

University of Nevada, Reno

**Resource-efficient Blockchains for Reliable Data
Management in Large-scale Distributed Systems**

A dissertation submitted in partial fulfillment
of the requirements for the degree of Doctor of
Philosophy in Computer Science and Engineering

by

Abdullah Al-Mamun

Dr. Dongfang Zhao, Dissertation Advisor

August 2022

© by Abdullah Al-Mamun 2022
All Rights Reserved



The Graduate School

We recommend that the dissertation prepared under our supervision by

Abdullah Al-Mamun

entitled

**Resource-efficient Blockchains for Reliable Data Management in Large-scale
Distributed Systems**

be accepted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Dongfang Zhao, Ph.D., Advisor

Feng Yan, Ph.D., Committee Member

Lei Yang, Ph.D., Committee Member

Tin Nguyen, Ph.D., Committee Member

Yan Wang, Ph.D., Graduate School Representative

Markus Kimmelmeier, Ph.D., Dean, Graduate School

August 2022

Abstract

Nowadays, reliable data management is an essential task due to the explosion of data. The volume of data is increasing rapidly in each paradigm such as scientific computing and edge computing, etc. Consequently, reliable data management such as dependable distributed resiliency, provenance tracking, and data fidelity become paramount in high-performance computing (HPC) systems. Besides, trustworthy data management in edge computing poses new technical security challenges. Although blockchain-based decentralized protocols exhibit a lot of potential for reliable data management in such paradigms, adopting the blockchain in its current model both in high-performance computing (HPC) and in edge computing ecosystems comes with a vast array of challenges. The shortcomings of the current model of blockchain lead to many open questions in terms of adaptability, scalability, privacy, and usability. To overcome the challenges lying with the state-of-the-art solutions, this dissertation aims to investigate numerous techniques to adopt blockchain in HPC and edge computing ecosystems. This dissertation takes a holistic view of the various infrastructure gaps and presents scalable approaches to provide reliable data management through resource-efficient blockchain protocols.

This dissertation mainly consists of two parts. The first part investigates techniques on how to adopt blockchain-like decentralized protocols to ensure reliable distributed data management in the HPC system. More specifically, first, we introduce our work on blockchain that incorporates lightweight protocol to adopt blockchain in the HPC infrastructure to guarantee data fidelity for scientific provenance. Second, we introduce a series of protocols on blockchain that bridge the HPC-blockchain gap with two key components: (i) Lightweight consensus protocols for the HPC's shared-storage architecture, and (ii) A new fault-tolerant mechanism compensating for the MPI to guarantee consistency.

The second part studies trustworthy data management in edge computing infrastructure. We propose a resource-efficient mechanism to adopt blockchain seamlessly in an edge computing infrastructure to prevent data manipulation and allow fair data sharing with quick recovery under resource constraints of limited storage, computing, and network capacity. We also focus on an edge computing use case: secure data management in electronic supply chain ecosystem. We propose a framework with the following novelties: (i) it leverages uniquely hashed physical unclonable function (PUF) challenge-response-pairs (CRPs) to manage the security of sensitive PUF data; (ii) it allows most transactions to be processed in parallel by distributing the workload among the participating blockchain nodes to maximize system efficiency; and (iii) it employs PUF-based keyless encryption to protect users' privacy from the blockchain system.

Acknowledgments

Indeed all praise is due to the almighty Allah. I would like to express my deepest gratitude to my advisor Dr. Dongfang Zhao for the exceptional guidance, caring, patience, and engagement throughout my study at the University of Nevada, Reno. Thank you for all your intellectual and emotional support throughout this dissertation work, especially during hard times. He has been a great friend and guide to me during my Ph.D. study. I am very fortunate to have him as my advisor. I would like to express my sincere gratitude to my committee members, Dr. Feng Yan, Dr. Lei Yang, Dr. Tin Nguyen, and Dr. Yan Wang, for their invaluable intellectual input in improving the dissertation work.

I would also like to thank all the members of the High Performance and Data-Intensive Computing (HPDIC) Lab at UNR, my collaborators from Lawrence Berkeley National Lab, and Dr. Haoting Shen from Zhejiang University, with whom I had a great chance to collaborate. Their diverse backgrounds, inspiring suggestions, and in-depth discussions have formed an intellectual environment for interdisciplinary research. I am eternally grateful to my family for their endless love and support. I dedicate all my works in this dissertation to my parents, without whom I couldn't have come this far.

TABLE OF CONTENTS

Abstract	i
Acknowledgments	iii
List of Tables	xiii
List of Figures	xiv
1 Introduction	1
1.1 Overview	1
1.1.1 Lightweight blockchains for reliable data management in HPC systems	2
1.1.2 Lightweight blockchains for reliable data management in Internet of Things	3
1.1.3 Summary	4
I Lightweight Blockchains for Reliable Data Management in HPC Systems	6
2 SciChain: Scientific Blockchains for Immutable Provenance in Large-scale	

Distributed Systems	7
2.1 Introduction	7
2.1.1 Motivation	7
2.1.2 Challenges in current solutions	9
2.1.3 Proposed Approach	12
2.2 Background and Related Work	14
2.3 System Design	17
2.3.1 Architecture Overview	17
2.3.1.1 Lightweight Distributed Blockchain	17
2.3.1.2 Shared Storage Persistence	18
2.3.1.3 Ledger Synchronization Protocol	18
2.3.1.4 Consensus Protocol	19
2.3.2 Protocols	20
2.3.2.1 Proof of Extended Traceability (POET)	21
2.3.2.2 Proof of Scalable Traceability (POST)	23
2.3.2.3 In-memory Block Manager	25
2.3.2.4 Query Utility	26
2.3.3 Analysis of POST	26

2.3.3.1	Safety of POST	26
2.3.3.2	Liveness of POST	28
2.3.4	Overhead Reduction	29
2.3.5	Reliability	30
2.4	System Implementation	32
2.4.1	Overview	32
2.4.2	Node Threads	33
2.4.3	Network Queues	34
2.4.4	Data and Storage Models	35
2.4.5	Data Collection and Query	36
2.5	Evaluation	37
2.5.1	Experiment Setup	37
2.5.1.1	Testbed	37
2.5.1.2	Evaluated Systems	37
2.5.1.3	Workload	38
2.5.2	Reliability and Trustworthiness	39
2.5.3	Overhead	40
2.5.4	Scalability	42

2.5.5	Query Performance	44
2.5.6	Large-Scale Applications	45
2.6	Summary	46
3	BAASH: Lightweight, Efficient, and Reliable Blockchain-As-A-Service for HPC Systems	47
3.1	Introduction	47
3.1.1	Motivation	47
3.1.2	Challenges	49
3.1.3	Our Contributions	51
3.2	BAASH Design	53
3.2.1	Architecture	53
3.2.1.1	Resilient distributed ledger	53
3.2.1.2	Scalable in-memory & shared-storage consensus protocol	54
3.2.1.3	Parallel processing layer	55
3.2.1.4	Fault monitor	55
3.2.2	BAASH Protocols	56
3.2.2.1	Parallel transaction manager	57
3.2.2.2	BAASH consensus	57

3.2.2.3	Block validation	59
3.2.2.4	Persist-in-storage	60
3.3	System Implementation	62
3.3.1	Data Models	63
3.3.2	Worker Nodes	63
3.3.3	Block and Transaction Queue	64
3.3.4	Parallel Processing Module	65
3.3.5	Consensus Manager	67
3.3.6	Fault Monitor	67
3.4	Evaluation	69
3.4.1	Experimental Setup	69
3.4.2	Throughput	71
3.4.3	Performance and Memory Overhead	72
3.4.4	Reliability and Fault Tolerance	74
3.5	Related Work	77
3.6	Summary	79

II Lightweight Blockchains for Reliable Data Management in In-

ternet of Things **80**

4 DEAN: A Lightweight and Resource-efficient Blockchain Protocol for Reliable Edge Computing	81
4.1 Introduction	81
4.1.1 Motivation	81
4.1.2 Challenges	84
4.1.3 Proposed Solution	85
4.2 Threat Model	88
4.3 Decentralized-Edge Autonomous Network	89
4.3.1 System Components	89
4.3.1.1 Block Validation	89
4.3.1.2 Leaders Election	91
4.3.1.3 Load Balance	95
4.3.2 Protocols	96
4.3.3 Analysis	103
4.3.3.1 Security Guarantee	103
4.3.3.2 Time Complexity	104
4.3.3.3 Space complexity	105

4.4	Evaluation	105
4.4.1	Experimental Setup	105
4.4.2	Fault Tolerance	106
4.4.3	Throughput	108
4.4.4	Latency	110
4.4.5	CPU and Memory Usage	111
4.4.6	Scalability	112
4.4.7	Sensitivity with increasing workload	114
4.5	Summary	115
5	HWCOVER: Hardware-based Consortium Blockchain Verification with Privacy Preservation	116
5.1	Introduction	116
5.2	Related Works and Challenges	120
5.2.1	Device Verification	120
5.2.2	Blockchains for Supply Chain	122
5.2.3	Integrating PUF into Blockchains	123
5.2.4	Challenges	124
5.2.4.1	Risk of PUF information leakage.	124

5.2.4.2	Privacy concerns.	125
5.3	Threat Model	125
5.4	HWCOVER Design	127
5.4.1	Overview	127
5.4.2	Registration	130
5.4.3	Transaction	134
5.4.4	Consensus	136
5.4.5	Assembling and Disassembling	140
5.5	Security Discussion	141
5.5.1	Unauthorized Devices	141
5.5.2	PUF-data Leakage	142
5.5.3	Malicious Users	142
5.5.4	Privacy Concerns	143
5.5.5	Manipulated Voting	143
5.5.6	Ledger Tampering	146
5.6	Performance Evaluation	147
5.6.1	Implementation and Experimental Setup	147
5.6.2	Cost during Registration	149

5.6.3	Throughput of Transactions	150
5.6.4	Latency	153
5.7	Summary	154
6	Conclusions and Future works	155
6.1	Conclusions	155
6.2	Future works	156
	Bibliography	158

LIST OF TABLES

2.1	Summary of limitations of present blockchain-based provenance systems. . .	11
2.2	Performance of PUSH mechanism to persist in shared storage.	41
2.3	Performance of PULL mechanism to synchronize nodes.	42
3.1	Summary of limitations of present blockchain-based provenance systems.	49
3.2	Maximum memory requirement per node in BAASH at different scales. . .	74
4.1	Feature comparison among existing blockchain protocols and the pro- posed protocol, DEAN.	84
4.2	Complexities of various blockchain sharding protocols.	104
4.3	CPU utilization and memory footprint of DEAN.	111
5.1	Comparison among PoW, Tendermint, and HWCOVER.	139
5.2	Required cluster size to keep consensus error lower than 1×10^{-5} in two rounds.	151

LIST OF FIGURES

2.1	Comparing centralized and distributed provenance.	8
2.2	Conventional blockchain architecture on shared-nothing platforms. This vanilla construction is inappropriate for the HPC infrastructure.	10
2.3	Proposed SciChain architecture. A HPC blockchain crafted to function with shared storage ecosystem.	12
2.4	SciChain as a middleware in HPC infrastructure assists in managing distributed reliable provenance.	12
2.5	Parallel operations of POET and POST protocols assist in validating and storing meta data before queried by others.	19
2.6	Shared storage accumulates consensus. Note that the ‘shared storage’ is a cluster of nodes, in despite of being drawn as a single cylinder for simplicity here.	20
2.7	Overview of system interaction in current SciChain implementation. . . .	32
2.8	Data structures of SciChain transactions. The shared storage is in practice a cluster of physical nodes.	35
2.9	POST guarantees more than 50% of nodes always hold valid ledger. . . .	39
2.10	Scalability of SciChain against the baseline during the block with provenance data storing time.	40

2.11	Memory overhead of SciChain. Push-Pull mechanism minimizes in-memory load while ensuring the reliable persistence through shared-storage.	41
2.12	Scalability of three blockchain systems for block validation time during the reproduction.	43
2.13	Comparing provenance query performance. SciChain with double audibility assists in verification of data again through decentralized protocol during the query. SciChain with pre-audibility relies on POST and POET protocols and doesn't require further verification during the query.	44
2.14	Latency distribution on 1,024 nodes.	45
3.1	BAASH service is useful in a typical data movement workflow in an HPC environment.	48
3.2	Overall architecture of BAASH on HPC systems.	52
3.3	BAASH implementation with MPI and shared storage.	61
3.4	Parallel processing module in BAASH manages blocks in parallel through distributed coordinators or consensus managers.	65
3.5	A consensus manager (i.e., a coordinator) guarantees the consistency in block management through the two-phase distributed quorum commit protocol.	66
3.6	Workflow of BAASH's monitor to handle MPI failures. The monitor ensures a consistent BAASH service against arbitrary node failures.	68

3.7	Transactions generated per second. BAASH outperforms the conventional blockchains. BAASH exhibits scalability compared to the state-of-the-art HPC blockchain systems.	70
3.8	Performance and overhead comparison.	72
3.9	Ledger validity in BAASH (in all scales at-least 99% nodes hold consistent ledger).	74
3.10	Reliability in BAASH resilience support.	75
3.11	BAASH recovery overhead by fault monitor.	76
4.1	Four-tier edge computing architecture with DEAN. Exhibiting sample scenarios where DEAN can assure reliable data management on edge nodes.	86
4.2	Three parallel processes work together in DEAN.	89
4.3	Block validation continues while leaders are being elected in DEAN.	90
4.4	Finding the optimal m to avoid sub-linear or super-linear nature of the nodes' adjacency. $7 < m \leq 10$ produces more than 50% leaders with reasonable adjacency at various scales.	92
4.5	Comparing various blockchain management approaches against DEAN.	95
4.6	Trustworthy nodes assists in load-balancing through side chain in DEAN.	95

4.7	DEAN's Fault Tolerance. DEAN guarantees that more than 50% of nodes remain intact and hold valid blockchain in practice in front of various number of nodes failure.	106
4.8	Throughput Comparison between DEAN and state-of-the-art blockchain systems. IOTA is set with difficulty=1. DEAN outperforms major blockchain systems with orders of magnitude higher throughput.	108
4.9	Latency comparison between DEAN and state-of-the-art blockchain systems. DEAN is orders of magnitude faster than others and incurs only sub-second for adding 10,000 blocks.	109
4.10	Scalability of DEAN's block processing time. DEAN takes less than five seconds for processing a new block on extreme scales of 1,000 nodes. . . .	112
4.11	DEAN's block appending time with various distributions of sensor and edge nodes.	113
5.1	Electronics supply chain include multiple participants from different countries. Unreliable validations (with question marks) and lacks of data sharing (with cross marks) render the global-scale supply chain vulnerable. . .	117
5.2	Typical supply chain activities.	127
5.3	Registration and transaction processes.	130
5.4	Data chain (i.e., ledger) structure.	133
5.5	HWCOVER manages load balance through dynamic sub-clusters. Sub-clusters get merged after a job is finished.	133

5.6	Assembling of related previous transactions into a new transaction.	140
5.7	HWCOVER protocol follows two rounds to guarantee consensus. There are 1,000 consortium nodes in total and the cluster size changes from 3 to 101.	145
5.8	A product registration cost comparison both in broadcast and parallel peer-to-peer (p2p) method at various scales.	148
5.9	Different number of products registration cost at various scales through batching in parallel peer-to-peer (p2p) method.	148
5.10	Throughput comparison. PoW is set with minimum difficulty (= 1).	150
5.11	Latency comparison in various systems.	153

CHAPTER 1

INTRODUCTION

1.1 Overview

Nowadays, due to the explosion of data, reliable data management is an essential task. The volume of data is increasing rapidly every moment in each paradigm, such as scientific computing, cloud computing, edge computing, health care, digital currency, etc. Many organizations are trying to adopt blockchain into their ecosystem to guarantee reliable data management. For secure data management, blockchain plays a significant role, as it (i.e., blockchain) offers immutability and consistency among the data that enhance trustworthiness.

Blockchain, also referred to as Distributed Ledger Technology (DLT), tracks the entire history of a digital asset. DLT (i.e., blockchain) leverages decentralized protocol and hashing mechanism to provide immutability and transparency during the coordination of the history of an asset. Blockchain can track tangible (a house, car, cash, land) or intangible (intellectual property, patents, copyrights, branding) assets. A blockchain network can track scientific data and financial transactions such as orders, payments, production, etc. Due to the decentralization property, all members in a blockchain network share a single view of the truth. We can observe all details of a transaction end to end, which provides greater confidence in data.

However, adopting blockchain in its current model comes with a lot of critical challenges.

The current blockchain is crafted based on shared-nothing model that makes it highly resource intensive. All the nodes in the network require to have a huge amount of disk space to hold the individual blockchain copy. Although, a few works introduce the sharding mechanism that allows us to split the entire blockchain and to distribute the pieces among the nodes, the present sharding techniques are not dependable enough to apply in some infrastructure such as edge computing. Besides, the consensus protocols used in the current blockchains are very heavy and unreliable. These protocols either require to perform enormous computation (e.g., proof of work) or communication (e.g., byzantine fault tolerance) to reach to a consensus. While a few works attempt to amend the cost of these protocols through a leader election approach (e.g., proof of stake), the present models are not able to establish the trustworthiness of this technique. This is because the current solutions either lead to centralization or faulty leader election approach. Therefore, the goal of this dissertation is to explore the various approaches to overcome these challenges and to adopt blockchain seamlessly in any resource constraint environment.

In the following sections, we present a brief overview of this dissertation that explores various mechanisms to leverage blockchain to ensure reliable data management in different paradigms.

1.1.1 Lightweight blockchains for reliable data management in HPC systems

Scientific computing community utilizes high-performance computing (HPC) infrastructure to conduct scientific experiments in order to obtain results quickly. This is because compute nodes in an HPC cluster are usually equipped with much more memory and com-

putational CPU cores compared to any personal computer. In HPC system, people use centralized provenance to track down the meta data of their scientific computing results. Besides, in exascale systems, the compute nodes tend to keep the data in cache to facilitate the quick data supply due to enormous applications interdependence.

However, the current practice of tracking meta data in scientific provenance and caching distributed data across distributed nodes lead to several issues. First, as the provenance is centralized, there is always a chance of single point of failure. There is no guarantee that the provenance data is not falsified intentionally by an intruder or unintentionally due to software exceptions. Besides, we can not trust a single node verification while storing the meta data. On top of that, while caching a specific application data across the distributed nodes, there is no guarantee that the cached data is in same state across all the nodes. Therefore, the first part of this dissertation explores several approaches to adopt blockchain that enforce decentralized data management technique to address the aforementioned issues.

1.1.2 Lightweight blockchains for reliable data management in Internet of Things

In edge computing, an efficient means to process collected data is to persist data among the edge nodes rather than transferring data back to remote data centers. Edge computing assists in reducing a lot of network traffic that provide a better experience in the application performance. However, edge computing poses new technical challenges, including security and privacy concerns with edge nodes and sensors.

The state-of-the-art techniques are only focusing on finding out an encryption mechanism

to protect the data among the edge nodes. However, we can not apply heavy encryption on the edge nodes as they (edge nodes) are interconnected with weak WiFi connections and contain limited CPU, memory, and storage. Besides, there is no assurance that a single node equipped with the most effective encryption can protect the data. Hence, the second part of this dissertation presents a lightweight approach to adopting blockchain in the edge computing ecosystem to resolve the security challenges.

To date, privacy is a major concern in a blockchain database. For instance, the conventional electronic supply chain is vulnerable to counterfeiting attacks due to the limited traceability of transactions and lack of unclonable product identities. Although blockchain combining physical unclonable functions (PUFs) has been suggested to improve security, discussions on PUF data management and users' privacy preservation in such a decentralized supply chain management system are insufficient. Therefore, the last part of this dissertation discusses a lightweight blockchain-based privacy preservation mechanism powered by differentiated hashing on PUF challenge-response pairs.

1.1.3 Summary

This dissertation can be categorized into two main parts. Part I studies various decentralized protocols under the hood of blockchain for reliable data management in the HPC ecosystem. Part II investigates the reliable data management through the blockchain-like decentralized protocol in the internet of things (IoT) infrastructures. More specifically, in chapter 2, we present SciChain, which presents a unique solution to adopt blockchain to provide reliable data management in scientific provenance. Chapter 3 discusses the first-ever scalable blockchain for HPC, namely, BAASH, that allows parallel block processing.

Chapter 4 introduces the very first practical blockchain for IoT infrastructures, namely, DEAN, that seamlessly adopts blockchain in the edge computing ecosystem through a lightweight protocol. Chapter 5 presents HWCOVER that proposes a lightweight privacy preservation approach co-designed with PUF (physically unclonable functions) to secure data in a decentralized electronic supply chain management system.

Part I

Lightweight Blockchains for Reliable Data Management in HPC Systems

CHAPTER 2
**SCICHAIN: SCIENTIFIC BLOCKCHAINS FOR IMMUTABLE PROVENANCE
IN LARGE-SCALE DISTRIBUTED SYSTEMS**

2.1 Introduction

2.1.1 Motivation

A fundamental means to reproduce computational scientific results is through data provenance, which tracks the entire lifespan of the data during the experiments and simulation at various phases such as data creation, data changes, and data archival. Data provenance plays a critical role in guaranteeing the validity of scientific discoveries and research results, as data fabrication and falsification could happen to meet research objectives or personal interests or both. For instance, the National Cancer Institute found that 0.25% of trial data are fraudulent in the year of 2015 [41, 151]. In earth sciences, scientists emphasized the importance of maintaining data provenance in achieving the transparency of scientific discoveries [140].

As shown in Figure 2.1, conventional provenance systems can be categorized into two types: centralized provenance systems and distributed provenance systems. One popular centralized provenance system is SPADE [60], where the provenance (from various data sources) is collected and managed by a centralized relational database management system (RDBMS). Domain-specific systems based on such centralized design paradigms



Figure 2.1: Comparing centralized and distributed provenance.

are also available in biomedical engineering [35], computational chemistry [113], among others. Although having been reasonably adopted by various disciplines, the centralized provenance systems are being increasingly criticized due to the exponentially-grown data: the centralized provenance system becomes a performance bottleneck and a single point of failure as shown in Figure 2.1(a), and to this end, we witness the inception of various distributed approaches toward *scalable provenance* [38, 168]. Indeed, those distributed provenance systems, mostly built upon distributed file systems as opposed to centralized databases, eliminated the performance bottleneck and proved to deliver orders of magnitude higher performance than centralized approaches.

As a double-edged sword, however, distributed provenance systems raise a new concern [20] on the provenance itself: *while the provenance is supposed to audit the execution of the application, who then should audit the provenance as illustrated in Figure 2.1(b)?* Do we need to build the provenance of provenance? So the recursion goes on and on, indefinitely. Note that this concern was not that critical in a centralized approach as long as we can, which is the case, apply robust reliability mechanisms to the centralized node. However, it turns out to be an extremely challenging problem for all the participating nodes in a (large-scale) distributed system: if any single node of the entire deployment is com-

promised, the provenance as a whole becomes invalid.

To this end, *distributed provenance systems* were recently proposed, inspired by blockchains. These systems (e.g., ProvChain [93], SmartProvenance [116], LineageChain [120]) are also called blockchain-based provenance systems that are both tamper-evident and autonomous, thus guarantee trustworthiness of the provenance data. They share the same key idea: *instead of storing the data on a single node or splitting the data into n exclusively distinct chunks on n nodes, let us **replicate** the data and maintain a **hashed linkedlist** for each copy of the data.* The *replication* guarantees the provenance is tolerant to a certain degree of fault (e.g., $\lfloor \frac{n-1}{3} \rfloor$ in a Byzantine system, where the fault is arbitrary including malicious activities and even possible coalition among participants), and the *hashed linkedlist* guarantees that the provenance data cannot be tampered with without being noticed by a simple hash verification.

2.1.2 Challenges in current solutions

In the context of scientific applications and high-performance computing (HPC), we encounter unique challenges in employing current blockchain-based provenance services. There is a series of concerns on resource utilization: the space efficiency is low, the network bandwidth consumption is high, and the CPU cycles are “wasted” for meaningless mining, to name a few. Besides, existing blockchain-based provenance systems are built in such a way that the underlying blockchain infrastructure is a black box, and the provenance service works as a higher-level application by calling the programming interfaces provided by the blockchain infrastructure such as Hyperledger Fabric [70] and Ethereum [53]. In the best case, the provenance service might miss optimization and customization opportunities

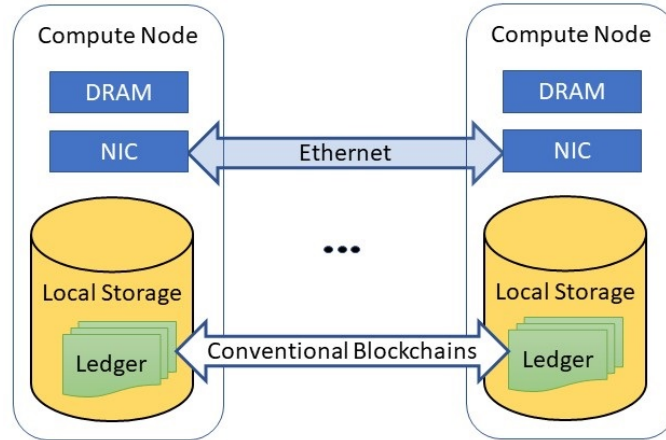


Figure 2.2: Conventional blockchain architecture on shared-nothing platforms. This vanilla construction is inappropriate for the HPC infrastructure.

because it cannot modify the lower blockchain layer; to make it worse, the applicability of those blockchain-based provenance systems is constrained by the underlying blockchain infrastructure, which is not optimized for, or not applicable to, HPC platforms.

To be more specific, we highlight (shown in Table 2.1) the most outstanding limitations exhibited by existing blockchain-based provenance systems in the context of scientific computing and HPC systems.

1. If the compute nodes of the scientific experiments have no local disks, which is very common for large-scale HPC systems, then the blockchain-based provenance system (at least in its current form) becomes useless as all blockchain systems require node-local persistent storage, as shown in Figure 2.2 (more discussion in Section 2.2).
2. If the consensus protocol of the underlying blockchain system is inappropriate for the scientific applications, then again, the provenance system becomes useless. To make the matter more concrete, the popular proof-of-work (PoW) consensus protocol is taken by many public blockchains that can hardly be applied to large-scale scientific simulations: it is a hard call to ask every single compute node to (re-)run

Table 2.1
SUMMARY OF LIMITATIONS OF PRESENT BLOCKCHAIN-BASED PROVENANCE SYSTEMS.

Features	ProvChain [93]	SmartProv [116]	LineageChain [120, 122]	SciChain
Support diskless nodes	×	×	×	✓
Light computation	×	×	✓	✓
Light communication	✓	✓	×	✓
HPC compatible protocol	×	×	×	✓
Multi-tiered storage support	×	×	×	✓
Lightweight ledger support	×	×	×	✓

the experiments.

3. If the experiment testbed has a multi-tiered storage architecture (e.g., burst buffers, I/O nodes, remote parallel file systems), the blockchains will be agnostic of such heterogeneity, leading to sub-optimal performance.

In summary, a highly desired provenance system for scientific applications should be crafted with a balance between scalability, reliability, and applicability. Unfortunately, existing provenance systems failed to meet the above requirements from scientific computing and HPC communities. Of note, a few recent works indeed proposed blockchains deployed to HPC systems. These works (more details are in Section 2.2) are either focused on parallel block processing or distributed trustworthy data resiliency [5,99] in the HPC systems or solely depend on conventional compute-intensive proof-of-work (PoW) protocol [7] which is hardly applicable to real-world settings for the HPC architecture.

To this end, the SciChain is the first blockchain framework crafted to guarantee distributed reliable provenance service in the HPC ecosystem. What is desired is a decentralized protocol that is particularly crafted to adopt distributed provenance in HPC systems and overcomes the resource utilization challenges from various perspectives, such as replacing the conventional compute-intensive consensus with more cost-effective ones, enforcing memory constraints on compute nodes, and limiting inter-node communications to reduce network overhead.

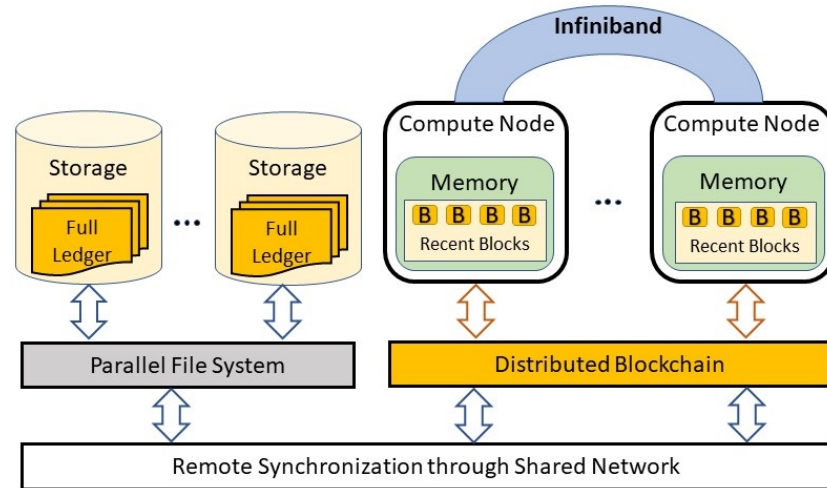


Figure 2.3: Proposed SciChain architecture. A HPC blockchain crafted to function with shared storage ecosystem.

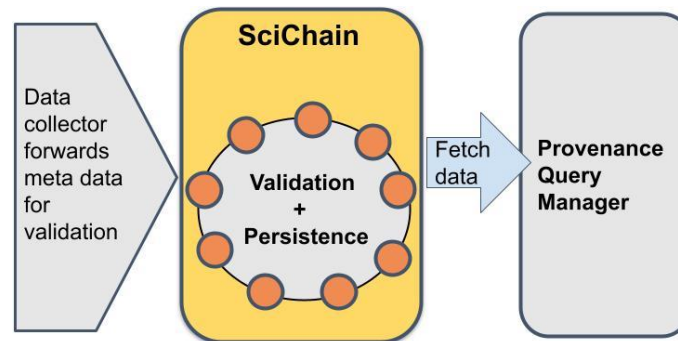


Figure 2.4: SciChain as a middleware in HPC infrastructure assists in managing distributed reliable provenance.

2.1.3 Proposed Approach

This chapter proposes a new distributed approach to manage the data provenance of scientific applications deployed to HPC systems. Rather than only taking an existing blockchain system as a block box, we hack into blockchain internals to improve the applicability and performance of provenance services built upon blockchains. Specifically, as shown in Figure 2.3, we propose a new blockchain architecture supporting multi-tier storage and then design new consensus protocols aiming to optimize the distributed provenance services in an HPC environment. The proposed architecture is deployable as a middleware to any

resource-constrained environment as depicted in Figure 2.3 and 2.4, where the nodes need to operate with limited memory and require no local disks (i.e., lightweight replication support) but are supported by a distributed remote-storage that persists the blockchain. Besides, the secured lightweight privacy mechanism applied in the newly proposed consensus protocol (i.e., HPC compatible lightweight custom protocol) allows the proposed system to be able to work in the blockchain-based HPC ecosystem.

In summary, this chapter discusses the following contributions:

- A new architecture for secure and reliable distributed data provenance on HPC systems. The new architecture is tailored to the HPC environment: compute nodes can maintain the blockchain in local memory with minimum overhead (i.e., lightweight ledger support) while using distributed shared ledger as a persistent medium (i.e., multi-tiered storage support) for enhanced reliability and as a precaution for any catastrophes (e.g., compute nodes failure or restart).
- A set of consensus protocols, namely, proof-of-scalable-traceability (POST), for validating applications' data provenance following a *push-pull* mechanism that promises memory optimization (i.e., HPC compatible lightweight custom protocol). The key idea of POST is that the consensus comes not only from the fellow compute nodes but also from the remote shared storage through proof-of-extended-traceability (POET). POET comes into action only if the compute nodes are unable to reach consensus. We also design a *query utility* protocol that offers a full-fledged provenance query support through an independent external query manager.
- The proofs for safety and liveness of the proposed consensus protocols. In particular, we show that all of the participating nodes would eventually reach consensus in front of arbitrary (Byzantine) failures.

- A details discussion of the proposed system prototype, SciChain, and experimental results that verify the system’s effectiveness (i.e., reliability in 2.5.2, extensive analysis on lightweightness in 2.5.3, in-depth scalability analysis in 2.5.4 and 2.5.6) with more than one million transactions derived from both micro-benchmarks and real-world applications on up to 1,024 nodes.

2.2 Background and Related Work

Data Provenance. Data provenance tracks the source of data and the movement among the different data sources [152]. It facilitates the debugging process that helps to ensure the reproducibility of results. Therefore, security and privacy issues, e.g., integrity, confidentiality, and availability, are the utmost concerns [20]. State-of-the-art filesystem-based provenance systems [38,168], and their variants such as in-memory blockchains [7], lack abstractions that are essential for enabling trustworthiness and reliability in provenance, calling for evaluations of the practical effectiveness of trustworthy provenance techniques [42].

Blockchain. A blockchain system, on the other hand, is a distributed ledger that consists of multiple nodes that are either fully or partially trustworthy. Even while most of the nodes are honest, some nodes tend to exhibit Byzantine behavior because of the unexpected attack. A faulty node that can have arbitrary behavior due to malicious attack [28] is called Byzantine. All the nodes are responsible for generating, validating, and appending blocks with transactions in their local ledger as well as maintaining a shared replica of a blockchain or ledger (i.e., a set of shared blocks of transactions) and global states. In a blockchain-like distributed ledger, more than 50% of nodes need to be tamper-proof to attain consensus about the validity of a block.

Blockchain Types. Two of the most popular types of blockchain systems so far, in terms of access permissions, are public and private blockchains. Other types are derived from these two basic types, such as inter-blockchains [73, 166]. In the private blockchain system (i.e., permissioned network), a node requires special permission. The network initiator then either verifies the node or prepares a set of rules to verify the node. This mechanism puts a restriction on who is allowed to participate in the network, and only in certain transactions. Once a node joins the network, it starts to play a role in maintaining the blockchain in a distributed manner. In a public blockchain network, anyone can join and participate. The network follows an incentivizing mechanism to encourage more participants to join the network. However, public blockchain systems have mainly two downsides. First, the openness of public blockchain supports weak security. Second, to select a tamper-proof leader, each node is required to solve a compute-intensive puzzle. The difficulty in the puzzle needs to be adjusted across time to ensure the authenticity of the blocks.

Protocols. Several consensus protocols have been employed in the blockchain systems to validate blocks such as proof-of-work (PoW), proof-of-stake (PoS), and practical byzantine fault tolerance (PBFT). In PoW, each node in the network needs to solve a complex puzzle that requires a significant amount of computation, and the winner gets incentivized [98]. A critical parameter in PoW-based blockchains is how to control the computational complexity to select the winner who can append the new block. The canonical approach is to ask the node to find a “nonce” number, combined with the hash value of the previous block, which will result in a hash value of the current block in a specific format, usually with a certain number of zeros. It is quite difficult to predict the right combination of bits that will result in the right hash; hence, it requires many different trials of computation until a hash with the required number of zero is discovered. The term *difficulty* sets the number of required zero bits. The resulting hash must contain a value less than the current difficulty and will hold a certain number of leading zero bits less than that. We will use the short term

nonce-number to indicate the number of leading zeros required by the PoW consensus. In PoS, the creator of a new block is chosen in a deterministic way, based on the wealth (i.e., stake) of the node and there is no reward for mining a block [80]. In PBFT, all of the nodes in a permissioned network within the system extensively communicate with each other to reach an agreement about the state of the system through a majority [61].

Blockchain Provenance. There are some recent works that aim to develop a blockchain system for HPC systems. Both HPChain [5] and BAASH [99] are orthogonal to this work as they presented an idea on MPI-based parallel block processing using an off-chain strategy. A consensus protocol, namely proof-of-reproducibility (PoR [7]) took the remote parallel filesystem as a pseudo independent local disk to adopt the black box of blockchain systems; the system represented a simple aggregation of parallel filesystem and blockchain systems, which brought limited insight and was not evaluated with real-world applications. That is, PoR does not follow a practical approach to apply in HPC systems for several reasons: (i) all the compute nodes are required to have a node-local disk (despite a mount point of the remote parallel filesystem) to replicate the entire blockchain; and (iii) PoR solely depends on the conventional compute-intensive PoW protocol, which is hardly applicable to real-world settings for the HPC architecture.

There are a few more efforts that address security and reliability challenges in distributed provenances in other areas with blockchain-based approach. SEDNP [96] provides decentralized provenance management tools to fix security challenges for operating distributed network provenance in the IoT ecosystem. Besides, UAVN-pro [59] proposes a blockchain-based provenance model to detect malicious activities in an unmanned aerial vehicle (UAV) network. Nonetheless, none of the aforementioned works address the unique challenge of deploying blockchain-based provenance in HPC infrastructure and still follow the vanilla shared-nothing blockchain mechanism.

Figure 2.2 shows the system architecture of conventional blockchains deployed to a shared-nothing cluster. Regardless of private (Hyperledger [70]) or public (Ethereum [53]), all existing blockchain systems assume that the underlying computer infrastructure is shared-nothing: the memory subsystems and I/O subsystems are all independent of the participant nodes which are often connected through commodity networking. In contrast, Figure 2.3 illustrates one possible practical blockchain deployment on HPC. In addition, Figure 2.4 depicts how SciChain as middleware in HPC infrastructure assists in managing distributed reliable provenance. We will discuss more detail in Section 2.4.

2.3 System Design

2.3.1 Architecture Overview

Our proposed system consists of four key modules: a lightweight distributed blockchain, shared storage persistence, a ledger synchronization protocol, and a consensus protocol. We will discuss each of them in more detail in the following.

2.3.1.1 Lightweight Distributed Blockchain

The first module is a distributed blockchain specially crafted to alleviate the memory constraints in compute nodes. The module helps enable the compute nodes to perform the block validation process while keeping a minimum number of recent blocks (e.g., 100-200 blocks) in memory. As we are interested in permissioned blockchains for HPC, a high-

performance network infrastructure (e.g., Infiniband) interconnects all the compute nodes to speed up the communication.

2.3.1.2 Shared Storage Persistence

The second module acts as a persistent medium to hold the entire blockchain. As we assume that the compute nodes are generally volatile and store the distributed blockchain in memory, persistent shared storage is essential to provide more reliable backup in case of any catastrophes (e.g., more than 50% compute nodes crash or restart or lose their in-memory blocks).

2.3.1.3 Ledger Synchronization Protocol

The third module is an extended validator protocol (i.e., proof-of-extended-traceability) that supports a faster validation mechanism. Whenever a block is broadcasted, first, the compute nodes come forward to validate the block with the support of the in-memory blocks. If more than 50% of nodes fail to validate the block, the protocol then employs the shared storage to provide the validation service. The protocol also supports the synchronization among the distributed ledger of the compute nodes and the shared storage. The benefit of this protocol is two-fold: (i) it synchronizes the compute nodes in case of any catastrophes; (ii) it provides a reliable guaranteed validation through the shared storage. We will discuss more details in Section 2.3.2.1.

