Latency-Optimized Distributed Storage for Blockchain in IoT Network

by

Junyuan Leng

B.Eng., Xi’an Jiaotong University, 2013
M.Eng., Xi’an Jiaotong University, 2016

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE
in
THE COLLEGE OF GRADUATE STUDIES
(Electrical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA
(Okanagan)
March 2019
© Junyuan Leng, 2019
The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Latency-Optimized Distributed Storage for Blockchain in IoT Network

submitted by Junyuan Leng in partial fulfillment of the requirements for the degree of Master of Applied Science in Electrical Engineering

Dr. Chen Feng, School of Engineering
Supervisor

Dr. Julian Cheng, School of Engineering
Co-supervisor

Dr. Zheng Liu, School of Engineering
Supervisory Committee Member

Dr. Yang Cao, School of Engineering
Supervisory Committee Member

Dr. Warren Hare, School of Arts and Sciences
University Examiner
Abstract

The architecture of integrating blockchain with Industry 4.0 IoT network has emerged in recent years. Industry 4.0 refers to the new concept of intelligent manufacturing paradigm, including Internet of Things (IoT), Cloud computing and Cyber-Physical System (CPS). Blockchain is a decentralized, distributed ledger system used to record transactions across multiple participants. Blockchain offers a way of recording data that is designed to be secure, reliable and auditable, which makes it a preferred solution for securing IoT network. However, traditional blockchain suffers from storage scalability issue because each blockchain node stores the entire blockchain. The storage scalability problem can be even worse when blockchain is used in IoT network due to the high volume of data generated by massive number of IoT sensors.

In this thesis, we propose a distributed storage architecture for blockchain based on erasure coding. The property of erasure coding makes it possible to greatly reduce storage overhead for each node without affecting the overall data integrity. Furthermore, to minimize the latency of rebuilding data in this distributed storage architecture, we formulate a latency-cost trade-off optimization problem and propose an efficient and scalable algorithm. Simulation results show that our architecture can achieve up to 80% latency reduction compared with other widely used distributed storage architectures. In addition, our architecture can save up to 90% storage space for each node, which removes the largest obstacle for integrating blockchain into IoT network.
Lay Summary

A new intelligent manufacturing paradigm, which is often referred to as Industry 4.0, is emerging in recent years. To meet with the data security and data integrity requirements of Industry 4.0, we propose an architecture integrating Industry 4.0 with a reliable, immutable and secure data recording technology called blockchain. Blockchain is a decentralized, distributed ledger system used to record transactions across multiple participants. Built on the basis of public key encryption, consensus mechanism and full decentralization, blockchain offers strong data privacy protection and data security assurance in a distributed manner. However, simply integrating Industry 4.0 with blockchain is not practical because traditional blockchain suffers from storage scalability issue. To solve the blockchain storage scalability issue, we propose a distributed storage architecture for blockchain which greatly reduces storage space requirement (up to 90% less storage) without affecting the overall data integrity. Furthermore, to minimize the data access latency of our proposed storage architecture, we formulate a latency-cost trade-off optimization problem and propose an efficient and scalable algorithm to solve this optimization problem. Simulation results show that our architecture can achieve up to 80% latency reduction.
Preface

This thesis is based on the research work conducted under the supervision of Dr. Chen Feng and Dr. Julian Cheng in the School of Engineering at The University of British Columbia, Okanagan Campus.
# Table of Contents

Abstract ................................. iii

Lay Summary .............................. iv

Preface .................................. v

Table of Contents ........................ vi

List of Figures ........................... viii

List of Acronyms ........................ ix

List of Symbols ........................... x

Acknowledgements ........................ xii

Chapter 1: Introduction .................. 1

Chapter 2: Related Work .................. 4

Chapter 3: Background ................... 7

  3.1 Blockchain ........................... 7

  3.2 Blockchain Security Mechanism .......... 8

      3.2.1 Immutability .................... 8

      3.2.2 Consensus ....................... 9

      3.2.3 Auditable ....................... 9

  3.3 Blockchain IoT Integration .............. 9

Chapter 4: System Architecture .......... 11

  4.1 Fog Layer Blockchain Integration ........ 11
4.2 IoT Layer Blockchain Architecture ........................................... 13

Chapter 5: System Model ....................................................... 16
  5.1 Placement Subproblem .................................................. 18
  5.2 RS Coding Parameter Subproblem .................................. 18
     5.2.1 Delay Function ................................................. 20
     5.2.2 Cost Function .................................................. 21
     5.2.3 Problem Formulation ......................................... 21

Chapter 6: Optimization Solution ........................................... 23
  6.1 Convex Relaxation ..................................................... 23
  6.2 Single File Case ....................................................... 25
  6.3 Two-File Case .......................................................... 28
     6.3.1 Time Complexity Analysis of Two-File Case ............... 32
  6.4 Multiple-File Case .................................................... 32
     6.4.1 Time Complexity Analysis of Multiple-File Case ....... 32

Chapter 7: Evaluation .......................................................... 34
  7.1 Simulation Parameter Selection ..................................... 34
     7.1.1 Selection of $\mu$ ............................................. 34
     7.1.2 Selection of $\gamma$ ........................................... 35
     7.1.3 Selection of cost coefficient $c$ ............................. 35
  7.2 Simulation Results .................................................... 35

Chapter 8: Conclusion ........................................................ 43

Bibliography ................................................................. 44
List of Figures

Figure 3.1  Blockchain data structure .............................................. 8
Figure 4.1  Integration architecture .................................................. 11
Figure 4.2  Fog layer blockchain architecture ..................................... 12
Figure 4.3  IoT layer blockchain verification mechanism ....................... 14
Figure 5.1  System model ................................................................. 17
Figure 5.2  Water-filling algorithm .................................................... 19
Figure 6.1  Upper and lower bounds tightness .................................... 24
Figure 6.2  Plot of $f'(x)$ when $k = 90, l = 10, \gamma = \mu = 1$ ............ 26
Figure 6.3  Plot of $f'(x)$ when $c = 0, k = 100, m = 400, \gamma = \mu = 1$ .... 27
Figure 6.4  Plot of $f'(x)$ when $c = 0.004, k = 100, m = 400, \gamma = \mu = 1$ ... 27
Figure 6.5  Plot of $f'(x)$ when $c = 0.2, k = 100, m = 400, \gamma = \mu = 1$ ... 28
Figure 6.6  Plot of $x^*_1 = k_1 + l_1, x^*_2 = m - x^*_1$ ....................... 29
Figure 6.7  Plot of $x^*_2 = k_2 + l_2, x^*_1 = m - x^*_2$ ....................... 30
Figure 6.8  Plot of critical condition when $x^*_1 + x^*_2 = m$ .................. 30
Figure 7.1  System trade-off when $s + t = 0.1$ ................................. 36
Figure 7.2  System trade-off when $s + t = 0.5$ ................................. 37
Figure 7.3  System trade-off when $s + t = 1.0$ ................................. 38
Figure 7.4  System trade-off when $s + t = 0.1, s + t = 0.5, s + t = 1.0$ .... 39
Figure 7.5  Simulated delay advantage vs. cost function when $\lambda = 0.1, \lambda = 0.2$ .... 40
Figure 7.6  Simulated storage consumption comparison ........................ 41
Figure 7.7  Storage redundancy comparison .................................... 42
# List of Acronyms

<table>
<thead>
<tr>
<th>Acronyms</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>SDN</td>
<td>Software-Defined Network</td>
</tr>
<tr>
<td>MTTF</td>
<td>Mean-Time-To-Failure</td>
</tr>
<tr>
<td>QBD</td>
<td>Quasi-Birth-Death</td>
</tr>
<tr>
<td>PoW</td>
<td>Proof-of-Work</td>
</tr>
<tr>
<td>PoS</td>
<td>Proof-of-Stake</td>
</tr>
<tr>
<td>PBFT</td>
<td>Practical Byzantine Fault Tolerance</td>
</tr>
<tr>
<td>RS</td>
<td>Reed Solomon</td>
</tr>
</tbody>
</table>
## List of Symbols

