UNLV | University Libraries
University of Nevada, Las Vegas

UNLV Theses, Dissertations, Professional Papers, and Capstones

May 2018

Concurrency in Blockchain Based Smartpool with Transactional Memory

Laxmi Kadariya
laxmi.jhapa@gmail.com

Follow this and additional works at: https://digitalscholarship.unlv.edu/thesesdissertations

Part of the Computer Sciences Commons

Repository Citation

Kadariya, Laxmi, "Concurrency in Blockchain Based Smartpool with Transactional Memory" (2018). *UNLV Theses, Dissertations, Professional Papers, and Capstones*. 3271.
https://digitalscholarship.unlv.edu/thesesdissertations/3271

This Thesis is protected by copyright and/or related rights. It has been brought to you by Digital Scholarship@UNLV with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in UNLV Theses, Dissertations, Professional Papers, and Capstones by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.

CONCURRENCY IN BLOCKCHAIN BASED SMARTPOOL WITH TRANSACTIONAL MEMORY

by

Laxmi Kadariya

Bachelor's Degree in Computer Engineering
Tribhuvan University, Kathmandu, Nepal
2013

A thesis submitted in partial fulfillment of
the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2018

© Laxmi Kadariya, 2018
All Rights Reserved

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

May 4, 2018

This thesis prepared by

Laxmi Kadariya

entitled

Concurrency in Blockchain Based Smartpool with Transactional Memory

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Ajoy K. Datta, Ph.D.
Examination Committee Chair

John Minor, Ph.D.
Examination Committee Member

Laxmi Gewali, Ph.D.
Examination Committee Member

Emma E. Regentova, Ph.D.
Graduate College Faculty Representative

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

Abstract

Blockchain is the buzzword in today's modern technological world. It is an undeniably ingenious invention of the 21st century. Blockchain was first coined and used by a cryptocurrency named Bitcoin. Since then bitcoin and blockchain are so popular that every single person is taking on bitcoin these days and the price of bitcoin has leaped to a staggering price in the last year and so. Today several other cryptocurrencies have adapted the blockchain technology.

Blockchain in cryptocurrencies is formed by chaining of blocks. These blocks are created by the nodes called miners through the process called Proof of Work(PoW). Mining Pools are formed as a collection of miners which collectively tries to solve a puzzle. However, most of the mining pools are centralized.

P2Pool is the first decentralized mining pool in Bitcoin but is not that popular as the number of messages exchanged among the miners is a scalar multiple of the number of shares. SmartPool is a decentralized mining pool with the throughput equal to that of the traditional pool. However the verification of blocks is done in a sequential manner.

We propose a non-blocking concurrency mechanism in a decentralized mining pool for the verification of blocks in a blockchain. Smart contract in SmartPool is concurrently executed using a transactional memory approach without the use of locks. Since the SmartPool mining implemented in ethereum can be applied to Bitcoin, this concurrency method proposed in ethereum smart contracts can be applicable in Bitcoin as well.

Acknowledgements

”I would like to express my sincerest gratitude to my thesis advisor, Dr. Ajoy K. Datta for his tremendous guidance, encouragement, motivation and supervision throughout my work. I cannot express enough thanks to him for his continued support and encouragement even in his difficult period of his life.

I am also grateful to my thesis committee members Dr. Laxmi Gewali, Dr. John Minor and Dr. Emma E. Regentova for reviewing my work and providing valuable comments. I would like to take this opportunity to thank my parents, brothers and sisters. My completion of this thesis could not have been accomplished without their support and love. Finally, I would like to thank my friends, seniors and juniors who made life easier here in Vegas and those back home who always had my back when I needed the motivation to go on. ”

Laxmi Kadariya

University of Nevada, Las Vegas

May 2018

Table of Contents

Abstract iii

Acknowledgements iv

Contents v

List of Tables viii

List of Figures ix

List of Algorithms x

List of Acronyms xi

Chapter 1 Introduction 1

1.1 Centralized Vs Decentralized Systems 1

1.1.1 Centralized System 1

1.1.2 Decentralized System 2

1.2 Concurrent Vs Parallel Computing 3

1.2.1 Concurrency 3

1.2.2 Parallel Computing 4

1.2.3 Necessity of Concurrency 5

1.2.4 Shared Memory And Concurrency 6

1.2.4.1 Atomic Primitives for Shared Memory 6

Chapter 2 Background 9

2.1 Bitcoin. 9

2.1.1 History of Bitcoin. 10

2.1.2 Keys. 11
2.1.3 Address. 12
2.1.4 Wallet 12
2.1.5 Transactions. 12
2.1.5.1 Transaction Inputs and Outputs. 13
2.1.5.2 Transaction Fee. 15
2.1.6 Blockchain. 15
2.1.6.1 Block. 16
2.1.6.2 Structure of Block 17
2.1.6.3 Block Header 17
2.1.6.4 Block Header Hash 17
2.1.6.5 Genesis Block 18
2.1.6.6 Chain of Blocks in Blockchain. 18
2.1.7 Merkle Tree. 19
2.1.8 Mining and Consensus 20
2.1.8.1 Proof of Work (PoW) 22
2.1.8.2 Target difficulty 23
2.1.8.3 Nonce 23
2.1.9 Decentralized Consensus. 23
2.1.9.1 Verification of each Transaction by FullNode. 24
2.1.9.2 Combining Transaction into Blocks. 24
2.1.9.3 Verification of Block and adding to a chain forming a Blockchain . 25
2.1.9.4 Resolving Forks 25
2.2 Pool Mining 26
2.2.1 How Mining Pool Works 27
2.2.2 Distributed Mining Pool 27
2.2.2.1 P2Pool 27
2.3 Ethereum 28
2.3.1 Origin of Ethereum 28
2.3.2 Smart Contract 29
2.4 SmartPool 29
2.4.1 How SmartPool Works. 29