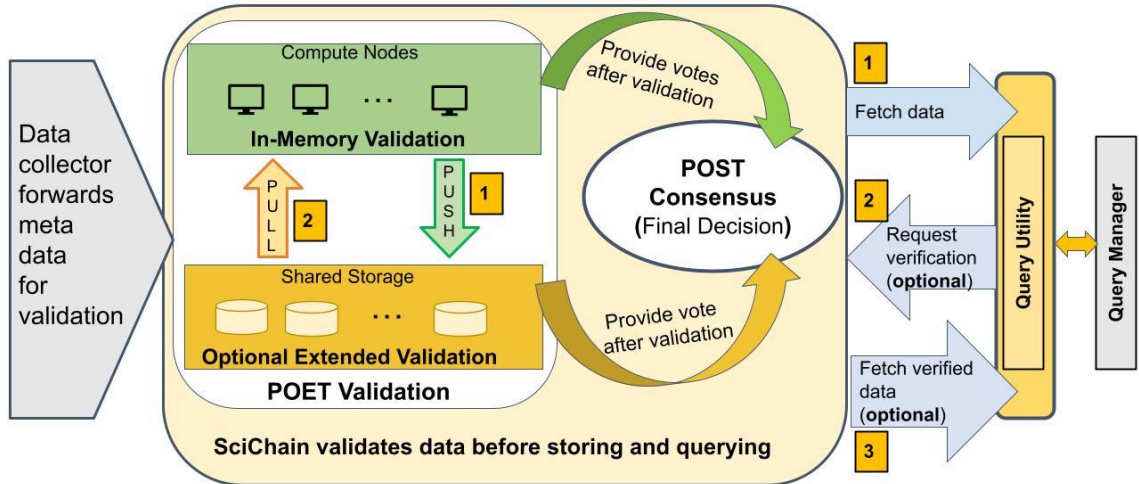


Figure 2.5: Parallel operations of POET and POST protocols assist in validating and storing meta data before queried by others.

2.3.1.4 Consensus Protocol

The fourth module manages the consensus mechanism among the compute nodes and the shared storage. The consensus protocol (i.e., proof-of-scalable-traceability) leverages the third module (i.e., ledger synchronization protocol) to employ the nodes in the validation process, and finally, helps in aggregating the compute nodes' votes to provide the system the consensus for a block. The benefit of this protocol is that it minimizes the extensive communication overhead between the compute nodes and the shared storage during the consensus gathering process. This is because the consensus process stops as soon as 51% of nodes can attain the consensus for a block. Section 2.3.2.2 provides more details on this protocol.

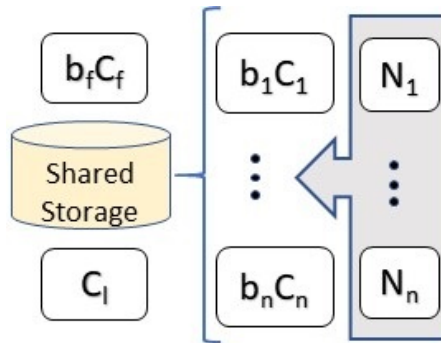


Figure 2.6: Shared storage accumulates consensus. Note that the ‘shared storage’ is a cluster of nodes, in despite of being drawn as a single cylinder for simplicity here.

2.3.2 Protocols

The proposed architecture is designed to work through primarily two parallel protocols as shown in Figure 2.5. Protocol 1 (i.e., proof-of-extended-traceability) helps in *validating a block consists of new provenance records*, while Protocol 2 (i.e., proof-of-scalable-traceability) works with *gathering consensus from the nodes after the validation and helps Protocol 1 in making the final decision* about storing the validated block both in the shared storage’s disk and compute nodes’ in-memory. The Protocol 3 assists the Protocol 1 to manage the block in compute nodes’ memory and the Protocol 4 acts as a query utility that helps the external provenance query manager to retrieve data from the SciChain.

The proposed protocol optimizes memory consumption following two phases, as shown in Figure 2.5. In Phase 1 (i.e., *Push method*), the shared storage is being leveraged to persist the entire ledger replica with minimum communication overhead as all ledgers on compute nodes are necessarily volatile and only keep the recent blocks in a limited amount of memory, as the nodes are usually disk-less. In Phase 2 (i.e., *Pull method*), Protocol 3 syncs the inconsistent compute nodes’ memory with the latest block to keep up the blockchain service. That is if a compute node is unable to validate a block and provide the vote, it is then added to the C_l list (i.e., shown in Figure 2.6) in the shared storage. If more than

50% of compute nodes fail to provide vote during a block validation process, the shared storage will update the nodes enlisted in C_l with the recent block from the storage node's blockchain.

To reduce the communication overhead, the overall consensus (i.e., 51% votes) aggregation is controlled by the shared storage, as shown in Figure 2.6. The compute nodes forward their *hashed-votes* (b_iC_i , where b indicates a block and C indicates a vote by a node for that block) to the shared storage after validating a block, where it (i.e., shared storage) has the aggregation mechanism to generate the final consensus for a block (b_fC_f). During the aggregation process, the shared-storage randomly picks one of the compute nodes' hashes from the *hashed-votes* list to set the *new-id* for the newly validated block. As soon as 51% consensus is reached, the shared storage broadcasts the final consensus along with the hashed-votes list to the compute nodes to suspend further validation. A node's hashed-vote is encrypted with its private key.

2.3.2.1 Proof of Extended Traceability (POET)

POET (i.e., shown in Protocol 1 helps in validating a block and works in two steps. First, it checks whether it can validate a block (i.e., new provenance records) with the help of the recent blocks stored in the compute nodes memory, as shown in Line 5. If a compute node is unable to validate a block, it is enlisted to the C_l (Line 7). If more than 50% compute nodes fail (Line 14), which is rare, the storage node then comes forward to proceed further with the block validation process as shown in Line 15. The block is stored both in the shared storage (*Push method* in Line 22) as well as in the compute nodes with the help of Protocol 3 (Line 25) only if more than 50% nodes jointly from compute nodes and shared storage provide consensus (Line 21).

Protocol 1 Proof-of-Extended-Traceability on storage

Require: Compute nodes C where the i -th node is C^i ; C_B^i the in-memory recent block list on C^i ; b_o oldest block in the i -th compute node's C_B^i ; a newly mined block b ; shared storage D ; D_B the blockchain copy on the storage; Synced compute nodes C_l where the k -th node is C_l^k ; Total nodes N .

Ensure: b is validated by both the local ledger C_B^i and the shared ledger D_B , and then appended to all C_B 's and D_B .

```

1: function POET-VALIDATION( $b, C, D$ )
2:    $i \leftarrow 1$ 
3:   for  $C^i \in C$  do
4:     while  $|votes| \leq \frac{N}{2}$  do
5:       Validate  $b$  in  $C_i$ 
6:       if  $C_i$  can not validate  $b$  then
7:          $C_l \leftarrow C_l \cup C^i$ 
8:       else
9:          $votes \leftarrow$  POST-CONSENSUS( $b, C^i$ )            $\triangleright$  Protocol 2
10:      end if
11:       $i \leftarrow i + 1$ 
12:    end while
13:  end for
14:  if  $|votes| \leq \frac{N}{2}$  then
15:    Validate  $\bar{b}$  with  $D$ 
16:     $votes \leftarrow$  POST-CONSENSUS( $b, D$ )            $\triangleright$  Protocol 2
17:    for  $C_l^k \in C_l$  do
18:      Sync recent block from  $D_B$             $\triangleright$  Pull method
19:    end for
20:  end if
21:  if  $|votes| > \frac{N}{2}$  then
22:     $D_B \leftarrow D_B \cup b$             $\triangleright$  Push method
23:    for  $C_i \in C$  do
24:      if  $b \notin C_B^i$  then
25:        BLOCK-MANAGER( $C^i, b$ )            $\triangleright$  Protocol 3
26:      end if
27:    end for
28:  end if
29: end function

```

The benefits of this new validation method are two-fold: (i) A faster validation process achieved by the agreement of the compute nodes, thanks to the in-memory support, (ii) Extended and stable validation support is achieved from the shared storage that serves as a reliable persistent medium if more than 50% of compute nodes are compromised or fail to provide consensus. The time complexity of Protocol 1 is $O(|C|)$, which is on par with the original PoW consensus. However, most of the time, it is possible to reduce the number of iterations, as we stop the validation process as soon as 51% consensus is attained.

Protocol 2 All compute and storage nodes reach consensus

Require: A new block b ; shared storage D ; D_B the blockchain copy on the storage; a node n ; Total nodes N .

Ensure: At least 51% nodes provide *votes* who validate b both with in memory latest block list and with shared ledger D_B .

```

1: function POST-CONSENSUS( $b, n$ )
2:   if  $b \notin D_B$  then
3:     if  $n$  can validate  $b$  then
4:        $votes \leftarrow votes \cup n$ 
5:        $counter \leftarrow counter + 1$ 
6:       if  $|votes| > \frac{N}{2}$  then
7:         break ▷ Stop consensus process.
8:       end if
9:     end if
10:    if  $counter \geq \frac{N}{2}$  then ▷ majority nodes checked?
11:      if  $|votes| \leq \frac{N}{2}$  then
12:        Push  $b$  to wait queue
13:      end if
14:    end if
15:  end if
16: end function

```

Protocol 3 In-Memory Block Manager

Require: Compute nodes C where the i -th node is C^i ; C_B^i the in-memory recent block list on C^i ; a newly mined block b ; b_o block with minimum hit-ratio in the i -th compute node's C_B^i .

Ensure: b_o is selected from the local SciChain ledger C_B^i .

```

1: function BLOCK-MANAGER( $C^i, b$ )
2:    $b_o \leftarrow$  Find a block with minimum hit-ratio from  $C_B^i$ 
3:    $C_B^i \leftarrow C_B^i - b_o$  ▷ Discard unused block
4:    $C_B^i \leftarrow C_B^i \cup b$  ▷ Append new block in memory
5: end function

```

2.3.2.2 Proof of Scalable Traceability (POST)

To achieve the consensus, as shown in Figure 2.5, Protocol 2 is applied in parallel with Protocol 1 (i.e., POET). Protocol 2 ensures that most of the compute nodes (more than 50%) guarantee the validity of the newly proposed block. If 51% compute nodes get compromised, it is guaranteed that at-least the shared storage can ensure the validity. To avoid duplication, the new block validation and consensus collection process start only if the block is not already validated, as shown in Line 2 of Protocol 2. If the provenance records

Protocol 4 Query Utility

Require: Compute nodes C where the i -th node is C^i ; C_B^i the in-memory recent block list on C^i ; a block b ; queries Q where the i -th query is Q^i ; shared storage D ; D_B the blockchain copy on the storage; synced compute nodes C_l where the i -th node is C_l^i ; valid data M ; data validation status M_v .

Ensure: Q is selected and optionally re-verified from the SciChain ledger.

```

1: function QUERY-UTILITY( $Q$ )
2:    $M \leftarrow$  Find  $b$  containing  $Q^i$  in closest up to three nodes
3:   if Re-verification is off then
4:     if  $M$  is  $\emptyset$  then
5:       Find data in  $D$  ▷ Shared storage
6:     end if
7:     if  $M$  is not  $\emptyset$  then
8:       return  $M$ 
9:       break ▷ No further verification
10:    end if
11:  else
12:    for  $C^i \in C$  do
13:      while  $|votes| \leq \frac{C}{2}$  do
14:        if  $C^i$  can re-verify hash of  $M$  then
15:           $votes \leftarrow votes \cup C^i$ 
16:        else
17:           $C_l \leftarrow C_l \cup C^i$ 
18:        end if
19:      end while
20:    end for
21:    if  $|votes| \leq \frac{C}{2}$  then
22:      if Re-verify with  $D_B$  then
23:         $votes \leftarrow votes \cup D$ 
24:        for  $C_l^i \in C_l$  do
25:          Update ledger of  $C_l$  with  $b$ 
26:        end for
27:         $M_v \leftarrow valid$ 
28:      end if
29:    else
30:       $M_v \leftarrow valid$ 
31:    end if
32:    if  $M$  is not  $\emptyset$  &  $M_v$  is valid then
33:      return  $M$ 
34:    end if
35:  end if
36: end function

```

in the newly proposed block are new, a node attempts to validate the block (Line 3). Each node provides a vote after validating the block (i.e., Line 4). If more than 50% nodes agree on the validity of the new block, the block will be decided to append both in the shared storage and the compute nodes. However, if at-least 51% nodes jointly from compute nodes

and share storage are unable to reach the consensus (i.e., Line 11), then the block is then pushed in the wait queue (i.e., Line 12). To minimize the block validation cost, we stop the consensus attaining process as soon as at least 51% compute nodes provide consensus, as shown in Line 7.

Protocol 2 utilizes Protocol 1 and does not impact the agreement formed between participant nodes and the storage node. Therefore, we need to demonstrate that the new protocol indeed leads to consensus, meaning that at least 51% of the compute nodes are guaranteed to hold the valid ledgers in front of arbitrary attacks. We show that this is indeed the case in the next section.

2.3.2.3 In-memory Block Manager

To optimize the memory of a compute node, we remove the most unused block from the in-memory block list of a compute node before appending the newest block using Protocol 3. In this protocol, SciChain finds the block with the minimum hit ratio as shown in Line 2. The transactions in a block with a minimum hit ratio exhibit the least accessed provenance records; thus, moving these records assist in clearing up space in a compute node's memory. Once SciChain identifies the most unused block, Protocol 3 removes the block from the compute node's memory (Line 3). Finally, the protocol helps the new block to get appended in the compute node's local ledger as shown in Line 4.

2.3.2.4 Query Utility

The Protocol 4 serves as a query utility to the external provenance query manager. When an external query manager submits a query to SciChain, the query utility process the query request with or without post-verification service with the support of the SciChain. The external query manager can access the SciChain API to submit the query request to the utility service.

The protocol process the query request in two steps. First, it (i.e., protocol) looks for the data requested by the query in the nearby compute nodes' memory with a maximum of three attempts as shown in Line 2. If the first three nodes are unable to provide the data, the shared storage provides the data (Line 5). If the external query manager submits a query with the post-verification request, the authenticity of the data is verified again with the majority of compute nodes (Line 13). The shared storage comes forward if more than 50% compute nodes can not post-verify the data as shown in Line 22. If the shared storage is able to verify the data, the inconsistent compute nodes' local ledgers get updated with the latest block (Line 25) before the query utility responds to the external query manager with the requested data (Line 33).

2.3.3 Analysis of POST

2.3.3.1 Safety of POST

In the literature of distributed computing, the *safety* property of a consensus protocol requires that “nothing bad will happen.” That is, whatever value agreed upon by all the nodes

should be a valid value proposed by the nodes—it cannot be an arbitrary value. That is the value upon which all nodes reach consensus is the same as the original shared-nothing cluster, which essentially guarantees the safety property. More specifically, we would like to analyze that if a shared-nothing cluster consists of N -nodes is sustainable to up to $\lfloor \frac{N}{2} \rfloor$ tampered nodes, then with at-least one additional shared-storage node the expanded cluster $N + 1$ is also sustainable to up to $\lfloor \frac{N+1}{2} \rfloor$ tampered nodes.

Assume the first half of the nodes (i.e., $\lfloor \frac{N}{2} \rfloor$) are endangered. We will prove that the proof-of-scalable-traceability protocol that leverages the shared storage node (index $N+1$), will secure at least $\lfloor \frac{N+1}{2} \rfloor$ nodes are tamper-proof (i.e., holding valid blocks).

As $\lfloor \frac{N}{2} \rfloor \equiv \frac{N-1}{2}$ implies that at least $\frac{N+1}{2}$, which is equal to $\lfloor \frac{N+1}{2} \rfloor$, nodes are tamper-proof, as long as N is indivisible by two. In other words, in the shared-storage cluster where there are $N + 1$ nodes, we can ensure that at least 50% of nodes hold the valid blocks.

If N is divisible by two, i.e., $\lfloor \frac{N}{2} \rfloor \equiv \frac{N}{2}$, we will show this case by contradiction. As in the beginning, we assume the first half of nodes are endangered; so, let us also consider that the storage node itself is endangered. To be more specific, we now assume all nodes [with indices $1, 2, \dots, \frac{N}{2}, N + 1$] are endangered. According to Lines 4 and 21 in Protocol 1, the blocks in the ledger of the storage node all validated by more than 50% of nodes, as opposed to the compute-node-only validation where more than 50% of compute nodes could have validated a new block. Thus, if node $N+1$ is tampered, then nodes $\frac{N}{2}+1, \frac{N}{2}+2, \dots, N$ are not able to hold valid blocks because all of them are double-checked by node $N + 1$. However, our assumption says the second half of the compute nodes are tamper-proof, which leads to a contradiction. Therefore, we show that the shared-storage ($N + 1$)-node cluster can still sustain up to $\lfloor \frac{N+1}{2} \rfloor$ tampered nodes, even though N is divisible by two.

2.3.3.2 Liveness of POST

In this section, the non-blocking property of the POST protocol is demonstrated. That is, the consensus reaching process (i.e., voting) will not be blocked due to the fail/restart of the nodes. To be more explicit, there is no “partial consensus” (i.e., either *reached* or *cancelled*) as long as the failed nodes can eventually recover. In the context of transactions, it is also referred to as *commit* and *abort*. In distributed computing, this non-blocking property is also called *liveness*. Note that POST works with POET in parallel to collect consensus.

As demonstrated in [131], there are two necessary and sufficient conditions to ensure a non-blocking protocol:

- C1 Between commit and abort, there exists no such state which could help us to make a decision.
- C2 There is no direct link between an indeterministic¹ state and a committed state.

The proof states as follows. To verify C1, let’s assume the node P who initiates the transaction fails after $(m - 1)$ other nodes have validated P ’s request. That is, total m nodes, including P itself, have verified the transaction before P is failed and restarted. There are two cases to consider:

- If $m \leq \frac{N}{2}$, then P has not committed anything because the execution is incomplete according to Line 14 of Protocol 1. In other words, the state always indicates an *abort*, and the restarted P node will simply forward the transaction to fellow nodes to start the validation.

¹An indeterministic state is defined as a state from which no final decision can be made.

- If $m > \frac{N}{2}$, then P should have persisted the change to the disk according to Line 21 of Protocol 1. In this case, when P restarts, it will definitely lead to a *commit* status.

Therefore, we see that no matter how many nodes have verified the transactions requested by P before the latter fails, for each possible case, there is only one possible outcome. That is, we never need to decide a commit or abort operation given a specific case. C1 is thus satisfied.

It is straightforward to verify C2 because POST does not exhibit an indeterministic state. Once P restarts from failure (i.e., crash), it simply checks whether the transaction is stored in the persistent storage. If so, P will mark the transaction completed—a “commit” state; otherwise, P will resend everything and restart the consensus procedure—an “abort” state. C2 is thus satisfied.

2.3.4 Overhead Reduction

Computation and Communication. The POST protocol is carefully crafted to control the overhead of the block validation and replication process. **First**, we check the shared storage (i.e., Line 2 of Protocol 2) before starting the block validation process. That is, we start accessing the compute nodes only when the block is not validated and appended already in the persistent medium (i.e., shared storage). **Second**, to avoid intensive communication, during the consensus aggregation process, we leverage the shared storage to avoid extensive peer-to-peer communication among the compute nodes to reach the consensus. Besides, we stop the block validation process as soon as 51% of compute nodes provide consensus, as shown in Line 7 in Protocol 2. If we can not achieve consensus from the first 51% of

nodes (i.e., $\frac{N}{2} + 1$) and need to continue the block validation process for the rest of the nodes (i.e., $N - (\frac{N}{2} + 1)$), maximum $N + 1$ messages will be exchanged in total between the compute nodes and the shared storage. **Finally**, in the persisting step, the shared storage's blockchain accessed only once to persist the new block after attaining the 51% consensus, as shown in Line 22 of Protocol 1.

Storage. The POST protocol assumes the compute nodes are disk-less and therefore addresses the memory constraints of compute nodes in HPC systems through the *push-pull* mechanism. That is, compute nodes hold only a few hundred (e.g., 100-200) of recent blocks instead of the entire blockchain that occupy only a limited space of the memory to continue the block validation process, whereas the shared storage secures the entire blockchain as an extra layer of backup (i.e., *push method*). In any case, if the in-memory blockchain of a compute-node is unable to validate a block, the nodes can pull at least the latest block from the shared storage to synchronize (i.e., *pull method*). This is because, in the blockchain, each block contains the immutable hash-id of the parent block that always helps in finding back the parent blocks from the fellow compute nodes or the shared storage. To circumvent further memory burden in a compute node, we always remove the most unused block from a node's in-memory blockchain before appending a new block using Protocol 3, as shown in Line 25 in Protocol 1.

2.3.5 Reliability

The question may arise that what if the shared storage gets compromised? The shared storage does not hold the single authority to validate a block. This is because shared storage can not create a block and only stores a block with a hash-ID generated from a compute

node. Although in case of more than 50% nodes failure, shared storage attempts to validate a block, during the block appending process in its persisted blockchain, it always uses one of the hashes of the compute-nodes who provided valid *hashed-votes*. The honest compute nodes easily can verify, if the shared storage tries to forge the persisted blockchain or a block's hash or tries to validate a block against the forged blockchain.

For authenticity, we take the conventional public-private key authentication approach. When a new node validates a block, it generates its (i.e., block) *hashed-vote* using its (i.e., node) private key. Besides, when a new node joins the network, its public key is already known to the other nodes. Therefore, the fellow nodes can always verify the authenticity of a block by re-generating the hash based on the public key. If shared storage tries to forge the persisted blockchain stored in the disk, it needs to re-compute the hashes of each block using the private key of the respective compute nodes, thanks to the immutability property of the hashing mechanism. A slight change in a block's hash will impact the hashes of all the blocks in the entire blockchain. As the private keys of the compute nodes are unknown to the shared storage, it is quite impossible for it (i.e., shared storage) to forge the blockchain.

Another question is what if the consensus aggregation in shared storage creates congestion or becomes a performance bottleneck? This is alleviated by following a customized network queue that carefully manages the blocks of transactions in proper order. We will discuss more details in §2.4.3.

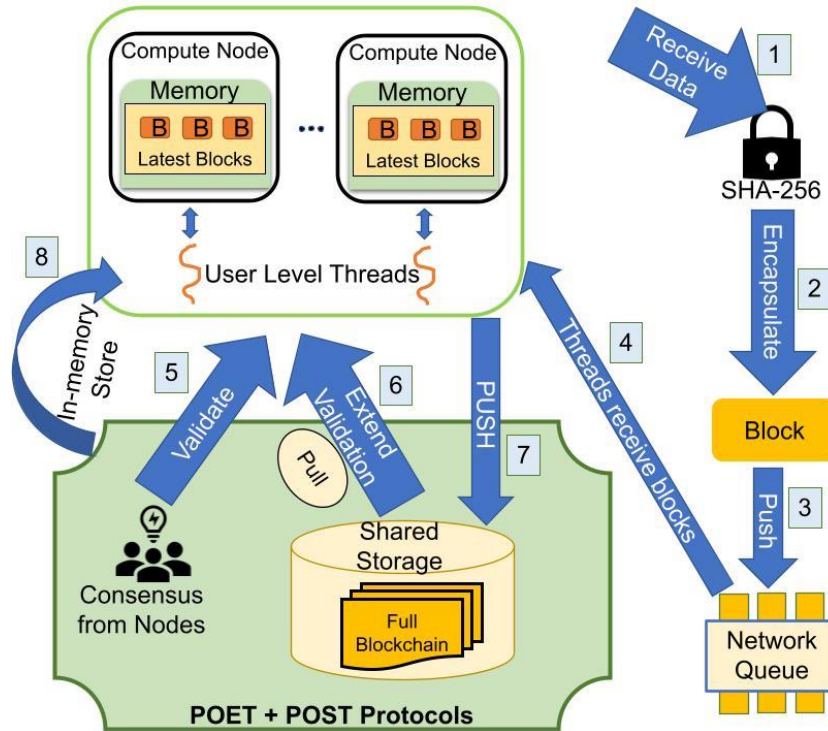


Figure 2.7: Overview of system interaction in current SciChain implementation.

2.4 System Implementation

2.4.1 Overview

We have implemented a prototype system of the proposed blockchain architecture and consensus protocols with Python. At this point, we only release the core modules of the prototype; some complementary components and plug-ins are still being tested with more edge cases and will be released once they are stable enough. The source code is currently hosted on Github. Figure 2.7 illustrates the overview of the implemented architecture of the proposed SciChain. The prototype system currently runs on a high-performance computing (HPC) cluster consisting of 58 nodes where (i) each node is equipped with 32 cores; hence each node can be emulated with up to 32 individual nodes through user-level threads, (ii)

ledgers are stored on in-memory, (iii) shared storage's ledgers are stored as files.

As shown on Figure 2.7, newly received transactions from the data collector are first encrypted using SHA-256 algorithm [128] (step 1). Then, the transactions are encapsulated in a block by the respective nodes (step 2) before being pushed into a queue (step 3), from which the queued blocks are propagated across the network (step 4). Finally, the blocks are validated both in compute nodes (step 5) and optionally, in the shared storage (step 6) (i.e., *Pull*) to achieve POST consensus before storing them in the shared storage (step 7) (i.e., *Push*) and the compute nodes (step 8). It should be noted that the shared storage takes part in the validation process (i.e., step 6), only if more than 50% of compute nodes are unable to provide consensus (i.e., step 5).

As proposed in [163], each blockchain system can be decomposed into four main components: ledger, consensus, cryptography, and smart contract. Because the work in this chapter focuses on ledger and consensus, we take SHA-256 [128] as the cryptography function for the chained blocks. The current system implementation does not support smart contract: we wrap up the applications using pseudo transactions such that the prototype can take in a variety of applications. The remainder of this section focuses on the ledger and consensus implementation.

2.4.2 Node Threads

A unique user-level thread is generated for controlling each node. All the properties of the node, such as node ID, a pointer to the local file, and node type (storage or sensor) are stored in a Node object. More details about the data and storage structure will be discussed

in §2.4.4.

The nodes in our system prototype are controlled by user-level threads spawning transactions in random orders with a fixed time interval. The threads are responsible for packing the transactions into blocks; each block comprises about a thousand transactions such that the launching overhead is amortized and negligible to each transaction. The blocks are then pushed into the network queue that will be discussed in the following section §2.4.3. The nodes are also responsible for validating the blocks.

2.4.3 Network Queues

All of the newly submitted blocks by the nodes are pushed into a FIFO queue. Similarly to how the blocks of transactions are spawned, a block also periodically pops out from the head of the queue at a fixed time interval, which then is transferred to the connected nodes.

It could be argued that a queue might not deliver data at a sufficient rate to feed the network because a queue is a linear data structure that can hardly be parallelized for the sake of scalability. This is alleviated by the following two approaches in our implementation. First, we adjust the time interval to more substantial than the latency for the queue to pop an element. In doing so, the overhead from the queue itself is masked completely. Second, we implement the queue using a loosely-coupled linked-lists such that the queue can be split and reconstructed arbitrarily.

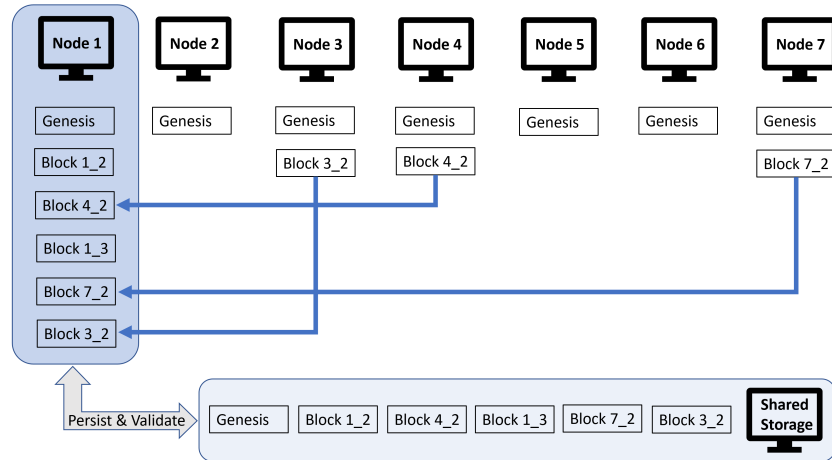


Figure 2.8: Data structures of SciChain transactions. The shared storage is in practice a cluster of physical nodes.

2.4.4 Data and Storage Models

The data structure for the proposed SciChain ledger is a linked list and stored in a hash table where each tuple corresponds to a block, which references a list of transaction records stored in another table, similar to the traditional relational database tables. In each hash table, a corresponding hash acts as a primary key that helps in identifying a block or a transaction.

Figure 2.8 shows a concrete example of the structure to store the blockchain on a specific node (i.e., Node 1). The block *genesis* indicates the very first block of the blockchain for that node. Each cell under the node represents a block. For example, under Node 1, Block 1_2 indicates the second block created by Node 1 and is a child of the genesis block. Block 1_2 is also the parent of Block 4_2, which is created by Node 4. Block 4_2 is appended as the child of Block 1_2 when Node 4 successfully appends this block and broadcasts it to other nodes. Each block contains a list of transactions stored in a hash table, where each transaction has a unique hash-id. In addition to the inter-compute-node propagation, the new block is also persisted and validated on the *Shared Storage* Node, which is the

bookkeeper of the ground truth of the distributed blockchains. It should be noted that the shared storage persists the entire blockchain in the disk, whereas the compute nodes only need to keep the most recent blocks within a limited space of the memory.

2.4.5 Data Collection and Query

It should be clear that SciChain is not responsible for *collecting* the raw provenance data, but only managing the provenance data in a trustworthy manner backed by blockchains. A separate process, or an interface from other subsystems such as filesystems or operating systems, is expected to import and transfer the raw provenance data (i.e., the input) to SciChain. A close analog is an RDBMS: although all RDBMS provide an interface to import the data from other formats (e.g., comma-separated values, spreadsheets), how to gather or formulate those data in the raw format is not a concern of the RDBMS. However, once the data is imported into the RDBMS, the latter should guarantee various properties such as reliability and atomicity (for transactions, for example).

Similarly, for queries, SciChain can leverage any external provenance query manager to help with submitting and fetching data. However, SciChain comes with a *Query Utility* that assists the external query manager in searching and optionally with re-verification of the data stored in SciChain before forwarding it (i.e., data).

2.5 Evaluation

2.5.1 Experiment Setup

2.5.1.1 Testbed

All experiments with the proposed system are carried out on a high-performance computing (HPC) cluster comprised of 58 nodes interconnected with FDR InfiniBand. Each node is equipped with an Intel Core-i7 2.6 GHz 32-core CPU along with 296 GB 2400 MHz DDR4 memory; hence each node can be emulated with up to 32 nodes through user-level threads. There is no local disk on compute nodes, which is a typical configuration on HPC systems; yet, a remote 2.1 PB storage system is available managed by GPFS [126]. Each node is installed with Ubuntu 16.04, Python 3.7.0, and NumPy 1.15.4. We deploy the system prototype mostly on 100 cores except for the scalability test, where we use 1,024 cores.

2.5.1.2 Evaluated Systems

We evaluate the SciChain prototype against two other blockchain systems. The first blockchain is a *Conventional Blockchain* system deployed to a shared-nothing cluster comprised of 10 nodes interconnected with Ethernet. Each node in the cluster is equipped with an Intel Core-i7 2.6 GHz 32-core CPU along with 128 GB 2400 MHz DDR4 memory. The second blockchain is a *Memory-only Blockchain* deployed to the same cluster (i.e., HPC cluster with 58 nodes) same as the proposed system with high-performance networking interconnections (i.e., InfiniBand, RDMA) without any persistent storage; this is not a practical

solution due to the lack of data persistence but is considered as the performance upper bound of the proposed *SciChain*. We implement all three blockchain systems in Python and make a reasonable effort in optimizing all of them.

2.5.1.3 Workload

To demonstrate practical data reproducibility, each node requires to perform a validation for a simple scientific experiment. More specifically, each node needs to re-produce a solution for a puzzle with given metadata (nonce number, target difficulty to solve the puzzle) before forwarding the vote to the peers to reach consensus to persist the validated metadata in the *SciChain* as an immutable reliable source for provenance records.

For micro-benchmarks, we use YCSB workload [156] commonly used for transaction evaluation. We map each metadata record for the scientific experiments in a YCSB transaction format before starting the validation process. On average, each block contains about a hundred transactions in our experiments. We deploy more than one million transactions (1,036,303) to the system prototype and compare it to the other two baseline systems.

For real-world applications, The testing workload is derived from a trace of the FusionFS [167] filesystem initially deployed to a 1,024-node cluster at the Argonne National Laboratory. The trace includes four real-world scientific applications: Plasma Physics, Turbulence, AstroPhysics, and Parallel BLAST [100].

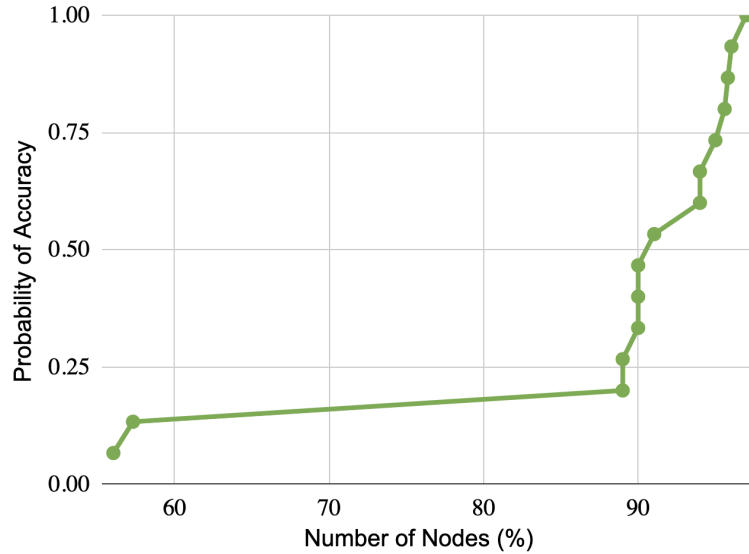


Figure 2.9: POST guarantees more than 50% of nodes always hold valid ledger.

2.5.2 Reliability and Trustworthiness

This section demonstrates that the proposed POST consensus can be achieved by more than 50% of participants. In other words, we want to show that introducing the shared storage as an additional node does not reduce the portion of good compute nodes to under 51%. To put it in another way, we want to verify, if we try to inject tampered blocks (i.e., provenance records) randomly, whether the new consensus protocol leads to the same longest valid blockchain compared to other consensus protocols such as PoW from the compute nodes. Note that we prove the safety and liveness in Section 2.3.3; we will experimentally verify the trustworthiness and reliability here.

To demonstrate the trustworthiness and reliability, we run the system prototype given random transactions for 10 minutes, and repeat the execution 15 times, as shown in Figure 2.9. All of the 15 executions lead to more than 50% nodes holding the correct blockchains: 13 out of 15 executions yield more than 89% validity while two executions exhibit lower ratios because we terminate the execution once more than 50% of nodes hold the correct blocks.

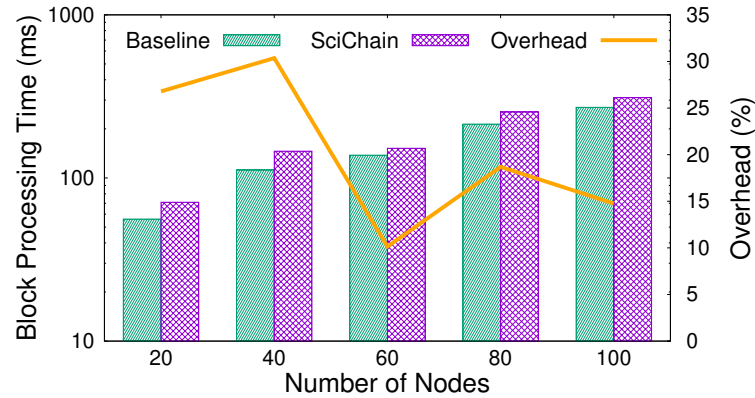


Figure 2.10: Scalability of SciChain against the baseline during the block with provenance data storing time.

The bottom line is that we need to guarantee at least 51% of nodes' data are not tampered with, which is the case.

2.5.3 Overhead

This section reports the provenance overhead incurred by the proposed SciChain. The memory-only baseline system persists the data provenance to the disk with no security or audibility guarantees. We turn on SciChain atop the baseline and measure the end-to-end block processing time compared to the baseline performance. We run both systems in parallel given random transactions for 10 minutes, repeat the execution five times, and report the average.

As shown in Figure 2.10, we observe the overhead incurred by the proposed SciChain is noticeable (25% – 30%) at small scales of 20 and 40 nodes. This is the price we have to pay to achieve high security. The good news is, however, the overhead is reduced to under 20% on larger scales; in particular, the overhead is only 15% on 100 nodes. That is, unlike conventional provenance systems whose overhead increases proportionally to the

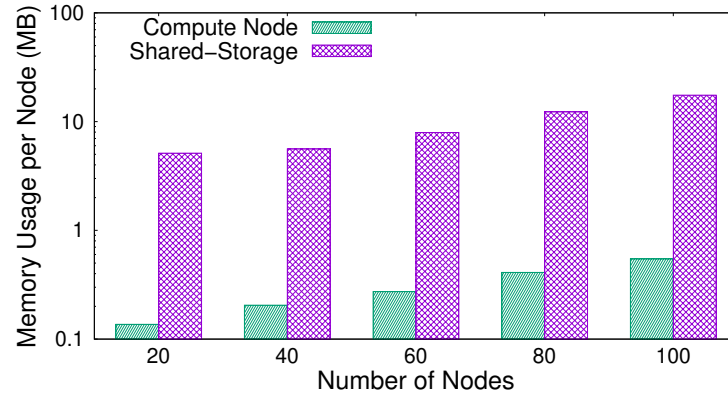


Figure 2.11: Memory overhead of SciChain. Push-Pull mechanism minimizes in-memory load while ensuring the reliable persistence through shared-storage.

Table 2.2
PERFORMANCE OF PUSH MECHANISM TO PERSIST IN SHARED STORAGE.

No. of Transactions	Time required (ms)
1	0.2
10	1.025
100	12.89
1000	122.5
10000	906.5

number of nodes, SciChain’s overhead ratio does not significantly increase, thanks to the POST consensus. In the POST mechanism: (i) a transaction gets persisted only once after it gets qualified by majority votes; (ii) the larger the scale, the fewer chances the POST will redirect the validation to the shared storage that incurs reasonable, if not negligible, overhead.

As we can see in Figure 2.11, a compute node leverages a negligible amount of memory on 100 nodes (i.e., below 1 MB). However, the shared storage does incur more overhead (i.e., ~ 18 MB) compared to a compute node. This is because shared storage persists the entire blockchain, whereas a compute node only keeps a few hundred blocks in-memory. The *push-pull* and *consensus aggregation* mechanisms in POST reduce the pressure both on communication and storage with the scaling of the nodes.

Table 2.3
PERFORMANCE OF PULL MECHANISM TO SYNCHRONIZE NODES.

No. of Transactions	Time required (ms)
1	1.54
10	4.96
100	63.62
1000	449.5
10000	4365

We analyze the *PUSH-PULL* mechanism performance at 100 nodes cluster with various numbers of transactions. Note that the *PUSH* guarantees the persistence of all the blocks in shared storage, while *PULL* assists the compute nodes to synchronize the local ledgers with the shared storage in case of any inconsistencies. Table 2.2 reports the time required by *PUSH* mechanism to persist various numbers of transactions in the shared storage after the validation process. It is noticeable that to persist a large number of transactions (i.e., ten thousand transactions), SciChain only incurs less than a second. As shown in Table 2.3, to synchronize the compute nodes' local ledgers with share storage through the *PULL* mechanism, SciChain takes less than five seconds for ten thousand transactions. Note that the *PUSH* touches only the shared storage once for a transaction whereas the *PULL* is to update all the compute nodes' memory for a transaction; thus, the latter (i.e., *PULL*) incurs more time than the former (i.e., *PUSH*). However, in practice, only a few nodes require to sync with the shared storage; hence, the *PULL* incurs much less time.