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_i$</td>
<td>Raw blockchain data file chunks</td>
</tr>
<tr>
<td>$k_i$</td>
<td>Number of splitted original file chunks</td>
</tr>
<tr>
<td>$n_i$</td>
<td>Number of encoded file chunks</td>
</tr>
<tr>
<td>$N$</td>
<td>Total number of encoded chunks in the system</td>
</tr>
<tr>
<td>$\ell$</td>
<td>Storage capacity of single node</td>
</tr>
<tr>
<td>$m$</td>
<td>Number of nodes in the system</td>
</tr>
<tr>
<td>$T_i$</td>
<td>Trunk request processing time of node $i$</td>
</tr>
<tr>
<td>$\mu$</td>
<td>Parameter of the exponential distribution of $T_i$</td>
</tr>
<tr>
<td>$D$</td>
<td>Delay of file retrieve request</td>
</tr>
<tr>
<td>$E[D]$</td>
<td>Expectation of file retrieve request $D$</td>
</tr>
<tr>
<td>$SD[D]$</td>
<td>Standard deviation of file retrieve request $D$</td>
</tr>
<tr>
<td>$g(n;k)$</td>
<td>Delay function</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Expectation guarantee level of file retrieve request</td>
</tr>
<tr>
<td>$s$</td>
<td>Unit storage cost</td>
</tr>
<tr>
<td>$t$</td>
<td>Unit bandwidth cost</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Tradeoff parameter between delay and cost</td>
</tr>
<tr>
<td>$c$</td>
<td>Cost coefficient, $c = \lambda(s + t)$</td>
</tr>
<tr>
<td>$l_i$</td>
<td>Fault tolerance level of file $i$</td>
</tr>
</tbody>
</table>
## List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(n;k)$</td>
<td>Upper bound of $g(n;k)$</td>
</tr>
<tr>
<td>$h(n;k)$</td>
<td>Lower bound of $g(n;k)$</td>
</tr>
<tr>
<td>$c_u$</td>
<td>Upper bound of binary search</td>
</tr>
<tr>
<td>$c_l$</td>
<td>Lower bound of binary search</td>
</tr>
<tr>
<td>$\tilde{x}$</td>
<td>Candidate optimal solution of unconstrained optimization problem</td>
</tr>
<tr>
<td>$x^*$</td>
<td>Final optimal solution</td>
</tr>
<tr>
<td>$c'$</td>
<td>Critical cost</td>
</tr>
<tr>
<td>$\hat{x}$</td>
<td>Candidate optimal solution of critical cost</td>
</tr>
</tbody>
</table>
Acknowledgements

After two years of studying at UBC Okanagan, finally it comes to the day when I have finished all my courses and my thesis. The feeling is complicated. It is a mixture of excitement and calmness instead of just feeling happy. On this special day, there are many of people that I must give thanks to.

First of all, I give my greatest appreciation to my supervisor Dr. Chen Feng and co-supervisor Dr. Julian Cheng. I was given the valuable chance to study at UBC Okanagan by the selfless welcome of Dr. Chen Feng and Dr. Julian Cheng. Not only their caring of my life but also their broad knowledge about academic research impressed and helped a lot. Without their patient instruction, I could hardly imagine that I can successfully finish my degree in such a short time.

Second, I would like to thank my family. Thank my parents for providing financial and emotional support for my studying abroad. In my mind, I have the best parents in the world. I would also like to thank my girlfriend Huizhe Chen for her encouragement and companionship when I feel depressed. When I was struggling to get through hard times, her words and smiles always made me feel full of energy and spirit.

Third, I would like to thank all my friends who has helped me during my days spent at Kelowna: Chunpu Wang, Renming Qi, Junxiong He, Wanbing Xu, Dahai Wang, Weisheng Du, Yonghui Lv, Jianyu Niu, Yuwei Guo, Pengxia Wu, Xueting Yang and Fangyu Gai. My friends are always ready to offer help and getting together with them are the happiest time I’ve spent.

Last but not least, thanks for the gorgeous view of Kelowna and Okanagan lake, both day and night. Just sitting beside the lake shore or at the top of Knox mountain help me get out of depression, many times. Keep calm and fight back. That is what I think the mountain and water here try to teach me.
Chapter 1

Introduction

As an awakening technology, Internet of Things [1] (IoT) is considered revolutionary not only because it enables unlimited connectivity [2] for almost everything, but also because of the exciting prospects it indicates when combined with other emerging technologies including Big Data, Fog computing [3], Artificial Intelligence (AI), and Industry 4.0.

The massive application of IoT in Industry 4.0 [4] brings enormous changes to what is believed to be the next industrial development stage. By providing network connectivity to smart sensors, micro controllers and other intelligent devices, IoT fills the gap between physical space and cyber space [5]. The integration of IoT and Industry 4.0 greatly reshapes the way that Industry 4.0 generates, collects and processes data. For example, real-time data from temperature sensors and humidity sensors make it possible for Industry 4.0 monitoring system to instantly detect anomalies during the whole process of manufacturing; pipeline control data and product inspection data bring convenience to quality assurance and quality control in Industry 4.0, which helps increase the product quality; aggregation and analysis of historical manufacturing data build a solid foundation for business decision making and business model update, which helps to realize the smart manufacturing in Industry 4.0 [6].

Data, as we can conclude from the above scenarios, has become the main focus and core competence of IoT and Industry 4.0. As almost all data from IoT and Industry 4.0 is highly confidential and sensitive, data security has become an extremely important topic. When it comes to data security, the massive scale and distributed nature of Industry 4.0 IoT become nightmares for researchers. A large number of solutions has been proposed to solve the seemingly impossible task of keeping data secure in a distributed Industry 4.0 IoT network full of vulnerable devices and potential security threats, of which the most recent one is integrating blockchain [7] technology into the current Industry 4.0 IoT architecture.

Blockchain [8] is a decentralized, distributed ledger system used to record transactions across multiple participants [9]. Built on the basis of public key encryption [10], consensus mechanism
Chapter 1. Introduction

[11] and full decentralization, blockchain offers strong data privacy protection and data security assurance in a distributed manner [12]. Blockchain brings a way of recording data that is designed to be secure, reliable and auditable, making it a perfect solution for securing IoT and Industry 4.0 [13]. However, there is still a huge obstacle for integrating blockchain into IoT and Industry 4.0: storage scalability.

Traditional blockchain architecture requires each participating node to store its own copy of the entire blockchain data. Over time, as transactions are appended to the tail of blockchain, the entire blockchain grows longer and consumes more storage space, causing the storage scalability issue [14]. The storage scalability issue may not be quite serious for traditional blockchain applications like Bitcoin [15] and Ethereum [16] because their participants have sufficient storage space (e.g. computer hard-drive). But for IoT and Industry 4.0, it is a totally different story.

As intelligent devices like smart sensors and micro controllers in IoT and Industry 4.0 have rather limited storage capacity, the storage scalability issue of blockchain can be quite severe. Massive data generated by IoT and Industry 4.0 can instantly occupy all storage space and paralyse the whole network. The integration of blockchain with IoT and Industry 4.0 will not be possible until we fix the storage scalability issue.

In this thesis, we propose a distributed storage architecture for blockchain based on erasure coding [17]. The property of erasure coding makes it possible to greatly reduce storage consumption for each node without affecting the overall data integrity. Furthermore, to minimize the latency of rebuilding data in this distributed storage architecture, we formulate a latency-cost trade-off optimization problem and propose an efficient and scalable algorithm. Simulation results show that the proposed architecture can achieve up to 80% latency reduction compared with other widely used distributed storage architectures. In addition, our architecture can save up to 90% storage space for each node, which removes the largest obstacle for integrating blockchain into IoT network. In summary, we make the following contributions.