2.4.1.1 Claim Submission 30
2.4.1.2 Batching and Probabilistic Verification 30
2.5 Transactional Memory 33
2.5.1 Types of Transactional Memory 34
2.5.1.1 Non-Blocking Transactional Memory 34

Chapter 3 Literature Review **36**
3.1 Bitcoin and Other Cryptocurrencies 36
3.2 Pooled Mining 37
3.3 Parallel Computing 39
3.4 Transactional Memory 39
3.5 Concurrency to Smart Contract 40

Chapter 4 Proposed Solution **41**
4.1 Concurrency in SmartPool 41
4.2 Overview of Algorithm 41
4.3 Tools and Techniques 46
4.3.1 Atomicity in Counter with CAS 47
4.3.2 Atomicity in Counter with LL/SC 48
4.3.3 STM for Concurrency for Counter 49
4.3.3.1 STM Algorithm with Ownership for Atomic Count 51
4.3.3.2 Overview of STM Algorithm with ownership for atomic count . . . 54
4.4 Correctness 55

Chapter 5 Conclusion and Future work **58**

Bibliography **59**

Curriculum Vitae **62**

List of Tables

List of Figures

1.1 Centralized System . . . 1
1.2 Decentralized System . . . 3
1.3 Concurrent Computing. . . . 4
1.4 Parallel Computing. . . . 5

2.1 Keys in Bitcoin[Nak09]. . . . 11
2.2 Transaction with single input . . . 14
2.3 Transaction with multiple inputs . . . 14
2.4 Blockchain . . . 16
2.5 Structure of Block in Bitcoin. . . . 17
2.6 Structure of Block in Bitcoin. . . . 17
2.7 Chain of block in block chain. . . . 19
2.8 Structure of Merkle Tree . . . 20
2.9 Augmented Tree for a list of Shares . . . 32
2.10 Error in Augmented Merkle Tree due to duplicate share . . . 33

4.1 CAS flowchart . . . 48
4.2 LL/SC flowchart . . . 49
4.3 counter with STM . . . 50

List of Algorithms

1 Compare and Swap(CAS) 7
2 Load-Linked/Store-Conditional(LL/SC). 8
3 Proposed Algorithm for concurrency in SmartPool 45
4 Update Counter using CAS 47
5 Update Counter using LL/SC 49
6 Start Transaction 51
7 Transaction 52
8 Ownership 53
9 Memory Access 54

List of Acronyms

CAS Compare and Swap

COMA Cache only memory Architecture

CPU Central Processing Unit

LL/SC Load-Linked/Store-Conditional

NUMA Non Memory Access

PoA Proof of Activity.

PoW Proof of Work.

ShareAugMT Share Augmented Merkle Tree

STM Software Transaction Memory.

TM Transaction Memory.

UMA Uniform Memory Access.

UTXO Unspent Transaction Output

Chapter 1

Introduction

1.1 Centralized Vs Decentralized Systems

1.1.1 Centralized System

The Currencies that are used today are mostly fiat currency. Fiat currency are fiat money whose value is backed by the government. Examples of fiat currency are US dollar, euro etc. The supply of this currency is managed by the central bank. When two parties need to make any transaction, there always exists a third party which will first verify the transaction and then the transaction is made successful. Figure. 1.1 is an example of a centralized transaction system where any

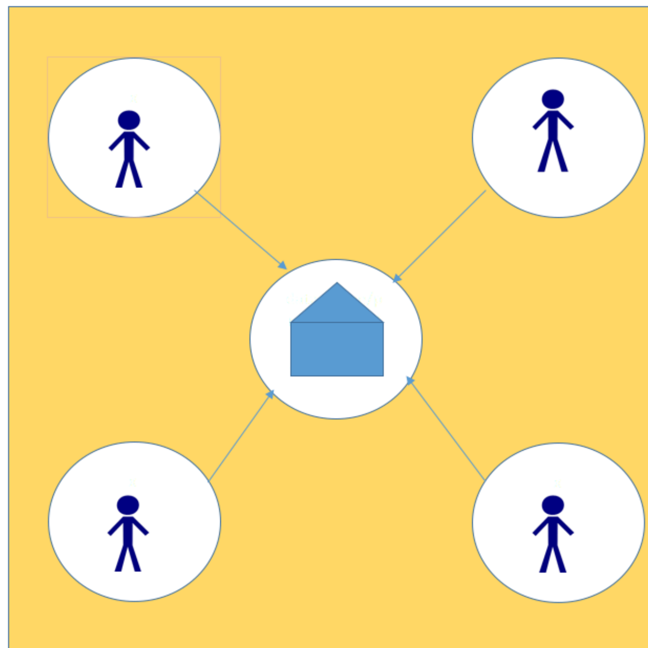

Figure 1.1: Centralized System

transaction needs to pass through the centralized third party. This third party first verifies the transaction whether it is valid or not. Depending upon the validity of the transaction, transactions are either successful or rejected. PayPal, Amazon are some of the online transaction systems that are using centralized transaction systems to verify the transactions. There are several advantages and disadvantages of the centralized transaction system.