2.5.4 Scalability

We observe the scalability of the SciChain against two other baselines at various scales. We run all the systems in parallel with the same number of given random transactions for 10

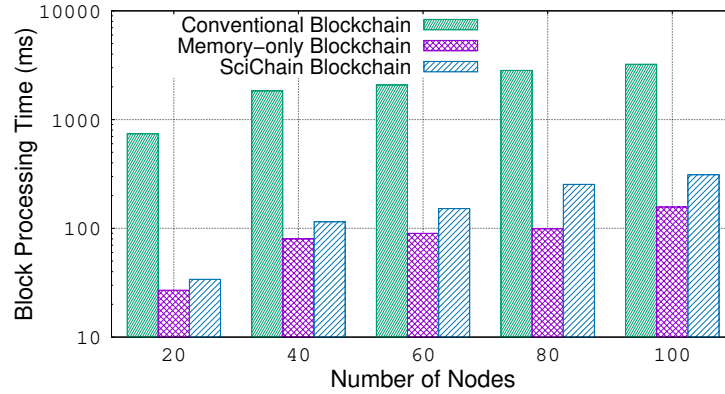


Figure 2.12: Scalability of three blockchain systems for block validation time during the reproduction.

minutes, repeat the execution 5 times and report the average. Figure 2.12 reports the performance of the three systems on 20-, 40-, 60-, 80-, and 100-node scales. The memory-only blockchain, as expected, achieves the highest performance (i.e., lowest processing time). The proposed SciChain with shared storage does not exhibit a significant slowdown than the upper bound; for instance at 100-node scale, the comparison is 157 *ms* vs. 311 *ms*, at the same order of magnitude. However, the conventional blockchain on 100 nodes appends a new block in 3,231 *ms*, significantly slower than SciChain. Specifically, SciChain shows significant speedup in performance compared with existing systems: $\frac{3,231}{311} = 10.4\times$.

We also compare the recent work [7] that solely depends on PoW. In PoR [7], each node takes 22 *ms* on average to process a block, whereas, in SciChain, each node takes only $\frac{34}{20} = 1.7$ *ms* on average at a 20-node scale. The reason behind the notable performance degradation in [7] is, unlike the POST protocol in SciChain, the protocol in [7] does depend on conventional PoW that requires constant adjustment (increment or decrement) in nonce; hence an impractical approach to applying in HPC systems.

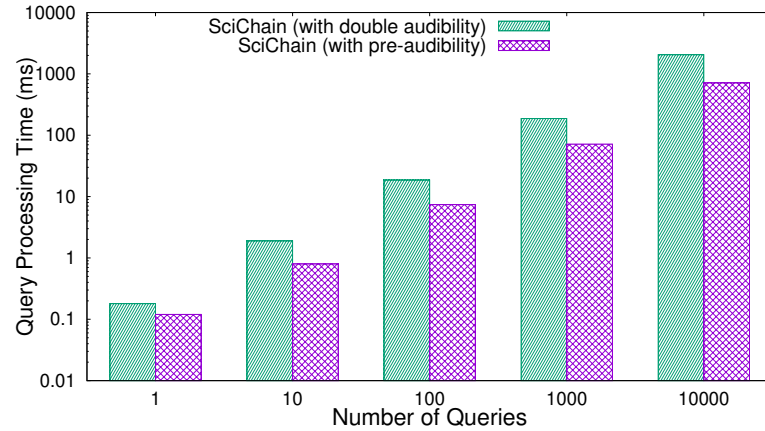


Figure 2.13: Comparing provenance query performance. SciChain with double audibility assists in verification of data again through decentralized protocol during the query. SciChain with pre-audibility relies on POST and POET protocols and doesn't require further verification during the query.

2.5.5 Query Performance

We test the performance of SciChain query utility on a cluster of 100 nodes with various numbers of queries, as shown in Figure 2.13. During this experiment, we switch on and off the double verification (i.e., audibility) on SciChain to measure the performance difference. When a query is submitted, SciChain with double-audibility support verifies the hashes of the stored data with the help of the decentralized protocols again before forwarding it (i.e., data). In contrast, the SciChain with pre-audibility support relies on the POST and POET protocols that validate the stored data earlier.

Figure 2.13 illustrates that even if we turn on double verification for proving further authentication of the data, SciChain incurs reasonable overhead compared to the SciChain with no further audibility support. Note that double verification is an optional task, and we can simply rely on the already stored data, which is always validated through decentralized protocols (i.e., POST and POET) before getting stored in SciChain.

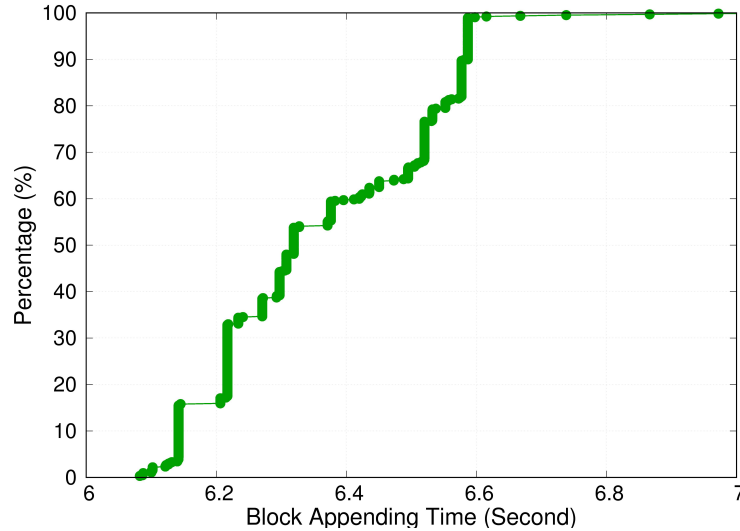


Figure 2.14: Latency distribution on 1,024 nodes.

2.5.6 Large-Scale Applications

We study a trace of the FusionFS [167] filesystem initially deployed to a 1,024-node cluster at the Argonne National Laboratory (ANL). Four real-world scientific applications are being traced: Plasma Physics, Turbulence, AstroPhysics, and Parallel BLAST [100], all of which run for five hours. These applications cover a broad spectrum of data I/O patterns exhibited by scientific applications [165]. We assign random numerical values to the I/O operations traced in FusionFS such that the SciChain prototype can take in the provenance in the same way as other blockchain systems. We execute the workloads by feeding the transactions into the system prototype deployed on 1,024 nodes—the same scale when the applications were executed at ANL.

As shown in Figure 2.14, the 1,024-node cluster appends a new block in 6–7 seconds. We do observe a few outliers (data points at 6.7 s and beyond along the X-axis), although the quantity is tiny: only four out of 1,024 nodes, less than 0.4%. The real difference is also insignificant, about 0.4 seconds out of seven seconds. It should be noted that each

block consists of about a hundred transactions. Therefore, on 1,024 nodes, the provenance time incurred by SciChain is sub-second—a reasonable, if not negligible, overhead for large-scale scientific applications in practice.

2.6 Summary

This chapter proposes a new blockchain consensus protocol, namely POST, to enable immutable and autonomous data provenance for scientific applications deployed to HPC systems. POST is implemented in a prototype system called SciChain, the first practical HPC-blockchain system towards trustworthy data provenance service in HPC. The effectiveness and efficiency of POST are analyzed, and experimentally demonstrated through both micro-benchmarks and real-world applications on up to 1,024 nodes. Experimental results showed that SciChain guaranteed trustworthy data provenance while incurring orders of magnitude lower overhead than existing solutions.

CHAPTER 3

BAASH: LIGHTWEIGHT, EFFICIENT, AND RELIABLE BLOCKCHAIN-AS-A-SERVICE FOR HPC SYSTEMS

3.1 Introduction

3.1.1 Motivation

Distributed consistent caching. Exascale systems are likely to have extreme power constraints; yet, follow an expensive approach to move data anywhere, necessarily near the processors. However, the problem can be mitigated by replicating [51] or caching data distributedly through a distributed ledger. This is especially useful when the scientific workflows are coupled, as illustrated in Figure 3.1, such as multi-scale, multi-physics turbulent combustion application S3D [32], where the data moving cost often grows exponentially. Besides, performance in exascale systems will require enormous concurrency, requiring that data on each node be in the same state through synchronization leading to the eventual consistency for readers to see data from heterogeneous sources, which is desirable in modern large scale HPC systems [9].

Provenance tracking. On top of that, extreme-scale HPC systems (e.g., Cori [139]) rely on data provenance to reproduce and verify the scientific experimental results generated during the application executions. Essentially, data provenance is a well-structured log for efficient data storage along with an easy-to-use query interface. Data provenance is

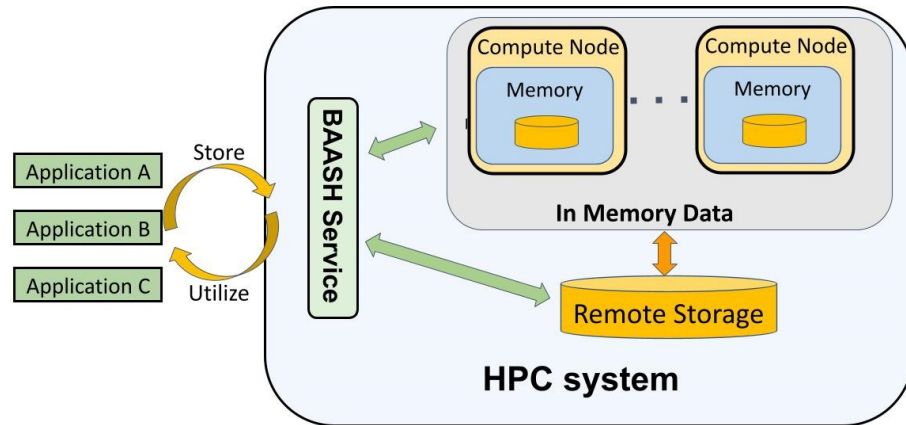


Figure 3.1: BAASH service is useful in a typical data movement workflow in an HPC environment.

conventionally implemented through file systems [130,168] or relational databases [60,90]. However, none of the current efforts exhibit the *immutability* property that guarantees the reliability of the provenance history.

Data fidelity. Over and above that, data fidelity is of prominent importance for scientific applications deployed to HPC systems, as the data upon which scientific discovery rests must be trustworthy and retain its veracity at every point in the scientific workflow. Silent data corruption has been a critical problem [47, 52] that leads to data loss or incorrect data, impacting the application performance at an extreme scale. Extreme-scale workflows are often long-lasting and dependent on intermediate results collected from a data source or other applications, as shown in Figure 3.1. Therefore, those workflows typically require careful verification before caching the data in-memory or persisting them in remote storage. There have been noticeable incidents [56, 132] where the error rates grow exponentially as the system reaches extreme scale. Besides, there have been more than enough incidents of data falsification and fabrication, causing the withdrawal of scientific publications and other consequences. To this end, developing trustworthy data services for scientific computing and HPC systems has been recently incentivized by various federal funding agencies, such as the National Science Foundation [109] and the U.S. Department of Energy [49].

Table 3.1
SUMMARY OF LIMITATIONS OF PRESENT BLOCKCHAIN-BASED PROVENANCE SYSTEMS.

Features	ProvChain [95]	SmartProv [117]	LineageChain [121]	IMB [7]	SciChain [8]	BAASH
Support diskless nodes	×	×	×	✓	✓	✓
Light computation	×	×	✓	×	✓	✓
Light communication	✓	✓	×	✓	✓	✓
MPI Compatibility	×	×	×	×	×	✓
Parallel processing	×	×	×	×	×	✓
Parallel resiliency	×	×	×	×	×	✓
Realtime fault monitoring	×	×	×	×	×	✓
500+ cores Scalability	×	×	×	×	×	✓

Blockchain-as-a-service for HPC. We argue that all of the aforementioned services could be built upon or improved by a decentralized consensus protocol—the very core technical innovation of blockchains. In essence, a blockchain can enable data synchronization in HPC systems with strong reliability in an autonomous fashion. The Oak Ridge National Laboratory has released a white paper [3] discussing a wide range of potential applications that can benefit from blockchains on the Oak Ridge Leadership Computing Facility. Recent studies [6–8, 39, 94, 95, 117, 121] showed that blockchains could be leveraged to provide a variety of services on HPC systems.

3.1.2 Challenges

While blockchains have drawn much research interest in many areas, such as cryptocurrency [54, 61, 81] and smart government [110], the HPC and scientific computing communities, although regarding resilience as one of their top system design objectives, have not taken blockchains into their ecosystems due to various (both technical and administrative) reasons, but most notably on the following two challenges: the shared-storage system infrastructure of HPC systems and the MPI programming model for scientific applications. By contrast, all the mainstream blockchain systems and frameworks assume the underlying

systems are shared-nothing clusters with the TCP/IP network stack (rather than MPI and ranks).

Specifically, although blockchain itself exhibits many research and application opportunities (comprehensive reviews available from [46,163]), one of the most impelling challenges for employing blockchains into HPC systems lies in the consensus protocols for the unique I/O subsystem. Existing consensus protocols used in mainstream blockchains are either based on intensive computation (the so-called proof-of-work, or POW, for instance) or intensive network communication (e.g., practical byzantine fault tolerance, PBFT), which are inappropriate for HPC in terms of both performance and cost, as shown in Table 3.1.

Nonetheless, some of the efforts [7,8], recently attempt to resolve some of the challenges to adopt blockchain or blockchain-like data caching in the HPC ecosystem. However, these systems either follow conventional POW [7]; hence, not appropriate for permissioned environment like HPC, or are not yet equipped with the most desired essential features as shown in Table 3.1, for instance, *(i)* extensive computation or communication free scalable consensus mechanism to establish trustworthiness among the distributed cached data, *(ii)* full-competent HPC protocol account for both the compute nodes and the remote storage accounting for the parallel file system (e.g., [126]) with reasonable overhead; *(iii)* parallel block processing, *(iv)* parallel consistency in the distributed ledger with lower latency, and lastly, *(v)* fault-tolerance support addressing MPI node failure that strengthens consistency in distributed ledger across the nodes. Hence, the state-of-the-art systems yet suffer from a lightweight strategy to deploy the blockchain-like resiliency in HPC infrastructure.

Although MPI brings a lot of opportunities to facilitate parallel processing in scientific computing, deploying blockchain with MPI is a very challenging job. In MPI, one rank failure brings down the entire communicator; hence, it raises a critical issue for a

blockchain service: one single failure would crash the entire blockchain service. Several fault-tolerance approaches [24,64] (mostly through software exception handlers) have been designed to address MPI failures. However, no work exists to address the unique requirement of blockchains: a crashed blockchain node has to restart from scratch and verify all the hash values between every adjacent pair of blocks (i.e., there is no such a concept of *checkpoint*, unfortunately).

3.1.3 Our Contributions

In this work, we design a new blockchain framework for HPC environments, deployed as a middleware, namely BAASH, as illustrated in Figure 3.1. BAASH serves as a *distributed trustworthy ledger* fully compatible with the shared-storage infrastructure of HPC systems. The proposed framework overcomes the shortcomings of the state-of-the-art blockchain systems by completely *eliminating the resource-intensive requirements* through a set of specially crafted protocols. Moreover, BAASH is equipped with a *parallel processing layer* compatible with MPI, which enables BAASH to deliver low-overhead and scalable performance. The protocols assure *parallel resiliency* and account for both the compute nodes and the remote storage (e.g., [126]) with reasonable overhead. Therefore, a system implementing the proposed protocols can be smoothly deployed to an HPC infrastructure. On top of that, the *realtime fault monitor* co-designed with BAASH takes care of MPI's rank failures and thus supports the highly-desired (*eventual*) *consistent caching* across the compute nodes.

To summarize, this chapter discusses the following contributions:

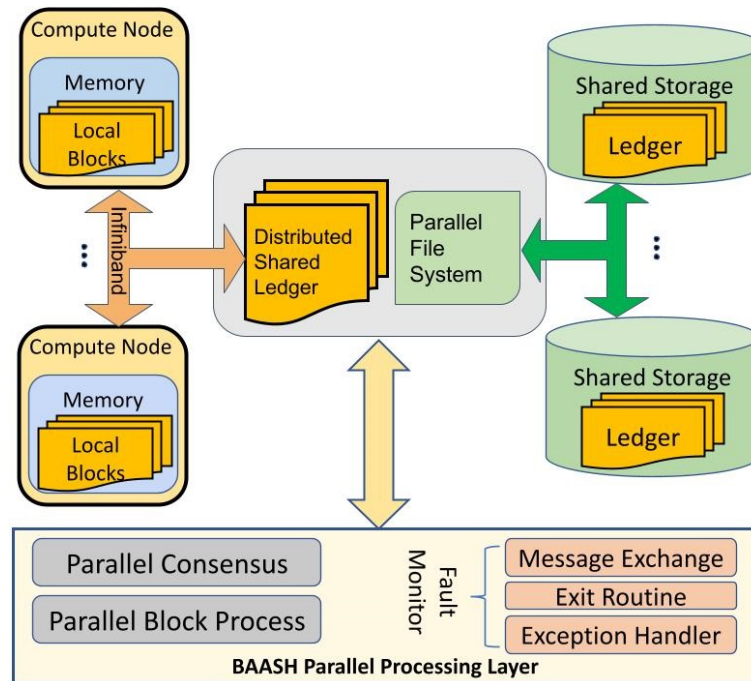


Figure 3.2: Overall architecture of BAASH on HPC systems.

- We design a set of HPC-specific scalable consensus protocols to facilitate a parallel block processing to provide a lightweight distributed in-memory and shared-storage resiliency support;
- We develop a reliable fault monitoring mechanism to ensure consistent BAASH service that can tolerate MPI rank failures to ensure the availability of the BAASH service;
- We implement a system prototype of the proposed consensus protocols and parallelization approaches with OpenMP and MPI; and
- We carry out an extensive evaluation of the system prototype with millions of transactions on an 500-core HPC platform and experimentally demonstrate the effectiveness of BAASH.

3.2 BAASH Design

3.2.1 Architecture

Figure 3.2 illustrates the high-level overview of our envisioned distributed BAASH system, which is deployed to a typical HPC system. Note that some customized HPC systems may have node-local disks, although this work assumes that the compute nodes are diskless. For instance, a top-10 supercomputer called Cori [139] located at Lawrence Berkeley National Laboratory does have local SSD storage, namely, the burst buffer. However, the burst buffer is not a good candidate for ledgers because it is not designed for long-term data archival; its only purpose is to alleviate the I/O pressure for those I/O-intensive applications by caching the intermediate data locally. Besides, because scientific applications do not time-share the resources, the ledgers stored on the local storage (e.g., burst buffer) are usually purged after the job execution (for performance and security reasons). In other words, even if a ledger can be technically persisted to a local disk in some scenarios, that persistence is not permanent, which motivates us to develop a secondary ledger and validator on the remote storage. Specifically, four key modules of our envisioned system are highlighted in Figure 3.2: a resilient distributed ledger, a parallel consensus protocol, a parallel processing layer, and a fault monitor. We will discuss each of them in more detail in the following.

3.2.1.1 Resilient distributed ledger

The first module is a resilient distributed ledger implementation optimized for high-performance interconnects (e.g., InfiniBand) and protocols (RDMA) across compute nodes. Because all

compute nodes fetch and update the ledger with few latest blocks only in volatile memory (or in persistent local storage but with a short lifetime—purged after the job is complete) with minimum overhead, there must be at least one permanent, persistent storage (which cannot be compromised) to store back the ledger replicas in case of catastrophes (e.g., more than 50% of compute nodes crash and lose their ledgers). Note that the memory-only compute nodes are not necessarily less reliable than those with persistent storage; yet, the data stored on memory would get lost if the process that initiates the memory is killed. Note that remote storage is a cluster of nodes in a typical high-performance computing ecosystem that makes the shared blockchain highly reliable against any unexpected catastrophe.

3.2.1.2 Scalable in-memory & shared-storage consensus protocol

The second module is a consensus protocol that supports attaining consensus in parallel (neither costly computational overhead nor extensive network communication) to achieve the highest possible throughput through in-memory blockchain support of the compute nodes and the resilient remote ledger. In BAASH, a block is validated with two consecutive steps. *First*, the block is validated through a parallel mechanism with in-memory blockchain support in each node. To minimize the block validation time, we avoid any traditional serialized block processing mechanism followed by state-of-the-art blockchain systems. *Second*, if the majority of the compute nodes (i.e., at least 51%) are unable to reach a consensus about the validity of a block, the remote storage then participates in the block validation process with reasonably minimum overhead. We discuss more details in Section 3.2.2.

3.2.1.3 Parallel processing layer

The third module is the parallel processing layer between the diskless compute nodes and the remote persistent storage. The layer has three purposes: (i) It injects the parallel distribution mechanism for disseminating independent blocks with transactions into the logically divided cluster of compute nodes, (ii) It facilitates parallel block processing and achieving consensus without a communication-intensive peer-to-peer mechanism, and (iii) The ledger persisting process in the remote storage (i.e., shared-blockchain), as well as synchronization with the distributed nodes, can continue independently in parallel, without interfering with the block validation. More details about this layer is explained in Section 3.3.4 and Section 3.3.5.

3.2.1.4 Fault monitor

The fourth module is a real-time fault tolerance mechanism implemented with three inevitable checkpoints:

1. Message exchange among peers: Check the communication status between the sender and receiver during each message exchange.
2. Exit routine of each process: An exit routine that checks the status of each node when the node finishes a validation process.
3. Exception handler: Set checkpoints around the node validation process to catch any unexpected exceptions.

Fault monitor also restores the BAASH service if any of the cases occur while handling the

Protocol 5 Parallel transaction manager

Require: Clusters C where the i -th cluster is C^i ; Transactions T where the i -th transaction is T^i ; Entities E where the i -th entity is E^i ; Remote storage R ; a new block b .

Ensure: No duplicate processing of T^i .

```

1: function TXN-PROCESS( $b, C, R, E$ )
2:   for  $T^i \in T$  do
3:     if  $T^i$  is not locked then
4:       Create hash  $H_T^i$  with timestamp  $s$  and entity-id  $E^i$ 
5:       Push in a block  $b$ 
6:     else
7:       Push  $T^i$  to transaction wait queue
8:     end if
9:     if time  $\geq t$  then
10:      BAASH-Consensus( $b, C, R, E$ ) ▷ Protocol 6
11:    end if
12:  end for
13: end function

```

block validation process by the shared storage's ledger. We discuss more details about the fault monitor in Section 3.3.6.

3.2.2 BAASH Protocols

To overcome the limitations of the conventional protocols (i.e., PoW and PBFT), as a co-design of the proposed BAASH blockchain framework for scientific applications, we propose a set of protocols. Protocol 5 coordinates parallel processing of transactions of blocks. The Protocol 6 assists in managing consensus of blocks in parallel through the distributed consensus managers. The block validation process is managed by the Protocol 7. Finally, the Protocol 8 helps in managing the resilient distributed ledger by storing the validated block in nodes' memory and persisting in the shared storage. We will discuss each protocol in detail in the following sections.

3.2.2.1 Parallel transaction manager

Protocol 5 aims to ensure the distinctive operation of transactions during the parallel process. The BAASH engine leverages a queue that keeps track of the individual active and pending transactions. All the active transactions are locked to prevent duplicate transaction processing by creating a unique hash with the timestamp and the entity's address (e.g., node) that issues the transactions. When a transaction arrives, it (Protocol 5) first checks (Line 3) if the transaction is unlocked (i.e., pending) and creates a unique hash (Line 4) for it with the entities addresses and the current timestamp before pushing to a block (Line 5). The transaction batching process in a block continues until a specific time (t). The time (t) is the minimum latency over a network. Finally, Protocol 5 forwards the block to start the consensus process (Line 10).

3.2.2.2 BAASH consensus

Protocol 6 steers the entire block validation and persisting process. The *parallel processing module* (more details in Section 3.3.4) leverages this protocol to manage the equal dissemination of blocks among the clusters. At first, the BAASH engine creates a set of coordinators and logically distributes the compute nodes into a group of clusters to manage the smart load-balance of the workload that facilitates parallel block processing. Each cluster is managed by a coordinator (Line 3). Each cluster is then further split into a set of sub-clusters (Line 5) where each sub-cluster processes a block (Line 6). A sub-cluster dynamically get constructed with $3f + 1 + c$ number of nodes, where f is the maximum faulty nodes allowed in a sub-cluster, and c is the maximum number of attempts of nodes adjustment from a parent cluster (more details in Section 3.3.4). To be more specific, at

Protocol 6 BAASH consensus

Require: Clusters C where the i -th cluster is C^i ; Sub-clusters N where the i -th sub-cluster is N^i managed by a coordinator N_c^i ; Total nodes n in a sub-cluster; Entities E where the i -th entity is E^i ; a new block b ; hash list b_h^l for b ; remote storage R .

Ensure: At least 50% compute node list $agreedNodes$ who validate b both with local blockchain and with remote persistent ledger R_B .

```

1: function BAASH-CONSENSUS( $b, C, R, E$ )
2:   while  $|agreedNodes| \leq \frac{n}{2}$  do
3:     for  $C^i \in C$  do                                     ▶ In parallel
4:       Push a block  $b$  in  $C^i$ 's local queue
5:       for  $N^i \in C^i$  do                                   ▶ In parallel
6:          $N_c^i \leftarrow \text{Validation}(b, N^i, E)$           ▶ Protocol 7
7:       end for
8:     end for
9:   end while
10:  if  $|agreedNodes| \leq \frac{n}{2}$  then
11:    if  $R$  validates  $b$  then
12:      Generate hash  $b_h$ 
13:      Persist-in-Storage( $b, C, R$ )                          ▶ Call Protocol 8
14:      Release block  $b$  from active queue
15:    else
16:      Push the block to pending queue
17:    end if
18:  else
19:    Pick a hash  $b_h$  from  $b_h^l$ 
20:    Persist-in-Storage( $b, C, R$ )                              ▶ Call Protocol 8
21:    Release block  $b$  from active queue
22:  end if
23:   $N^i \leftarrow \emptyset$                                    ▶ Deconstruct the sub-cluster
24: end function

```

first, $3f + 1$ nodes in a sub-cluster attempt to validate a block. If more than 50% nodes in a sub-cluster fail, the coordinator attempts a maximum of c times to add an idle node from the parent cluster to continue the validation.

If a validation process exceeds the total limit (i.e., c) while attaining consensus from more than 50% nodes, the coordinator involves the remote storage to come forward to provide consensus (Line 11). Possible node failures would be either software exceptions, false validation, or crashes. If the block is valid the protocol picks a hash from the list of its (block) validators (Line 19) or create a hash within remote storage (Line 12) before persisting it (i.e., block) through the help of Protocol 8 (Line 13 and 20). Finally, the sub-

Protocol 7 Block validation

Require: Sub-clusters N where the i -th sub-cluster is N^i ; Nodes n where the i -th node is n^i . Entities E where the i -th entity is E^i ; a new block b ; hash list b_h^l for b where the b_h^i is the hash from n^i .

Ensure: Block b contains valid transactions T before persisted.

```

1: function VALIDATION( $b, N^i, E$ )
2:   for  $n^i \in N^i$  do ▷ In parallel
3:     if  $n^i$  validates  $b$  with affected  $E$  then
4:        $agreedNodes \leftarrow agreedNodes \cup n^i$ 
5:       Create hash  $b_h^i$ 
6:        $b_h^l \leftarrow b_h^l \cup b_h^i$ 
7:     else
8:        $b_h^i \leftarrow NULL$ 
9:     end if
10:  end for
11:  Return  $agreedNodes, b_h^l, b$ 
12: end function

```

cluster (N^i) is deconstructed again at the end of a block processing (Line 23) to leverage the free nodes for the next block processing. The time complexity of this protocol is $O(|C|)$, which is significantly lower than the existing PBFT algorithm (i.e., $O(|C|^2)$).

3.2.2.3 Block validation

Protocol 7 works on block validation. The validation process checks whether a block with transactions is valid (e.g., legitimate provenance operations). When a node in a cluster receives a block, it starts checking the transactions in the block against the entities that are affected, as shown in Line 3. The entities consist of any files, nodes, or any other external source, etc.

A node provides a vote after validating the block (Line 4). A validator (i.e., Node) creates a hash with its private key and the current timestamp after processing all the transactions in a block (Line 5). The validator sets the hash with *Null* value if a block is invalid (Line 8). Fi-

Protocol 8 Persist in storage

Require: Compute nodes n where the i -th node is n^i ; n_B^i the local blockchain on n^i ; Clusters C where the i -th cluster is C^i ; a newly mined block b ; remote storage R ; H_T the hash table that contains hashes of blocks stored in remote storage; R_B the blockchain copy on the storage;

Ensure: Validate b , store it to C , and persist it to R

```

1: function PERSIST-IN-STORAGE( $b, C, R$ )
2:   for  $C^i \in C$  do
3:     for  $n^i \in C^i$  do
4:       if  $b \notin n_B^i$  then
5:          $n_B^i \leftarrow n_B^i \cup b$                                  $\triangleright$  store in local chain
6:       end if
7:     end for
8:   end for
9:   if  $b \notin H_T$  then                                        $\triangleright$  Only one look-up is needed
10:     $R_B \leftarrow R_B \cup b$                                     $\triangleright$  store in shared chain
11:   end if
12: end function

```

nally, the Protocol 7 takes advantage of *consensus manager* (more details in Section 3.3.5) to aggregate the votes (i.e., `agreedNodes`) and forwards the consensus along with the hash list (i.e., b_h^l) as well as the block with valid transactions (Line 11) to Protocol 6.

3.2.2.4 Persist-in-storage

Protocol 8 assists in managing the resilient distributed ledger by storing blocks in parallel to the nodes' local memory in all clusters (Line 2 and 3) while persisting blocks to the remote storage through a parallel file system (e.g., GPFS) (Line 10). Persisting in remote storage adds one more layer of reliability to the data on volatile memory. However, while attaining high reliability, the system should not exhibit significant overhead. We use a key-value model to store both the blocks and transactions both in the in-memory ledger and the remote storage. In the key-value data model, a hash represents a key. Therefore, when a node attempts to store a block in the storage node, it first looks up quickly in the storage to check whether the block is already stored. If the block hash is already in the storage, the

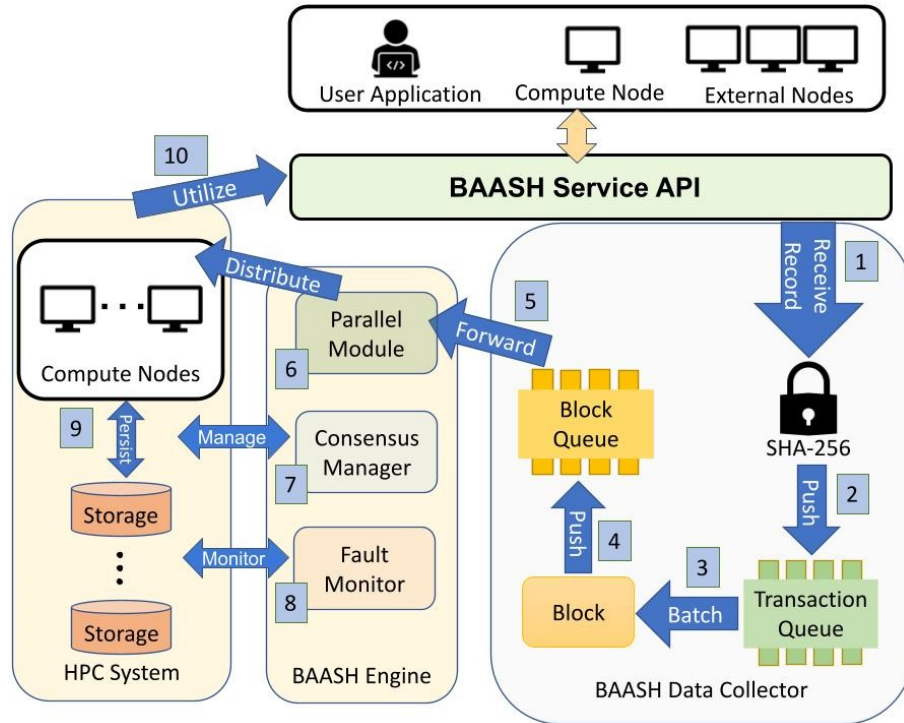


Figure 3.3: BAASH implementation with MPI and shared storage.

respective node does not attempt to access the remote storage further. This is addressed in Line 9, where we only need one look-up to check in the storage. This prevents touching the shared storage more than once when other nodes try to store the same block again.

This replication process is independent and hence, does not interrupt the overall performance of BAASH during the block validation. The theoretical time complexity of this protocol is $O(|C|)$, and $|C|$ could be a fairly large number (e.g., tens of thousands of cores in leading-class supercomputers [139]). Note that in HPC system, remote storage is managed through a distributed cluster of storage nodes; hence, failure of a single storage node does not lead to any loss to the full ledger stored in the remote storage. We will demonstrate the effectiveness of the protocol in the evaluation section.

3.3 System Implementation

We have implemented a prototype system of the proposed blockchain architecture in the user space and consensus protocols with Python-MPI. Although this chapter presents BAASH as a user library with a callable programming interface, BAASH can also be deployed at the system level through a wrapper. MPI is a message-passing application programming interface that allows us to develop our parallel processing layer. We use the mpi4py package [107] for leveraging MPI with Python. This MPI package for Python provides bindings of the MPI standard for the Python programming language, allowing any Python program to exploit multiple processors across machines. Typically, for maximum performance, each CPU (or core in a multi-core machine) will be assigned one single process or a distinct *rank*.

At this point, we only release the very core modules of the prototype. Some complementary components and plug-ins are still being tested with more edge cases and will be released once they are stable enough. Figure 3.3 illustrates the overview of the implemented architecture of the proposed parallel blockchain on MPI. The prototype system has been deployed to 500 cores on an HPC cluster.

As shown in Figure 3.3, new transactions received through BAASH service API from the nodes or any other sources (e.g., external HPC cluster or scientific workbench) are first hashed using SHA-256 [128] (step 1). Then, the transactions are pushed in a queue (step 2) to be batched or encapsulated in a block (step 3) and added to the block queue (step 4). The parallel processing module monitors both the clusters of nodes as well as the block queue. The parallel module receives a block (step 5) and distributes it whenever a cluster is available (step 6). Afterward, the blocks are validated first in compute nodes. The consensus manager collects the consensus of the block from the cluster (step 7), and if the

block is validated successfully, BAASH decides to append the block both in the in-memory ledger of the compute nodes and in the remote storage through a parallel file system (i.e., GPFS) (step 9). During the entire block validation process, a fault monitor keeps track of each node and takes the necessary initiatives to recover any node failure and reinstate the BAASH service (step 8). Once the data from a source is validated and appended in the distributed ledger, other applications can access the data as a reliable source through BAASH API (step 10).

3.3.1 Data Models

The data structure for the proposed BAASH ledger is a linked list stored in a hash table where each tuple corresponds to a block, which references a list of transaction records stored in another table. In each hash table, a corresponding hash acts as a primary key that helps in identifying a block or a transaction. All properties of a block (e.g., block ID, parent block hash, transactions list, and time stamp) and all properties of a transaction (e.g., transaction ID, transaction data, time stamp, and entities ID such as Node-ID, application-ID, file-ID) are encapsulated respectively in a `Block` object and a `Transaction` object at runtime.

3.3.2 Worker Nodes

In BAASH, transactions are first appended to the transaction queue, which is discussed in detail in §3.3.3. When the created transaction queue reaches a limit, the nodes encapsulate them (i.e., transactions) in a block. When a block is encapsulated with several transactions,

it is then pushed into a queue (discussed in §3.3.3) before it is broadcasted in the network.

The nodes are responsible for validating the blocks and sending consensus to the respective coordinators (MPI communicators) through the *parallel processing module* and distributed *consensus managers* (more details in Section §3.3.4 and §3.3.5). A coordinator will store a block both in nodes' local in-memory as well as in the remote storage after validation. The format of storing data in a compute node and in remote storage is explained in §3.3.1. Each node communicates with the respective coordinator through our parallel mechanism implemented with MPI discussed in §3.3.4. The communication between compute nodes and the remote storage is managed through a parallel file system (e.g., GPFS).

3.3.3 Block and Transaction Queue

The newly created or received transactions are pushed into a *pending transaction* queue. In BAASH, we can adjust the block size according to our demand. For instance, for all the experiments presented in this chapter, each block consists of 4 transactions on average. Therefore, the block encapsulation process is triggered as soon as the queue size reaches 4. This is because we want to compare our experimental results with the benchmarks [46]. Once the pending transactions are encapsulated they are moved to the *active transaction* queue. When a block is ready to propagate, it is pushed to a *pending block* queue. At a fixed time interval, a block pops out from the queue periodically and is transferred to a coordinator. Once the block is transferred, it is then moved to the *active block* queue.

It could be argued that a queue might not deliver data at a sufficient rate to feed the net-

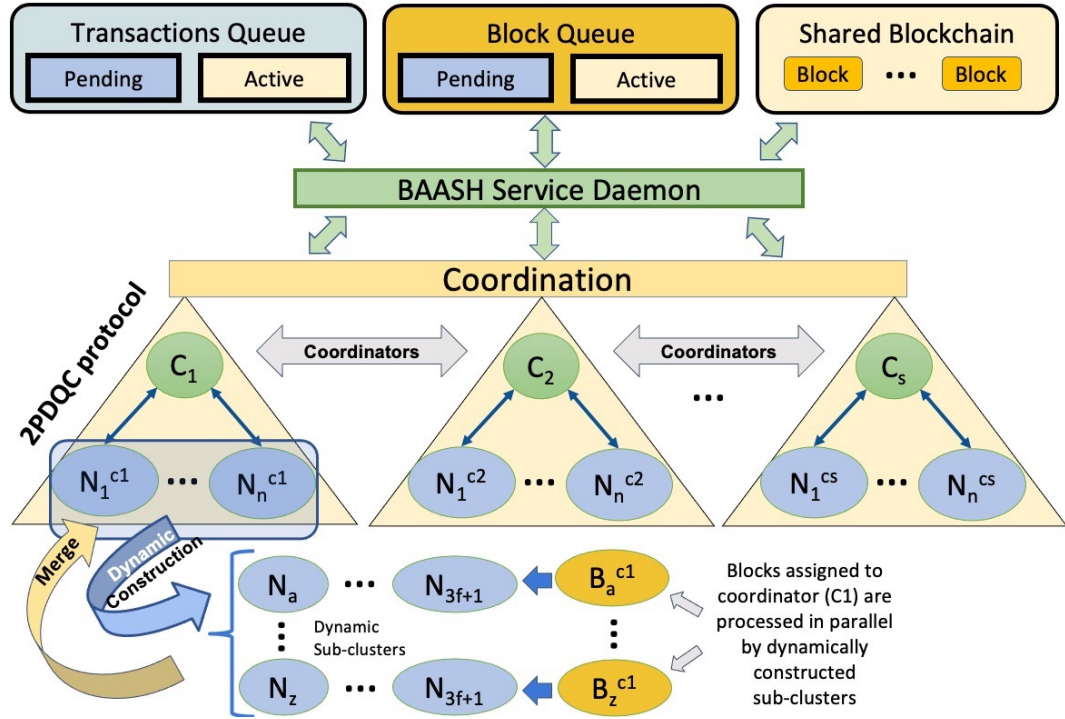


Figure 3.4: Parallel processing module in BAASH manages blocks in parallel through distributed coordinators or consensus managers.

work because a queue is a linear data structure that can hardly be parallelized for the sake of scalability. This is alleviated by the following two approaches in our implementation. First, we adjust the time interval larger than the latency for the queue to pop an element. In doing so, the overhead from the queue itself is masked completely. Second, we implement the queue using a loosely-coupled linked-lists such that the queue can be split and reconstructed arbitrarily.

3.3.4 Parallel Processing Module

As shown in Figure 3.4, the entire network is logically divided into s number of clusters. BAASH creates a set of dynamic coordinators (i.e., MPI communicators) to manage the clusters from the list of nodes. The selection process of coordinators is random, and a coor-

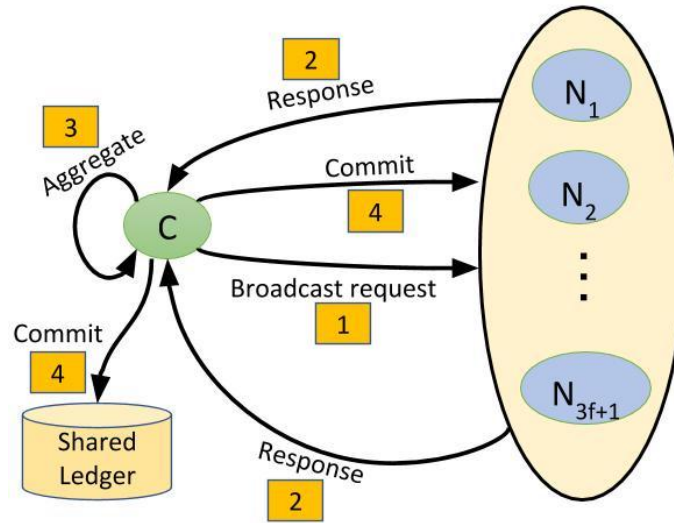


Figure 3.5: A consensus manager (i.e., a coordinator) guarantees the consistency in block management through the two-phase distributed quorum commit protocol.

dinator can not actively participate in the block validation process. Each cluster is managed by a single coordinator. A cluster consists of n number of nodes from where the *parallel processing module* dynamically creates z number of sub-clusters. Figure 3.4 shows how the parallel module in BAASH coordinates with the transaction queue, block queue, and the shared blockchain to distribute the blocks with transactions among a set of dynamically constructed sub-clusters of nodes from the parent cluster assigned to a coordinator (i.e., MPI communicator).

