中 BSS⁽³⁾ 运行三副本初始散布和动态复制，BSS⁽²⁾ 在 BSS⁽³⁾ 的基础上加入存储优化且编码后的区块保留 1 个副本。这两个系统都具有 2 节点容错能力。表格4.1给出了系统实现的相关程序，具体执行流程如下。BSS⁽³⁾ 系统：第 h 个 epoch 开始后每个节点执行 NewBlock(b) 和初始散布 InitAlloc(b, \hat{r})；执行持有者复制 OwnerR($p, \Delta p$) 以及随机区块访问 RqstSet(λ, K) 和 BlockRqst(R)；执行请求者复制 RqstR(n) 和区块删除 Deletion(T)。BSS⁽²⁾ 系统：在 BSS⁽³⁾ 的基础上，执行完区块删除后，每个节点在满足编码条件时执行 Encode(k, m) 并删除 1 个初始副本。实验开始前的初始状态设置为：区块链上有 10 个区块（区块 0 到 9），其中每个区块被随机分配给 3 个初始宿主⁷。实验过程中模拟从 epoch 10 到 epoch 399 的动态过程。

BFT-Store 实验设置。BFT-Store 是一个动态区块散布方案，采用了纠删码与副本结合的存储模式。纠删码的设计在低存储开销的前提下保证容错性，副本的设计可以提升数据访问性能。我们模拟两个 BFT-Store 系统，其中系统 BFT⁽²⁾ 使用 (8, 2) 的 RS 编码和二副本策略，与 BSS⁽²⁾ 进行对比；系统 BFT⁽³⁾ 使用 (8, 2) 的 RS 编码和三副本策略，与 BSS⁽³⁾ 进行对比。BFT⁽²⁾ 和 BFT⁽³⁾ 系统都能够容忍 2 个节点出现拜占庭错误⁸。假设每个 BFT-Store 节点存储全部区块头，用

⁷ 设置如上初始状态是因为在使用公式4.1为每个节点生成访问集合时区块链长度 ℓ 必须大于 0。初始状态下，所有节点本地维护的区块列表中都包含 10 个条目，热度表中平均有 3 个条目。由于尚未执行区块访问所以区块热度均为 0。

⁸ 虽然 BFT⁽³⁾ 比 BFT⁽²⁾ 多了一份区块副本，但系统容错能力不变。这是因为增加的副本被散布给了相同的 10 个节点。

表 4.1 BSS 系统实现的主要程序。

程序名	描述
NewBlock(b)	更新区块链。向区块列表中添加新区块 b 。
InitAlloc(b, \hat{r})	初始散布。参数 \hat{r} 表示初始宿主的个数。程序输出初始宿主的 $nodeID$ 。
RqstSet(λ, \hat{k})	获取访问集合。根据泊松分布从数值区间 $[0, \hat{k}]$ 随机选出一个值 k , 根据 Zipf 分布或随机 Zipf 分布从区块链上随机选出 k 个区块。程序输出当前 epoch 内将要访问的区块集合 R 。
BlockRqst(R)	区块请求。对于在集合 R 中且未在本地保存的区块, 向其初始宿主和动态宿主发送请求。
RqstR(n)	请求者复制。给定节点在本 epoch 请求过的区块, 从中选出 $\lceil \frac{1}{n} \rceil$ 比例的区块复制在本地。
OwnerR($\hat{p}, \Delta p$)	持有者复制。程序检查热度表并选择热度超过 \hat{p} 的区块。根据热度的不同, 被选中的区块将被复制 1 次、2 次或 3 次。
Deletion(T)	副本删除。参数 T 表示动态副本生命周期, 设置 $T = 3$ 。
Encode(k, m)	区块编码。程序以 k 个冷区块为一组进行编码并生成 m 个校验块。

于在数据请求阶段验证获取到的区块是否正确⁹。实验测试中, 每隔 8 个 epoch 执行一次区块散布, 过程如下: 以 8 个区块为一个编码组生成 2 个校验块, 将这 10 个块统一散布给系统内的 10 个节点, 且每个块在系统中有 2 或 3 个副本。图4.9给出系统 BFT⁽³⁾ 的散布过程示例。在执行区块请求时, 我们假设每个节点都会保存尚未执行散布的区块, 因此节点只请求已经散布完成的区块。

4.5.2 测试结果

存储测试。 BFT-Store 系统的存储开销是可以计算出来的。对于 BFT⁽²⁾, 每 8 个区块生成 $2 * (8 + 2) = 20$ 个区块副本, 因此复制度 = $20/8 = 2.5$ 。对于 BFT⁽³⁾, 每 8 个区块生成 30 个区块副本, 复制度 = $30/8 = 3.75$ 。对于 BSS 系统, 我们通过实验测试得出各节点的平均存储开销。在每个 epoch 内, 执行完区块删除或是区块编码后, 统计系统内各节点存储开销, 再将所有节点总存储开销除以区

⁹原论文的设计里并没有这个假设, 但在这里我们对 BFT-Store 进行优化, 以便它与 BSS 方案“站在一起跑线上”。存储所有区块头的好处是, 节点在区块请求阶段能够判断获取到的区块是否正确。如果不存储区块头, 则节点必须从不同节点处获取多个副本, 从而通过对比来确认正确的区块。例如, 当系统内存在 2 个拜占庭节点时, 必须获取到 3 个相同的副本才能够确认该数据是正确的。在原论文里, 使用参数为 (k, m) 的 RS 编码时方案只能容忍 $m/2$ 个节点发生拜占庭错误。而增加了存储区块头的假设之后, 使用参数 (k, m) 的 RS 编码时方案能够容忍 m 个拜占庭节点。在存储开销不变的前提下, 容错能力提高了。

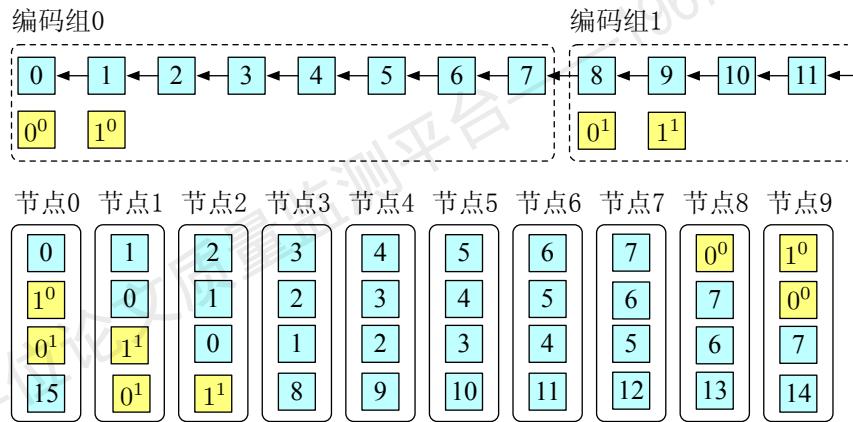


图 4.9 BFT-Store 区块散布示意图。

块链总数据量得到平均复制度 (ARD, averaged replication degree)。我们测试了 1000 个 epoch 内的 ARD 值，并每隔 20 个 epoch 取一个数据点，得到实验结果如图4.10所示。可以看到，BSS⁽³⁾ 的 ARD 值逐渐趋近于 3.0 而 BSS⁽²⁾ 的 ARD 值逐渐趋近于 2.5。将 BFT-Store 和 BSS 进行对比我们发现：1) BFT⁽³⁾ 和 BSS⁽³⁾ 都是三副本的设置，不过前者的复制度比后者高了约 0.75；2) BFT⁽²⁾ 和 BSS⁽²⁾ 都是二副本的设置，两者的复制度几乎相同。经计算，BSS⁽²⁾ 系统测得的 ARD 平均值为 2.9，即每个 BSS 节点平均存储的区块链数据比例为 $2.9/10 = 29\%$ 。与全副本存储模式相比，每个 BSS 节点所需要存储的数据量降低了约 71%。

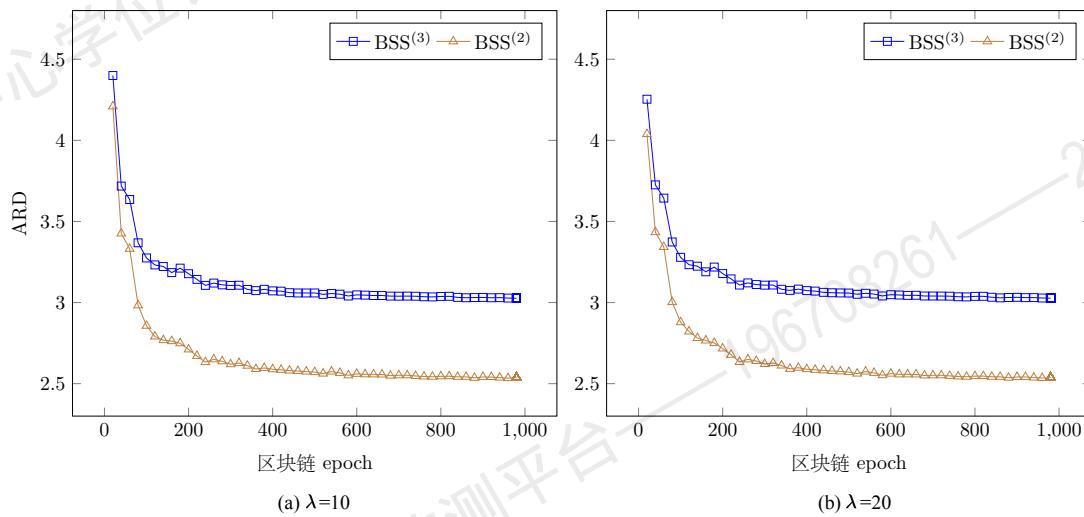


图 4.10 BSS 存储测试结果

性能测试。 BSS 性能测试方法如下：在每个 epoch 内，执行完随机区块请求后，每个节点分别计算请求一个区块所需平均时间，再将 10 个节点的计算结果取平均值，得到平均区块请求时延（ARL, averaged request latency）。BFT-Store 性能测试方法如下：在每个 epoch 内，若当前 epoch 序号为 8 的倍数则先执行区块散布再执行 Zipf 随机区块请求，否则直接执行随机区块请求；统计每个节点每次访问所花费的时间，并计算出 ARL 值。测试 400 个 epoch 内的 ARL 值，并每隔 5 个 epoch 取一个数据点，得到实验结果如图4.11所示。首先我们看到， $BSS^{(2)}$ 和 $BFT^{(2)}$ 都采用了纠删码加二副本的混合散布策略，复制度也几乎一样，但前者的性能明显优于后者，这是因为 BSS 只对冷区块进行编码，热区块仍旧为副本态，而 BFT-Store 对所有区块都采取了相同的混合散布策略。再者，虽然 $BSS^{(3)}$ 系统的存储开销比 $BFT^{(3)}$ 低，但前者的性能比后者更优。这是因为 BSS 能够根据区块热度动态调整区块副本数量，从而大幅提升访问性能。

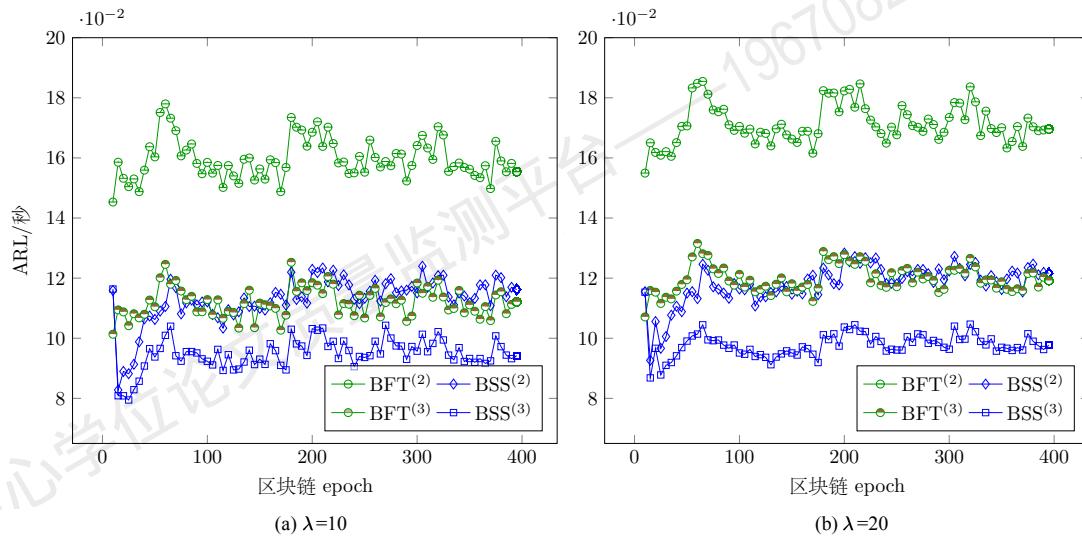


图 4.11 性能测试结果

故障测试。 我们模拟两类故障：crash 故障指节点接收请求后不返回任何数据，byzantine 故障指节点故意返回错误数据。针对 crash 故障设置最大等待时长，若请求节点等待到设定时间后未收到回复则认为被请求节点为 crash 节点。针对 byzantine 故障，节点在接收到区块后利用本地存储的区块头执行数据验证，以确认接收到的数据是否正确。我们针对 $BSS^{(2)}$ 和 $BFT^{(2)}$ 进行节点故障下的性能测试。这两个系统都采用了编码加二副本的散布策略。当系统中有 1 个节

点发生故障时不需要执行数据恢复；当系统中有 2 个节点故障时，若被请求区块的 2 个副本的存储节点刚好均遭遇故障，则请求节点需要执行数据恢复。设置两组实验，测试节点故障下的数据请求性能。第 1 组实验模拟 1 节点故障，并测试连续 400 个 epoch 下的 ARL 值（测试方法性能测试相同），实验结果如图 4.12(a) 所示。对比无故障下的测试结果，BSS⁽²⁾ 和 BFT⁽²⁾ 的 ARL 都有一定程度的增加。第 2 组实验模拟系统中有 2 个节点发生故障的情况，测试连续 400 个 epoch 下的 ARL 值，实验结果如图 4.12(b) 所示。我们还测试了节点单次执行数据恢复的平均时间开销，发现 BSS 单次数据恢复的时间开销稍大于 BFT-Store¹⁰。从图 4.12(b) 可以看到 BSS⁽²⁾ 测得的 ARL 上升幅度比 BFT⁽²⁾ 更小。这是因为在 BSS⁽²⁾ 中少数区块处于编码态（只有 2 个副本），多数区块处于副本态（有不少于 3 个副本），而 BFT⁽²⁾ 中全部区块都只有 2 个副本。在两节点故障的情况下，BSS⁽²⁾ 系统执行数据恢复的次数更少，因而对数据请求性能的影响更小。

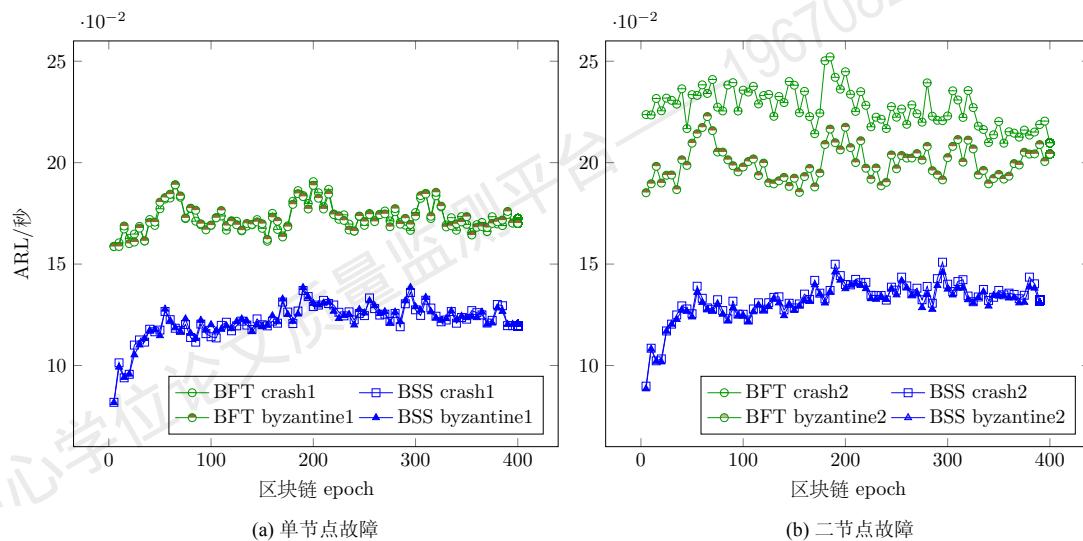
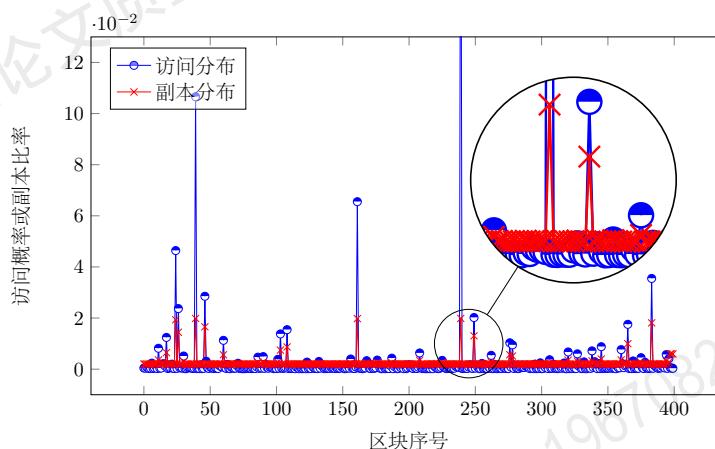


图 4.12 节点故障下的性能测试结果

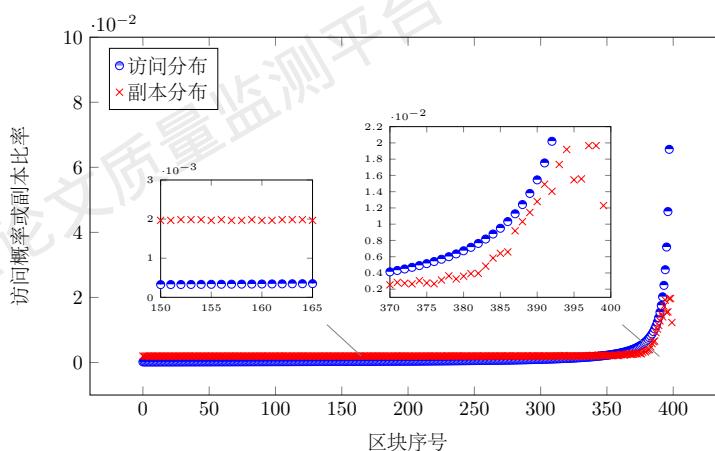
自适应性测试。自适应性指系统内区块副本的分布情况与区块访问概率分布吻合。本实验用以评估动态复制过程是否及时将热区块的副本数增加，以使得区块副本分布的变化与区块访问概率变化吻合。本实验针对 BSS⁽³⁾ 系统进行测试。在第 399 个 epoch 执行结束后，统计各区块的副本个数，并将区块副本个

¹⁰ 在 crash2 故障下，BFT 测得值为 2.760 而 BSS 测得值为 2.849；在 byzantine2 故障下，BFT 测得值为 1.103 而 BSS 测得值为 1.269

数除以副本总数得到各区块的副本比率。将副本比率曲线与两种访问分布曲线进行对比，得到测试结果如图4.13所示。图中x轴代表区块序号，y轴代表区块的访问概率或者副本比率。可以看到在Zipf和随机Zipf两种访问概率分布下，副本分布与访问分布都吻合良好。这说明动态复制过程较为精准地复制了当前访问频次较高的区块。



(a) Zipf 分布



(b) 随机 Zipf 分布

图 4.13 自适应性测试结果

4.5.3 CUB 性能测试结果

本实验用以评估CUB静态散布策略在性能方面的表现，并与BSS系统进行对比。我们实现了CUB中所提出的基于贪心算法的静态散布策略，并对其进行性能测试。

本部分实验中的区块链设置、节点设置与 BSS 实验相同。所不同的是对区块访问分布的设置。CUB 实验开始前需要为每个节点预先构建频繁访问集合 (FAS, frequently access set)。该集合给出节点在实验过程中访问的区块及每个区块的访问次数。根据原文实验部分的描述^[157]，我们采用如下方法构建 FAS：给定 FAS 包含的区块个数 f ，为每个节点从 400 个区块中随机选择 f 个不同的区块加入其 FAS，并将 FAS 中每个区块的访问次数设为 1。参数 f 的作用类似于 BSS 实验中的参数 λ ，其决定了实验过程中的总体访问负载大小。在实验过程中每个节点执行 f 次区块访问，系统共有 10 个节点，因此总访问次数为 $10f$ 。另外，由于贪心算法要求输入节点存储空间上限，我们将每个节点的存储空间上限设置为区块链数据总量的五分之一。

