分类号：TP393　　单位代码：10335

密　级：公开　　学　　号：

浙江大学

硕士学位论文

（专业学位）

中文论文题目：面向物联网时序数据的高效可信去中心化存储技术研究

英文论文题目：Research on Efficient and Reliable Decentralized Storage Technology for IoT Time Series Data

申请人姓名：

校内导师（组）：

行业导师：

专业学位类别、领域：电子信息 计算机技术

研究方向：区块链数据存储

培养类型：全日制

所在学院：计算机科学与技术学院

论文提交日期 2024-12-24

面向物联网时序数据的

高效可信去中心化存储技术研究

论文作者签名： _____

指导教师签名： _____

论文评阅人 1：_____

评阅人 2：_____

评阅人 3：_____

评阅人 4：_____

评阅人 5：_____

答辩委员会主席：_____

委员 1：_____

委员 2：_____

委员 3：_____

委员 4：_____

委员 5：_____

答辩日期 _____

Research on Efficient and Reliable Decentralized Storage Technology for IoT Time Series Data

Author's signature: _____

Supervisor's signature: _____

External reviewers: _____

_____

_____

_____

_____

Examining Committee Chairperson:

_____

Examining Committee Members:

_____

_____

_____

_____

_____

Date of oral defence: _____

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 **浙江大学** 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名: 签字日期： 年 月 日

学位论文版权使用授权书

本学位论文作者完全了解 **浙江大学** 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 **浙江大学** 可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名: 导师签名:

签字日期： 年 月 日 签字日期： 年 月 日

摘要

与传统集中式存储相比，基于区块链的去中心化存储系统提供了更高的安全性、透明度和更低的成本，使其成为点对点协作的理想选择。然而，较低的事务处理速度和较高的计算要求限制了其对于物联网时序数据等高密度数据场景中的应用。

为了解决这个问题，我们提出了 TimeChain，一种面向物联网时序数据的高效链下区块链存储系统。TimeChain 对离散时序数据进行批处理，并仅将每个批次的哈希值存储在链上，而完整的数据则保留在链下。这种方法显著降低了区块链上的存储开销，减少了 37.4 倍的存储延迟。

为了进一步减少范围查询中的传输延迟，TimeChain 引入了自适应打包机制。通过将数据和历史协同查询分别表示为图的顶点和边权重，我们将批处理问题转化为图划分问题，显著提高了查询效率。为了减少节点选择过程中的时延，TimeChain 将节点选择和共识过程结合，提出了基于共识协议的节点选择机制。为了最小化数据完整性验证中的传输量，TimeChain 采用了一种基于局部敏感哈希（LSH）的数据完整性校验机制。该机制通过仅传输非冗余部分来最大限度地减少完整性检查所需的数据量。

我们基于开源组件实现了 TimeChain，并对其性能进行了评估。实验结果显示，与现有的基于区块链的存储系统相比，TimeChain 平均减少了 64.6% 的查询延迟和 35.3% 的存储延迟。这表明 TimeChain 在提高物联网时序数据处理效率方面具有显著优势。

关键词：物联网、区块链、数据存储、时序数据

Abstract

Blockchain-based distributed storage systems offer enhanced security, transparency, and lower costs compared to traditional centralized storage, making them ideal for peer-to-peer collaboration. However, their lower transaction processing speed and higher computational requirements demands restrict their deployment in high-density data scenarios such as the Internet of Things (IoT).

To address this, we propose TimeChain, an efficient off-chain blockchain storage system for IoT time series data. TimeChain batches discrete time series data, storing only the hash value of each batch on-chain while keeping the complete data off-chain. This significantly reduces storage overhead on the blockchain and storage latency by 37.4 times.

In order to reduce the additional transmission latency in range queries, TimeChain employs an adaptive packaging mechanism. We convert the batching problem to a graph partitioning problem by representing data and historical co-query as graph vertices and edge weights respectively. To reduce the size of the transmission size in data integrity verification, a Locality-Sensitive Hashing (LSH)-based data integrity verification mechanism, which minimizes the data required for integrity checks by transmitting only non-redundant parts. TimeChain also integrates a node selection mechanism based on consensus protocol, which reduces the overhead by combining node selection and consensus processes.

We implement TimeChain based on top of production-ready open-source components such as Hyperledger Fabric and IPFS, and evaluate the performance of TimeChain. The result shows that compared to existing blockchain-based storage systems, TimeChain reduces 64.6% query latency and 35.3% storage latency on average.

Key words: IoT, Blockchain, Data Storage, Time Series Data

缩略词表

英文缩写	英文全称	中文全称
IoT	Internet of Things	物联网
ZB	ZettaBytes	泽字节
IPFS	InterPlanetary File System	星际文件系统
LSH	Locality Sensitive Hashing	局部敏感哈希
CRUSH	Controlled Replication Under Scalable Hashing	可扩展哈希下的受控副本
PG	Placement Group	放置组
MST	Merkle Semantic Trie	默克尔语义树
MPT	Merkle Patricia Tree	默克尔帕特里夏树
AMT	Authenticated Multipoint Evaluation Tree	认证多点评估树
GMM	Gaussian Mixture Models	高斯混合模型
EM	Expectation-Maximization	期望最大化
HMM	Hidden Markov Model	隐马尔可夫模型
PoW	Proof of Work	工作量证明
PBFT	Practical Byzantine Fault Tolerance	实用拜占庭容错
POR	Proofs of Retrievability	可检索性
UWG	Undirected Weighted Graph	无向加权图
PC	Personal Computer	个人电脑

目录

图目录

表目录

1 绪论

1.1 研究背景及意义

近年来，物联网（Internet of Things，IoT）技术的快速发展正在逐渐改变人们的生活和工作方式，其应用范围覆盖了农业、制造业、医疗保健、零售业和公用事业等多个领域。随着物联网技术的不断进步和应用领域的拓展，可以预见，未来将有越来越多的设备加入互联网。如图 1.1所示，根据 Gartner 的分析[1]，2027 年将有三千亿物联网设备接入网络。这些设备产生的离散数据将达到产生泽字节（ZettaBytes, ZB）[2]，这一数字预示着未来数据的爆炸式增长，形成庞大的数据海洋。

面对如此大规模的数据增长，传统的集中式服务器管理方式，如 AWS IoT、阿里云等，可能会遇到诸多挑战。这些挑战中最为突出的是单点故障问题[3]，即所有数据都集中在一个或几个服务器上，一旦这些中心节点发生故障，整个系统的稳定性和可用性将受到严重影响，甚至可能导致数据的不可用或丢失。此外，集中式存储方案在数据传输效率、扩展性和维护成本方面也存在一定的局限性。这些问题限制了集中式存储系统在处理大规模、高动态物联网数据时的灵活性和效率。

尽管分布式数据库通过在多个节点间分散存储数据来避免单点故障问题，但它们在数据安全性和不可篡改性方面仍然存在弱点。在需要高度透明的应用场景下，如金融交易、医疗记录等，分布式数据库容易受到数据篡改攻击，这直接威胁到数据的完整性和安全性。因此，开发一种既能够提供高可靠性又能够确保数据安全性的存储解决方案，对于物联网数据的管理和保护至关重要。

区块链技术以其可追溯性和不变性为特征，为解决集中式存储的单点故障问题和分布式数据库的恶意篡改问题提供了新的途径。区块链通过将数据存储在去中心化的账本中，并利用共识协议来解决各个节点之间可能发生的冲突，从而增强了数据的安全性和透明度，降低了成本，并支持节点间的平等协作。这种去中心化的数据存储方式，使得任何一个节点的故障都不会影响整个系统的运行，从而有效避免了单点故障的风险。同时，区块链的不可篡改性确保了数据一旦被记录，就无法被更改或删除，这对于需要高度数据完整性的应用场景尤为重要。

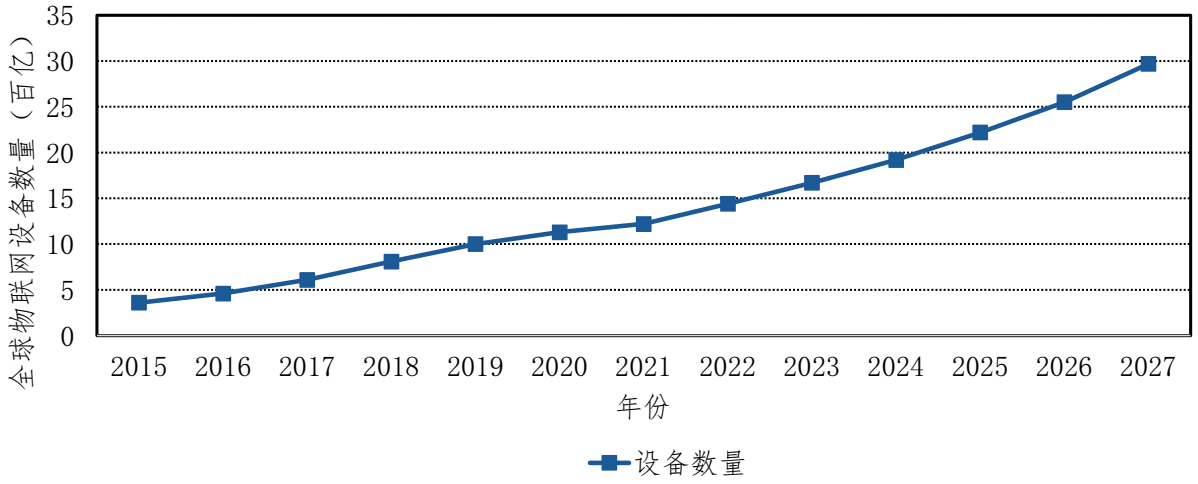


图 1.1 2015-2027 年全球物联网设备预测数量

然而，基于区块链的存储系统也面临着挑战。由于其较低的交易处理速度和较高的计算要求[4]，区块链技术目前主要应用于价值巨大但数据密度较低的领域，如金融服务、供应链管理等。但是，对于面向生成频率快、数据体量大的物联网时序数据的应用，如智能城市、工业自动化等，区块链技术的性能瓶颈限制了其广泛应用。因此，如何优化区块链技术，提高其处理速度和效率，使其能够更好地服务于物联网数据的存储和管理，是当前研究和开发的重要方向。

目前，有许多研究人员致力于提高基于区块链的存储系统的性能。根据数据的存储位置，这些工作可以分为两类，即链上存储和链下存储。

对于链上存储，数据作为存储在区块链上的交易记录的一部分。用户通过区块链中的索引，即默克尔树来获取这些数据。现有工作通过提供用户友好的查询语言[5-7]提高了系统可用性，并通过改进索引方案[8-9]、区块链存储分片[10-12]等方式提高系统吞吐量。然而，在链上存储物联网时序数据的开销非常高。对于大量且快速生成的物联网时序数据，将其连续存储在区块链上需要实现共识和账本复制的过程，这可能会导致巨大的存储压力和通信开销。因此，物联网数据存储的一种更实用的方法是使用链下存储方案。

对于链下存储，数据存储在区块链之外的存储节点，区块链只存储必要的元数据或对数据的引用，如哈希或加密指针。链下存储提供了比链上解决方案更大的可扩展性和更低的成本，因此业界和学术界对链下存储非常有兴趣，例如 Storj[13]、BigchainDB[14]、Sia[15]等。然而，现有的工作大多是为存储大文件而设计的。相比于较大的物联网视频流数据，物联网时序数据的产生频率快、数据体量大，在区块链中存储每个离散数据项

的哈希值将产生难以接受的开销。此外，物联网应用场景通常需要存储系统支持高效查询（例如聚合查询），而现有的基于文件的存储系统不支持这一点。

本文基于链下存储的方案，提出了一种面向物联网的区块链存储系统，并基于该系统进行了一系列广泛的测量。针对测量中发现的低效查询性能问题，本文提出了TimeChain，一个基于区块链的物联网时序数据高效可信存储系统。TimeChain 分别针对测量中发现的三个性能根因提出了自适应聚合机制、基于共识的存储节点选择机制和基于局部敏感哈希树的数据完整性验证机制。实验结果表明，TimeChain 相比现有的基于区块链的存储系统有显著的性能提升，对于实现物联网数据的可信高效存储具有一定的理论和实践价值。

1.2 论文主要工作

为了对海量物联网数据进行可信、高效的存储，受链下存储方案的启发，本文提出了一种基于区块链的物联网时序数据基本存储方案。该方案将数据聚合成多个批处理单元，仅将每批数据的哈希值存储在链上，并将完整的原始数据保存在链外。这种批处理存储方法大大减少了数据开销。

本文基于 Hyperledger Fabric[16]和星际文件系统（InterPlanetary File System，IPFS）[17]等开源组件实现了该系统，并基于 YCSB 数据集对该方案的性能进行了测量。结果表明，与每个数据单独存储相比，将数据批量存储的存储延迟平均减少了 37.4 倍。这种存储性能的显著提升使得区块链支持物联网数据存储成为可能。

然而，不可否认的是，这样的批处理方案会影响查询性能。具体来说，批处理存储的平均延迟达到了 165.4ms，这一延迟水平对于许多对实时性要求高的物联网应用来说是不可接受的。以自动驾驶为例，其对延迟的要求是低于 50ms，而在地震监测领域，理想的延迟更是要低于 100ms。为了深入理解这一性能瓶颈，本研究详细分析了影响查询性能的三个关键原因：

1. **范围查询跨越多个批处理单元。**在处理时间序列数据时，查询操作如范围查询、聚合查询和过滤查询等，通常涉及多个数据点。如果数据的批处理方式不够合理，单个查询可能需要访问多个批处理单元。实验结果显示，超过 84.25% 的查询需要跨越至少 10 个批处理单元。当这些单元分散在不同的存储节点上时，就会增加额外

的查询和传输延迟，影响整体的查询响应速度。

2. **存储节点选择不当。**在本研究的测量中，传输延迟在总查询延迟中占据了显著的比例。不同地理位置的存储节点之间存在传输距离的差异，这直接影响了数据传输的延迟，选择地理位置较远的存储节点会导致传输延迟显著增加。此外，如果存储网络中存在恶意节点，即使所选节点的传输延迟较小，数据的完整性也可能受到威胁，从而影响数据存储服务的完整性和可靠性。

3. **完整性证明数据量大。**为了使存储提供商能够快速响应数据拥有者的数据查询请求，并提供数据完整性的证明，这些证明需要与存储提供商一同存储。实验表明，由于原始数据本身较小，对小粒度数据哈希产生的完整性证明相对较大，在总传输数据量中占据了相当大的比例，达到了接收数据的48.8%。在网络繁忙时段，大量的完整性证明数据传输会加剧网络拥塞，增加传输延迟，从而影响查询性能。

针对测量中发现性能瓶颈的三个根本原因，本文提出了TimeChain，一种高效的物联网时序数据区块链存储技术，以解决这些问题。本文的贡献主要如下：

- **在数据聚合阶段，本文提出了一种面向链下存储的自适应聚合机制。**为了减少范围查询过程中跨越的存储节点数量，本文通过构建自适应无向加权图，将批处理问题转化为图划分问题。对于该图划分问题，由于物联网数据的查询通常不遵循特定的规律，数据分布可能不会形成传统的圆形。因此传统方法如K-means[18]需要固定图的形状和数量，GMM[19]则计算复杂度较高，因此不适合物联网数据的聚合。由于谱聚类算法适合用于不规则形状的图划分问题，因此本文使用谱聚类算法进行子图划分，以减少聚合查询期间的节点访问次数。

- **在存储节点选择阶段，本文提出了一种基于共识的存储节点选择机制。**对于节点选择过程，本文主要考虑选择过程的正确性和安全性。选择过程的正确性指的是选择的节点传输时延较短、提供存储服务质量较高，选择过程的安全性则指的是决策过程本身不会受到单点故障或恶意节点的影响。对于选择过程的正确性，受已有方案的启发，TimeChain同时考虑存储节点信誉、剩余存储空间和传输距离。对于选择过程的安全性，目前已有的节点选择机制都是中心化地进行节点选择，这可能会产生单点故障和数据篡改的风险。考虑到共识过程中的两次信息广播过程，

为了在去中心化的环境中高效地选择存储节点，本文将共识过程与节点选择相结合，借助共识的两次广播过程完成节点选择所需的信息交换，以此减少数据传输和一致性开销。

- **在数据验证阶段，本文提出了一种基于局部敏感哈希树的数据完整性验证机制。**目前区块链存储系统中常用的验证数据结构，如默克尔树，其构建过程中会产生与原始数据规模相同的哈希值，这会导致传输开销过大。针对该问题，本文提出了一种基于局部敏感哈希（Locality Sensitive Hashing，LSH）树的数据完整性验证机制。该机制利用物联网场景下相邻时间序列数据点之间的相似性，通过仅传输 LSH 证明的非冗余部分显著减少了完整性验证所需的数据。

本文基于开源组件实现了 TimeChain，并基于此评估 TimeChain 的性能。结果表明，相比于现有的基于区块链的存储系统，它平均减少了 64.6% 的查询延迟和 35.3% 的存储延迟。这一显著的性能提升证明了 TimeChain 在提高物联网时序数据处理效率方面的优势。此外，TimeChain 在支持最大存储设备数量方面展现了良好的可扩展性。与 SEBDB 和 FileDES 相比，TimeChain 分别增加了 1.63 倍和 3.55 倍的设备支持能力。这一结果表明，TimeChain 能够适应不断增长的物联网设备数量，满足未来物联网应用的发展需求。通过消融实验，本文进一步验证了 TimeChain 中自适应聚合机制、基于共识的节点选择机制和基于 LSH 树的验证机制对于提升系统性能的重要性。

本文的相关研究工作已被以第一作者身份投稿至 CCF A 类国际会议 The ACM Web Conference 2025（ACM WWW 2025）上，目前在审稿阶段。此外，本文的相关研究工作也以第二发明人（学生第一发明人）身份申请发明专利一项，目前处于实质审查阶段。