- Advantage:
  - Security:
    Centralized systems provides better security as all the transactions are passed through the centralized party. This centralized party verifies the transactions so the security is high in the centralized system
  - Efficient and Easy to build:
    Centralized systems are easy to create and verification of the transactions are more efficient with the third party as a centralized system.
- Disadvantages:
  - Overall Control
    The centralized system has all the data and information stored in the centralized server and this centralized party gets the sole power of controlling the system. When a single party has the sole power of the system, there is a high risk that the centralized party can abuse that power for its own advantages.

1.1.2 Decentralized System

Decentralization is the process of distributing the power away from the sole authority of a single process. As the centralized system has overall control by the single party, this sole authority is taken away by the decentralized system. In the decentralized system as shown in Figure. 1.2, the centralized third party is not required, which eliminates the risk of sole control of power. Moreover, instead of going through the middleman, each party communicates with each other and performs the transaction as required. Elimination of the third party by this system also eliminates the unnecessary fees required to be paid to the third party. With all these advantages explained, the decentralized system is not that easy to build as similar to the centralized system. Several consensus

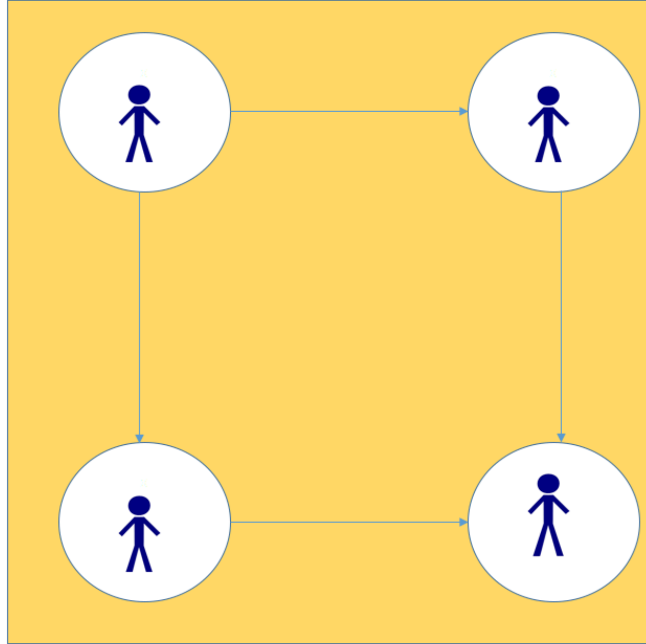

Figure 1.2: Decentralized System

rules and algorithms have to be developed so that there is no difference in operational output as similar to the centralized system.

1.2 Concurrent Vs Parallel Computing

1.2.1 Concurrency

Concurrency is the execution of tasks in interleaved fashion on a single processor or on multiple physical processors. It is the execution of several tasks in an overlapped time period instead of sequential. In this system, one task can advance without waiting for other tasks to complete. Concurrent computing is possible in a single processor, as computing consists of overlapping task execution through time sharing. Only one process runs at a time and it is not necessary that it completes during its time slice.Figure. 1.3 shows an example of how threads are executed in a single CPU for concurrent computing. It is not necessary for the single thread to complete its task as a whole at a single computation but can continue at multiple times with overlapping task execution.

Figure 1.3: Concurrent Computing.

1.2.2 Parallel Computing

Parallelism is the simultaneous execution of tasks on different processors. Execution of tasks occur at a same physical instance in parallel fashion on separate processors of multicore processor machines. Unlike concurrency, Parallel computing is not possible in a single processor machine. It needs a multiple processor machine. A complex problem is subdivided into multiple smaller tasks and then each task is computed in a different processor with an intention of increasing computation power. All Parallel computing is concurrent computing and not vice versa. Figure. 1.4 shows the execution of tasks in different processors at the same time.