When a block arrives, BAASH starts looking for available $3f + 1$ nodes to construct a sub-cluster to start processing the block. f is the maximum faulty nodes allowed in a sub-cluster. If more than 50% nodes in a sub-cluster fail, BAASH attempts to add new nodes in the initial cluster. The initial cluster then becomes $3f + 1 + c$, where c is the maximum number of nodes adjustment from a parent cluster in case of more than 50% nodes failure in the sub-cluster. To be more specific, c denotes how many times the parallel module attempts to add a new node from the parent cluster if the majority (more than 50%) nodes in a sub-cluster fail. The cluster is flexible enough to deconstruct again to merge with the

parent cluster when no blocks are available to process.

3.3.5 Consensus Manager

Distributed *consensus managers* (i.e., coordinators) assist in managing votes for a block processed in parallel by a set of nodes in the logically divided sub-clusters. It (i.e., consensus manager) guarantees consistency in parallel block processing through a newly designed protocol named *two-phase distributed quorum commit (2PDQC)* protocol.

Figure 3.5 illustrates the entire workflow of the proposed protocol. First, a coordinator broadcasts a block validation request among the nodes in a cluster. Second, the nodes forward their votes to the coordinator after the validation phase. Third, the coordinator aggregates the votes to prepare the final decision about the legitimacy of the block. Finally, the coordinator commits the block in the shared storage (shared blockchain) and synchronizes the nodes with the shared blockchain. It should be noted that all the distributed coordinators (Figure 3.4) keep monitoring the shared blockchain during the entire process to avoid any duplicate processing or double-spending in logically distributed clusters.

3.3.6 Fault Monitor

A real-time distributed monitoring process (daemon service) as shown in Figure 3.6 is designed to keep monitoring the BAASH service to intercept the three aforementioned checkpoints (Section 3.2.1.4). If any of the checkpoints raises an alert during the block validation process, the daemon service stores (i.e., check pointing) the progress and attempts to restart

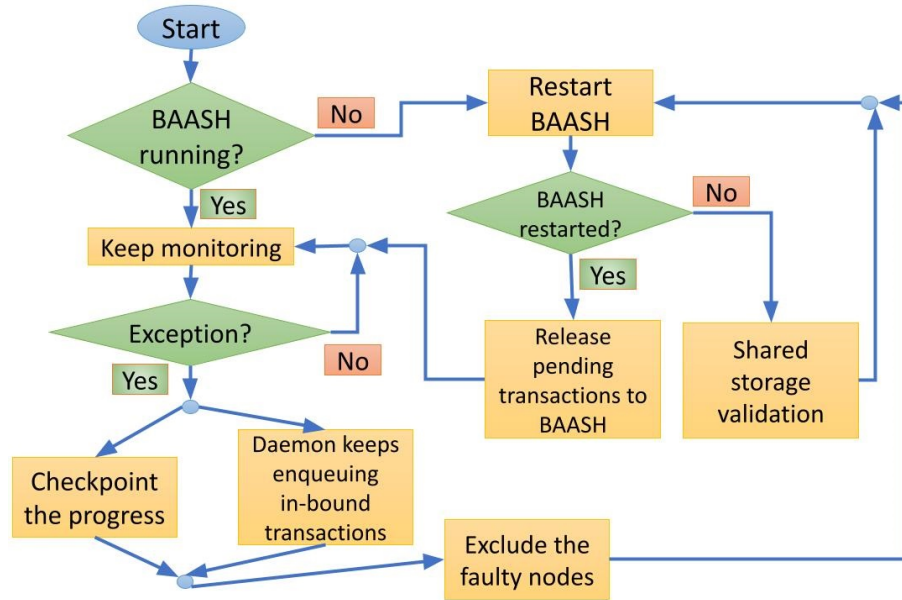


Figure 3.6: Workflow of BAASH's monitor to handle MPI failures. The monitor ensures a consistent BAASH service against arbitrary node failures.

the BAASH service. It (i.e., BAASH) excludes the faulty nodes during the construction of the logically distributed clusters.

The daemon keeps en-queuing the in-bound transactions and leverages the remote storage (i.e., shared blockchain) to continue the validation process while the BAASH service takes time to restart. Once the service restarts, the daemon starts forwarding the pending transactions to BAASH. In the case of remote storage failure, the entire validation process is restarted automatically, which is the worst case and very rare to happen.

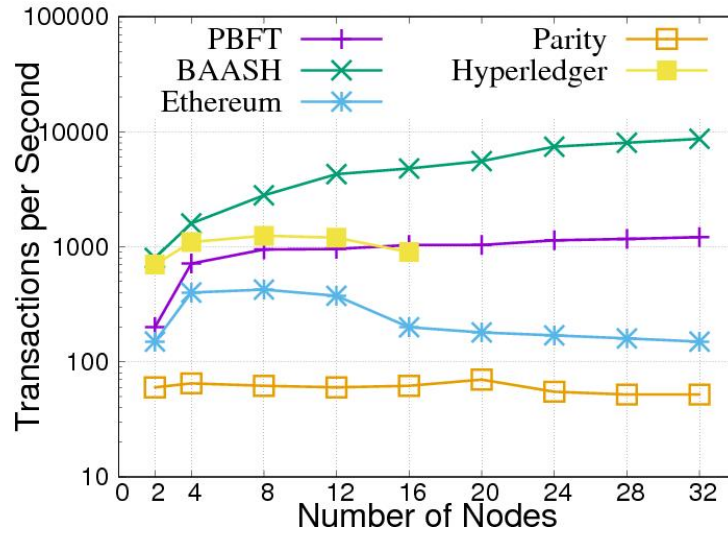
3.4 Evaluation

3.4.1 Experimental Setup

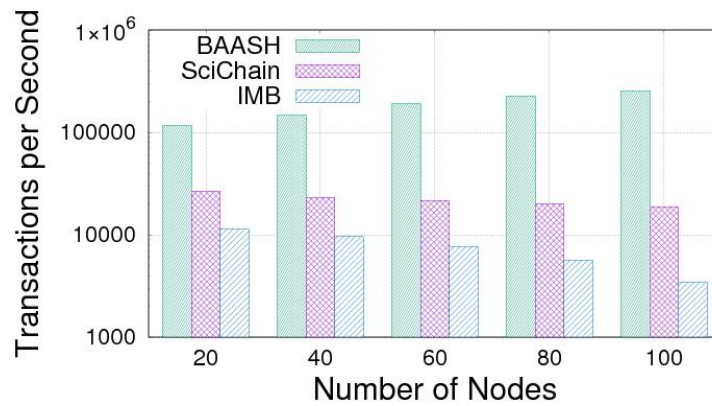
Testbed. All experiments are carried out on a high-performance computing cluster comprised of 58 physical nodes interconnected with FDR InfiniBand. Each node is equipped with an Intel Core-i7 2.6 GHz 32-core CPU along with 296 GB 2400 MHz DDR4 memory; hence each node can be emulated with up to 32 nodes through user-level threads. There is no local disk on compute nodes, which is a typical configuration on HPC systems; a remote 2.1 PB storage system is available and managed by GPFS [126]. Each node is installed with CentOS Linux 7 (Core), Python 3.7.0, NumPy 1.15.4, mpi4py v2.0.0, and mpich2 v1.4.1. We deploy our system prototype on up to 500 cores and report the average results unless otherwise noted.

Workloads. We use YCSB [156], the common benchmark for evaluating blockchains [46]. The YCSB benchmark generates data in a standard format that is flexible to abstract the scientific data from arbitrary applications. There are many existing tools and frameworks (e.g., Myria [102]) for migrating scientific data sets into transactional format; we do not discuss the migration procedure in this work and simply assume the scientific data are already in a format that can be dealt with blockchains and specifically, BAASH. In our system prototype, the default batch size of a block is set to 4, and we deploy more than two million transactions (2,013,590) in all of the experiments. It should be noted that BAASH is flexible enough to encapsulate any number of transactions in a block.

Systems for Comparison. We compare Ethereum [53], Parity [111], and Hyperledger



(a) Performance in Conventional blockchains (batch = 4)



(b) Performance in HPC blockchains (batch = 250)

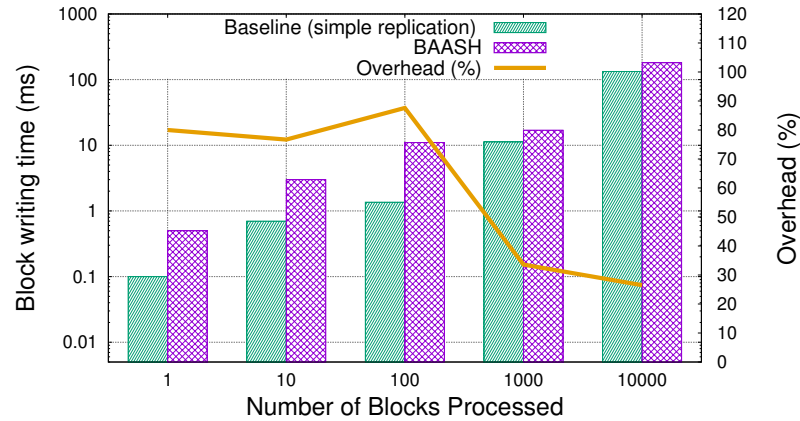
Figure 3.7: Transactions generated per second. BAASH outperforms the conventional blockchains. BAASH exhibits scalability compared to the state-of-the-art HPC blockchain systems.

Fabric [70] with system prototype of BAASH, whose original codename is HPC-blockchain. We also compare the state-of-the-art HPC blockchains SciChain [8], and In-memory blockchain (IMB) [7]. The sixth system is a *Conventional Blockchain* based on the practical-byzantine-fault-tolerance (i.e., PBFT) protocol deployed to the same cluster (i.e., 58 nodes). We apply PBFT in each individual dynamically constructed cluster (i.e., with $3f + 1$ nodes) instead of the entire blockchain network during the validation of a block to ensure the fairness of the comparison.

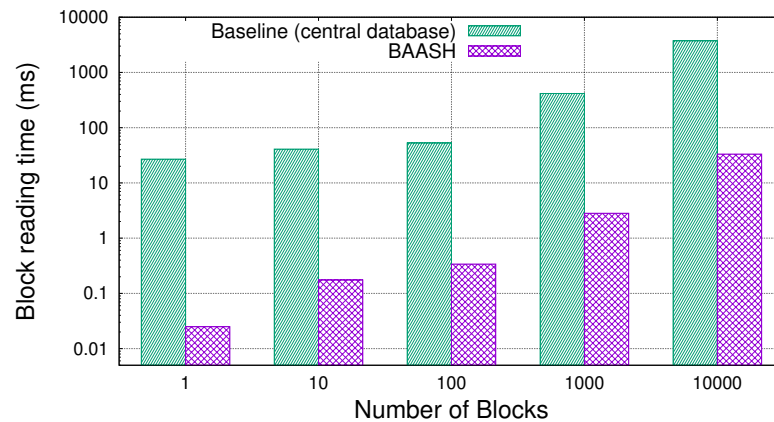
3.4.2 Throughput

In this section, we measure the throughput by BAASH against the other blockchain systems. In the first experiment, we keep the batch size to BAASH's default (i.e., 4) while using 16 nodes as clients where each node issue 1024 transactions per second. Figure 3.7 shows the throughput with varying node scales. In terms of throughput, BAASH outperforms other conventional systems at all scales, as shown in Figure 3.7(a). Specifically, it has up to $6\times$ higher throughput than Hyperledger and PBFT on 16 nodes, as well as $12\times$, and $75\times$ higher throughput than Ethereum, and Parity, respectively, on 16 nodes. Thanks to the proposed consensus protocols that are optimized for HPC systems, BAASH's throughput is not degraded at larger scales beyond 16 nodes, while both Ethereum and Parity show some degradation. Worse yet, Hyperledger cannot scale beyond 16 nodes at all.

We also measure the performance of the BAASH against state-of-the-art HPC blockchains where each node issues 1024 transactions per second. In the experiment, we increase the batch size to 250 to analyze the robustness. As shown in Figure 3.7(b), it is noticeable that throughput in BAASH reasonably increases while scaling; whereas, the throughput tends to decrease in the other blockchain systems. It is a fact that the parallel block processing mechanism injected in BAASH assembles the consensus protocol scalable enough to generate an HPC-compatible blockchain-like resilient system. On the contrary, the IMB [7] follows the proof-of-work, although with a lower nonce, and the SciChain [8] does not offer a parallel processing mechanism that could leverage the distributed nature of the HPC infrastructure.



(a) Performance overhead comparison with replication across nodes.



(b) Performance comparison between BAASH and a central database.

Figure 3.8: Performance and overhead comparison.

3.4.3 Performance and Memory Overhead

We report the performance overhead incurred by BAASH while providing distributed reliable resiliency on a 100-node cluster with increasing workloads. That is, we increase the workload from 1 to 10,000 blocks of transactions with the default batch size 4 to measure the overhead. First, we compare the block writing performance of BAASH against the baseline that only provides resiliency through in-memory simple data replication across the nodes without any reliable verification through a decentralized protocol. We choose the simple replication as the baseline because it provides in-memory resiliency support with

the minimal writing effort. As shown in Figure 3.8(a), although the performance overhead is around 80% while processing smaller workloads, what is more interesting and more important is the fact that the overhead starts to decline as the workload increasing, e.g., the overhead is only around 20% while processing 10,000 blocks, thanks to the parallel processing layer in BAASH which *amortizes* the cost across the participating nodes.

Second, we compare the block reading performance of BAASH against the baseline that provides resiliency through the central database structured in a simple *SQLite* database in the remote storage instead of distributed resiliency through in-memory support across the nodes. We do not note the reading performance against the other baseline (i.e., simple replication without blockchain service) because both BAASH and the other baseline exhibit identical performance while reading blocks from in-memory. Figure 3.8(b) illustrates that the block reading performance of BAASH is significantly faster compared to the centralized database. For example, BAASH can perform almost 112× quicker compared to the baseline while reading 10,000 blocks.

We further assess the memory footprint requirement per node incurred by the BAASH service with varying node scales up to 100 nodes, where each node issues 1,024 transactions per second. Table 3.2 illustrates that with the increment of nodes, the workload increases; yet, each node requires a reasonable amount of memory (e.g., less than 7 MB at 100 nodes) to provide reliable distributed resiliency support across the nodes through BAASH.

Table 3.2
 MAXIMUM MEMORY REQUIREMENT PER NODE IN BAASH AT DIFFERENT SCALES.

Number of nodes	Max memory per node (MB)
20	1.4
40	2.8
60	4.1
80	5.5
100	6.9

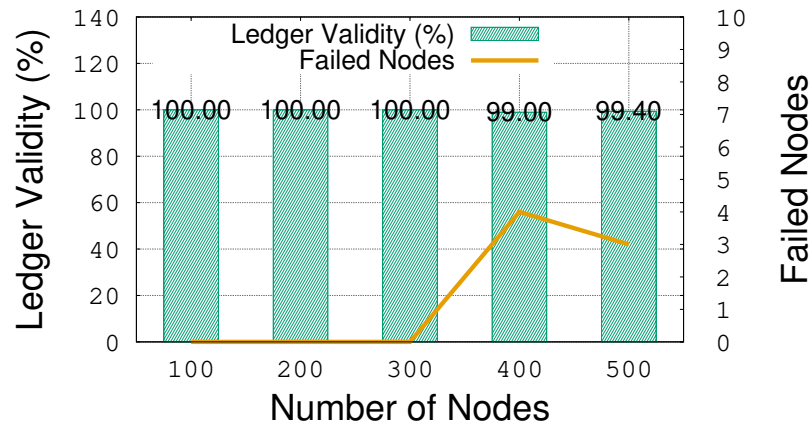
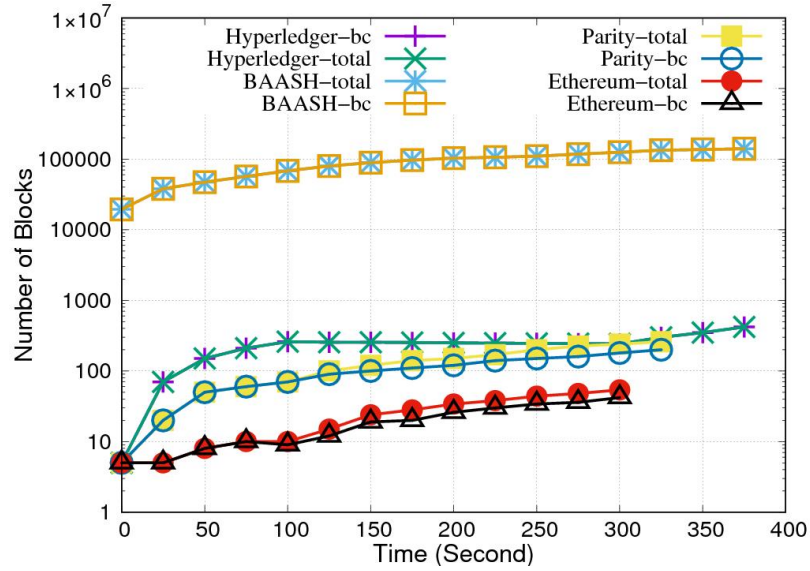


Figure 3.9: Ledger validity in BAASH (in all scales at-least 99% nodes hold consistent ledger).

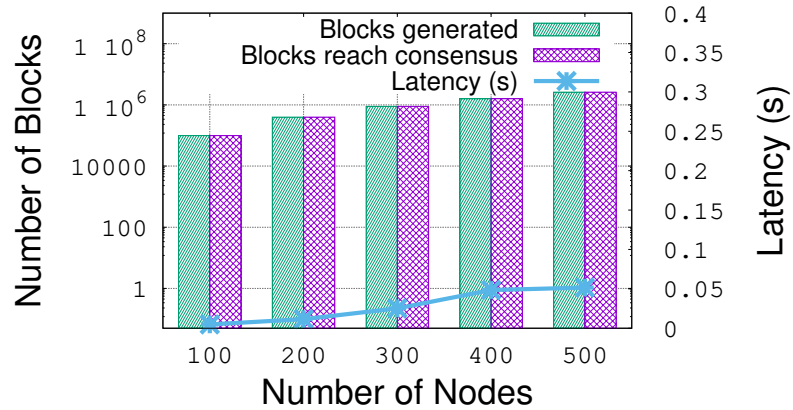
3.4.4 Reliability and Fault Tolerance

We measure the reliability of BAASH on up to 500 nodes and check how many nodes can hold 100% valid ledgers. We also keep track of how much node failure occurs during this experiment. Figure 3.9 shows that in all scales, at least 99% nodes keep the valid ledger. We find 100% nodes hold valid ledger on up to 300 nodes. BAASH guarantees the resiliency as long as more than 50% nodes hold consistent ledger at all scale.

To further measure the reliability of BAASH, we investigate how many blocks reach consensus in all the systems on 16 nodes, where 8 concurrent nodes issue 1024 transactions per second. The reason for choosing 16 nodes is to keep the result comparable to [46]. Figure 3.10(a) reports the number of blocks that reach the consensus and are appended to the



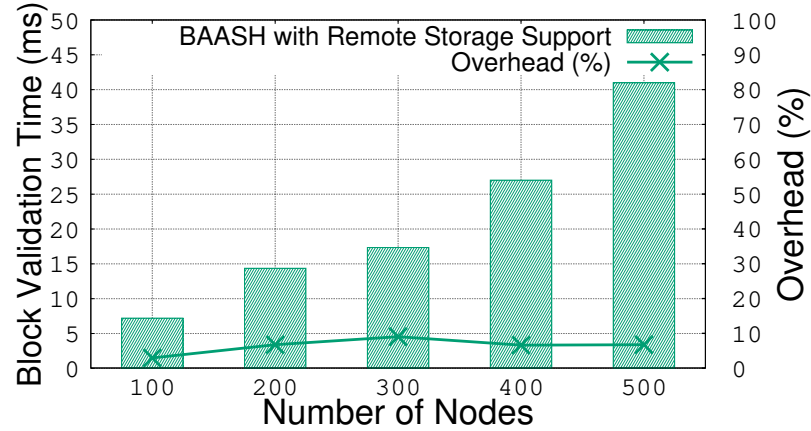
(a) All of the blocks in BAASH safely get persisted after successful validation: the BAASH-total line is completely covered by the BAASH-bc line.



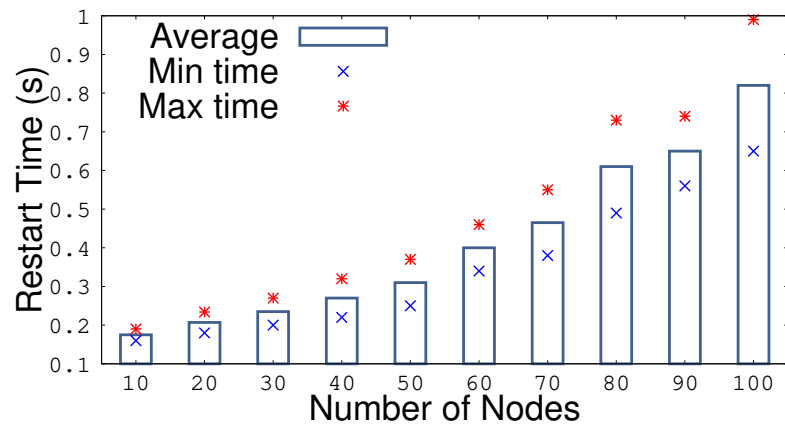
(b) 99%+ blocks reach consensus at a large scale while incurring negligible latency.

Figure 3.10: Reliability in BAASH resilience support.

blockchain. Both in Ethereum and Parity, some blocks are unable to reach consensus due to double spending or selfish mining attacks, and the difference increases as time passes. Though Hyperledger is not vulnerable to those attacks because of not allowing any fork and 100% blocks tend to reach consensus, it is significantly slower than BAASH. In BAASH, almost 100× more blocks are generated compared to Hyperledger, and 100% blocks reach consensus. This is because, BAASH implementation relies on the specially crafted parallel transaction processing mechanism and the newly designed two-phase distributed quorum



(a) Validation through remote storage handled by the fault monitor during BAASH recovery exhibits reasonable performance overhead.



(b) BAASH's recovery controlled by the fault monitor lies within decent limit while the remote storage handling the validation.

Figure 3.11: BAASH recovery overhead by fault monitor.

commit protocol to ensure consistent block processing in a parallel manner. Figure 3.10(b) reports the number of blocks that reach consensus in BAASH at different scales ranging from 100 to 500 nodes. We notice that in all scales, more than 99% generated blocks can reach consensus with negligible latency.

The entire MPI network fails if a node crashes; hence, could hinder the entire block validation process. Therefore, we keep the remote storage to continue the validation. To further study the fault tolerance, we check the block validation time by BAASH with remote storage support at different scales by switching off a random node in the middle of a block val-

validation process and compare the overhead. During the experiment, each node issues 1024 transactions per second. The goal of this experiment is to see how much is the overhead if we leverage the remote storage during the entire MPI network failure while restoring the BAASH service. As shown in Figure 3.11(a), even with failures of the compute node, the block validation process continues successfully with the support of the remote storage. Besides, BAASH with remote ledger incurs negligible overhead even at a large scale (i.e., roughly 5% at 500 nodes cluster).

In case of a severe catastrophe (e.g., compute nodes crash), the distributed fault monitor in BAASH simply restarts the BAASH service, excluding the faulty nodes while continuing the validation process with the support of the shared storage. During the restarting phase, the BAASH daemon needs to restore the nodes with the latest block so that the nodes can join the validation process. To measure the BAASH service restoration time on up to 500 nodes cluster, we switch off a random node in the middle of a block validation process and restore the service (i.e., BAASH), excluding the failed node. We experiment several times at different scales to measure the minimum and maximum restart time. Figure 3.11(b) exhibits that BAASH service requires reasonable time during the restoration phase. However, the good news is that the block validation is not stopped during this restoration phase because BAASH leverages shared storage's blockchain to continue the successful block validation process.

3.5 Related Work

At present, blockchain research focuses on various system perspectives. Algorand [61] proposes a technique to avoid the Sybil attack and targeted attack. Bitcoin-NG [54] in-

creases throughput through a leader selection from each epoch who posts multiple blocks. Monoxide [146] distributes the workload for computation, storage, and memory for state representation among multiple zones to minimize the responsibility of full nodes. Sharding protocols [82, 160] have been proposed to scale out distributed ledger. Hawk [85] is proposed to achieve transactional privacy from public blockchains. A blockchain protocol based on proof-of-stake called Ouroboros [80] is proposed to provide stern security guarantees in blockchains and better efficiency than those blockchains based on proof-of-work (i.e., PoW). Some recent works [63, 81, 105] propose techniques to optimize Byzantine Fault Tolerance (BFT).

Being inspired by Hyperledger [70], Inkchain [71] is designed as another permissioned blockchain solution that enables customization and enhancement in arbitrary scenarios. To improve reliability and achieve better fault tolerance, BigchainDB [21] is built upon the Practical Byzantine Fault Tolerant (PBFT) and comes with blockchain properties (e.g., decentralization, immutability, owner-controlled assets) as well as database properties (e.g., high transaction rate, low latency, indexing and querying of structured data). A 2-layer blockchain architecture (2LBC) is proposed in [11] to achieve stronger data integrity in distributed database systems based on a leader-rotation approach and proof-of-work. Unfortunately, none of the aforementioned work addresses the underlying platform architecture other than shared-nothing clusters assumed by existing blockchain systems that could help us to bridge the gap between the HPC and blockchain technology. The proposed blockchain framework presented by this chapter, therefore, for the first time, showcases a practical parallel blockchain-like framework developed with MPI that allows us to leverage the decentralized mechanism in HPC systems.

We are well aware of recent advances in MPI. Various approaches [24, 30, 34, 64, 92, 97, 124] were proposed to improve or characterize the MPI features in order to design various

solutions. However, all of these works are orthogonal to our work, and none of these aim to develop a lightweight blockchain framework with MPI to facilitate parallel processing in managing distributed ledgers. Therefore, the aforementioned works can be merged into our work for further improvement in MPI-specific packages.

3.6 Summary

This chapter proposes two key techniques to enable a blockchain service in HPC systems. First, a much-needed lightweight set of scalable consensus protocols is designed to account for both the diskless compute nodes and the remote shared storage in HPC. Second, in an HPC environment, a must-have property of blockchain, i.e., the reliability in front of MPI, is guaranteed by multiple layers of subsystems, from background daemon services to callable routines to applications. The HPC-aware consensus protocols and MPI-compatible reliability, under the framework coined as BAASH, collectively enable a blockchain service for HPC systems. A system prototype of the proposed techniques is implemented and evaluated with more than two million transactions on a 500-core HPC cluster. Results show that the system prototype of the proposed techniques significantly outperforms state-of-the-art blockchain systems and exhibits strong reliability with MPI.

Part II

Lightweight Blockchains for Reliable Data Management in Internet of Things

CHAPTER 4

DEAN: A LIGHTWEIGHT AND RESOURCE-EFFICIENT BLOCKCHAIN PROTOCOL FOR RELIABLE EDGE COMPUTING

4.1 Introduction

4.1.1 Motivation

Edge computing [129] offers an efficient means to process collected data (e.g., through sensors and other end devices) at nearby edge nodes as opposed to transferring data back to remote data centers—a common practice in cloud computing. Edge computing saves the network traffic and improves the application performance, particularly for those latency-sensitive applications such as virtual reality [68]. Nevertheless, edge computing poses new technical challenges, including security and privacy concerns with edge nodes and sensors [162]. The root cause of these concerns lies in the fact that existing security techniques used in data centers and cloud computing [25, 101, 169] are hardly applicable to the edge nodes. We highlight two outstanding discrepancies between cloud computing and edge computing in the following.

- **System Infrastructure.** The edge nodes constitute a loosely-coupled distributed system of highly heterogeneous participants with wireless connections. For instance, edge computing nodes range from smartphones to workstations mostly connected

through wireless networking such as WiFi, whilst cloud computing data centers comprise racks of on-premises homogeneous blade-servers interconnected with high-speed network infrastructures.

- **Resource Constraints.** The edge nodes are equipped with many constrained resources in terms of CPU, memory, storage, network, and power. Therefore, many assumptions well-accepted in data centers do not hold in edge computing. As a case in point, a typical edge node has about 2-4 GB memory compared with tens of gigabytes of memory available on the blade servers in data centers. Other resource constraints in edge computing including but not limited to: no or small storage capacity, limited power, to name a few.

Due to the openness and resource-constraint security mechanisms of edge computing systems, various security incidents were repeatedly reported. One notable incident was that a Jeep SUV was remotely hijacked through its UConnect's cellular connection [65]. In addition, mobile devices are vulnerable to malicious applications to a large degree [119], e.g., the users install applications from untrusted third-party sources [91].

One plausible approach to tackle the security challenges in edge computing is to encrypt the edge data with stronger mechanisms such as public key infrastructure (PKI) [31]. Unfortunately, the computing capability of edge nodes is limited; even if they can carry out the encryption (of every single message), the power consumption is prohibitive as they contain limited resource [148]. Besides, the conventional approach fails to solve various security threats in several edge computing platforms such as supply chains, smart grids, and vehicular edge computing. For instance, counterfeiting in the supply chain is a serious concern of the government and the industry [15,72,144]. Apart from that, efficient resource management in smart grid [58,106] and reliable resource sharing in vehicular edge comput-

ing [149] recently become a pressing issue. Consequently, the critical problem is *a single edge node with the highest possible efficient encryption can hardly protect the data*. Therefore, the data management in the edge ecosystem should be supervised with cooperation between multiple edge nodes powered by a decentralized protocol.

In *dependable systems*, there are a vast of studies derived from Paxos [88] and Practical Byzantine Fault Tolerance (PBFT) [27]. Both Paxos and PBFT require a lot of message passing, likely saturating the bandwidth of edge node's wireless network fairly quick. One notable variant is based on full replication: each node stores a replica of the data initiated by the leader, and the integrity is guaranteed as long as no more than 50% of nodes are compromised. This approach requires less communication and instead takes more storage space. Of note, the latter approach reflects the design philosophy of public blockchains (e.g., Bitcoin [22] being, arguably, the most popular blockchain application).

If we reexamine the aforementioned approaches to trustworthy edge computing, the blockchain-based approach has the potential to become a viable solution. Various fields have spent much effort in investigating taking blockchain or its variants for their application-specific needs. For instance, “with the advent of blockchain, decentral[ized] data management can be implemented in a privacy-preserving and efficient way,” recently stated car manufacturer BMW [62]. However, blockchain at the time of writing this chapter is mostly used as a black box without a clear pathway to overcome the aforementioned obstacles: Although blockchain has proven its high security in cryptocurrency, the consensus protocols (e.g., proof-of-work, practical Byzantine fault tolerance, or hybrid) taken by all popular blockchains assume abundant resources (in terms of computation, storage, and network) of the system infrastructure.

Table 4.1
FEATURE COMPARISON AMONG EXISTING BLOCKCHAIN PROTOCOLS AND THE PROPOSED PROTOCOL, DEAN.

Features	Hyperledger [70]	Ethereum [53]	Parity [111]	Omniledger [83]	RapidChain [161]	IOTA [115]	DEAN
IoT specific blockchain	×	×	×	×	×	✓	✓
Lightweight protocol	×	×	×	×	×	×	✓
IoT-specific quorum selection	×	×	×	×	×	×	✓
IoT-specific sharded chain	×	×	×	×	×	×	✓
Cost-effective data replication	×	×	×	✓	✓	✓	✓
IoT-specific data replication	×	×	×	×	×	×	✓

4.1.2 Challenges

Deploying an existing blockchain to an edge network is unquestionably challenging due to several critical constraints: (i) current vanilla blockchains require all participants (full nodes) to store the entire history of the data provenance, (ii) the faulty and unreliable leader selection approach, (iii) compute-intensive, and extensive message exchanging nature of the protocols (iv) in sharded-based approach losing a single or couple of shards due to a few edge nodes failure lead to inconsistencies in the data integrity of the entire blockchain. More specifically, as the edge nodes can arbitrarily come and go offline, in the sharded-based approaches the blockchain integrity might break at any time even with a few nodes failure. Although the proof-of-stake appears to be suitable to apply in edge computing as it offers low complexity of communication and computational work, due to several reliability issues (e.g., nothing-at-stake, or Sybil attack due to the possibility of initial stake procurement from the public pool), it does not meet the requirements to apply directly to edge computing yet. Table 4.1 summarizes the most outstanding limitations of the existing blockchain systems in edge computing and the Internet-of-Things (IoT) ecosystem.

Ideally, *the blockchain crafted for edge computing should be operated among the trustworthy edge nodes elected through an IoT-specific quorum selection policy so that the entire blockchain remains stable and incurs lightweight resource usage on the edge nodes*

in terms of computation, storage, and networking. Although a recent work [69] focuses on edge computing at a small scale, it follows an expensive pre-computation to decide which edge nodes will store the block before a traditional proof-of-stake supported mining process starts. Besides, the approach depends on the traditional proof-of-stake consensus mechanism that is vulnerable to several attacks (e.g., Sybil attack, nothing-at-stake). Moreover, a few more works [89, 147] design protocols that attempt to minimize the resource-intensiveness of either PBFT or PoW; yet, these still require a significant amount of energy both in terms of consensus and storage management. To summarize, *none of the current protocols offer an IoT-specific blockchain management technique that provides a precise balance among the reliability, computation, and storage requirements while affording scalability among the thousands of nodes.*

4.1.3 Proposed Solution

This chapter proposes a set of new blockchain protocols for edge nodes under resource constraints. We name the edge network powered by the new protocol DEAN: Decentralized-Edge Autonomous Network that mainly operates on edge nodes, illustrated in Figure 4.1. The key idea of our decentralized protocols is threefold: *(i)* leveraging the resources (i.e., persistent space and computational power) available in the edge nodes to ameliorate the pressure on the end devices (e.g., edge sensors); *(ii)* designing a lightweight but reliable consensus protocol to support scalability and fast processing of client requests; and *(iii)* balancing the storage pressure among edge nodes that enables fair sharing of the data. In addition to the above requirements, the new protocol must ensure, with provable guarantees, that the entire system's data integrity is not tampered with. To this end, we design new protocols to allow blockchains work with *(i)* among the most trustworthy nodes elected

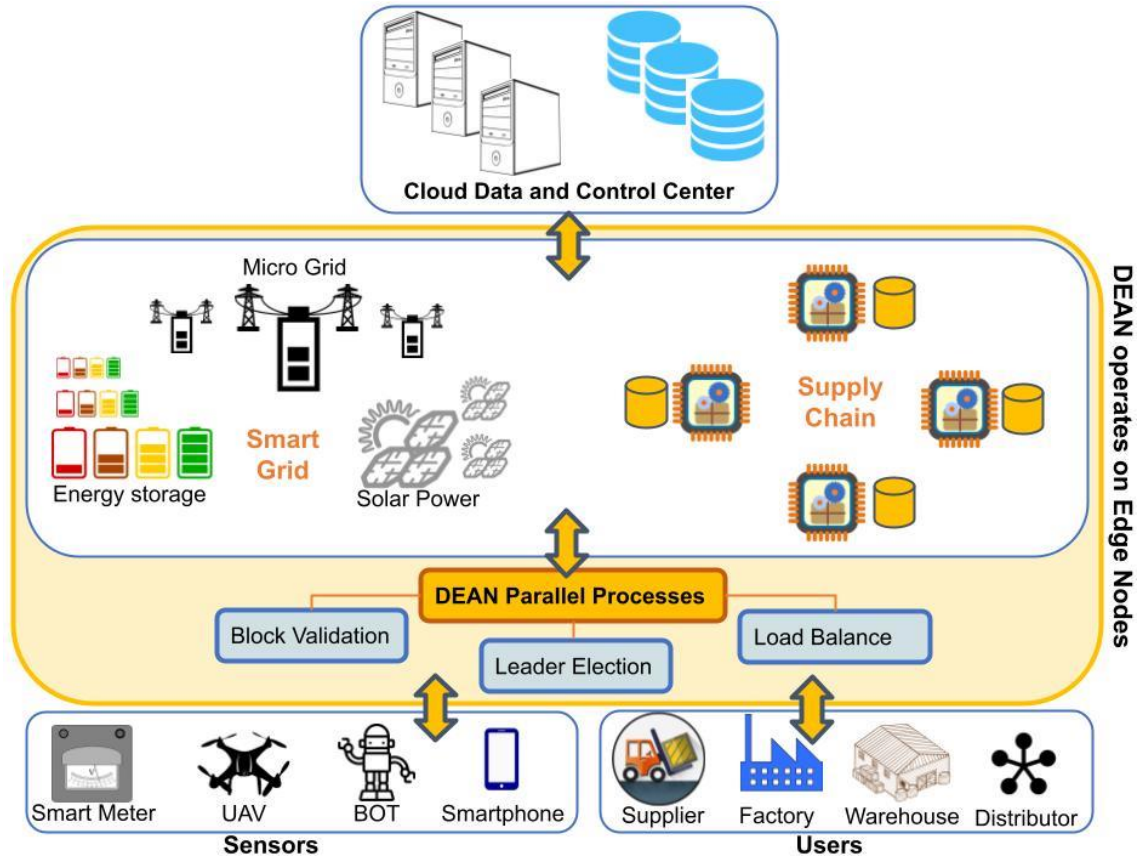


Figure 4.1: Four-tier edge computing architecture with DEAN. Exhibiting sample scenarios where DEAN can assure reliable data management on edge nodes.

through an unique IoT-specific quorum election policy *(ii) limited* computation power and network bandwidth and *(iii) relaxed* storage requirements.

DEAN offers reliable and secure decentralized data and resource management in those edge computing scenarios where edge nodes are available between the end devices and the remote cloud servers. The following list gives a few examples of such scenarios.

- The conventional centralized approach for supply chain management exhibits questionable reliability due to counterfeiting attack [15, 72, 144].
- Vehicular edge computing suffers from malicious behavior [149] (e.g., transaction falsification) among the participant vehicles while sharing the resource.

- Centralized cloud centers fail to assure fair sharing of energy and resources in smart grid [58, 106].

These representative applications are better off equipped with a decentralized system on their edge nodes that demands insignificant resources because these edge nodes are usually commodity machines rather than those high-end servers in the data centers (cf. Fig. 4.1). As such, a lightweight blockchain protocol like DEAN fits ideally to these edge-computing applications.

In addition to exploiting lightweight consensus protocols, we will demonstrate that the proposed design achieves high efficiency: Experimental results show that the system prototype built upon DEAN outperforms mainstream blockchain systems (Ethereum [53], Parity [111], IOTA [115], and Hyperledger Fabric [70]) by orders of magnitude in terms of both throughput and latency, which is attributed to the parallelism of block processing in DEAN.

In summary, this chapter present the following contributions:

- We characterize the edge computing ecosystem with a series of modules that are naturally aligned with blockchain design principles. (§4.3.1).
- We propose a lightweight consensus protocol to adopt blockchains in edge computing that supports IoT-specific energy-efficient quorum building and block processing in parallel, which allows us to scale the system out to thousands of nodes while keeping low the latency of processing client requests. Furthermore, we propose a parallel data dissemination strategy that not only maintains the equitable sharing of the storage among the edge nodes but also enables recovery of and fast accesses to the data even

when in front of edge node failures. (§4.3.2).