1.3 全文结构

围绕上述研究内容和贡献，本文将内容分为如 图 1.2所示的八个章节展开，各章节组织如下：

第一章介绍了本文的研究背景和意义，概述了本文的主要工作和贡献，以及全文的结构安排。

第二章详细回顾了中心化存储系统、分布式存储系统、区块链存储系统等相关技术领域，分析了区块链现有存储技术在本场景下的的局限性。本文探讨了区块链链上存储

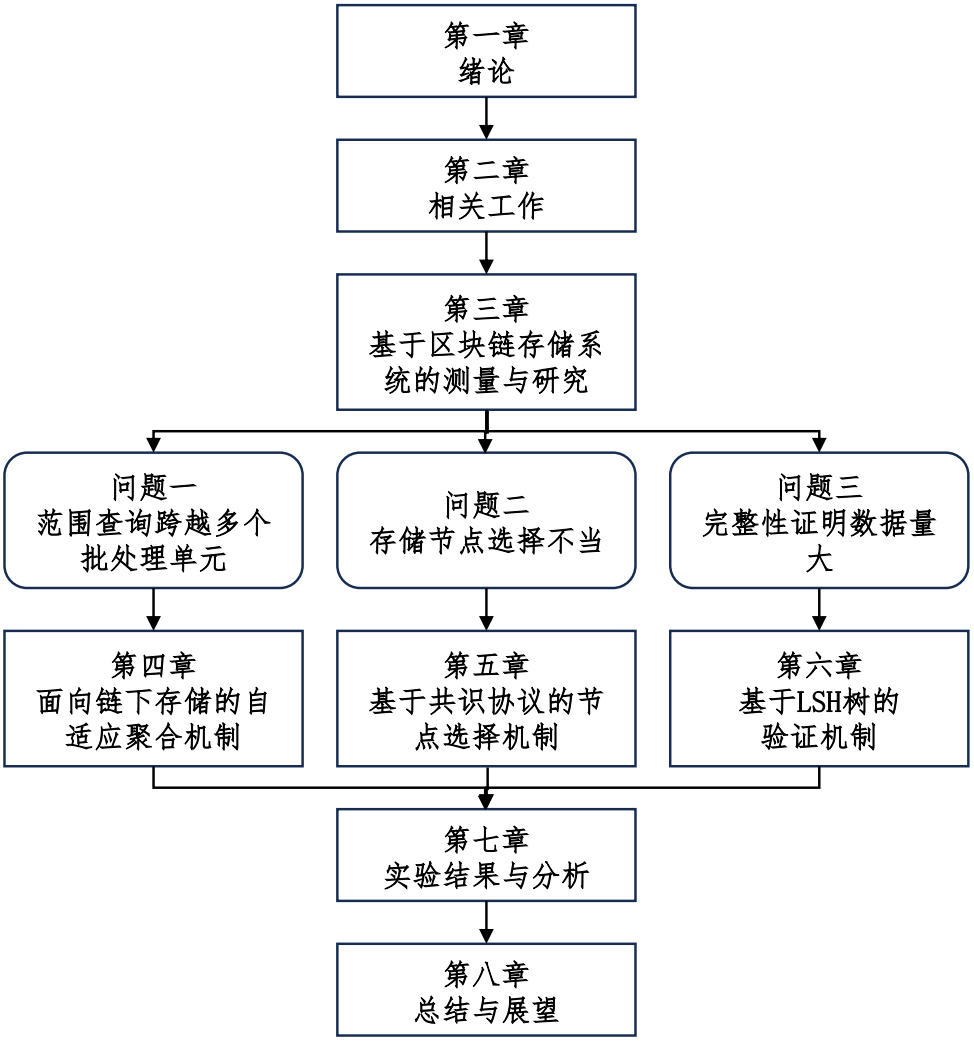


图 1.2 基于区块链的物联网时序数据高效可信存储技术研究内容

机制和链下存储机制的研究进展，为本文提出的 TimeChain 系统提供了改进的方向和理论支持。

第三章提出了基于区块链的物联网时序数据存储系统的基础架构，并通过对系统性能的实验测量评估，揭示了现有系统的性能瓶颈。本文进一步分析了这些瓶颈的根本原因，并提出了针对性的优化策略，提出了高效的存储技术 TimeChain。本文介绍了 TimeChain 的核心模块和各模块的功能，为后续章节的详细设计提供了基础。

第四章针对测量揭示的跨越多个聚合单元问题，提出了一种自适应聚合机制。本文通过构建无向加权图和采用谱聚类算法减少查询延迟，提高了数据存储的效率。本文也进一步讨论了该机制如何根据历史查询模式动态调整数据聚合策略，以适应不断变化的查询需求。

第五章针对测量揭示的存储节点选择错误问题，提出了一种基于共识协议的节点选择机制。该机制综合考虑了节点信誉和传输距离，提高了节点选择的安全性和准确性，降低了数据传输延迟。为了确保去中心化节点选择的效率，该机制将节点选择过程与共识过程相结合，以提高整个系统的抗攻击能力和可靠性。

第六章针对测量揭示的完整性验证数据较大问题，设计了一种基于 LSH 树的数据完整性验证机制。本文利用物联网数据的局部相似性，使用 LSH 算法减少了验证过程中的数据传输量，提高了验证效率。本文也进一步分析了 LSH 树在提高数据验证速度和减少网络负载方面的潜力，为物联网数据的高效验证提供了新的解决方案。

第七章实现并基于公开数据集评估了 TimeChain。本文将 TimeChain 与现有较为先进的解决方案进行性能对比，也通过消融实验进一步证明了 TimeChain 各技术点的有效性。

第八章总结了 TimeChain 系统的主要研究成果，并对未来的研究方向和 TimeChain 系统的改进进行了展望。本文也进一步讨论了如何将 TimeChain 系统拓展到更广泛的应用场景，以及如何进一步提升系统性能和安全性，为后续研究提供了方向。

2 相关工作

2.1 面向物联网时序数据的集中式存储系统

在物联网的传统存储系统中，中心化存储系统通常以其强大的数据处理能力和集中的数据控制而受到青睐。这种存储方式依托于中心节点的强大计算和存储能力，处理来自众多物联网设备的数据。集中式存储简化了数据管理流程，便于实现数据的统一备份和恢复，同时易于实施安全控制措施，如访问权限管理和数据加密。

AWS IoT[20]是亚马逊云服务（Amazon Web Services）提供的一个集中式云平台，专门设计用于支持和简化物联网设备的连接、管理和数据收集工作。作为一个集中式的解决方案，AWS IoT 提供了一个中心化的服务，允许来自全球各地的设备安全地将数据流传输到亚马逊的云基础设施中。通过集中式架构，AWS IoT 能够为设备数据提供一个统一的接入点，使得数据的管理和监控变得更加高效。这种集中式存储方式意味着所有的设备数据都汇聚到 AWS 的数据中心，由 AWS 进行集中管理和处理。

TDengine[21]是一个为处理时序数据而特别设计的高性能、分布式、支持 SQL 的集中式数据库系统。它以其卓越的性能和对时间序列数据的优化而受到关注，能够处理高吞吐量的数据写入和查询。为了减少每个数据存储分片之间的通信开销，TDengine 将分片存储在中心节点上，此外，TDengine 支持多种编程语言的 API，包括 C、Python 和 Java，这使得它可以轻松地集成到各种应用程序中。

TinyDB[22]是一个为传感器网络设计的查询处理系统，它通过一种称为获取式查询处理的方法来优化传感器网络中的时序数据采集和查询处理。TinyDB 将数据集中存储在中心节点，以此来减少数据传输和处理的开销，同时提供了对数据的高效查询和分析功能。

然而，尽管中心化存储系统在管理和控制方面具有优势，但它们也存在着一些不可忽视的问题。最主要的问题之一是单点故障风险，即如果中心存储节点发生故障，整个系统的数据存储功能可能会受到严重影响，甚至导致数据的不可用或丢失。恶意行为者只需针对中心节点发起攻击，就有可能破坏整个系统的稳定性，甚至盗取存储的数据。此外，随着物联网设备的激增和数据量的爆炸式增长，中心化存储系统可能会遇到扩展

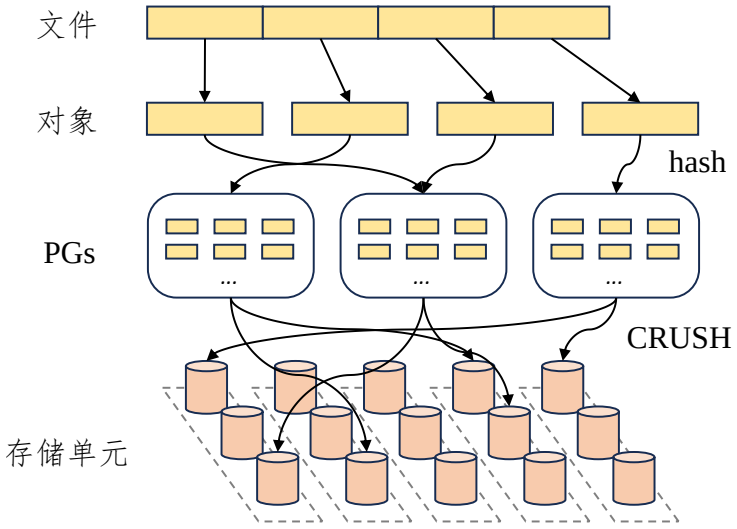


图 2.1 Ceph 存储流程

性瓶颈，处理和存储大量数据的需求可能会导致中心节点的性能瓶颈。

随着技术的发展，特别是区块链和分布式存储技术的进步，越来越多的解决方案开始倾向于去中心化，以提高系统的可靠性和安全性。中心化存储系统的局限性促使研究者和开发者寻求更为稳健的存储解决方案，以应对日益增长的数据存储需求。

2.2 面向物联网时序数据的分布式存储系统

与集中式存储相比，分布式数据库将数据分散存储在多个节点或数据中心中，这使系统本身就具备了抗单点故障的能力。即使某个节点或数据中心发生故障，系统仍然可以继续运行并提供服务，从而保障了数据的可靠性和稳定性。因此，随着对数据安全性、可用性和性能要求的不断提高，分布式数据库作为一种能够克服集中式存储固有弱点的解决方案，逐渐成为了当今大数据时代的主流选择。

Apache Cassandra[23]作为一个分布式结构化键值存储系统，借鉴了 Dynamo[24]的分布式系统技术和 Google BigTable[25]的数据模型优势，通过特殊的数据分发策略确保数据存储的高可用性和容错性。其灵活的数据模型和分布式特性使得它适用于需要横向扩展和高性能的应用场景。

Spanner[26]是谷歌开发的全球分布式数据库。它通过 TrueTime API[27]提供的时间戳服务实现全球性一致性和高可用性，并结合关系型数据库的 ACID 属性和分布式数据库的横向扩展，适用于需要高度一致性和水平扩展的工业应用场景。

Ceph[28]作为一个开源的分布式存储系统，提供对象存储、块存储等功能。为了平衡各存储节点的压力，它提出了可扩展哈希下的受控副本（Controlled Replication Under Scalable Hashing，CRUSH）算法，用于数据的分布式存储和数据冗余备份。如图2.1所示，对于需要被存储的文件而言，Ceph会首先将文件划分成多个独立的对象，然后Ceph将对象根据哈希分成若干放置组（Placement Group，PG）。随后，CRUSH算法结合地理位置信息将PG映射到存储集群中的存储节点，实现数据的均衡分布和高效访问。这种分布式的数据分布方式使Ceph系统具有高度的可扩展性和容错性，即使在节点故障或网络分区的情况下，系统仍能保持数据的可靠性和可用性。

CockroachDB[29]在Spanner的基础上，通过设计结合地理信息的复制和自动恢复机制来实现高容错性。它将数据分布在不同的地理位置或数据中心，确保数据的冗余备份分布在不同的地理区域。这种地理信息复制策略使得即使某个地理区域发生故障，数据仍然可以从其他地理位置的备份节点进行恢复。

然而，这些分布式存储系统仍然存在着单点故障和高存储成本的问题。这是因为即使数据存储在不同位置，数据的管理仍然是中心化的。在这些存储系统中，公司有可能在数据拥有者不知情的情况下轻松篡改数据，直接危及数据的完整性和安全性。这些缺陷限制了这些系统在处理物联网数据激增和复杂协作需求方面的灵活性和效率，同时也使得数据的安全性和可靠性备受质疑。因此，解决这些问题对于推动分布式存储系统在未来应用中获得广泛应用至关重要。

2.3 基于区块链的存储系统

区块链技术的出现为分布式存储系统的安全性和可靠性提供了新的解决方案。区块链存储系统是一种基于区块链技术的分布式存储解决方案，其核心概念是将数据分布式存储在网络中的多个节点上，并使用区块链作为数据的验证和记录机制。在区块链存储系统中，数据被分割成块，并以链式连接的方式进行存储，每个数据块包含了前一个块的哈希值，从而构成了不可篡改的数据记录链。这种数据结构使得数据的完整性和安全性得到了极大的保障，任何尝试篡改数据的行为都会被系统所记录和拒绝。区块链存储系统的去中心化特性意味着数据的管理不再依赖于单一实体，而是由网络中的多个节点共同维护和验证。这种设计使得数据拥有者可以更加信任系统，因为数据的安全性和可

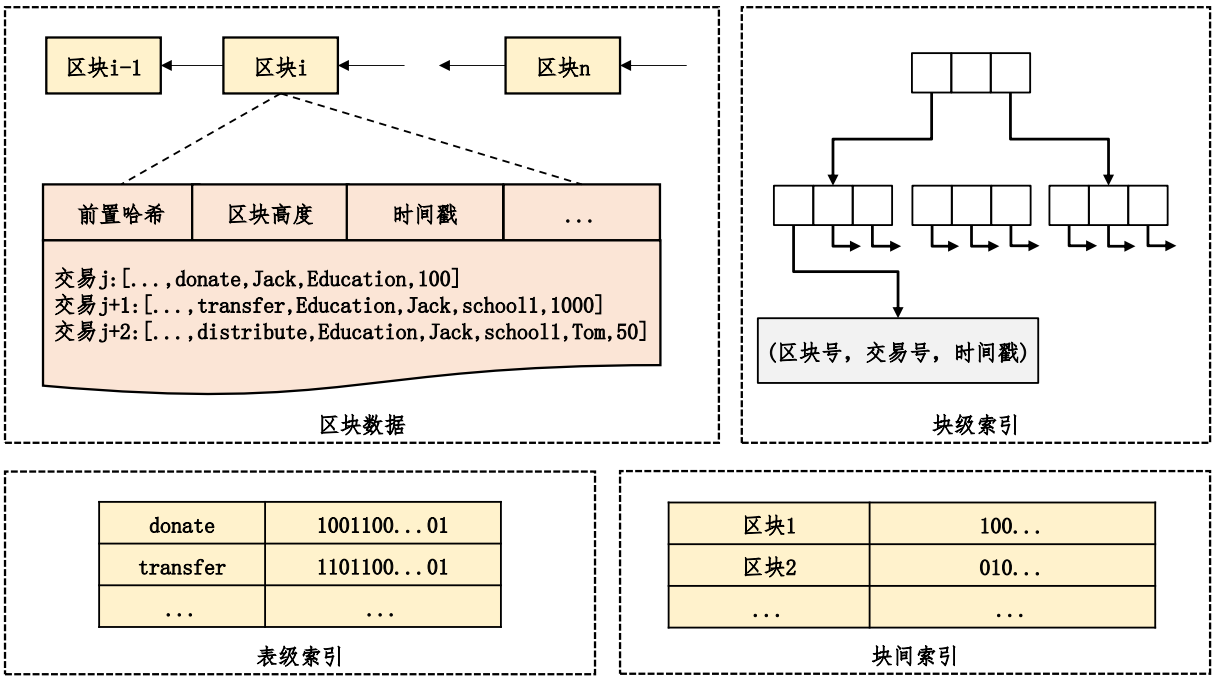


图 2.2 SEBDB 数据结构

靠性不再受限于单一实体的控制。

根据数据的存储位置，区块链存储系统可以分为链上存储和链下存储。

2.3.1 面向交易数据的链上存储

目前，许多链上存储的研究都集中在链上数据存储的性能上，包括优化链上存储的查询性能和减少链上存储负担等。

部分工作关注于增强链上存储的查询支持。由于区块链的交易数据都是以日志的形式存储在区块链上，无法提供类似关系型数据库的高效查找。因此，一些工作提出了在区块链上建立索引的方法，以支持对区块链数据的高效查询。SEBDB[5]将关系数据语义添加到区块链信息中，将每个事务视为预定义表中的元组，并使用类似 SQL 的语言作为通用接口，以支持方便的应用程序开发。如 图 2.2所示，SEBDB 使用 B+ 树建立了块级索引，将真实的数据组织成 B+ 树，以支持类 SQL 查询。SEBDB 也建立了表级索引和块间索引，便于快速定位区块位置。MSTDB[30]则建立了基于语义的索引结构默克尔语义树（Merkle Semantic Trie，MST），以此支持基于语义的多关键字查询、Top-K 查询、范围查询和跨链查询。MSTDB 也使用索引压缩和布隆过滤的方式减少索引的存储开销，提高了查询性能。

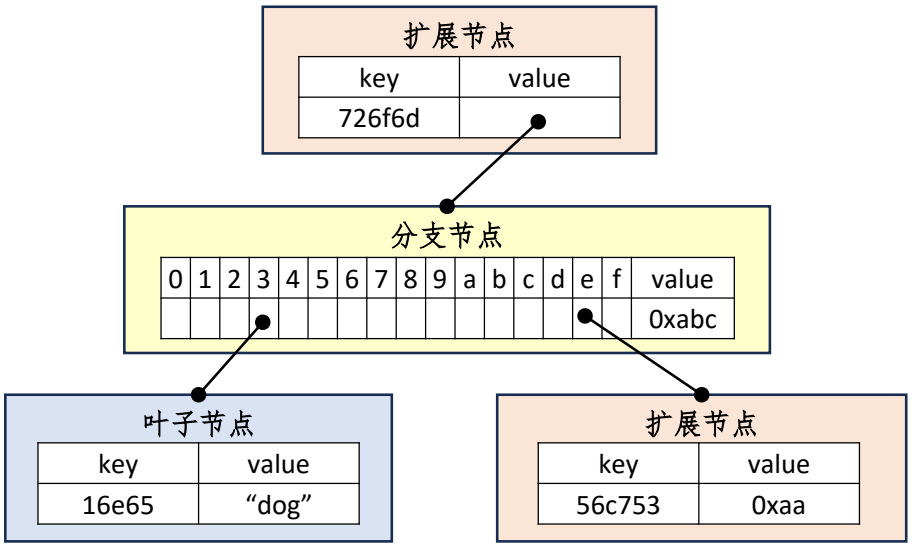


图 2.3 默克尔帕特里夏树结构

也有部分工作优化了区块上的索引和认证机制，以提高区块链状态的查询速度。默克尔帕特里夏树（Merkle Patricia Tree，MPT）是以太坊区块链中底层的数据结构，用于存储账户状态和交易信息。如 图 2.3所示，MPT 由三种类型的节点构成：叶子节点、扩展节点和分支节点。叶子节点包含实际的键值对，扩展节点包含指向其他节点的哈希值的键值对。分支节点包含多个可能的子节点路径，该节点使用十六进制进行编码分支。每个节点都用哈希值确保节点数据的完整性。针对 MPT 的磁盘访问开销，LVMT[8]设计了认证多点评估树（Authenticated Multipoint Evaluation Tree，AMT）在常数时间内更新完整性证明，并且采用多层设计来支持无限的键值对。LVMT 通过层级设计增强键值对可扩展性，通过存储版本号替代哈希的方式避免了昂贵的椭圆曲线乘法运算。针对 MPT 的索引重复存储开销，COLE[9]引入新兴的学习索引技术优化了区块链的存储索引结构。具体而言，COLE 遵循基于列的数据库设计来连续存储每个状态的历史值，这些历史值由学习模型索引，以促进高效的数据检索和出处查询。COLE 将数据按状态地址连续存储，通过学习索引模型减少存储索引开销；并针对区块链的存储磁盘特性引入了 LSM 树，以支持高效的读写性能。