− We propose a distributed storage architecture for blockchain and IoT integration based on erasure coding.

− We formulate an optimization problem to find the optimal erasure coding parameters when considering latency-cost trade-off.

− We find an efficient and scalable algorithm for our optimization problem and examine the effectiveness of our algorithm through simulation experiments.
Chapter 1. Introduction

- We build a prototype testbed based on our architecture and algorithm.

This thesis is organized in the following manner. Chapter 2 briefly discusses some related research work. Chapter 3 presents background information about blockchain and blockchain security mechanism. Chapter 4 presents a detailed description of our proposed distributed storage architecture for blockchain and IoT integration. Chapter 5 gives a formal representation of our optimization problem. The proposed optimization solution algorithm is presented in Chapter 6. Based on that algorithm, a series of simulation experiments data and in-depth analysis are given in Chapter 7. Finally, Chapter 8 concludes the thesis.
Chapter 2

Related Work

There have been several previous publications focusing on similar research problems discussed in this thesis, including the integration architecture of blockchain and Industry 4.0 IoT, application of coding theory in distributed storage and optimal parameter selection of erasure coding.

Integration of blockchain and Industry 4.0 IoT is attracting more and more attentions in recent years. Comprehensive surveys were presented in [18], [19] and [20], covering different application scenarios, usage patterns and potential challenges in blockchain and IoT integration. A blockchain-based platform architecture for industrial IoT was proposed in [21]. This architecture first combined previous information sharing platform called Smart-M3 and blockchain, then it used smart contracts to provide trust between participants of IoT production network. Similarly, the application of blockchain’s smart contracts in IoT network was also explored in [22]. This work discussed several ways in which blockchain and IoT can be used together to allow peer-to-peer communications and data exchange in a non-trusting environment. Another novel blockchain-based distributed cloud architecture integrating SDN, Fog computing and IoT was proposed in [23], providing low-cost, secure and on-demand access to IoT network nodes.

Additionally, considering the privacy and security properties of blockchain, using blockchain to provide security service for Industry 4.0 IoT has become the next hotspot. Survey papers like [24], [25] and [26] discussed state-of-the-art IoT security issues and potential blockchain-based solutions. A blockchain-based decentralized security framework for IoT was proposed in [27], giving an overview of using blockchain technology for realization of security across distributed participants in IoT network. More specifically, for IoT data security, [28] implemented blockchain-based protocol and prototype system to provide data integrity service data, eliminating the trust requirements on third party auditors; [29] added a blockchain-based auditable access layer to the IoT storage layer to enable a secure and resilient access control management for IoT data. The topic of IoT device authentication was covered in [30], where secure virtual zones can be built on the basis of blockchain to ensure a robust identification and authentication of IoT devices. By combining blockchain and
private-key encryption, a solution for IoT credibility verification was outlined in [31].

However, all these above papers suffer a common problem: they are more like future visions full of novel concepts and potential architectures, instead of practical systems with in-depth analysis and usable prototypes. On the contrary, our work consists of systematic problem modeling, in-depth theoretical analysis, and extensive simulation experiments.

We are not the first to apply erasure coding in distributed storage systems. As pointed out by comprehensive survey papers like [32], [33] and [34], there has been several efforts on using erasure coding to build a reliable and robust distributed storage system.

By quantitatively compare building a distributed storage system using erasure coding and replication, the authors in [35] showed that erasure coding had orders of magnitude higher mean-time-to-failure (MTTF) than replication, which means that erasure coding has far better system durability. Mean-field analysis between erasure coding and replication was also performed in [36], and the analysis revealed that erasure coding strictly outperformed replication at all traffic loads while improving reliability.

To meet different requirements in distributed storage systems, a series of erasure codes have been explored. Conventional Reed-Solomon code [37] is the most widely used erasure code. Though it achieved optimal storage efficiency, it suffers from high repair bandwidth. For performance considerations, LDPC code [38] was proposed to offer approximate erasure code properties while having a much faster encoding-decoding speed. Regenerating code [39] was proposed to provide reliability and efficient data repair in distributed storage systems with the help of cooperation among participating nodes.

More specifically, the latency metric was also explored from different perspectives in previous researches. One perspective taken by [40], [41] and [42] was to model the request queue for multiple data chunks in a distributed storage system as a multi-dimensional Markov chain, and quasi-birth-death (QBD) process was used to bound the Markov chain. Finally, numerical bounds of latency can be solved through matrix-analytic methods. Another perspective taken by [43] and [44] was to use fork-join models in queuing theory to bound the Markov chain. The subordinate fork-join system was then utilized to derive bounds on the mean latency.

The most significant difference between our work and previous work is the research methodology. Since most previous researchers in this area are from the field of coding theory, they focus on traditional coding theory topics such as deriving tight bounds of availability and achieving optimal
Chapter 2. Related Work

repair bandwidth. Correspondingly, they use traditional coding theory research methodologies such as fork-join model in queuing theory, information flow graph analysis and Markov process. Instead, our solution models the latency-cost trade-off as a joint optimization problem, then uses convex optimization method to solve the problem efficiently.

A number of attempts have been made to deal with the blockchain storage scalability problem. ElasticChain [45] uses its own duplicate-ratio-regulation algorithm to control the portion of the complete blockchain stored on each node. By storing only part of the whole chain on each node, ElasticChain improves the blockchain storage scalability, but its storage efficiency is not as high as erasure coding. Another solution proposed in [46] uses a novel combination of Shamir’s secret sharing scheme, private key encryption and distributed storage codes to construct a coding scheme that distributed blockchain data among subsets of nodes. Though this coding scheme saves storage space, it can lead to loss of data integrity compared with convention erasure code method. Authors of [47] proposed a low storage room requirement framework for distributed ledger in blockchain by simply using erasure coding to store blockchain data. Similar frameworks were also given in [48] and [49]. However, those frameworks lack in-depth discussion about erasure coding parameter selection or performance optimization.
Chapter 3

Background

3.1 Blockchain

As the major supporting technology for record-keeping used by crypto-currencies like Bitcoin and Ethereum, blockchain is attracting more and more attentions in recent years. Before we start finding more potential use-cases for this new technology, the nature of blockchain has to be explored.

As you may tell from its name, blockchain is literally just a “chain” made up of several “blocks”. In this context, a block is a data structure containing digital information (e.g. transactions in Bitcoin, or your order details for Amazon) while a chain can be seen as a database formed by linking all these growing blocks to record the complete history of the whole blockchain system.

Furthermore, blockchain can also be seen as a distributed and decentralized database ledger, that keeps public records in an append-only fashion. For use as a distributed ledger, blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for inter-node communication, called the blockchain network. A blockchain network can be constructed on the backbone of blockchain technology, in which there are multiple blockchain nodes packing digital information to generate new blocks and appending the new blocks to the tail of the chain.

Some people may ask that why we do not just use database instead of blockchain since from the above description blockchain does not seem to have much difference with traditional database technologies. Actually, this is not true. Traditional database technologies can be divided into centralized database and distributed database, blockchain has some advantages over both categories. Compared with centralized database, blockchain does not entirely rely on the normal operation of a central node, thus avoiding the single point of failure in traditional centralized database systems. A distributed blockchain can not be hacked, manipulated, or otherwise disrupted the way a database build on one single operator can be. In addition, a traditional centralized database requires a system directly operated by known and trustworthy individuals (e.g. a known administrator, a managing department). Blockchain, on the other hand, can be operated by unknown nodes and participants
all across the world. Compared with distributed database, blockchain does not require every node in the network to be cooperative. In other words, blockchain is resistant to malicious nodes and data modification. These great security properties are achieved by taking security mechanism into consideration from the first day blockchain is created. Further details about blockchain security mechanism will be given in the following section.