实验测试过程如下。给定各节点 FAS 及存储空间上限，执行贪心算法完成区块散布。然后，依照 FAS 执行区块访问，并统计每个节点每次访问所花费的时间。访问结束后，计算各节点执行区块请求所花费的平均时延，再计算总体平均区块请求时延得到 ARL。图4.14展示了 7 种不同 f 值下所得到的 ARL 值。我们将这一结果与 BSS⁽²⁾ 系统的性能测试结果进行对比。首先我们看到，CUB 测得的 ARL 值伴随 FAS 规模（即系统总访问负载）呈线性增长。作为对比，BSS 中 $\lambda = 20$ 时的性能表现与 $\lambda = 10$ 时的性能表现相比变化不大。另外，虽然 CUB 贪心算法得出的散布结果接近于最优解，但由于缺乏动态复制过程导致 CUB 静态散布的性能表现不佳。当 FAS 大小设置为 $f = 400$ 时，CUB 系统总访问次数为 4000，这与 BSS⁽²⁾ 系统 ($\lambda = 20$) 中 20 个 epoch 所执行的总访问次数相近。然而，CUB 所测得 ARL 是 BSS⁽²⁾ 测得 ARL 的 6 倍以上。

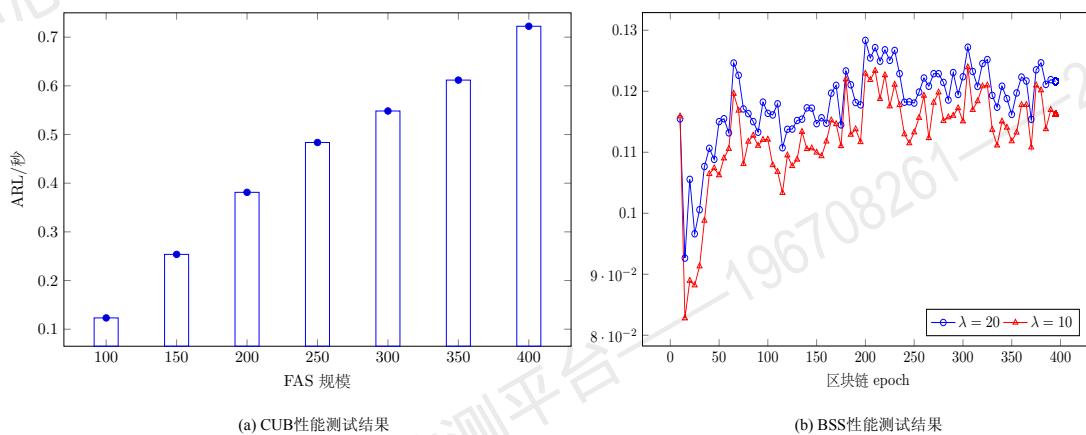


图 4.14 CUB 静态散布策略的性能测试结果对比 BSS 动态散布策略的性能测试结果。

第六节 审计区块链系统实验测试

在本小节中，我们基于以太坊构建一个审计区块链原型系统，并从节点存储开销、数据查询性能等方面对该原型系统进行测试。

4.6.1 审计区块链系统构建

联盟链搭建。使用 Truffle^[207] 测试框架及 Geth^[215] 客户端搭建本地区块链测试链并部署、测试智能合约。测试链出块间隔设置为 5 秒。在审计区块链系统内部署 60 个用户和 1 个云服务商。用户分为 10 个服务节点和 50 个轻节点，其中每个服务节点连接 5 个轻节点。

审计协议实现。使用第三章介绍的 zk-PoR^{*} 协议执行审计。审计合约伪代码见算法 4.2。该协议包含四个步骤：可信设置及文件上传，挑战阶段，回复阶段，验证阶段，执行流程如下。

- (1) 文件上传。我们为每个用户上传 10 个相同大小的文件到云端，使得每个用户的财富值相等。因此，每个用户在每次审计者选举中获胜的概率相等¹¹。
- (2) 随机挑战生成。使用泊松过程模拟用户作为挑战者触发合约生成挑战的过程。令 α 表示每个 epoch 内生成的挑战个数，则 α 的值服从泊松分布 $P(\alpha) = \frac{A^\alpha}{\alpha!} e^{-A}$ （其中 $\alpha = 0, 1, \dots, \hat{A}$ ）。参数 A 的值等于参数 α 的值的期望，因此 A 决定了审计频率。在实验中我们设置 $A = 5$ ，即平均每个 epoch 有 5 个挑战生成，亦即平均每 1 秒生成一个挑战。
- (3) 固定频率回复。云服务商每隔 1 秒调用一次合约以获取一个待回复的验证，检索被审计数据以计算零知识证明，然后再次调用合约以提交零知识证明。
- (4) 自动化验证。审计合约在接收到云服务商提交的零知识证明后，会调用验证合约验证零知识证明。验证合约使用 snarkjs 库根据零知识证明电路自动生成。

随机查询。根据泊松过程模拟用户随机查询链上审计记录的过程。令 k 表示在一个 epoch 内发起查询的用户个数，则 k 的值服从泊松分布

$$P(k) = \frac{\lambda^k}{k!} e^{-\lambda},$$

其中 $k = 0, 1, \dots, \hat{k}$ 。参数 λ 的值等于 k 的值的期望，因此 λ 决定了各个节点的

¹¹根据第三章第三节给出的内容，用户在审计者选举中获胜的概率与其拥有的财富值正相关。若所有用户财富值相等，则他们被选为审计者的概率也相等。

算法 4.2 审计合约

```

1 procedure Upload( $I^F, R_F, n$ )       $\triangleright I^F$  为文件标识符,  $R_F$  为默克尔根,  $n$  为文件块数
2    $userAddress, fileName \leftarrow I^F$ 
3   Files.Push( $userAddress, fileName, R_F, n$ )           $\triangleright$  Files 为文件信息列表
4
5 procedure Challenge( $time, addr, nonce, N$ )
6    $f \leftarrow \text{keccak256}(time, addr, nonce) \bmod N$ 
7    $nonce \leftarrow nonce + 1$ 
8    $F \leftarrow \text{FindFile}(f)$                                  $\triangleright$  选中文件  $F = \{I^F, R_F, n\}$ 
9    $u \xleftarrow{R} \{1, \dots, n\}$                              $\triangleright$  随机选中 1 个文件块
10  ChallengeList.Append( $C = (I^F, u, R_F)$ )            $\triangleright F = \{I^F, R_F, n\}$ 
11
12 procedure ReturnChallenge()                       $\triangleright$  返回一个待回复的挑战
13   for  $0 \leq i < \text{ChallengeList.length}()$ 
14     if  $\text{ChallengeList}[i].response = 0$ 
15     return  $\text{ChallengeList}[i]$ 

```

平均访问负载。在实验中设置 $\lambda = 20$, $k=25$ 。我们模拟三种用户查询方式:

- Query1: 查询某段时间内某个用户指定文件的审计记录。设置区块查询范围为最多 100 个区块。
- Query2: 查询某段时间内某个用户所有文件的审计记录。设置区块查询范围为最多 9 个区块。
- Query3: 查询某段时间内系统所产生的全部审计记录。设置区块查询范围为 1 个区块。

查询合约伪代码见算法4.3。实验过程中,每一次随机查询将随机选中 Query1、Query2、Query3 中的任意一种发起查询。不过,由于用户审计记录在区块链上的分布是随机的、不均匀的,某些查询的返回结果可能为空,这是因为所查询的区块范围内并不包含该用户的相关审计记录。也正因为该情况,每个 epoch 内实际发生的区块访问数量会比 λ 值要降低。因此在下面的实验中,我们不再测试 $\lambda = 10$ 的情况。

4.6.2 测试结果

本部分的测试方法与第五节类似。我们共设置三组实验,分别测试无节点故障下的访问时延,有节点故障下的访问时延,以及节点存储开销。

性能测试。与4.5.2小节的模块测试类似,我们分别测试了审计联盟链系统在

算法 4.3 查询合约

```

1 procedure Check( $U, I^F, b_1, b_2$ )            $\triangleright U$  为用户地址,  $b_1, b_2$  为查询的起始块和终止块
2    $blocks = []$ 
3   for  $block$  in  $[b_1, b_2]$ 
4     if  $1 \leftarrow \text{isContain}(block, U, I^F)$ 
5        $blocks.\text{Append}(block)$ 
6   return  $blocks$ 
7
8 procedure Check( $U, b_1, b_2$ )
9    $blocks = []$ 
10  for  $block$  in  $[b_1, b_2]$ 
11    if  $1 \leftarrow \text{isContain}(block, U)$ 
12       $blocks.\text{Append}(block)$ 
13  return  $blocks$ 
14
15 procedure Check( $b_1, b_2$ )
16    $blocks = []$ 
17   for  $block$  in  $[b_1, b_2]$ 
18      $blocks.\text{Append}(block)$ 
19   return  $blocks$ 

```

采用 BSS⁽²⁾, BSS⁽³⁾, 以及 BFT⁽²⁾, BFT⁽³⁾ 存储分片策略时, 用户查询历史审计记录时所花费的时间开销。图4.15给出了测试结果。相较于上一节的模块测试结果, 我们发现系统环境测试有两点变化。首先, 系统测试下数据查询时延稍有增加。例如, BFT⁽²⁾ 在模块测试测得最大 ARL 约为 0.19 秒, 在系统测试测得最大 ARL 约为 0.26 秒, 增加了约 36.8%。BSS⁽²⁾ 在模块测试测得最大 ARL 约为 0.13 秒, 在系统测试测得最大 ARL 约为 0.18 秒, 增加了约 38.5%。另外我们发现, 系统测试下不同 epoch 测得 ARL 值的波动变大。表格4.2统计了不同实验测得的 ARL 值的方差, 可以看到系统测试下的方差大了一个数量级。上述两点变化的主要原因在于访问分布的改变。模块测试采用 Zipf 访问分布, 而系统环境测试模拟用户随机查询链上审计记录的行为, 因此数据访问分布并不会如此规整。

表 4.2 性能测试 ARL 方差值统计 ($\times 10^{-5}$)

	BSS ⁽²⁾	BFT ⁽²⁾	BSS ⁽³⁾	BFT ⁽³⁾
模块测试	2.15	3.37	1.64	2.04
系统测试	26.8	60.2	21.6	22.6

第四章 实现存储优化的审计联盟链系统

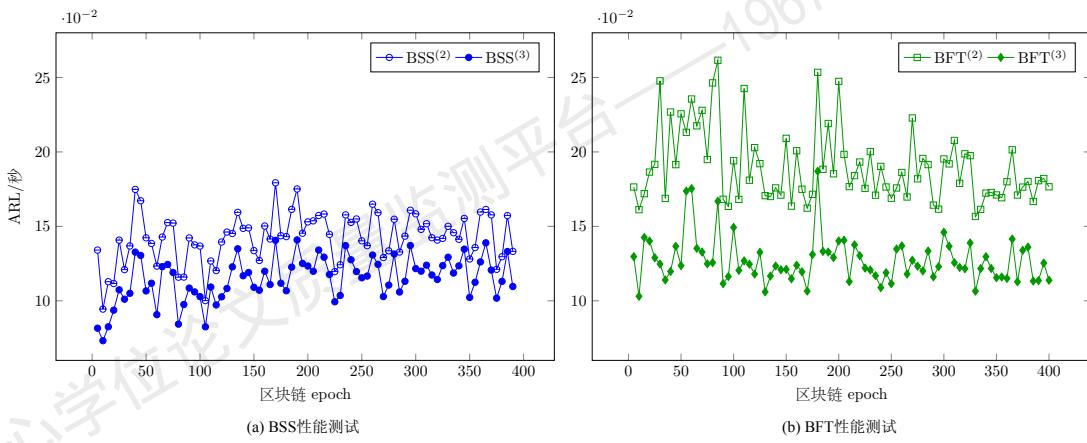


图 4.15 审计联盟链系统数据查询性能

故障测试。图4.16给出了运行 BSS⁽²⁾ 和 BFT⁽²⁾ 的审计联盟链系统存在节点故障时的数据查询时延。从图4.16(a)可以看出，系统内有一个 crash 节点或者一个 byzantine 节点时测得的 ARL 非常相近，这与模块测试结果一致。不过，BFT⁽²⁾ 测得的 ARL 值由有较为明显的波动。从图4.16(b)可以看出，运行 BFT⁽²⁾ 的审计联盟链系统在遭遇双节点故障时测得的 ARL 数值较大且波动性较大。而 BSS⁽²⁾ 测得的 ARL 与单节点故障时相差不大。我们统计了不同实验测得的 ARL 的平均值，如表格4.3所示。统计结果给出了同样的结论。可以看到 BSS⁽²⁾ 在不同故障下的平均值相差不大，但 BFT⁽²⁾ 的变化更为明显。

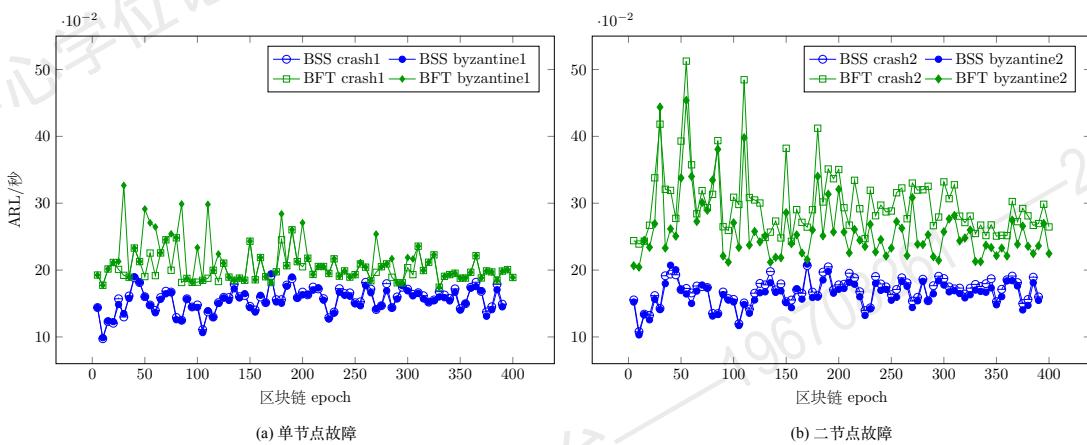


图 4.16 节点故障下的数据查询性能

表 4.3 节点故障下 ARL 值统计

		byzantine1	byzantine2	crash2
$BSS^{(2)}$	模块测试	0.12	0.13	0.13
	系统测试	0.15	0.16	0.17
$BFT^{(2)}$	模块测试	0.17	0.20	0.23
	系统测试	0.21	0.26	0.30

存储测试。图4.17给出审计联盟链系统内服务节点的总体复制率 ARD，计算方法是将各服务节点存储的总数据量除以一个区块链副本的数据量。 $BSS^{(3)}$ 系统运行三副本初始散布和动态复制， $BSS^{(2)}$ 在此基础上额外引入存储优化且编码后的区块保留 1 个副本。与上一节的模块测试相比，系统环境下测得节点总体的存储开销降低了。具体地，模块测试下 $BSS^{(3)}$ 系统 ARD 逐渐收敛到 3 而 $BSS^{(2)}$ 系统逐渐收敛到 2.5。系统测试下两个系统的 ARD 值都降低了约 0.5。这可能是由于访问分布的特点导致。我们统计了每个区块在实验中的平均访问次数，并发现有大约 100 个区块的访问次数为 0（在模块测试中这一情况几乎不会出现）。大量冷区块的存在使得处于编码态的区块数量增加，因此存储复制率降低。这也证明了编码机制在降低节点存储开销方面的效果。

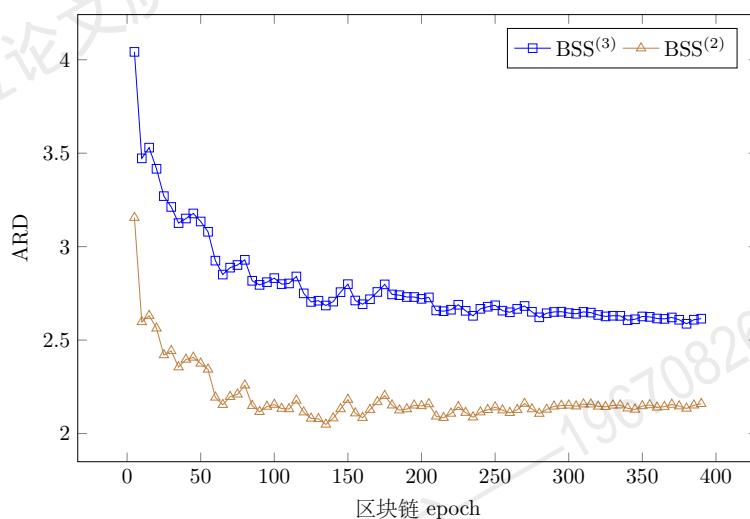


图 4.17 系统内服务节点的存储复制率（总存储数据量/区块链数据量）

第七节 本章小结

本文提出了一种审计联盟链系统，并针对该系统提出一套存储分片方案。本方案包含如下几个创新点。（1）采用审计网络与区块链网络合一的思想，大幅提高审计系统的可实施性和安全性。特别地，结合不同用户拥有资源参差不齐的现状，提出分层的网络架构，在网络中划分出全节点、服务节点和轻节点三种类型。（2）针对所提出的审计联盟链系统，提出存储优化方案，使得服务节点的存储开销得到显著降低，同时不损伤数据可用性。对每个新区块，初始随机散布保证了数据可靠性水平。动态的区块复制过程使得区块副本数量实时弹性变动，以吻合变化着的访问分布。后台执行的区块编码过程使得很少被访问的区块以编码形式存储，降低了存储开销。

存储分片方案在设计过程中的一大难点在于如何保证动态复制过程能够及时将访问频次较高的区块进行复制。为此，我们将区块热度定义为过去一个 epoch 的全局平均访问频次加上当前 epoch 的局部访问频次。为了统计区块热度，节点需要在每个 epoch 结束时执行一次广播。另一大难点在于如何在不牺牲可靠性和访问性能的基础上进一步降低存储开销。为此，我们设计了基于编码的存储优化机制，将最近一段时间内为被访问过的区块成批编码，由副本态切换为编码态进行存储。这一做法可以很好地在访问性能和存储开销之间取得平衡。

对比现有的全副本存储模式，BSS 使得节点仅需存储稍多于 \hat{r}/n 比例的区块链数据（其中 \hat{r} 指系统可靠性水平而 n 指系统节点个数）。相比较于修剪类方法，BSS 提供快速的区块链访问服务。另外，BSS 保证即使 $\hat{r}-1$ 个节点发生故障时数据仍旧可用。

第五章 多云审计系统存储可靠性评价

第一节 引言

使用云存储服务涉及到多方面的安全考虑，比如隐私保护^[216]、容错^[217]、数据完整性^[6]等等。如今云上所存储的数据越来越多，也越来越关乎用户的个人隐私和人身财产安全。这要求云存储服务能够提供更高的安全性、可靠性和隐私性。例如，医疗服务提供商 CareCloud¹将大量电子医疗数据储存在云平台上^[218]。为了保护用户隐私，CareCloud 需要确保存储到云上的医疗数据实现了较强的机密性保护措施；为了保证医疗大数据分析的准确性，CareCloud 需要确保数据的可检索性以及完整性。令人遗憾的是，云存储服务往往不能达到用户期望的安全性。最近几年，许多大型知名云服务商被曝出发生突发性服务故障，范围涉及 Google Docs^[3]，Amazon S3^[5]，Microsoft Azure^[4]，iCloud^[219] 等等。这使得大型企业和机构在使用云服务时存在担忧。

种种现状表明，使用单云存储服务已经无法保证足够的数据安全性。在这样的背景下，多云存储服务逐渐得到越来越多企业和机构青睐。IBM 商业价值研究院与牛津经济研究院的一项全球调查²显示，在 2019 年有 29% 的受访企业在使用单云架构，而在 2021 年这一比例降低到了 3%。多云存储或者混合云存储已经逐渐替代单云存储或者私有云存储，成为众多企业的首要选择。多云存储架构非常适合诸如 CareCloud 这类掌握大量敏感数据的企业或机构。一方面，多云存储系统将数据分散存储到不同的云上，使得每个云服务商只掌握部分用户数据，降低了用户数据泄露的风险。另一方面，多云存储架构可以规避单个云服务商发生故障或是内部人员恶意操作^[24] 等行为导致的数据损坏、服务故障等问题。另外，多云存储架构对信任的要求更低，用户无需完全信任某一个云服务商，而只需相信多个云服务商不会同时发生故障或者不会联合起来作恶。理论分析^[220] 已经证明多个云的合作对于云服务商也是更有益的（开销更低，服

¹www.carecloud.com

²这项全球调查由 IBM 商业价值研究院 (IBM Institute for Business Value, IBV) 与牛津经济研究院在 2021 年合作执行，对来自全球 47 个国家和地区的 28 个产业、约 7200 位高端主管进行访问。

务更好)。

如何在多云散布系统中实现对数据完整性的保护是一个十分重要的研究问题。多云存储的主要优势在于良好的隐私保护、服务和配置的灵活性以及避免单点故障。但是，从数据可靠性的角度看，多云系统中发生故障的概率实际上是提高了。假设单个云服务的故障率为 λ 。若系统中有 n 个云服务商，则多云存储系统的故障率为 $n\lambda$ ，是单个云服务商故障率的 n 倍。使用冗余散布策略可以在一定程度上降低数据丢失的可能性，但是效果有限。更为有效的方法是规律且频繁地执行数据审计，并在检测到数据损坏后立即执行修复。在前面的章节中，我们给出了基于区块链技术构建安全、高效、可扩展的分布式审计系统的研究方案。但是仍旧有一些重要问题尚未被讨论，这些问题也是在远程数据审计领域较少被关注的。(1) 相比较于冗余存储机制，审计机制对系统可靠性的提升效果如何？这里，我们想要得到一个量化的结果。(2) 在审计时，不同的数据选取策略是否会影响系统可靠性？(3) 如何设定合适的审计频次，在系统可靠性和系统开销之间取得平衡？要回答上述问题，我们需要一套针对多云存储系统的可靠性评价方法。