部分工作关注链上存储的性能优化。FalconDB[31]和 Chen 等人提出的仲裁模型[32]都关注区块链节点之间数据交互的效率。为了减轻区块链账本数据存储的负担，Rapid-chain[10]、SlimChain[33]和 GriDB[11]将账本的数据分配给其他分片存储，以减轻存储压力。

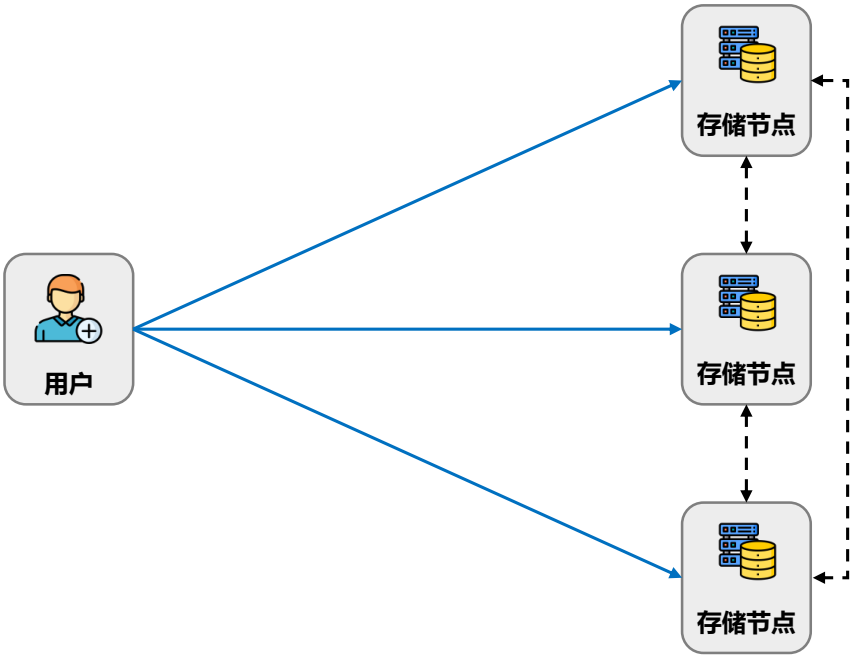


图 2.4 去中心化存储系统

在大规模数据生成迅速的物联网场景中，上述解决方案存在两方面的挑战：

- 首先，这些解决方案会给区块链带来额外的空间存储负担。由于物联网设备在短时间内生成的数据量庞大，将这些数据全部存储在区块链上可能会导致链的存储需求急剧增加，进而增加系统的复杂性和成本。

- 其次，这些方案可能导致数据隐私泄露的风险增加，因为在区块链上存储的数据通常是不可篡改和公开可见的，这会导致物联网的数据泄露。

综上所述，在数据密度低、生成速度快的物联网场景下直接使用传统的链上存储方案，会产生较大的开销和风险。物联网场景需要更加注重数据存储效率和隐私保护，因此需要针对这些特殊需求设计更为灵活和高效的数据管理方案。

2.3.2 面向文件数据的链下存储

基于区块链的文件系统因其较少的开销而备受广泛研究和关注。这些系统将文件本身存储在链下，规避了原始数据直接存储在链上带来的巨大开销，并通过在区块链上记录文件的哈希值和验证信息来确保文件的完整性和安全性。由于区块链的不可篡改性，用户可以方便地验证文件的来源和完整性，防止数据篡改和丢失。相较于传统的中心化

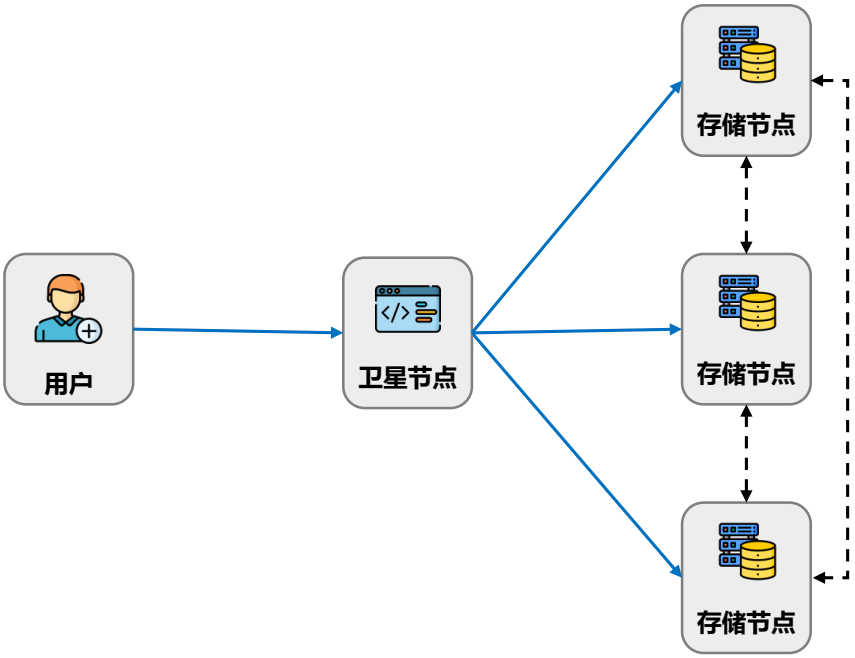


图 2.5 半去中心化存储系统

存储系统，基于区块链的文件系统通常不需要中心化的服务器和高昂的维护成本，从而降低了整体的运营成本。

FileCoin[34]是建立在 IPFS[17]上的去中心化文件存储系统。IPFS 是一种点对点的分布式文件系统，它通过内容寻址来存储和共享文件，使得文件在没有单点故障的分布式网络上可用。通过激励机制鼓励用户提供存储服务，FileCoin 利用区块链技术确保了文件的完整性和可靠性，同时为用户提供了一种去中心化的存储解决方案。如图 2.4所示，在 FileCoin 中，节点可以通过提供存储空间来获取 FileCoin 代币奖励，从而构建一个分布式的文件存储网络。对于这个完全去中心化的系统而言，所有存储节点之间都是平等的，每个节点将参与到用户的请求处理过程中。这种模式不仅激励了更多的存储资源加入网络，还提高了数据的持久性和可访问性，因为文件被分散存储在全球各地的节点上，减少了中心化存储的风险。为了确保文件的完整性，FileCoin 为每个文件建立默克尔树，通过分散存储文件块来提高可靠性和安全性，从而降低了数据丢失的风险。用户可以通过这些系统安全地存储和共享文件，同时保持数据的隐私和安全性。

Storj[13]和 Sia[35]是两个典型的半去中心化的文件存储系统。如图 2.5所示，在半去中心化网络中，除了存储节点，也存在一组卫星节点。这些卫星节点通过运行特定的智能合约来处理用户的请求，例如选择存储节点、进行文件切分、索引文件等。他们也为每个文件建立默克尔树以确保文件的完整性。

表 2.1 物联网存储系统比较

	去中心化	链上/链下	可扩展性	数据类型
AWS IoT[20]	○	-	●	传感器数据
TDengine[21]	○	-	●	传感器数据
TinyDB[22]	○	-	●	传感器数据
Spanner[26]	○	-	●	传感器数据
CockroachDB[29]	○	-	●	传感器数据
SEBDB[5]	●	链上	○	传感器数据
MSTDB[30]	●	链上	○	传感器数据
Filecoin[34]	●	链下	●	文件
Storj[13]	●	链下	●	文件
FileDES[36]	●	链下	●	文件
TimeChain	●	链下	●	传感器数据

FileDES[36]专注于数据的加密存储，借助零知识证明等技术来保护数据的安全性。通过加密存储数据，FileDES 可以确保用户的数据在存储和传输过程中不会被泄露或篡改。这种方法为用户提供了更高级别的数据安全保障，尤其适用于对数据隐私和安全性要求较高的场景。

然而，尽管这些方法在确保文件完整性和安全性方面表现出色，它们并不适用于高效存储物联网数据的存储。这是因为物联网数据通常具有较低的价值密度，而将单个数据以文件形式存储在区块链文件系统中，将导致极高的成本。物联网数据的特点决定了需要更为轻量级和高效的数据存储方案，以更好地应对数据生成量大、频繁性高的特点，同时降低存储和处理成本。因此，对于物联网大规模数据存储的需求，寻找轻量、高效、可信的数据解决方案是一个挑战。

2.4 本章小结

在本章中，本文广泛探讨了面向物联网的分布式存储系统和基于区块链的存储系统的相关研究工作。本文分析了集中式数据存储解决方案的局限性，特别是它们在面对单

点故障时的脆弱性，以及分布式数据库如何通过在多个节点间分散存储数据来提高数据的可靠性和稳定性。本文详细讨论了 Apache Cassandra、Spanner、Ceph 和 CockroachDB 等分布式存储系统，它们通过不同的技术手段实现了数据的高可用性和容错性。然而，这些系统仍然存在单点故障的风险，并且由于数据管理的中心化，数据的完整性和安全性仍然受到威胁。

进一步地，本文探讨了基于区块链的存储系统，它们通过将数据分布式存储在多个节点上，并利用区块链作为数据验证和记录的机制，提供了一种新的数据存储解决方案。本文分析了链上存储和链下存储两种方法，以及它们在提高数据存储性能和确保数据完整性方面的研究进展。尽管这些基于区块链的存储系统在确保数据完整性和安全性方面具有优势，但它们在处理大规模、低价值密度的物联网数据时面临效率和成本的挑战。

综上所述，现有的分布式存储系统和基于区块链的存储系统虽然在某些方面取得了进展，但它们在满足物联网数据存储需求方面仍存在不足。这些挑战包括处理大规模数据的效率、数据存储的成本以及数据隐私保护的需求。因此，开发一种轻量级、高效且可信的数据存储解决方案对于物联网领域来说是一个重要的研究方向。TimeChain 正是在这样的背景下应运而生，旨在结合区块链技术的优势，解决物联网时序数据存储中的关键问题。

3 基于区块链存储系统的测量与研究

本章将深入探讨基于区块链的物联网时序数据存储系统的基础架构，并对该系统进行了全面的测量研究。基于测量研究，本文发现聚合存储能够显著降低存储延迟，但在物联网场景中较为常见的范围查询中，同时也引入了查询延迟的问题。本章进一步分析了导致查询性能不佳的根本原因，包括查询跨越多个聚合单元、存储节点选择不当以及庞大的数据完整性证明。为了解决这些问题，本文提出了 TimeChain 架构，这是一种新颖的、基于区块链的物联网时间序列数据存储系统。本章介绍了 TimeChain 系统的核心模块，包括数据批处理模块、存储节点选择模块和数据验证模块。这些模块共同作用，旨在减少查询延迟、提高数据存储的效率和安全性。

3.1 基于区块链存储系统的架构

为了提高基于区块链的分布式数据库的性能，本文提出了一种基本的链下存储系统，并对其进行了测量研究。如 图 3.1所示，一个基于区块链的基本分布式存储系统有四个主要角色，即**数据拥有者**、**网关**、**卫星**和**存储提供商**。

- **数据拥有者**：数据拥有者是数据的生成者和消费者，他们发起上传和查询数据的请求。
- **网关**：网关是数据拥有者与存储提供商之间的中间层，负责数据的上传和下载。
- **卫星**：与 Storj[13]和 FileCoin[34]类似，卫星是一个智能合约，负责协调数据拥有者和存储提供商之间的交互。它们提供文件审计（检查）或可检索性（Proofs of Retrievability，POR）相关功能和对存储支付等操作的处理。
- **存储提供商**：存储提供商是存储数据的节点，通过提供存储服务获得奖励。为了使存储提供商能够通过灵活的查询快速向数据拥有者提供完整性证明，数据完整性证明也需要存储在存储提供商中，便于通过默克尔树的路径直接快速组装成完整性证明。存储提供商的服务信息，如剩余存储空间，将与交互记录一起记录在分布式账本中，以确保安全。

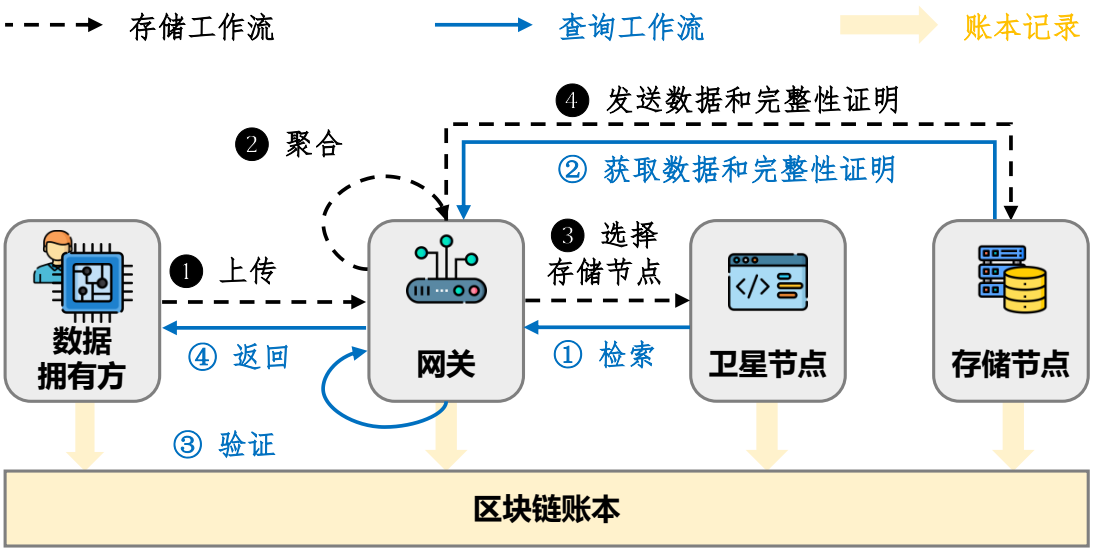
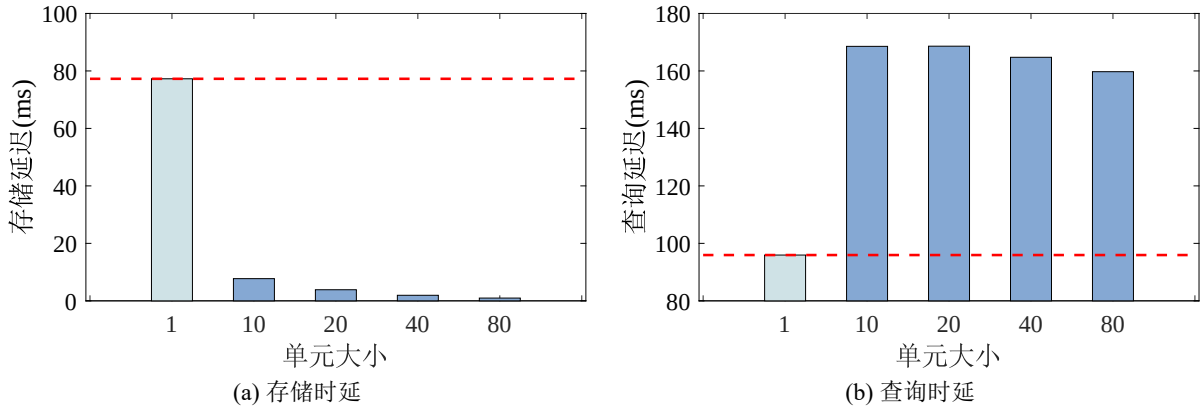


图 3.1 基于区块链的存储系统的工作流

一般来说，数据存储和查询过程可以概括如下：

数据存储： ❶上传：数据拥有者通过网关接口上传数据。❷聚合：网关接收到数据后，会对时间序列数据进行批处理。这一步骤涉及到将多个数据点聚合在一起，并为每批数据生成数据完整性证明，这对于确保数据在存储和后续查询过程中的完整性至关重要。❸选择存储节点：在聚合数据后，卫星节点介入，帮助网关识别和选择最佳的存储节点。这一步骤基于一系列选择标准，如节点的信誉、存储空间和地理位置等，以确保数据被存储在最可靠和最经济的节点上。❹发送数据和完整性证明：一旦确定了最佳存储节点，原始传感器数据及其完整性证明将被发送到这些节点。同时，数据批的元数据被记录在分布式账本中，利用区块链的不可篡改性确保数据的透明性和可追溯性。

数据查询： ①检索：当数据拥有者需要访问他们的数据时，他们通过网关发起数据下载请求。网关随后与卫星节点交互，以检索存储提供商的位置信息。②获取数据和完整性证明：网关根据从卫星节点获得的位置信息，从存储提供商处获取所需的数据及其完整性证明。这一步骤确保了数据在传输过程中的安全性和完整性。③验证：在获取数据后，网关将通过检查数据完整性证明来验证下载数据的完整性，确保数据未被篡改且准确无误。④返回：最后，经过验证的数据将通过网关安全地返回给数据拥有者，完成整个数据查询流程。