3.2 Blockchain Security Mechanism

3.2.1 Immutability

The most well-known security property of blockchain is data immutability, which means it is impossible for a malicious user to arbitrarily alter historical data. Blockchain network utilizes blockchain data structure to guarantee immutability of all data recorded on blockchain. A typical data structure of blockchain is shown in Fig. 3.1.

Figure 3.1: Blockchain data structure

Blockchain data structure is a linked list indexed by hash values of each single block data. Each block contains some data, a cryptographic hash value of the data, and a cryptographic hash value of its previous block. In Fig. 3.1 there are three different blocks in this blockchain, each storing its own data hash value (marked in red) as well as the hash value of its previous block. If a malicious user tries to alter or tamper some historical data, the corresponding hash value would change, making it inconsistent with hash values stored in its successor node. The immutability makes blockchain...
3.3 Blockchain IoT Integration

difficult to tamper with.

3.2.2 Consensus

However, data immutability only guarantees the recorded data on blockchain can not be arbitrarily altered, which is far from making the blockchain security mechanism flawless. Because any node or participant in blockchain network can submit information to the blockchain, it is necessary for the blockchain network to verify and agree on all addenda before they are permanently recorded on the blockchain. This process of verifying data and achieving agreement is called “consensus”.

Blockchain needs consensus mechanism to guarantee every participating node in the network agrees on state of the system. State can have a variety of different meanings, including real-time sensor data values, financial account balances, and access control and authentication records.

There have been three major categories of consensus algorithms for blockchain: the proof-of-work [50] algorithm (PoW), the proof-of-stake [51] algorithm (PoS), and the practical byzantine fault tolerance [52] algorithm (PBFT). Since consensus algorithm is not the focus of this thesis, we will not cover in-depth details of how these consensus algorithms work.

3.2.3 Auditable

Last but not least, blockchain provides the security property of auditable. In order to perform data verification or maintain data immutability, each record on the blockchain must be accessible by any node or participant of the blockchain network, allowing for any node or participant to individually verify the authenticity of each transaction recorded for any single record in the the blockchain. This transparency means that blockchains are auditable.

3.3 Blockchain IoT Integration

Thanks to all these security enhancement brought by blockchain, blockchain technology has been integrated into massive research and application areas such as crypto-currency [53], digital payment system [54], and asset registration [55]. Of all the above applications, the integration of blockchain and IoT is attracting intense interest based on the following reasons: blockchain technology can be used in tracking billions of connected IoT devices [56], enable a secure processing and storage of IoT data [57]; blockchain’s decentralized design helps eliminate single points of failure, creating a more resilient and reliable infrastructure for IoT devices to operate [58]. Blockchain has been
3.3. Blockchain IoT Integration

mentioned as the missing piece to solve security, privacy, and reliability issues in the IoT area. An integrated architecture of blockchain and IoT proposed in this thesis will be discussed in the following chapters.
Chapter 4

System Architecture

The integration of IoT and blockchain can happen in multiple layers and dimensions. A generic IoT architecture can be separated into three different layers: Cloud layer, Fog layer and IoT device layer. As shown in Fig. 4.1, our proposed architecture integrates blockchain with Fog layer and IoT device layer respectively, providing data security enhancement and distributed data storage service in a hierarchical manner, while maintaining function transparency between different layers. In the remaining part of this chapter, we provide an in-depth description of how the integration of IoT and blockchain is achieved at each layer.

Figure 4.1: Integration architecture

4.1 Fog Layer Blockchain Integration

The integration of blockchain with Fog layer is shown in Fig. 4.2.

In Fig. 4.2, several Fog nodes each controlling a certain stage of the whole production process are connected to form the Fog layer of the complete IoT architecture. Additionally, these Fog
4.1. Fog Layer Blockchain Integration

nodes run blockchain protocols to work as distributed nodes of the Fog level blockchain network. Blockchain serves as a core module of Fog layer, providing security enhancement services including production process audit, sensitive data access control and manufacturing control authentication. These security enhancement services ensure the data generated and collected from supply chain, manufacturing process and real-time monitoring are securely transmitted to and processed by the business intelligence part, achieving the ultimate intelligent manufacturing ecosystem of Industry 4.0.

For production process audit, blockchain can be utilized to store every auditable record in the blockchain data structure by treating these records as “transactions”. Hash algorithm provides immutability property for blockchain, preventing any third-party participant or malicious user from tampering the raw records. Blockchain makes it possible to perform comprehensive audit for historical production process data anytime in the future.

Also, blockchain can be applied in access control and operation authentication when business intelligence department needs some data from other departments. Request information including request source ID, request target ID, request time can be packed into a blockchain transaction and
recorded in the blockchain data structure. Especially, if a certain request requires authorization from higher level administrators, blockchain can also record the authorization operation for future revision. By integrating blockchain technology, access control and operation authentication become more transparent and traceable.

4.2 IoT Layer Blockchain Architecture

Apart from the Fog layer blockchain network composed of Fog nodes, our architecture has another underlying blockchain network formed by all IoT devices at the IoT layer. The purpose of this IoT layer blockchain is to securely store real-time data generated by various IoT devices in a distributed and autonomous manner.

To implement an autonomous blockchain network across massive number of nodes, the most important problem to solve is how to achieve consensus in a closed-form distributed system. This problem is even more important for our IoT layer blockchain because this blockchain network must have the ability to quickly and independently verify massive real-time data generated by large amount of IoT devices and reach agreement on data correctness among different nodes.

Previous research on blockchain data verification has been proposed in [59], in which the blockchain serves as a ledger of hash values of external data stored in an originating system. In other words, the original data and its corresponding hash values are separated. These hash values are anchored into the blockchain, with the blockchain being used as a means of validating the integrity of the original data. A major disadvantage of this solution is that once the external data storage is corrupted or compromised, the original data can not be recovered, which can bring catastrophic consequences.

To solve that, we design a distributed-hash-based verification mechanism for IoT layer blockchain. In our mechanism, not only the hash values but also the original data are distributed stored across all blockchain network nodes. The distributed storage of original data eliminated the weak point of external storage failure, providing a more reliable solution for data verification. A sample diagram of this verification mechanism is shown in Fig. 4.3.

In Fig. 4.3 we use circles numbered 1-6 to stand for different IoT nodes, the blue squares attached to each circle stand for chunks of distributed blockchain data stored on that node, and the red squares attached to each circle stand for hash values of every blockchain block data. As time goes by, the blockchain network packs new transactions into a new block and calculates hash
4.2. IoT Layer Blockchain Architecture

Figure 4.3: IoT layer blockchain verification mechanism
value of the newly generated block. Then blockchain network uses the hash values as indexes to form a linked-list, or a chain structure of blocks, which is called the blockchain data structure.

As opposed to blockchain generation, verification is a reverse process. In our system, verification is divided into two steps. The first step is to collect all chunks stored at other IoT nodes and rebuild the original block data. The second step is to calculate the hash value of the rebuilt block data and compare it with the hash value in the node’s local storage. If the two values are consistent, then we can verify the correctness of all associated transactions packed in that block. Combining this verification mechanism with existing consensus algorithms like PBFT can achieve the final consensus for the whole IoT layer blockchain network.

By integrating blockchain technology with both Fog layer and IoT layer, our proposed architecture provides a data security enhancement and distributed data storage while maintaining function transparency between different layers.
Chapter 5

System Model

In this chapter, we focus on the layer of distributed storage, aiming to improve its reliability and reduce its access delay. We consider an IoT cluster consisting of \( m \) nodes. For simplicity, the system holds the following assumptions:

*Assumption 1:* All \( m \) nodes are homogeneous, each with an identical storage capacity.

*Assumption 2:* Once an IoT node fails, all the data stored on that node will be lost.

*Assumption 3:* The number of nodes \( m \) is sufficiently large to store all files.