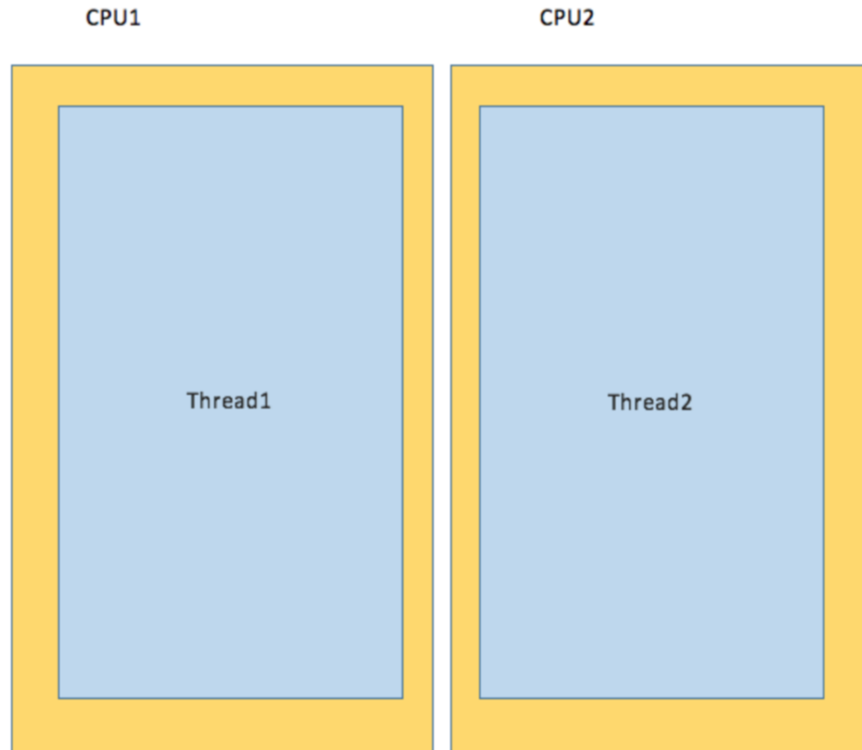

Figure 1.4: Parallel Computing.

1.2.3 Necessity of Concurrency

Concurrency is a computation task that is carried out concurrently at the same time. With the advent of technology and as everything is going online, large amounts of data is accumulated in a small fraction of time. The simplest way of processing the data is through sequential computing. However, the demerit of sequential processing is that it takes a significant amount of time for processing even a small amount of data. If only more than one task can be computed in a concurrent fashion on multiple processors, the time can be drastically reduced.

Moreover, with the advent of technology, computers are available with multicore processors, which are capable of carrying out multiple tasks concurrently in it. Without concurrency, the multicore processor is of no use as only a single core will be used each time and the overall computation power is of no difference with the sequential execution. The main goal of multiple processors is to speed up computation and these multiple processors can be utilized using concurrency to compute the tasks in different processors in a concurrent fashion. So with the concurrent application, the

time of execution is lowered. However achieving concurrency requires advance communication and consensus algorithms between the different tasks.

1.2.4 Shared Memory And Concurrency

Shared memory is a global memory which is accessed by multiple processes for a efficient means of passing data between them. Shared memory systems occupy a major portion in the field of multiprocessor and parallel computing. Tasks running on different processors communicate through reading from and writing to the global memory. Depending upon the access time of a memory, shared memory can be classified as the following:

- Uniform Memory Access(UMA):
  In UMA, access time to any memory location by all the processors is equal.
- Non Uniform Memory Access(NUMA):
  In NUMA, access time to memory is non-uniform with respect to different processor.
- Cache only memory Architecture(COMA):
  In COMA, local memories are used as cache. In order to access a remote data by a processor, the processor first looks in its local cache memory .If not available then it migrates the complete data to its cache and then reads from that local cache .

1.2.4.1 Atomic Primitives for Shared Memory

Execution of a sequential program in shared memory is really easy. The program executes one step at a time without worrying much. However, many modern applications are concurrent, so there are multiple threads executing in parallel. These multiple threads interact with shared memory with reading and writing on it. Handling of these concurrent threads in a shared memory system requires synchronization. Traditionally synchronization in multiple threads is obtained by means of locking. Semaphores, Monitors, Mutex are some software constructs for locking. However, locking can lead to deadlocks, live-locks, convoying and priority inversion. So some popular atomic primitives for concurrency without using locking are mentioned below.

1. Compare And Swap(CAS).
   CAS(A,E,D) verifies whether the value of address A has expected value E. If so, then the

value of A is set with the desired value D with returning true, otherwise the value of A remains unchanged with returning false and the value of A is stored in E. This CAS operation ensures that the address A is unchanged by the other threads since the last read by a particular thread.

Algorithm 1: Compare and Swap(CAS)

Result: Boolean

Input : *address*,*expected_value*,*desired_value*

```
1 if address == expected_value then
2     address ← desired_value
3     return True
4 end
5 else
6     expected_value ← address
7     return False
8 end
```

2. Load-Linked/Store-Conditional(LL/SC).

   LL/SC is a stronger primitive for atomicity. LL copies the shared variables to local variables. Subsequent SC variables store the local variables to shared variables if no other thread has made any changes to the shared variable. LL(Loc) returns the current value of address location Loc and with SC(Loc,V) , a thread P_i sets the value of Loc to V only in the condition that no thread has changed the value of address location Loc since the execution of p_i' s latest LL on Loc.

Algorithm 2: Load-Linked/Store-Conditional(LL/SC).

```
1  Function load_linked_i(Loc):
2      read location Loc
3      Marks location Loc as "read by i"
4      return Loc
5
6  Function Store_conditional_i(Loc, V):
7      if Loc marked as "read by i" then
8          write value V to Loc
9          return Success
10     end
11     else
12         return Failure
13     end
```

Chapter 2

Background

The easiest way to make a payment between two parties is through physical currencies. Such kind of payment doesn't require the third-party as it makes payment between two parties in person. With the advancement of technology, nowadays most of the transactions are done online through the internet and such payment cannot be done without third party involvement.This third party is centralized and must be trusted one as it has sole power of controlling the system. Moreover, involvement of third party in transaction increases cost. So In 2009, Decentralized payment system called Bitcoin was coined to make the payment over the internet without depending on trusted third parties. This Bitcoin is the hot topic in the market these days and most of the companies are accepting Bitcoin as the transaction medium and getting popular day by day.