- We carry out a thorough analysis of the proposed protocols in terms of both time complexity, spatial overhead, and number of messages. (§4.3.3).
- We implement the proposed protocol and optimization for edge computing and evaluate it with extensive experiments by including comparative study against state-of-the-art blockchain systems. (§4.4).

4.2 Threat Model

We consider the worst case in which an internal adversary has got authenticated to an edge node. With authorized access to the edge devices, the attacker may attempt to alter or modify a transaction record, commit a false transaction (i.e., fraud), perform a denial-of-service attack on other users through an artificial escalation of security level, or even send unauthenticated messages. To be more specific, we are interested to see if the internal adversary can compromise data in 51% edge devices powered by the proposed mechanism.

Another crucial challenge in DEAN is forging of node attributes (e.g., adjacency list, degree of adjacency, location, token, timestamp, etc.) to achieve unfair share. Nodes can closely monitor each other and check the attributes that are periodically updated. The attribute table is shared with the majority (i.e., at least 51%) of the adjacent nodes through the *Block-Validation* process (i.e., Protocol 9). Therefore, any inconsistent operation will help the fellow nodes to identify the malicious activities and eliminate the faulty nodes from the trusted node list.

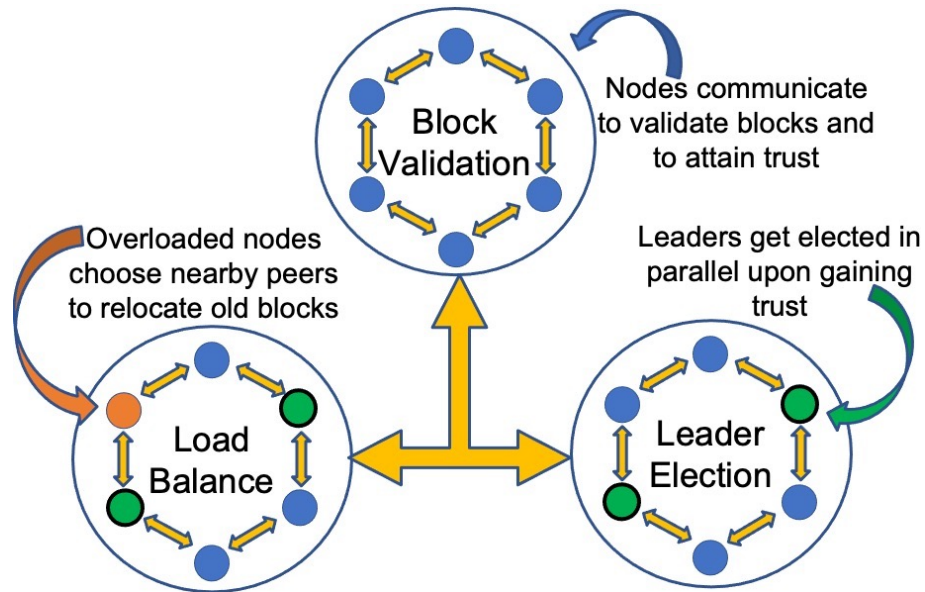


Figure 4.2: Three parallel processes work together in DEAN.

4.3 Decentralized-Edge Autonomous Network

4.3.1 System Components

In this section, we will explain the three parallel core elements of DEAN protocol, as shown in Figure 4.2.

4.3.1.1 Block Validation

Due to the typical nature of the consensus protocols (i.e., compute-intensive, communication-intensive, or Sybil attack due to initial token procurement), the traditional blockchain systems are not directly applicable in the resource-constrained edge computing environment. Thus, we build a protocol that holds both attributes: (i) reliability and (ii) lightweightness.

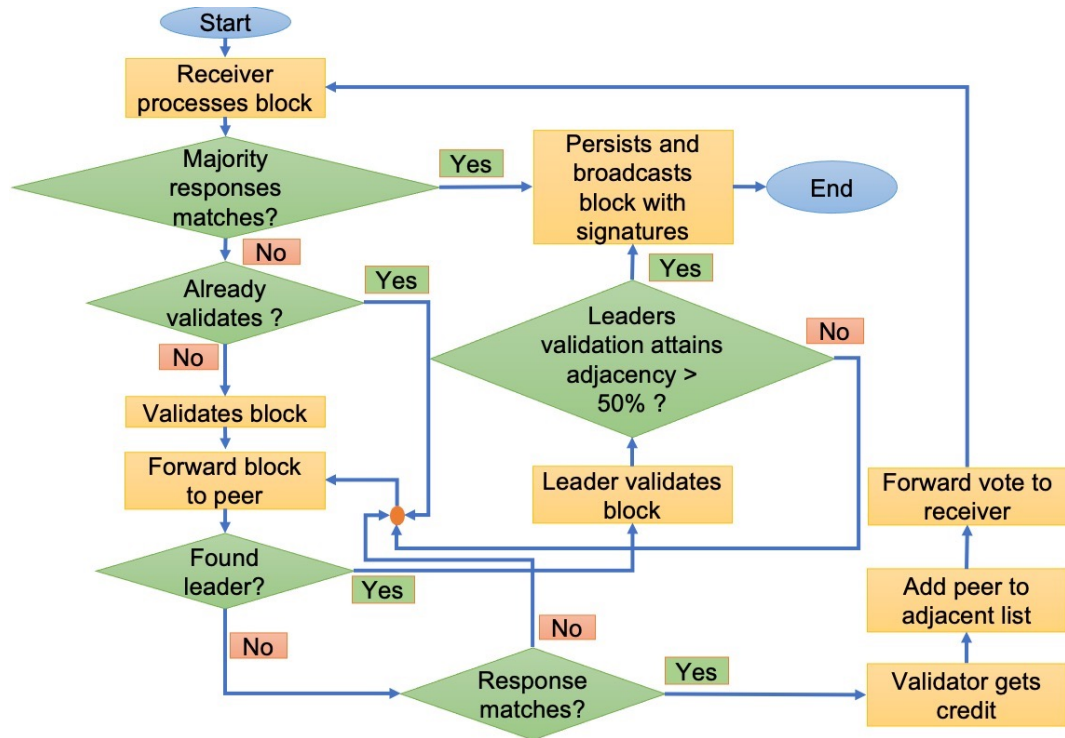


Figure 4.3: Block validation continues while leaders are being elected in DEAN.

The DEAN consensus mechanism works through two parallel steps: (i) validates a block through peer nodes while developing the trust till the leaders get elected, and (ii) builds the network of trustworthy leader nodes (see Section 4.3.1.2).

Figure 4.2 illustrates an overview. First, edge nodes participate in block validation with randomly chosen nodes while establishing the most trustworthy leaders in parallel. The nodes continue the block processing in parallel while developing the leaders through n number of phases (more discussion in Section 4.3.1.2). The initial participants are sorted based on the metadata such as degree of adjacency, token, timestamp, location, etc. For this initial verification of the trustworthiness, nodes need to share their metadata information such as location, timestamp, and degree of adjacency periodically. Therefore, if the initial network size is large, and there is no available leader node nearby, a block receiver node picks a sub-cluster of $2f + 1$ nodes based on the metadata as mentioned earlier to forward

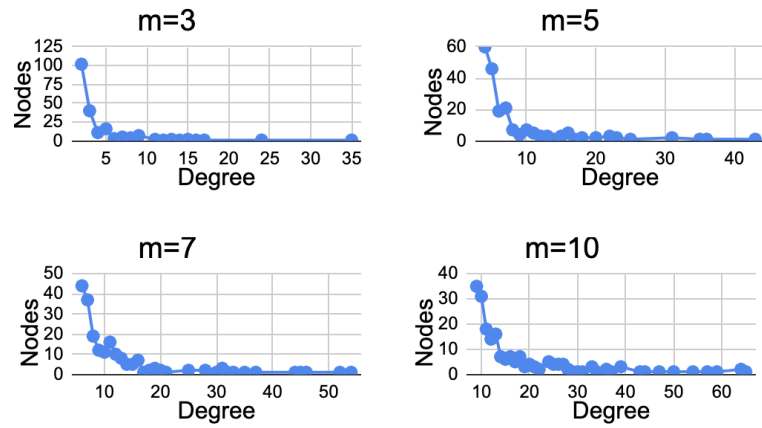
the block, where f is the maximum faulty nodes allowed in a sub-cluster. The receiver node continues to expand the initial cluster until the block attains consensus from more than 50% nodes in the entire network.

As shown in Figure 4.3, the receiver node validates the block and forwards the block to the peers for further validation. When a sufficient number of leaders are elected, the nodes stop exchanging excessive messages and continue to the next phase where the transaction validators list becomes minimal, such as randomly selected leader nodes can approve any new transaction. If the receiver finds enough leaders to validate, the block will not be broadcast among the general peers. If more than 50% peers provide valid votes or sufficient leaders provide a consensus who collectively are adjacent to more than 50% trustworthy nodes, the block is then persisted and broadcast as a valid one. At the end of the validation process, the validators receive the credit, and both the validators and the first node (i.e., receiver) will add themselves as trusted peers.

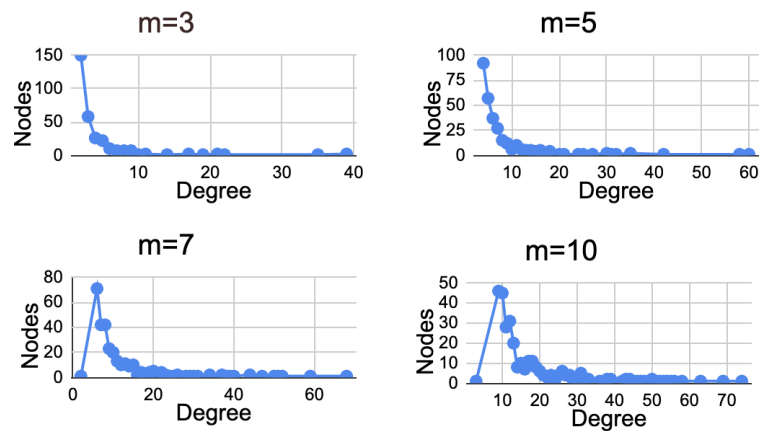
Multiple leaders can work in parallel to continue validating separate sets of blocks. However, DEAN ensures that no leader attempts to mine the same block based on a locking mechanism on a block that ensures atomic operation. Each block has a creator hash ($tHash$) built with the timestamp assigned by a receiver node. Therefore, any node can verify who is the first or original miner of the block from the hash. The consensus mechanism will be discussed in more detail in Protocol 9.

4.3.1.2 Leaders Election

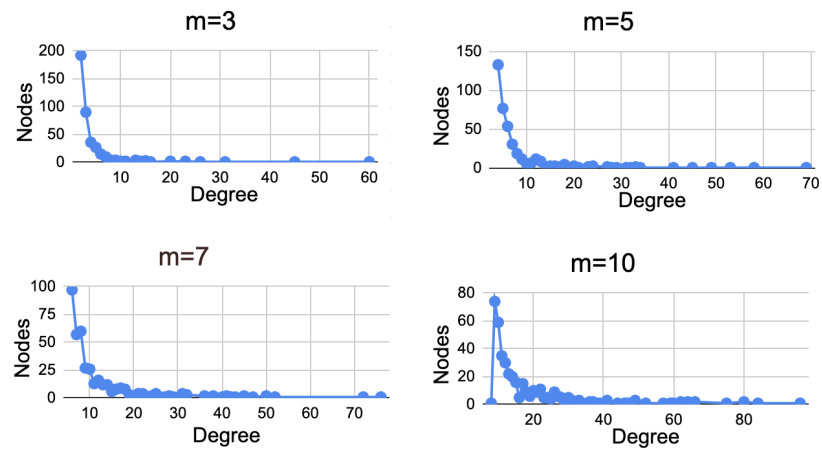
To avoid the problem in the conventional protocols, DEAN is equipped with an IoT-specific quorum selection policy that partly leverages the scale-free network development mech-



(a) 200 nodes



(b) 300 nodes



(c) 400 nodes

Figure 4.4: Finding the optimal m to avoid sub-linear or super-linear nature of the nodes' adjacency. $7 < m \leq 10$ produces more than 50% leaders with reasonable adjacency at various scales.

anism to build a reliable blockchain network in an edge computing ecosystem with the most reliable nodes known as leaders. The proposed technique offers a double-edged solution: *(i)* a reliable and lightweight IoT-specific leader selection protocol; *(ii)* build an unique blockchain replication model to guarantee the cost-effectiveness and stability of the blockchain across the distributed nodes.

Scale-free network comes with several benefits, such as (i) reachability, (ii) reliability, and (iii) stability. In large scale-free networks the small world principle [141] may often hold. This principle says that any node is on average connected to any other node in a small number of steps. The nodes with many connections act as a kind of hub between all the other nodes. Large scale-free networks are not vulnerable to random attacks at nodes. They are vulnerable to targeted attacks at the few highly connected nodes. But the network will often not lose connections if some hubs fail, due to the fact that other hubs remain.

A network is called scale-free when the behavior of the network remain stable along with the scaling of nodes. Therefore, when the network grows, the underlying structure remains almost identical. The core property of a scale-free network [29, 50] is that it follows a power-law degree distribution mechanism. In a vanilla scale-free network development mechanism, new nodes tend to connect to existing ones with linear probability in the degree of the existing nodes. If this probability is sub-linear, all new nodes tend to connect with every other node [19], and hubs are much smaller. If the probability is super-linear all new nodes will attempt to connect to a few nodes with the highest degree [19]; hence, most of the nodes get centralized due to few hubs. Therefore, for simplicity, to avoid both scenarios, in a network of size n , we consider the probability with which a node connects or attempts to forward a block for validation to an existing one follows a power function of the existing nodes' degree k :

$$\Pi(k) = A + k^\alpha \quad (4.1)$$

where A is the initial attractiveness of the node, $m \geq \alpha > 0$, m is an arbitrary integer number, and $\log n < k < n$ to distribute the nodes across the network in uniform manner. The value for A is decided based on the node's metadata such as location, timestamp, degree, etc. Note that, if $\alpha < 1$ then it is sub-linear and if $\alpha > 1$ then it is super-linear. To be more specific, superlinearity leads to centralization, where very few nodes develop the leadership by acquiring the most adjacency. Therefore, to overcome the problems with the vanilla scale-free network development, we need to find an optimum value for m that will assist in building a reliable distributed network of leaders with a reasonable balance of adjacency among the leaders.

To find an optimum m , we implement the equation 4.1 in DEAN and observe the distribution of adjacency when a node tends to forward a block validation request to the trustworthy nodes at various scales for different values of m . Figure 4.4 depicts that at different scales, the optimum value for m lies within a range $7 < m \leq 10$ that produces more than 50% nodes with leadership badge while avoiding excessive adjacency between every other node. Note that this leadership-building process is independent of the block validation process. However, finding a policy of an optimum m could be explored further and is part of another research scope.

The process of building the trusted leaders continues till at least 50% nodes achieve the leadership badge or a reasonable number of adjacencies based on Equation 4.1. A node gains a leadership badge if it meets the leader selection attributes such as *high degree*, *greater token*, *timestamp*, etc, and contains at least $2 \times (2f + 1)$ adjacency, where $(2f + 1)$ is an initial size of a sub-cluster of nodes. Once a few leaders are elected, the sorted leaders based on their attributes (e.g., *degree*, *token*, etc.) will be picked to serve in order based on their availability. *The rationale is a block will attain consensus quickly by exploring a few leaders when more than 50% leader nodes attain majority trusts*. To be more specific,

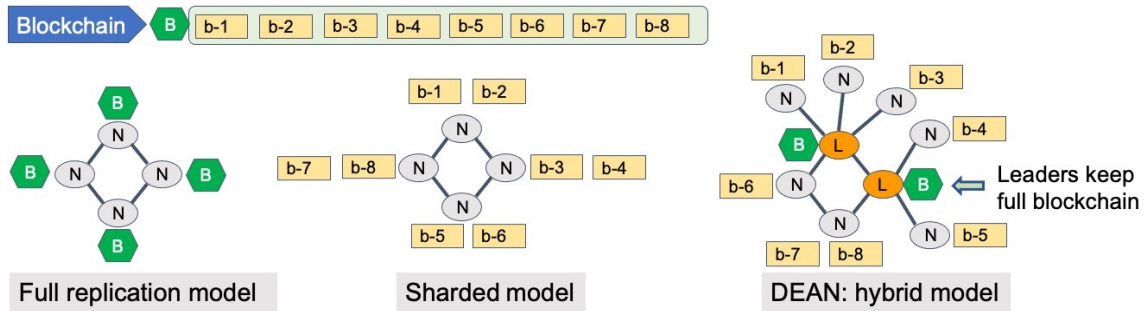


Figure 4.5: Comparing various blockchain management approaches against DEAN.

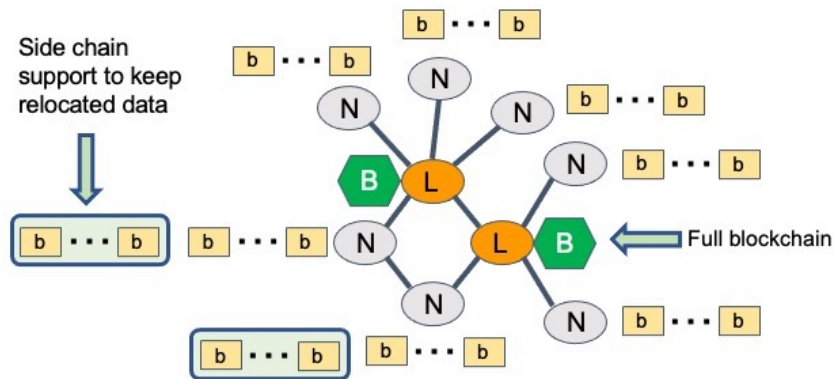


Figure 4.6: Trustworthy nodes assists in load-balancing through side chain in DEAN.

exploring a leader is sufficient enough than exploring a group of nodes connected to that leader; hence, the consensus process remains much simpler and faster.

4.3.1.3 Load Balance

Figure 4.5 depicts, in a vanilla blockchain system, the blockchain is either fully replicated or sharded among all the nodes in a network. In some blockchain systems, the entire blockchain is sharded [40] among the peers in a straight-forward fashion. Due to the several limitations, the traditional mechanisms will be impractical to apply in edge computing: (i) the edge devices have limited storage, (ii) the edge nodes arbitrarily come and go. Therefore, *the blockchain data in edge devices should be organized in such a way that the blockchain remains available despite a bunch of nodes' unavailability and does not*

overwhelm the disk space.

Figure 4.5 illustrates, through a hybrid model, DEAN allows storing the full blockchain only within the most trustworthy nodes known as leaders and manages the shards of a blockchain among the nearby peers of a specific leader. This smart sharding mechanism backed by the scale-free graph development approach offers a three-fold solution: (i) blockchain remains intact, even with the failures of a group of nodes, (ii) consensus process eventually turns faster as soon as a group of leaders gets elected, and finally, (iii) due to storage limitation, if an edge device encounters disk overload, it further disseminates the oldest blocks to the sidechains of the fellow nodes to get more space in the disk backed by a quick data recovery mechanism, as shown in Figure 4.6. We will discuss the data dissemination technique in detail in Protocol 12.

4.3.2 Protocols

Our proposed consensus protocol, namely Decentralized-Edge Autonomous Network (DEAN) consists of four sub-protocols. The first protocol steers both the consensus and the distributed network construction through a scale-free graph development approach. The second protocol assists in finding leaders from the edge devices. The third protocol allows edge nodes to extend the blockchain network by approving new nodes to join. Finally, the fourth protocol assists with smart data distribution and recovery in case of edge devices failure (e.g., storage overload); hence, it helps in the successful continuation of the block mining process.

Block Validation. The Protocol 9 explains the entire consensus mechanism and process

Protocol 9 Block-Validation

Require: N_i the total initial edge nodes in a network N ; b the new block; b_n^i is the total adjacency of a block; b_l is the set of leaders for a block; b_m is the total adjacency of a block through a set of leaders; Edge nodes E where the i -th node is E^i ; E_B^i the blockchain copy on E^i ; E_N^i the adjacent list on E^i ; n the new node; f is total faulty nodes allowed in a sub-network with $2f + 1$ nodes.

Ensure: b is validated by the majority of edge nodes.

```

1: function DEAN-CONSENSUS( $b^i, E, N_i$ )
2:    $N_i \leftarrow 2f + 1$ 
3:   for  $E^i \in N_i$  do
4:     while  $|b_n^i| \leq \frac{N}{2} \parallel |b_l^i| \leq \frac{N}{2}$  do
5:       if  $E^i$  validates  $b$  &  $b \notin E_B^i$  then
6:          $L \leftarrow$  FIND-LEADER ( $b^i, E^i, N_i$ ) ▷ Protocol 10
7:         if  $L$  is  $\emptyset$  then
8:            $L \leftarrow$  nearby peer  $E^j$ 
9:         else
10:           $b_l^i \leftarrow b_l^i \cup L$ 
11:        end if
12:         $v \leftarrow$  BUILD-NETWORK ( $b^i, L, N_i$ )
13:        if  $v$  is valid &  $E^j \notin E_N^i$  then
14:           $E_N^i \leftarrow E_N^i \cup E^j$ 
15:           $b_n^i \leftarrow b_n^i \cup E^j$ 
16:        end if
17:        if  $|b_n^i| \geq \frac{N}{2} \parallel |b_l^i| \geq \frac{N}{2}$  then
18:           $E_B^i \leftarrow E_B^i \cup b$  ▷ Commit block
19:          Forward final result to peers
20:        end if
21:      end if
22:    end while
23:  end for
24: end function
25: function BUILD-NETWORK( $b^i, E^j, N_i$ )
26:  if  $b^i \notin E_B^j$  & Verification Matches then
27:    if  $E^i \notin E_N^j$  then
28:       $E_N^j \leftarrow E_N^j \cup E^i$ 
29:       $E_{tok}^j \leftarrow E_{tok}^j + 1$ 
30:    end if
31:  end if
32:  if  $|b_n^i| \geq \frac{N}{2} \parallel |b_l^i| \geq \frac{N}{2}$  &  $b \notin E_B^j$  then
33:     $E_B^j \leftarrow E_B^j \cup b$  ▷ Commit block
34:  end if
35:  Forward vote  $v$  to sender
36: end function

```

of building the distributed trustworthy network. In essence, when a node receives a batch of new transactions packed into a block propagates the block among the edge nodes. As

Protocol 10 Find-Leader

Require: N_i the total initial edge nodes; b the new block; b_{tHash} the unique identifier of first miner of the block; b_l is the set of leaders for a block; Edge nodes E where the i -th node is E^i ; Leader nodes L where the i -th node is E_l^i ; L is a leader node.

Ensure: Most trustworthy node L gets elected.

```

1: function FIND-LEADER( $b, E^i, N_i$ )
2:   while new block arrives do
3:     if  $b_l^i$  is not  $\emptyset$  then
4:       Move to a disjoint set of nodes
5:     end if
6:      $E^j \leftarrow$  Find nodes with  $k^\alpha$  degree
7:     if  $E^j$  is  $\emptyset$  then
8:        $E^j \leftarrow$  Find nodes with max token
9:     else
10:       $E^j \leftarrow$  Find nearby peer with max adjacency
11:    end if
12:     $L \leftarrow E^j$ 
13:    if  $L \notin b_l^i$  then
14:      Forward  $L$ 
15:    end if
16:  end while
17: end function

```

shown in Line 5, the receiver node attempts to validate the block. After validation, the node starts looking for leader nodes to attain quickly more than 51% votes (Line 6) through their combined adjacency with the general nodes. In the absence of enough leaders, the block traverses the network to find available peers (Line 8). The validators (e.g., leaders or peers) attempt to validate the block (Line 12). If the block is valid, the receiver node and the validators add each other as trustworthy peers and share the transaction fee as shown in Line 13 and Line 26. The block gets persisted if the votes reach the minimum limit (i.e., 51% adjacency) either through the general peers or combined adjacency score from a set of leaders (Line 18 and Line 33).

The consensus mechanism (i.e., block validation) is independent of the *Leader Election* process. To be more specific, a block validation continues at any time even in the absence of leaders; however, the validation process gets faster when sufficient leaders become available through the *build-network* mechanism. In our blockchain system, each node will have

an attribute table that is shared and updated among its peers periodically. The table holds the most recent status of the different attributes of a node, such as degree of adjacency, token, timestamp, location, etc. Therefore, if a node becomes compromised at a certain point, the majority of the fellow nodes already have the updated attributes that help in verifying the messages sent from the compromised node. However, if a node denies sharing its table status with other nodes within a fixed clock period, the node is broadcast as a faulty one, and no further communication is allowed from it.

Find Leader. Protocol 10 assists in finding leaders or the most trustworthy nodes. The other protocols take the advantage of the Protocol 10 when in need of finding leader nodes. As the leading building process is parallel and develops during the block validation process, we do not expect to have leaders at the very first stage. However, once there are a sufficient number of leaders get developed, both the block validation and the blockchain persisting processes become more efficient.

At first, the protocol attempts to find a leader based on equation 4.1, as shown in Line 6. If no leader is located, which is the case at the very initial stage, the protocol then looks for the other criteria, such as max token (Line 8) or max adjacency (Line 10) to find out the most reliable node. The protocol keeps track of whether the same leader is being elected every time (Line 13). It (i.e., Protocol 10) prohibits the leader election from the same set of nodes and prefers to choose a leader each time from a disjoint set of nodes (Line 4). Choosing leaders from a disjoint set of nodes each time keeps the entire validation process undoubtedly reliable.

New Node Approval. Avoiding the unnecessary communication overhead during the *Build-Network*, along with the growth of the network, is a critical issue. In DEAN, we propose a mechanism to address this point. Once the leaders are elected, the new nodes do

Protocol 11 New-Node-Approval

Require: N_i the total initial edge nodes; b the new block; Edge nodes E where the i -th node is E^i ; n the new node; n_s the trustworthiness flag of new node; n_h the hash of new node; n_l is the set of leaders for the new node; L is a leader node.

Ensure: n is verified before it is added to network.

```

1: function JOIN-NEWNODE( $b, E, N_i, n$ )
2:   while  $|n_l| \leq 2$  do
3:      $L \leftarrow$  FIND-LEADER( $b, n, N_i$ )                                 $\triangleright$  Protocol 10
4:     if  $n$  validates  $b$  then
5:       compute  $n_h$ 
6:     end if
7:     if  $L$  verifies  $n_h$  &  $L \notin n_l$  then
8:        $L \leftarrow L \cup n$ 
9:        $n_l \leftarrow n_l \cup L$ 
10:       $L$  earns joining fee from  $n$ 
11:    end if
12:    if  $|n_l| \geq 2$  then
13:       $E$  accepts  $n$  as new node
14:    end if
15:  end while
16: end function

```

not need to communicate extensively with all the nodes to develop trust. Instead, they can join the blockchain network based on the vote from a set of leaders as shown in Protocol 11. That is, new nodes will first try to achieve trust from the leaders.

As shown in Line 3, the new node will try first to find a qualified leader before joining the network. The leader forwards a block to the new node to verify (Line 4). The new node then shares the result with the leader after validating the block. The leader approves the node in the network if the result is valid (Line 7) and earns a joining fee (Line 10). The process continues as long as the new node attains trust at least from two disjoint leaders (Line 12). When other edge devices find votes from the leaders, they automatically add the new node into their trusted adjacent list (Line 13).

Load Balance. We explain the process of data dissemination and recovery in case of disk overload in a node in Protocol 12. As shown in Line 2, an edge node can start disseminating the oldest blocks to nearby fellow nodes if the disk space is occupied by more than 80%.

Protocol 12 Load-Balance

Require: Edge nodes E where the i -th node is E^i ; Adjacent nodes list E_a where the i -th node is E_a^i ; E_B^i the blockchain copy on E^i ; E_{disk}^i the disk space on E^i ; E_{BT}^i the side blockchain of E^i ; E_{bp}^i the block pointer to adjacent edge node; b the oldest block; b_h the hash of a block; b_l the flag to decide block location; t_h^b the hash pointer of relocated adjacent edge node.

Ensure: Overloaded node E_i shares block with reliable nodes.

```

1: function DISSEMINATION( $b, E$ )
2:   if  $E_{disk}^i \geq 80\%$  then
3:     while  $c_l \leq 2$  do
4:        $L \leftarrow \text{FIND-LEADER}(b, E^i, N_i)$  ▷ Protocol 10
5:       if  $L$  is not  $\emptyset$  &  $L_{disk} \leq 80\%$  then
6:          $b_l \leftarrow \text{True}$  ▷ Block for side chain
7:         if  $L \notin E_a$  then ▷ Not in adjacent nodes
8:           STORE-BLOCK( $b, E^i, L$ )
9:            $c_l \leftarrow c_l + 1$ 
10:        end if
11:       end if
12:     end while
13:   end if
14: end function
15: function STORE-BLOCK( $b, E^i, E^j$ )
16:    $t_h^b \leftarrow \text{RECEIVE-BLOCK}(b, E^j)$ 
17:   if  $t_h^b$  is valid then
18:      $E_{bp}^i \leftarrow E_{bp}^i \cup t_h^b$  ▷ Keep pointer
19:     Empty the data from block  $b$ 
20:     Split rewards with  $E^j$ 
21:   end if
22: end function
23: function RECEIVE-BLOCK( $b, E^j$ )
24:   while Block  $b$  arrives do
25:     if  $b_l$  &  $b \notin E_{BT}^j$  &  $b_h$  confirms source then
26:       Compute  $t_h^b$ 
27:        $E_{BT}^j \leftarrow E_{BT}^j \cup b$  ▷ Add to side chain
28:       Return  $t_h^b$ 
29:     end if
30:   end while
31: end function

```

Note that at the early stage (i.e., Protocol 9), after validating a block, each leader stores the hash pointers (i.e., $rList$) of adjacent nodes before disseminating the block to the fellow nodes. Therefore, to disseminate the old blocks, the edge nodes select only those fellow nodes that are not available in the $rList$ (Line 7). The nearby disjoint leaders or the most trustworthy nodes get the highest priority for a relocated block (Line 4).

The sender node sets the block relocation flag before disseminating (Line 6) to distinguish between a regular block and an old block. As shown in Line 25, first, the receiver node will double-check if the block is sent for relocation purposes and is not already stored in the local blockchain. Before storing the block in the disk, the receiver node will first verify the source node's hash (Line 25). If the hash is valid, the node will compute a new hash (i.e., t_b^h) (Line 26) for the relocated block and store it (i.e., block) along with the current hash (i.e., $cHash$).

We use a secured side chain in each node to store the relocated blocks to keep them separate from the regular (i.e., already mined) blocks. The blocks in a side chain can always be verified from the $rList$ stored in the source node. The node will store the relocated block in a side chain (Line 27) before responding to the sender node (Line 28) with the new relocated hash (i.e., t_b^h). The sender node will include the new hash in the block's relocation hash pointer list (i.e., $rList$), as shown in Line 18. Finally, the sender node prepares a hollow block case for the transferred block. That is, it (i.e., sender) keeps only all the hash values of the block for future quick data recovery purposes and erases the data content from the disk to make more space (Line 19). As the receiver node helps in relocating the block, it splits the reward for the block with the sender node, as shown in Line 20. The entire process continues as long as at least two reliable nodes replicate the block.

4.3.3 Analysis

4.3.3.1 Security Guarantee

The security of conventional blockchains relies on the one-way function presumably realized by (cryptographical) hash functions such as SHA-256. DEAN does not introduce new hash functions, and therefore the security guarantee of DEAN-based blockchain implementations is on par with that of mainstream blockchains as long as we can *simulate* DEAN with well-known consensus protocols such as PoW or PBFT. While a full mathematical proof of simulating DEAN with PoW is beyond the scope of this chapter, in the following we sketch the construction of such a simulation by showing that the extra information exposed by DEAN is negligible¹ according to complexity theory [16]. We by n denote the number of nodes in the blockchain network. PoW incurs $O(n)$ messages from the 1-to- n block broadcasting. Let m denote the number of leading nodes, then the voting procedure among m leading nodes incurs $O(m)$ messages. If the voting is propagated outward all the n nodes, then the number of messages in this phase is $O(mn)$. The overall number of messages is $O(m) + O(mn) = O(mn)$. That is, DEAN introduces up to $O(m)$ -fold more messages than PoW, meaning that the adversary might observe up to $O(m)$ more (plaintext, ciphertext) pairs. Let N denote the message length (in bit-string), i.e., the block size in the blockchain. It is easy to see that the additional $O(m)$ messages are *negligible* compared with the entire message space because $m \ll 2^N$. To get a quick idea of m and 2^N , in Bitcoin the block size is 1 MB meaning that $2^N = 8,000,000$, $N = \log 8,000,000 \approx 300$, while m is a much smaller number, e.g., $m \leq 10$, cf. Fig. 4.6.

¹There is a more formal definition of *negligible function* in cryptography using polynomials.

Table 4.2
COMPLEXITIES OF VARIOUS BLOCKCHAIN SHARDING PROTOCOLS.

Protocols	Time	Space
RapidChain	$O(n^2)$	$O(m \cdot B /n)$
Omniledger	$O(n^2)$	$O(m \cdot B /n)$
DEAN	$O(\log n)$	$O(L \cdot B + (n - L) \cdot \log B)$

4.3.3.2 Time Complexity

Without loss of generality, we calculate the complexity of DEAN and compare against the vanilla sharded-based approaches to analyze the resource efficiency by DEAN, as shown in Table 4.2. For validation of a block, DEAN initially starts with $m = 2f + 1$ number of nodes and eventually expands the cluster to attain more than 50% votes among a total network of size n , where f is an arbitrary number and denotes the maximum faulty nodes allowed in a $2f + 1$ cluster. Eventually, in parallel, DEAN dynamically develops some leaders L , connected to a set of random nodes from the edge network k that jointly present more than 50% trustworthy adjacency.

Let tx denote a transaction that belongs to a dynamically constructed cluster m . First, the sensor sends tx to the edge receiver node. The receiver validates and routes the tx to an available cluster m , which incurs a computation overhead of $O(m)$, where $m \approx \log n$. The nodes in the responsible cluster validate the transaction before communicating with the client with the validation result (i.e., encrypted vote) with m messages. Finally, the client aggregates the votes and responds to the DEAN server at a constant rate. Therefore, the communication and computation overhead for a transaction in total is $O(m + m + 1) = O(m) = O(\log n)$.

4.3.3.3 Space complexity

Let $|B|$ denote the size of the blockchain. In DEAN, not every node replicates the entire blockchain. Rather, only the leader nodes store the full blockchain, whereas the other nodes will only hold a fraction of the blockchain, which relaxes the disk requirements to a large extent compared to the proof-of-work (PoW) and practical byzantine fault tolerance (PBFT) protocols. Besides, this hybrid model brings more reliability compared to the vanilla sharding mechanisms because even with a failure of a fraction of nodes, the entire blockchain network remains intact. Therefore, the space required for storing the blockchain, powered by DEAN is only $O(L \cdot |B| + (n - L) \cdot \log|B|)$. Whenever a node receives a new block, the adjacent leaders synchronize the local ledgers. As the nodes do not perform peer-to-peer communication during the synchronization phase, the cost of data synchronization lies within a reasonable limit as it requires only $O(\log n)$ messages.

4.4 Evaluation

4.4.1 Experimental Setup

We have implemented a prototype system of the proposed DEAN blockchain architecture and consensus protocols with Python and deployed to a cluster comprised of 58 nodes interconnected with 802.11n communication protocol. Each node is equipped with an Intel Core-i7 2.6 GHz 32-core CPU along with 296 GB 2400 MHz DDR4 memory; hence, each node can be emulated with up to 32 nodes. We emulate multiple edge nodes with Docker [48] and distribute them equally among the physical nodes. A docker container

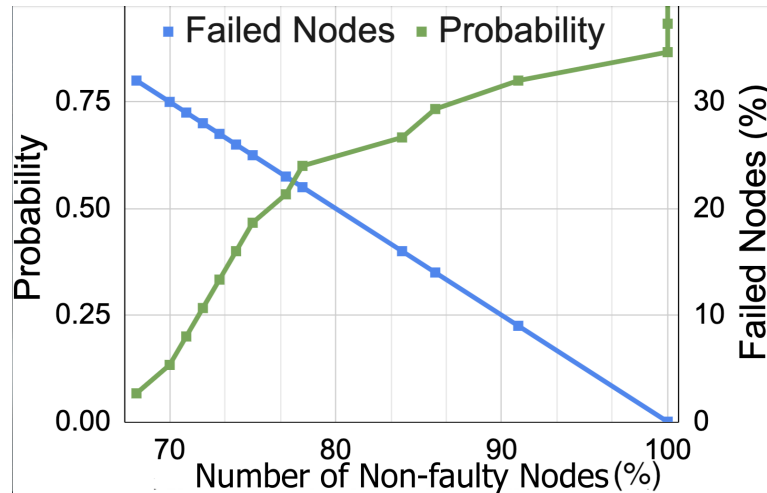


Figure 4.7: **DEAN's Fault Tolerance.** DEAN guarantees that more than 50% of nodes remain intact and hold valid blockchain in practice in front of various number of nodes failure.

represents either as an edge node or a sensor node, and nodes communicate with each other through a standard socket interface. Sensor nodes mainly forward transaction from client to edge nodes for further processing. The ratio between edge and sensor nodes is set to 1:3 by default, i.e., $\frac{|S|}{|M|} = 3$. The main workload under evaluation comprises 2.8 million queries similar to *YCSB* [156]. *YCSB* is widely accepted in measuring the performance of blockchain systems (e.g., BlockBench [46]).

4.4.2 Fault Tolerance

In this section, we experimentally verify the fault tolerance of the proposed protocol. The approach we take is to feed the system with a large number of random transactions extracted from *YCSB* [156] and let the system run for 180 minutes in a 1000-node cluster. We repeated the experiments 15 times. During each run, we randomly chose several edge nodes that either deny to provide validation service (i.e., denial of service attack) or inject false messages during broadcasting the block validation result. To be more specific, we

would like to observe that even with the failure of several nodes, the remaining network still holds the correct blockchain.

We report the results in Figure 4.7 that shows all of the 15 executions lead to more than 68% nodes holding the correct blockchains: While in most cases (i.e., 13 out of 15 executions), more than 70% nodes hold the valid blockchain, only two executions exhibit slightly lower ratios (i.e., hardly below 70%). The bottom line is that, in practice, we need to guarantee at least 51% nodes' data are not tampered with. Besides, the experiment shows more than $\frac{2}{3}$ nodes remain intact, which is the case in a byzantine fault-tolerant system. Note that, by definition, a consensus is reached by having more than 50% of nodes holding valid blockchains. This experiment demonstrates that the real quorum in practice is far more than 51%. Therefore, in DEAN, the lower bound 51% could have been elevated. Since we are interested in only the safety of the protocol at this point, it is sufficient to show that 51% quorum is reached—finding out the minimal quorum (possibly higher than 51%) is beyond the scope of this chapter. This result, however, suggests that we might be able to further trade the additional 17% quorum for even higher performance.

Another important phenomenon is that a larger portion (68%) of nodes hold the correct chain. We did not conduct any theoretical analysis on how many nodes could achieve unanimous agreement (i.e., 100% agreement); however, this experiment demonstrates that a significant portion (68% in this case) of nodes might not experience the so-called “divergence” problem—referring to that subsets of nodes (less than 51%) not holding the valid blockchain. Therefore, there might be some room in the DEAN protocol to further relax the constraints, possibly leading to higher performance.