虽然远程数据审计的思想已经提出许多年，但是很少有工作从理论层面分析和评价运行审计机制的存储系统的可靠性。这使得我们在设计审计方案时缺少理论指导。本文的研究工作扩展了这一领域。在本章中，我们使用马尔可夫分析模型来评价远程数据审计对提升云存储系统可靠性的贡献。我们着重分析如下两点：(1) 引入审计机制为存储系统可靠性带来的提升作用，(2) 不同的审计策略对系统可靠性的影响。本文的分析结论可以帮助审计机制设计者更好地制定审计策略，例如确定审计频次、是否采用关联数据块批量审计策略等等。在多云存储环境下，数据完整性验证应与故障状态下的数据恢复相结合，以加速数据恢复过程、使系统尽快回到原有的安全防护水平。不同的数据恢复策略会如何影响多云存储系统的可靠性，也是非常值得进行细致分析的。量化分析的结果将能够指导多云审计机制以及数据恢复机制的设计和制定，从而更好地保证多云存储系统的可靠性和数据完整性。

本章的后续小节以如下方式组织：第二节介绍相关背景知识；第三节介绍了多云审计系统模型；第四节建立马尔可夫数学模型并给出求解方法；第五节进行实验验证，使用蒙特卡洛仿真算法验证马尔科夫模型求解结果，并根据分析结论给出一个适合多云散布系统的审计方案；第六节总结了本章的研究工作。

第二节 背景知识

5.2.1 多云散布系统

在多云存储系统中，一般使用信息散布机制将文件切分成若干个文件分片并将每个文件分片散布到不同的云服务商。一个 (k, n) 散布方案指的是将原数据 D 切分成 n 个分片且使用其中任意 k 个分片即可还原出原数据。为了保证数据机密性，一般使用 AES 加密算法单独加密每个文件。由用户本地管理加密密钥会给用户带来较大负担，且密钥容易丢失或损坏。将加密密钥存储在云端是一种更为可取的办法。一些散布方案^[37, 40] 提出将文件和密钥拼接成一个混合数据块后进行散布，这类方法无法达到足够的安全性。一个原因在于，密钥相对于经过加密的文件属于更为敏感的数据，应该被给予更高的安全防护级别，而不应进行与文件同样的处理。另一个原因是，混合散布方案使得数据分片既包含一部分文件信息又包含一部分密钥信息，因此一个分片损坏可能会同时威胁到文件和密钥的安全性。并且混合散布方案无法对文件或者密钥单独执行修复，使得修复过程较为低效。另一些方案^[36] 采用对文件和密钥分别进行散布的策略，并往往采取不同的分片生成方式以保证密钥具有更高的安全防护水平。我们认为这样的散布方式是更为安全高效的。

密文密钥分开散布的一个研究难点在于如何在保证安全性的同时尽量降低开销。秘密共享方案能够保证较高的安全性，不过计算复杂度较高、开销较大^[221]，因此比较适合用于密钥散布。相对应地，信息散布算法（IDA, information dispersal algorithms）的安全性稍弱，同时计算复杂度较低、开销较小^[34]，比较适合用于对密文进行散布。将秘密共享与 IDA 结合可以在安全性和计算开销之间取得良好平衡。例如，SSMS^[36] 使用 Shamir 秘密共享^[33] 来生成密钥分片，保证了足够高的密钥安全性，不过密钥散布过程较为低效，导致故障状态下的密钥修复过程会比较耗时，给了攻击者更多可乘之机。为了提升密钥生成和修复时的速度，可以考虑设计更为高效的秘密共享方案，从而快速地对密钥执行重新散布。

基于以上，我们在设计针对多云散布系统的审计策略时，必须考虑到文件和加密密钥安全防护需求不同的问题。例如，我们可以分别为文件和密钥针设计不同的审计策略，对文件和密钥的审计和修复也可以分别进行。另外，数据修复的过程要足够高效，以免造成更多数据损坏或丢失。

5.2.2 云存储可靠性评价

对云存储系统可靠性的分析和评价是云存储领域的研究热点之一。马尔可夫模型被广泛应用于这一领域，指导云存储系统设计者构建更为可靠的存储服务。例如，文献 [174] 使用马尔可夫模型分析磁盘清洗机制对于云存储系统可靠性的提升作用，并通过对分析不同的磁盘清洗策略的效果差异得出最优策略。文献 [222] 使用马尔可夫模型预测存储集群中一个数据校验组的可用性。文献 [168] 使用马尔可夫决策过程模型帮助云服务商得到最优的存储策略，从而使较低的存储开销实现较高的存储可靠性。上述工作主要考虑云存储系统内部磁盘故障（如磁盘老化、运行故障、潜在快损坏等等）引发的数据丢失问题。然而，随着云存储服务广泛普及，云上数据受到的针对性的、恶意的攻击也越来越普遍^[182]。根据 2020 年发布的一份互联网安全性报告^[223]，目前威胁云上数据安全性³的前三大风险分别是不安全的接口和 API、数据丢失及泄露、硬件故障，其中前两大风险多由恶意攻击导致。因此，除了评价云存储自身能够提供的可靠性外，一些研究工作^[184, 185, 224]开始关注如何评价云存储系统在可能遭遇外部攻击的情况下生存性⁴。例如，文献 [225] 使用马尔科夫模型和半马尔可夫模型分析了系统在遭受持续性攻击下的生存性。文献 [186] 针对基于 RAID 的云存储系统，使用马尔可夫模型分析中磁盘级别的可靠性和生存性，并使用组合数学模型分析系统级别的可靠性和生存性。

上述可靠性评价方法大多着眼于单个云存储系统发生的故障，如磁盘运行故障、潜在块损坏、恶意攻击等等。而本章的工作关注的是单个云服务商的可靠性机制失效导致多云存储系统发生数据丢失的情况。

我们使用连续时间马尔可夫模型^[226]来描述多云存储系统的动态运行过程。马尔可夫模型的一个主要假设是不同故障之间彼此独立且故障率恒定。我们认为这一假设适用于我们的分析。首先，在分析模型中我们假设同一文件校验组的分片被存储在不同的云上，而不同云服务商发生分片损坏的事件可被视为彼此独立。此外，鉴于云存储系统故障发生的复杂性，我们认为有必要进行适当简化才能使分析得以进行。假设故障率恒定可以帮助我们将关注范围限定在几个重要的系统参数，并直观分析这些参数对于系统可靠性的影响。马尔可夫模型

³ 数据安全性可进一步划分为机密性（confidentiality）、完整性（integrity）和可用性（availability），通常使用缩写 CIA 来表示。

⁴ 生存性（survivability）指云存储系统在内部（偶发性）故障和外部（针对性）攻击同时存在时系统能够正常运转的能力

的另一个主要假设是故障事件发生的时间服从指数分布。尽管目前一些实验和测试结果证明磁盘故障发生的时间服从韦布分布 [175, 178–180, 227]，但我们认为指数分布的假设仍旧适用于我们的分析。一方面，韦布分布虽然能够很好地刻画存储系统中的磁盘老化现象，但是在大型云存储系统中一般会运行磁盘故障预警机制来削弱磁盘老化对系统可靠性的影响。再加上大型数据中心一般是新老磁盘混杂使用，这使得系统整体的平均故障率趋于常数。另外，云存储系统中并不仅仅存在磁盘故障，还包括电力故障、系统漏洞、自然灾害、人为失误等等复杂因素，因此磁盘老化带来的影响会被进一步削弱。最后，本章分析的重点在于对比不同参数之间的相对差异，例如云服务商可靠性的变化对多云存储系统可靠性的影响，单云系统和多云系统可靠性的差异，以及不同审计策略对系统可靠性的效果等。相对地，我们并不要求模型预测出的结果与现实情况精确近似。因此，指数分布假设带来的模型预测结果的误差是可接受的。在文献 [222] 中也证实，使用马尔可夫模型预测的平均故障发生时间（MTTF, mean time to failure）与在实际谷歌存储集群中测试的结果是较为相近的（指数量级相同）。

第三节 多云审计系统介绍

我们着眼于运行审计机制的多云存储系统。该系统由多个云服务商共同提供数据存储服务。但是，由于内部故障或是外部攻击等原因，云服务商有可能无法向用户提供被请求的数据，甚至云上存储的数据可能会被损坏。当发生故障的云服务商个数超过多云系统能够承受的范围，数据丢失便会发生。审计机制的作用是尽可能降低发生数据丢失的可能性。

5.3.1 多云数据散布

多云存储系统由 n 个云服务商（CSP, cloud service provider）构成，采用基于纠删码的冗余存储策略以提升存储可靠性，避免发生数据丢失。每个校验组包含 n 个分片，其中包括 k 个数据分片和 $m = n - k$ 个校验分片。同一校验组的 n 个分片被分别散布到 n 个云服务商。使用其中任意 k 个分片即可恢复原始数据。分散式存储有很多好处。（1）即使某个云服务商发生故障也并不会影响数据可用性，因为只要 n 个服务商中有 k 个服务商能够提供正确的数据分片则原数据可被成功还原。（2）每个云服务商只存储原始数据的一部分，降低了数据泄露的风险。

如图5.1给出一个示例。用户文件 F 经过 AES 加密得到密文 C ，再经过信息

散布算法 (IDA, information dispersal algorithm) 处理得到 n 个文件分片, 其中包含 k 个原始数据分片和 m 个校验分片。类似地, 加密密钥 K 经过秘密共享 (SS, secret sharing) 得到 k 个预备分片, 再经过 IDA 处理得到 n 个密钥分片, 其中包含 m 个校验分片。所生成的 n 个文件分片和 n 个密钥分片分别散布给 n 个云服务商。采用混合散布策略, 使得每个云服务商既保存原始数据分片也保存校验分片。当用户想要下载文件时, 只需获取到 k 个正确的文件分片和 k 个正确的密钥分片便可成功重构出原文件。由于采用了冗余编码和散布存储的机制, 单个云服务商发生故障或者遭遇攻击并不会导致用户文件无法还原。假设某个外部攻击者想要篡改用户文件或是导致用户文件不可用, 也必须要成功破坏至少 $m+1$ 个文件分片或 $m+1$ 个密钥分片。

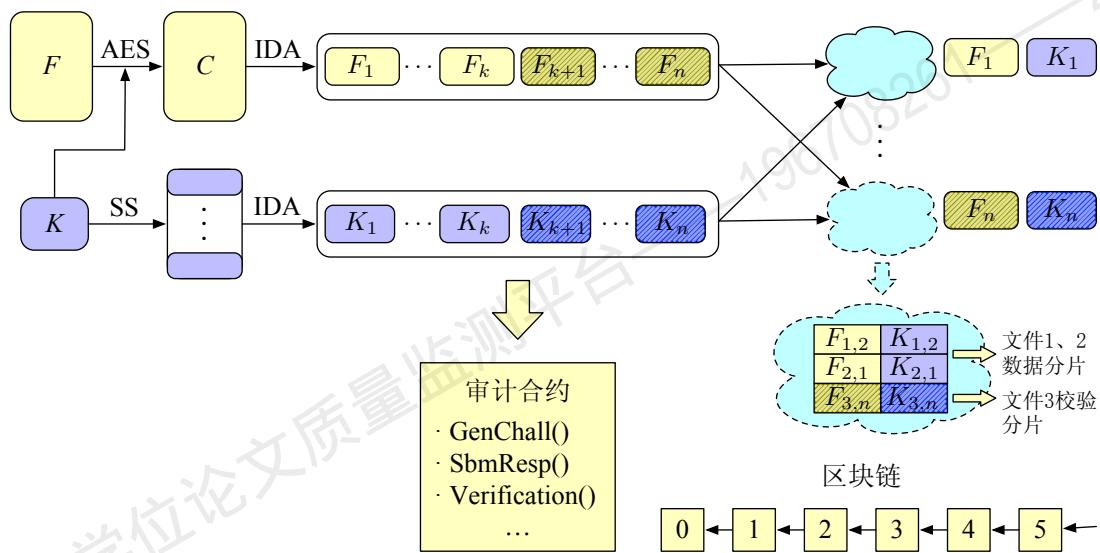


图 5.1 多云存储系统散布过程

5.3.2 数据审计和修复

多云存储系统运行基于区块链的远程数据审计机制来及时检测到已经发生的分片损坏。审计每隔固定时间执行一次, 每次随机选中若干个数据分片, 并执行基于区块链的完整性验证 (如第三章所述) 以确认该分片的完整性。被检测到损坏的分片会立即执行修复。

策略 1: 随机审计。在每次审计过程中随机选中 k 个数据分片。被选中的分片可能位于同一云上, 也可能位于不同的云上。一旦发现有损坏的分片立即执行修复。修复过程不再对分片执行完整性验证, 而是直接从云端下载除损坏分

片外的 k 个数据分片并重构出原数据，进而计算出损坏分片的值（由于可能下载到错误的分片，因此计算出的分片值也可能是错误的）。

策略 2：批量审计。在每次审计过程中随机选中某个数据校验组，并选中其中任意 k 个分片。由于同一数据校验组的 n 个不同分片被散布到不同的云，因此一次批量审计的 k 个分片一定来自不同的云。审计过程中若发现有损坏分片，立即修复该分片。修复策略与随机审计相同。

策略 3：更新审计。更新审计的数据选取策略与随机审计相同，不过在审计过程中一旦发现有损坏分片则将该分片所在数据校验组的 n 个分片统一进行修复。修复过程需要下载 k 个正确的数据分片（下载后执行完整性验证）并重构出正确的原数据，进而计算出 n 个分片的值。这一修复过程可能会将尚未被审计到的损坏分片进行修复。

我们将在第四节和第五节中详细对比这三种策略对多云存储系统可靠性的提升效果。具体地，通过对策略 1 和策略 2，我们能够得出审计过程中不同的数据选取策略对多云存储系统可靠性的影响；通过对策略 1 和策略 3，我们能够分析和评价不同的数据修复策略对多云存储系统可靠性的提升效果。特别地，通过对比无审计机制的多云存储系统与运行审计机制的多云存储系统，我们可以评价审计机制对于系统可靠性的影响。

第四节 数学模型

在本小节中，我们通过建立数学模型分析远程数据审计机制对于提升多云存储系统可靠性的效果。平均数据丢失时间（MTTDL, mean time to data loss）是可靠性评价领域最常用的评价指标^[167,176,228,229]。该指标衡量存储系统平均发生一次数据丢失事件的期望时间。我们针对某一个数据校验组进行分析，并使用该校验组的 MTTDL 代表多云存储系统的可靠性。

如图5.2所示，使用马尔可夫模型模拟校验组发生分片损坏及修复的动态过程。初始状态 0 表示校验组中未发生任何数据损坏的状态。在此状态下，若某个云服务商发生故障导致该校验组中任意一个分片发生损坏，则校验组由状态 0 转移至状态 1，参数 $\lambda_{0,1}$ 为平均转移速率⁵。校验组处于状态 $1 \leq i \leq m$ 时，虽有 i 个分片发生损坏但尚未造成数据丢失。当校验组转移到状态 $m < i \leq n$ 时，原数据将无法还原。我们使用吸收态 F 表示校验组发生数据丢失的状态集合。当校

⁵在这里我们只关心能够被多云系统感知到的分片损坏故障。对于单个云服务商内部发生的能够被其可靠性保护机制修复的故障，我们不作考虑。

验组处于吸收态时状态转移率为 0，这意味着校验组将不会再转移到非吸收态。由于多云存储系统实施审计机制以及数据修复机制，状态 $1 \leq i \leq m$ 有可能会转移到状态 $i-1, \dots, 0$ ，这表示一部分已经发生损坏的数据分片在校验组转移到吸收态之前被审计机制检测并进行修复。平均修复率 μ 表示一个时间单位内执行修复的期望次数。在分析过程中我们将时间单位设置为天，因此平均修复率等于在一天内成功执行修复的次数。可靠性评价指标 MTTDL 等于校验组从初始状态 0 开始直到进入吸收态的时间的期望。

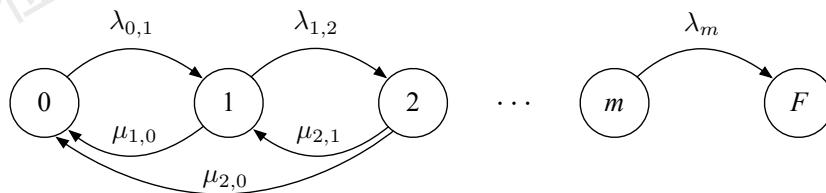


图 5.2 多云存储系统可靠性马尔可夫模型

在接下来的分析中，我们假设多云存储系统中共有 6 个云服务商，并且采用 $(4, 6)$ 的冗余存储策略（即 $k = 4, m = 2, n = 6$ ），使用 6 个分片中的任意 4 个便可恢复原文件。系统总共存储 F 个数据校验组，因此分片总数 $N = nF$ 。假设系统每隔时间 T 执行一次审计，审计频率 $\beta = 1/T$ 。我们假设不同云上的分片的损坏速率相同，并使用 λ 表示这一速率。假设在审计机制检测到损坏分片之后立即执行数据修复。假设 MTTDL 的单位为天。

5.4.1 随机审计策略

若系统执行随机审计策略，每次审计将随机选中 4 个分片验证其完整性。如图 5.3 给出随机审计策略下的马尔可夫模型状态转移图。在状态 0, 1, 2 时的故障率分别为 $n\lambda, (n-1)\lambda, (n-2)\lambda$ 。

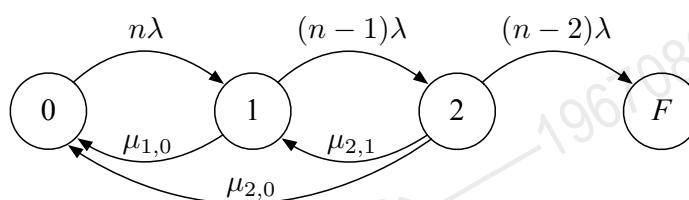


图 5.3 随机审计策略下的马尔可夫模型

根据状态转移图我们可以得到 4 行 4 列的状态转移矩阵，其中非对角线元

素均为非负数，对角线元素均为非正数，且每行元素和为 0。

$$\begin{pmatrix} -n\lambda & n\lambda & 0 & 0 \\ \mu_{1,0} & -(n-1)\lambda - \mu_{1,0} & (n-1)\lambda & 0 \\ \mu_{2,0} & \mu_{2,1} & -(n-2)\lambda - \mu_{2,0} - \mu_{2,1} & (n-2)\lambda \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

平均修复率等于一天中成功执行数据修复的次数，在审计系统中平均修复率的值与审计频次密切相关。给定审计频次为 β ，在随机审计策略下每一次分片选取之间相互独立，因此这相当于我们在一天内进行了 $k\beta$ 次随机抽取，且每次抽取时任意一个分片被抽到的概率是 $\frac{1}{nF}$ 。接下来我们给出修复率的具体推导过程。

推导 $\mu_{1,0}$ 。如图 5.4 所示，若校验组中发生一个分片损坏，且刚好某次审计选中了损坏的分片，系统开始执行修复过程。修复程序下载校验组中除损坏分片外的另外 4 个分片，重构出原数据并计算出损坏分片的值重新上传到云端，至此修复成功。

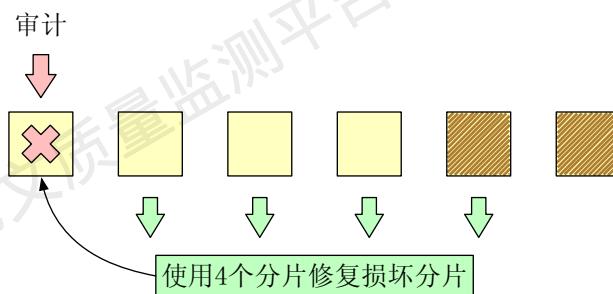


图 5.4 随机审计数据修复（一个分片损坏）。

$\mu_{1,0}$ 指的是当校验组发生一个分片损坏时，该分片在一天中被成功修复的次数的期望。由于在随机审计策略下一天中总共进行 $k\beta$ 次抽取，因此概率上最多能成功修复 $k\beta$ 次，最少成功修复 0 次。令 X 表示在一天中抽到损坏分片的次数，则有

$$P\{X = i\} = \binom{k\beta}{i} \left(\frac{1}{nF}\right)^i \left(1 - \frac{1}{nF}\right)^{k\beta-i}$$

上式中 $\frac{1}{nF}$ 表示在一次随机抽取中抽到损坏分片的概率，表达式 $(\frac{1}{nF})^i$ 表示有 i 次抽到损坏分片的概率，表达式 $(1 - \frac{1}{nF})^{k\beta-i}$ 表示有 $k\beta - i$ 次抽取到非损坏分片