The purpose of this distributed storage is to store a series of files \( F_1, F_2, \ldots, F_i, \ldots \) across several selected nodes in a distributed manner, while enabling a fast and reliable recovery of the original file when needed. In this context, the term “file” stands for raw blockchain data generated by the blockchain network. Each file may contain only a single blockchain data, or may contain several blockchain data, which is decided by the system administrator.

In order to distribute a series of files \( F_1, F_2, \ldots, F_i \) to be stored on the IoT nodes, we split each file \( F_i \) into \( k_i \) fixed-size chunks, then use an \((n_i, k_i)\) Reed-Solomon code (RS code) to generate \( n_i \) encoded chunks, including \( n_i - k_i \) parity chunks. According to the property of Reed-Solomon coding, the original data can be recovered by collecting any \( k_i \) encoded chunks \([60]\). The system model is shown in Fig. 5.1

As shown in Fig. 5.1, the working process of our system can be described as the following steps:

- **Step 1:** Our system reads a part of the whole blockchain data file \( F_i \) which is newly generated. This blockchain data is generated by the Fog node, using all data collected and sent to the Fog node by subordinate IoT nodes.

- **Step 2:** Our system splits the read data into \( k_i \) fixed-size file chunks, then uses Reed-Solomon coding algorithm to generate \( n_i \) encoded chunks including \( n_i - k_i \) parity chunks.

- **Step 3:** Our systems schedules the \( n_i \) encoded chunks onto several IoT nodes to store.
Figure 5.1: System model
5.1. Placement Subproblem

- Step 4: When receives blockchain reading request, our system retrieves any $k_i$ different chunks and recover the original blockchain data.

From the working process, we can see there are two key points for our system: the placement of $n_i$ encoded chunks to selected $n_i$ nodes in IoT cluster leads to a placement subproblem, and the choice of adequate Reed-Solomon coding parameter leads to a coding parameter subproblem. We will give in-depth analysis of these two subproblems in Section 5.1 and Section 5.2.

5.1 Placement Subproblem

In this section, we solve the placement subproblem. For simplicity, we assume that each IoT node can only store one encoded chunk. (We will relax this assumption later.) Then, the problem reduces to the following question: How shall we select $n_i$ nodes in the IoT cluster for each file $F_i$? Let $N = \sum_i n_i$ denote the total number of encoded chunks in the system. We randomly select $N$ nodes (out of $m$ nodes) and then partition them into groups of size $n_i$. We can check that this solution is optimal since all $m$ nodes are homogeneous according to Assumption 1.

We then proceed to a general case in which each IoT node can store $\ell$ encoded chunks. We can think of this cluster consisting of $\ell \times m$ “virtual” IoT nodes of unit storage capacity. Now, suppose that $N \leq \ell \times m$ and $\max_i n_i \leq m$ according to Assumption 3. We can apply a “water-filling” strategy so that

- every IoT node stores at most one encoded chunk from each file;
- the maximum number of encoded chunks stored by an IoT node is minimized.

An example of the water-filling strategy is shown in Fig. 5.2.

This strategy has two advantages. First, when an IoT node fails, it will affect at most one encoded chunk for each file. Second, it minimizes the number of affected files under a node failure. Therefore, we will use this strategy to solve the placement subproblem.

5.2 RS Coding Parameter Subproblem

For the RS coding parameter subproblem, our objective is to find the optimal parameter which maximizes the system “utility” and minimize the system “cost”. Here, both “utility” and “cost” stand for abstract concepts composed of several different system metrics.
5.2. RS Coding Parameter Subproblem

Figure 5.2: Water-filling algorithm
5.2. RS Coding Parameter Subproblem

In this thesis, we choose to minimize a weighted sum of retrieval delay and storage/bandwidth cost, as we will explain shortly by formulating the delay and cost functions.

5.2.1 Delay Function

According to the property of RS coding, an \((n,k)\) RS code can encode \(k\) original chunks of a file to generate \(n\) encoded chunks. The original file can be retrieved by collecting any \(k\) out of \(n\) encoded chunks. (Note that we have dropped the subscript \(i\) to simplify the notation.) The file retrieve process is described as follows:

- Step 1: The file retrieve request is divided into \(n\) individual chunk requests to be processed by \(n\) nodes.
- Step 2: Once \(k\) out of \(n\) individual chunk requests are completed and returned, the file retrieve process is completed.

If we denote the chunk request processing time by \(T_i\) for node \(i\), then the delay of file retrieve request is the \(k\)th smallest random variable in the set \(\{T_1, T_2, \ldots, T_n\}\), which is often called the \(k\)th order statistic. For ease of presentation, we make the following assumption about the distribution of \(T_i\).

**Assumption 4:** The chunk request processing time \(T_i\) follows an exponential distribution with mean value \(1/\mu\).

If we denote the delay of file retrieve request (i.e., the \(k\)th order statistic) by \(D\), then the expectation and standard deviation of \(D\) are given by

\[
E[D] = \frac{1}{\mu} \sum_{i=1}^{k} \frac{1}{n - k + i},
\]

\[
SD[D] = \frac{1}{\mu} \sqrt{\sum_{i=1}^{k} \frac{1}{(n - k + i)^2}}.
\]  

(5.1)

These results follow from the properties of order statistics of exponential distributions. We observe from Eq. (5.1) that \(E[D]\) and \(SD[D]\) both decrease as \(n\) grows (for a fixed \(k\)). This suggests that we should use as many nodes as possible (up to the storage constraint) in order to achieve a small delay.
5.2. RS Coding Parameter Subproblem

We are now ready to define the delay function $g(n; k)$:

$$g(n; k) \triangleq \mathbb{E}[D] + \gamma \text{SD}[D] = \frac{1}{\mu} \sum_{i=1}^{k} \frac{1}{n - k + i} + \gamma \frac{1}{\mu} \sqrt{\sum_{i=1}^{k} \frac{1}{(n - k + i)^2}}.$$

Here, $\gamma$ is a parameter inspired by Chebyshev’s inequality\(^1\)

$$\Pr (D \geq \mathbb{E}[D] + \gamma \text{SD}[D]) \leq \frac{1}{\gamma^2}. $$

That is, the event that the delay $D$ is no less than $g(n; k)$ happens with probability at most $\frac{1}{\gamma^2}$. Note that the above inequality applies to arbitrary distribution of $D$. Hence, it is conservative and often serves as a “worst-case” bound. For instance, if $D$ is a Gaussian random variable, then we have

$$\Pr (D \geq \mathbb{E}[D] + 3 \times \text{SD}[D]) \leq 0.3\%$$

which is referred to as the three-sigma rule of thumb.

5.2.2 Cost Function

The cost consists of two parts: storage cost and bandwidth cost.

**Storage Cost:** We assume that storing a single chunk on node $i$ requires a storage cost of $s_i$. According to Assumption 1, all nodes have an identical storage cost denoted by $s$. In particular, if we have $n$ encoded chunks in total, the storage cost will be $s \cdot n$.

**Bandwidth Cost:** We assume that transmitting a single chunk on node $i$ requires a bandwidth cost of $t_i$. According to Assumption 1, all links have an identical bandwidth cost denoted by $t$. In particular, if we have $n$ encoded chunks transmitted, the bandwidth cost will be $t \cdot n$.

5.2.3 Problem Formulation

We are now in a position to formulate the parameter selection subproblem. In particular, we would like to minimize a weighted sum of the delay function $g(n_i; k_i)$ and the cost function $(s + t)n_i$ for each file $i$.

\(^1\)Note that one version of Chebyshev’s inequality states that $\Pr (|D - \mathbb{E}[D]| \geq \gamma \text{SD}[D]) \leq \frac{1}{\gamma^2}$. 