(a) 存储时延 (b) 查询时延

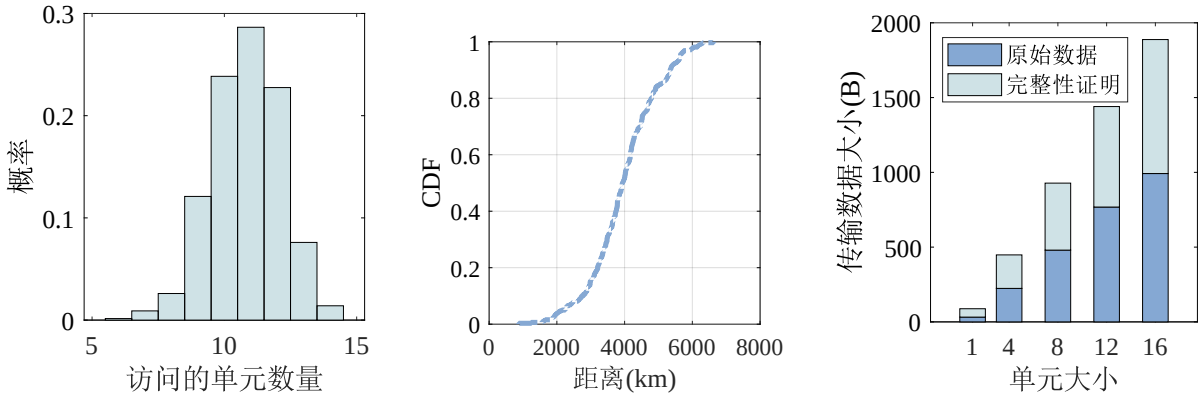
图 3.2 区块链存储系统的性能

3.2 基于区块链存储系统的测量

在本节中，本文进行了一项初步测量，以评估基于区块链的基本存储系统的性能。本文使用 Hyperledger Fabric 作为区块链平台，IPFS 作为文件存储系统，实现了上述基本存储系统。该测试网络由 5 个节点组成，其中 1 个节点是网关，4 个节点是卫星。为了更准确地模拟真实世界中的存储网络环境，本文基于真实世界的数据存储网络模拟了一个包括 320 个存储节点的网络，其中包括 189 个真实存储提供商[37]和 131 个模拟提供商（代表提供闲置存储的个人提供商）。为了确保实验结果的准确性，网关与这些模拟提供商之间的传输延迟是通过一个基于现有研究的线性回归模型[38]进行预测的。

存储性能。在本文设置的系统中，数据拥有者每秒生成 20 个 56 字节的数据包，并在 20 秒内存储它们。存储性能结果如图 3.2a所示。与单独存储每个数据相比，批存储将延迟减少到约 37.4 分之一，且随着聚合单元增大，存储延迟逐渐降低。这主要是因为较大的聚合单元规模减少了链上操作和数据传输的次数，从而降低了存储延迟。

查询性能。然后，本文测试范围查询的性能，如图 3.2b所示。不幸的是，结果显示，相对于存储单个数据而言，批处理存储解决方案的查询性能相对较差。批量存储在不同聚合单元大小下的平均延迟为 165.4ms，无法满足许多物联网场景的需求。例如，自动驾驶应用程序的延迟小于 50ms[39]，地震监测的延迟小于 100ms[40]。



(a) 查询跨越多个 batch (b) 不正确的存储节点选择 (c) 传输数据较大

图 3.3 性能低下的根本原因

3.3 性能瓶颈的根本原因

为了找出该方案查询性能不佳的原因，本文进行了深入研究，并将原因总结为以下三点：

1. **范围查询跨越多个批处理单元。**时间序列数据的查询通常包含多个数据点，如范围查询、聚合查询、过滤查询等。如果批处理不当，单个查询可能会跨越多个批处理单元。本文使用现有的数据集 YCSB[41]评估每个查询所跨越的批处理单元数量。如图 3.3a所示，超过 84.25% 的查询跨越了 10 个批处理单元。当这些批处理单元被存储在不同的节点上时，向不同存储节点获取数据的过程将引入额外的查询和传输延迟。

2. **存储节点选择不当。**在这个测量中，本文发现传输延迟占总查询延迟的很大一部分。如图 3.3b所示，世界各地存储节点的距离存在很大差异，导致节点之间的传输延迟存在很大差异。选择非常远的存储节点将导致传输延迟显著增加。此外，在整个存储网络中存在恶意节点的情况下，尽管最终选择的节点可能传输延迟较小，然而，由于恶意节点的存在，数据的完整性可能会受到威胁，从而无法提供完整的数据存储服务。

3. **完整性证明数据量大。**在数据获取阶段中，存储提供商为了证明数据的完整性，需要生成数据完整性证明并发送给数据拥有者。为了使存储提供商能够通过灵活的查询快速向数据拥有者提供完整性证明，数据完整性证明也需要与存储提供商一

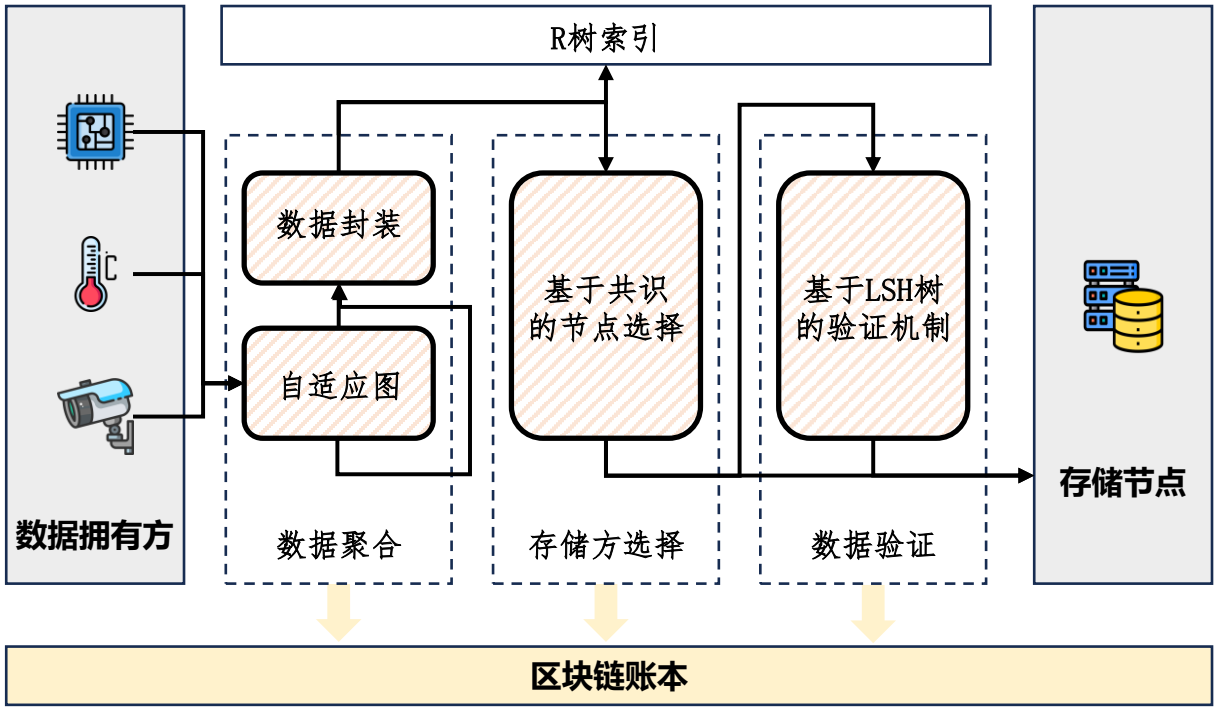


图 3.4 TimeChain 架构

起存储。图 3.3c显示了在数据获取阶段时，总传输数据量的各组成部分。从图中可以看出，数据完整性证明大小占接收数据的 48.8%，几乎是接收数据的一半。当网络繁忙时，大量的完整性证明数据传输会增加网络传输延迟。

3.4 高效的链下存储架构

为了提高区块链存储系统的查询性能，本文设计了一种基于区块链的新型物联网时间序列数据存储系统，TimeChain。图 3.4显示了 TimeChain 的架构。TimeChain 中的模块包括数据聚合、存储方选择、数据验证、R 树索引和区块链账本。TimeChain 建立在区块链平台上，所有操作都记录在分布式账本上，以方便数据的审计和追踪。TimeChain 的索引结构是 R 树，可以加速物联网场景中常用的时空聚合搜索。

原始数据经过数据聚合模块，确定哪些数据被放在一个聚合单元中；经过存储方选择模块，确定聚合单元的存储位置。这些信息将被放在 R 树中，以便快速查询。接下来，本文介绍 TimeChain 的关键模块。

数据聚合模块：本文的测量研究表明，不正确的聚合方法会增加查询的跨单元数量，从而增加网络传输延迟。本文构建了一个自适应的无向加权图（Undirected Weighted

Graph，UWG），以基于数据拥有者的历史查询准确捕获用户查询信息（§-4.1）。该图可以根据数据拥有者的历史查询记录动态更新，具有良好的适应性和准确性。基于本文构建的 UWG，本文将数据打包成批处理单元的问题转换为聚类问题[42]，根据用户的查询请求将所有原始数据划分为多个聚类。解决聚类问题的传统算法有很多，如 K-means[18]、GMM[19]等。然而，这种传统的聚类算法不适合分割物联网设备生成的数据。这是因为在 TimeChain 中，用户的查询没有遵循特定的特征，这可能会导致数据图的聚类形成复杂的形状，而不是常见的圆形。此外，传统的聚类算法需要将所有数据划分为固定数量的集合，但并非所有集合都等于批大小，这将给索引查询带来额外的开销。因此，本文使用谱聚类算法来打包数据（§-4.2），这非常适合处理不规则和非固定数量的聚类。

存储方选择模块：存储节点的选择至关重要。正如本文之前在图 3.3b中发现的那样，存储节点和网关之间的距离会影响数据访问延迟。此外，对于链下存储系统，存储空间不足的节点或恶意节点可能会导致数据丢失、篡改或服务中断，进而影响整个系统的安全性和稳定性。因此，本文在存储节点的评估过程中综合考虑距离和历史服务记录等信息。存储节点选择过程的安全性也非常重要。Storj[13]、CoopEdge[43]和 PipeEdge[44]通过一组固定的节点选择服务节点，并通过区块链确认决策。换句话说，他们集中决策，但仍然面临单点失败的威胁。然而，使用类似于 PBFT[45]的投票机制，节点选择过程通常需要多轮任务计算和消息广播。如果共识过程和节点选择过程完全解耦，系统安全将受到威胁。为了解决这个问题，本文将节点选择过程与共识相结合，提出了一种基于共识的节点选择机制（§-5）。本文也进一步分析了该机制的安全性。

数据验证模块：从之前的测量结果中本文可以发现，传输的数据中接近一半是数据完整性证明。数据完整性证明被组织为默克尔树的形式，它是由一系列哈希构建的。在默克尔树中，非叶子节点的哈希数几乎等于原始数据点的数量。由于物联网数据单元的大小大约等于哈希值，这意味着需要发送以验证数据的数据量几乎是原始数据的两倍。在这样的数据传输过程中，减小数据证明的大小是一个挑战。通过分析物联网数据，本文观察到物联网数据变化缓慢，在短时间内很少出现突然变化。对于这些相似的数据，LSH 算法可以从相似的原始数据中生成相似的哈希结果。LSH 算法确保类似的物联网数据即使在哈希后也保持相似。因此，本文提出了一种新的基于 LSH 树的验证机制（§-6），该机制采用 LSH 而不是传统默克尔树中使用的通用哈希。通过差分传输 LSH 哈希值，可以显著减小传输数据的大小。

3.5 本章小结

在本章中，受链下存储启发，本文先提出了一种基于区块链的物联网时序数据基本存储系统。对于这个基本系统，本文进行了一些性能测试，发现该系统确实可以降低存储延迟，但查询延迟较高。本章进一步分析了查询性能不佳的根本原因，并针对性能瓶颈提出了高效链下存储系统 TimeChain 的架构。对于跨越多个聚合单元问题，本文在聚合阶段提出了一种自适应聚合机制。对于存储节点选择错误问题，本文在节点选择阶段提出了一种基于共识协议的节点选择机制。对于完整性验证数据较大问题，本文在验证阶段设计了一种基于 LSH 树的数据完整性验证机制。本文将在后续段落中进一步介绍这三个机制的设计。

4 面向链下存储的自适应聚合机制

本文的测量结果揭示了一个关键问题：不当的数据聚合策略会导致查询过程中需要跨越更多存储单元，进而增加了网络传输的延迟。在本章中，本文提出了一种自适应UWG的方法，它能够根据数据所有者的历史查询模式精确捕捉用户的查询行为。利用这个自适应UWG，本文将数据分批的问题转化为一个聚类问题，即根据用户的查询需求将海量原始数据划分成多个有意义的聚类。

4.1 基于历史查询的自适应图构建过程

由于物联网的原始数据是孤立的点，本文在数据点之间创建加权边，表示被联合查询的概率。数据点 a 和 b 之间的边的权重 l_{ab} 表示为：

$$l_{ab}=\begin{cases}\sqrt{(id_a-id_b)^2+(t_a-t_b)^2} & ,k=0\\ \theta\cdot l_{ab}+(1-\theta)\cdot x_{ab}^{k} & ,k\geq 1\end{cases} \tag{4-1}$$

其中 l_{ab} 被初始化为两个数据点设备ID和数据到达时间之间的欧氏距离。当用户的请求到达时，UWG会根据请求中涉及的数据查询范围进行动态调整。为了避免查询图的过度存储开销，本文在更新图时忽略了数据的时间维度，只考虑数据的设备ID维度。

本文使用变量 x_{ab}^{k} 来指示第 k 个查询的内容。当第k个查询包含设备 d_a 和设备 d_b 时，$x_{ab}^{k}=1$，否则 $x_{ab}^{k}=0$。然后，距离 l_{ab} 将根据 x_{ab}^{k} 进行更新。

由于用户请求可能非常随机，因此图的更新和变化可能会非常频繁。因此，本文设置了一个影响因子 θ 来确定用户请求对图更新的影响。当 θ 接近1时，权重受查询的影响更大。当 θ 接近0时，这意味着图尽可能保持初始状态。

同时，TimeChain使用滑动窗口动态更新UWG，尽可能避免单个查询的对系统的鲁棒性影响。虽然查询模式的突然变化最初会降低打包准确率，但系统会在多次查询后快速适应。通过自适应权重聚类算法，本文可以根据节点之间的距离和查询的相关性动态调整节点之间的权重，以更好地反映它们的相似性。这有助于在打包过程中更准确地确定哪些节点的数据应放置在同一批中，以提高查询的效率和准确性。

表 4.1 TimeChain 关于聚合的符号和定义

符号	描述
d_i	第 i 个设备
D	所有设备集合，$D=\{d_0,...,d_n\}$
s_i	物联网传感器产生的第 i 条数据
S	所有数据集合，$S=\{s_0,...,s_m\}$
Q^i	第 i 条用户查询集合，$Q^i=\{q_0,...,q_r\}$
q_k	k 中的第 Q^i 条查询
l_{ab}	查询权重包含设备 a 和设备 b
L	一起查询的数据权重，$L=\{l_{ab}\|\exists_{a,b}\in D\}$
x_{ab}^k	设备 a 和设备 b 是否同时被查询
X^k	设备访问信息，$X^k=\{x_{ab}^k\|\exists_{a,b}\in D\}$
P	数据打包结果，$P=\{\{s_i,...\},...\}$

4.2 基于谱聚类算法的封装机制

4.2.1 聚类算法选择

通过建立自适应的 UWG，本文将数据分批的问题转化为聚类问题。本文综合考虑了目前的几类聚类算法：基于划分的算法、基于模型的方法以及基于图论的方法。

基于划分的算法。基于划分的算法通过迭代优化过程，将数据集划分为预定义数量的簇。这类算法中最著名的是 K-means[18]。K-means 算法首先随机选择 k 个点作为初始质心，然后将每个数据点分配给最近的质心所在的簇，并重新计算每个簇的质心。这个过程不断重复，直到质心不再显著变化或达到最大迭代次数。尽管这种方法简单直观且计算效率较高，但它对簇的数量 k 非常敏感，初始质心的选择也会影响最终结果。此外，K-means 假设簇是球形分布的，难以处理非球形簇或复杂形状的数据。因此，对于 TimeChain 输入的时间序列数据，这些算法可能无法有效捕捉数据的真实分布，导致聚类效果不佳。

基于模型的方法。基于模型的方法通过假设数据是由特定的概率分布生成的，并使用统计模型进行参数估计来实现聚类。最典型的方法，例如高斯混合模型[18]（Gaussian

Mixture Models, GMM），通过迭代地更新每个数据点属于各个高斯分布的概率，以及各个高斯分布的均值、协方差矩阵等参数，逐步优化模型，直到收敛。虽然 GMM 能够提供概率解释并适用于特定类型的分布，但其计算复杂度较高，特别是在处理大规模数据集时，需要大量的计算资源和时间。对于物联网设备来说，由于资源受限（如内存、CPU 性能），这种复杂的计算对于弱性能设备而言难以承受，从而影响系统的实时性和响应速度。另一种基于模型的方法是隐马尔可夫模型[19]（Hidden Markov Model, HMM），假设数据是由隐藏状态序列生成的，通过训练 HMM 模型，可以发现隐藏状态的转换模式，从而实现聚类。然而，HMM 的训练过程同样涉及复杂的概率计算，特别是当状态空间较大时，计算复杂度会显著增加。因此，尽管这些方法在某些场景下非常有效，但在 TimeChain 的应用中并不理想，因为它们不适合资源受限的物联网环境。

基于图论的方法。基于图论的方法，特别是谱聚类[46]（Spectral Clustering），利用图的拉普拉斯矩阵的特征向量来进行聚类，非常适合处理复杂的簇结构。谱聚类的基本思想是将数据点表示为图中的顶点，顶点之间的边权重表示数据点之间的相似性或距离。具体来说，首先根据数据点之间的相似性或距离度量构建一个加权图，其中每个节点代表一个数据点，边的权重表示节点之间的相似性。接下来，从相似性图中计算出图的拉普拉斯矩阵。拉普拉斯矩阵是一种特殊的矩阵，反映了图的拓扑结构和节点之间的关系。随后，对拉普拉斯矩阵进行特征分解，提取前几个最小的非零特征值对应的特征向量。这些特征向量捕获了图的主要结构信息。最后，将提取的特征向量作为新的低维表示，应用传统的聚类算法（如 K-means）进行聚类。谱聚类不仅能够发现任意形状的簇，而且相对 GMM 和 HMM 而言，其计算复杂度是可以接受的，特别适合处理大规模数据集。此外，谱聚类在处理噪声数据和不规则形状的簇方面表现优异。因此，TimeChain 选择了谱聚类算法进行聚类分析，以高效地处理物联网时序数据，并确保在资源受限的环境中也能保持良好的性能。

4.2.2 数据封装过程

4.1显示 TimeChain 的整个打包过程。该算法的输入包括输入设备集 D、数据集 S 和用户的历史查询集 Q^i。由于该 UWG 的数据点是忽略时间维度的，因此 TimeChain 在预处理数据集 S' 时，将同一个设备 ID 的数据点组织到单位时间窗口长度对应的一个集合中（第 1 行）。TimeChain 根据式 4-1计算每个点之间的权重 l，并将权重组合成权重集