的概率。根据数学期望计算公式

$$E(X) = \sum_k^{\infty} X_k \cdot p(X_k)$$

我们得到：

$$\begin{aligned}\mu_{1,0} &= \sum_{i=0}^{k\beta} i \cdot P\{X = i\} \\ &= \sum_{i=0}^{k\beta} i \cdot \binom{k\beta}{i} \left(\frac{1}{nF}\right)^i \left(1 - \frac{1}{nF}\right)^{k\beta-i}\end{aligned}$$

推导 $\mu_{2,0}$ 。在随机审计策略下，当校验组中有两个分片发生损坏时存在修复失败的可能性。如图5.5给出一个示例。校验组中有两个分片发生损坏，且刚好某次审计选中了其中一个损坏的分片，系统开始执行修复过程。执行分片修复过程中不检测同一校验组的其它分片是否损坏，而是直接下载数据并重构。因此，假如修复程序恰好下载到了另一个已损坏的分片，那么程序重新计算出的分片值仍旧是错误的，这将导致修复失败。

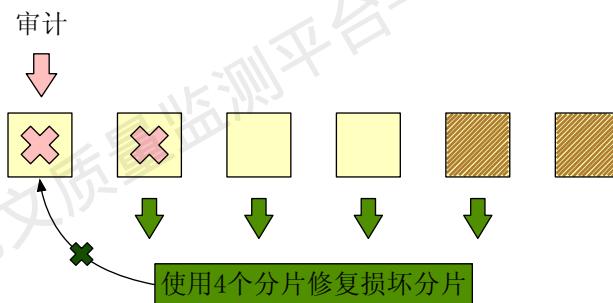


图 5.5 随机审计数据修复（两个分片损坏）。

$\mu_{2,0}$ 指的是当校验组发生两个分片损坏时，这两个分片在一天中全部被成功修复的次数的期望。在计算 $\mu_{2,0}$ 时我们要去除修复失败的情况。令 X_1 表示在一天中抽到损坏分片 1 且修复成功的次数，令 X_2 表示在一天中抽到损坏分片 2 且修复成功的次数，则有

$$P\{X_1 = i \cap X_2 = i\} = \binom{k\beta}{i} \left(\frac{1}{nF}\right)^i \left(\frac{1}{5}\right) \cdot \binom{k\beta-i}{i} \left(\frac{1}{nF}\right)^i \left(\frac{1}{5}\right) \cdot \left(1 - \frac{2}{nF}\right)^{k\beta-2i} \cdot \binom{2}{1}$$

上式中 $\binom{k\beta}{i} \left(\frac{1}{nF}\right)^i$ 表示在总共 $k\beta$ 次抽取中有 i 次抽到损坏分片 1 的概率，乘以 $\frac{1}{5}$ 表示在执行对损坏分片 1 的数据修复时从校验组中剩下的 5 个分片中刚好选择

了正确的 4 个分片重构数据的概率，即

$$P\{\text{修复时选中 } k \text{ 个正确的分片}\} = \frac{\binom{k}{k}}{\binom{n-1}{k}} = \frac{\binom{4}{4}}{\binom{5}{4}} = \frac{1}{5}$$

表达式 $\binom{k\beta-i}{i} \left(\frac{1}{nF}\right)^i$ 表示在剩下的 $k\beta - i$ 次抽取中有 i 次选中了损坏分片 2 的概率，乘以 $\frac{1}{5}$ 表示在执行对损坏分片 2 的数据修复时从校验组中剩下的 5 个分片中刚好选择了正确的 4 个分片重构数据的概率。表达式 $(1 - \frac{2}{nF})^{k\beta-2i}$ 表示有 $k\beta - 2i$ 次抽取到非损坏分片的概率。表达式 $\binom{2}{1}$ 表示前 i 次抽到损坏分片 1 或者前 i 次抽到损坏分片 2 这两种情况。因此，在一天中两个损坏分片都被修复的次数的期望为

$$\begin{aligned}\mu_{2,0} &= \sum_{i=0}^{k\beta} i \cdot P\{X_1 = i \cap X_2 = i\} \\ &= \sum_{i=0}^{k\beta} \frac{2i}{25} \binom{k\beta}{i} \binom{k\beta-i}{i} \left(\frac{1}{nF}\right)^{2i} \left(1 - \frac{2}{nF}\right)^{k\beta-2i}\end{aligned}$$

推导 $\mu_{2,1}$ 。 修复率 $\mu_{2,1}$ 指的是当校验组发生两个分片损坏时，在一天中只有其中一个分片被成功修复的次数的期望。与推导 $\mu_{2,0}$ 时相同，我们需要排除掉选中损坏分片但是修复失败的情况。令 X 表示在一天中两个损坏分片有且只有一个被成功修复的次数，于是有

$$P\{X = i\} = \binom{k\beta}{i} \left(\frac{1}{nF}\right)^i \left(1 - \frac{2}{nF}\right)^{k\beta-i} \cdot \binom{2}{1} \cdot \left(\frac{1}{5}\right)$$

上式中 $\binom{k\beta}{i} \left(\frac{1}{nF}\right)^i \left(1 - \frac{2}{nF}\right)^{k\beta-i}$ 表示在总共 $k\beta$ 次抽取中有 i 次抽到损坏分片且剩下的 $k\beta - i$ 次抽到未损坏分片的概率，表达式 $\binom{2}{1}$ 表示抽到的是损坏分片 1 或者损坏分片 2 这两种情况。乘以 $\frac{1}{5}$ 表示修复成功的概率。因此，在一天中两个损坏分片中有且只有一个被修复的次数的期望为

$$\begin{aligned}\mu_{2,1} &= \sum_{i=0}^{k\beta} i \cdot P\{X = i\} \\ &= \sum_{i=0}^{k\beta} \frac{2i}{5} \binom{k\beta}{i} \left(\frac{1}{nF}\right)^i \left(1 - \frac{2}{nF}\right)^{k\beta-i}\end{aligned}$$

5.4.2 批量审计策略

若系统执行批量审计策略，在每次审计时将随机选中某一个校验组，再随机选中组内的 4 个分片验证其完整性。如图 5.6 给出批量审计策略下的马尔可夫模型状态转移图。

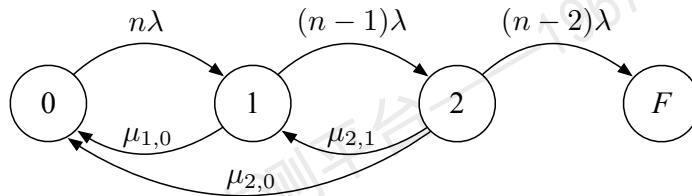


图 5.6 批量审计策略下的马尔可夫模型

根据状态转移图得到 4 行 4 列的状态转移矩阵

$$\begin{pmatrix} -n\lambda & n\lambda & 0 & 0 \\ \mu_{1,0} & -(n-1)\lambda - \mu_{1,0} & (n-1)\lambda & 0 \\ \mu_{2,0} & \mu_{2,1} & -(n-2)\lambda - \mu_{2,0} - \mu_{2,1} & (n-2)\lambda \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

在批量审计策略下，要先随机选中校验组再随机选中其中的分片。给定审计频次为 β ，则相当于在一天内进行 β 次随机抽取，且每次抽取时任意一个校验组被抽到的概率是 $1/F$ 。若某个校验组已经被选中，则其中每个分片被选中的概率是 k/n 。接下来我们给出修复率的具体推导过程。

推导 $\mu_{1,0}$ 。 如图5.7所示，校验组中发生一个分片损坏。若刚好某次审计选中了该校验组并且选中了该损坏分片，系统开始执行修复过程。修复程序下载校验组中除损坏分片外的另外 4 个分片，重构出原数据并计算出损坏分片的值重新上传到云端，至此修复成功。

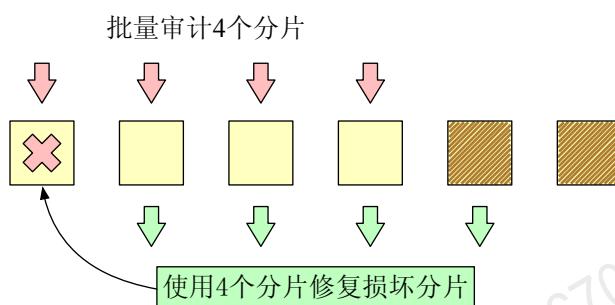


图 5.7 批量审计数据修复 (一个分片损坏)。

令 X 表示在一天中抽到损坏分片所在校验组的次数，则有

$$P\{X = i\} = \binom{\beta}{i} \left(\frac{1}{F}\right)^i \left(1 - \frac{1}{F}\right)^{\beta-i}$$

令事件 Y 表示在已经抽到损坏分片所在校验组的情况下刚好抽到损坏分片，则有

$$P\{Y\} = \frac{\binom{n-1}{k-1}}{\binom{n}{k}} = \frac{\binom{5}{3}}{\binom{6}{4}} = \frac{2}{3}$$

因此，在一天中损坏分片被修复的次数的期望为

$$\begin{aligned}\mu_{1,0} &= \sum_{i=0}^{\beta} i \cdot P\{(X=i) \cap Y\} \\ &= \sum_{i=0}^{\beta} \frac{2i}{3} \binom{\beta}{i} \left(\frac{1}{F}\right)^i \left(1 - \frac{1}{F}\right)^{\beta-i}\end{aligned}$$

推导 $\mu_{2,0}$ 。 在批量审计策略下，当校验组中有两个分片发生损坏时仍旧存在修复失败的可能性。如图5.8所示，校验组中发生两个分片损坏，某次批量审计刚好抽中其中一个分片，系统开始执行修复过程。过程中不检测同一校验组剩余两个分片是否损坏，而是直接下载数据并重构。若修复程序恰好下载到了另一个已损坏的分片，则本次修复失败。

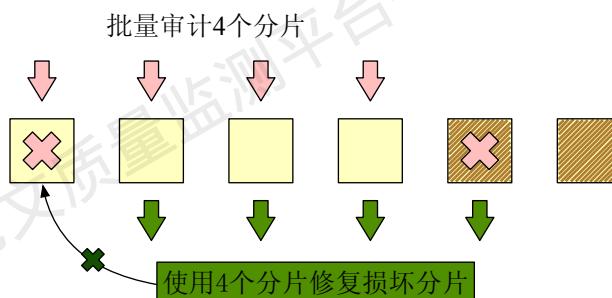


图 5.8 批量审计数据修复（两个分片损坏）。

在计算 $\mu_{2,0}$ 时，我们考虑如下两种可能性：(1) 有两次审计分别抽中了损坏分片 1 和损坏分片 2，并且两次都修复成功，于是校验组由状态 2 回复到状态 0；(2) 一次审计同时抽中了两个损坏分片，这种情况下一定能修复成功，于是校验组由状态 2 回复到状态 0。因此我们统计三类事件：

- 事件 1：在一次审计中只抽到损坏分片 1 并成功修复；
- 事件 2：在一次审计中只抽到损坏分片 2 并成功修复；
- 事件 3：在一次审计中同时抽到损坏分片 1 和损坏分片 2。

令 X_1, X_2, X_3 表示在一天中事件 1、事件 2、事件 3 发生的次数，则在一天中校验

组有 i 次由状态 2 修复到状态 0 的概率可表示为

$$\begin{aligned} & P\{X_1 = j \cap X_2 = j \cap X_3 = i - j\} \\ &= \binom{\beta}{j} \left(\frac{1}{F}\right)^j c_1^j \cdot \frac{1}{2} \cdot \binom{\beta-j}{j} \left(\frac{1}{F}\right)^j c_1^j \cdot \frac{1}{2} \cdot \binom{2}{1} \cdot \binom{\beta-2j}{i-j} \left(\frac{1}{F}\right)^{i-j} c_2^{i-j} \cdot \left(1 - \frac{1}{F}\right)^{\beta-i-j} \end{aligned}$$

上式中 $j \in \{0, \dots, i\}$ 。表达式 $\binom{\beta}{j} \left(\frac{1}{F}\right)^j c_1^j$ 表示有 j 次抽中校验组并抽中损坏分片 1，其中 $c_1 = \binom{n-2}{k-1} / \binom{n}{k}$ 表示抽到损坏分片 1 的概率。乘以 $\frac{1}{2}$ 表示修复成功的概率（在剩余尚未审计的 2 个分片中选择了未损坏的那个分片进行数据修复）。表达式 $\binom{\beta-j}{j} \left(\frac{1}{F}\right)^j c_1^j$ 表示在剩下的 $\beta - j$ 次抽取中有 j 次抽中校验组并抽中损坏分片 2。乘以 $\frac{1}{2}$ 表示修复成功的概率。表达式 $\binom{2}{1}$ 表示前 j 次抽到损坏分片 1 或者前 j 次抽到损坏分片 2 这两种情况。表达式 $\binom{\beta-2j}{i-j} \left(\frac{1}{F}\right)^{i-j} c_2^{i-j}$ 表示在剩下的 $\beta - 2j$ 次抽取中有 $i - j$ 次抽中校验组并同时抽中两个损坏分片，其中 $c_2 = \binom{n-2}{k-2} / \binom{n}{k}$ 。

因此，在一天中两个损坏分片都被修复的次数的期望为

$$\begin{aligned} \mu_{2,0} &= \sum_{i=0}^{\beta} \sum_{j=0}^i i \cdot P\{X_1 = j \cap X_2 = j \cap X_3 = i - j\} \\ &= \sum_{i=0}^{\beta} \sum_{j=0}^i \frac{i}{2} c_1^j c_2^{i-j} \binom{\beta}{j} \binom{\beta-j}{j} \binom{\beta-2j}{i-j} \left(\frac{1}{F}\right)^{i+j} \left(1 - \frac{1}{F}\right)^{\beta-i-j} \end{aligned}$$

推导 $\mu_{2,1}$ 。 修复率 $\mu_{2,1}$ 指的是当校验组发生两个分片损坏时，在一天中有且只有一个分片被成功修复的次数的期望。我们需要排除掉选中损坏分片但是修复失败的情况。令 X 表示在一天中抽到损坏分片所在校验组的次数，于是有

$$P\{X = i\} = \binom{\beta}{i} \left(\frac{1}{F}\right)^i \left(1 - \frac{1}{F}\right)^{\beta-i}$$

令事件 Y 表示在已经抽到损坏分片所在校验组的情况下刚好抽到其中一个损坏分片，则有

$$P\{Y\} = \frac{\binom{1}{2} \binom{n-2}{k-1}}{\binom{n}{k}} = 2 \times \frac{\binom{4}{3}}{\binom{6}{4}} = \frac{8}{15}$$

事件 Z 表示在已经抽到其中一个损坏分片的情况下成功执行修复，则有

$$P\{Z\} = \frac{1}{2}$$

因此，在一天中两个损坏分片有且只有一个被修复的次数的期望为

$$\begin{aligned}\mu_{2,1} &= \sum_{i=0}^{\beta} i \cdot P\{(X=i) \cap Y \cap Z\} \\ &= \sum_{i=0}^{\beta} \frac{4i}{15} \binom{\beta}{i} \left(\frac{1}{F}\right)^i \left(1 - \frac{1}{F}\right)^{\beta-i}\end{aligned}$$

5.4.3 更新审计策略

若系统执行更行审计策略，则数据选取方式与随机审计相同，每次审计时随机选中 k 个分片，一天中共执行 β 次审计。所不同的是数据修复策略，一旦发现某分片错误则立即将该分片所在校验组的 n 个分片全部修复。如图5.9给出更新审计策略下的马尔可夫模型状态转移图。当状态 $i = 2$ 时只有指向状态 0 的修复箭头没有指向状态 1 的修复箭头。

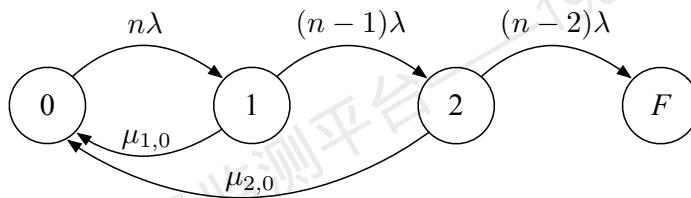


图 5.9 更新审计策略下的马尔可夫模型

根据状态转移图得到 4 行 4 列的状态转移矩阵

$$\begin{pmatrix} -n\lambda & n\lambda & 0 & 0 \\ \mu_{1,0} & -(n-1)\lambda - \mu_{1,0} & (n-1)\lambda & 0 \\ \mu_{2,0} & 0 & -(n-2)\lambda - \mu_{2,0} & (n-2)\lambda \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

推导 $\mu_{1,0}$ 。当校验组中只有一个分片损坏时 $\mu_{1,0}$ 的值与随机审计时相同。

$$\mu_{1,0} = \sum_{i=0}^{\beta} i \cdot \binom{\beta}{i} \left(\frac{1}{F}\right)^i \left(1 - \frac{1}{F}\right)^{\beta-i}$$

推导 $\mu_{2,0}$ 。根据更新审计策略，当校验组中有两个分片发生损坏时，假如某次审计选中了其中一个损坏分片，则系统执行修复过程。该过程将从云端下载 k

个正确的分片（下载分片后执行校验以确认分片完整性），重构出原数据后重新计算 n 个分片的值，并重新上传到云端。因此，另一个损坏分片也会被修复。在一天中，只要有 i 次抽取到两个损坏分片中的任意一个则两个分片都会被修复。因此有

$$\begin{aligned}\mu_{2,0} &= \sum_{i=0}^{k\beta} i \cdot P\{\text{有 } i \text{ 次抽到损坏分片中任意一个}\} \\ &= \sum_{i=0}^{k\beta} i \cdot \binom{k\beta}{i} \left(\frac{2}{nF}\right)^i \left(1 - \frac{2}{nF}\right)^{k\beta-i}\end{aligned}$$

5.4.4 无审计的多云存储系统可靠性模型

为便于衡量审计机制对于多云存储系统可靠性的提升效果，我们也给出无审计机制的多云存储系统的马尔可夫模型状态转移图，如图5.10所示。

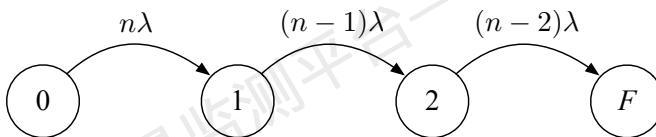


图 5.10 无审计机制的多云存储系统马尔可夫模型

根据状态转移图得到 4 行 4 列的状态转移矩阵

$$\begin{pmatrix} -n\lambda & n\lambda & 0 & 0 \\ 0 & -(n-1)\lambda & (n-1)\lambda & 0 \\ 0 & 0 & -(n-2)\lambda & (n-2)\lambda \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

5.4.5 模型求解

在求解模型时，我们关注的是以状态 0 为起始，直到校验进入吸收态的期望时间。我们将这一期望值作为多云存储系统可靠性的评价指标（即（MTTDL））。参考文献 [230]，我们给出根据上述马尔可夫模型求解 MTTDL 的方法。

以随机审计马尔可夫模型来举例。给定状态转移矩阵

$$Q = \begin{pmatrix} -n\lambda & n\lambda & 0 & 0 \\ \mu_{1,0} & -(n-1)\lambda - \mu_{1,0} & (n-1)\lambda & 0 \\ \mu_{2,0} & \mu_{2,1} & -(n-2)\lambda - \mu_{2,0} - \mu_{2,1} & (n-2)\lambda \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

将所有状态空间 $S = \{0, 1, \dots, 6\}$ 划分成两个集合：非吸收态 $T = \{0, 1, 2\}$ 和吸收态 $T^a = \{3, 4, 5, 6\}$ 。根据这一划分规则，状态转移矩阵可被划分为四个子矩阵：

$$Q = \left[\begin{array}{cccc|c} -n\lambda & n\lambda & 0 & 0 & 0 \\ \mu_{1,0} & -(n-1)\lambda - \mu_{1,0} & (n-1)\lambda & 0 & 0 \\ \mu_{2,0} & \mu_{2,1} & -(n-2)\lambda - \mu_{2,0} - \mu_{2,1} & (n-2)\lambda & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \end{array} \right] = \begin{pmatrix} V & R \\ O & O \end{pmatrix}$$

定义转移方程矩阵 $P(t)$ ，其元素 $P_{ij}(t) = P\{X(t+s) = j | X(t) = i\}$ 表示在时间 t 系统由状态 i 转移到状态 j 的概率。则转移方程矩阵可被划分为四个子矩阵

$$P(t) = \begin{pmatrix} T(t) & S(t) \\ O & I \end{pmatrix}$$

进一步地，定义停留时间矩阵 $H(t)$ ，其元素 (i, j) 表示系统以 i 为起始状态时，从时间 0 到时间 t 之间系统处于状态 j 的期望时间（有 $i, j \in \{0, 1, \dots, 6\}$ ）。则停留时间矩阵也可被划分为四个子矩阵

$$H(t) = \begin{pmatrix} N(t) & M(t) \\ O & tI \end{pmatrix}$$

其中矩阵 $N(t)$ 的元素 (i, j) 表示系统以 $i \in T$ 为起始状态时，从时间 0 到时间 t 之间系统处于状态 $j \in T$ 的期望时间。那么，矩阵 $N(\infty)$ 第一行元素的和即可表示系统以状态 0 为起始状态，由时间 0 开始直到进入吸收态的期望时间。接下来我们求解 $N(t)$ 。根据文献 [230] 给出的公式 5.2.5 有

$$H(t) = \int_0^t P(x)dx = \int_0^t \sum_{k=0}^{\infty} \frac{x^k Q^k}{k!} dx = \sum_{k=0}^{\infty} \frac{Q^k}{k!} \int_0^t x^k dx = \sum_{k=0}^{\infty} \frac{Q^k t^{k+1}}{(k+1)!}$$