2.1 Bitcoin.

Bitcoin[Nak09] is the decentralized, distributed, peer to peer virtual cryptocurrency which does not depend on any centralized financial institution called third parties for managing the flow of currency. It is the instant way of exchanging values with the promise of not needing a middleman. Bitcoin is not physical coins. It is purely virtual currency. Since Bitcoin doesn't have any centralized server for controlling the flow of currency, it is created by the process called mining[2.1.8]. Every user in Bitcoin has sets of public and private keys[2.1.2]. These keys are used to prove the ownership of Bitcoin in the network. When one user needs to transfer a certain amount to another user, then the user creates the transaction and signs it with its own private key and spend it by transferring to a new owner. Thus the only prerequisite to spend Bitcoin is that it needs to possess the key to sign the transaction. Every transaction created must have reference to a previous trans-

action crediting the user. This possession of key puts the control entirely in each user. Bitcoin is called cryptocurrency since encryption techniques are used to regulate the generation of currency and verify transfer of funds without the need of central bank.

Mining[2.1.8] is the cryptographic competition to find the solution to a mathematical puzzle. Built-in algorithms are used to control the mining function in the network. Any machine in the Bitcoin network can act as a miner. This miner uses its processing power to solve the puzzle and broadcast in the network. A new block of transactions is created by miners in the network in every 10 minutes on average. The difficulty of the mathematical puzzle is adjusted dynamically such that average of 10 min is required to solve the puzzle irrespective of the numbers of miners active in solving the puzzle. Thus, Bitcoin is the internet form of money that can be used as a replacement of physical money for buying and selling of goods. It can be purchased, sold and even exchanged for other physical currencies

2.1.1 History of Bitcoin.

Bitcoin was coined by a mystery person named Satoshi Nakamoto in 2008 by publishing of paper entitled “Bitcoin: A Peer-to-Peer Electronic Cash System” [Nak09]. Before the invention of Bitcoin, there were several digital cash systems like hash cash[Bac], b-money[Dai18]. These systems weren't able to address the double spend problem[dou] properly which was one the most important weakness. Then a new cryptocurrency called Bitcoin was emerged that used the computation system called proof-of-work algorithm(PoW) to reach the consensus and election of a new block in every 10 minutes[Nak09]. This PoW elegantly solved the double spend problem existed in earlier digital currencies. No one controls the power over the Bitcoin system. It is operated fully on transparent mathematical principles, open source code and consensus among participants. Due to this features, Bitcoin was soon popular and currently is the most expensive cryptocurrency in the market.
The Bitcoin network was started in 2009 with the release of Bitcoin client and Bitcoins by Satoshi Nakamoto. Though Nakamoto published the paper and initially started a new technological leap in digital currency, he didn't remain involved in the Bitcoin technology for long. He left the responsibility of developing code and maintaining Bitcoin network to some volunteers and disappeared.

2.1.2 Keys.

Bitcoin is based on the branch of mathematics known as cryptography. Digital signature of cryptography is used in Bitcoin to prove the ownership or authenticity of data. Digital Signature, digital keys are used in Bitcoin to establish the ownership of Bitcoin. Digital keys are the private and public keys extensively used in cryptography. These digital keys are stored in a simple database called wallet[2.1.4]. These keys in a wallet are completely independent on Bitcoin protocol and generated in the wallet and managed by wallet software.

The private key is used to create the digital signature and the public key is used to validate the signature without knowing the private key. In Bitcoin system, when a user wants to spend a Bitcoin, the owner signs the transaction with private keys and sends the public key along with the transaction. With this public key and the digital signature in the transaction, everyone in the network can verify that the transaction is properly spent by the rightful owner.

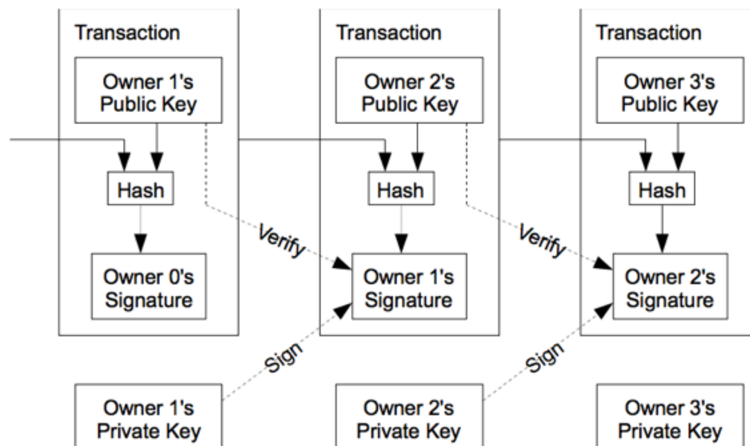

Figure 2.1: Keys in Bitcoin[Nak09].

2.1.3 Address.

Every user in Bitcoin network has address. Address is just like the account number in a banking. system. Just like account number are used to transfer the amount from one person to another in a banking system, the same way address are used to transfer Bitcoin in the Bitcoin network. Bitcoin address is shared with anyone who wants to send you money.
e.g. of Bitcoin address:

1F7mDg5rbQyUHENY2x39WVWK7FsLpEoYZy

Let us suppose my bitcoin address is as mentioned above, Now when any other users want to send some bitcoin to me, then they use my address as the receiving address and then send bitcoin to me.