5.2. RS Coding Parameter Subproblem

\[
\begin{aligned}
\min & \quad \sum_i (g(n_i; k_i) + cn_i) \\
\text{s.t.} & \quad \forall i, k_i + l_i \leq n_i \leq m \\
& \quad \sum_i n_i \leq \ell \cdot m
\end{aligned}
\] (5.2)

Here, \( c = \lambda(s + t) \) and \( \lambda \) is a parameter providing a tradeoff between the delay and cost. The parameter \( l_i \) indicates the fault tolerance level of file \( i \) (i.e., the number of node failures file \( i \) can tolerate). Clearly, Problem (5.2) is an integer programming problem, since our control variables \( n_i \) are integer numbers. We will apply convex relaxation to solve for Problem (5.2) in the next chapter.
Chapter 6

Optimization Solution

In this chapter, we present a solution to the optimization problem (5.2) based on convex relaxation.

6.1 Convex Relaxation

To apply convex relaxation, we need to define two functions

\[ f(n; k) = \frac{1}{\mu} \ln \left( \frac{n}{n-k} \right) + \frac{\gamma}{\mu} \sqrt{\frac{1}{n-k} - \frac{1}{n}} \]
\[ h(n; k) = \frac{1}{\mu} \ln \left( \frac{n+1}{n-k+1} \right) + \frac{\gamma}{\mu} \sqrt{\frac{1}{n-k+1} - \frac{1}{n+1}}. \]

The functions \( f(n; k) \) and \( h(n; k) \) have several properties. First, \( f(n; k) \) and \( h(n; k) \) are both decreasing functions of \( n \) (for any fixed \( k \)). Second, \( f(n+1; k) = h(n; k) \). Third, \( f(n; k) \geq g(n; k) \geq h(n; k) \) due to the following inequalities

\[ \ln \left( \frac{n+1}{n-k+1} \right) \leq \sum_{i=1}^{k} \frac{1}{n-k+i} \leq \ln \left( \frac{n}{n-k} \right) \]
\[ \frac{1}{n-k+1} - \frac{1}{n+1} \leq \sum_{i=1}^{k} \frac{1}{(n-k+i)^2} \leq \frac{1}{n-k} - \frac{1}{n}. \]

That is, \( f(n; k) \) and \( h(n; k) \) serve as upper and lower bounds of \( g(n; k) \), respectively.

Tightness of upper bound \( f(n; k) \) and lower bound \( h(n; k) \) is shown in Fig. 6.1. In this figure, we set \( k = 100, l = 20, \mu = 100, \gamma = 3 \) which are typical values also used in the evaluation chapter. From Fig. 6.1, we can see the curve of \( f(n; k) \) and the curve of \( h(n; k) \) converge together as \( n \) increases, which means the gap between these two bounds are sufficiently small. So the upper bound \( f(n; k) \) and lower bound \( h(n; k) \) are tight enough.
This suggests the following optimization problem:

\[
\min \sum_i (f(x_i; k_i) + cx_i) \\
\text{s.t.} \forall i, k_i + l_i \leq x_i \leq m \\
\sum_i x_i \leq \ell \cdot m
\] (6.1)

where \( x_i \) are real numbers rather than integer numbers and the function \( f(x; k) \) is given by

\[
f(x; k) = \frac{1}{\mu} \ln \left( \frac{x}{x - k} \right) + \frac{\gamma}{\mu} \sqrt{\frac{1}{x - k} - \frac{1}{x}}.
\]

Intuitively, Problem (6.1) can be viewed as an “approximation” of our original problem (5.2), especially for large \( k_i \) (so that \( f(n_i; k_i) - h(n_i; k_i) \) is small).

It turns out that Problem (6.1) is a convex optimization problem. To see this, it suffices to show that the function \( f(x; k) \) is a convex function with respect to \( x \). For notational convenience,
we write \( f(x) \) as a short-hand notation of \( f(x; k) \). Then, the second derivative of \( f(x) \) is

\[
\frac{d^2}{dx^2} f(x) = \frac{\gamma}{4} \cdot \frac{8k x (x-k) + 3k^3}{x^3(x-k)^3} + \frac{k (2x - k)}{x^2(x-k)^2}. \tag{6.2}
\]

Clearly, \( \frac{d^2}{dx^2} f(x) \geq 0 \) when \( k + l \leq x \leq m \). This is a sufficient condition for \( f(x) \) to be convex.

Although a convex problem can be solved by many existing tools and algorithms, we will develop a fast algorithm by making a special use of the structure of Problem (6.1).

### 6.2 Single File Case

To explain our fast algorithm, we start from the simplest case: There is a single file in the system. In this case, Problem (6.1) reduces to the following convex problem:

\[
\min f(x) + cx \quad \text{s.t.} \quad k + l \leq x \leq m \tag{6.3}
\]

Let \( \tilde{x} \) be the solution to the equation\(^2\)

\[
f'(x) + c = 0. \tag{6.4}
\]

Then, \( \tilde{x} \) is the optimal solution to the unconstrained optimization problem

\[
\min f(x) + cx \tag{6.5}
\]

because \( f(x) \) is a convex function.

If \( \tilde{x} \in [k + l, m] \), then \( \tilde{x} \) is also the optimal solution to Problem (6.3). Otherwise, the optimal solution \( x^* \) is either \( k + l \) or \( m \), depending on whether \( \tilde{x} < k + l \) or \( \tilde{x} > m \).

To provide more intuition, we give an illustration of \( f'(x) \). In this example, we let \( \gamma = \mu = 1 \) for simplicity, and let \( k = 100, l = 20 \), then \( k + l = 120 \).

From Fig. 6.2, we can see \( f'(x) \) is a monotonically increasing function. It has a vertical asymptote at \( x = k \) and a horizontal asymptote. Since \( c \) is a positive value, the plot of \( f'(x) + c \) is the plot of \( f'(x) \) shifted upward by \( c \) units. After the shifting, the plot of \( f'(x) + c \) will have an intersection

\(^2\)The existence and uniqueness of \( \tilde{x} \) can be easily proved.
6.2. Single File Case

Figure 6.2: Plot of $f'(x)$ when $k = 90, l = 10, \gamma = \mu = 1$

point with $x$-axis. At this point, $f'(x) + c = 0$ and the point’s $x$-coordinate is $\tilde{x}$.

As $c$ increases, the intersection point will move towards the negative direction of $x$-axis and $\tilde{x}$
will decrease. The relative position of $\tilde{x}$ with the range $[k + l, m]$ depends on the value of $c$. Next we
will give a case-by-case discussion on how the value of $c$ affects the optimal solution $x^*$ to Problem
(6.3).

Case 1: $c < -f'(m)$

The relation between $\tilde{x}$ and the feasible range $[k + l, m]$ when $c < -f'(m)$ is shown in Fig. 6.3.

When $c < -f'(m)$, the intersection point $(\tilde{x}, 0)$ is to the right of point $(m, 0)$. In this case, we
can not choose $x^* = \tilde{x}$. Instead, we choose $x^* = m$. Intuitively, if a system’s cost is sufficiently
small to be neglected, our best strategy is to spread encoded chunks on all IoT nodes to achieve
the lowest delay. We call this type of system Low-Cost-System.

Case 2: $-f'(m) \leq c \leq -f'(k + l)$

The relation between $\tilde{x}$ and the feasible range $[k + l, m]$ is shown in Fig. 6.4.

When $-f'(m) \leq c \leq -f'(k + l)$, the intersection point $(\tilde{x}, 0)$ lies between point $(k + l, 0)$ and
point $(m, 0)$. As $\tilde{x}$ satisfies the constraint, the optimal solution is $\tilde{x}$, i.e. $x^* = \tilde{x}$. Intuitively, in a
system with medium cost, our best strategy is to find a balance between delay and cost trade-off.
We call this type of system Medium-Cost-System.