于是我们可以得出

$$QH(t) = \sum_{k=0}^{\infty} \frac{(Qt)^{k+1}}{(k+1)!} = P(t) - I$$

根据上面等式关系进行矩阵乘法，可以得到

$$\begin{aligned} \begin{pmatrix} V & R \\ O & O \end{pmatrix} \begin{pmatrix} N(t) & M(t) \\ O & tI \end{pmatrix} &= \begin{pmatrix} T(t) & S(t) \\ O & I \end{pmatrix} - \begin{pmatrix} I & O \\ O & I \end{pmatrix} \\ \begin{pmatrix} VN(t) & tR + VM(t) \\ O & O \end{pmatrix} &= \begin{pmatrix} T(t) - I & S(t) \\ O & O \end{pmatrix} \end{aligned}$$

由此我们得到 $N(t)$ 的表达式

$$N(t) = V^{-1}[T(t) - I]$$

其中，矩阵 $T(t)$ 是仅包含非吸收态的转移方程矩阵。不难看出

$$\lim_{t \rightarrow \infty} T(t) = 0$$

由此我们得到

$$N(\infty) = -V^{-1}$$

我们可以通过矩阵 V 求得矩阵 $N(\infty)$ ，再计算其第一行元素之和，便可得到 MTTDL。

第五节 实验验证

在本小节中，我们将讨论以下问题：

- 问题 1：假设每个云服务商可靠性相同，我们使用多个云服务商构建出多云存储系统后，系统整体的可靠性会有什么变化？
- 问题 2：审计机制能够为多云存储系统的可靠性带来多大提升？
- 问题 3：什么是最适合多云存储系统的审计策略？

为了回答上述问题，我们根据上一节介绍的马尔可夫模型来及其求解方法，使用 Matlab 计算出多云存储系统的可靠性，并进行分析。为了验证 Matlab 结果的准确性，我们使用了蒙特卡洛（Monte Carlo）仿真方法统计系统的 MTTDL，并与求得的马尔可夫模型结果进行对比。

5.5.1 蒙特卡洛实验设置

为了验证马尔可夫模型的准确性，我们设计了一个蒙特卡洛仿真算法，模拟多云存储系统的动态运行过程，并统计目标校验组发生数据丢失的时间。

算法模拟一个由 6 个云服务商组成的多云存储系统，每个云服务商有一个唯一标记 $cloud_0, cloud_1, \dots, cloud_5$ 。系统内共存储 5000 个文件。每个文件对应 1 个文件校验组和 1 个密钥校验组，因此系统内共有 10000 个校验组。每个校验组有一个唯一标记 $file_0, file_1, \dots, file_{9999}$ 。每个校验组包含 6 个分片，使用其中任意 4 个分片便可恢复原数据。每个分片使用二元组进行唯一标记 $slice_no = (file_no, cloud_no)$ ，即校验组 $file_no$ 存储在云服务商 $cloud_no$ 上面的分片。

算法模拟三类事件：

- (1) 基于指数分布模拟随机的分片损坏事件。算法选定目标校验组为 $file_{560}$ 。给定指数分布参数 λ ，目标校验组中任意一个分片发生损坏的时间 t 的概率密度函数为：

$$f(t) = \begin{cases} \frac{1}{\lambda} e^{-\frac{t}{\lambda}} & t \geq 0 \\ 0 & t < 0 \end{cases}$$

累积密度函数为

$$F(t) = \begin{cases} 1 - e^{-\frac{t}{\lambda}} & t \geq 0 \\ 0 & t < 0 \end{cases}$$

根据指数分布为目标校验组中每个分片随机生成损坏时间，并使用二元组进行唯一标记 $fail_no = (time_curr, cloud_no)$ ，即在时间 $time_curr$ 该校验组存储在云 $cloud_no$ 上的分片损坏。一共生成 n 个分片损坏事件。

- (2) 基于不同审计策略模拟定期发生的审计事件。给定审计频率 β ，算法从时间 0 开始每隔 $1/\beta$ 时间生成一条审计记录。对于随机审计和更新审计策略，算法在每次审计时随机选中一个云 $cloud_no$ 再随机选中某个校验组 $file_no$ 并重复 k 次。对于批量审计策略，算法在每次审计时随机选中一个校验组 $file_no$ 再随机选中该校验组中不重复的 k 个分片。使用二元组唯一标记审计记录 $audit_no = (time_curr, slice_no)$ ，即在时间 $time_curr$ 分片 $slice_no$ 被审计。
- (3) 分片修复。给定由所有分片损坏事件构成的故障列表以及由所有审计记录构成的审计列表，我们可以判断是否发生分片修复事件。在随机审计和批量审计策略下，假如我们发现某分片在时间 $time_1$ 发生损坏，而该分片在之后的某个时间 $time_2$ 被审计，则尝试修复该分片。若修复时使用的 k 个

分片都为正确分片则修复成功，反之修复失败。在更新审计策略下，假如我们发现某分片在时间 $time_1$ 发生损坏，而该分片在之后的某个时间 $time_2$ 被审计，则我们将该分片所在校验组的全部分片标记为已修复。更新审计能够提前修复一部分尚未审计到的分片，因而能够提升系统可靠性。

- (4) 数据丢失。给定由所有分片损坏事件构成的故障列表以及由所有审计记录构成的审计列表，判断目标校验组是否发生数据丢失事件。假设目标校验组已经有 2 个分片发生损坏且这两个分片都没有被修复，这时在 $time_{dl}$ 校验组中的第三个分片损坏，则我们认为多云存储系统中发生了数据丢失事件。此时仿真程序输出运行结果 $MTTDL=time_{dl}$ 。

事件（1）模拟了单个云服务商发生可被多云系统感知到的数据损坏的过程。
 事件（2）模拟了多云存储系统定期执行审计的过程，并模拟了不同的审计策略。
 事件（3）模拟了多云存储系统执行数据修复的过程，并模拟了不同的修复策略。
 事件（4）模拟了目标校验组发生数据丢失事件。我们运行 50 轮模拟算法求得平均 MTTDL 值作为程序输出结果。

5.5.2 实验结果

单云与多云存储可靠性对比。多云存储系统将多个云服务商进行整合，虽然提供了更高的安全性，但是系统发生故障的可能性也随之增加。若系统内有 n 个云服务商，每个云服务商的故障率为 λ ，则多云存储系统的故障率为 $n\lambda$ 。理论上，冗余存储策略能够在一定程度上降低故障率，但是仅仅依靠冗余存储策略能获得的降低程度有限。在本小节开始提出的三个问题中，问题 1 想要探究使用多个云服务商构建的存储系统的可靠性会有什么变化，本组实验正是为了回答这一问题。在不运行审计机制的情况下，我们从两个维度对比单个云服务商与多云存储系统的可靠性差异。（1）将单个云服务商的存储可靠性（即 MTTF）从 1000 天逐渐增加到 10000 天，测试多云存储系统整体的 MTTDL。（2）在不改变编码容错能力的前提下将多云系统规模逐渐增大，测试多云存储系统整体的 MTTDL。我们分别测试了云服务商个数为 4、6、8、16，冗余编码参数为 $(2,4), (4,6), (6,8), (14,16)$ 这四种情况。

马尔可夫实验结果如图 5.11 所示。可以看到，在不运行审计机制的情况下多云存储系统的可靠性与存储开销有直接关系。定义存储冗余度为 $(n-k)/k$ ，编码参数为 $(2,4)$ 的多云系统的冗余度达到了 1，这相当于两副本存储系统的冗余度。其余三个多云系统的冗余度分别为 $\frac{1}{2}, \frac{1}{3}, \frac{1}{7}$ 。在马尔可夫模型的预测结果中，编码

参数(2,4)多云系统的可靠性稍好于单云，其余三个系统的可靠性低于单云。伴随冗余度的降低系统的存储开销降低，可靠性也逐渐降低。图5.12给出的蒙特卡洛仿真结果印证了马尔可夫模型的结论。总的来看，虽然单云可靠性的提升能够使得多云系统整体可靠性有所提升，但始终低于构成多云的每个单云可靠性。

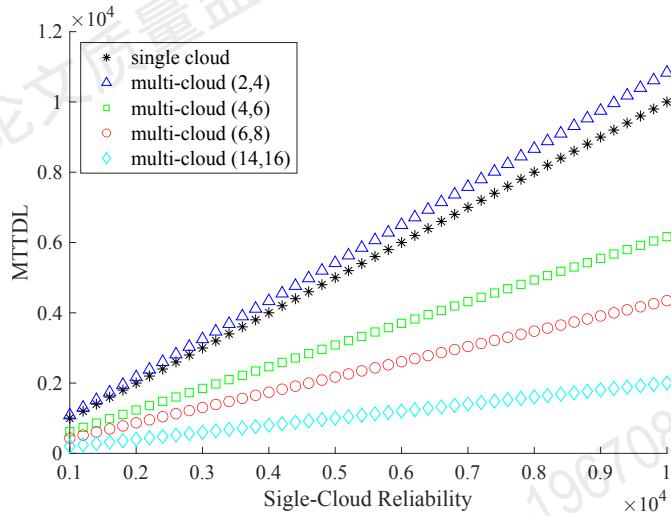


图 5.11 单云与多云存储系统可靠性对比（马尔可夫模型结果）

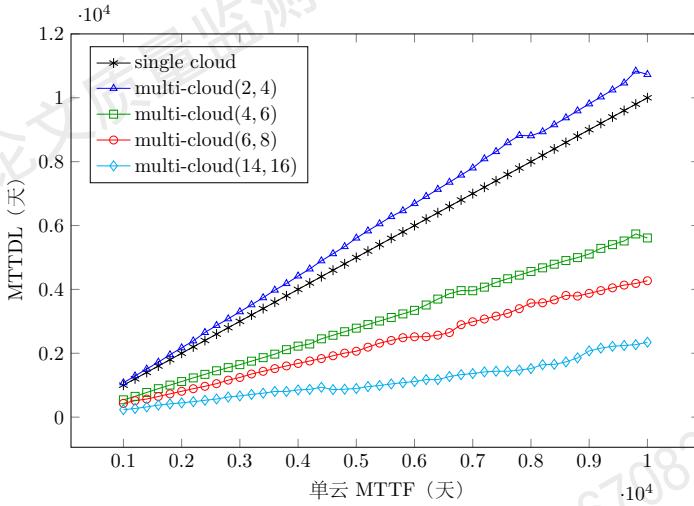


图 5.12 单云与多云存储系统可靠性对比（蒙特卡洛仿真结果）

由此看出，在不采取额外的数据审计和数据修复等可靠性保护措施的情况下，多云存储系统整体的可靠性无法达到原本单云存储系统的水平。提升多云存储系统的可靠性水平是十分重要的。

不同审计策略对比。本组实验用于回答本小节开始提出的第二个问题，探究审计机制对于多云存储系统可靠性的提升效果。我们将测试运行审计机制的多云存储系统的可靠性，然后对比实施不同审计策略时系统可靠性的变化。实验结果和分析结论将用于指导多云存储系统审计策略的制定。在本组实验中，我们保持多云系统的规模、冗余编码参数以及审计频率为恒定值。具体地，设置参数 $(k, n) = (4, 6)$ ，审计频率为 40 次/天，即参数 $\beta = 40$ 。实验过程中将单云可靠性由 1000 天增大至 10000 天，即参数 λ 由 $1/1000$ 逐渐降低至 $1/10000$ ，分别测试运行不同审计策略的多云存储系统的 MTTDL。

图5.13为马尔可夫模型计算的结果。可以看出，未实施审计机制的多云存储系统的可靠性远远低于实施审计机制的多云存储系统。更新审计（renew audit）的效果较之随机审计（random audit）和批量审计（batch audit）有成倍提升，而随机审计和批量审计的效果则较为接近（后者稍好）。蒙特卡洛模拟程序得出的测试结果印证了上述结论。如图5.14所示，在不运行审计机制的情况下，多云存储系统的可靠性处在较低水平。随机审计和批量审计使得系统可靠性有了较为明显的提升，其中批量审计的可靠性已经基本接近于单云存储系统可靠性。更新审计对系统可靠性的提升幅度更大。

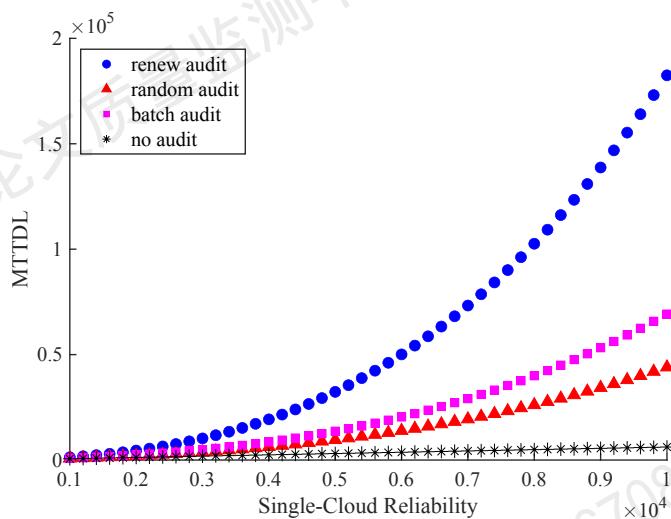


图 5.13 不同审计策略下多云存储系统可靠性对比（马尔可夫模型结果）

结合上一组实验的结论，我们认为审计机制对于多云存储系统是十分 important且必要的。另外，通过三种审计策略的对比可以看出，不同的数据修复策略对系统可靠性的影响较大。若修复过程中不验证所使用的数据分片是否正确，那

么便存在修复失败的可能性（且在随机审计策略下修复失败的可能性最大），而这将直接影响系统可靠性。

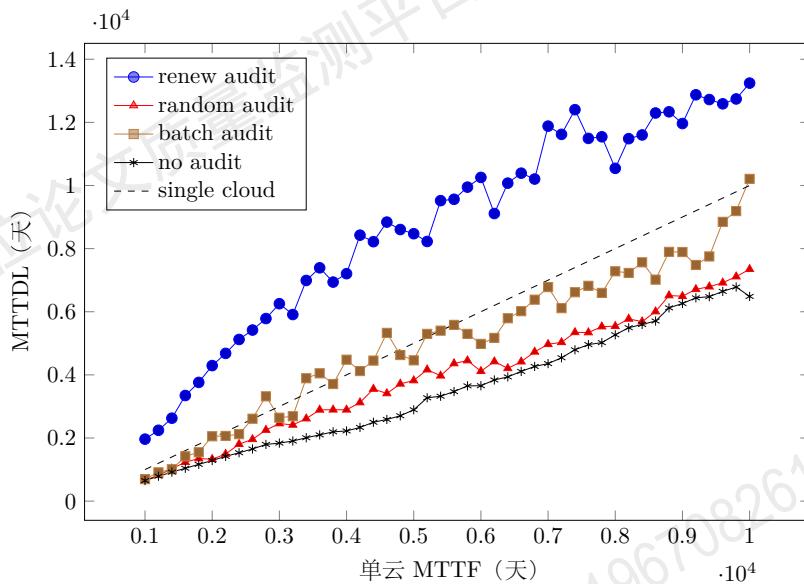


图 5.14 不同审计策略下多云存储系统可靠性对比（蒙特卡洛仿真结果）

审计频率对可靠性的影响。从概率意义上讲，提高审计频率可以使云存储系统更早发现分片损坏，从而提升系统 MTTDL。然而过高的审计频率会给系统造成较大工作负担，影响系统响应速度。那么，应该如何为多云存储系统设定合适的审计频率呢？在这部分实验中我们想要回答这一问题，并探讨审计频率与多云存储系统可靠性之间的关系。分析结论将用于指导多云存储系统审计频率的确定。在本组实验中，我们保持单个云服务商的可靠性、多云系统规模以及冗余编码参数为恒定值，设定单云的存储可靠性为 3650 天（约为 10 年），即参数 $\lambda = 1/3650$ ，设置参数 $(k, n) = (4, 6)$ 。实验过程中将审计频次逐渐增加，分别测试运行不同审计策略的多云存储系统的 MTTDL。

图 5.15 为马尔可夫模型计算的结果。在审计频次由 10 逐渐增大到 200 的过程中系统 MTTDL 持续上升。不过，更新审计上升的速度更快而随机审计和批量审计上升速度较慢。图 5.16 为蒙特卡洛模拟程序得出的结果。更新审计在审计频次为 0–40 这段区间上升速度最快，之后上升速度逐渐放缓，并在审计频次为 40–50 次/天时接近系统最大 MTTDL 值⁶。更新审计在审计频次约为 100 次/天时

⁶ 在仿真程序中我们设置了系统 MTTDL 的上限为 1 万天（作为程序运行的终止点）。

达到峰值。随机审计和批量审计的情况有所不同，伴随审计频次的增大，系统MTTDL在持续缓慢上升。其中批量审计在审计频次为40–50次/天时已经达到单云可靠性水平（即3650天）。

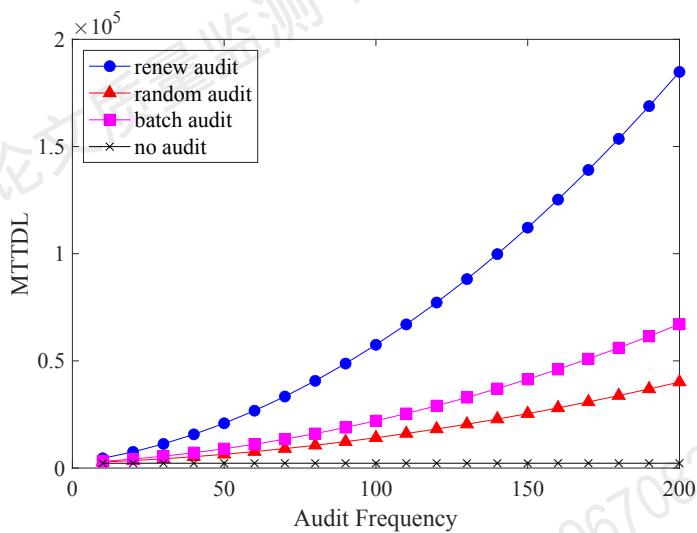


图 5.15 不同审计频率下的可靠性对比 (马尔可夫模型结果)

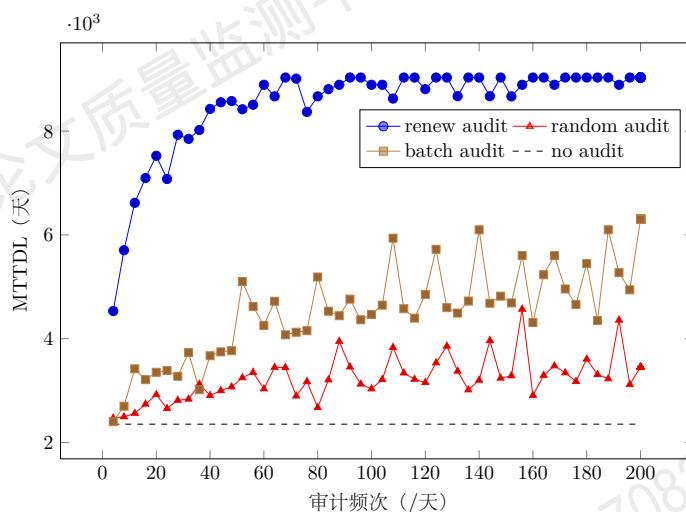


图 5.16 不同审计频率下的可靠性对比 (蒙特卡洛仿真结果)

上述结果指出，更新审计频次的合适范围是40–50。批量审计频次应保持在40次/天以上。不过随机审计的效果不佳，难以达到单云可靠性水平。下文我们将会结合网络流量分析给出更为全面的结果。

存储冗余度对可靠性的影响。在进行单云多云可靠性对比实验时我们发现，对于不运行审计机制的多云系统，在编码容错能力不变的前提下，伴随存储冗余度的降低系统实际的可靠性也会降低。本组实验旨在分析运行审计机制的多云存储系统是否具有同样的特性。在实验过程中，保持单个云服务商的可靠性以及审计频率不变，设置单云可靠性为 3650 天，审计频率为 40 次/天。实验过程中改变云服务商个数以及编码参数 (k, n) 的值但始终保持 $n - k = 2$ ，分别测试运行不同审计策略的多云存储系统的 MTTDL。图5.17为马尔可夫模型计算的结果。伴随云服务商个数的增加和存储冗余度的降低，系统 MTTDL 呈现下降的趋势，并逐渐与无审计机制的多云存储系统趋同。图5.18蒙特卡洛模拟程序得出的仿真结果印证了上述结论。实验结果表明，系统规模会在一定程度上影响多云存储系统的可靠性。