2.1.4 Wallet

Wallet is basically the equivalent form of user bank account.User's digital keys and address are stored in the wallet. It is a user interface to receive Bitcoin, store them and send them to others. It is used to manage user's money, key and addresses. It is also used in computing the balance, creating a transaction and signing it.

Amount that can be spent by the user always comes form the output of previous transaction that is credited to the user address. This spendable output is created by wallet as UTXO(Unspent Transaction Output Set)[2.1.5.1] by aggregating all the output credited to that address. Whenever a user needs to send Bitcoin to some other address, wallet creates the transaction from the UXTO, digitally signs the transaction and send it to the Bitcoin network with receiver address in it. Wallet displays overall amount that can be spent, however, it keeps the output separate and distinct internally.

2.1.5 Transactions.

Transaction is the important part of Bitcoin system. Transaction is the transfer of Bitcoin value between different addresses which is analogous to a transfer of physical money between bank accounts. The Bitcoin value is transferred using transaction which is then broadcasted to network and then collected to the blocks[2.1.6.1]. During the creation of new transaction, new transaction

input is the references to an output of previous transactions that is already verified in blockchain network . Every transaction is public in Bitcoin's blockchain.

2.1.5.1 Transaction Inputs and Outputs.

Transaction input and output are the fundamental things of Bitcoin system. Every other thing are designed in order to propagate these transactions through the network, validate and addition to the ledger. Unspent transaction output(UTXO) are the Bitcoin values that are available and are spendable. The collection of all UTXO is called UTXO set. The UTXO is kept tracked by the full nodes in the Bitcoin network. When a wallet received Bitcoin, it means that UTXO has been detected by wallet which can be spent with the digital keys it has. The Total balance in a wallet is calculated by the sum of all the UTXO which can be spend by the user's wallet. This UTXO may be scattered through thousands of transaction and blocks. Wallet scans all the blockchains and transactions, then the value of Bitcoin for the particular wallets are aggregated to calculate the total available Bitcoin amount for the wallet.

As we know earlier in Bitcoin wallet [section 2.1.4], wallet is used to create the input transaction. This input transaction is created by using UTXO. The wallet tracks the UTXO for the particular address and gives the total spendable output for that particular address.This spendable output is used to create future transactions by the wallet. Every new current transaction created must be referenced to previous transactions through UTXO. This way Bitcoin value moves from one owner to another owner in a chain of transaction consuming and creating UTXO. Satoshis is the smallest unit in Bitcoin just like cents are in dollar. As dollars can be divided to cents as two decimal values, similarly Bitcoins can be divided to 8 decimal values called Satoshis.

So in overall, When a user A wants to spend bitcoin to buy certain material, he first have to look in to his wallet whether he has sufficient available UTXO or not. As UTXO refers to output of previous transaction, this unspent output(UTXO) may be larger than the value that we are willing to spend. In such case when input is greater than the required output, the changed must be generated and returned to the same user. Suppose a user named Ram has UTXO of 1 Bitcoin and if he requires just 0.2 Bitcoin to buy certain material. Then change of 0.8 Bitcoin is required to be returned to the same user.In another case a single UTXO to create new transaction may be smaller than the required output, then multiple UTXO must be aggregated to create the output

with larger value than required.