算法 4.1 聚合算法

Require: 输入设备集 D，数据集 S，用户的历史查询集 Q^i

Ensure: 最优聚合结果 P

1: $S' \leftarrow \{s^a | a \in D \ \& \ s^a \subset S\}$ //根据设备 ID 梳理原始数据
2: $L \leftarrow \left\{ \sqrt{(id_a - id_b)^2 + (t_a - t_b)^2} \middle| \exists_{a,b \in D} \right\}$ //根据式 4-1初始化权重集 L
3: $X \leftarrow \{0 | \exists_{a,b \in D}\}$
4: **for** $q^k \in Q^i$ **do**
5: **if** $a, b \in q^k$ **then**
6: 使用 $x_{ab}^k \leftarrow 1$ 更新 X //将用户查询请求信息 Q^{i-1} 打包成集合 X^k
7: **end if**
8: **end for**
9: $L \leftarrow \left\{ \theta \cdot l_{ab} + (1-\theta) \cdot x_{ab}^k \middle| \exists_{a,b \in D} \exists_{l_{ab} \in L} \right\}$ //更新 UWG 的权重集 L
10: $D' \leftarrow cluster(D, L)$ //使用谱聚类算法获得聚合结果 D'
11: $P \leftarrow \{\}$ //初始化聚合结果 P
12: **for** $d^j \in D'$ **do**
13: 增加 $\{s^a | \exists_{a \in d^j}\}$ 到 P 中 //将设备数据合并到 P 中
14: **end for**
15: **return** P

L（第 2 行）。对于上一个滑动窗口中的历史查询记录集合 Q^{i-1}，TimeChain 根据每个查询涉及的具体数据点整合查询特征集合 X（第 4-8 行）。即，当 q^k 中包含设备 a 和设备 b 时，x_{ab}^k 被设置为 1。否则，x_{ab}^k 被设置为 0。然后，本文根据收集到的查询信息 X，根据 $l_{ab} = \theta \cdot l_{ab} + (1-\theta) \cdot x_{ab}^k$ 更新权重集 L（第 9 行）。对于更新后的权重集 L，TimeChain 可以构建出迭代后的 UWG(D, L)，图中的节点表示设备数据点，边长为数据点之间的查询权重。对于 UWG(D, L)，根据之前的分析，TimeChain 使用谱聚类算法来获得聚合结果 D'（第 10 行）。D' 中被聚合在一起的点表示经常被一起查询的数据点。根据聚合结果 D'，本文将数据整合到 P 中，这是本文得到的聚合结果（第 12-14 行）。

4.3 本章小结

在本章中，本文深入研究了面向链下存储的自适应聚合机制，旨在解决物联网时序数据存储中的查询延迟问题。本文发现，不当的数据聚合策略会导致查询过程中需要跨越更多存储单元，从而增加了网络传输的延迟。为此，本文提出了一种基于历史查询的自适应无向加权图（UWG）方法，该方法能够精确捕捉用户的查询行为，并根据这些行为将数据分批处理，以优化查询效率。对于根据用户查询特征生成的图，本文综合分析了目前的聚类算法，针对当前问题的特点提出了一种基于谱聚类算法的封装机制，以更好地处理不规则形状的数据聚类问题。通过这种方法，本文可以更准确地确定哪些节点的数据应放置在同一批中，以提高查询的效率和准确性。

5 基于共识协议的节点选择机制

确定最佳的存储节点选择是构建高效区块链存储系统的关键一环。根据本文测量中的分析，如图 3.3b所示，存储节点与网关之间的距离对数据访问延迟有着直接的影响。此外，在链下存储环境中，节点存储空间不足或恶意行为可能会导致数据的丢失、被篡改或服务中断，这些都会对系统的安全性和稳定性造成严重影响。因此，如何选择最佳的存储节点是一个挑战。

5.1 中心化节点选择

在区块链和分布式存储系统中，存储节点的选择是确保数据安全性和可靠性的关键步骤。目前，许多系统如 Storj[13]、CoopEdge[43]和 PipeEdge[44]根据节点的信誉选择存储节点，通过区块链来确认节点选择的决策。鉴于 PBFT 共识算法在分布式系统中的普遍应用，本文将阐述一个基于 PBFT 算法的去中心化存储系统中节点选择的具体步骤和流程。对于一个具有 $3f+1$ 个节点的网络中，PBFT 共识协议要求至多存在 f 个恶意节点。

如图 5.1所示，中心化节点选择的流程如下：

1. **决定提案**：由一个或一组节点负责评估网络中各个存储节点的信誉，根据节点的历史表现、可靠性和性能指标等因素，选择出适合承担数据存储任务的节点。

2. **广播**：决策节点（组）将最佳存储节点的提案广播给 $3f+1$ 个存储节点。

3. **准备**：每个存储节点决策是否通过最佳存储节点的提案，并将决策结果广播给其他的存储节点。

4. **预提交**：当每个节点收到大于 $2f+1$ 个相同信息（是否通过提案）时，视作该节点与其他节点的信息充分交流，并将综合的投票结果广播给其他存储节点。

5. **提交**：当每个节点收到 $2f+1$ 个信息时，选择多数一致的决策记录上链。由于该网络中最多存在 f 个恶意节点，因此该阶段的决策中，虚假信息的数量一定少于诚实信息，所以该系统并不会受到恶意节点的威胁。

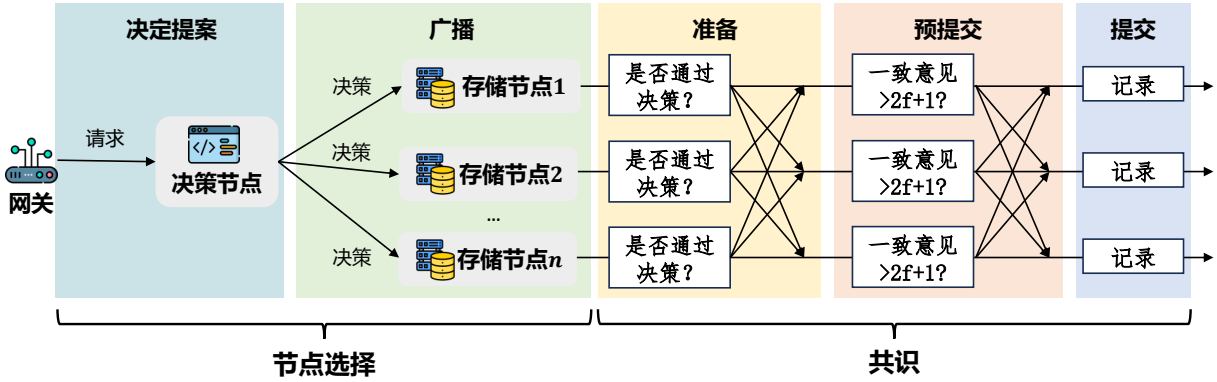


图 5.1 中心化节点选择的工作流

在这样的节点选择流程中，中心化节点选择引入了单点故障的风险，即如果中心节点发生故障或被恶意攻击，整个系统的节点选择和数据存储过程可能会受到影响，导致服务中断或数据丢失。同时，这种中心化的节点选择也会导致系统的扩展性受限。随着网络节点数量的增加，中心节点需要处理的信息量也会增加，这可能导致选择过程的延迟增加，影响整个系统的响应速度。中心节点的计算和存储能力可能成为系统的瓶颈，限制了系统处理大规模数据的能力。

而且，共识形成过程与节点选择过程完全分离会使系统的安全性会受到威胁。这是因为，共识过程负责确保所有节点对数据状态达成一致，而节点选择过程决定了哪些节点参与到共识中。如果这两个过程分离，恶意节点可能利用这一点来破坏系统的安全性，例如通过选择不可靠的节点参与共识过程，从而影响数据的完整性。

5.2 结合共识的节点选择过程

在前一节中，本文介绍了中心化节点选择的具体过程。仔细观察该流程可以发现，共识过程中涉及了两次信息广播，每次信息广播都需要所有存储节点去中心化地决策。考虑到中心化节点选择带来的风险，TimeChain 创新性地将节点选择过程与共识的计算和广播过程结合，提出了一种基于共识机制的节点选择算法。

在该节点选择过程中，TimeChain 利用信息广播阶段来交换各节点对存储节点的去中心化评估过程。具体而言，TimeChain 的节点选择过程包括**请求**、**准备**、**预提交**、**提交**和**回复**，其详细步骤如 图 5.2所示，伪代码如 算法 5.1所示。

与 PBFT 共识类似，在**请求**阶段，网关向系统中的所有节点发送请求，共识节点将在**回复**阶段将获得的结果返回给网关。同样，类似于 PBFT 共识，本文假设恶意节点的

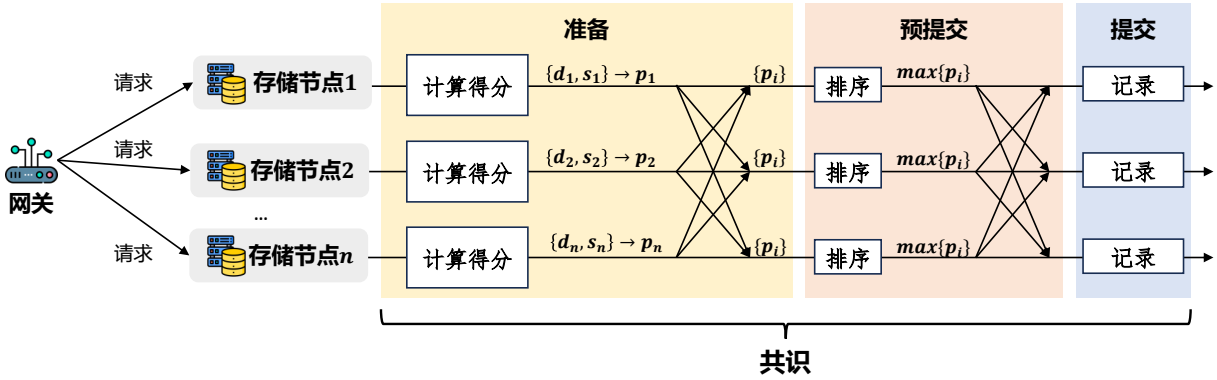


图 5.2 基于共识的节点选择机制工作流

数量为 f，所有存储节点的数量超过 $3f+1$。接下来，本文将分别介绍其中的核心阶段：**准备**、**预提交**和**提交**。

5.2.1　准备阶段

在该阶段，每个节点通过考虑存储节点的距离、信誉等来计算分数，并将分数广播给所有其他节点。TimeChain 选择距离、剩余存储和存储质量作为指标，原因如下：根据图 3.3b，本文发现存储节点的距离会影响数据访问延迟。因此，本文选择距离作为一个重要的指标。此外，受到 FileCoin[34]的启发，剩余存储空间和存储质量也是选择存储节点的重要因素。剩余存储空间可以直接影响存储提供商的服务能力，而存储质量可以影响数据的完整性。第 i 个节点的得分计算公式如下：

$$p_i = \alpha \cdot d_i + \beta \cdot s_i + \gamma \cdot q_i \tag{5-1}$$

其中 d_i 表示第 i 节点和客户端节点之间的距离；q_i 表示存储服务质量，可以从链上的服务记录中评估；s_i 表示节点的剩余存储空间。所有这些数据都可以在链上找到。α、β 和 γ 是加权参数，这些系数可以根据特定的系统需求和性能要求进行调整。例如，可以增加的权重以优先考虑存储完整性，而可以调整的权重以降低延迟。

5.2.2　预提交阶段

在本阶段，共识节点从其他节点接收准备好的消息集 $\{p_i\}$。当此节点的计时器超时并且收到超过 $2f+1$ 个 PREPARE 消息时，共识节点根据它们收到的信誉优先级 $\{p_i\}$ 决定最佳存储提供者。如果每一轮共识只返回最近的节点，由于距离对信誉计算机制的影

算法 5.1 共识过程

准备阶段

1: $request \leftarrow receive(REQUEST)$

2: **if** *request* 有效 **then**

3: $d_i \leftarrow distance(i.pos, request.pos)$ //计算节点 i 和请求节点之间的距离

4: $s_i \leftarrow i.space$ //获取节点 i 的剩余存储空间

5: 通过式 5-1计算 p_i

6: $broadcast(p_i, PREPARE)$ //广播 p_i

7: **end if**

预提交阶段

8: $\{p_i\} \leftarrow receive(p_i, PREPARE)$ //接收其他节点的 *PREPARE* 消息

9: **if** $count(PREPARE) > 2 * f + 1$ 且超时 **then**

10: $P' \leftarrow sort\{p_i\}$ //将 n 个节点根据信誉从高到低排序

11: $P \leftarrow \{\}$ //初始化结果集

12: **for** $p_i \in P'$ **do**

13: **if** p_i的计算过程可信 **then**

14: 增加 p_i 到 P 中 //验证节点选择是否可信

15: **end if**

16: **if** $count(P) > n$ **then**

17: *break* //最佳存储节点已满足要求

18: **end if**

19: **end for**

20: $broadcast(P, PRECOMMIT)$ //广播最佳存储节点

21: **end if**

提交阶段

22: $P \leftarrow receive(P, PRECOMMIT)$ //接收其他节点的 *PRECOMMIT* 消息

23: **if** $count(P) > f + 1$ **then**

24: $commit(P)$ //提交最佳存储节点

25: **end if**

响，一些较近节点的存储压力可能会非常高。为了平衡负载，共识节点将返回一组信誉最高的 n 节点，供网关随机选择，而不是信誉最高的节点。

为了防止恶意节点在**准备**阶段中的欺诈行为，在**预提交**阶段，共识节点的排序过程也需要对源节点的计算过程进行验证。每个共识节点首先会对所有节点的信誉进行排序。然后，共识节点会根据信誉由高到低的顺序，逐个验证节点的计算过程是否可信。如果发现某个节点的计算过程不可信，共识节点将忽略该节点的信誉，继续验证下一个节点。当共识节点收到的信誉最高的 n 个节点的信誉时，共识节点将这些节点返回给网关。

5.2.3 提交阶段

在**提交**阶段，所有节点都将收到其他节点推荐的最佳存储决策。当相同的预提交消息的数量超过 $f+1$ 时，此节点将向网关提交最佳存储节点集，由网关来决定最终的存储节点。由于该网络中最多存在 f 个恶意节点，因此在**提交**阶段，虚假信息的数量一定少于诚实信息，所以该系统并不会受到恶意节点的威胁。

5.3 共识过程的安全性分析

本文在这里考虑这一共识协议的安全性。由于 TimeChain 的共识协议只是基于 PBFT 添加了额外的信息，本文只考虑**准备**和**预提交**阶段额外信息带来的安全风险。

在**准备**阶段，如果一个节点伪造了自己的分数，网关可以很容易地检查分数的真实性，因为评估数据源都可以在链上找到。一旦节点伪造了其信誉，该行为也将被记录在链上，从而影响下一次的信誉评估。此外，由于最后只选择了一个存储节点，网关不关注所有节点得分的真实性，只关注所选节点的得分。

在**预提交**阶段，如果任何节点伪造了最终得分，则不会影响最终结果。这是因为对于包含 f 拜占庭节点的 $3f+1$ 节点的网络，必须在**提交**阶段获得相同的结果。

5.4 本章小结

在本章中，本文专注于 TimeChain 中存储节点选择模块。从前面的测试中，本文认识到，存储节点与网关之间的距离直接影响数据访问延迟，而存储节点的可靠性则关乎

数据的完整性和安全性。因此，本文提出了一种基于共识协议的节点选择机制，该机制综合考虑了节点间的距离和历史服务记录，以全面评估存储节点的优劣。

本文分析了现有系统如 Storj、CoopEdge 和 PipeEdge 的局限性，指出它们依赖于集中式的节点选择服务，存在单点故障的风险。为了解决这一问题，本文将节点选择过程与共识机制紧密结合，提出了一种新的节点选择算法。该算法包括请求、准备、预提交、提交和回复五个阶段，通过这一流程，本文能够在确保安全性的同时，选择出最佳的存储节点。

在安全性分析中，本文证明了所提出的共识协议在 PBFT 基础上增加的信息并未引入额外的安全风险，因此该机制是安全的。本文的共识协议考虑了准备阶段和预提交阶段的安全性，确保即使在存在恶意节点的情况下，系统也能正确选择出最佳的存储节点。

6 基于 LSH 树的验证机制

从本文的测量数据中可以分析出，在传输的数据中，数据完整性证明几乎占据了一半的比例。这些证明通常以默克尔树的形式组织，由一连串的哈希值构成。在默克尔树结构中，非叶子节点的哈希数量与原始数据点的数量大致相同。鉴于物联网数据单元的大小与哈希值相近，这意味着为了验证数据的完整性，需要传输的数据量几乎是原始数据量的两倍。在数据传输过程中，如何减少完整性证明大小，成为了一个亟待解决的难题。

6.1 LSH 树结构与工作流程

默克尔树（Merkle Tree）作为一种轻量级的完整性验证机制，在许多区块链平台中广泛使用。如 图 6.1所示，默克尔树是一种二叉树结构，其中每个非叶节点都是其两个子节点的哈希值。具体来说，叶节点通常包含底层交易的哈希值，而非叶节点则包含其左右子节点哈希值的组合哈希。树的根节点最终形成一个单一的哈希值，称为默克尔根（Merkle Root），这个哈希值将被存储在链上。通过这种方式，默克尔树能够将大量数据压缩成一个固定长度的哈希值，并允许高效地验证任何单个数据项是否属于该集合。

为了给出完整性证明，给定一个数据项，可以通过提供一条“默克尔路径”来证明该数据项属于某个默克尔树。这条路径包括从叶节点到根节点所需的所有兄弟节点哈希值，使得验证者能够在不获取整个数据集的情况下确认成员资格。例如，要证明交易 8 的有效性，数据存储方可以提供 $Hash7, Hash56, Hash1234$。如果最终发现 $hash(hash(hash(hash(\text{交易 } 8), Hash7), Hash56), Hash1234)$ 与记录在链上的 $Hash1-8$ 相符，则说明交易 8 确实在该交易集合中。

在大量数据存储的场景中，为了尽量减少重构默克尔树的计算开销，使数据存储方可以对用户的请求做出快速响应，这个默克尔树的非叶子节点也需要被存储在存储节点。然而，默克尔树的节点数量大，节点数据较长。如 图 6.1所示，对于 8 个交易，默克尔树需要生成 15 个非叶子节点的哈希值。由于物联网元数据较小，与哈希值长度基本相当，因此默克尔树的完整性证明较大，在验证阶段需要占据大量带宽，在存储方也