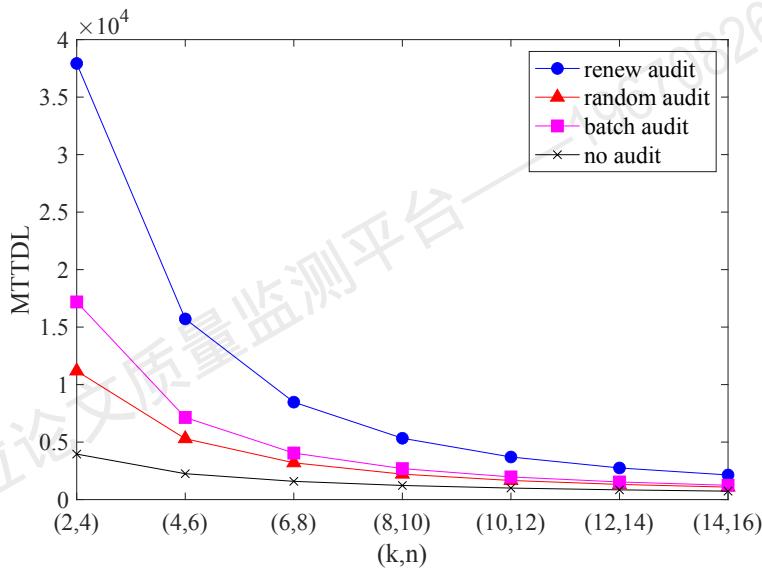


图 5.17 不同系统规模下的可靠性对比（马尔可夫模型结果）

当系统规模增大时，一个比较好的解决办法是提高审计频率。如图5.19所示给出一个例子。当多云系统共有 6 个云服务商且采用参数为 $(4, 6)$ 的冗余编码时，审计频次为约 40 次/天时测得 MTTDL 为 3035.597 天。假如系统扩张到 10 个云服务商且并且采用 $(8, 10)$ 的冗余编码时，将审计频次提高到约 80 次/天便可保证同样的 MTTDL 水平。因此，通过调节审计频次可以保证多云存储系统可靠性水平的稳定。

网络流量分析。以上实验详细分析了不同审计策略以及不同审计频次对于

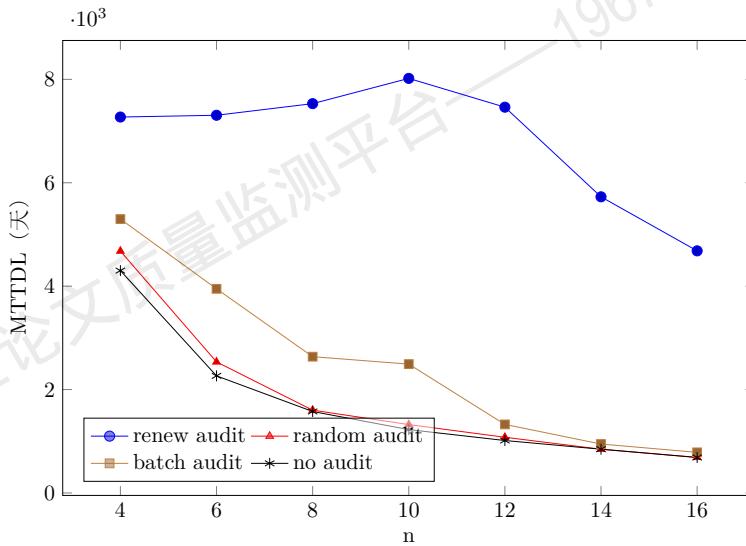


图 5.18 不同系统规模下的可靠性对比（蒙特卡洛仿真结果）

多云存储系统可靠性的提升效果。接下来分析不同审计策略在执行数据修复过程中耗费的网络流量。结合网络流量分析，我们能够制定更为高效的审计方案。

数据修复可分为三个阶段，首先是从云端下载分片，然后是重构出原始数据并计算损坏数据的正确值，最后是将修复出的数据重新上传到云端。我们进行如下假设：

- 上传/下载开销：上传或下载一个分片耗费的网络流量是 d ；
- 数据修复开销：执行基于纠删码的数据修复的计算开销是 r ；
- 完整性校验开销：执行一次分片完整性校验的开销是 c 。

表格5.1给出了三种审计策略的开销统计。(1) 随机审计。随机审计在执行数据修复时需要从云端下载 k 个分片，因此分片下载的流量开销是 kd 。下载分片后不执行完整性校验，因此校验开销为 0。随机审计只修复已检测到的损坏分片，因此重新上传阶段的流量开销是 d 。(2) 批量审计。批量审计在执行数据修复时需要从云端下载 k 个分片，因此分片下载的流量开销是 kd 。下载分片后不执行完整性校验，因此校验开销为 0。批量审计一次性验证同一校验组的 k 个分片，因此有可能一次审计检测到 2 个损坏分片，重新上传阶段的最大流量开销是 $2d$ 。(3) 更新审计。更新审计在执行数据修复前要检查所下载的分片是否正确，假如下载到错误的则需重新下载，最坏情况下需要下载 n 个分片，并执行 n 次完整性校验。更新审计将整个校验组的损坏分片全部修复，最坏情况下需要

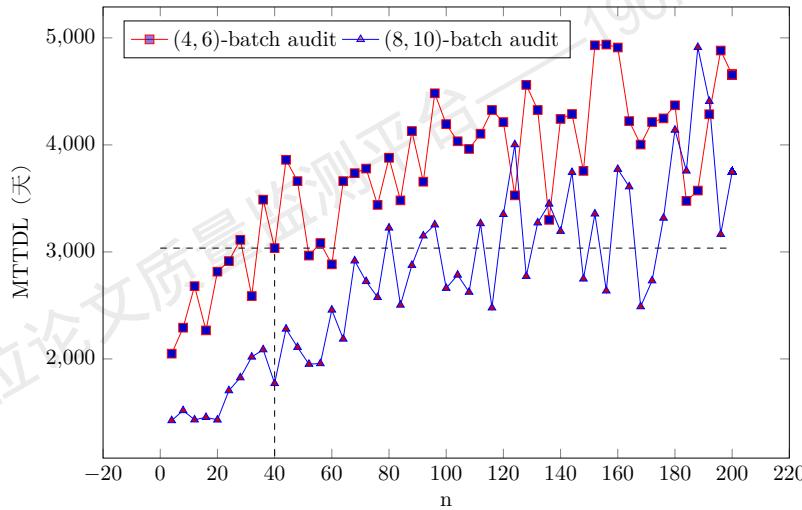


图 5.19 示例：通过提高审计频率提升系统可靠性

审计策略	分片下载	校验	数据重构	数据上传
随机审计	kd	0	r	d
批量审计	kd	0	r	$2d$
更新审计	nd	nc	r	$2d$

表 5.1 不同审计策略的开销对比。

修复 2 个分片，因此重新上传阶段的流量开销最大是 $2d$ 。我们仅考虑分片下载和上传阶段的网络流量开销，可以得到如下结果：

$$\text{随机审计流量开销} = (k+1)d$$

$$\text{批量审计流量开销} = (k+2)d$$

$$\text{更新审计流量开销} = (n+2)d$$

5.5.3 实验结论

根据上述实验和分析结果，我们首先可以肯定，运行远程数据审计机制是提升多云存储系统可靠性的重要手段。不过需要注意的是，不同审计策略所带来的可靠性提升效果以及数据修复时的开销有一定差异。随机审计策略效果最差，且数据修复时的流量开销也不低，因此下文不再进行讨论。批量审计策略虽效果不如更新审计策略，但胜在数据修复时的开销较小。更新审计策略虽效果很好，但执行数据恢复的流量开销和计算开销都较大。我们可以根据不同数

据种类对于可靠性的不同需求，再结合不同审计策略的特点，制定适合的方案。

接下来，我们基于第三节介绍的多云散布系统，结合上述可靠性分析和流量分析结果，给出一个安全、可靠、高效的远程数据审计方案。

在原系统中对文件和密钥的散布过程是分别执行的，数据重构过程也是分别执行的。同时，密钥和文件对于安全性的需求并不相同。密钥相对于经过加密的文件属于更为敏感的数据，应该被给予更高的安全防护级别。基于以上两点，我们认为对文件和密钥的审计也可以分别进行。更新审计能够保证更高的数据可靠性，但是执行数据修复时的开销较大，因此比较适合用于审计数据量较小但安全级别较高的数据，例如加密密钥分片。而批量审计虽然可靠性稍弱，但是执行数据修复时的开销小，并且当审计频率设置为 40–50 次/天时能够达到与单云可靠性相近水平。因此，批量审计可以应用于文件分片。基于上述考虑，我们为多云审计系统制定了如下的审计方案。

(1) 定期执行的审计。系统后台定期运行审计过程，对文件采用批量审计，对密钥采用更新审计。假如发现文件分片错误则系统后台执行文件分片修复过程，向云端获取 k 文件分片并计算出损坏分片的值，重新上传到云端。若需进一步提升文件可靠性，可在修复过程中加入对文件分片的完整性验证。假如发现密钥分片错误则系统后台执行密钥更新过程，先获取 k 个正确的密钥分片以重构出密钥 K ，然后重新生成一组全新的密钥分片以覆盖掉云上原本的分片。密钥分片更新过程不仅能够修复已损坏的数据，还能预防密钥泄露、抵御针对密钥的持续性攻击。由于密钥的数据量较小，密钥更新过程并不会非常耗时。结合第三章的内容，我们可以将定期执行的审计策略以智能合约的方式在区块链上公开，并实现自动化审计。并且，用户可以自定义文件和密钥的审计频次。例如，用户文件的安全性需求较高，则可以为文件分片设置更高的审计频次。再如，用户可以设置定期执行密钥分片更新操作，从而进一步提高密钥的安全性和可靠性。

(2) 伴随下载过程执行的审计。当用户想要下载文件时，系统先下载目标文件所对应的 k 个文件分片和 k 个密钥分片，然后批量审计这 $2k$ 个数据分片。这能够在数据下载过程中帮助用户定位分片级别的数据错误，避免了使用错误的分片重构出错误的数据，从而可以辅助和加速数据重构和数据修复的过程，也能够使系统在故障状态下快速恢复原有的安全防护水平。

第六节 本章小结

本章为引入远程数据审计机制的多云存储系统提出了一套可靠性评价方法。现有可靠性分析多针对单个云存储系统内部的磁盘运行故障、潜在块损坏等等。我们将研究对象扩展为多云存储系统。对比单云存储系统，多云存储系统的结构更为复杂，故障更为多元化。我们从系统整体可靠性的角度进行分析，评价单个云服务商运行的可靠性机制失效导致多云存储系统发生数据丢失的情况。我们建立连续时间马尔可夫模型来模拟多云存储发生故障及执行修复的过程。马尔可夫模型计算出不同审计策略和参数设置下的系统可靠性。我们设计了蒙特卡洛算法并实现了仿真程序模拟多云存储系统运行过程。仿真程序的运行结果验证了模型计算结果的准确性。

我们进行了如下三方面研究。(1) 评价远程数据审计机制对多云存储系统数据可靠性的提升效果。在不实施审计机制等可靠性保护措施的情况下，多云存储系统的可靠性无法达到原本单云存储系统的水平。而引入审计机制使得系统可靠性有了大幅提升，甚至远远超过单云系统可靠性。(2) 对比不同审计策略在可靠性提升和资源消耗等方面的差异。不同的数据选取策略和修复策略的效果差异明显。批量审计效果较好且开销较小。在审计频率为 40–50 时批量审计能够达到单云存储系统可靠性水平。批量审计比较适合对安全性要求不太高但是数据量比较大的数据，例如文件分片。更新审计效果最好但开销也最大，因此比较适合审计数据量小且安全需求高的数据，例如密钥分片。(3) 结合上述分析结论为多云存储系统提出高效的区块链审计方案，以较低的成本保证较高的数据可靠性。

本文假设单云故障发生的时间服从指数分布，未来一个可能的研究方向是将概率分布函数扩展至韦布分布、伽马分布等等，以使得可靠性评价模型能够适用于更多场景。

第六章 总结与展望

第一节 本文工作总结

随着全球进入大数据时代，云存储服务得到广泛普及。一方面，越来越多企业、机构和个人开始构建面向社会大众的云服务平台。云存储作为其中的基础模块提供数据存储和数据访问等功能，是云计算的基石。另一方面，随着数据呈现爆炸式增长，个人电脑或是单台服务器已无法满足数据存储的需求。而自建服务器集群不仅维护复杂而且成本较高，已逐渐被企业和机构摒弃。云存储服务凭借高效且低廉的特性逐渐受到越来越多欢迎。然而这也引发新的问题。云上存储的很多数据涉及个人隐私、商业机密或是组织机构的重要信息，一旦丢失或损坏将造成巨大的经济损失，引发不良社会影响。因此，提高云存储服务的安全性和可靠性、保护数据完整性成为十分重要的研究问题。

远程数据审计技术致力于解决上述问题并为云存储用户提供安全高效的数据完整性保护方案。审计者采用抽查的方式规律性地对云上数据发起挑战，云服务商必须按照挑战内容计算完整性证明，最终审计者验证证明并得出审计结果。理论上，审计机制配合适当的数据修复机制可以很好地保护云上数据的完整性。然而从系统实施和实际使用的角度，目前提出的方案仍旧存在一些不足。首先，目前普遍采用的中心化审计模型存在安全隐患。审计规则不公开、审计过程不透明、审计结果不可验证导致审计过程中出现的问题无法被及时发现，更给了攻击者和恶意审计者可乘之机。另外，目前缺少一套评价手段为审计策略的制定提供指导。从数据选取策略的选择，到数据修复策略的制定，再到审计频次的确定，都需要理论分析结果作为重要依据。上述问题阻碍了远程数据审计机制在云存储系统的实际应用。

本学位论文针对大规模云存储系统，研究安全、可信、高效的分布式数据完整性审计方案。具体来说，我们开展了如下三个方面的工作：

1. 提出去中心化的审计模型，依靠算法、协议而不是第三方来保证审计的可靠可信。为彻底解决传统中心化审计的单点故障问题，基于区块链技术构建分布式审计网络，由所有云存储用户作为审计者发起挑战或验证完整性

证明，并通过分布式共识得出审计结果。设计审计者选举算法避免恶意用户操控审计结果，确保审计者中大部分为诚实用户。设计激励机制提高用户参与度、维持网络活性，从而进一步保证审计安全性。为实现审计的公开透明以及可监督，将审计协议实现在智能合约中实现自动化执行，同时审计规则和审计记录以区块链交易的形式记录上链、向所有用户公开。为避免审计记录公开可能引发的隐私泄露风险，设计具有零知识隐私保护性质的分布式审计协议。

2. 提出适用于云存储的审计联盟链系统模型，并针对性地提出区块链存储优化方案以提升系统可扩展性和安全性。在设计系统模型时采用审计网络与区块链网络合一的思想，使得云存储用户既参与审计又参与区块链维护。无需外部区块链网络提供服务支持，全部审计记录由云存储用户维护和存储，使得审计结果更加可靠可信。为解决不同云存储用户拥有资源参差不齐的问题，针对云存储场景提出具有分层结构的审计联盟链系统模型，为每类节点分配不同等级的任务和权利。服务节点参与共识并负责维护区块链、存储区块链数据；轻节点只使用区块链服务，不维护数据。以此来提升系统的可扩展性。更进一步，为解决节点存储开销过大的问题，设计动态弹性存储分片方案以进行存储模式的优化。根据实时区块访问热度动态调整区块的副本个数、存储位置和存储模式，从而在数据可靠性、数据访问性能和节点存储开销间取得平衡。
3. 为引入审计机制的多云存储系统设计可靠性评价方法，用于对比不同审计策略的实际效果从而为策略制定提供指导。相较于单云存储系统，多云存储系统的复杂度更高、引发故障的因素更多。因此本文从系统整体可靠性入手进行分析。使用指数分布函数模拟单个云存储系统上发生的故障事件，使用马尔可夫模型模拟云存储系统的运行状态，使用蒙特卡洛算法来模拟多云存储系统动态运行和修复的过程。仿真程序的运行结果验证了马尔可夫的计算结果。分析结论证明审计机制对于提升多云系统可靠性有非常好的效果。结合多角度的可靠性分析和网络流量分析，我们为多云存储系统制定了恰当的区块链审计策略。

本文研究的三个方面密切相关，从不同角度提升远程数据审计方案的安全性和可实施性。工作 1 从审计的可信性入手，提出基于区块链的分布式审计模型和审计协议；工作 2 从审计的可行性入手，提出高可扩展的审计联盟链系统

模型；工作3从审计机制评价入手，提出审计系统可靠性评价方法。三个部分相互配合，共同融入云存储系统，可以帮助我们构建安全、可信、高效的云存储数据审计系统。

第二节 未来工作展望

为了保证审计的公开透明以及可监督，我们采用了将审计记录公开在区块链上的设计。为避免可能的隐私泄露风险，我们设计了基于零知识证明的审计协议。不过，零知识证明技术存在一定的局限性。例如，由于生成证明时的计算复杂度较高，目前主要使用哈希和签名这两种算法来构建零知识证明算术电路。这使得我们在设计审计协议时存在一些局限。未来一个可考虑的研究方向是如何在保留隐私保护性质的前提下设计更为复杂的审计协议以满足不同的审计需求，如支持数据更新操作、支持证明压缩等等。如何平衡隐私保护、协议运行效率与协议的普适性是一个有意义的研究内容。

针对审计联盟链系统内服务节点存储开销过高的问题，我们设计了动态弹性存储分片方案。该方案的主要思想是降低全网存储的区块链副本数量以节约存储空间。每一个存储组内的节点共同维护一份区块链副本，通过节点间协作动态调整区块副本个数、存储位置和存储模式，以此实现数据可靠性、存储开销和访问性能之间的平衡。另一个可考虑的思路是对区块链数据进行压缩或者重复数据删除。这可以从根本上大幅缩减区块链数据量，从而达到降低节点存储开销的目的。不过，数据压缩或者重复数据删除算法必须仔细进行设计以避免削弱区块链系统安全性，并且需要针对方案安全性进行理论证明。

最后，本文提出的多云审计系统可靠性评价方法基于指数分布假设，并使用了马尔可夫模型模拟多云存储系统的运行状态。一方面，马尔可夫模型能够较为精准地刻画系统执行数据修复的过程，并且非常适合用来对比不同策略之间的效果差异。另一方面，综合考虑多云存储系统内故障的多元性和突发性，使用指数分布刻画系统平均故障率是更为合适的。未来一个可考虑的研究方向是将故障分布函数扩展至韦布分布、伽马分布等等。使用这些函数可以更为精准地刻画磁盘老化、外部攻击等现象，并分析这些现象对于审计系统可靠性的影响。结合系统整体可靠性分析以及针对系统重要模块的可靠性分析，我们能够设计更为有效的云存储数据审计方案。

第六章 总结与展望

参考文献

- [1] 华经产业研究院. 2021-2026年中国医疗云服务行业发展前景预测及投资战略研究报告. <https://www.huaon.com/channel/medicine/667816.html>, 2020.
- [2] 前瞻产业研究院. 2020年中国政务云行业市场现状及发展前景分析. <https://bg.qianzhan.com/report/detail/300/200806-9aa9ec05.html>, 2020.
- [3] Christian Cachin, Idit Keidar, and Alexander Shraer. Trusting the cloud[J]. *SIGACT News*, 40(2):81–86, 2009.
- [4] SC Media. Open AWS S3 bucket exposes private info on thousands of Fedex customers. <https://www.scmagazine.com/>, 2018.
- [5] Peter Judge. Microsoft confirms Azure outage was human error. <https://www.datacenterdynamics.com/en/news/>, 2014.
- [6] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores[C]. In *Proceedings of the 14th ACM Conference on Computer and Communications Security(CCS)*, page 598–609, 2007.
- [7] Ari Juels and Burton S Kaliski. PoRs: Proofs of retrievability for large files[C]. In *Proceedings of the 14th ACM Conference on Computer and Communications Security(CCS)*, page 584–597, 2007.
- [8] Lanxiang Chen. Using algebraic signatures to check data possession in cloud storage[J]. *Future Generation Computer Systems*, 29(7):1709–1715, 2013.
- [9] David Cash, Alptekin Küpcü, and Daniel Wichs. Dynamic proofs of retrievability via oblivious RAM[J]. *Journal of Cryptology*, 30(1):22–57, 2017.
- [10] Chris Erway, Alptekin Küpcü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession[C]. In *Proceedings of the 16th ACM Conference on Computer and Communications Security(CCS)*, page 213 – 222, 2009.