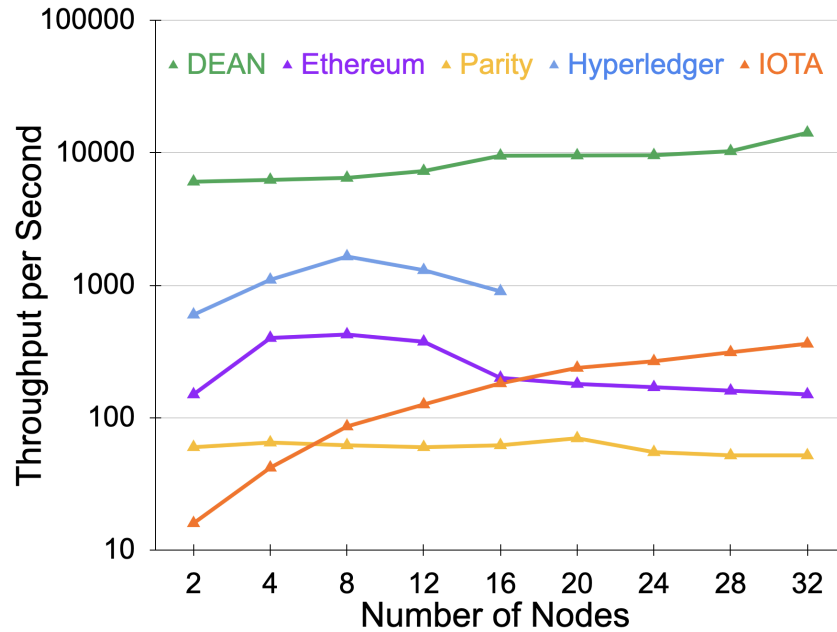


Figure 4.8: **Throughput Comparison between DEAN and state-of-the-art blockchain systems.** IOTA is set with difficulty=1. DEAN outperforms major blockchain systems with orders of magnitude higher throughput.

4.4.3 Throughput

We report the throughput of the DEAN-based blockchain system prototype and compare its performance to other leading blockchain systems: Ethereum [53], Parity [111], Hyperledger [70], and IOTA [115]. We do not compare the RapidChain and Omniledger as the current sharding mechanisms are not appropriate for the IoT environment (see Section 4.1.2). We measure the performance of DEAN on up to 32 nodes (the ratio of edge and sensor nodes is 1:3) over the period of five minutes, where each sensor node issues up to 10,000 queries per second and each block contains more than 12 transactions. The workload is similar to [156] used for other blockchain systems [46].

Figure 4.8 reports that DEAN-based system outperforms all other systems in terms of the throughput. DEAN provides up to 88.6 \times , 16.6 \times , 6.7 \times , and 25 \times more throughput compared to Parity, Ethereum, Hyperledger, and IOTA respectively. DEAN exhibits significantly

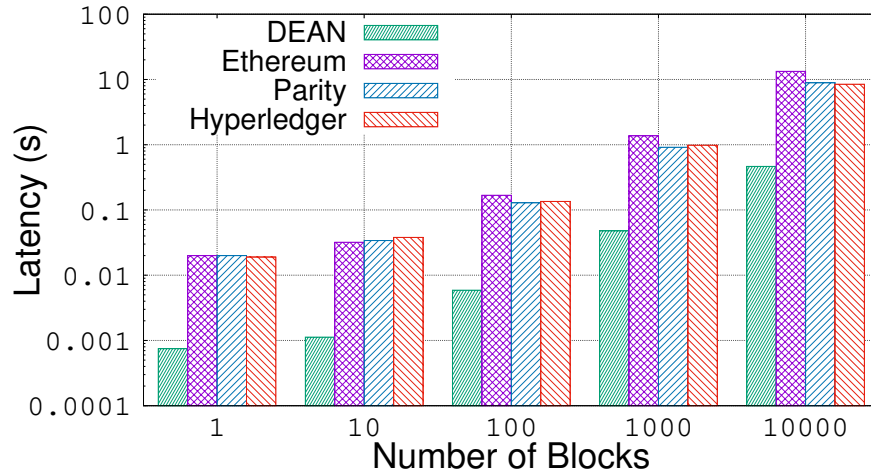


Figure 4.9: **Latency comparison between DEAN and state-of-the-art blockchain systems.** DEAN is orders of magnitude faster than others and incurs only sub-second for adding 10,000 blocks.

higher throughput because it could shorten the validation time without compromising the security guaranteed by the protocol, which exploits the unique property in edge computing. Hyperledger delivers a much higher throughput than Parity because Hyperledger is not based on PoW, but relies on leader selection protocol (practical byzantine fault tolerance, PBFT), whose bottleneck lies on the network rather than the computing time. That is also why Hyperledger shows poor scalability in literature [46] (usually not scalable beyond tens of nodes). DEAN, in contrast, not only achieves higher throughput than Hyperledger but also scales the throughput beyond 16 nodes—the known limit for Hyperledger [46].

Parity delivers the lowest throughput among the five systems under comparison. Parity’s consensus is based on a simplified version of Proof-of-Stake (PoS), which is obviously inappropriate to edge computing (see Section 4.1.2) because it is difficult to estimate the “stake” of the edge and sensor nodes as they may come and go at arbitrary times. Although IOTA tends to exhibit improvement in throughput with the lowest difficulty (i.e., 1), in practice, the throughput is much lower due to higher difficulty, usually greater than 13 [125]. Performance-wise, this experiment suggests that DEAN is a preferable protocol for edge sensors and edge computing applications.

4.4.4 Latency

Similar to the previous section, we compare DEAN's latency with Ethereum, Parity, and Hyperledger, respectively. We do not compare IOTA here as it (i.e., IOTA) processes individual transactions rather than a block. The latency here in this context refers to the latency from the perspective of the entire blockchain system rather than the conventional per-block latency. The latter is less interesting in this context because per-block performance is insufficient to fully describe the performance characteristic of the entire blockchain system, whereas the aggregated block-appending time represents the system latency as a whole.

As shown in Figure 4.9, DEAN incurs the lowest latency when appending various numbers of blocks ranging from one to 10,000. In particular, for appending 10,000 blocks, DEAN takes only 0.5 second while Ethereum takes 13 seconds, leading to 26× speedup; DEAN is also at least 10× faster than both Parity and Hyperledger.

Note that in DEAN all the blocks contain more than 12 transactions, whereas each block generated in the experiments demonstrated at [46] contains only three transactions. Therefore, if we reduce the workload of DEAN, the latency gap would have been larger. In our current implementation of DEAN's system prototype, the number of transactions per block is hard-coded. Future releases will allow users to adjust the transaction density—the maximal number of transactions allowed in a single block (as long as size allows).

Table 4.3
CPU UTILIZATION AND MEMORY FOOTPRINT OF DEAN.

No. of Nodes	CPU Usage per Node (%)	Memory per Node (MB)
4	0.92	12.23
8	1.06	12.61
12	1.15	12.65
16	1.13	12.66
32	1.05	12.83

4.4.5 CPU and Memory Usage

We verify the resource consumption by DEAN at various scales on up to 32 nodes. To test the lightweight nature of DEAN, during this experiment, we use a single physical node equipped with a 32-core CPU and create all the edge nodes along with the sensors with Docker within the physical node with the default ratio (i.e., 1:3). During the experiment the sensor nodes issue up to 1000 queries.

We carry the experiment for five times and report the average CPU usage rate by each node in a specific cluster along with the memory footprint by a node in that cluster in Table 4.3. It is noticeable that at all scales, the memory footprint in each node remains within a reasonable limit (i.e., below 13 MB). It is also notable that the average CPU usage rate per node remains within a negligible range (almost constant) along with the scaling of nodes. For instance, at a large scale (e.g., 32 nodes), CPU usage is only about 1% per node; hence, the DEAN exhibits promising scalability and resource efficiency. Note that the goal of this experiment is to find out the maximum resource efficiency by DEAN with limited resources. The resource requirements in DEAN is fairly smaller compared to the current industrial specifications [118, 145].

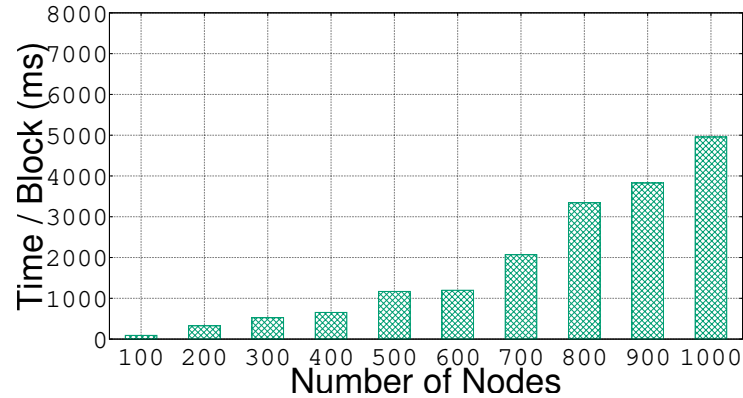


Figure 4.10: **Scalability of DEAN’s block processing time.** DEAN takes less than five seconds for processing a new block on extreme scales of 1,000 nodes.

4.4.6 Scalability

In this section, we report DEAN’s performance at various scales on up to 1,000 nodes. To the best of our knowledge, little literature exists on the mainstream blockchain systems and protocols at such scales. In fact, it is well-accepted that existing PoW, PoS, and PBFT blockchain protocols hardly perform well at a large scale [163]. For this reason, we did not compare DEAN to other blockchains in terms of scalability.

Figure 4.10 reports the time taken by DEAN for processing a single block at different scales ranging from 100 to 1,000 nodes. On 100 nodes, DEAN needs less than a quarter second for processing a block. On 1,000 nodes, the protocol can still deliver a new block within five seconds. Note that in our experimental setup, each block comprises 12 transactions, implying that the overall system can process multiple transactions each second, which is on par with the largest production blockchain system Bitcoin [22]. Indeed, Bitcoin consists of about 10,000 nodes globally [23], much more than the scale we are experimenting here. However, we argue that for edge computing applications where wireless networks are the norm and the performance bottleneck, 1,000 nodes are more than enough for edge computing applications to saturate the hardware resources as reported in [148].

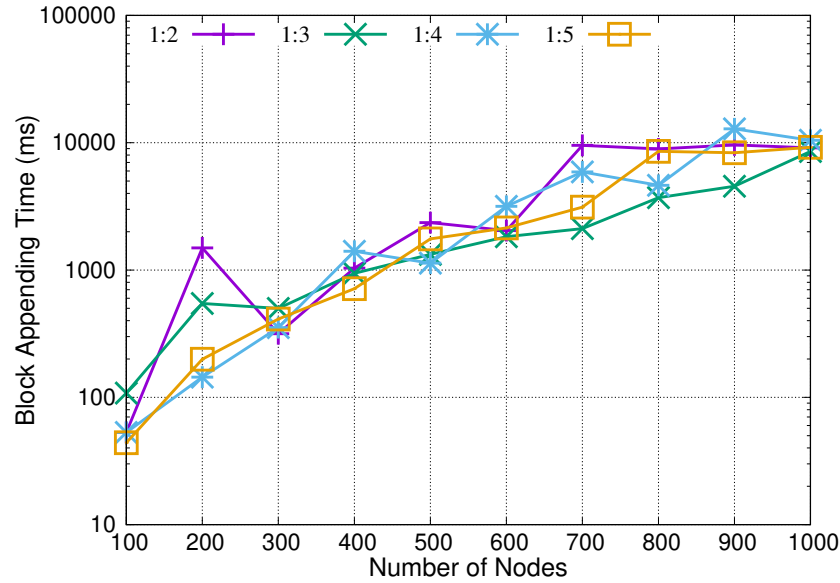


Figure 4.11: DEAN’s block appending time with various distributions of sensor and edge nodes.

While the next-generation high-performance wireless network, e.g., 5G network, is being widely deployed, the network’s limitation will be somewhat relaxed, and then the bottleneck will likely come back to the node scalability. For that reason, we believe further scaling the number of nodes will be an important research question to be addressed in our future work.

We also compare the block processing time with a recent IoT blockchain system [69] that operates on a very small scale (i.e., 50 nodes). We observe that at a 50-node scale, the protocol [69] requires almost 3 seconds, whereas, at a 100-node scale, our proposed DEAN protocol needs less than a quarter second. The reason behind the notable performance degradation in [69] is the protocol does not support parallel block processing and requires a pre-computation of block allocation before each block mining process that consumes a significant amount of processing time. In contrast, the block dissemination mechanism proposed in DEAN is independent of the mining process.

4.4.7 Sensitivity with increasing workload

All experiments thus far discussed assume the ratio between sensor nodes and edge nodes is 1:3, where each edge node is connected by three sensor nodes and all edge nodes are connected with a full mesh network. It is a natural question to ask how, if at all, the ratio would impact the performance of DEAN. The correctness of DEAN is out of the question regarding the ratios as long as the ratio M is larger than one. Therefore, the remainder of this section will evaluate the impact of different sensor:edge node ratios.

We vary the ratios between sensor and edge nodes as 1:2, 1:3, 1:4, and 1:5, and report the block appending time at different scales from 100 to 1,000 nodes. More sensor nodes generate more transaction requests. We change the default block size (i.e., 12 transactions) to 24 transactions to check the robustness. Ideally, the performance should be stable with little impact from the ratios—implying a strong stability of the proposed consensus and system implementation. Finding out the optimal ratio is part of the parameter tuning procedure and is beyond the scope of this chapter. In addition, extreme large ratios (e.g., 1:100 or more) is not the norm of today’s real systems [148] and thus will be investigated in our future work.

In Figure 4.11, we compare the block appending time incurred by different sensor:edge node ratios. Similarly to throughput, appending time does not seem impacted much by varying the ratios, and a smaller ratio exhibits more zigzags than larger ratios due to the same reasons discussed before: with more sensor nodes in the network, the entire system works more like a homogeneous system exhibiting a smoother curve in the performance plot. To summarize, from Figure 4.11, we can draw a conclusion that with the increment of transaction requests from sensors, the DEAN protocol guarantees consistent block pro-

cessing with the growing workload.

4.5 Summary

This chapter presents the DEAN consensus protocol to adopt blockchain in a resource-constrained environment like edge computing. The key idea of DEAN is partly enlightened by a scale-free graph development mechanism to find the most trusted edge nodes through an IoT-specific unique quorum election policy that assures the blockchain service through any uncertain state, and its safety is experimentally verified on a system prototype. We experimentally verified its (i.e., prototype) effectiveness with a comparison with four popular blockchain implementations: Ethereum, Parity, IOTA, and Hyperledger Fabric. Experimental results show that the system prototype exhibits high resilience to arbitrary failures. In addition to the improved data fidelity, the system prototype also delivers orders of magnitudes higher performance (up to $88.6\times$ higher throughput and $26\times$ lower latency) than the state-of-the-art alternatives thanks to DEAN's unique design regarding the load balance on both computation and storage.

CHAPTER 5
**HWCOVER: HARDWARE-BASED CONSORTIUM BLOCKCHAIN
VERIFICATION WITH PRIVACY PRESERVATION**

5.1 Introduction

With the division of labor and the globalization of the integrated circuit (IC) industry in the last few decades, the electronics supply chain has become exceptionally complex [1]. Although the resource allocation is optimized [10, 66], how to ensure the supply chain security remains a challenging problem. One of the major security threats arises from counterfeiting, which has been identified as a critical concern of the government and the industry [15, 44, 72, 144].

Generally, core objectives of supply chain security include (i) ensuring the genuineness of the products, (ii) providing reliable transaction history, and (iii) preserving desired privacy of participants. The lack of a reliable track-and-trace system and information sharing mechanism along the supply chain makes counterfeiting and tampered products serious threats. In practice, most tracking systems of electronics rely on plaintext identities (IDs) printed on their bodies (e.g., serials number and bar code) or written in the embedded memories, which provide little security against counterfeiting attacks as the IDs can be easily cloned. To solve this problem, silicon-physical-unclonable-function (Si-PUF) has been proposed [134] and implemented in various IC products, such as system on chip (SoC) [136], processor [142], field-programmable-gate-array (FPGA) [103], bank chip

cards [17, 79], and even lightweight internet of things (IoTs) [26, 158]. Traditionally, PUF-based security is practiced in a centralized model, where one administrator holds the PUF's pre-recorded challenge-response-pairs (CRPs) or PUF's behavior model in secret and is independently responsible for the verification and authentication.

While the centralized paradigm enables highly efficient verification, its reliability has been criticized. For example, a successful denial-of-service (DoS) attack [37] on a single node will suspend the entire service. If a manufacturer or supplier ceases the operation, the verification it is responsible for can no longer be supported. Furthermore, a standard PUF verification only ensures the device identity, which helps very little with tracing products' transaction history and leaves the supply chain vulnerable to illegal device recycling and reselling. At the same time, given the globalized electronics supply chain scenario, we are facing the challenge of *administrative scalability*: different regions have their administrative agencies, where the trust and integration of management among them are limited. As a result, a centralized management paradigm is not always applicable for supply chains on a global scale. An example of IC supply chain deployment and the related issues are depicted in Figure 5.1.

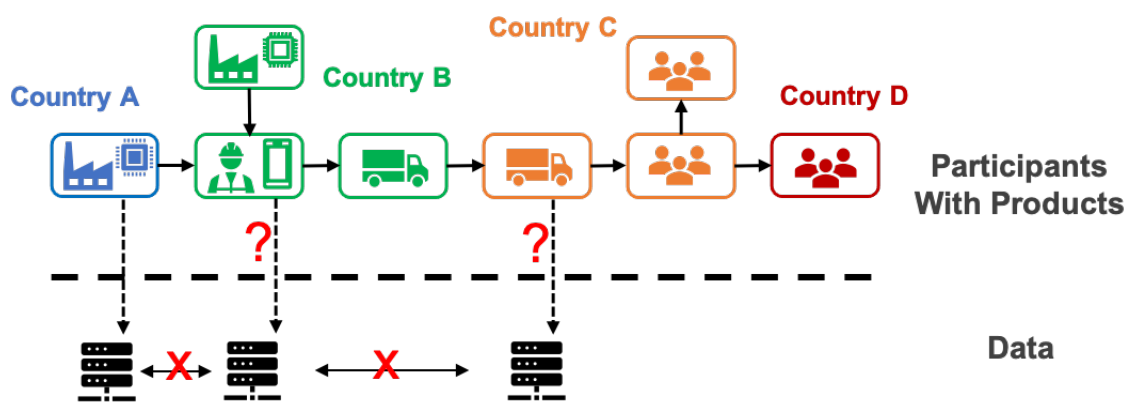


Figure 5.1: Electronics supply chain include multiple participants from different countries. Unreliable verifications (with question marks) and lacks of data sharing (with cross marks) render the global-scale supply chain vulnerable.

From the above observation, many believe that a decentralized management system would overcome many challenges posed by the conventional centralized solution. Specifically, a decentralized system would provide better administrative scalability, as well as more reliable product verification and trustworthy tracing functions. For example, several PUF-based blockchain techniques [12, 74, 153] were recently proposed for IC supply chains. PUFs are employed as the hardware security module (HSM) for device verification in those architectures. The blockchain system, usually in a consortium structure, is responsible for validation service and endorses the transaction records.

Unfortunately, even state-of-the-art systems exhibit multiple limitations. Firstly, those PUF CRPs in existing decentralized systems were shared among all blockchain nodes, which exposes new security vulnerabilities: one compromised node will result in the leakage of all PUF verification data. Secondly, although it is well accepted that preserving participants' privacy is critical to encourage the adoption of a decentralized data management system, the feature of privacy preservation is rarely supported in production blockchain systems for supply chains. There are theoretical efforts for supply chain blockchains, such as [12] based on encryption and zero-knowledge proof. However, it is still unclear how to incorporate practical building blocks such as key-exchange channels or public-key infrastructure (PKI) in an IC supply chain environment, especially for personal customers.

In this chapter, we propose a **HardWare-based Consortium blockchain VERification and tRacking system (HWCOVER)** to achieve the following four goals at the same time: ensuring the genuineness of the products, providing reliable transaction history, preserving desired privacy of participants, and maintaining the system efficiency. Our HWCOVER is implemented with PUF in a consortium blockchain structure. PUF is taken as the hardware security module (HSM) and is a crucial building block for security and privacy strategies. At this point, HWCOVER focuses only on the post-packaging IC products, leaving other

considerations, such as intellectual properties (IP) piracy and tampering, for future works.

The contributions of this work include:

Hashed CRPs. To protect CRP information leakage, we develop a multi-party verification protocol for HWCOVER based on hashed PUF CRPs, and the hashing is differentiated for each verifier/node. As a result, even if the hashed CRPs in a specific verifier's database are compromised by an adversary, those compromised CRPs cannot help the adversary pass the verification performed by another verifier because each verifier uses a different set of CRPs (i.e., different hash functions). The impacts of the differentiated CRPs are multifaceted, and we specifically elaborate on the protocols of product registration (see §5.4.2) and transaction management (see §5.4.3).

PKI-less privacy. We integrate the privacy-preserving technique into the transaction and verification procedures in HWCOVER. HWCOVER endorses the integrity of transaction records but does not have direct access to encrypted sensitive information. The encryption for privacy is based on PUF-generated keys, where a PKI or other subsystems for key management is not necessary (see §5.4.2).

Evaluation. We perform both principle discussions and experimental evaluations of HWCOVER. The principle discussions imply that the proposed protocol in HWCOVER guarantees reliable product verification even a significant number of blockchain nodes are malicious. At the same time, the system cost is kept minimum in time complexity and computational resource intensity, along with the scaling of nodes (see Table 5.1 and Table 5.2). We also deploy a prototype system of the proposed HWCOVER blockchain architecture that is implementable to any consortium network as a middleware. The experimental results exhibit:

- The proposed hashed CRP method (see §5.4.2) requires affordable overhead cost. For instance, only 1.2 seconds at 32 nodes for 1,000 products (see Fig. 5.9, §5.6.2).
- The proposed protocol produces $1\times$ (worst case) to $5\times$ (best case) more throughput and incurs up to $9.1\times$ lower latency compared to currently popular blockchain platforms (see Fig. 5.10, §5.6.3 and Fig. 5.11, §5.6.4).

To the best of our knowledge, this is the first work *investigating the feasibility of adopting a privacy-preserving mechanism along with the co-design of decentralized data management protocol and hardware-based encryption scheme*, which, in a more general sense, represents a canonical example to achieve a good balance between security and performance in the context of supply chains. Our principle discussions and experimental results suggest that the HWCOVER blockchain represents a solid step toward an efficient, reliable, secure, and privacy-preserving supply chain management system.

5.2 Related Works and Challenges

5.2.1 Device Verification

Reliable device verification is a fundamental step to ensuring the security of the IC supply chain. Traditional and emerging techniques that have been applied or demonstrated are discussed below:

Optical ID, such as bar-code and quick response (QR) code, which is the most widely used product ID technique. The advantages of these optical ID techniques are low cost and

convenience. While due to the vulnerability to cloning attacks, they are more suitable for product track-and-trace, combined with other security mechanisms (e.g., digital signatures) and prerequisite trust [87, 154].

Nanoparticles in different formats have been proposed for anti-counterfeit labels in recent years, including plasmonic metals [86], molecular aggregates [104, 114], upconversions [137, 164], carbon dots [78], and so on. Due to the complicated self-assembled microstructure, it is difficult to clone IDs same to the authorized one. However, most of these materials are fluorescent, requiring special optical equipment and analysis for ID verification.

RFID (radio-frequency ID) is another type of commonly used ID technique because of the convenience and cost-efficiency. However, simple RFID is based on antennas to send and receive wireless signals, which can be reverse-engineered and suffer from counterfeiting attacks. A few recent research reports on unclonable RFID [133, 138] borrow the PUF design idea and demonstrate robustness against counterfeiting attacks. The challenge is that an accurate ID reading relies on wide-spectrum RF measurement systems. However, it is not always achievable with a convenient setup.

PUF, integrated with RFID, might be a more acceptable strategy [43, 143, 170], where RFID provides readability while the security is on PUF. Besides RFID, PUF can be used for authentication, signature, key generation, and more. It exploits the uncontrollable randomness during the manufacturing processing, keeping the original ID manufacturing cost reasonable while dramatically increasing the counterfeiting cost. In the 2000s, this concept was employed by the semiconductor research community, and silicon-based PUF (Si-PUF) was proposed. Since then, various types of Si-PUF have been designed and discussed. Nowadays, Si-PUF is widely used as security primitives in many integrated circuits, such

as processors [136], FPGA [103], system-on-chip (SoC) [142], and more.

The security of PUF relies on unpublished challenge-response pairs (i.e., PUF's behavior). Hence a secure database recording PUF's behavior is required during the verification and/or authorization. It is worth noting that for ensuring a reasonable yield, the random processing variations utilized by PUFs are limited in a narrow range. Therefore, the variations can be affected by external fluctuations (e.g., voltage, temperature, and aging), making the error correction circuit necessary for PUF in most cases [159]. The cost of a comparably expensive correction circuit means that reliable PUF is usually adopted by large chips.

5.2.2 Blockchains for Supply Chain

As mentioned in §5.1, a decentralized platform, which requires limited trust, allows convenient public access, and offers immutable transaction history, is desired for supply chain security in practice. These features are well aligned with what the blockchain technique promises. Therefore, a considerable effort has been devoted to employing blockchain for the security of supply chain [2, 33, 57, 84, 123], including IC supply chain [36, 153]. Blockchain is a decentralized data management system [135]. It consists of nodes with limited mutual trust. In a blockchain-based distributed validation process, a system does not trust a single node to finalize a transaction. Instead, all the nodes or a group of nodes participate in the entire validation process of a new activity (e.g., transaction) before recording and syncing the activity. A blockchain-like distributed ledger is based chain structure, linked by a series of hash pointers and considered tamper-proof.

In terms of access permissions, blockchain falls into three main groups: public, private,

and consortium blockchains. In a public blockchain (e.g., bitcoin [108]), anyone can join the network and participate in the competition of appending a new block. In comparison, private blockchains keep the write permissions centralized to a single organization. Consortium blockchains use a mixed approach, where a node requires special permission. The network initiator either pre-verifies the node or prepares a set of rules to verify the node before appending it to the consortium network. As consortium blockchain has the potential to balance security and efficiency flexibly, it is suggested for the supply chain management in recent studies [18,45].

5.2.3 Integrating PUF into Blockchains

The blockchain for the IC supply chain shares similar conceptions with blockchains for crypto-currencies, but there are some differences as well. The most significant difference is that the one for the supply chain deals with transactions based on physical items and needs to ensure the corresponding verification. At the same time, those for crypto-currencies only handle data without physical items directly involved.

Due to the demands on both device authentication and transaction traceability, employing PUF and blockchain techniques at the same time for the IC supply chain is getting more attentions [36,67,153]. A series of important issues for the implementation of such PUF-blockchain-based solutions have been discussed. For example, in [74,76], general protocols of PUF-based enrollment and verification in a blockchain environment were described. It was proposed that only qualified participants (e.g., manufacturers) can register new devices with embedded PUF information; and only the current device owner, who has physical access to the device (and PUF) can start a new transaction. Later in [12,14], authors

pointed out that directly using PUF's CRPs will result in a disclosure of the CRPs, which can be used for counterfeiting attacks by adversaries in the future. Privacy concerns were also briefly mentioned, and private-public-key-based preservation strategies were shortly suggested.

Besides the discussions made for the supply chain, the conception of combing PUF and blockchain has been elaborated for various internet of thing systems, such as lightweight IoTs [155, 157], smart vehicles [77], medical sensor networks [150], and more. This category of studies focuses on the authentication and data integrity of real-time communications in IoT circumstances. Therefore, solutions and protocols developed by such IoT studies take care of different demands compared to desired supply chain security solutions.

5.2.4 Challenges

While PUF-enabled blockchain is the potential candidate to offer reliable supply chain data management, there are two unique challenges in this scenario that have not been elaborated: PUF data management and feasible privacy preservation.

5.2.4.1 Risk of PUF information leakage.

PUF-based verification requires pre-registered PUF information (e.g., CRPs) stored on the verifiers' database. In existing systems [13, 75], identical PUF information is shared by all blockchain nodes. However, if the database on one node is compromised, the leaked data will allow the adversary to obtain the PUF information of all devices and consequently, pass

the verification with counterfeited devices in the future. To recover from such an attack, we will need to update all devices' PUF CRPs somehow. However, this is not always feasible as we cannot expect all device owners, especially personal consumers, to collect and upload new CRPs from their devices. To fulfill the updating, the current owners also need to acquire certain access permissions, which brings even more security vulnerabilities as some owners may be malicious.

5.2.4.2 Privacy concerns.

For cryptocurrency applications [53, 70], the privacy of blockchain users is protected by anonymous accounts that are derived from the user's public key. While a blockchain data management system for supply chains shall keep participants' sensitive information (e.g., price, buyer ID) from non-related ones, the blockchain nonetheless holds verifiable information transparent to stake-related ones, such as buyer or government supervisors, when necessary. Although asymmetric-key encryption methods with a trusted PKI have been suggested [12, 74] for privacy in general, a detailed study of its feasibility, such as PKI in supply chain environment, key-free strategies, and the possibility of anonymous participants and devices, is still in a lack.

5.3 Threat Model

The electronics supply chain discussed here starts from the manufacturers to assemblers, transporters, sellers, and end customers. There are several types of attacks that are aimed explicitly at this environment, including:

Counterfeiting attack. We assume the device is commercially available, allowing the adversary to learn the design, and the adversary has access to the same manufacturing technology as the original manufacturer. From the adversary's viewpoint, a successful attack requires counterfeiting a "considerable" amount of the products at a reasonable cost.

Verification information leakage. The information used for product verification that should not be publicly shared in a wide range can be the attacking target, such as serial numbers (SNs) and PUF CRPs. The leakage of information about a specific product is not considered high risk. We assume the adversary has a chance (although it could be small) to compromise a limited number of servers that store the verification information, extract and use the information in later malicious activities.

Transaction record tampering. The attacker may want to tamper with transaction history to manipulate the produce status in the supply chain system, such that some illegal transactions can be approved improperly. The tampering include (i) forging transaction with non-authorized or incorrect information and (ii) changing existing transaction records.

Privacy compromising. Although in most cases, product transaction information is not considered secret, it is common that sellers and/or buyers may want to keep the transaction details from other ones, especially from competitors in the market. We assume adversaries will try to utilize shared information to extract sensitive transaction details (in batches, particularly), including participants' identities, product details, and packed verification information.

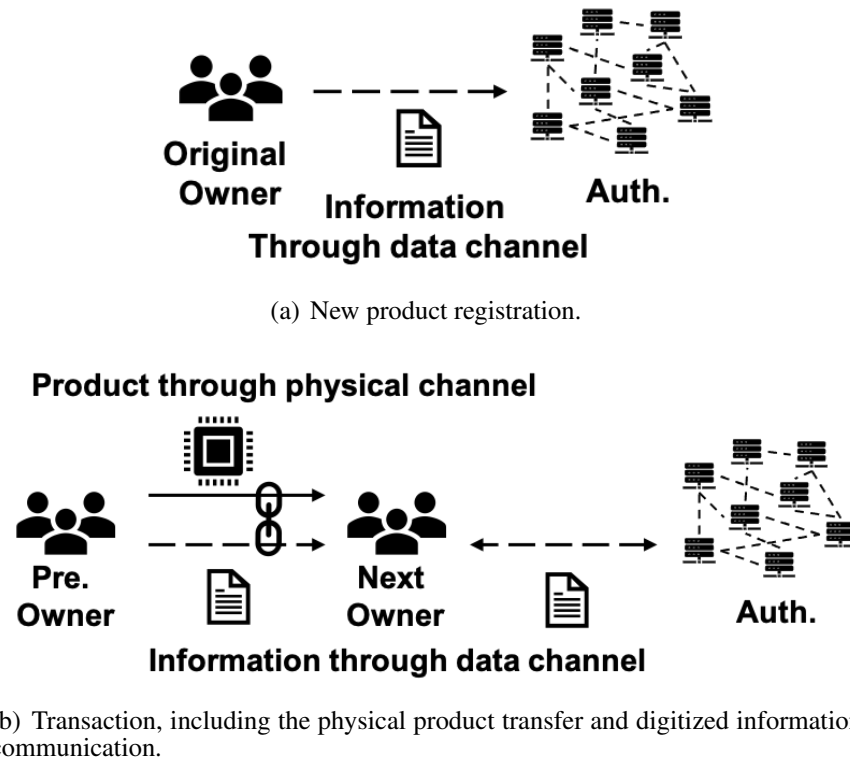


Figure 5.2: Typical supply chain activities.

5.4 HWCOVER Design

5.4.1 Overview

The supply chain discussed here starts from product registrations when the original owners submit information to third-party authorities for future validation (Fig. 5.2(a)). After that, there can be a series of transactions wherein a device is transferred from the previous owner to the next owner through physical channels. At the same time, the related information is managed in data format to assist the transaction (Fig. 5.2(b)). For the supply chain, hash-pointed-based data chain and consensus-based decision strategies that are from traditional blockchain techniques are commonly used to protect the system from record tampering and malicious voting.

However, as the vanilla blockchains are not designed for the supply chain, they cannot directly fulfill all the requirements for the supply chain. We propose HWCOVER as a newly crafted blockchain technique equipped with a privacy-preservation technique, specifically for the supply chain ecosystem. It requires electronic devices to carry PUF modules. During the registration phase, HWCOVER performs differential hashing on the PUF CRPs (more details in §5.4.2), which helps to maintain the system authentication service when data breaching occurs on one or multiple server(s). Once the registration phase is complete, the product is ready to supply in the market. When a new buyer wants to verify before purchasing the product, the HWCOVER then comes forward to validate the product authenticity (see details in §5.4.3 and §5.4.4). HWCOVER keeps track of the entire product transaction cycle (from product registration to buying and selling) to ensure reliable and secure traceability.

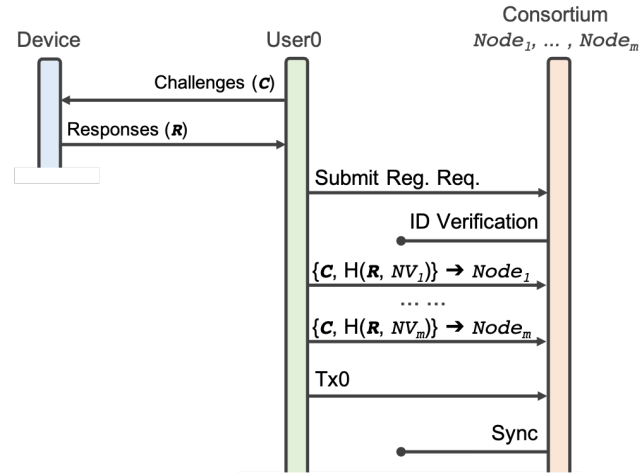
For convenience, we call the product with PUF as *Device*, the participants, who physically hold the product in certain periods, as *Users*, such as original manufacturers, assemblers, logistics, and end-customers. Particularly, the original device owner that introduces the device into the supply chain is called *User0*. In each transaction, the previous owner (e.g., seller) that sends the device out is called *UserA* and the next owner (e.g., buyer) who receives the device is called *UserB*. The consortium network consists of different servers, which are called consortium *Nodes*.

Before further discussions, we introduce several assumptions, which HWCOVER is built on:

- Assumption 1: *Nodes*. Only parties, such as government agencies, authoritative associations, and reputed companies, who are approved by the consortium blockchain network, can serve as the consortium *Nodes*. The approval can be processed based

on comprehensive considerations in various aspects, including political ones, economic ones, etc. Any *Node* should have workstation- or server-level computational resources and large-capacity communication bandwidth.

- Assumption 2: *User0*. Only qualified *Users* can serve as the original *Device* owner (*User0*), who is usually a manufacturer or assembler, and has access to register corresponding new *Devices*. Since *User0* is responsible for the registered *Device* until the next transaction, it has motivations to maintain the record and hence is trusted. The computational resources that *User0* can access is similar to what *Nodes* have.
- Assumption 3: *UserA/B*. The transaction is open to the public, which means anyone with resources satisfying minimum requirements (e.g., internet access and chip reading) can participate in the transaction. For example, *UserA/B* should have access to cellphone-level resources, such as energy-efficient computing power, limited storage space, and reasonable communication bandwidth. In a transaction, the previous owner (*UserA*) is to be released from the responsibility and thus untrusted, while the next owner (*UserB*) is to be responsible for the *Device* and thus is trusted.
- Assumption 4: PUF and cryptographic algorithms. We assume the PUF and cryptographic algorithms are cryptographically secure, which indicates they are resistant to attacks with reasonable cost. For example, a security design requiring \$10 implementation is expected to defend \$100 or \$1,000 attacking but not a millions-dollar-attacking supported by a big company or even a government.
- Assumption 5: public network. We assume communications among different participants go through the public network in the real world, with common network security protocols applied, such as HTTPS, SSL, ICMP, and others. The security discussion on this part is not included in this work.



(a) User0 performs new device registration.

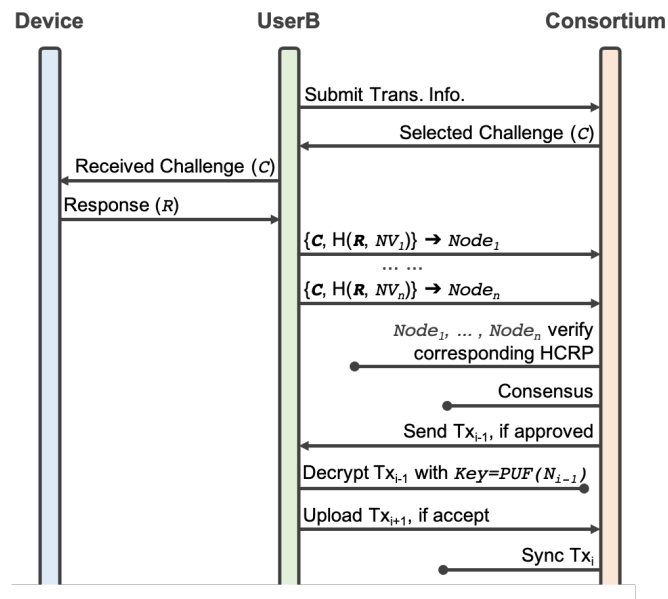
(b) Device's i -th transaction.

Figure 5.3: Registration and transaction processes.

5.4.2 Registration

During the Registration, *User0* submits *Device* ID, PUF CRPs, and related information to all consortium nodes. *User0* ID (i.e., qualification of the original owner) is verified by the consortium. The submission is accepted and synced with a successful verification.

Unlike unilateral PUF-based verification mechanisms, in a decentralized data management environment like blockchain, all consortium nodes need PUF CRPs for Device verification. It greatly increases the risk of leaks because once a leakage occurs, all *Nodes* have to replace the existing PUF CRPs to prevent potential counterfeiting with leaked PUF CRPs. However, as explained in §5.2.4, this is not always practical.

Here, we propose using **PUF challenges and hashed responses (HCRPs) with a unique vector (NV) for each Node** to replace the original PUF CRPs. *User0* collects sufficient CRPs, performs hashing for *Node_i* with its own vector NV_i , and uploads the HCRPs to the corresponding *Node_i*, individually, in point-to-point (P2P) manners. In this way, even if the attacker compromises one consortium node and obtains the HCRPs, the information cannot be used to pass the verification on other nodes. We name the above process *differentiated hashing*, as illustrated in Figure 5.3(a) and formalized in Protocol 13.

With differentiated hashing, we can simply remove an abnormal *Node*—whose HCRP database is compromised—from the list of qualified verifiers; while authentication services, provided by other *Nodes* are not impacted. The system resilience is thus significantly improved, although it comes with a cost. The major part of the extra cost in Registration is on *User0*, who needs to upload the multiple HCRP profiles. The number of the HCRP profiles is the same as the number of *Nodes* to be potentially able to participate in the authentication. For example, given the information data of a *Device* is 5 KB, the plain PUF CRP data is 320 KB(yte) (128-bit challenge, 128-bit response, 10,000 pairs), and the number of *Nodes* is 100. The total data amount to be uploaded is about 32MB. It should be pointed out that the extra cost on *Node* side is insignificant, as each *Node* just receives and maintains its own HCRPs.

For privacy preservation, a part of the submitted information can be encrypted by a secret

Protocol 13 Product Registration

Require: Consortium nodes N where the i -th node is N^i ; External supplier S where the i -th node is S^i ; Secret key pairs k where $k_{n_i}^{s_i}$ -th is encrypted challenge-response pair for N^i generated from S^i ; p_{ch}^i is the challenge for a product verification request; p_r^i is the response for a specific challenge from a supplier.