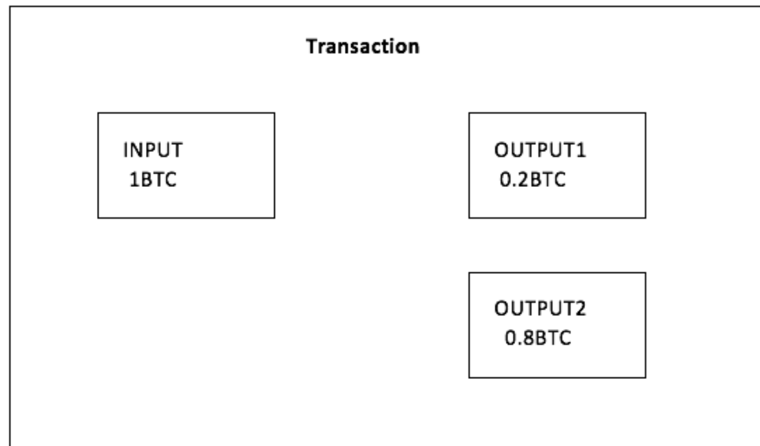

Figure 2.2: Transaction with single input

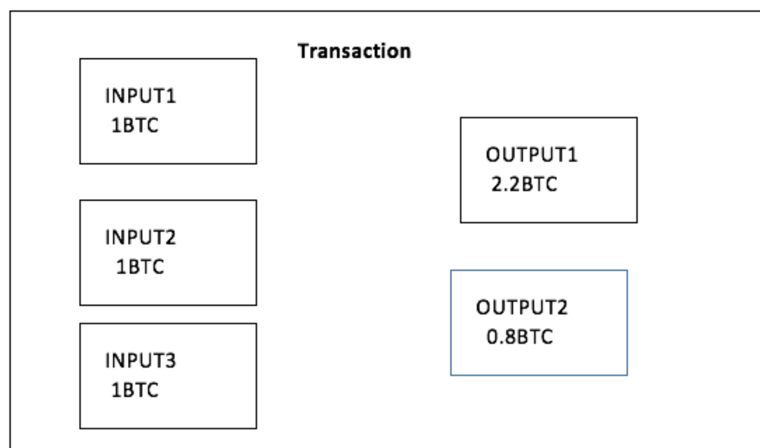

Figure 2.3: Transaction with multiple inputs

Figure 2.2 is about the single input where an input is greater than the required output. Here 1 BTC is broken into 0.2 required output and 0.8 returning output. Similarly, Figure 2.3 is multiple inputs where single input may be insufficient to create required output. 2.2 BTC is required to buy some product, thus multiple UTXO is aggregated to form 2.2 BTC. Since the accumulated multiple inputs are greater than 2.2 thus second output 0.8BTC is the returning output to the owner who created the transaction.

2.1.5.2 Transaction Fee.

As in Figure 2.2 and Figure 2.3, Output may not be always equal to the input. The difference in the outputs and inputs is the Transaction fee. Most transactions include fee. This fee is used as an incentive to the miner to mine the block and include the transaction in a block. This is important to protect the Bitcoin system against abuse by the fraudulent user. This fee prevents an attacker to flood the network with transactions. It is based on the size of the transaction in kilobyte. Any transaction fee is calculated as the excess of inputs minus outputs as:

$$TransactionFees = \textsc{sum}(Inputs) - \textsc{sum}(Outputs)$$

Thus in overall, to create a new transaction, a user first has to have UTXO received from the previous transaction addressed to their address. With this UTXO, the user then includes the receiver address and the amount to be spent. The wallet then creates the full transaction with input as the reference of previous transaction and outputs as how much Bitcoin it is sending to a new address. The wallet will sign the transaction with its private key and also sends the public key with it so that any nodes in the network can verify the signature for the ownership. Each transaction can have one or more inputs and two or more outputs. The difference of inputs and outputs is always the transaction fee which is collected by the miner[2.1.8] as an incentive for mining a block.

2.1.6 Blockchain.

Blockchain[Nak09] is the undeniably ingenious invention of the 21st century. It was first implemented by Bitcoin thus is the important innovation of Bitcoin. Originally implemented by Bitcoin for the digital currency and now its potential is being researched to other fields as well. It is thus regarded revolutionary technology having the potential to change the world with Bitcoin gaining more and more popularity both technically and economically.

The Blockchain is a back-linked list of blocks. It is a distributed ledger technology which makes Bitcoin as a decentralized system. It is a technology that makes the Bitcoin network independent on a centralized third party.

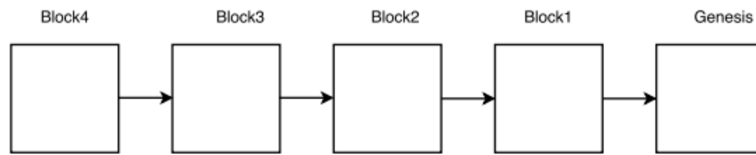

Figure 2.4: Blockchain

Figure 2.7 is the blockchain which is linked to each other. The beauty of this blockchain is that any changes in the content of the block in the network is identified
Blockchain is categorized in to two kinds:

1. Permissioned Blockchain.

2. Permissionless Blockchain.

The Blockchain is permissionless blockchain if any node or actor can join the network at any time without taking permission from other nodes in the network. Bitcoin is an example of permissionless blockchain. Furthermore, any node should not prove their identity to be a part of a network for extending the chain. Whereas permissioned blockchain has the requirement to prove the identity before joining a network for extending the chain and building consensus. Mostly Financial institution that requires private blockchain uses such kind of permissioned blockchain.

2.1.6.1 Block.

Blockchain consists of a chain of blocks linked to each other. Block is the building block of blockchain. Blocks consist of transaction data in it. Every block have referenced to a previous block thus they are linked in a chain called blockchain. Blocks can be considered as an individual page of a book where each line in a page as a transaction. New transactions are processed to a block by miners and then added to the chain. Once the block is added to the chain, then they are permanent which cannot be removed or changed by the network. The first created block is called Genesis block. After that, the miners create the valid block and then are continuously added on top of the already created block. Figure 2.7 shows the blockchain of 5 blocks. The first block in Bitcoin is called genesis block.

2.1.6.2 Structure of Block

Block is a data structure for holding the transaction to include in a blockchain. Block structure consists of block header followed by transactions. The average number of transaction in a block is more than 500 transactions. Figure 2.5 shows the metadata contained inside block.

Field	Description
Block size	The size of the block in bytes.
Block header	Block header with several fields in it.
Counter	The total number of transaction.
Transactions	Transactions in the block.

Figure 2.5: Structure of Block in Bitcoin.

2.1.6.3 Block Header

Block header is an important metadata of block structure. Hash of the Block header is the block identifier which is distinctive and different from each other. As we know that in the blockchain, the blocks are linked to each other with each block referencing to the previous block and this referencing is done through the block header. The Hash of the block header is used to link to the previous block. Figure 2.6 shows the structure of block header included inside block structure.

Version	Version number
Previous Block Hash	Reference to the hash of parent block in blockchain
Merkel root	Hash of root of the merkel tree of block's transaction
Timestamp	Creation time of the block
Difficulty Target	Difficulty target to Proof of Work algorithm.
Nonce	Counter used for Proof of Work algorithm.