- [11] Qingji Zheng and Shouhuai Xu. Fair and dynamic proofs of retrievability[C]. In *Proceedings of the 1st ACM Conference on Data and Application Security and Privacy(CODASPY)*, page 237–248, 2011.
- [12] Hovav Shacham and Brent Waters. Compact proofs of retrievability[J]. *Journal of Cryptology*, 26(3):442–483, 2013.
- [13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <http://www.bitcoin.org/bitcoin.pdf>, 2008.
- [14] Bin Liu, Xiao Liang Yu, Shiping Chen, Xiwei Xu, and Liming Zhu. Blockchain based data integrity service framework for IoT data[C]. In *Proceedings of the 24th IEEE International Conference on Web Services(ICWS)*, pages 468–475, 2017.
- [15] Yining Qi and Huang Yongfeng. DIRA: Enabling decentralized data integrity and reputation audit via blockchain[J]. *Science China*, 62(04):698–701, 2019.
- [16] Yang Xu, Ju Ren, Yan Zhang, Cheng Zhang, Bo Shen, and Yaoyue Zhang. Blockchain empowered arbitrable data auditing scheme for network storage as a service[J]. *IEEE Transactions on Services Computing*, 13(2):289–300, 2020.
- [17] Pei Huang, Kai Fan, Hanzhe Yang, Kuan Zhang, Hui Li, and Yintang Yang. A collaborative auditing blockchain for trustworthy data integrity in cloud storage system[J]. *IEEE Access*, 8:94780–94794, 2020.
- [18] Yuan Zhang, Chunxiang Xu, Xiaodong Lin, and Xuemin Shen. Blockchain-based public integrity verification for cloud storage against procrastinating auditors[J]. *IEEE Transactions on Cloud Computing*, 9(3):923–937, 2021.
- [19] L J García Villalba, Kai He, Chunxiao Huang, Jiaoli Shi, Xinrong Hu, and Xiying Fan. Enabling decentralized and dynamic data integrity verification for secure cloud storage via T-merkle hash tree based blockchain[J]. *Mobile Information Systems*, pages 1–17, 2021.
- [20] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Perma-coin: Repurposing bitcoin work for data preservation[C]. In *Proceedings of the 35th IEEE Symposium on Security and Privacy(SP)*, pages 475–490, 2014.
- [21] Binanda Sengupta, Samiran Bag, Sushmita Ruj, and Kouichi Sakurai. Retricoin: Bitcoin based on compact proofs of retrievability[C]. In *Proceedings of the 17th*

- International Conference on Distributed Computing and Networking(ICDCN)*, pages 1–10, 2016.
- [22] Protocol Labs. Filecoin: A decentralized storage network. Technical report, Protocol Labs, 2017. <https://filecoin.io/filecoin.pdf>.
- [23] Protocol Labs. Proof-of-replication. Technical report, Protocol Labs, 2017. <https://filecoin.io/proof-of-replication.pdf>.
- [24] Mohammed A AlZain, Eric Pardede, Ben Soh, and James A Thom. Cloud computing security: From single to multi-clouds[C]. In *Proceedings of the 45th Hawaii International Conference on System Sciences(HICSS)*, pages 5490–5499, 2012.
- [25] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. OceanStore: An architecture for global-scale persistent storage[C]. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS)*, page 190–201, 2000.
- [26] S Rhea, C Wells, P Eaton, D Geels, B Zhao, H Weatherspoon, and J Kubiatowicz. Maintenance-free global data storage[J]. *IEEE Internet Computing*, 5(5):40–49, 2001.
- [27] Mark W Storer, Kevin M Greenan, Ethan L Miller, and Kaladhar Voruganti. Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage[C]. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies(FAST)*, pages 1–16, 2008.
- [28] Mark W Storer, Kevin M Greenan, Ethan L Miller, and Kaladhar Voruganti. POTSHARDS—a secure, recoverable, long-term archival storage system[J]. *ACM Transactions on Storage*, 5(2):1–35, 2009.
- [29] Garth R Goodson, Jay J Wylie, Gregory R Ganger, and Michael K Reiter. Efficient byzantine-tolerant erasure-coded storage[C]. In *Proceedings of the 34th International Conference on Dependable Systems and Networks(DSN)*, pages 135–144, 2004.
- [30] Arun Subbiah and Douglas M Blough. An approach for fault tolerant and secure

- data storage in collaborative work environments[C]. In *Proceedings of the 1st ACM Workshop on Storage Security and Survivability(StorageSS)*, page 84–93, 2005.
- [31] Tahoe-LAFS. Welcome to the least-authority file store. <https://tahoe-lafs.org/trac/tahoe-lafs>, 2021.
- [32] George R Blakley. Safeguarding cryptographic keys[C]. In *Proceedings of the 1st International Workshop on Managing Requirements Knowledge(MARK)*, pages 313–318, 1979.
- [33] Adi Shamir. How to share a secret[J]. *Communications of the ACM*, 22(11):612–613, 1979.
- [34] Michael O Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance[J]. *Journal of the ACM*, 36(2):335–348, 1989.
- [35] Elwyn R Berlekamp. *Algebraic coding theory (revised edition)*. World Scientific, 2015.
- [36] Hugo Krawczyk. Secret sharing made short[C]. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology(CRYPTO)*, page 136–146, 1994.
- [37] Jason K Resch and James S Plank. AONT-RS: Blending security and performance in dispersed storage systems[C]. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies(FAST)*, pages 1–12, 2011.
- [38] Ronald L Rivest. All-or-nothing encryption and the package transform[C]. In *Proceedings of the 4th International Workshop on Fast Software Encryption(FSE)*, volume 1267, pages 210–218, 1997.
- [39] F Jessie MacWilliams and Neil J A Sloane. *The Theory of Error-Correcting Codes*. North-Holland Publishing Co., 1977.
- [40] Mingqiang Li, Chuan Qin, Patrick P C Lee, and Jin Li. Convergent dispersal: Toward storage-efficient security in a cloud-of-clouds[C]. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems(HotStorage)*, pages 1–5, 2014.
- [41] Lu Shen, Shifang Feng, Jinjin Sun, Zhongwei Li, Gang Wang, and Xiaoguang Liu. CloudS: A multi-cloud storage system with multi-level security[C]. In *Pro-*

- ceedings of the 15th International Conference on Algorithms and Architectures for Parallel Processing(ICA3PP), pages 703–716, 2015.
- [42] Mingqiang Li, Chuan Qin, Jingwei Li, and Patrick P C Lee. CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal[J]. *IEEE Internet Computing*, 20(3):45–53, 2016.
- [43] Katarzyna Kapusta, Gerard Memmi, and Hassan Noura. POSTER: A keyless efficient algorithm for data protection by means of fragmentation[C]. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security(CCS)*, page 1745–1747, 2016.
- [44] Napster. The napster homepage. <https://us.napster.com/home>, 2022.
- [45] Petros Maniatis, Mema Roussopoulos, Thomas J Giuli, David Stuart Holmes Rosenthal, and Mary Baker. The LOCKSS peer-to-peer digital preservation system[J]. *ACM Transactions on Computer Systems*, 23:2–50, 2005.
- [46] Steven Hand and Timothy Roscoe. Mnemosyne: Peer-to-peer steganographic storage[C]. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems(IPTPS)*, page 130–140, 2002.
- [47] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system[J]. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003.
- [48] Douglas J Santry, Michael J Feeley, Norman C Hutchinson, and Alistair C Veitch. Elephant: The file system that never forgets[C]. In *Proceedings of the 7th Workshop on Hot Topics in Operating Systems(HOTOS)*, pages 2–7, 1999.
- [49] Peter Druschel and Antony Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility[C]. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems(HOTOS)*, pages 75–80, 2001.
- [50] Bram Cohen. Incentives build robustness in BitTorrent. <http://www.bittorrent.org/bittorrentecon.pdf>, 2003.
- [51] H Yamamoto, D Maruta, and Y Oie. Replication methods for load balancing on distributed storages in P2P networks[C]. In *Proceedings of the 2005 Symposium on Applications and the Internet(SAINT)*, pages 264–271, 2005.
- [52] Eng Keong Lua, J Crowcroft, M Pias, R Sharma, and S Lim. A survey and comparison of peer-to-peer overlay network schemes[J]. *IEEE Communications*

- Surveys & Tutorials*, 7(2):72–93, 2005.
- [53] Moufida Rahmani and Mahfoud Benchaba. A comparative study of replication schemes for structured P2P networks[C]. In *Proceedings of the 9th International Conference on Internet and Web Applications and Services(ICIW)*, page 147 – 158, 2014.
- [54] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web[C]. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing(STOC)*, page 654 – 663, 1997.
- [55] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network[J]. *SIGCOMM Computer Communication Review*, 31(4):161–172, 2001.
- [56] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications[C]. In *Proceedings of the 15th Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications(SIGCOMM)*, page 149–160, 2001.
- [57] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems[C]. In *Proceedings of the 2001 IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing(Middleware)*, pages 329–350, 2001.
- [58] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly[C]. In *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing(PODC)*, page 183–192, 2002.
- [59] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks[C]. In *Proceedings of the 16th International Conference on Supercomputing(ICS)*, page 84–95, 2002.
- [60] Sabu M Thampi. Survey of search and replication schemes in unstructured P2P networks[J]. *Network Protocols & Algorithms*, 2(1):93–131, 2010.
- [61] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W Hong. Freenet: A

- distributed anonymous information storage and retrieval system[C]. In *Proceedings of the International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, page 46–66, 2001.
- [62] Edith Cohen and Scott Shenker. Replication strategies in unstructured peer-to-peer networks[J]. *SIGCOMM Computer Communication Review*, 32(4):177–190, 2002.
- [63] Jussi Kangasharju. *Internet content distribution[D]*. PhD thesis, TU Darmstadt, 2002.
- [64] Jussi Kangasharju, Keith W Ross, and David A Turner. Adaptive content management in structured P2P communities[C]. In *Proceedings of the 1st International Conference on Scalable Information Systems(InfoScale)*, page 24–33, 2006.
- [65] Jussi Kangasharju, Keith W Ross, and David A Turner. Optimizing file availability in peer-to-peer content distribution[C]. In *Proceedings of the 26th IEEE International Conference on Computer Communications(INFOCOM)*, pages 1973–1981, 2007.
- [66] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric[C]. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems(IPTPS)*, page 53–65, 2002.
- [67] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility[C]. In *Proceedings of the 8th ACM Symposium on Operating Systems Principles(SOSP)*, page 188–201, 2001.
- [68] B Y Zhao, Ling Huang, J Stribling, S C Rhea, A D Joseph, and J D Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment[J]. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.
- [69] Haiping Huang, Yan Zheng, Hongwei Chen, and Ruchuan Wang. PChord: a distributed hash table for P2P network[J]. *Frontiers of Electrical and Electronic Engineering in China*, 5(1):49–53, 2010.
- [70] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems[C]. In *Proceedings of the 17th ACM Symposium*

- on Theory of Computing(STOC), pages 291–304, 1985.
- [71] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge[J]. *SIAM Journal on Computing*, 20(6):1084–1118, 1991.
- [72] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract)[C]. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing(STOC)*, page 723–732, 1992.
- [73] Silvio Micali. Computationally sound proofs[J]. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.
- [74] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK[C]. In *Advances in Cryptology(CRYPTO)*, pages 97–111, 2006.
- [75] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments[C]. In *Advances in Cryptology(ASIACRYPT)*, pages 321–340, 2010.
- [76] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge[J]. *Journal of the ACM*, 59(3):1–35, 2012.
- [77] Christian Reitwiessner. zkSNARKs in a nutshell. <https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell/>, 2016.
- [78] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin[C]. In *Proceedings of the 35th IEEE Symposium on Security and Privacy(SP)*, pages 459–474, 2014.
- [79] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specication (version 2022.3.0). <https://github.com/zcash/zips/blob/main/protocol/protocol.pdf>, 2022.
- [80] Roman Pertsev, Alexey Semenov and Roman Storm. Tornado cash privacy solution. https://tornado.cash/Tornado.cash_whitepaper_v1.4.pdf, 2019.
- [81] zkSync. Introduction to zksync for developers. <https://docs.zksync.io/dev/>, 2022.
- [82] Dark Forest Team. Announcing dark forest (zkSNARK space warfare game).

- <https://blog.zkga.me/announcing-darkforest>, 2020.
- [83] Yang Li, Guangzong Zhang, Jianming Zhu, and Xiuli Wang. Privacy protection model for blockchain data sharing based on zk-SNARK[C]. In *Proceedings of the 2021 International Conference of Pioneering Computer Scientists, Engineers and Educators(ICPCSEE)*, pages 229–239, 2021.
- [84] ProtoSchool. Tutorials - verifying storage on filecoin. <https://proto.school/verifying-storage-on-filecoin>, 2022.
- [85] Muhammad ElSheikh and Amr M Youssef. Dispute-free scalable open vote network using zk-SNARKs[J]. *arxiv.org*, pages 1–17, 2022.
- [86] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. <https://ia.cr/2018/046>, 2018.
- [87] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more[C]. In *Proceedings of the 39th IEEE Symposium on Security and Privacy(SP)*, pages 315–334, 2018.
- [88] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. <https://ia.cr/2019/953>, 2019.
- [89] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation[C]. In *Proceedings of the 34th IEEE Symposium on Security and Privacy(SP)*, pages 238–252, 2013.
- [90] Jens Groth. On the size of pairing-based non-interactive arguments[C]. In *Proceedings of the 35th Annual international conference on the theory and applications of cryptographic techniques(EUROCRYPT)*, pages 305–326, 2016.
- [91] Thomas Chen, Hui Lu, Teeramet Kunpittaya, and Alan Luo. A review of zk-SNARKs[J]. *arXiv preprint arXiv:2202.06877*, pages 1–34, 2022.
- [92] Cloud Security Alliance (CSA). Cloud computing vulnerability incidents: A statistical overview. <https://cloudsecurityalliance.org/artifacts/cloud-computing-vulnerability-incidents-a-statistical-overview/>, 2013.

- [93] Gmail disaster: Reports of mass email deletions. <https://techcrunch.com/2006/12/28/gmail-disaster-reports-of-mass-email-deletions/>, 2006.
- [94] The AWS Team. Summary of the Amazon EC2 and Amazon RDS service disruption in the US east region. <https://aws.amazon.com/cn/message/65648/>, 2011.
- [95] Cong Wang, Kui Ren, Wenjing Lou, and Jin Li. Toward publicly auditable secure cloud data storage services[J]. *IEEE Network*, 24(4):19–24, 2010.
- [96] Yan Zhu, Hongxin Hu, Gail-Joon Ahn, and Stephen S Yau. Efficient audit service outsourcing for data integrity in clouds[J]. *Journal of Systems and Software*, 85(5):1083–1095, 2012.
- [97] Giuseppe Ateniese, Roberto Di Pietro, Luigi V Mancini, and Gene Tsudik. Scalable and efficient provable data possession[C]. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks(SecureComm)*, pages 1–10, 2008.
- [98] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li. Enabling public auditability and data dynamics for storage security in cloud computing[J]. *IEEE Transactions on Parallel and Distributed Systems*, 22(5):847–859, 2011.
- [99] Kan Yang and Xiaohua Jia. An efficient and secure dynamic auditing protocol for data storage in cloud computing[J]. *IEEE Transactions on Parallel and Distributed Systems*, 24(9):1717–1726, 2013.
- [100] Bo Chen, Reza Curtmola, Giuseppe Ateniese, and Randal Burns. Remote data checking for network coding-based distributed storage systems[C]. In *Proceedings of the 2nd ACM Workshop on Cloud Computing Security Workshop(CCSW)*, page 31–42, 2010.
- [101] Huaqun Wang. Proxy provable data possession in public clouds[J]. *IEEE Transactions on Services Computing*, 6(4):551–559, 2013.
- [102] Yihua Zhang and Marina Blanton. Efficient dynamic provable possession of remote data via balanced update trees[C]. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security(ASIA CCS)*, page 183–194, 2013.

- [103] Christian Hanser and Daniel Slamanig. Efficient simultaneous privately and publicly verifiable robust provable data possession from elliptic curves[C]. In *Proceedings of the 10th International Conference on Security and Cryptography(SECRIPT)*, pages 1–12, 2013.
- [104] Cong Wang, Sherman S M Chow, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for secure cloud storage[J]. *IEEE Transactions on Computers*, 62(2):362–375, 2013.
- [105] Lifei Wei, Haojin Zhu, Zhenfu Cao, Xiaolei Dong, Weiwei Jia, Yunlu Chen, and Athanasios V Vasilakos. Security and privacy for storage and computation in cloud computing[J]. *Information sciences*, 258:371–386, 2014.
- [106] Jiawei Yuan and Shucheng Yu. Proofs of retrievability with public verifiability and constant communication cost in cloud[C]. In *Proceedings of the 8th International Workshop on Security in Cloud Computing(Cloud Computing)*, page 19–26, 2013.
- [107] M T Goodrich, R Tamassia, and A Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing[C]. In *Proceedings of the DARPA Information Survivability Conference and Exposition II.(DISCEX)*, pages 68–82, 2001.
- [108] William Pugh. Skip lists: A probabilistic alternative to balanced trees[J]. *Communications of the ACM*, 33(6):668–676, 1990.
- [109] Ralph C Merkle. Protocols for public key cryptosystems[C]. In *Proceedings of the 1st IEEE Symposium on Security and Privacy(SP)*, pages 122–122, 1980.
- [110] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps[C]. In *Proceedings of the 22nd International conference on the theory and applications of cryptographic techniques(EUROCRYPT)*, pages 416–432, 2003.
- [111] Alptekin Kupcu. *Efficient cryptography for the next generation secure cloud*. Brown University, 2010.
- [112] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing[C]. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*

- ogy(ASIACRYPT), page 514–532, 2001.
- [113] J S Plank and Lihao Xu. Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications[C]. In *Proceedings of the 5th IEEE International Symposium on Network Computing and Applications(NCA)*, pages 173–180, 2006.
- [114] Yevgeniy Dodis, Salil Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification[C]. In *Proceedings of the 6th Theory of Cryptography Conference(TCC)*, pages 109–127, 2009.
- [115] Oded Goldreich. A sample of samplers: A computational perspective on sampling[J]. *Electronic Colloquium on Computational Complexity*, 4(20), 1997.
- [116] Michael T Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation[C]. In *Proceedings of the 23rd Annual ACM-SIAM symposium on Discrete Algorithms(SODA)*, pages 157–167, 2012.
- [117] Y Govinda Ramaiah and G Vijaya Kumari. Complete privacy preserving auditing for data integrity in cloud computing[C]. In *Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications(TrustCom)*, pages 1559–1566, 2013.
- [118] Yong Yu, Man Ho Au, Giuseppe Ateniese, Xinyi Huang, Willy Susilo, Yuanshun Dai, and Geyong Min. Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage[J]. *IEEE Transactions on Information Forensics and Security*, 12(4):767–778, 2017.
- [119] Sanjeet Kumar Nayak and Somanath Tripathy. Privacy preserving provable data possession for cloud based electronic health record system[C]. In *Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA*, pages 860–867, 2016.
- [120] Boyang Wang, Baochun Li, and Hui Li. Knox: Privacy-preserving auditing for shared data with large groups in the cloud[C]. In *Proceedings of the 10th International Conference on Applied Cryptography and Network Security(ACNS)*, pages 507–525, 2012.
- [121] Boyang Wang, Hui Li, and Ming Li. Privacy-preserving public auditing for shared cloud data supporting group dynamics[C]. In *Proceedings of the 2013*

- IEEE International Conference on Communications(ICC)*, pages 1946–1950, 2013.
- [122] Boyang Wang, Baochun Li, and Hui Li. Panda: Public auditing for shared data with efficient user revocation in the cloud[J]. *IEEE Transactions on Services Computing*, 8(1):92–106, 2015.
- [123] Jiawei Yuan and Shucheng Yu. Efficient public integrity checking for cloud data sharing with multi-user modification[C]. In *Proceedings of the 33rd IEEE Conference on Computer Communications(INFOCOM)*, pages 2121–2129, 2014.
- [124] Tao Jiang, Xiaofeng Chen, and Jianfeng Ma. Public integrity auditing for shared dynamic cloud data with group user revocation[J]. *IEEE Transactions on Computers*, 65(8):2363–2373, 2016.
- [125] Hao Yan and Wenming Gui. Efficient identity-based public integrity auditing of shared data in cloud storage with user privacy preserving[J]. *IEEE Access*, 9:45822–45831, 2021.
- [126] Anmin Fu, Shui Yu, Yuqing Zhang, Huaqun Wang, and Chanying Huang. NPP: A new privacy-aware public auditing scheme for cloud data sharing with group users[J]. *IEEE Transactions on Big Data*, 8(1):14–24, 2022.
- [127] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. ProvChain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability[C]. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing(CCGRID)*, pages 468–477, 2017.
- [128] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. Towards blockchain-based auditable storage and sharing of IoT data[C]. In *Proceedings of the 9th Cloud Computing Security Workshop(CCSW)*, page 45–50, 2017.
- [129] Ethereum. <https://ethereum.org/en/>.
- [130] Ethereumbook. <https://github.com/ethereumbook/ethereumbook/blob/develop/07smart-contracts-solidity.asciidoc>.
- [131] Stephanos Matsumoto and Raphael M Reischuk. IKP: Turning a PKI around with decentralized automated incentives[C]. In *Proceedings of the 38th IEEE*