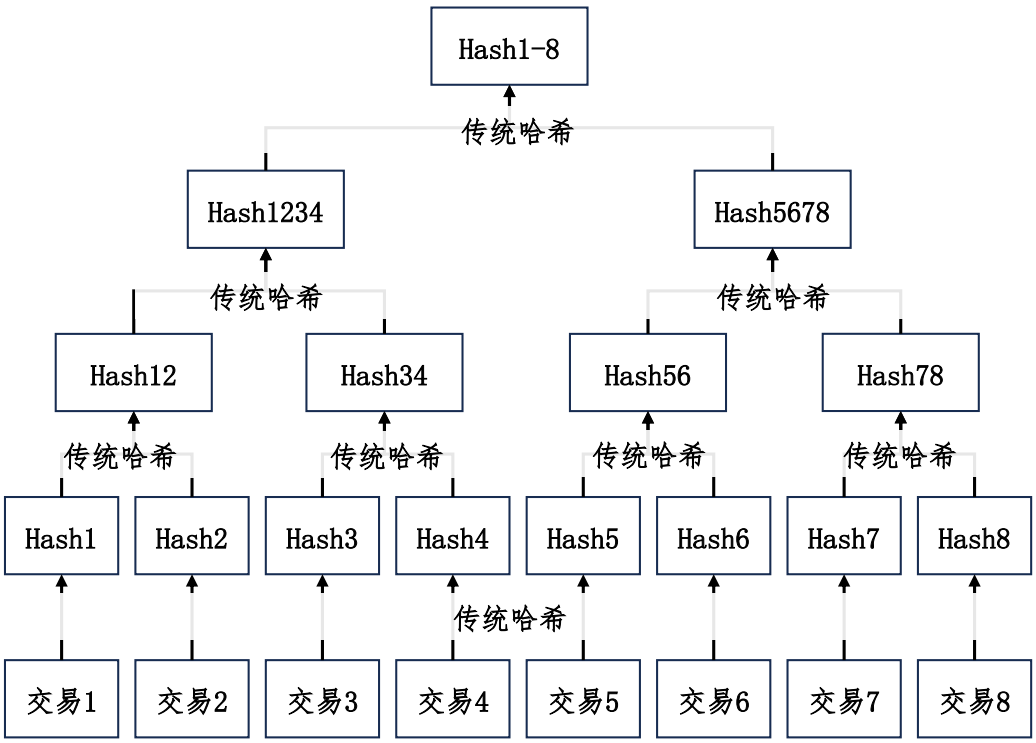


图 6.1 默克尔树的结构

会占用较多存储空间。

深入分析物联网数据的特性后，本文注意到物联网数据的变化通常较为缓慢，短期内不太可能发生剧烈变动。针对这些高度相似的数据点，LSH 算法能够从相似的原始数据中产生相似的哈希结果。LSH 算法的这一特性保证了即使在哈希处理之后，相似的物联网数据点仍然保持相似性。基于这一观察，本文设计了一种创新的基于 LSH 树的验证机制，它采用 LSH 算法替代了传统默克尔树中使用的通用哈希函数，如 图 6.2所示。通过仅传输 LSH 哈希值的差异部分，本文能够显著减少在验证过程中需要传输的数据量，从而提高了数据传输的效率。

本文在 图 6.3中展示了 LSH 树的一个具体例子。具体来说，对于批处理中的数据，本文采取类似于默克尔树的步骤，首先对原始数据执行局部敏感哈希。在叶子层，对于相似的原始数据 0...000 和 0...001，他们的哈希结果也说类似的，分别是 `d...eac` 和 `d...ead`。因此，在传输第一级哈希值时，TimeChain 只需要最后一位哈希值即可。针对这两个相似的哈希值，TimeChain 也对其做局部敏感哈希，得到的结果与另一组原始数据的哈希值相似，这也可以极大地减少需要传输的哈希位数。然后，TimeChain 逐层向上重复这个过程，直到得到一个唯一的哈希值，即哈希根。通过类比，对于每一层的哈

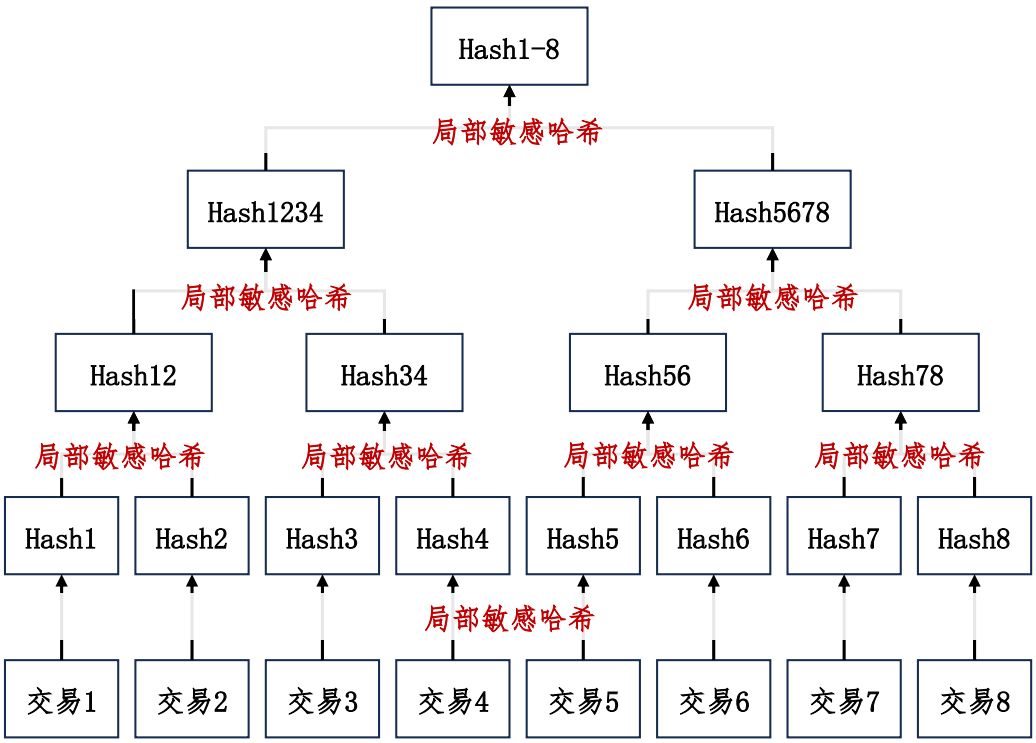


图 6.2 LSH 树的结构

希值，TimeChain 只需要传输差分的哈希比特，从而进一步减少了传输的数据量。

6.2 LSH 树的尾部合并机制

在一个完整的二叉树中，LSH 树可以通过成对地批量合并数据来执行局部敏感哈希。然而，如果批处理中的数据数量不足以形成完整的二叉树，那么像默克尔树一样构建哈希树将导致相似性的丧失。在 图 6.3中，一批中有 7 个数据点，对于一个完整的二叉树来说，这个大小是不满足的。在第一轮哈希中，前 6 个数据成对执行 LSH。由于原始数据的相似性，这 6 个数据的哈希结果是相似的。在第二轮哈希中，由于数据 5-6 和第 7 个原始数据的第一轮哈希结果非常不同，这两个数据的哈希结果也与数据点 1-4 的哈希结果非常不一样。在执行完整性证明时，需要传输所有这些不同的哈希数据位，这增加了传输的数据量。

为了解决这个问题，本文引入了尾部合并策略，将非全二叉树的尾部节点与同一层的前节点合并。如 图 6.4所示，在第一轮哈希中，本文将第 7 个节点和第 6 个节点合并，以尽可能保持数据的相似性。数据 6-7 的哈希结果是 `efcbeb`，数据 5-6 的哈希值是 `efcbef`。显然，在第一轮哈希中存在很高的相似性，并且可以保持到下一级哈希。这将

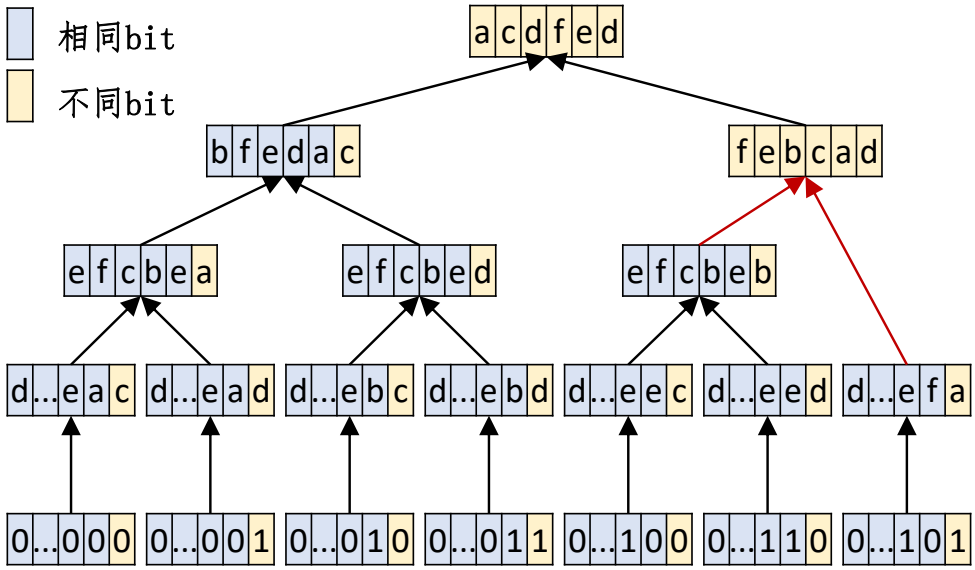


图 6.3 非满二叉 LSH 树

传输的哈希值大小从 12 位减少到 7 位，代价是在第一轮哈希中只传输了 1 个不同的比特。这样，在进行完整性证明时，本文只需要传输不同的哈希位，从而减少了传输的数据量。

6.3 LSH 树的安全性分析

鉴于本文提出的解决方案中采用了 LSH 树替代了传统的默克尔树，对 LSH 树的安全性进行深入分析变得至关重要。LSH 树的设计初衷是为了减少数据传输量，同时保持数据验证的准确性。然而，这种替代可能会引入新的安全风险。本文主要关注数据篡改问题，即在 LSH 树结构中，存储提供者是否能够通过改变数据内容来得到与未篡改数据相同的哈希值。为了评估这一风险，本文对 LSH 树的不同层级进行了广泛的测试，以确定在何种程度上篡改数据而不被检测到是可行的。

测试结果表明，在最接近数据源的哈希层，也就是 LSH 树的最底层，哈希值的平均差位数达到了 170 位。这一数值远超过了广泛使用的 MD5 和 SHA1 哈希算法的安全性标准，这些标准现在在物联网场景中非常常用[47-48]。因此，LSH 树在底层提供的 170 位的哈希差位数，为数据完整性提供了更强的安全保障。

对于 LSH 树靠近根节点的高层级，虽然哈希差位数较少，但这并不意味着篡改数据变得容易。实际上，由于 LSH 树的结构特性，即使是高层级的少量哈希差异也足以被检测出来，因为这些差异会随着树的结构向上传播，最终影响到根哈希值。因此，任

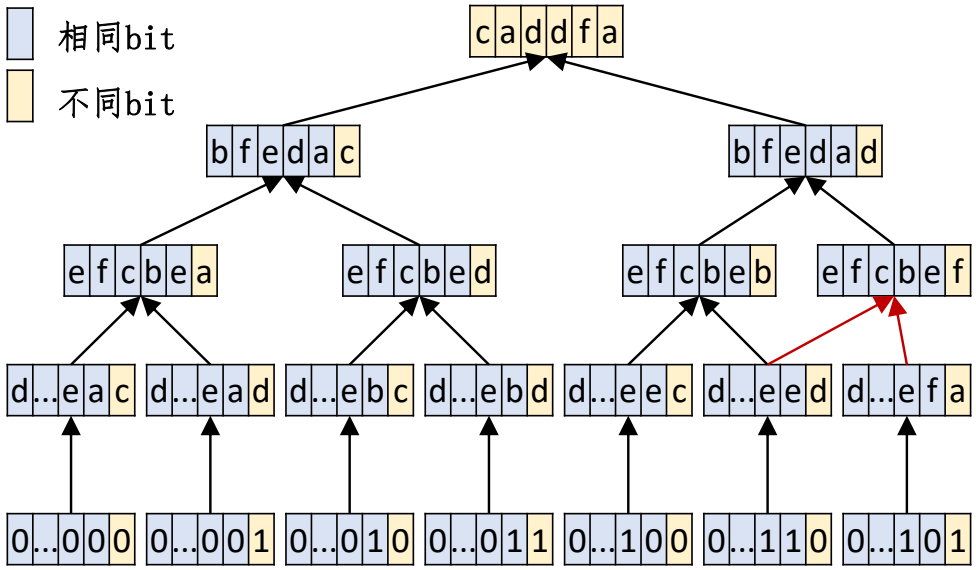


图 6.4 尾部合并下的非满二叉 LSH 树

何对原始数据的篡改，无论在树的哪一层，都很难不被察觉。

综上所述，LSH 树不仅提供了与传统默克尔树相似的数据验证功能，而且在安全性方面仍然符合物联网场景的需要。

6.4 本章小结

在本章中，本文针对物联网数据完整性验证过程中的数据传输延迟问题，提出了一种基于 LSH 树的新型验证机制。本文发现，在传输的数据中，数据完整性证明占据了相当大的比例，且通常以默克尔树的形式组织，这导致为了验证数据完整性需要传输的数据量几乎是原始数据量的两倍。为了解决这一问题，本文利用物联网数据变化缓慢且高度相似的特性，采用 LSH 算法替代了传统默克尔树中的通用哈希函数，通过仅传输 LSH 哈希值的差异部分，显著减少了验证过程中需要传输的数据量。

本文详细分析了 LSH 树的结构和工作流程，并针对非满二叉树的情况，引入了尾部合并策略以保持数据的相似性，进一步减少了传输的数据量。此外，本文还对 LSH 树的安全性进行了分析，确保了其在最接近数据源的哈希层中具有足够的安全性，超过了现有的 MD5 和 SHA1 标准，在物联网场景中足够安全。

7 实验结果与分析

在本章中，本文将对 TimeChain 系统进行全面的实验评估，从存储延迟和查询延迟两个关键维度来验证 TimeChain 设计的有效性。本文将分析 TimeChain 在不同查询负载和存储网络条件下的性能表现，并针对系统中的自适应聚合机制、基于共识的节点选择机制和基于 LSH 树的验证机制进行消融实验，以展示这些技术点对系统性能的具体提升。通过这些实验，本文旨在展示 TimeChain 相比于现有解决方案在区块链存储系统中的性能优势。

7.1 实验设置

本文基于一些开源项目（如 Hyperledger Fabric 和 IPFS）实现 TimeChain。本文使用阿里云的 50 台虚拟机构建了一个基于 Hyperledger Fabric 的底层区块链系统，每个节点配备 2 核 CPU 和 4GB 内存。块大小设置为 1500 个交易，块间隔为 1 秒。

本文基于一个真实世界的存储网络[37]，模拟了分布在世界各地的 320 个云服务器节点，并基于该集群进行了实验。每个存储节点配置 2 核 CPU 和 4GB 内存，每个存储节点的存储空间为 512GB。存储节点和网关之间的距离从 800 公里到 6000 公里不等，平均为 4000 公里。本文使用现有的线性回归模型[38]模拟存储节点和网关之间的传输延迟。考虑到一些存储提供商存在欺诈行为，部分远程存储的数据将有概率无法被访问到，概率为 60%。

本文使用电脑（Personal Computer，PC）作为物联网传感器的网关节点，它配备了 Intel(R)Core i7-13700K CPU@5.4GHz、32GB DRAM，并运行 Ubuntu 22.04。默认聚合存储单元大小和查询大小设置为 20。

7.1.1 基线

SEBDB[5]是链上存储的典型代表。它通过将所有数据存储在区块链上，并使用 B+树创建时间戳和设备名称的快速索引实现了对链上区块的高效访问。在数据验证方面，SEBDB 使用传统的默克尔树进行数据验证。默克尔树通过计算数据块的哈希值，并将

这些哈希值逐层组织成树结构，来实现数据完整性验证。尽管 SEBDB 不是专门为物联网数据设计的链下存储解决方案，但是因为它的数据验证机制类似于 TimeChain，本文将它作为对比方案。为了保证实验的公平性，本文调整了 SEBDB 来记录单个数据项的存储位置并将其上传到链中，并随机选择存储节点。这些修改确保了实验的公平性，同时更好地反映了物联网数据的特点和需求。

FileDES[36]是一个基于文件的链下存储系统。它通过在远程节点上存储文件并在链上记录文件的哈希值来实现文件的安全存储和可靠性。当客户端需要搜索文件时，FileDES 会遍历区块链上的所有块，以获取文件的存储位置。在数据验证方面，FileDES 也使用与 SEBDB 相同的默克尔树。由于 FileDES 是为文件存储设计的，为了确保实验的公平性，本文将 FileDES 改造成为适用于物联网数据的链下存储系统。对于到达网关的物联网数据，FileDES 将这些数据以固定大小、按照到达网关的顺序进行打包，并将这些数据上传到远程存储节点。

7.1.2 数据集和工作负载

本文使用了以下三个数据集进行实验：

港珠澳大桥（Bridge）[49]: Bridge 数据集包含 4M 条数据记录，这些数据是在 5 天内从港珠澳大桥上的 6 种传感器采集的。这 6 种传感器与桥梁健康监测有关，包括桥梁各个位置的振动加速度、挠度等，共计 53 个。由于数据生成频率为 10Hz，本文将平均数据查询范围设置为 40。

RT-IFTTT（RT）[50]: RT 数据集包含 680K 条数据记录。这些数据是在 10 天内从 10 个真实传感器采集的值，包括温度、湿度、可见光和其他传感器。数据以秒为单位收集，因此查询的平均范围设置为 20。

天气 (WX) [1]: WX 数据集包含 1.5M 条数据记录，其中包括来自五年内 30 多个城市的各种天气属性每小时测量的数据。鉴于数据收集频率相对较低，间隔最多一小时，本文假设查询范围为 10。

考虑到时间序列存储系统的存储特性[51]，本文将这些数据集的格式转换为 $<measurement, field\ name, field\ value, timetamp>$。*measurement* 类似于数据表，用于组织数据，例如“天气”；*field name* 用于标识不同的字段，例如“温度”；*field value* 存

[1]https://www.kaggle.com/selfishgene/historical-hourly-weather-data/

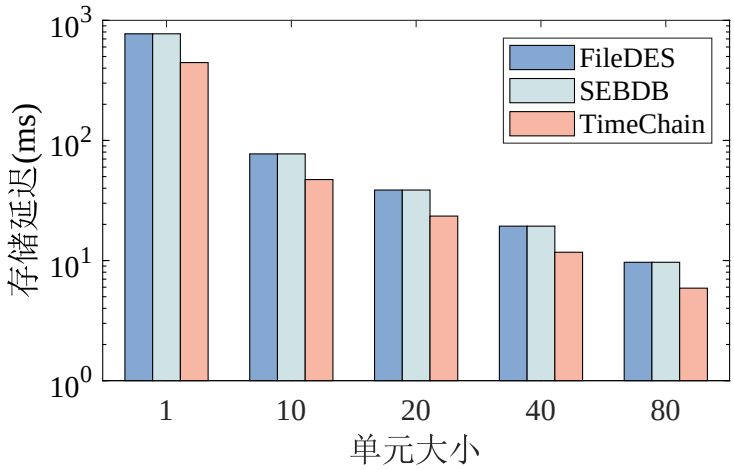


图 7.1 TimeChain 与基线的存储延迟对比

储的是实际数据，比如温度传感器的实际值；*timestamp* 是时间戳，记录了数据的时间信息。考虑到数据集的特点，本文将这些数据的大小分别固定为 8B、32B、8B、8B。因此，一条数据总共占用的存储空间为 56B。