Ensure: All N are registered with the secret key pairs with the external suppliers S .

```

1: function REGISTRATION( $S, N, p_{ch}^i$ )
2:   while  $S_i$  sends product registration request do
3:     for  $N^i \in N$  do
4:        $p_{ch}^i \leftarrow CHALLENGE(p_{rq}^i, N_i)$ 
5:        $p_r^i \leftarrow RESPONSE(p_{ch}^i, S_i)$ 
6:        $k_{n_i}^{s_i} \leftarrow KEYGENERATION(p_{ch}^i, p_r^i)$ 
7:       Forward key  $k_{n_i}^{s_i}$  to  $N_i$ 
8:     end for
9:   end while
10: end function
11: function CHALLENGE( $p_{rq}^i, N_i$ )
12:   while Receives a challenge at  $N_i$  do
13:      $p_{ch}^i \leftarrow$ Generate challenge
14:     Return  $p_{ch}^i$ 
15:   end while
16: end function
17: function RESPONSE( $p_{ch}^i, S_i$ )
18:   while Receives a challenge at  $S_i$  do
19:      $p_r^i \leftarrow$ Generate response
20:     Return  $p_r^i$ 
21:   end while
22: end function
23: function KEYGENERATION( $p_{ch}^i, p_r^i$ )
24:    $r_k \leftarrow$ Encrypt  $p_r^i$ 
25:    $key \leftarrow$ Pair( $p_{ch}^i, r_k$ )
26:   return key
27: end function

```

key so that *UserA* (e.g., sellers) does not have to share all details about the device and the sales with the consortium in future transactions. At the same time, the details shall be available for *UserB* (e.g., buyer) for product checking on receipt. To avoid sophisticated secret key management, HWCover will **use PUF as a key generator**. We use plain text, such as a public nonce (N), as the PUF challenge and the corresponding response as the key for sensitive information encryption and decryption. Eventually, the submitted data during the registration includes *User0*'s ID, anonymous *Device* ID, HCRPs, and encrypted

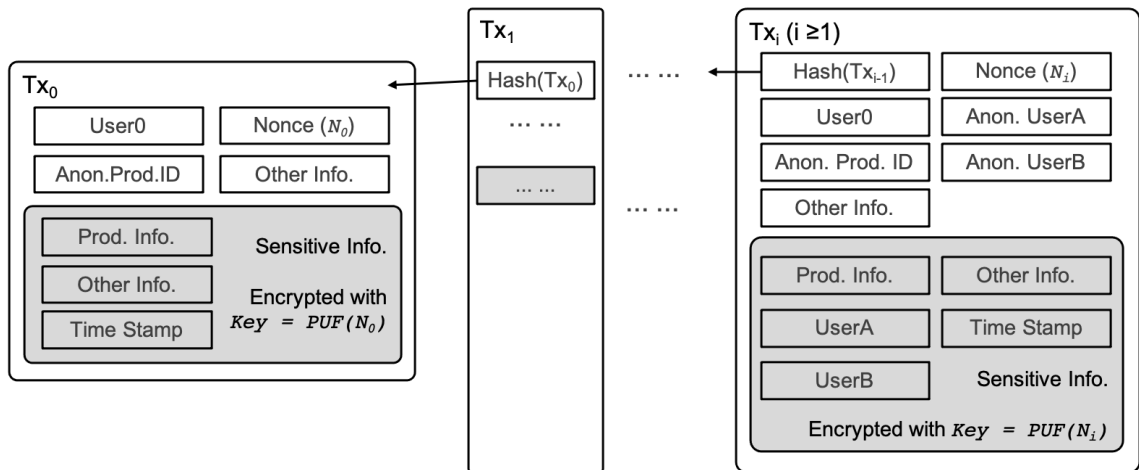


Figure 5.4: Data chain (i.e., ledger) structure.

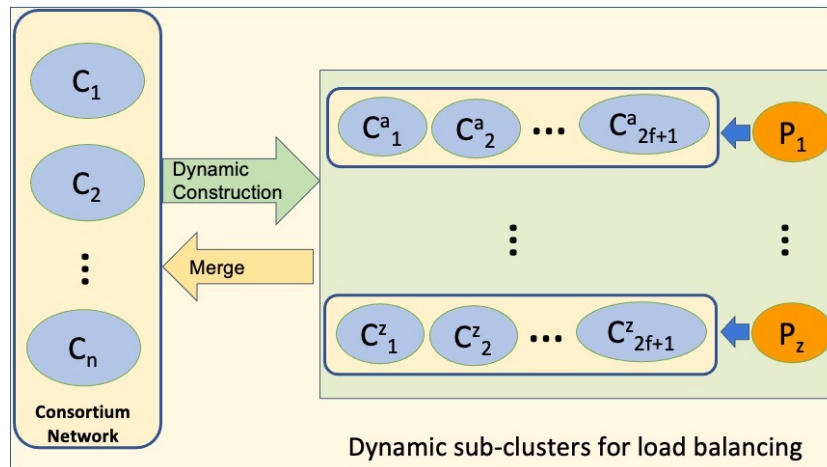


Figure 5.5: HWCOVER manages load balance through dynamic sub-clusters. Sub-clusters get merged after a job is finished.

information (e.g., product details).

The HCRPs are different for each consortium Nodes, which have to be uploaded through P2P methods, while other data can be uploaded through broadcasting (with consensus, discussed in §5.4.4) on the consortium network. The registration is considered as the pseudo-transaction (Tx_0), as shown in Figure 5.4. Once the registration is completed, Tx_0 is generated, and the product is ready for real transactions (see §5.4.3).

5.4.3 Transaction

For the Transaction, buyer (*UserB*) access is open to the public, which means anyone with minimum resources (e.g., internet access and chip reading) can participate in the transaction. After a successful transaction, *UserB* becomes the device owner and thus plays the role of *UserA* in the next transaction. In a transaction, the previous owner (*UserA*) is untrusted while the next owner (*UserB*) is trusted, as *UserB* will be responsible for the Device once the transaction is accepted. *UserB* has the motivation to verify the Device and related transaction history.

In HWCOVER, a transaction is initialized from validation request submitted by *UserB* as shown in Figure 5.3(b). After receiving the device and the related information from *UserA*, *UserB* checks the status of *Nodes*, selects idle or less busy ones, and submits validation request. We design a load balancer to assign the workload across the consortium network in an equitable manner, as shown in Figure 5.5. The load balancer assists in three ways: (i) enables *UserB* to create a temporary cluster with a few idle nodes to assign a validation request job, (ii) helps to process multiple transactions in parallel through independent temporary clusters, (iii) merge the busy nodes from the temporary clusters in the idle network once a job is finished. HWCOVER assigns $2f + 1$ number of nodes in a temporary cluster, where f is the maximum number of faulty nodes allowed and set by the system at the beginning.

As summarized in Figure 5.3(b), based on the submitted validation request, the selected *Nodes* (i.e., cluster) send one or more random PUF challenges to *UserB*. *UserB* collects PUF responses accordingly, performs hashing with *Nodes*' vectors (*NVs*), and sends the results (HCRPs) to corresponding *Nodes*. If the results match and the consensus (see §5.4.4)

is achieved among *Nodes*, the consortium network acknowledges that *UserB* physically holds the *Device* and releases the transaction records, including the encrypted sensitive information to *UserB*. With unencrypted nonces and the PUF on the *Device*, *UserB* is able to decrypt the transaction records and access the sensitive details. If the transaction is accepted, a new transaction (Tx_i) is generated and added to the data chain (synced by the consortium network) as depicted in Figure 5.4.

Compared to other blockchain systems (e.g., PoW and Tendermint), HWCOVER requires more efforts on *Users'* side to submit verification information during a transaction, as *UserB* needs to perform hashed CRP responses for each selected *Node*, just like what *User0* does during the registration. The difference between registration and transaction is that during transaction validation, only a small part of *Nodes* are selected for a single transaction, and a limited number of challenges need to be responded to. Therefore, the transaction uploading load on *UserB* is remarkably lower than the registration uploading load on *User0*. The exact overhead communication cost (compared to PoW and Tendermint) depends on the number of selected *Nodes* for verification, which will be thoroughly discussed in §5.4.4 and §5.5.5.

Protocol 16 (i.e., consensus manager) steers the entire transaction validation and persisting process. After finishing the validation for a specific transaction by attaining the maximum consensus through Protocol 14 (see §5.4.4), finally, Protocol 15 assists HWCOVER to persist the block along with all the necessary information such as new owner's information, block's hash, and product id. Protocol 15 also synchronizes all the *Nodes'* local ledgers as well as the shared ledger. Note that to avoid excessive communication overhead, during the synchronization phase, HWCOVER touches the shared ledger only once for a specific block of transactions.

Protocol 14 Transaction Validation

Require: Consortium nodes N where the i -th node is N^i ; Clusters C where the i -th cluster is C^i ; Clustered consortium nodes N_C in a cluster C^i where the i -th node is N_C^i ; N_B^i the blockchain copy on N^i ; N_{BC}^i the blockchain copy on N_C^i ; p_r the product validation request; q_r the ready queue; w_q the waiting queue; new nodes m ; block decision D where D_b is consensus, D_c is the list of verifier nodes for product p , and D_o is the owner of product p .

Ensure: 100% of $VNodes$ or majority of $MNodes$ have validated p_r with local blockchain in consortium nodes ledger N_B and reach consensus D_b for the product p .

```

1: function VALIDATION( $p_r, U^i$ )
2:   while  $q_r \neq \emptyset$  do
3:     if  $C^i$  is available then
4:       Pop a  $p_r$  from  $q_r$ 
5:       Assign the  $p_r$  to  $C_i$ 
6:       while  $|VNodes| \leq N_C$  do                                     ▶ First round
7:         if  $p_r$  is validated with  $N_{BC}^i$  then
8:            $VNodes \leftarrow VNodes \cup N_C^i$ 
9:         end if
10:      end while
11:      if  $|VNodes| < N_C$  then                                       ▶ Second round
12:         $|MNodes| \leftarrow |VNodes|$ 
13:         $C^i \leftarrow C^i \cup |m|$                                        ▶ Add nodes
14:        while  $|MNodes| \leq \frac{N_C}{2}$  do
15:          Continue validation
16:        end while
17:      end if
18:      if  $|VNodes| \geq N_C \vee |MNodes| > \frac{N_C}{2}$  then
19:         $D_s^i \leftarrow true$                                              ▶ Product is valid
20:         $D_c^i \leftarrow VNodes$                                          ▶ Record validators
21:         $D_h^i \leftarrow uid$                                            ▶ Set hash
22:         $D_o^i \leftarrow U^i$                                            ▶ Transfer ownership
23:      else
24:        Push  $p_r$  to  $w_q$  to start over
25:      end if
26:    else
27:      Push  $p_r$  to  $q_r$                                                ▶ Push to ready queue
28:    end if
29:    Return  $D^i$ 
30:  end while
31: end function

```

5.4.4 Consensus

While PUF as HSM is utilized to prevent counterfeiting attacks from the outside of the consortium network, malicious consortium node attacking is another concern. It occurs when

Protocol 15 Persist Transaction

Require: Consortium nodes N where the i -th node is N^i ; S_B the shared blockchain; N_B^i the blockchain copy on N^i ; a new block b ; block decision D^i where D_s^i is consensus, D_o^i is owner, D_h^i is hash and D_c^i is cluster for product p^i .

Ensure: b is stored in the shared ledger and all consortium nodes N .

```

1: function PERSIST( $D^i, p^i$ )
2:   if  $|D_s^i| = true$  then
3:      $b_p \leftarrow p^i$ 
4:      $b_o \leftarrow D_o^i$ 
5:      $b_h \leftarrow D_h^i$ 
6:      $S_B \leftarrow S_B \cup b$  ▷ Store on shared blockchain
7:     for  $N_i \in N$  do
8:        $N_B^i \leftarrow N_B^i \cup b$  ▷ Store on consortium node
9:     end for
10:  end if
11: end function

```

Protocol 16 HWCover Consensus Manager

Require: Consumers U where the i -th node is U^i ; Clusters C where the i -th cluster is C^i ; Consortium nodes N where the i -th node is N^i ; N_B^i the blockchain copy on N^i ; a new block b ; Products p where the i -th product p^i ; Product requests p_{rq} where the i -th request p_{rq}^i ; D^i is verification result.

Ensure: All products p gets validated based on challenge-response pair.

```

1: function CONSENSUS-MANAGER( $VNodes, U_i, N$ )
2:   while  $p_{rq} \notin \emptyset$  do
3:     Assign a request  $p_{rq}^i$  to a cluster  $C_i$ 
4:      $D^i \leftarrow VALIDATION(p_{rq}^i, U^i)$  ▷ Call Protocol 14
5:     if  $D_s^i$  is true then
6:       PERSIST( $D^i, p^i$ ) ▷ Call Protocol 15
7:     end if
8:   end while
9: end function

```

one or more consortium *Nodes* are controlled or interfered with by adversaries. Assume the adversaries can manipulate the communication to/from the affected *Nodes* and vote dishonestly during the consensus. Our first security goal is to eliminate the probability of a manipulated voting result (error) within a low range, saying, 1×10^{-5} (i.e., 1 out of 100,000). Given the reported counterfeiting rate in the real world (up to 15% [127]), 1×10^{-5} is believed to be sufficiently safe. Our second goal is to keep the system available for the supply chain even a considerable percentage of the consortium *Nodes* are compromised.

In traditional blockchain systems, all nodes or full nodes are expected to participate in reaching a consensus. Considering that the value of each transaction may vary, it is not economically acceptable to call all *Node* devoted in every single transaction for the validation, especially when the product is low cost (e.g., less than \$100) and the number of total *Nodes* is significant (e.g., hundreds). Our consensus protocol is equipped with 2-round voting as illustrated in Protocol 14 to reduce the workload on the consortium network. The conception is to select a dynamic cluster of a comparably small number of *Nodes* for the consensus of one transaction in the most time as shown in Figure 5.5. Given the reduced number of selected *Nodes*, the chance of the adversaries compromising sufficient (e.g., 51%) votes from the selected *Nodes* is increased. To compensate for the security lost due to the reduced number of voting *Nodes*, we set a stricter security requirement that enforces an agreement by **all** of the selected *Nodes*. Meanwhile, the *Node* selection has to be cryptographically random. Otherwise, the attacker can predict the cluster *Nodes* and apply intensively focused attacking, which puts the cluster at high risk.

If there are inconsistent votes in the cluster, in the 2nd-round voting, HWCOVER applies a 51% policy where the cluster will be expanded to include statistically sufficient *Nodes* or optionally all *Nodes*. How frequently we need the 2nd round depends on the portion of malicious/incorrect *Nodes* in the blockchain and the number of selected *Nodes* in the 1st round. Based on the frequency, the total computational cost on two rounds of consensus processing can either be saved (expected the most time) or cause extra overhead (in rare cases, when 1st-round should be skipped as an agreement is hardly achieved in the 1st-round). We provide a thorough analysis of both rounds of the consensus process in §5.5.5.

Space Complexity of Transaction Persistence. We also calculate the blockchain storage requirement by the HWCOVER as illustrated in Table 5.1 and compare it with the current blockchain protocols. Let $|B|$ denote the size of the blockchain, the total number of trans-

Table 5.1
COMPARISON AMONG POW, TENDERMINT, AND HWCOWER.

Protocol	PoW	Tendermint	HWCOWER
Time	$O(n)$	$O(n^2)$	$O(\log n)$
Space	$O(n \cdot B)$	$O(n \cdot B)$	$O(n \cdot B)$
Privacy	×	×	✓
Concurrency	×	×	✓

actions in the blockchain $|B|$ is t , and the total number of nodes in a consortium network is n . Each node stores $O(|B|)$ amount of data which is on par with the proof-of-work (PoW) and practical byzantine fault tolerance (PBFT) protocols. A node stores each transaction encapsulated in a block in the local ledger. Note that we have a shared ledger that stores the transaction first after receiving the return message from a client before synchronizing all the nodes. Whenever the shared ledger is updated, the *transaction manager* throws a trigger that initiates the update procedure; hence, each node synchronizes the local ledgers against the shared ledger. As the nodes do not perform peer-to-peer communication during the synchronization phase, the cost of data synchronization lies within a reasonable limit as it requires only $n - m$ messages.

Time Complexity of Consensus. Without loss of generality, we calculate the complexity of consensus per transaction, as shown in Table 5.1 and compare it with the vanilla blockchain protocols. HWCOWER dynamically constructs several clusters C with a set of random nodes from the consortium network n to manage c numbers of transactions in parallel. Let Tx denote a transaction that belongs to a dynamically constructed cluster C with size m . First, the user sends Tx to the HWCOWER. HWCOWER then routes the Tx to an available cluster C , which incurs a computation overhead of $O(m)$, where $m = \log(n)$. The nodes in the responsible cluster validate the transaction in parallel before communicating with the HWCOWER client deployed at the user end with the validation result (i.e., encrypted vote) with m messages. Finally, the HWCOWER client aggregates the votes and

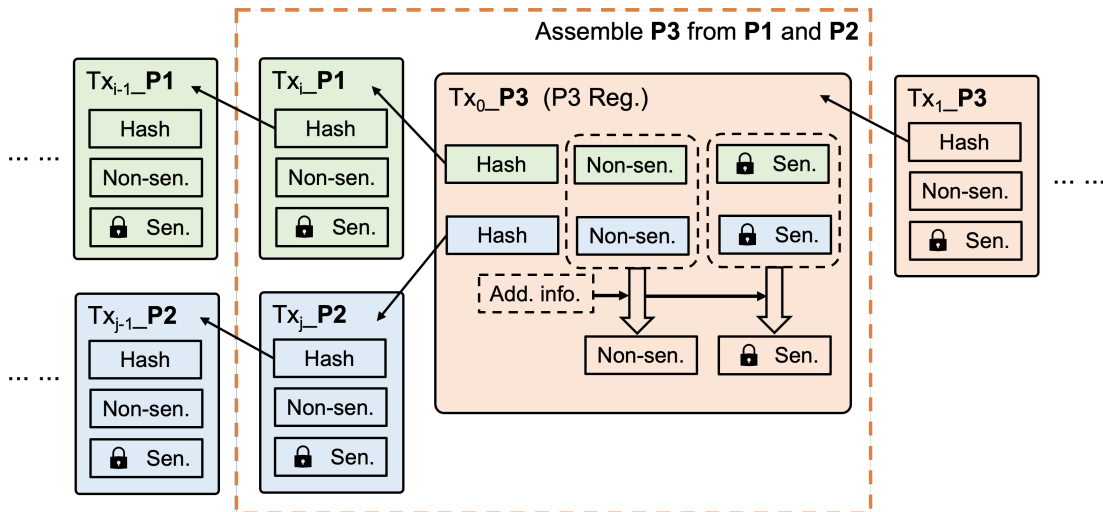


Figure 5.6: Assembling of related previous transactions into a new transaction.

responds to the HWCORVER server at a constant rate. Therefore, the communication and computation overhead for a transaction in total is $m + m + 1 = O(m) = O(\log n)$.

5.4.5 Assembling and Disassembling

Modern electronic systems usually contain multiple IC chips. When an assembler builds a system or sub-system with several chips, the related transactions shall be merged into a new one with the assembling as well to allow continuous tracking of the chips. Meanwhile, the newly assembled system shall be considered as a new *Device* and be registered. As illustrated in Fig. 5.6, the assembler will receive *Devices* from suppliers as the new owner (*UserB*) of the *Devices* in the related transactions (e.g., Tx_P1 and Tx_P2), and then register the assembled system as the original owner of the assembled system (i.e., *User0* bringing in $P3$) in the new transaction. Assembling more than two *Devices* can be done similarly. The information used for the assembled system ($P3$) verification will be defined from the component *Devices*' ($P1$ and $P2$) PUF CRPs like the approaches described in [76, 153]. When a larger system is disassembled into smaller parts for future transac-

tions, the workflow is also similar. It is worth noting that each part needs to contain an independent PUF module to be qualified for future transactions supported by HWCOVER.

5.5 Security Discussion

5.5.1 Unauthorized Devices

Given properly designed PUFs, most counterfeiting attacks, including cloning, recycling, and overproducing, can be efficiently restricted by checking PUF's CRPs [55, 112]. For transactions involving genuine devices but obtained through improper sources, such as stealing or concealed recycling, both CRPs and transaction records stored on the blockchain will be used to check whether the *Device* is in legal status. According to the registration and transaction protocol, authorized *Device* can be easily traced back to the original owner(*User0*). We assume *User0* and *UserB* of each transaction are honest. Otherwise, in the next transaction, the next buyer will receive mismatched results (see §5.5.3) from HWCOVER and reject the transaction. Then if *User0* and *UserB* perform appropriate registration and validation through the consortium, respectively, the adversary has to obtain control of users' consoles to bring in an unauthorized *Device*. This refers to the topic of console security, which is out of the scope of this chapter. In addition, even if such an attack succeeds, related transactions can be flagged to terminate the future circulation of the corresponding *Device*.

5.5.2 PUF-data Leakage

The leakage of PUF-related data, or more specifically, the hashed CRPs, can take place on the blockchain *Nodes*. Extracted HCRPs can be used for counterfeited devices to pass verification, but just the HCRPs extracted from one *Nodes* is far from enough to achieve this purpose. As explained in §5.4.4, multiple *Nodes* are randomly selected, verifying the *Device* with each one's unique HCRPs. The adversary will need to predict which *Nodes* to be selected, and then conduct the corresponding hash values from the obtained hashed CRPs. As long as the *Nodes* selection follows a cryptographically secure random or pseudo-random algorithm, and the hashing adopts secure hash algorithms, it is hardly possible for the adversary to deploy meaningful attacks.

5.5.3 Malicious Users

As mentioned in §5.4, we assume the original *Device* owner (*User0*) and the buyer in a transaction (*UserB*) are benign. *User0* only registers its brand products. Therefore it is reasonable to assume it will behave to establish a reputation and revenue. Otherwise, revenue lost, and legal sanctions will follow. While for the buyers, who can be personal users or transient small companies, it is less reliable to leave the security to the market and policies. Fortunately, the manipulation they can do in each transaction is limited. Because the blockchain is linked by the hash pointer, the source, basic information, and history owners of the *Device* has to be consistent across the chain, which is distributed and cannot be easily tampered with. The malicious *User* only generated the most recent transaction block (saying Tx_i), where only newly added information after Tx_{i-1} can be forged. Such information is highly limited, which may include the self-claimed buyer's identity (*UserB_i*)

and the recent *Device* working status. These kinds of information cannot be guaranteed by HWCOVER but rely on legal obligations and future buyers' attention.

5.5.4 Privacy Concerns

The privacy here refers to the sensitive information that is encrypted in each transaction (see Figure 5.4). It is protected through access control and encryption, as described in §5.4.2. To extract the sensitive data, an adversary needs to steal the current owner's account information to pass the identity verification first, and then request the corresponding *Device* transaction records from the consortium. After getting the records, the adversary also needs to generate the correct key to decrypt the data, which requires either physical access to the PUF or successfully modeling the corresponding PUF. Assuming the PUF is properly designed, it seems unlikely for an adversary to succeed in all of the above steps and achieve privacy-compromising. In addition, even the adversary succeeded, the affected record is limited to a single device/transaction. This is considered a trivial reward to the attacker, which helps to prohibit the attacking from the aspect of attacking efficiency consideration.

5.5.5 Manipulated Voting

In this section, we discuss the tolerance of manipulated voting during the consensus achieving. Note N , N_b , and N_g as the numbers of total consortium *Nodes*, compromised ("bad") consortium *Nodes*, and "good" consortium *Nodes*, respectively; M , M_b , and M_g as the numbers of total *Nodes* in a cluster, "bad" cluster *Nodes*, and "good" cluster *Nodes*, respectively. According to the design explained in §5.4.4, all M cluster *Nodes* have to be

malicious to generate a tampered result in the 1st round, while only a simple majority of malicious *Nodes* are needed to generate a tampered result in the 2nd round. Given the rate ($x = Nb/N$) of the compromised *Nodes* over all of the consortium *Nodes*, we have the probabilities ($Pr_1(x), Pr_2(x)$) of getting a manipulated consensus result in the 1st round and 2nd round consensus:

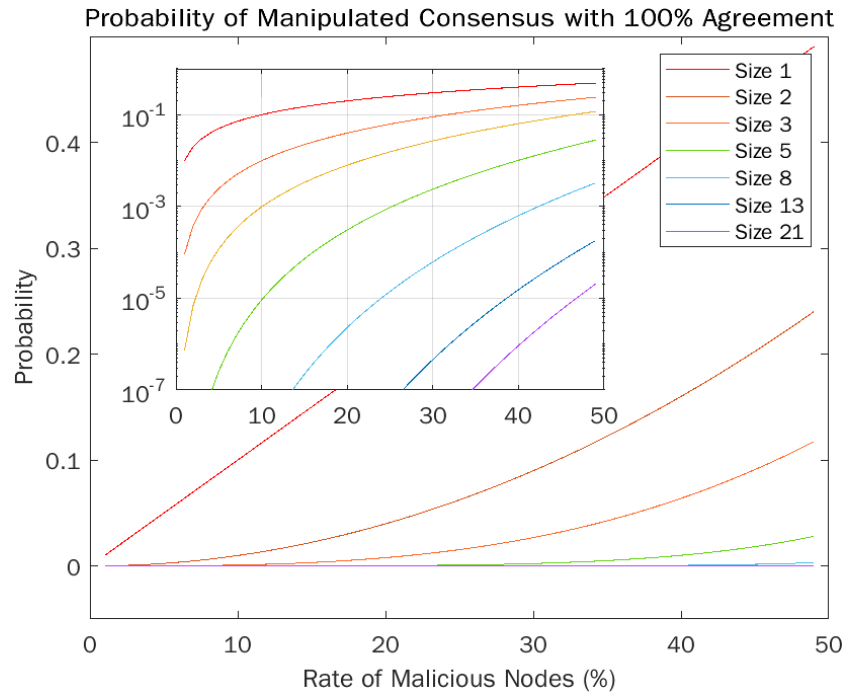
$$Pr_1(x) = \begin{cases} \frac{C(Nb, M)}{C(Nt, M)}, & \text{if } Nb \geq M \\ 0, & \text{otherwise} \end{cases} \quad (5.1)$$

$$Pr_2(x) = \begin{cases} \frac{\sum_{Mb=f+1}^M C(Nb, Mb) \cdot C(Ng, Mg)}{C(N, M)}, & \text{if } Nb > f \\ 0, & \text{otherwise} \end{cases} \quad (5.2)$$

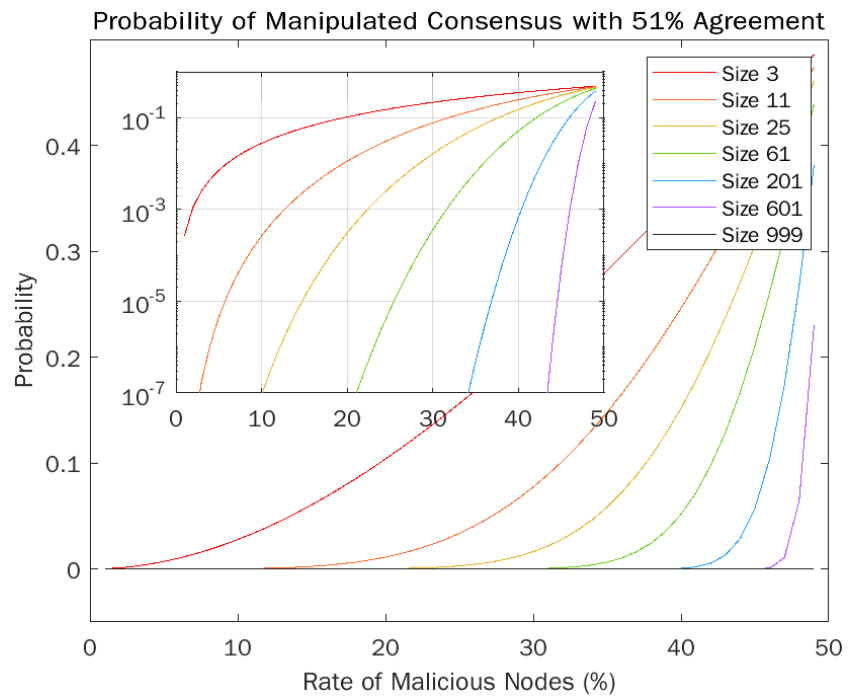
where C is the combination formula (i.e., $C(n, k) = \frac{n!}{k!(n-k)!}$), $Nb = N \cdot x$, $N = Nb + Ng$, and $M = Mb + Mg$. It is also worth noting that due to the majority requirement for the 2nd-round voting, cluster size $M = 2f + 1$, where f is a positive integer.

We set $N = 999$ as it represents a realistic scale of real-world global supply chain deployments [4], and plot the $Pr_{r1}(x)$ vs. x and $Pr_{r2}(x)$ vs. x for different cluster sizes in Figures 5.7(a) and 5.7(b), respectively. In Figure 5.7(a), we can see that if a maximum of 10%, 20%, 30%, or 40% compromised consortium nodes are allowed, we have to increase the minimum cluster size (M) from 5 to 8, 10, or 13, accordingly, to maintain the error rate below 1×10^{-5} when 100% agreement is required. Again, we would like to emphasize that this requirement is to prevent a “bad” consensus. It is not equivalent to achieving a “good” consensus.

As long as there are conflicting votes in the 1st round, consensus processing enters the 2nd



(a) First round.



(b) Second round.

Figure 5.7: HWCOVER protocol follows two rounds to guarantee consensus. There are 1,000 consortium nodes in total and the cluster size changes from 3 to 101.

round. Figure 5.7(b) shows that in order to achieve a low error tolerance and successfully achieve consensus at the same time, we need to expand the cluster size and accept the majority decision. It should be pointed out that a 10% or higher compromised rate is practically rare in a consortium network when we assume the consortium nodes are run by listed qualified agencies.

5.5.6 Ledger Tampering

The question may arise what if the shared blockchain gets compromised? The HWCOVER encapsulates a transaction in a block before appending it to the ledger. The block's hash is determined from one of the hashes provided for the validated transaction by the verifiers (i.e., consortium *Nodes*) who provided valid *hashed-votes*. The honest *Nodes* can easily verify if the persisted blockchain or a block's hash has been tampered with. For simplicity, in this phase, we take the traditional public-private key authentication approach. When a *Node* validates a transaction, it generates *hashed-vote* using the *Node*'s private key. When a new *Node* joins the network, its public key is already known to the other *Node*. The fellow *Node* can always verify the authenticity of a transaction encapsulated in a block by re-computing the hash based on the public key and can match with the verifier list.

If an adversary tries to forge the persisted shared blockchain, it needs to re-compute the hashes of each block using the private key of the respective consortium nodes, thanks to the immutability property of the hashing mechanism. A minor change in a block's hash will impact the hashes of all the blocks in the entire blockchain. As the private keys of the consortium nodes are unknown to the adversary, it is quite impossible for the adversary to forge the shared ledger. Besides, during each synchronization process between the shared

ledger and the nodes, if the majority of nodes find a mismatch between the shared ledger and the local chains, the shared ledger gets replaced with the longest valid ledger to keep up the blockchain service.

5.6 Performance Evaluation

We report the performance of the proposed techniques both theoretically and experimentally. We compare the performance of the proposed HWCover against multiple baselines, including the state-of-the-art blockchain systems. Our goal is to analyze with minimal resource consumption whether HWCover can exhibit better performance while providing guaranteed support for securing the privacy of sensitive data over the traditional blockchains, which are based on PoW and PBFT protocols.

5.6.1 Implementation and Experimental Setup

We have implemented a prototype system of the proposed blockchain architecture and consensus protocols with Python, C++, and Shell scripts. We leverage the Python threading library to implement the threading and locking mechanism. The experimental results reported here are from the implementation consisting of about 1,500 lines of code.

We carried out a performance evaluation on a consortium cluster comprised of 58 physical nodes interconnected with FDR InfiniBand. Each physical node is equipped with an Intel Core-i7 2.6 GHz 32-core CPU along with 296 GB 2400 MHz DDR4 memory; hence we can map up to 32 consortium nodes through the user-level threads in a single physical node.

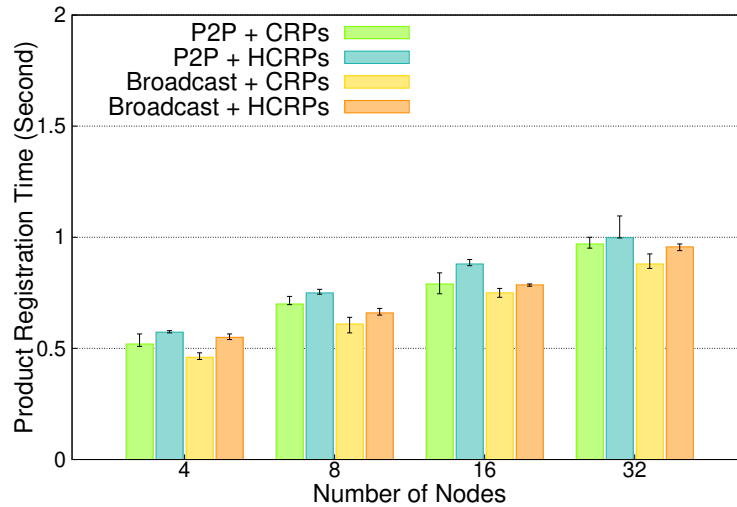


Figure 5.8: A product registration cost comparison both in broadcast and parallel peer-to-peer (p2p) method at various scales.

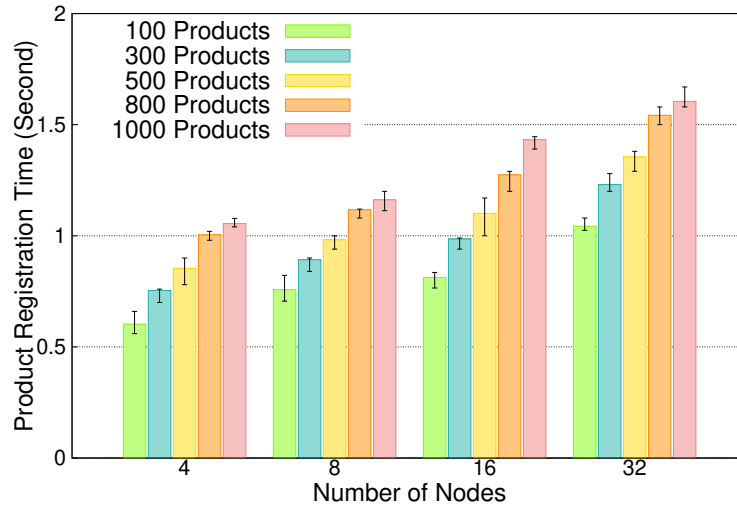


Figure 5.9: Different number of products registration cost at various scales through batching in parallel peer-to-peer (p2p) method.

However, we reserve a few cores in each physical node for the operating systems and other resources to receive optimum performance. Each physical node is installed with CentOS Linux 7 (Core), Python 3.7.0, MPICH 3.4.3, and NumPy 1.15.4.

5.6.2 Cost during Registration

In HWCOVER, to avoid vulnerabilities due to PUF CRPs leakage, all the original PUF CRPs are replaced with PUF challenges and responses hashed (HCRPs) with individual consortium *Node*'s vector before registering the product to the corresponding nodes in a parallel P2P method. To measure the HCRP registration overhead through the parallel P2P method, we compare three baselines against the proposed registration process of HCRPs. The first baseline system is based on P2P registration with plain CRPs (i.e., no hashing of CRPs). The second one performs registration through broadcasting with plain CRPs, rather than P2P. The third baseline follows broadcasting with hashed CRPs (i.e., HCRPs) to complete the registration process. In all baselines, we produce 100 CRPs for a single product.

Figure 5.8 shows that for a single product, the proposed registration method incurs negligible overhead communication cost compared to the other baselines. To further investigate the overhead, we perform the registration process of HCRPs for the various number of products through the parallel P2P method. During this communication, we batch the products in a single packet, which may be closer to the situation in the real world. Figure 5.9 illustrates that even with the increasing workload at a larger scale, the registration time does not increase linearly and remains within a reasonable limit (i.e., around 1.6 seconds at 32 nodes for 1000 products). This is because, in the parallel P2P method, a supplier (i.e., *User0*) client establishes parallel connections through HWCOVER API service to communicate with each node in the consortium network to pass the products CRPs in batches. For parallel connection establishment, HWCOVER leverages MPI parallel programming module.

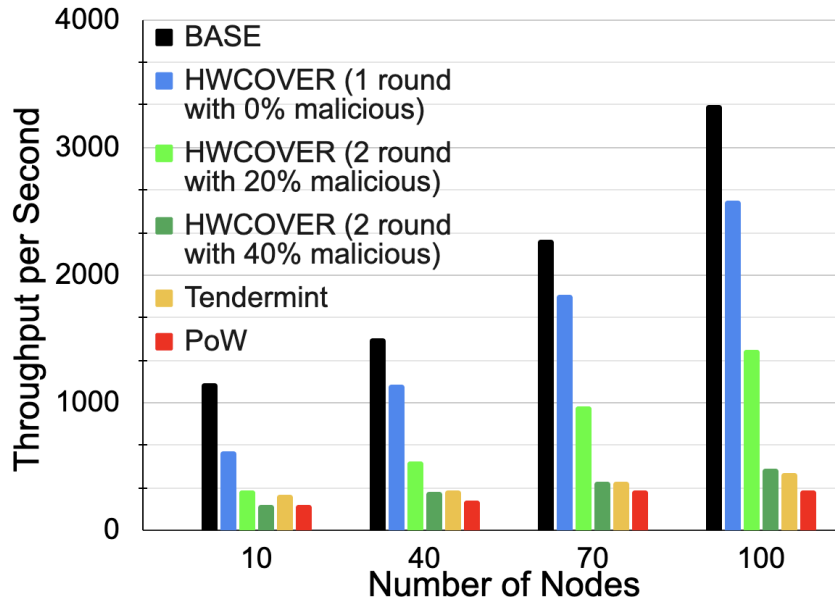


Figure 5.10: Throughput comparison. PoW is set with minimum difficulty (= 1).

5.6.3 Throughput of Transactions

To measure throughput, we issue 10,000 transactions at various scales with default block size 4 in all the evaluated systems and compare with the proposed HWCOVER (1 round with 0% malicious nodes) system. The first baseline system (i.e., BASE) is based on the central authority to verify the transactions, whereas the second one is implemented with the traditional PoW protocol of minimum difficulty (= 1). Note that Bitcoin applies a much higher difficulty (32) for the race among nodes (as known as miners) to append a new block of transactions. We inject the HWCOVER protocol in the Tendermint blockchain to develop the third baseline. In the case of the fourth (2-round with 20% malicious nodes) and fifth (2-round with 40% malicious nodes) baselines, we inject 20% and 40% malicious nodes in the HWCOVER system, respectively. The malicious nodes act differently against the honest nodes by providing invalid votes. The aim of having a different number of malicious nodes is to observe the liveness of the HWCOVER protocol during the transaction validation process while preserving a reasonable performance with varying scales of nodes.

Table 5.2
REQUIRED CLUSTER SIZE TO KEEP CONSENSUS ERROR LOWER THAN 1×10^{-5} IN TWO ROUNDS.

Malic. Rate	10%	20%	30%	40%
Total nodes: 10				
Rd. 1 (100%)	2	3	4	5
Rd. 2 (51%)	3	5	7	9
Total nodes: 40				
Rd. 1 (100%)	5	7	8	10
Rd. 2 (51%)	9	17	25	33
Total nodes: 70				
Rd. 1 (100%)	5	7	9	11
Rd. 2 (51%)	13	23	39	57
Total nodes: 100				
Rd. 1 (100%)	5	7	9	12
Rd. 2 (51%)	13	27	49	79
Total nodes: 1000				
Rd. 1 (100%)	5	8	10	13
Rd. 2 (51%)	17	41	95	311

Limited by the computational resources in our lab, the maximum node number tested in our experiments is 100.