Case 3: $c > -f'(k + l)$
6.2. Single File Case

Figure 6.3: Plot of $f'(x)$ when $c = 0, k = 100, m = 400, \gamma = \mu = 1$

Figure 6.4: Plot of $f'(x)$ when $c = 0.004, k = 100, m = 400, \gamma = \mu = 1$
6.3. Two-File Case

The relation between $\bar{x}$ and the feasible range $[k + l, m]$ is shown in Fig. 6.5.

![Plot of $f'(x)$](image)

Figure 6.5: Plot of $f'(x)$ when $c = 0.2, k = 100, m = 400, \gamma = \mu = 1$

In this case, we can not choose $x^* = \bar{x}$. Instead, we choose $x^* = k + l$. Intuitively, if the cost of a system is too high to be affordable, our best strategy is to use the minimum required number of nodes to store our file chunks. We call this type of system *High-Cost-System*.

As shown in Fig. 6.2, the plot of $f'(x)$ has a vertical asymptote at $x = k$. The vertical asymptote at $x = k$ means $f'(k) = -\infty$, i.e. $-f'(k) = +\infty$. It is impossible for $-f'(k)$ to be smaller than $c$.

6.3 Two-File Case

In this section we will extend our discussion to the case of two files. Note that if the storage capacity of each node is at least 2 (i.e., $\ell \geq 2$), then Problem (6.1) can be decoupled into two subproblems of the single-file case. (This is because the constraint $x_1 + x_2 \leq \ell m$ is redundant when $\ell \geq 2$.) Hence, we only need to discuss the case of $\ell = 1$, which is given below.

\[
\begin{align*}
\min & \quad f_1(x_1) + cx_1 + f_2(x_2) + cx_2 \\
\text{s.t.} & \quad x_1 + x_2 \leq m \\
& \quad k_1 + l_1 \leq x_1 \leq m \\
& \quad k_2 + l_2 \leq x_2 \leq m
\end{align*}
\] (6.6)
6.3. Two-File Case

Here, $f_1(x_1)$ and $f_2(x_2)$ are short-hand notations of $f(x_1; k_1)$ and $f(x_2; k_2)$.

Let $\tilde{x}_i$ be the optimal solution to the following convex problem:

$$
\begin{align*}
\min & \quad f_i(x_i) + cx_i \\
\text{s.t.} & \quad k_i + l_i \leq x_i \leq m.
\end{align*}
$$

(6.7)

If $\tilde{x}_1 + \tilde{x}_2 \leq m$, then $(\tilde{x}_1, \tilde{x}_2)$ is the optimal solution to Problem (6.6). Otherwise, the optimal solution $(x_1^*, x_2^*)$ satisfies one of the following conditions:

- $x_1^* = k_1 + l_1$, and $x_2^* = m - x_1^*$;
- $x_2^* = k_2 + l_2$, and $x_1^* = m - x_2^*$;
- $x_1^* + x_2^* = m$, and $f_1'(x_1^*) = f_2'(x_2^*)$.

The first two cases of $x_1^* = k_1 + l_1, x_2^* = m - x_1^*$ and $x_2^* = k_2 + l_2, x_1^* = m - x_2^*$ are shown in Fig. 6.6 and Fig. 6.7 respectively.

![Figure 6.6](image_url)

Figure 6.6: Plot of $x_1^* = k_1 + l_1, x_2^* = m - x_1^*$

The three different conditions motivate us to define a “critical cost” $c'$ as follows. There exists a cost value $c'$, which makes the intersection points with $x$-axis of plot $f_1'(x)$ and plot $f_2'(x)$, denoted by $\hat{x}_1$ and $\hat{x}_2$ respectively, to have a sum exactly equal to $m$, i.e. $\hat{x}_1 + \hat{x}_2 = m$. An example of critical condition is given in Fig. 6.8.
6.3. Two-File Case

Figure 6.7: Plot of $x_2^* = k_2 + l_2, x_1^* = m - x_2^*$

Figure 6.8: Plot of critical condition when $x_1^* + x_2^* = m$
6.3. Two-File Case

The critical cost $c'$, together with the values of $\hat{x}_1$ and $\hat{x}_2$, can be found through a standard binary search.

With $c'$ and $(\hat{x}_1, \hat{x}_2)$, we can use a simple strategy to find the optimal solution $(x_1^*, x_2^*)$, as explained in Algorithm 1.

\textbf{Algorithm 1: Algorithm for the Two-File Case}

\textbf{Input:}
\begin{itemize}
  \item cost function: $c$
  \item node number: $m$
  \item RS coding parameters, $k_1$, $l_1$, $k_2$, $l_2$
  \item two functions: $f_1'(x_1)$, $f_2'(x_2)$
\end{itemize}

\textbf{Output:}
\begin{itemize}
  \item optimal solution $(x_1^*, x_2^*)$
\end{itemize}

1: Solve the convex problem of $\min f_1(x_1) + cx_1$ s.t. $k_1 + l_1 \leq x_1 \leq m$, find the solution $\tilde{x}_1$
2: Solve the convex problem of $\min f_2(x_2) + cx_2$ s.t. $k_2 + l_2 \leq x_2 \leq m$, find the solution $\tilde{x}_2$
3: if $\tilde{x}_1 + \tilde{x}_2 \leq m$ then
4: $x_1^* \leftarrow \tilde{x}_1$
5: $x_2^* \leftarrow \tilde{x}_2$
6: return $x_1^*$, $x_2^*$
7: else
8: if $k_1 + l_1 \leq \tilde{x}_1 \leq m$ and $k_2 + l_2 \leq \tilde{x}_2 \leq m$ then
10: $x_1^* \leftarrow \tilde{x}_1$
11: $x_2^* \leftarrow \tilde{x}_2$
12: return $x_1^*$, $x_2^*$
13: else
14: if $\tilde{x}_1 < k_1 + l_1$ then
15: $x_1^* \leftarrow k_1 + l_1$
16: $x_2^* \leftarrow m - x_1^*$
17: return $x_1^*$, $x_2^*$
18: else
19: if $\tilde{x}_2 < k_2 + l_2$ then
20: $x_2^* \leftarrow k_2 + l_2$
21: $x_1^* \leftarrow m - x_2^*$
22: return $x_1^*$, $x_2^*$
23: end if
24: end if
25: end if
26: end if
6.4. Multiple-File Case

6.3.1 Time Complexity Analysis of Two-File Case

Algorithm 1 consists of three parts: part one is to find the candidate optimal solution \( \hat{x} \), part two is to find \( \hat{x} \) using binary search, part three is to go through several conditional expressions and output the final optimal solution. We only focus on part two (binary search) in the following discussion since it “dominates” the time complexity.

Using standard argument, one can show that the time complexity of the binary search in part two is given below:

\[
O \left( \log_2 \left( \frac{c_u - c_l}{\epsilon} \right) \right)
\]

(6.8)

where \( c_u = \max \{-f'_1(k_1 + l_1), -f'_2(k_1 + l_1)\} \) and \( c_l = c \).

The two parameters \( c_u \) and \( c_l \) are upper and lower bounds for our binary search range. The reason is as follows. If we set \( c_u \) as an upper bound, the corresponding \( x_i < k_i + l_i \) and we have \( x_1 + x_2 < m \). On the other hand, if we set \( c_l \) as a lower bound, the corresponding \( x_1 + x_2 > m \) (because otherwise the algorithm already stops). Hence, we can perform a standard binary search to obtain \( \hat{x}_1 \) and \( \hat{x}_2 \) with \( \hat{x}_1 + \hat{x}_2 = m \). For \( \epsilon \), we can set it to be \( 10^{-4} \) since the final output of \( x_1^* \) and \( x_2^* \) will be rounded.