- Symposium on Security and Privacy(SP), pages 410–426, 2017.
- [132] Nesrine Kaaniche and Maryline Laurent. A blockchain-based data usage auditing architecture with enhanced privacy and availability[C]. In *Proceedings of the 16th IEEE International Symposium on Network Computing and Applications(NCA)*, pages 1–5, 2017.
- [133] Christian Cachin et al. Architecture of the hyperledger blockchain fabric[C]. In *Proceedings of the 2016 Workshop on distributed cryptocurrencies and consensus ledgers*, pages 1–4, 2016.
- [134] Jae Kwon. Tendermint: Consensus without mining[J]. *Draft v. 0.6, fall*, 1(11), 2014.
- [135] Gideon Greenspan et al. Multichain private blockchain-white paper. <http://www.multichain.com/download/MultiChain-White-Paper.pdf>, 2015.
- [136] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery[J]. *ACM Transactions on Computer Systems*, 20(4):398–461, 2002.
- [137] Ying Miao, Qiong Huang, Meiyang Xiao, and Hongbo Li. Decentralized and privacy-preserving public auditing for cloud storage based on blockchain[J]. *IEEE Access*, 8:139813–139826, 2020.
- [138] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation[C]. In *Proceedings of the 35th IEEE Symposium on Security and Privacy(SP)*, pages 475–490, 2014.
- [139] Henning Kopp, Christoph Bösch, and Frank Kargl. Koppercoin - A distributed file storage with financial incentives[C]. In *Proceedings of the 12th International Conference on Information Security Practice and Experience(ISPEC)*, pages 79–93, 2016.
- [140] David Vorick and Luke Champine. Sia: Simple decentralized storage. Technical report, Nebulous Inc., 2014. <https://sia.tech/sia.pdf>.
- [141] Sushmita Ruj, Mohammad Shahriar Rahman, Anirban Basu, and Shinsaku Kiyomoto. Blockstore: A secure decentralized storage framework on blockchain[C]. In *Proceedings of the 32nd IEEE International Conference on Advanced Information Networking and Applications(AINA)*, pages 1096–1103, 2018.
- [142] Storj Labs. Storj: A decentralized cloud storage network framework. <https://storj.io/>

- //<https://storj.io/storjv3.pdf>, 2018.
- [143] Danilo Francati, Giuseppe Ateniese, Abdoulaye Faye, Andrea Maria Milazzo, and Giuseppe Giordano. Audita: A blockchain-based auditing framework for off-chain storage[C]. In *Proceedings of the 16th ACM Asia Conference on Computer and Communications Security(ASIA CCS)*, pages 5–10, 2021.
- [144] Haiwen Chen, Huan Zhou, Jiaping Yu, Kui Wu, Fang Liu, Tongqing Zhou, and Zhiping Cai. Trusted audit with untrusted auditors: A decentralized data integrity crowdauditing approach based on blockchain[J]. *International Journal of Intelligent Systems*, 36(7):6213–6239, 2021.
- [145] Hao Yan, Yanan Liu, Shuo Qiu, Shengzhou Hu, Weijian Zhang, and Jinyue Xia. Towards public integrity audition for cloud-IoT data based on blockchain[J]. *Computer Systems Science and Engineering*, 41(3):1129–1142, 2022.
- [146] ConsenSys. Blockchain use cases and applications by industry. <https://consensys.net/blockchain-use-cases/>, 2022.
- [147] Hyperledger. Case studies: browse various use cases powered by Hyperledger technologies. <https://www.hyperledger.org/learn/case-studies>.
- [148] Blockchain.com. Bitcoin blockchain size. <https://www.blockchain.com/charts/blocks-size>, 2021.
- [149] Abdurrashid Ibrahim Sanka and Ray C C Cheung. A systematic review of blockchain scalability: Issues, solutions, analysis and future research[J]. *Journal of Network and Computer Applications*, 195:103232, 2021.
- [150] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work[C]. In *Proceedings of the 24th International Conference on Financial Cryptography and Data Security(FC)*, pages 505–522, 2020.
- [151] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. FlyClient: Super-light clients for cryptocurrencies[C]. In *Proceedings of the 41st IEEE Symposium on Security and Privacy(SP)*, pages 928–946, 2020.
- [152] Bitcoin core version 0.11.0. <https://bitcoin.org/en/release/v0.11.0#block-file-pruning>, 2015.
- [153] Bitcoin core version 0.14.0. <https://bitcoin.org/en/release/v0.14.0#manual-pruning>, 2017.

- [154] Roman Matzutt, Benedikt Kalde, Jan Pennekamp, Arthur Drichel, Martin Henze, and Klaus Wehrle. How to securely prune bitcoin's blockchain[C]. In *Proceedings of the 19th IFIP Networking Conference(Networking)*, pages 298–306, 2020.
- [155] Xiaohai Dai, Jiang Xiao, Wenhui Yang, Chaofan Wang, and Hai Jin. Jidar: A jigsaw-like data reduction approach without trust assumptions for bitcoin system[C]. In *Proceedings of the 39th IEEE International Conference on Distributed Computing Systems(ICDCS)*, pages 1317–1326, 2019.
- [156] Emanuel Palm, Olov Schelén, and Ulf Bodin. Selective blockchain transaction pruning and state derivability[C]. In *Proceedings of the 1st Crypto Valley Conference on Blockchain Technology(CVCBT)*, pages 31–40, 2018.
- [157] Zihuan Xu, Siyuan Han, and Lei Chen. CUB, A consensus unit-based storage scheme for blockchain system[C]. In *Proceedings of the 34th IEEE International Conference on Data Engineering(ICDE)*, pages 173–184, 2018.
- [158] Dapeng Li, Jinsen Dai, Rui Jiang, Xiaoming Wang, and Youyun Xu. GAPG: A heuristic greedy algorithm for grouping storage scheme in blockchain[C]. In *Proceedings of the 2020 IEEE/CIC International Conference on Communications in China(ICCC Workshops)*, pages 91–95, 2020.
- [159] Jianyu Chen, Keke Gai, Peng Jiang, and Liehuang Zhu. Heuristic-based blockchain assignment: An empirical study[C]. In *Proceedings of the 19th IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking(ISPA)*, pages 916–923, 2021.
- [160] JianHong Zhou, Gang Feng, Yao Sun, and Hongxin Luo. Intelligent block assignment for blockchain based wireless IoT systems[C]. In *Proceedings of the 2020 IEEE International Conference on Communications(ICC)*, pages 1–6, 2020.
- [161] Bin Qu, Li-E Wang, Peng Liu, Zhenkui Shi, and Xianxian Li. GCBLOCK: A grouping and coding based storage scheme for blockchain system[J]. *IEEE Access*, 8:48325–48336, 2020.
- [162] Xiaodong Qi, Zhao Zhang, Cheqing Jin, and Aoying Zhou. BFT-Store: Storage partition for permissioned blockchain via erasure coding[C]. In *Proceedings*

- of the 36th IEEE International Conference on Data Engineering(ICDE)*, pages 1926–1929, 2020.
- [163] Dayu Jia, Junchang Xin, Zhiqiong Wang, Wei Guo, and Guoren Wang. ElasticChain: Support very large blockchain by reducing data redundancy[C]. In *Proceedings of the 2nd International Joint Conference on Web and Big Data(APWeb/WAIM)*, pages 440–454, 2018.
- [164] Dayu Jia, Junchang Xin, Zhiqiong Wang, Han Lei, and Guoren Wang. SE-Chain: A scalable storage and efficient retrieval model for blockchain[J]. *Journal of Computer Science and Technology*, 36(3):693–706, 2021.
- [165] Alessandro Birolini. *Reliability engineering: theory and practice*. Springer Science & Business Media, 1999.
- [166] Felix C Gärtner. Byzantine failures and security: Arbitrary is not (always) random[J]. *INFORMATIK 2003-Mit Sicherheit Informatik, Schwerpunkt "Sicherheit-Schutz und Zuverlässigkeit"*, pages 127–138, 2003.
- [167] Jon G Elerath. A simple equation for estimating reliability of an n+1 redundant array of independent disks (RAID)[C]. In *Proceedings of the 39th IEEE/IFIP International Conference on Dependable Systems & Networks(DSN)*, pages 484–493, 2009.
- [168] Rui Zhang, Chuang Lin, Kun Meng, and Lin Zhu. A modeling reliability analysis technique for cloud storage system[C]. In *Proceedings of the 15th IEEE International Conference on Communication Technology(ICCT)*, pages 32–36, 2013.
- [169] Ilias Iliadis, Dmitry Sotnikov, Paula Ta-Shma, and Vinodh Venkatesan. Reliability of geo-replicated cloud storage systems[C]. In *Proceedings of the 19th IEEE Pacific Rim International Symposium on Dependable Computing(PRDC)*, pages 169–179, 2014.
- [170] Qisi Liu and Liudong Xing. Reliability modeling of cloud-RAID-6 storage system[J]. *International Journal of Future Computer and Communication*, 4(6):415–420, 2015.
- [171] Rekha Nachiappan, Bahman Javadi, Rodrigo N Calheiros, and Kenan M Matawie. Cloud storage reliability for big data applications: A state of the art

- survey[J]. *Journal of Network and Computer Applications*, 97:35–47, 2017.
- [172] Patricia Takako Endo, Guto Leoni Santos, Daniel Rosendo, Demis Moacir Gomes, André Moreira, Judith Kelner, Djamel Sadok, Glauco Estácio Gonçalves, and Mozghan Mahloo. Minimizing and managing cloud failures[J]. *Computer*, 50(11):86–90, 2017.
- [173] Lavanya Mandava and Liudong Xing. Balancing reliability and cost in cloud-RAID systems with fault-level coverage[J]. *International Journal of Mathematical, Engineering and Management Sciences*, 4(5):1068–1080, 2019.
- [174] Thomas JE Schwarz, Qin Xin, Ethan L Miller, Darrell DE Long, Andy Hospodor, and Spencer Ng. Disk scrubbing in large archival storage systems[C]. In *Proceedings of the 12th IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems(MASCOTS)*, pages 409–418, 2004.
- [175] Jing Li, Peng Li, Rebecca J Stones, Gang Wang, Zhongwei Li, and Xiaoguang Liu. Reliability equations for cloud storage systems with proactive fault tolerance[J]. *IEEE Transactions on Dependable and Secure Computing*, 17(4):782–794, 2018.
- [176] Yan Gao, Dirk Meister, and André Brinkmann. Reliability analysis of declustered-parity RAID 6 with disk scrubbing and considering irrecoverable read errors[C]. In *Proceedings of the 5th IEEE International Conference on Networking, Architecture, and Storage(NAS)*, pages 126–134, 2010.
- [177] Purushottam Sigdel, Xu Yuan, and Nian-Feng Tzeng. Realizing best checkpointing control in computing systems[J]. *IEEE Transactions on Parallel and Distributed Systems*, 32(2):315–329, 2021.
- [178] Bianca Schroeder and Garth A Gibson. Understanding disk failure rates: What does an MTTF of 1,000,000 hours mean to you?[J]. *ACM Transactions on Storage*, 3(3):1–16, 2007.
- [179] Bianca Schroeder and Garth A Gibson. A large-scale study of failures in high-performance computing systems[J]. *IEEE Transactions on Dependable and Secure Computing*, 7(4):337–350, 2010.
- [180] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. Fail-

- ures in large scale systems: Long-term measurement, analysis, and implications[C]. In *Proceedings of the 2017 International Conference for High Performance Computing, Networking, Storage and Analysis(SC)*, 2017.
- [181] H Zhang and W Xue. Reliability analysis of cluster raid5 storage system. *Journal of Computer Research & Development*[J], 47(4):727–735, 2010.
- [182] Te Shun Chou. Security threats on cloud computing vulnerabilities[J]. *International Journal of Computer Science & Information Technology*, 5(3):79, 2013.
- [183] S Meena, Esther Daniel, and N A Vasanthi. Survey on various data integrity attacks in cloud environment and the solutions[C]. In *Proceedings of the 2013 International Conference on Circuits, Power and Computing Technologies(ICCPCT)*, pages 1076–1081, 2013.
- [184] Nancy R Mead, Robert J Ellison, Richard C Linger, Thomas Longstaff, and John McHugh. Survivable network analysis method. Technical report, Carnegie-Mellon University Pittsburgh PA Software Engineering Institute, 2000.
- [185] Casey Fung, Yi Liang Chen, Xinyu Wang, J Lee, R Tarquini, M Anderson, and R Linger. Survivability analysis of distributed systems using attack tree methodology[C]. In *Proceedings of the 2005 IEEE Military Communications Conference(MILCOM)*, pages 583–589, 2005.
- [186] Qisi Liu and Liudong Xing. Survivability and vulnerability analysis of cloud RAID systems under disk faults and attacks[J]. *International Journal of Mathematical, Engineering and Management Sciences*, 6(1):15–29, 2021.
- [187] Amin Jula, Elankovan Sundararajan, and Zalinda Othman. Cloud computing service composition: A systematic literature review[J]. *Expert systems with applications*, 41(8):3809–3824, 2014.
- [188] Mehdi Sookhak, Abdullah Gani, Hamid Talebian, Adnan Akhunzada, Samee U Khan, Rajkumar Buyya, and Albert Y Zomaya. Remote data auditing in cloud computing environments: A survey, taxonomy, and open issues[J]. *ACM Computing Surveys*, 47(4):1–34, 2015.
- [189] Faheem Zafar, Abid Khan, Saif Ur Rehman Malik, Mansoor Ahmed, Adeel Anjum, Majid Iqbal Khan, Nadeem Javed, Masoom Alam, and Fuzel Jamil. A survey of cloud computing data integrity schemes: Design challenges, taxonomy

- and future trends. *Computers & Security*[J], 65:29–49, 2017.
- [190] Mingqiang Li and Patrick P C Lee. STAIR codes: A general family of erasure codes for tolerating device and sector failures[J]. *ACM Transactions on Storage*, 10(4):1–30, 2014.
- [191] Yilun Wu, Xinye Lin, Xicheng Lu, Jinshu Su, and Peixin Chen. A secure light-weight public auditing scheme in cloud computing with potentially malicious third party auditor[J]. *IEICE Transactions on Information and Systems*, E99.D(10):2638–2642, 2016.
- [192] Mazhar Ali, Saif U R Malik, and Samee U Khan. DaSCE: Data security for cloud environment with semi-trusted third party[J]. *IEEE Transactions on Cloud Computing*, 5(4):642–655, 2017.
- [193] Kun Qian and Hui Huang. A new identity-based public auditing against malicious auditor in the cloud[J]. *International Journal of Embedded Systems*, 11(4):452–460, 2019.
- [194] Xiaojun Zhang, Jie Zhao, Chunxiang Xu, Hongwei Li, Huaxiong Wang, and Yuan Zhang. CIPPPA: Conditional identity privacy-preserving public auditing for cloud-based WBANs against malicious auditors[J]. *IEEE Transactions on Cloud Computing*, 9(4):1362–1375, 2021.
- [195] Hovav Shacham and Brent Waters. Compact proofs of retrievability[C]. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security(ASIACRYPT)*, page 90–107, 2008.
- [196] Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. MR-PDP: Multiple-replica provable data possession[C]. In *Proceedings of the 28th International Conference on Distributed Computing Systems(ICDCS)*, pages 411–420, 2008.
- [197] Frederik Armknecht, Jens Matthias Bohli, Ghassan O Karame, Zongren Liu, and Christian A Reuter. Outsourced proofs of retrievability[C]. In *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security(CCS)*, page 831–843, 2014.
- [198] Yuan Zhang, Chunxiang Xu, Shui Yu, Hongwei Li, and Xiaojun Zhang. SCLPV: Secure certificateless public verification for cloud-based cyber-physical-social

- systems against malicious auditors[J]. *IEEE Transactions on Computational Social Systems*, 2(4):159–170, 2015.
- [199] Jingting Xue, Chunxiang Xu, Jining Zhao, and Jianfeng Ma. Identity-based public auditing for cloud storage systems against malicious auditors via blockchain[J]. *Science China Information Sciences*, 62(3):32104, 2019.
- [200] Binanda Sengupta, Samiran Bag, Sushmita Ruj, and Kouichi Sakurai. Retricoin: Bitcoin based on compact proofs of retrievability[C]. In *Proceedings of the 17th International Conference on Distributed Computing and Networking(ICDCN)*, pages 1–10, 2016.
- [201] Silvio Micali, Salil Vadhan, and Michael Rabin. Verifiable random functions[C]. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science(FOCS)*, pages 120–130, 1999.
- [202] Mohamed Seifelnasr, Hisham S Galal, and Amr M Youssef. Scalable open-vote network on ethereum[C]. In *Proceedings of the 24th International Conference on Financial Cryptography and Data Security(FC)*, pages 436–450, 2020.
- [203] Circom 2 documentation. <https://docs.circom.io>, 2021.
- [204] Ralph C Merkle. A digital signature based on a conventional encryption function[C]. In *Proceedings of the 1987 Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology(CRYPTO)*, page 369–378, 1987.
- [205] Ben Lynn. The pairing-based cryptography library(PBC Library). <https://crypto.stanford.edu/pbc/>, 2015.
- [206] Ethereum. Solidity v0.8.11. <https://docs.soliditylang.org/en/v0.8.11/>.
- [207] Truffle. Ganache. <https://trufflesuite.com/ganache/>.
- [208] Ethereum. Nethereum. <https://github.com/Nethereum/Nethereum>.
- [209] snarkjs. <https://github.com/iden3/snarkjs>, 2021.
- [210] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, and Elaine Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, Booz-allen and hamilton inc mclean va, 2001. <https://doi.org/10.6028/NIST.SP.800-22r1a>.

- [211] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies[C]. In *Proceedings of the 26th Symposium on Operating Systems Principles(SOSP)*, page 51–68, 2017.
- [212] Nicolas Gailly, Mary Maller, and Anca Nitulescu. SnarkPack: Practical SNARK aggregation. <https://ia.cr/2021/529>, 2021.
- [213] Hakim Weatherspoon and John Kubiatowicz. Erasure coding vs. replication: A quantitative comparison[C]. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems(IPTPS)*, page 328–338, 2002.
- [214] Bitcoin cryptocurrency (Database). <https://console.cloud.google.com/marketplace/product/bitcoin/crypto-bitcoin?q=search&referrer=search&project=hexo-calendar>, 2020.
- [215] The go-ethereum Authors. Installing Geth. <https://geth.ethereum.org/docs/install-and-build/installing-geth>, 2022.
- [216] Hassan Takabi, James B D Joshi, and Gail-Joon Ahn. Security and privacy challenges in cloud computing environments[J]. *IEEE Security and Privacy*, 8(6):24–31, 2010.
- [217] Kevin D Bowers, Ari Juels, and Alina Oprea. HAIL: A high-availability and integrity layer for cloud storage[C]. In *Proceedings of the 16th ACM Conference on Computer and Communications Security(CCS)*, page 187–198, 2009.
- [218] AWS Services. AWS partner story: Carecloud. <https://aws.amazon.com/cn/partners/success/carecloud/>, 2019.
- [219] L Razavi. The icloud leak: Weak security isn’t only a problem for Apple’s backup service. <https://www.newstatesman.com/sci-tech/>, 2014.
- [220] Dusit Niyato, Athanasios V Vasilakos, and Zhu Kun. Resource and revenue sharing with coalition formation of cloud providers: Game theoretic approach[C]. In *Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing(CCGRID)*, pages 215–224, 2011.
- [221] Mark W Storer, Kevin M Greenan, Ethan L Miller, and Kaladhar Voruganti. POTSHARDS—A secure, recoverable, long-term archival storage system[J]. *ACM Transactions on Storage*, 5(2):24–31, 2009.

- [222] Daniel Ford, François Labelle, Florentina I Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. Availability in globally distributed storage systems[C]. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation(OSDI)*, page 61–74, 2010.
- [223] Check Point Software Technologies Ltd. Security report 2020. <https://www.bristol.de/wp-content/uploads/2020/03/2020-security-report.pdf>, 2020.
- [224] Gregory Levitin, Liudong Xing, and Yuanshun Dai. Co-residence based data vulnerability vs. security in cloud computing system with random server assignment[J]. *European Journal of Operational Research*, 267(2):676–686, 2018.
- [225] Qisi Liu, Liudong Xing, and Chencheng Zhou. Probabilistic modeling and analysis of sequential cyber-attacks[J]. *Engineering Reports*, 1(4):1–19, 2019.
- [226] Gregory F Lawler. *Introduction to stochastic processes*. Chapman and Hall/CRC, 2018.
- [227] Waloddi Weibull. A statistical distribution function of wide applicability[J]. *Journal of applied mechanics*, page 1–6, 1951.
- [228] Ajay Dholakia, Evangelos Eleftheriou, Xiao Yu Hu, Ilias Iliadis, Jai Menon, and K K Rao. A new intra-disk redundancy scheme for high-reliability RAID storage systems in the presence of unrecoverable errors[J]. *ACM Transactions on Storage*, 4(1), 2008.
- [229] Jehan François Pâris, Ahmed Amer, Darrell D E Long, and Thomas J E Schwarz. Evaluating the impact of irrecoverable read errors on disk array reliability[C]. In *Proceedings of the 15th IEEE Pacific Rim International Symposium on Dependable Computing(PRDC)*, pages 379–384, 2009.
- [230] Edward PC Kao. *An introduction to stochastic processes*. Courier Dover Publications, 2019.

参考文献

个人简历

博士在读期间发表的论文以及取得的科研成果

1. 第一作者, SRDS 2020, 论文一篇, 中国计算机协会CCF推荐的 **B** 类会议
论文提出一种基于区块链的分布式审计模型以及两个分布式审计协议, 与
学位论文第三章内容对应。
2. 第一作者, 《电子学报》, 论文一篇, 中国计算机协会CCF推荐的 **A** 类中文
期刊
论文提出一种适用于联盟链的存储优化方案, 与学位论文第四章内容对
应。
3. 第一作者, IEEE Access, 论文一篇, 中科院分区SCI 3 区期刊
论文提出一种能够执行安全的私有审计的多云存储系统, 是学位论文第五
章的研究基础。