When there are no malicious nodes, HWCOVER successfully achieves consensus in the lightweight 1st round. As shown in Figure 5.10, it (“1-round with 0% malicious”) is almost 4× and 5× faster than Tendermint and PoW, respectively, at various scales. In PoW, the nodes perform the costly computation to solve a puzzle difficulty in order to select the leader. Although Tendermint does not employ heavy computation, it requires 100% nodes to participate in the consensus process. On the other side, the centralized baseline performs better (i.e., almost 2.5× more throughput at a lower scale) compared to HWCOVER. This is because the centralized baseline does not validate the block through distributed nodes, whereas HWCOVER leverages a dynamic cluster of nodes during the validation of a transaction to attain the high legitimacy of a product.

In HWCOVER 1-round with 0% malicious nodes (i.e., Rd. 1), the proposed protocol requires 100% valid votes from each dynamic cluster to finish the validation process during the first round. The number of cluster nodes under different conditions are calculated by

Equations (5.1) and (5.2), and used for the throughput testing, as listed in Table 5.2. As the requirement of 100% agreement is strong, the probability of tampered consensus can be kept low with limited cluster size in the 1st round. Therefore, the processing time on the 1st round, which depends on the cluster size, should increase negligibly even though the total node number increases a lot (e.g., from 100 to 1000).

The cost increases when there are malicious nodes in the 1st round cluster, and the HWCOVER protocol requires to shift to the 2nd round, where we add more nodes in a cluster and only need 51% agreement. With the presence of malicious nodes, the effective consensus processing time includes the time spent on the 1st round and the 2nd round. The disagreement will occur more often with the rate of increase of malicious nodes, making the 2nd round call more frequent. As shown in Figure 5.10, when the malicious node rate increases up to 20% and 40%, the HWCOVER throughput decreases, getting close to the throughput of Tendermint and PoW-based consensus at lower scales. The extra time spent by HWCOVER on the 1st round does not help the throughput due to the disagreement, but slows down the total processing, resulting in outperformed by Tendermint at a lower scale. However, the good news is the performance in both HWCOVER (2-round with 20% malicious) and HWCOVER (2-round with 40% malicious) gets better along with the scaling of nodes.

To be more specific, the 2nd round in HWCOVER does not require all nodes to participate, as the protocol is not pursuing a strict zero-tolerance of tampered consensus. Instead, it targets to control the probability of tampered consensus within a preset tolerance, such as 1×10^{-5} . Compared to the design in Tendermint, which pursues zero-tolerance as long as the malicious node rate is no more than 33%, the tolerance in the HWCOVER 2nd round releases a part of nodes for parallel processing to improve the throughput. According to the calculation results, the effect of the tolerance on throughput improvement becomes more

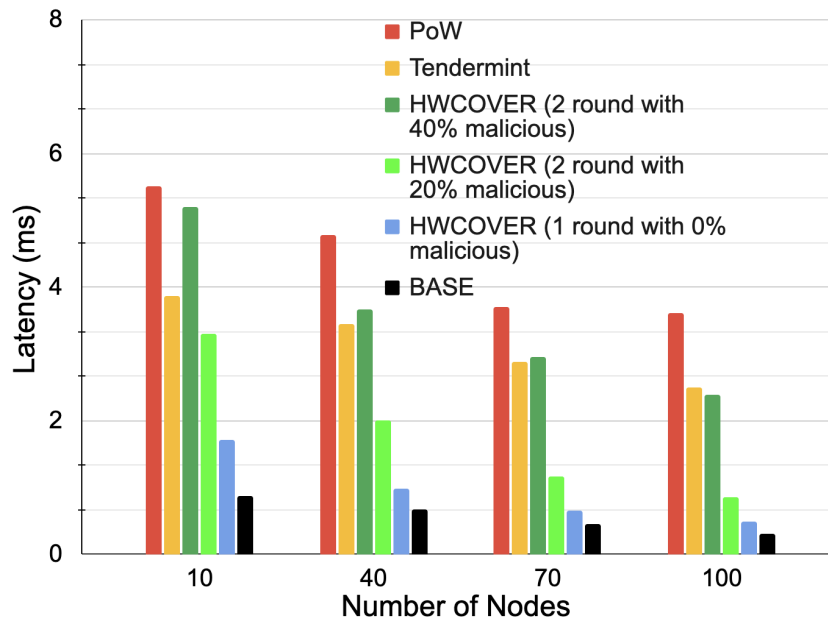


Figure 5.11: Latency comparison in various systems.

impressive with the increase of the total node number. For example, when the node number is 10, it requires 9 nodes in the 2nd round (for 1×10^{-5} error rate), taking 90% nodes to process the consensus. While the node number is 1000, it only requires 311 nodes in the 2nd round, taking only 31% nodes to process the consensus, which allows HWCOVER to outperform Tendermint even with the presence of a significant number of malicious nodes.

5.6.4 Latency

HWCOVER employs a specially crafted blockchain framework to overcome the limitation of the present centralized supply chain management. However, having a blockchain-based validation should not cause significant overhead. To determine the transaction latency, we issue a block of 4 transactions in all systems and observe the latency time by each of them.

Figure 5.11 depicts that both PoW with minimum difficulty (i.e., difficulty = 1) and Ten-

dermint exhibit almost $10.6\times$ and $8\times$ slower performance respectively than the centralized processing system. On the other hand, HWCOVER with 0% malicious nodes (i.e., 1 round) shows only $1.5\times$ slower performance at a higher scale (i.e., 100 nodes) compared to the centralized system. Although with two rounds, HWCOVER with 20% and 40% malicious nodes take a little longer ($1.6\times$ and $4.2\times$ respectively) to process a transaction compared to round one (i.e., 1 round with 0% malicious), these are the worst cases, and most transactions are likely to get validated during round one. It is noticeable that even with a significant number of malicious nodes, the HWCOVER outperforms the Tendermint and PoW on a larger scale.

5.7 Summary

This chapter presents a lightweight, blockchain-like decentralized electronic supply chain manager, namely, HWCOVER, equipped with a set of new protocols codesigned with the latest hardware identification techniques to achieve privacy and high product trustworthiness in a resource-constrained environment. At the core of the system lies a new set of consensus protocols and a series of optimizations specifically crafted to the unique requirements posed by supply chains. We developed a prototype system based on proposed protocols and deployed on a cluster of 58 nodes. The system is compared to both state-of-the-art blockchain protocols and the conventional supply chain management's centralized approach. Experimental results demonstrate that compared to vanilla decentralized solutions, HWCOVER incurs well-controlled overhead cost in the product registration ($< 10\%$) while outperforming the state-of-the-art by up to $5\times$ (in the best case) in the case of transaction throughput.

CHAPTER 6

CONCLUSIONS AND FUTURE WORKS

6.1 Conclusions

In this dissertation, we have studied several lightweight decentralized protocols to adopt blockchain to guarantee reliable data management in high-performance computing (HPC) systems and resource constraint environments such as the edge computing paradigm. In Chapter 2, first, we studied SciChain, which enables plugging in a blockchain-like decentralized mechanism in scientific provenance to ensure data fidelity. SciChain not only guarantees the data fidelity of the provenance data but also allows leveraging the blockchain in HPC through a shared-storage model. Experimental results showed that SciChain guaranteed trustworthy data provenance while incurring orders of magnitude lower overhead than existing solutions.

In Chapter 3, we present BAASH, an optimal blockchain service equipped with a parallel processing module on top of the MPI programming model. BAASH not only supports the common scientific programming model (i.e., MPI) that supplies scalability but also comes with a unique fault tolerance mechanism that assists to keep up the reliable blockchain service with an extra layer of error handling support on top of native MPI error handlers. The system prototype of the proposed BAASH framework demonstrates both high performance and reliability compared to state-of-the-art solutions.

In Chapter 4, we formulated a blockchain-like decentralized protocol, namely DEAN

that empowers reliable data management in edge computing through a lightweight unique leader election approach partly enlightened by the scale-free graph development mechanism. The three parallel processes in DEAN overcome all the challenges that create obstacles to attaching blockchain in any resource constraint ecosystem. Experimental results show that along with the improved data fidelity, the system prototype of DEAN delivers orders of magnitudes higher performance than the state-of-the-art alternatives.

In chapter 5, we present HWCOVER set up with a set of lightweight blockchain protocols codesigned with the latest hardware identification techniques (i.e., PUF) to achieve privacy and high product trustworthiness in an electronic supply chain ecosystem. The system prototype of HWCOVER incurs reasonable overhead during product registration ($< 10\%$) while outperforming the state-of-the-art solutions in the case of transaction throughput.

6.2 Future works

Our future work is two-fold. First, we desire to explore various data privacy preservation techniques in blockchain for HPC or cloud infrastructures. At present, blockchain is being leveraged as a black box along with several vanilla encryption methods, which are extremely heavy, unreliable, and not equipped with the unique requirements of having blockchain in HPC or any other resource constraint ecosystem. We aim to discover distinctive lightweight privacy preservation methods that will permit the application of blockchain effortlessly along with lightweight data privacy mechanisms to guarantee the ultimate reliable data management.

Second, we will develop a theoretical foundation of the proposed protocols for supply

chains and conduct more in-depth experiments to determine the performance overhead against the baseline systems and robustness of the HWCOVER in front of the arbitrary number of nodes failure. We will also focus on the development of a more generalized supply chain blockchain framework that can be integrated as a plug-n-play module for secure data management in diverse supply chain ecosystems.

BIBLIOGRAPHY

- [1] U.S. Senate Committee on Armed Services. May 2012. Inquiry into counterfeit electronic parts in the Department of Defence supply chain. <https://www.armed-services.senate.gov/download/inquiry-into-counterfeit-electronic-parts-in-the-department-of-defense-supply-chain>, Accessed 2020.
- [2] Saveen A Abeyratne and Radmehr P Monfared. Blockchain ready manufacturing supply chain using distributed ledger. *International Journal of Research in Engineering and Technology*, 5(9):1–10, 2016.
- [3] Advancing the Science and Impact of Blockchain Technology at Oak Ridge National Laboratory. <https://info.ornl.gov/sites/publications/Files/Pub118487.pdf>, Accessed 2019.
- [4] Abdullah Al-Mamun, Jun Dai, Xiaohua Xu, Mohammad Sadoghi, and Haoting Shen Dongfang Zhao. Poster: Dean: A blockchain-inspired consensus protocol enabling trustworthy edge computing. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.
- [5] Abdullah Al-Mamun, Tonglin Li, Mohammad Sadoghi, Linhua Jiang, Haoting Shen, and Dongfang Zhao. Hpchain: An mpi-based blockchain framework for data fidelity in high-performance computing systems. In *Supercomputing*, 2019.
- [6] Abdullah Al-Mamun, Tonglin Li, Mohammad Sadoghi, Linhua Jiang, Haoting Shen, and Dongfang Zhao. Poster: An mpi-based blockchain framework for data fidelity in high-performance computing systems. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2019.
- [7] Abdullah Al-Mamun, Tonglin Li, Mohammad Sadoghi, and Dongfang Zhao. In-memory blockchain: Toward efficient and trustworthy data provenance for hpc systems. In *IEEE International Conference on Big Data (BigData)*, 2018.
- [8] Abdullah Al-Mamun, Feng Yan, and Dongfang Zhao. SciChain: Blockchain-enabled lightweight and efficient data provenance for reproducible scientific computing. In *IEEE 37th International Conference on Data Engineering (ICDE)*, 2021.

- [9] Benjamin S Allen, Matthew A Ezell, Paul Peltz, Doug Jacobsen, Eric Roman, Cory Lueninghoener, and J Lowell Wofford. Modernizing the hpc system software stack. *arXiv preprint arXiv:2007.10290*, 2020.
- [10] Sajede Aminzadegan, Mohammad Tamannaee, and Morteza Rasti-Barzoki. Multi-agent supply chain scheduling problem by considering resource allocation and transportation. *Computers & Industrial Engineering*, 137:106003, 2019.
- [11] L. Aniello, R. Baldoni, E. Gaetani, F. Lombardi, A. Margheri, and V. Sassone. A prototype evaluation of a tamper-resistant high performance blockchain-based transaction log for a distributed database. In *13th European Dependable Computing Conference (EDCC)*, 2017.
- [12] Leonardo Aniello, Basel Halak, Peter Chai, Riddhi Dhall, Mircea Mihalea, and Adrian Wilczynski. Anti-bluff: towards counterfeit mitigation in ic supply chains using blockchain and puf. *International Journal of Information Security*, 20(3):445–460, 2021.
- [13] Leonardo Aniello, Basel Halak, Peter Chai, Riddhi Dhall, Mircea Mihalea, and Adrian Wilczynski. Anti-bluff: towards counterfeit mitigation in ic supply chains using blockchain and puf. *International Journal of Information Security*, 20(3):445–460, 2021.
- [14] Leonardo Aniello, Basel Halak, Peter Chai, Riddhi Dhall, Mircea Mihalea, and Adrian Wilczynski. Securing hardware supply chain using puf. In *Authentication of Embedded Devices*, pages 115–144. Springer, 2021.
- [15] Matthew Areno. Supply chain threats against integrated circuits, 2020.
- [16] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.
- [17] Yonghong Bai, Liji Wu, Xingjun Wu, Xiangyu Li, Xiangmin Zhang, and Beibei Wang. Puf-based encryption method for ic cards on-chip memories. *Electronics Letters*, 52(20):1671–1673, 2016.
- [18] Arnab Banerjee. Blockchain technology: supply chain insights from erp. In *Advances in computers*, volume 111, pages 69–98. Elsevier, 2018.
- [19] Barabasi-Albert Model, Network Science. <http://networksciencebook.com/>, Accessed 2022.

- [20] Adam Bates, Dave Jing Tian, Kevin RB Butler, and Thomas Moyer. Trustworthy whole-system provenance for the linux kernel. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 319–334, 2015.
- [21] BigchainDB. <https://github.com/bigchaindb/bigchaindb>, Accessed 2018.
- [22] Bitcoin. <https://bitcoin.org/bitcoin.pdf>, Accessed 2022.
- [23] Bitcoin Scale. <https://bitnodes.earn.com>, Accessed 2019.
- [24] Darius Buntinas. Scalable distributed consensus to support mpi fault tolerance. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pages 1240–1249. IEEE, 2012.
- [25] Anton Burtsev, David Johnson, Josh Kunz, Eric Eide, and Jacobus Van der Merwe. Capnet: security and least authority in a capability-enabled cloud. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 128–141, 2017.
- [26] Yuan Cao, Le Zhang, Chip-Hong Chang, and Shoushun Chen. A low-power hybrid ro puf with improved thermal stability for lightweight applications. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 34(7):1143–1147, 2015.
- [27] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, November 2002.
- [28] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.
- [29] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. *Advances in neural information processing systems*, 32:4868–4879, 2019.
- [30] Lu Chao, Chundian Li, Fan Liang, Xiaoyi Lu, and Zhiwei Xu. Accelerating apache hive with mpi for data warehouse systems. In *2015 IEEE 35th International Conference on Distributed Computing Systems*, pages 664–673. IEEE, 2015.
- [31] J. Chen, S. Yao, Q. Yuan, K. He, S. Ji, and R. Du. Certchain: Public and efficient certificate audit based on blockchain for tls connections. In *IEEE INFOCOM*, 2018.
- [32] Jacqueline H Chen, Alok Choudhary, Bronis De Supinski, Matthew DeVries, Evatt R Hawkes, Scott Klasky, Wei-Keng Liao, Kwan-Liu Ma, John Mellor-

- Crummey, Norbert Podhorszki, et al. Terascale direct numerical simulations of turbulent combustion using s3d. *Computational Science & Discovery*, 2(1):015001, 2009.
- [33] Si Chen, Rui Shi, Zhuangyu Ren, Jiaqi Yan, Yani Shi, and Jinyu Zhang. A blockchain-based supply chain quality management framework. In *2017 IEEE 14th International Conference on e-Business Engineering (ICEBE)*, pages 172–176. IEEE, 2017.
- [34] Sudheer Chunduri, Scott Parker, Pavan Balaji, Kevin Harms, and Kalyan Kumaran. Characterization of mpi usage on a production supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC '18*, pages 30:1–30:15. IEEE Press, 2018.
- [35] Tim Clark, Paolo N. Ciccarese, and Carole A. Goble. Micropublications: a semantic model for claims, evidence, arguments and annotations in biomedical communications. *Journal of Biomedical Semantics*, 5(1):28, Jul 2014.
- [36] Pinchen Cui, Julie Dixon, Ujjwal Guin, and Daniel Dimase. A blockchain-based framework for supply chain provenance. *IEEE Access*, 7:157113–157125, 2019.
- [37] Cybersecurity and Infrastructure Security Agency. Understanding Denial-of-Service Attacks, 2019.
- [38] D. Dai, Y. Chen, P. Carns, J. Jenkins, and R. Ross. Lightweight provenance service for high-performance computing. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017.
- [39] Hao Dai, H Patrick Young, Thomas JS Durant, Guannan Gong, Mingming Kang, Harlan M Krumholz, Wade L Schulz, and Lixin Jiang. Trialchain: A blockchain-based platform to validate data integrity in large, biomedical research studies. *arXiv preprint arXiv:1807.03662*, 2018.
- [40] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. Towards scaling blockchain systems via sharding. In *ACM International Conference on Management of Data (SIGMOD)*, 2019.
- [41] Data fraud in clinical trials. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4340084/>, Accessed 2019.
- [42] Susan B Davidson and Sudeepa Roy. Provenance: Privacy and security. 2017.

- [43] Srinivas Devadas, Edward Suh, Sid Paral, Richard Sowell, Tom Ziola, and Vivek Khandelwal. Design and implementation of puf-based” unclonable” rfid ics for anti-counterfeiting and security applications. In *2008 IEEE international conference on RFID*, pages 58–64. IEEE, 2008.
- [44] Daniel DiMase, Zachary A Collier, Jinae Carlson, Robin B Gray Jr, and Igor Linkov. Traceability and risk analysis strategies for addressing counterfeit electronics in supply chains for complex systems. *Risk Analysis*, 36(10):1834–1843, 2016.
- [45] Qingyang Ding, Sheng Gao, Jianming Zhu, and Chongxuan Yuan. Permissioned blockchain-based double-layer framework for product traceability system. *IEEE Access*, 8:6209–6225, 2019.
- [46] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. Blockbench: A framework for analyzing private blockchains. In *ACM International Conference on Management of Data (SIGMOD)*, 2017.
- [47] Harish Dattatraya Dixit, Sneha Pendharkar, Matt Beadon, Chris Mason, Tejasvi Chakravarthy, Bharath Muthiah, and Sriram Sankar. Silent data corruptions at scale. *CoRR*, abs/2102.11245, 2021.
- [48] Docker. <https://github.com/docker/docker>, Accessed July 16, 2015.
- [49] DOE SBIR. <https://www.sbir.gov/sbirsearch/detail/1307745>, Accessed 2017.
- [50] Sergey N Dorogovtsev, Alexander V Goltsev, and José Ferreira F Mendes. Pseudofractal scale-free web. *Physical review E*, 65(6):066122, 2002.
- [51] Shaohua Duan, Pradeep Subedi, Philip Davis, Keita Teranishi, Hemanth Kolla, Marc Gamell, and Manish Parashar. Corec: Scalable and resilient in-memory data staging for in-situ workflows. *ACM Transactions on Parallel Computing (TOPC)*, 7(2):1–29, 2020.
- [52] Shaohua Duan, Pradeep Subedi, Philip E Davis, and Manish Parashar. Addressing data resiliency for staging based scientific workflows. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–22, 2019.
- [53] Ethereum. <https://www.ethereum.org/>, Accessed 2022.

- [54] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoinng: A scalable blockchain protocol. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pages 45–59, Berkeley, CA, USA, 2016. USENIX Association.
- [55] Fadi Farha, Huansheng Ning, Karim Ali, Liming Chen, and Christopher Nugent. Sram-puf-based entities authentication scheme for resource-constrained iot devices. *IEEE Internet of Things Journal*, 8(7):5904–5913, 2020.
- [56] David Fiala, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira, and Ron Brightwell. Detection and correction of silent data corruption for large-scale high-performance computing. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2012.
- [57] Kristoffer Francisco and David Swanson. The supply chain has no clothes: Technology adoption of blockchain for supply chain transparency. *Logistics*, 2(1):2, 2018.
- [58] Keke Gai, Yulu Wu, Liehuang Zhu, Lei Xu, and Yan Zhang. Permissioned blockchain and edge computing empowered privacy-preserving smart grid networks. *IEEE Internet of Things Journal*, 6(5):7992–8004, 2019.
- [59] Chunpeng Ge, Lu Zhou, Gerhard P Hancke, and Chunhua Su. A provenance-aware distributed trust model for resilient unmanned aerial vehicle networks. *IEEE Internet of Things Journal*, 2020.
- [60] Ashish Gehani and Dawood Tariq. SPADE: Support for Provenance Auditing in Distributed Environments. In *Proceedings of the 13th International Middleware Conference (Middleware)*, 2012.
- [61] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, SOSP '17*, pages 51–68, New York, NY, USA, 2017. ACM.
- [62] GM and BMW Back Blockchain for Self-driving Cars. <https://www.coindesk.com/gm-bmw-back-blockchain-data-sharing-for-self-driving-cars>, Accessed 2019.
- [63] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu. Sbft: A scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019.

- [64] Yanfei Guo, Wesley Bland, Pavan Balaji, and Xiaobo Zhou. Fault tolerant mapreduce-mpi for hpc clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15*, pages 34:1–34:12, 2015.
- [65] Hackers Remotely Kill a Jeep on the Highway. <https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway>, Accessed 2019.
- [66] Alborz Hassanzadeh and Morteza Rasti-Barzoki. Minimizing total resource consumption and total tardiness penalty in a resource allocation supply chain scheduling and vehicle routing problem. *Applied Soft Computing*, 58:307–323, 2017.
- [67] Vikas Hassija, Vinay Chamola, Vatsal Gupta, Sarthak Jain, and Nadra Guizani. A survey on supply chain security: Application areas, security threats, and solution architectures. *IEEE Internet of Things Journal*, 8(8):6222–6246, 2020.
- [68] Brandon Haynes, Amrita Mazumdar, Armin Alaghi, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. Lightdb: A dbms for virtual reality video. *Proc. VLDB Endow.*, 11(10):1192–1205, June 2018.
- [69] Yaodong Huang, Jiarui Zhang, Jun Duan, Bin Xiao, Fan Ye, and Yuanyuan Yang. Resource allocation and consensus on edge blockchain in pervasive edge computing environments. In *IEEE ICDCS 2019*, pages 1476–1486, 2019.
- [70] Hyperledger. <https://www.hyperledger.org/>, Accessed 2022.
- [71] Inkchain. <https://github.com/inklabsfoundation/inkchain>, Accessed 2018.
- [72] Institute for Defense Analyses. Supply chain risk in leading-edge integrated circuits, 2021.
- [73] Inter blockchain. <https://ibcprotocol.org/>, Accessed 2022.
- [74] Md Nazmul Islam and Sandip Kundu. Enabling IC traceability via blockchain pegged to embedded PUF. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 24(3):1–23, 2019.
- [75] Md Nazmul Islam and Sandip Kundu. Enabling ic traceability via blockchain pegged to embedded puf. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 24(3):1–23, 2019.

- [76] Md Nazmul Islam, Vinay C Patii, and Sandip Kundu. On ic traceability via blockchain. In *2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–4, 2018.
- [77] Uzair Javaid, Muhammad Naveed Aman, and Biplab Sikdar. Drivman: Driving trust management and data sharing in vanets with blockchain and smart contracts. In *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pages 1–5. IEEE, 2019.
- [78] Sergii Kalytchuk, Yu Wang, Kateřina Poláková, and Radek Zbořil. Carbon dot fluorescence-lifetime-encoded anti-counterfeiting. *ACS applied materials & interfaces*, 10(35):29902–29908, 2018.
- [79] Bohdan Karpinskyy, Yongki Lee, Yunhyeok Choi, Yongsoo Kim, Mijung Noh, and Sanghyun Lee. 8.7 physically unclonable function for secure key generation with a key error rate of $2e-38$ in 45nm smart-card chips. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 158–160. IEEE, 2016.
- [80] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology (CRYPTO)*, 2017.
- [81] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, 2016.
- [82] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.
- [83] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.
- [84] Kari Korpela, Jukka Hallikas, and Tomi Dahlberg. Digital supply chain transformation toward blockchain integration. In *proceedings of the 50th Hawaii international conference on system sciences*, 2017.
- [85] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain

- model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.
- [86] Cyrill Kuemin, Lea Nowack, Luisa Bozano, Nicholas D Spencer, and Heiko Wolf. Oriented assembly of gold nanorods on the single-particle level. *Advanced Functional Materials*, 22(4):702–708, 2012.
- [87] Randhir Kumar and Rakesh Tripathi. Traceability of counterfeit medicine supply chain through blockchain. In "*2019 11th International Conference on Communication Systems & Networks (COMSNETS)*", pages 568–570. IEEE, 2019.
- [88] Leslie Lamport. Paxos made simple, fast, and byzantine. In *OPODIS 2002, Reims, France, December 11-13, 2002*, pages 7–9, 2002.
- [89] Laphou Lao, Xiaohai Dai, Bin Xiao, and Songtao Guo. G-pbft: a location-based and scalable consensus protocol for iot-blockchain applications. In *2020 IEEE IPDPS*, pages 664–673. IEEE, 2020.
- [90] S. Lee, S. Kohler, B. Ludascher, and B. Glavic. A sql-middleware unifying why and why-not provenance for first-order queries. In *IEEE International Conference on Data Engineering (ICDE)*, 2017.
- [91] Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-an, and Heng Ye. Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225, 2018.
- [92] Mingzhe Li, Khaled Hamidouche, Xiaoyi Lu, Hari Subramoni, Jie Zhang, and Dhaleswar K Panda. Designing mpi library with on-demand paging (odp) of infiniband: challenges and benefits. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 433–443. IEEE, 2016.
- [93] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017.
- [94] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017.

- [95] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla. Prochain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017.
- [96] Dongxiao Liu, Jianbing Ni, Cheng Huang, Xiaodong Lin, and Xuemin Sherman Shen. Secure and efficient distributed network provenance for iot: A blockchain-based approach. *IEEE Internet of Things Journal*, 7(8):7564–7574, 2020.
- [97] Xiaoyi Lu, Fan Liang, Bing Wang, Li Zha, and Zhiwei Xu. Datampi: extending mpi to hadoop-like big data computing. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 829–838. IEEE, 2014.
- [98] Sujaya Maiyya, Victor Zakhary, Divyakant Agrawal, and Amr El Abbadi. Database and distributed computing fundamentals for scalable, fault-tolerant, and consistent maintenance of blockchains. *Proceedings of the VLDB Endowment*, 11(12):2098–2101, 2018.
- [99] Abdullah Al Mamun, Feng Yan, and Dongfang Zhao. Baash: lightweight, efficient, and reliable blockchain-as-a-service for hpc systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–18, 2021.
- [100] David R. Mathog. Parallel BLAST on split databases. *Bioinformatics*, 19(4):1865 – 1866, 2003.
- [101] Bruno Medeiros, Marcos A Simplicio Jr, and Ewerton R Andrade. Multi-tenant isolation of what? building a secure tenant isolation architecture for cloud networks. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 518–518, 2018.
- [102] Parmita Mehta, Sven Dorckenwald, Dongfang Zhao, Tomer Kaftan, Alvin Cheung, Magdalena Balazinska, Ariel Rokem, Andrew Connolly, Jacob Vanderplas, and Yusra AlSayyad. Comparative evaluation of big-data systems on scientific image analytics workloads. In *Proceedings of the 43rd International Conference on Very Large Data Bases (VLDB)*, 2017.
- [103] Nathan Menhorn. "external secure storage using the puf", 2021.
- [104] Marion Mille, Jean-François Lamère, Fernanda Rodrigues, and Suzanne Fery-Forgues. Spontaneous formation of fluorescent nanofibers from self-assembly of low-molecular-weight coumarin derivatives in water. *Langmuir*, 24(6):2671–2679, 2008.

- [105] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42. ACM, 2016.
- [106] Muhammad Baqer Mollah, Jun Zhao, Dusit Niyato, Kwok-Yan Lam, Xin Zhang, Amer MYM Ghias, Leong Hai Koh, and Lei Yang. Blockchain for future smart grid: A comprehensive survey. *IEEE Internet of Things Journal*, 8(1):18–43, 2020.
- [107] MPI4PY. <https://mpi4py.readthedocs.io/en/stable/intro.html>, Accessed 2022.
- [108] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.
- [109] NSF CICI. <https://www.nsf.gov/pubs/2018/nsf18547/nsf18547.htm>, Accessed 2018.
- [110] Svein Ølnes. Beyond bitcoin enabling smart government using blockchain technology. In *International Conference on Electronic Government*, pages 253–264. Springer, 2016.
- [111] Parity. <https://ethcore.io/parity.html/>, Accessed 2018.
- [112] So-Yeon Park, Sunil Lim, Dahee Jeong, Jungjin Lee, Joon-Sung Yang, and HyungJune Lee. Pufsec: Device fingerprint-based security architecture for internet of things. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.
- [113] EF Pettersen, TD Goddard, CC Huang, GS Couch, DM Greenblatt, EC Meng, and TE Ferrin. Ucsf chimera—a visualization system for exploratory research and analysis. *Journal of Computational Chemistry*, 25(13):1605–1612, Oct 2004.
- [114] Hung H Pham, Ilya Gourevich, JUNG KWON Oh, James EN Jonkman, and Eugenia Kumacheva. A multidye nanostructured material for optical data storage and security data encryption. *Advanced Materials*, 16(6):516–520, 2004.
- [115] Serguei Popov. The tangle. *White paper*, 1(3), 2018.
- [116] Aravind Ramachandran and Murat Kantarcioglu. Smartprovenance: A distributed, blockchain based data provenance system. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY '18*, pages 35–42, 2018.

- [117] Aravind Ramachandran and Murat Kantarcioglu. Smartprovenance: A distributed, blockchain based data provenance system. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY '18*, pages 35–42, 2018.
- [118] Raspberry Pi. <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>, Accessed 2021.
- [119] Report: 97% of mobile malware is on android. This is the easy way you stay safe. shorturl.at/rvxF1, Accessed 2019.
- [120] Pingcheng Ruan, Gang Chen, Tien Tuan Anh Dinh, Qian Lin, Beng Chin Ooi, and Meihui Zhang. Fine-grained, secure and efficient data provenance on blockchain systems. *Proceedings of the VLDB Endowment*, 12(9):975–988, 2019.
- [121] Pingcheng Ruan, Gang Chen, Tien Tuan Anh Dinh, Qian Lin, Beng Chin Ooi, and Meihui Zhang. Fine-grained, secure and efficient data provenance on blockchain systems. *Proceedings of the VLDB Endowment*, 12(9):975–988, 2019.
- [122] Pingcheng Ruan, Tien Tuan Anh Dinh, Qian Lin, Meihui Zhang, Gang Chen, and Beng Chin Ooi. Lineagechain: a fine-grained, secure and efficient data provenance system for blockchains. *The VLDB Journal*, 30(1):3–24, 2021.
- [123] Sara Saberi, Mahtab Kouhizadeh, Joseph Sarkis, and Lejia Shen. Blockchain technology and its relationships to sustainable supply chain management. *International Journal of Production Research*, 57(7):2117–2135, 2019.
- [124] Emmanuelle Saillard, Patrick Carribault, and Denis Barthou. Static/dynamic validation of mpi collective communications in multi-threaded context. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2015*, pages 279–280. ACM, 2015.
- [125] Umair Sarfraz, Sherali Zeadally, and Masoom Alam. Outsourcing iota proof-of-work to volunteer public devices. *Security and Privacy*, 3(2):e98, 2020.
- [126] Frank Schmuck and Roger Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST)*, 2002.
- [127] Semiconductor Industry Association. Detecting and removing counterfeit semiconductors in the u.s. supply chain, 2011.

- [128] SHA-256. <https://en.bitcoin.it/wiki/SHA-256>, Accessed 2018.
- [129] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.
- [130] Chen Shou, Dongfang Zhao, Tanu Malik, and Ioan Raicu. Towards a provenance-aware distributed filesystem. In *TaPP Workshop, USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.
- [131] D. Skeen and M. Stonebraker. A formal model of crash recovery in a distributed system. *IEEE Trans. Softw. Eng.*, 9(3):219–228, May 1983.
- [132] Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, Kurt B Ferreira, Jon Stearley, John Shalf, and Sudhanva Gurumurthi. Memory errors in modern systems: The good, the bad, and the ugly. *ACM SIGARCH Computer Architecture News*, 43(1):297–310, 2015.
- [133] Andrew Stern, Ulbert Botero, Bicky Shakya, Haoting Shen, Domenic Forte, and Mark Tehranipoor. EMforced: EM-based fingerprinting framework for counterfeit detection with demonstration on remarked and cloned ICs. In *2018 IEEE International Test Conference (ITC)*, pages 1–9. IEEE.
- [134] G Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *2007 44th ACM/IEEE Design Automation Conference*, pages 9–14. IEEE, 2007.
- [135] Melanie Swan. *Blockchain: Blueprint for a new economy*. ” O’Reilly Media, Inc.”, 2015.
- [136] Synopsys. ”physically unclonable function (puf) solution for arc em processors”, 2016.
- [137] Haihu Tan, Guo Gong, Shaowen Xie, Ya Song, Changfan Zhang, Na Li, Dong Zhang, Lijian Xu, Jianxiong Xu, and Jie Zheng. Upconversion nanoparticles@ carbon dots@ meso-sio2 sandwiched core–shell nanohybrids with tunable dual-mode luminescence for 3d anti-counterfeiting barcodes. *Langmuir*, 35(35):11503–11511, 2019.
- [138] Mark M Tehranipoor, Kun Yang, Domenic J Forte, Ulbert Botero, and Haoting Shen. Cross-registration for unclonable chipless rfid tags, February 23 2021. US Patent 10,929,741.

- [139] The Cori Supercomputer. <http://www.nersc.gov/users/computational-systems/cori>, Accessed 2018.
- [140] The Importance of Data Set Provenance for Science. <https://eos.org/opinions/the-importance-of-data-set-provenance-for-science>, Accessed 2019.
- [141] The Small-World Phenomenon: An Algorithmic Perspective. <https://www.cs.cornell.edu/home/kleinber/swn.d/swn.html>, Accessed 2021.
- [142] Sean Atsatt Ting Lu, Ryan Kenny. "white paper: Secure device manager for intel stratix 10 devices provides fpga and soc security", 2015.
- [143] Ferucio Laurențiu Țiplea and Cristian Hristea. Puf protected variables: A solution to rfid security and privacy under corruption with temporary state disclosure. *IEEE Transactions on Information Forensics and Security*, 16:999–1013, 2020.
- [144] U.S. Department of Homeland Security. Combating trafficking in counterfeit and pirated goods - report to the president of the united states, 2020.
- [145] Vmware. <https://docs.vmware.com/en/VMware-NSX-T-Data-Center/3.1/installation/GUID-22F87CA8-01A9-4F2E-B7DB-9350CA60EA4E.html>, Accessed 2021.
- [146] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchain with asynchronized consensus zones. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, Boston, MA, 2019. USENIX Association.
- [147] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *NSDI*, pages 95–112, 2019.
- [148] Jiayao Wang, Abdullah Al-Mamun, Tonglin Li, Linhua Jiang, and Dongfang Zhao. Toward performant and energy-efficient queries in three-tier wireless sensor networks. In *ICPP*, 2018.
- [149] Siming Wang, Dongdong Ye, Xumin Huang, Rong Yu, Yongjian Wang, and Yan Zhang. Consortium blockchain for secure resource sharing in vehicular edge computing: A contract-based approach. *IEEE Transactions on Network Science and Engineering*, 2020.
- [150] Weizheng Wang, Chen Qiu, Zhimeng Yin, Gautam Srivastava, Thippa Reddy Gadekallu, Fawaz Alsolami, and Chunhua Su. Blockchain and puf-based

lightweight authentication protocol for wireless medical sensor networks. *IEEE Internet of Things Journal*, 2021.

- [151] Raymond B Weiss, Nicholas J Vogelzang, Bruce A Peterson, Lawrence C Panasci, John T Carpenter, Molly Gavigan, Karen Sartell, Emil Frei, and O Ross McIntyre. A successful system of scientific data audits for clinical trials: a report from the cancer and leukemia group b. *JAMA*, 270(4):459–464, 1993.
- [152] Yinjun Wu, Abdussalam Alawini, Daniel Deutch, Tova Milo, and Susan Davidson. Provcite: provenance-based data citation. *Proceedings of the VLDB Endowment*, 12(7):738–751, 2019.
- [153] Xiaolin Xu, Fahim Rahman, Bicky Shakya, Apostol Vassilev, Domenic Forte, and Mark Tehranipoor. Electronics supply chain integrity enabled by blockchain. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 24(3):1–25, 2019.
- [154] Yijing Xun, Zhijiang Li, Xiaolu Zhong, Sheng Li, Jiawang Su, and Ke Zhang. Dual anti-counterfeiting of qr code based on information encryption and digital watermarking. In *Advances in Graphic Communication, Printing and Packaging*, pages 187–196. Springer, 2019.
- [155] Wei Yan, Ning Zhang, Laurent L Njilla, and Xuan Zhang. PcbChain: Lightweight reconfigurable blockchain primitives for secure IoT applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(10):2196–2209, 2020.
- [156] YCSB. <https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads>, Accessed 2018.
- [157] Hüsnu Yıldız, Murat Cenk, and Ertan Onur. Plgakd: A puf-based lightweight group authentication and key distribution protocol. *IEEE Internet of Things Journal*, 8(7):5682–5696, 2020.
- [158] Yildiran Yilmaz, Steve R Gunn, and Basel Halak. Lightweight puf-based authentication protocol for iot devices. In *2018 IEEE 3rd international verification and security workshop (IVSW)*, pages 38–43. IEEE, 2018.
- [159] Meng-Day Yu and Srinivas Devadas. Secure and robust error correction for physical unclonable functions. *IEEE Design & Test of Computers*, 27(1):48–65, 2010.
- [160] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948. ACM, 2018.

- [161] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.
- [162] H. Zhang and K. Zeng. Pairwise markov chain: A task scheduling strategy for privacy-preserving sift on edge. In *IEEE INFOCOM*, 2019.
- [163] Kaiwen Zhang and Hans-Arno Jacobsen. Towards dependable, scalable, and pervasive distributed ledgers with blockchains. In *38th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2018.
- [164] Yuhai Zhang, Lixin Zhang, Renren Deng, Jing Tian, Yun Zong, Dayong Jin, and Xiaogang Liu. Multicolor barcoding in a single upconversion crystal. *Journal of the American Chemical Society*, 136(13):4893–4896, 2014.
- [165] Zhao Zhang, Daniel S. Katz, Michael Wilde, Justin M. Wozniak, and Ian Foster. Mtc envelope: Defining the capability of large scale computers in the context of parallel scripting applications. In *ACM International Symposium on High-performance Parallel and Distributed Computing (HPDC)*, 2013.
- [166] Dongfang Zhao. Cross-blockchain transactions. In *Conference on Innovative Data Systems Research (CIDR)*, 2020.
- [167] Dongfang Zhao, Ning Liu, Dries Kimpe, Robert Ross, Xian-He Sun, and Ioan Raicu. Towards exploring data-intensive scientific applications at extreme scales through systems and simulations. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 27(6), 2016.
- [168] Dongfang Zhao, Chen Shou, Tanu Malik, and Ioan Raicu. Distributed data provenance for large-scale data-intensive computing. In *IEEE International Conference on Cluster Computing (CLUSTER)*, 2013.
- [169] H. Zhou, X. Ouyang, Z. Ren, J. Su, C. de Laat, and Z. Zhao. A blockchain based witness model for trustworthy cloud service level agreement enforcement. In *IEEE INFOCOM*, 2019.
- [170] Feng Zhu, Peng Li, He Xu, and Ruchuan Wang. A lightweight rfid mutual authentication protocol with puf. *Sensors*, 19(13):2957, 2019.