6.4 Multiple-File Case

As the single file case and two-file case have been covered in previous sections, we can extend our algorithm to the case of multiple files. The algorithm follows a recursive manner. First, we solve for multiple individual convex problems and obtain \( \{\tilde{x}_i\} \). If \( \sum_i \tilde{x}_i \leq \ell m \), we stop and output \( \{\tilde{x}_i\} \). Otherwise, we use a binary search to find \( \{\hat{x}_i\} \). If all of them satisfy \( k_i + l_i \leq \hat{x}_i \leq m \), then we stop and output \( \{\hat{x}_i\} \). Otherwise, we set \( x_i^* = k_i + l_i \) for those files with \( \hat{x}_i < k_i + l_i \). This reduces the scale of the problem and we repeat the above process until we only have one or two files remaining.

6.4.1 Time Complexity Analysis of Multiple-File Case

Time complexity of multiple-file case can be given based on our discussion about the two-file case. Assume we have \( m \) files in total, the time complexity has the following formation:

\[
O \left( m \times \log_2 \left( \frac{c_u - c_l}{\epsilon} \right) \right)
\]

(6.9)
where \( c_u = \max\{-f'_1(k_1 + l_1), -f'_2(k_1 + l_1), \cdots, -f'_m(k_m + l_m)\} \) and \( c_l = c \).

Since we have \( m \) files, under the worst case we must repeat the binary search for \( m \) times. So, the time complexity is multiplied by \( m \). Another difference is the value of \( c_u \). For the multiple-file case, \( c_u \) is the largest among \(-f'_i(k_i + l_i)\).
Chapter 7

Evaluation

In this chapter we present evaluation results of our system.

7.1 Simulation Parameter Selection

7.1.1 Selection of $\mu$

According to the previous assumptions, the chunk request processing time $t_i$ follows an exponential distribution with mean value $\frac{1}{\mu}$, so the physical meaning of parameter $\mu$ is the average chunk request processing rate.

Chunk request processing rate is decided by chunk size and transmission bandwidth. For chunk size, the typical value in our system is 128$KB$. For transmission bandwidth, we investigate the network bandwidth of some widely-used IoT devices and tried to find a reasonable value. A table showing the comparison of network bandwidth between different IoT devices is given below.

<table>
<thead>
<tr>
<th>Device</th>
<th>RPi 3B</th>
<th>RPi 3B+</th>
<th>Onion Omega2</th>
<th>DragonBoard-410c</th>
<th>STM32L4 IoT</th>
<th>ESP32-DevKitC</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFi</td>
<td>802.11n</td>
<td>802.11ac</td>
<td>802.11n</td>
<td>802.11n</td>
<td>802.11n</td>
<td>802.11n</td>
</tr>
<tr>
<td>Ethernet</td>
<td>100Mbps</td>
<td>1000Mbps</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 7.1 shows that almost all IoT devices have 802.11n WiFi connectivity, which has a bandwidth of up to 300Mbps. The smallest bandwidth is the 100Mbps Ethernet of RPi 3B. For fairness, here we choose 100Mbps as the typical value of network bandwidth. The average chunk request processing rate $\mu$ can be calculated by

$$
\mu = \frac{100Mbps}{128KB} = \frac{100 \times 1024Kbps}{128 \times 8Kb} = 100 \text{ request/second}
$$

(7.1)

So $\mu = 100$ is a reasonable choice.
7.2 Simulation Results

7.1.2 Selection of $\gamma$

In the empirical sciences the so-called three-sigma rule of thumb expresses a conventional heuristic that majority of the values are taken to lie within three standard deviations of the mean. Hence, we choose $\gamma = 3$.

7.1.3 Selection of cost coefficient $c$

For cloud-based IoT storage and transmission cost, we take Amazon AWS as an example. Amazon AWS charges 0.023 USD/GB for data storage, and 0.09 USD/GB for data transmission. If IoT devices use local storage (e.g. SD card), a typical price for local storage is 0.16 USD/GB.

7.2 Simulation Results

We first explore the delay-cost trade-off of different $\lambda$ values. In our objective function, changing the value of $\lambda$ changes the system trade-off situation. In Fig. 7.1, Fig. 7.2 and Fig. 7.3 we plot 3 trade-off curves with cost function $s + t = 0.1$, $s + t = 0.5$ and $s + t = 1.0$ respectively. For each curve, we continuously change $\lambda$ from 0 to 1 and calculate the corresponding delay values and cost values. The three curves provide valuable insight on system trade-off decision making process.

Fig. 7.4 illustrates all three curves under the same coordinate, which makes the relationship between different system trade-off more clear.

We next compare the proposed algorithm with some well-known Reed-Solomon codes to explore how much advantage the new algorithm achieves. In this simulation, we choose the $RS(104, 100)$ code which is widely used in RAID systems [61] as the benchmark. For the advantage metric, we choose the delay advantage. The calculation of delay advantage is defined in Eq. (7.2), where $D_b$ stands for delay value of benchmark algorithm and $D_o$ stands of delay value of our algorithm.

$$
\text{delay advantage} = \frac{D_b - D_o}{D_b} \times 100\% \tag{7.2}
$$

Fig. 7.5 shows the simulated delay advantage of our algorithm compared with traditional RS coding algorithm when choosing different cost function $s + t$ values. The blue plot corresponds to $\lambda = 0.1$ while the orange plot corresponds to $\lambda = 0.2$. We can see that our algorithm can achieve up to 80% delay decrease when cost function $s + t < 1$. Our algorithm has delay advantage over traditional RS coding algorithm in the range of $s + t \in [0, 2]$. 

35
Figure 7.1: System trade-off when $s + t = 0.1$
7.2. Simulation Results

Figure 7.2: System trade-off when $s + t = 0.5$
7.2. Simulation Results

Figure 7.3: System trade-off when $s + t = 1.0$
7.2. Simulation Results

Figure 7.4: System trade-off when $s + t = 0.1$, $s + t = 0.5$, $s + t = 1.0$
7.2. Simulation Results

Figure 7.5: Simulated delay advantage vs. cost function when $\lambda = 0.1$, $\lambda = 0.2$
7.2. Simulation Results

Additionally, since the purpose of our proposed RS coding based distributed storage architecture is to minimize the space consumption on each blockchain network node, we compare the storage space consumption of our algorithm with traditional blockchain paradigm. A storage space consumption comparison based on simulation is shown in Fig. 7.6.

Figure 7.6: Simulated storage consumption comparison

Notice that the y-axis of Fig. 7.6 has a logarithmic scale. That is because our RS coding based storage has huge advantages over traditional blockchain storage paradigm. As can be concluded from Fig. 7.6, our architecture can save up to 90% storage space for each node, which removes the largest obstacle for integrating blockchain into IoT network.

Furthermore, as a distributed storage system, we should not only focus on the storage consumption of each individual node, but also on the whole system storage redundancy. There is a balance between single node storage consumption and whole system storage redundancy. We can minimize the single node storage consumption by encoding original data to more chunks, but that
would increase the total storage redundancy. A storage redundancy of the whole system is shown in Fig. 7.7.

![Figure 7.7: Storage redundancy comparison](image)

We can conclude from Fig. 7.7 that the total storage redundancy imported by our architecture is about 1.5 times, which is quite acceptable considering the huge storage space-saving of each individual node.
Chapter 8

Conclusion

In this thesis, we proposed a distributed storage architecture for blockchain based on erasure coding. The property of erasure coding makes it possible to greatly reduce storage overhead for each node without affecting the overall data integrity. Furthermore, to minimize the latency of rebuilding data in this distributed storage architecture, we formulated a latency-cost trade-off optimization problem and proposed an efficient and scalable algorithm. Simulation results show that the proposed architecture can achieve up to 80% latency reduction when compared with other widely used distributed storage architectures. In addition, our architecture can save up to 90% storage space for each node, which removes the largest obstacle for integrating blockchain into IoT network.
Bibliography


[61] Dimitri Pertin, Alexandre Van Kempen, Benoît Parrein, and Nicolas Normand. Comparison of 
   raid-6 erasure codes. In *The third Sino-French Workshop on Information and Communication 