7.2 总体性能评估

7.2.1 存储延迟

本文比较了不同聚合单元大小下的存储延迟。由于物联网设备数量众多，数据生成速度快，本文将存储延迟定义为存储 10000 个数据的总延迟，而不是关注单个数据的细微延迟。如 图 7.1所示，对于不同的聚合单元大小，TimeChain 的存储延迟低于 SEBDB 和 FileDES。这是因为 TimeChain 的独特打包机制和节点选择机制减少了数据传输延迟。

此外，随着聚合单元大小变大，存储延迟变小。这是因为，对于相同数量的数据，当聚合单元较大时，数据在链中打包和记录的次数会减少。然而，较大的聚合单元可能会引入较大的存储等待延迟。因为一个聚合单元可能需要等待较长时间才可以被填满，才能被打包和记录。具体的聚合单元大小需要根据实际数据产生速度和用户的需求来确定。用户可以通过调整网关的聚合单元大小来控制实际的存储延迟。

7.2.2 查询延迟

本文比较了不同聚合单元大小的查询延迟，如 图 7.2所示。查询延迟是指基于固定查询范围随机查询数据集的平均延迟，查询范围设置为 20。从 图 7.2可以看出，TimeChain

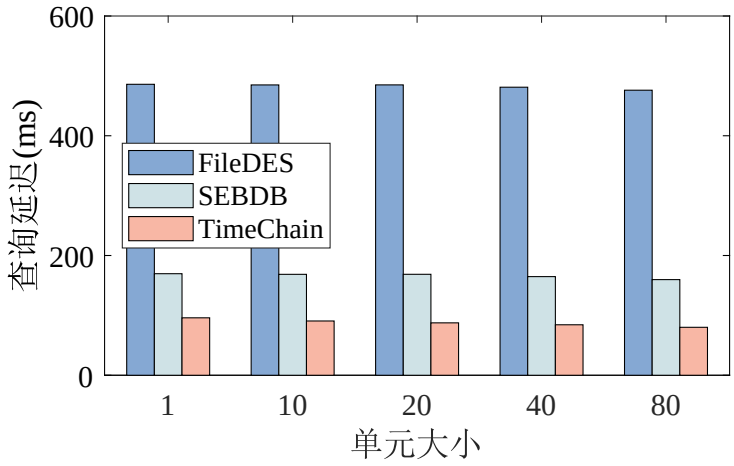


图 7.2 TimeChain 与基线的查询延迟对比

的查询延迟低于其他两种方案。这是由于 TimeChain 减少了数据传输延迟，原因将在 图 7.3中解释。

本文还发现，当聚合单元大小增加时，查询延迟会减少。这是因为当聚合单元大小增加时，同一查询中涉及的聚合单元数量会减少，用户的查询结果将大概率集中在一个存储节点上。然而，聚合单元越大，TimeChain 相对于其他解决方案的改进将随着聚合单元的增大而减少。这是因为当聚合单元非常大时，相当于将所有数据存储在一个聚合单元中。在这种情况下，对于数据进行谱聚类切分不会导致性能提高，反而会导致对多种范围查询的灵活性降低。此外，当大量数据集中在存储节点中时，存储系统的可扩展性和可靠性也会受到损害。

7.2.3 查询延迟分解

本文进一步分析了查询延迟的细分,以确定影响整体查询性能的主要因素,如 图 7.3所示。查询的延迟主要由 4 个阶段组成，检索、传输、验证和返回。考虑到传感器通常选择更近的网关，返回阶段的延迟几乎可以忽略不计。在验证阶段，三种方案的延迟相对接近，小于 1ms，也可以被忽略。

在传输和检索阶段，本文发现延迟占据了查询延迟的主要部分。特别是 TimeChain 的传输延迟明显低于 FileDES 和 SEBDB。这一优势归功于 TimeChain 的节点打包机制，该机制优化了数据的组织方式，减少了从存储节点获取数据的次数。同时，TimeChain 的节点选择机制通过选择地理位置更近的存储提供商进一步缩短了数据传输的距离，从而降低了传输延迟。

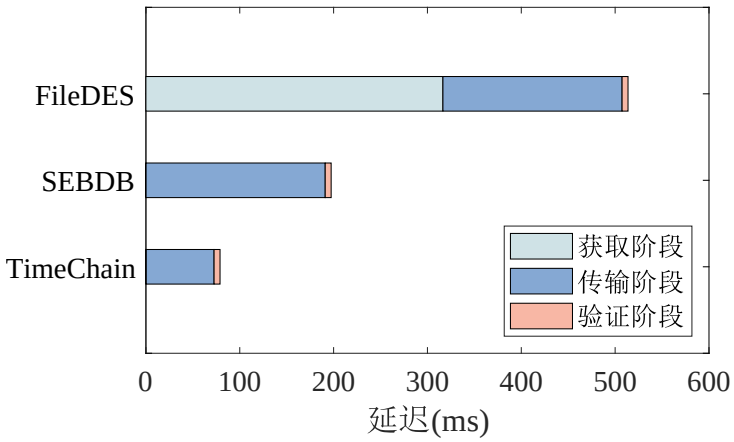


图 7.3 查询延迟分解对比

在检索阶段，FileDES 由于需要遍历区块链上的所有块来检索数据，导致了较高的检索延迟。这种机制在面对大量数据时效率较低，尤其是在数据量迅速增长的场景下。相比之下，SEBDB 和 TimeChain 通过使用 B+ 树和 R 树索引结构，显著提高了检索效率，降低了延迟。这两种索引结构允许系统快速定位到存储数据的位置，从而加快了检索速度。

7.2.4 支持的最大存储设备数量

在本小节中，本文使用的支持设备最大数量指标，即存储系统每秒可以提供存储服务的设备数量。本文假设网关可以同时处理来自多个物联网设备的数据存储请求，并忽略网关本身的处理延迟。所有物联网设备同时以 1Hz 的频率生成数据，并要求在生成下一个数据之前必须存储这些数据。在这样的限制下，存储系统可以并行支持的最多设备数量即为支持的最大存储设备数量。

如 图 7.4所示，TimeChain 支持最大设备数量分别是 SEBDB 和 FileDES 的 1.63 倍和 3.55 倍。这主要是由于 TimeChain 对数据传输延迟采取的优化技术，较小的数据传输延迟允许 TimeChain 以更快的速度存储数据。此外，对于所有测试的方案而言，随着聚合单元大小的增加，支持的最大设备数量也随之增加。这是因为较大的聚合单元意味着单个链上哈希可以代表更多的数据量，从而提高了单个存储单元的效率，使得系统能够支持更多的物联网设备。

特别地，当聚合单元大小达到 80 时，TimeChain 支持的最大设备数已经达到了千级，这一结果充分展示了 TimeChain 在大规模物联网环境中的扩展能力和高效性。这一

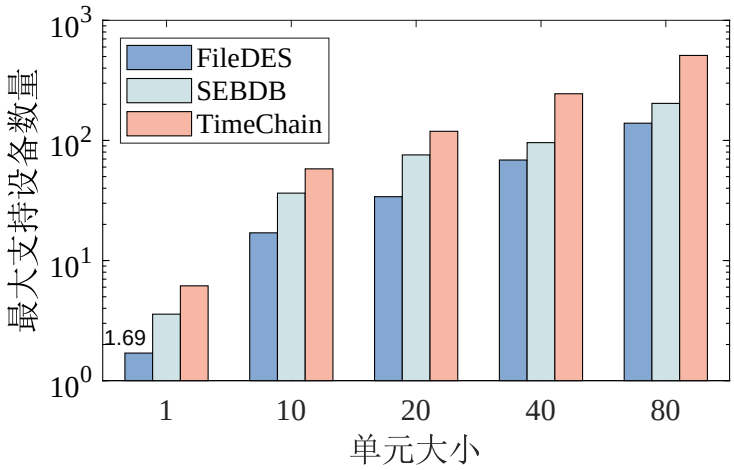


图 7.4 最大支持存储设备数

性能优势不仅证明了 TimeChain 在处理大规模数据时的可靠性，也为未来物联网应用的发展提供了强有力的技术支持。

7.3 性能影响因素评估

7.3.1 不同查询大小下的查询延迟

本文比较了三种不同查询大小数据集（Bridge、RT 和 WX）的查询性能，结果如图 7.5所示。三个数据集中工作负载的平均查询大小分别为 40、20 和 10。本文可以发现，这三种解决方案的查询延迟通常随着查询大小的增加而增加，即 WX 最小，RT 次之，Bridge 最大。这是因为更大的查询大小涉及了更多的聚合单元，因此可能需要去更多的存储节点获取数据，从而增加了数据传输延迟。

本文观察到，与 SEBDB 和 FileDES 相比，当查询大小变大时，TimeChain 的聚类算法带来的性能改进也会增加。这是因为当查询大小变大时，SEBDB 和 FileDES 通常需要从比 TimeChain 更多的节点获取数据，而 TimeChain 可以通过聚类算法减少数据获取的次数，从而减少数据传输延迟。

7.3.2 不同存储网络规模下的存储延迟

本文比较了不同存储网络规模下每种解决方案的存储延迟，如图 7.6所示。随着存储节点数量的增加，TimeChain 的存储延迟呈下降趋势。这一现象的原因在于，TimeChain 的网关在选择存储提供商时拥有更大的灵活性，能够从更多的节点中选择，从而增加了

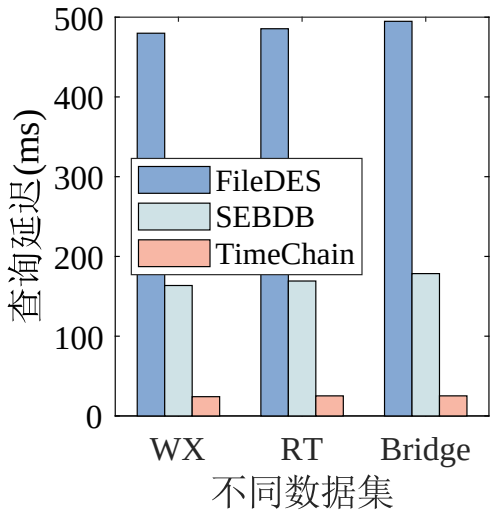


图 7.5 不同查询大小下的查询延迟

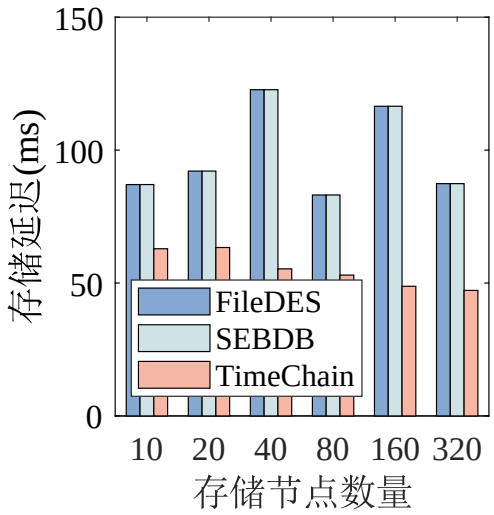


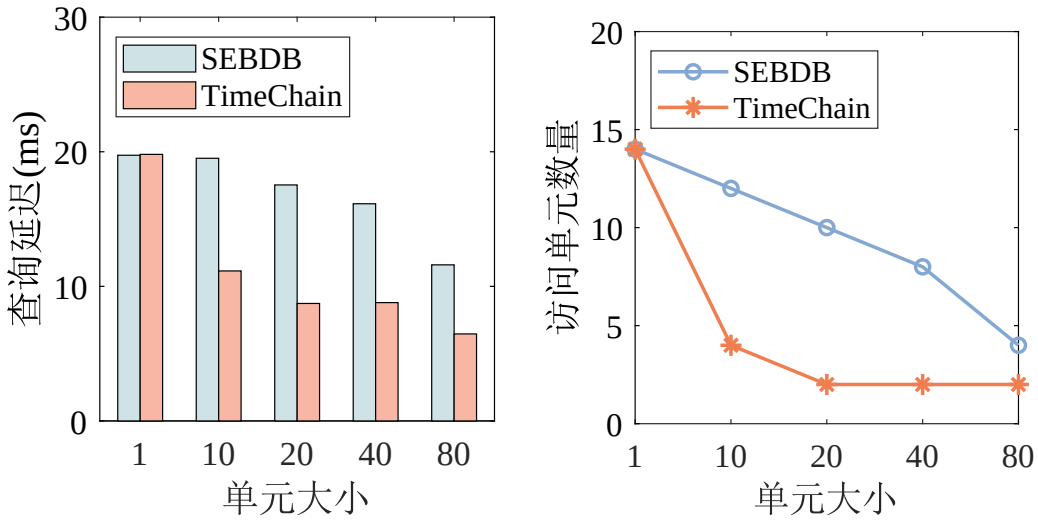
图 7.6 不同存储网络下的存储延迟

选择到地理位置更接近的存储节点的机会，这自然降低了数据传输的时间。相比之下，FileDES 和 SEBDB 由于采用了随机选择存储节点的机制，它们在存储节点数量增加时，并没有表现出存储延迟的显著变化。这种随机性导致了这两种方案在存储延迟上缺乏可预测性，且在不同存储网络规模下，它们的性能波动较大。由于 FileDES 和 SEBDB 在选择存储节点时没有考虑节点的具体位置信息，因此它们无法充分利用增加的节点来优化存储延迟。这就导致了即使在存储节点数量增加的情况下，这两种方案的存储延迟仍然保持在一个相对较高的水平，且波动较大，这在一定程度上限制了它们在大规模存储网络中的应用潜力。

7.4 消融实验

7.4.1 聚类算法

本文比较了聚类算法的效果，并将 TimeChain 与 SEBDB 进行了比较。如图 7.7a所示，与 SEBDB 相比，TimeChain 的网络传输延迟减少了 40.3%。这是因为 SEBDB 的打包节点未能充分考虑用户查询的规律性，只是将物联网数据根据到达网关的顺序进行打包。这导致在处理数据拥有者请求的数据时，SEBDB 不得不访问多个存储提供商所管理的聚合单元，增加了数据检索的复杂性和延迟。相比之下，TimeChain 采用了谱聚类算法，这种算法能够根据用户请求的特征对来自特定传感器的数据进行打包。这种方法不仅优化了数据在存储节点中的分布，还减少了数据访问时需要跨越的聚合单元数量。如



(a) 查询延迟 (b) 跨越的 batch

图 7.7 聚类算法消融实验

图 7.7b所示，TimeChain 访问的聚合单元数量比 SEBDB 少了 59.3%，这表明 TimeChain 在减少数据访问延迟和提高查询效率方面具有明显优势。

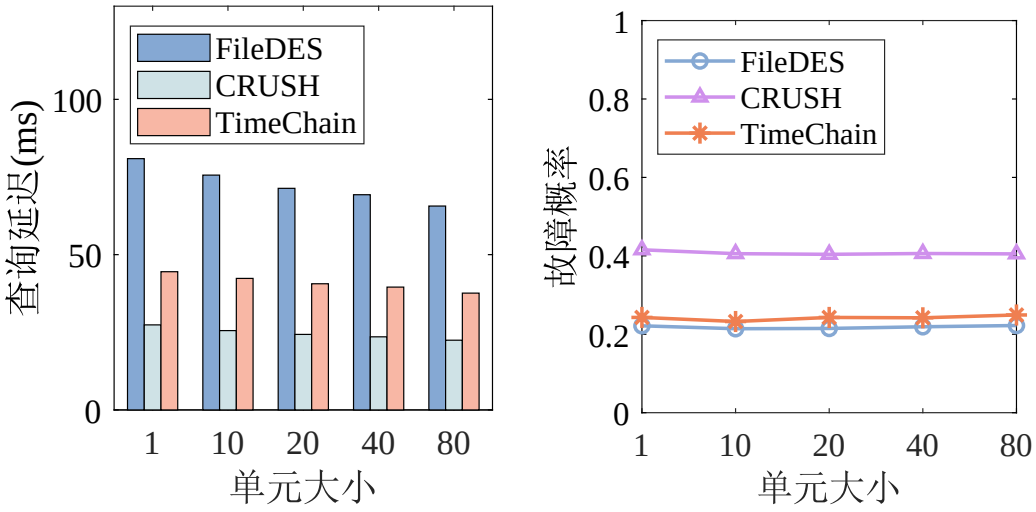
7.4.2 节点选择

本文在图 7.8a和图 7.8b中显示了 TimeChain、FileDES[36]和 CRUSH[28]在节点选择方面的性能差异。

FileDES 通过基于节点信誉的筛选机制，确定了一组受信任的存储节点，并在此基础上随机选择节点进行数据存储。这种方法的优点在于提高了数据存储的安全性，因为只有信誉良好的节点才被考虑用于存储。然而，由于缺乏对节点地理位置的考虑，FileDES 的随机选择可能导致选择了距离较远的节点，从而增加了数据传输的延迟，影响了整体性能。

CRUSH 则采取了一种基于物理位置的策略，优先选择距离最近的节点进行数据存储。这种策略在不考虑节点故障和可靠性的情况下，能够显著减少数据传输的时间。然而，CRUSH 方案的一个主要缺陷是它没有充分考虑节点的可靠性，如图 7.8b所示，这导致有 41% 的节点可能无法提供有效的存储服务，这对于需要高可靠性的存储系统来说是一个严重的缺陷。

与 FileDES 和 CRUSH 相比，TimeChain 在节点选择上采取了一种综合考虑节点物理距离和信誉的策略。TimeChain 不仅考虑了节点的地理位置以减少传输延迟，还考虑



(a) 查询延迟 (b) 存储服务质量

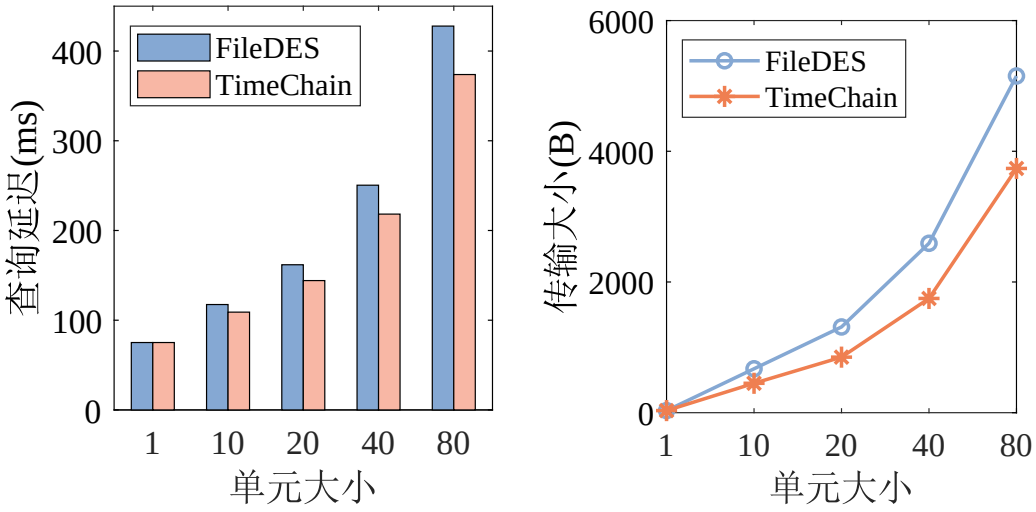
图 7.8 节点选择消融实验

了节点的历史服务记录和信誉，以确保所选节点的可靠性。这种综合考虑的方法使得 TimeChain 在响应时间和服务概率方面都展现出了最佳性能。尽管 TimeChain 选择的节点可能不是物理距离上最近的，但通过平衡节点的距离和服务质量，TimeChain 能够提供更稳定和高效的存储服务。

7.4.3 LSH 树

本文比较了 FileDES 和 TimeChain 的网络传输延迟，如图 7.9a所示。与 FileDES 相比，TimeChain 的数据传输延迟减少了 10.9%。这主要是来源于 TimeChain 对数据传输量的优化。在物联网中，局域网内传感器设备的数量众多，且这些设备通常以高频率持续生成数据。而且，TimeChain 相比于 FileDES 的传输优化随着数据量的增加而增加。这是因为当数据量增加时，LSH 树中的相似数据量也会增加，从而减少了传输的数据量，降低了网络传输延迟。

如图 7.9b所示，由于 TimeChain 使用 LSH 作为哈希算法，通过网络传输的数据量大大减少。这种数据量的减少不仅降低了存储和查询操作的延迟，还显著减轻了存储提供商的存储负担。对于存储提供商而言，这意味着更低的存储成本和更高的数据处理效率。同时，对于整个存储系统来说，减少了数据传输量还有助于提高系统的吞吐量和响应速度，尤其是在高负载或资源受限的环境中。



(a) 查询延迟 (b) 完整性证明大小

图 7.9 LSH 树消融实验

7.5 本章小结

在本章中，本文对 TimeChain 系统进行了全面的实验评估，以验证其在存储性能方面的优势。通过在模拟的全球存储网络上进行测试，本文比较了 TimeChain 与现有解决方案，SEBDB 和 FileDES，在存储延迟和查询延迟方面的表现。实验结果表明，TimeChain 在减少查询延迟和存储延迟方面具有显著优势，平均查询延迟降低了 64.6%，存储延迟降低了 35.3%。

本文还分析了不同查询负载和存储网络规模对 TimeChain 性能的影响，发现随着聚合单元大小的增加，TimeChain 的存储和查询性能得到了进一步的提升。此外，TimeChain 在支持最大存储设备数量方面也展现出了其可扩展性，与 SEBDB 和 FileDES 相比，分别增加了 1.63 倍和 3.55 倍。

为了进一步验证 TimeChain 设计中关键技术点的有效性，本文进行了消融实验。实验结果证实了自适应聚合机制、基于共识的节点选择机制和基于 LSH 树的验证机制在提升系统性能方面的重要作用。特别是 LSH 树机制，通过减少传输的数据量，显著降低了网络传输延迟。

8 总结与展望

8.1 总结研究

物联网技术的迅猛发展带来了数据量的爆炸性增长，尤其是时序数据，这对存储系统提出了前所未有的挑战。传统的集中式存储解决方案，尽管在管理和操作上相对简单，但它们存在明显的单点故障风险，一旦中心服务器发生故障，整个数据存储和处理流程可能会中断。尽管分布式存储系统可以解决单点故障的问题，但是这些系统在数据安全性和不可篡改性方面仍然存在不足，特别是在需要高度透明度和安全性的应用场景中。目前，区块链技术作为一种新兴的分布式存储技术，由于其去中心化、不可篡改和高度透明的特性，为物联网数据的安全存储和可信传输提供了新的解决方案。然而，现有的区块链存储系统在处理物联网时序数据时仍然存在性能上的挑战，如存储延迟高、查询效率低、数据传输量大等。

针对这些挑战，本文提出了 TimeChain，一种专为物联网时序数据设计的高效区块链存储技术。TimeChain 通过自适应聚合机制、基于共识的节点选择机制以及基于局部敏感哈希树的数据完整性验证机制，显著提升了存储效率和查询速度，同时确保了数据的安全性和完整性。TimeChain 的设计不仅解决了传统存储系统的局限性，也为物联网数据的高效可信存储提供了新的解决方案。

本文的主要研究工作可以概述如下：

1. 在**数据聚合**阶段，本研究提出了一个针对链下存储环境的自适应聚合策略。为了降低在执行范围查询时需要访问的存储节点数目，本文构建了一个自适应无向加权图，将数据批处理问题转化为图的分割问题。鉴于物联网数据查询的不规则性，传统的聚类方法如 K-means 和 GMM 并不适用，因其或需预设图的形状和数量，或计算复杂度过高。因此，本文选择了谱聚类算法来处理子图的划分，以减少在聚合查询过程中访问节点的次数。

2. 在**存储节点选择**阶段，本文提出了一个依托共识机制的存储节点选择方案。在此过程中，本文重点关注了选择的准确性和安全性。准确性涉及选择的节点应具有较短的传输延迟和较高的存储服务质量，而安全性则意味着选择过程应能抵御单

点故障或恶意节点的干扰。TimeChain 在此方面综合考虑了节点的信誉度、可用存储空间和传输距离。同时，为了在去中心化环境中高效地进行节点选择，本文将共识机制与节点选择过程相结合，以降低传播延迟。

3. 在**数据验证**阶段，本文设计了一种基于局部敏感哈希（LSH）树的数据完整性验证机制。传统的默克尔树在构建过程中会产生与数据量相等的哈希值，导致传输成本过高。针对这一问题，本文的 LSH 树机制利用物联网数据中时间序列点的相似性，仅传输非冗余部分，从而显著降低了验证所需的数据量。

基于开源组件实现的 TimeChain，在性能评估中显示出了卓越的性能。与现有的区块链存储系统相比，TimeChain 平均减少了 64.6% 的查询延迟和 35.3% 的存储延迟，证明了其在提升物联网时序数据处理效率方面的明显优势。此外，TimeChain 在支持的最大存储设备数量方面也显示出了优秀的扩展性，与 SEBDB 和 FileDES 相比，分别提升了 1.63 倍和 3.55 倍。这表明 TimeChain 能够适应日益增长的物联网设备数量，满足未来物联网应用的发展需求。通过消融实验，本文进一步证实了 TimeChain 中自适应聚合机制、基于共识的节点选择机制和基于 LSH 树的验证机制对于系统性能提升的关键作用。

8.2 未来工作

在未来的工作中，本研究计划针对 TimeChain 系统进行以下几方面的研究和改进，以进一步提升系统性能并适应不同的应用场景。

目前，TimeChain 的测试和评估工作仅在模拟环境中进行，这限制了对其在现实世界条件下性能的全面理解。未来的工作将从模拟环境转向实际应用场景，以全面评估 TimeChain 的性能和可靠性。这意味着 TimeChain 将被部署在真实环境中，从而验证其在现实世界条件下的实际表现，确保其能够满足各种应用场景的需求。这包括在不同网络条件下对 TimeChain 算法的响应时间和吞吐量进行详细评估，以确保其在实际应用中的效率和稳定性。通过这种方式，可以更准确地理解 TimeChain 在面对真实网络波动和数据流量时的行为，从而为进一步的优化和改进提供实际依据。

其次，针对 PBFT 算法在高吞吐量场景下可能遇到的性能瓶颈，本文计划研究并引入更轻量级或异步的共识算法，例如 Raft 和 Tendermint。这些算法被选中是因为它们

在保持安全性的同时，能够有效减少通信开销，提升系统的响应速度和处理能力。这对于资源受限的物联网设备尤为重要，因为它们往往对能耗和计算能力有严格的限制。此外，未来的研究将探索分片技术和侧链技术等其他提升区块链性能的方法，以增强TimeChain的可扩展性，并深入分析这些技术可能引入的安全风险，制定相应的对策。

最后，由于在TimeChain中引入了新共识算法，TimeChain的安全性和稳定性将成为本研究关注的焦点。在未来的工作中，本研究将对所提出的共识协议和基于LSH树的数据完整性验证机制进行严格的实验验证。这包括在模拟存在欺诈节点的物联网环境中进行实验，以评估TimeChain在面对恶意攻击时的表现，以及其保护数据完整性和用户隐私的能力。通过这些实验，本研究旨在确保TimeChain在提供高效存储解决方案的同时，也能保障数据的安全性和隐私性，满足物联网领域对高安全标准的严格要求。

综上所述，未来的工作将围绕TimeChain的实际部署、性能优化和安全性验证展开，以确保该系统能够在多样化的应用场景中提供可靠、高效和安全的数据存储服务。

参考文献

[1] HUNG M. Leading the iot, gartner insights on how to lead in a connected world[J]. Gartner Research, 2017, 1: 1-5.

[2] AL-SARAWI S, ANBAR M, ABDULLAH R, et al. Internet of things market analysis forecasts, 2020–2030[C]//2020 Fourth World Conference on smart trends in systems, security and sustainability (WorldS4). 2020: 449-453.

[3] GILL P, JAIN N, NAGAPPAN N. Understanding network failures in data centers: measurement, analysis, and implications[C]//Proceedings of the ACM SIGCOMM 2011 Conference. 2011: 350-361.

[4] DORRI A, KANHERE S S, JURDAK R. Towards an optimized blockchain for IoT[C]//Proceedings of the second international conference on Internet-of-Things design and implementation. 2017: 173-178.

[5] ZHU Y, ZHANG Z, JIN C, et al. SEBDB: Semantics empowered blockchain database[C]//2019 IEEE 35th international conference on data engineering (ICDE). 2019: 1820-1831.

[6] XU C, ZHANG C, XU J. vchain: Enabling verifiable boolean range queries over blockchain databases [C]//Proceedings of the 2019 international conference on management of data. 2019: 141-158.

[7] WANG H, XU C, ZHANG C, et al. vchain+: Optimizing verifiable blockchain boolean range queries [C]//2022 IEEE 38th International Conference on Data Engineering (ICDE). 2022: 1927-1940.

[8] LI C, BEILLAHI S M, YANG G, et al. {LVMT}: An efficient authenticated storage for blockchain [C]//17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23). 2023: 135-153.

[9] ZHANG C, XU C, HU H, et al. {COLE}: A Column-based Learned Storage for Blockchain Systems [C]//22nd USENIX Conference on File and Storage Technologies (FAST 24). 2024: 329-345.

[10] ZAMANI M, MOVAHEDI M, RAYKOVA M. Rapidchain: Scaling blockchain via full sharding[C]//Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. 2018: 931-948.

[11] HONG Z, GUO S, ZHOU E, et al. Gridb: scaling blockchain database via sharding and off-chain cross-shard mechanism[J]. Proceedings of the VLDB Endowment, 2023, 16(7): 1685-1698.

[12] EL-HINDI M, BINNIG C, ARASU A, et al. BlockchainDB: A shared database on blockchains[J]. Proceedings of the VLDB Endowment, 2019, 12(11): 1597-1609.

[13] LABS S. Storj: A decentralized cloud storage network framework[Z]. 2018.

[14] MCCONAGHY T, MARQUES R, MÜLLER A, et al. Bigchaindb: a scalable blockchain database[J]. white paper, BigChainDB, 2016: 53-72.

[15] Sia[Z]. https://sia.tech/. Accessed: [2024.4].

[16] Hyperledger Fabric Documentation[Z]. https://hyperledger-fabric.readthedocs.io/en/release-1.4/. Accessed: [2019.7].

[17] BENET J. Ipfs-content addressed, versioned, p2p file system[J]. arXiv preprint arXiv:1407.3561, 2014.

[18] KANUNGO T, MOUNT D M, NETANYAHU N S, et al. An efficient k-means clustering algorithm: Analysis and implementation[J]. IEEE transactions on pattern analysis and machine intelligence, 2002, 24(7): 881-892.

[19] HE X, CAI D, SHAO Y, et al. Laplacian regularized Gaussian mixture model for data clustering[J]. IEEE transactions on knowledge and data engineering, 2010, 23(9): 1406-1418.

[20] AWS IoT[Z]. https://aws.amazon.com/cn/iot/. Accessed: [2024.7].

[21] TDengine. TDengine[Z]. https://www.taosdata.com/. 2023.

[22] MADDEN S R, FRANKLIN M J, HELLERSTEIN J M, et al. TinyDB: an acquisitional query processing system for sensor networks[J]. ACM Transactions on database systems (TODS), 2005, 30(1): 122-173.

[23] LAKSHMAN A, MALIK P. Cassandra: a decentralized structured storage system[J]. ACM SIGOPS operating systems review, 2010, 44(2): 35-40.

[24] DECANDIA G, HASTORUN D, JAMPANI M, et al. Dynamo: Amazon's highly available key-value

store[J]. ACM SIGOPS operating systems review, 2007, 41(6): 205-220.
[25] CHANG F, DEAN J, GHEMAWAT S, et al. Bigtable: A distributed storage system for structured data [J]. ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 1-26.
[26] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google’s globally distributed database[J]. ACM Transactions on Computer Systems (TOCS), 2013, 31(3): 1-22.
[27] CERVIN A, HENRIKSSON D, OHLIN M. TrueTime 2.0–reference manual[J]. Department of Automatic Control, Lund University, 2016.
[28] WEIL S, BRANDT S A, MILLER E L, et al. Ceph: A scalable, high-performance distributed file system[C]//Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI’06). 2006: 307-320.
[29] TAFT R, SHARIF I, MATEI A, et al. Cockroachdb: The resilient geo-distributed sql database[C]//Proceedings of the 2020 ACM SIGMOD international conference on management of data. 2020: 1493-1509.
[30] ZHOU E, HONG Z, XIAO Y, et al. MSTDB: a hybrid storage-empowered scalable semantic blockchain database[J]. IEEE Transactions on Knowledge and Data Engineering, 2022.
[31] PENG Y, DU M, LI F, et al. FalconDB: Blockchain-based collaborative database[C]//Proceedings of the 2020 ACM SIGMOD international conference on management of data. 2020: 637-652.
[32] CHEN F, WANG J, JIANG C, et al. Blockchain based non-repudiable IoT data trading: Simpler, faster, and cheaper[C]//IEEE INFOCOM 2022-IEEE Conference on Computer Communications. 2022: 1958-1967.
[33] XU C, ZHANG C, XU J, et al. SlimChain: Scaling blockchain transactions through off-chain storage and parallel processing[J]. Proceedings of the VLDB Endowment, 2021, 14(11): 2314-2326.
[34] BAUER D P. Filecoin[G]//Getting Started with Ethereum: A Step-by-Step Guide to Becoming a Blockchain Developer. Springer, 2022: 97-101.
[35] VORICK D, CHAMPINE L. Sia: Simple decentralized storage[J]. Retrieved May, 2014, 8: 2018.
[36] XU M, ZHANG J, GUO H, et al. FileDES: A Secure, Scalable and Succinct Decentralized Encrypted Storage Network[C]//IEEE INFOCOM 2024-IEEE Conference on Computer Communications. 2024: 1-10.
[37] CORNEO L, EDER M, MOHAN N, et al. Surrounded by the clouds: A comprehensive cloud reachability study[C]//Proceedings of the Web Conference 2021. 2021: 295-304.
[38] ZIVIANI A, FDIDA S, DE REZENDE J F, et al. Improving the accuracy of measurement-based geographic location of Internet hosts[J]. Computer Networks, 2005, 47(4): 503-523.
[39] CAESAR H, BANKITI V, LANG A H, et al. nuscenes: A multimodal dataset for autonomous driving [C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020: 11621-11631.
[40] BHATIA M, AHANGER T A, MANOCHA A. Artificial intelligence based real-time earthquake prediction[J]. Engineering Applications of Artificial Intelligence, 2023, 120: 105856.
[41] BARATA M, BERNARDINO J, FURTADO P. Ycsb and tpc-h: Big data and decision support benchmarks[C]//2014 IEEE International Congress on Big Data. 2014: 800-801.
[42] XU R, WUNSCH D. Survey of clustering algorithms[J]. IEEE Transactions on neural networks, 2005, 16(3): 645-678.
[43] YUAN L, HE Q, TAN S, et al. Coopedge: A decentralized blockchain-based platform for cooperative edge computing[C]//Proceedings of the Web Conference 2021. 2021: 2245-2257.
[44] YUAN L, HE Q, CHEN F, et al. PipeEdge: A trusted pipelining collaborative edge training based on blockchain[C]//Proceedings of the ACM Web Conference 2023. 2023: 3033-3043.
[45] LI W, FENG C, ZHANG L, et al. A scalable multi-layer PBFT consensus for blockchain[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 32(5): 1146-1160.
[46] VON LUXBURG U. A tutorial on spectral clustering[J]. Statistics and computing, 2007, 17: 395-416.
[47] CHI L, ZHU X. Hashing techniques: A survey and taxonomy[J]. ACM Computing Surveys (Csur), 2017, 50(1): 1-36.
[48] LANDGE I A, SATOPAY H. Secured IoT through hashing using MD5[C]//2018 fourth international conference on advances in electrical, electronics, information, communication and bio-informatics (AEEICB). 2018: 1-5.

[49] ZHANG W, GUO C, GAO Y, et al. Edge Cloud Collaborative Stream Computing for Real-Time Structural Health Monitoring[J]. arXiv preprint arXiv:2310.07130, 2023.
[50] HEO S, SONG S, KIM J, et al. Rt-ifttt: Real-time iot framework with trigger condition-aware flexible polling intervals[C]//2017 IEEE Real-Time Systems Symposium (RTSS). 2017: 266-276.
[51] NAQVI S N Z, YFANTIDOU S, ZIMÁNYI E. Time series databases and influxdb[J]. Studienarbeit, Université Libre de Bruxelles, 2017, 12: 1-44.