Figure 2.6: Structure of Block in Bitcoin.

2.1.6.4 Block Header Hash

Cryptographic hash of the block header is the primary identifier of a block. Generally, it is called block hash, though hash is of block header only. Block header hash is different for each block thus is unique and different, which uniquely identifies the block. The first ever created block of Bitcoin

which is called genesis block has no previous hash field and all the blocks added after that has the previous hash field which has the hash value of previous block header. This block header hash is used to link the blocks into blockchain.

2.1.6.5 Genesis Block

The first block created in the Bitcoin is called Genesis block and was created in 2009. This block is the parent of all the blocks in Bitcoin. The height of this block is 0 after then with each block added on top of it increases the height of block with 1. If we trace back the block from any block, finally it must reach to the genesis block. There is only a single path to reach to genesis block from any block in Bitcoin blockchain. Genesis block is statically encoded in every node in Bitcoin network.

2.1.6.6 Chain of Blocks in Blockchain.

Blocks are linked in a chain forming of blockchain.The chain of blocks is formed through the reference of the hash of previous block. Before building of blocks, previous block's hash is collected in a block header and hash for that block is created. Thus the blocks are linked together to one another through the hash of the previous block. Hash of any block is always unique and considered as the identity of the block as well. Due to linked references of the blockchain, any changes of the data in the parent block changes the header (identity) of the child which in turn changes its hash because the child uses the hash of the parent. This changes in the hash(identity)in a child causes to change the hash of a grandchild and so on. The changes in the hash (identity) of parent effects all the following children to change its hash(identity) which requires a lot of computation to change all the subsequent blocks. This linking of blocks through the hash of previous block makes blockchain immutable. If any of this block is tried to be altered by the illegal miner, then the all the children below its blocks need to be changed as its children are linked through it, which is impossible for any miner to do so unless it has computation power of more than 51% [mom14].

Generally, a block is referenced to a single parent block. However, under certain situation, a single parent can have multiple children. When two blocks are mined simultaneously and route through a different route in a network, then there is a possibility of multiple children for a parent. This situation is called fork. The fork is resolved by using longest chain principle and finally, there is a single child for a single parent. Blocks in a shorter chain are invalid block. When the fork

is resolved, the blocks in shorter blockchain are invalid and the transactions in shorter blocks are rewinded back to the unconfirmed pools[2.1.8] and will be included in later blocks.

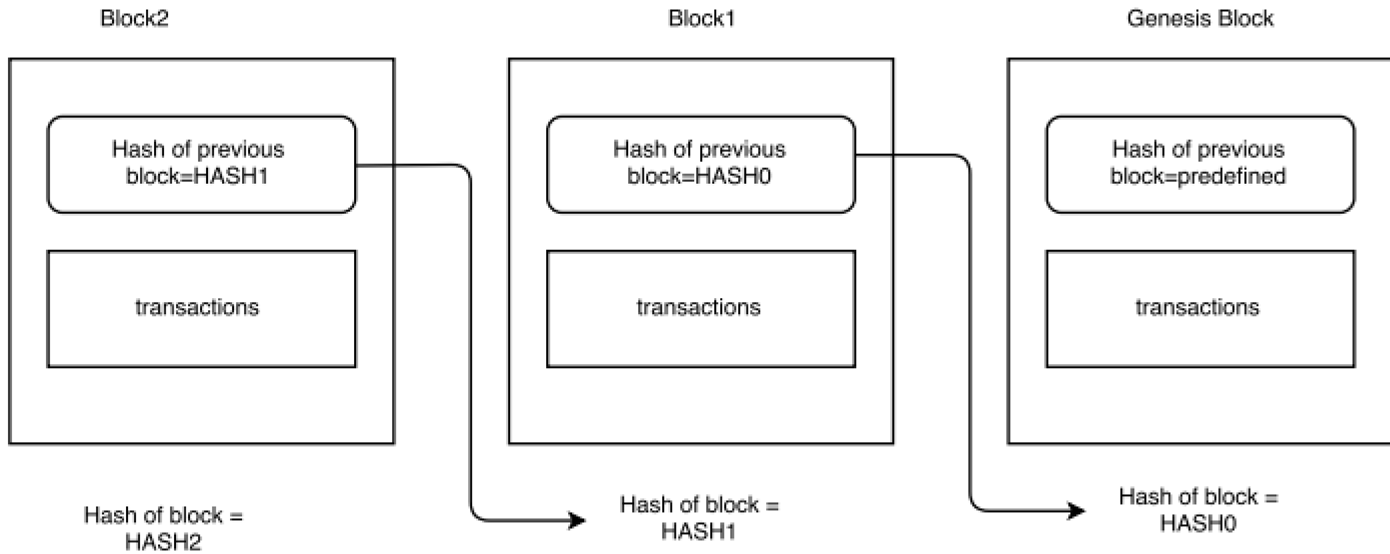

Figure 2.7: Chain of block in block chain.

So Figure 2.7 shows how the blocks are linked together in a blockchain using a hash. Genesis block hash is HASH0. This hash value is used by Block1 header and using this HASH0, hash value of Block1 is created as HASH1. Now the hash value of Block1 which is HASH1 is used by Block2 and creates the hash for Block3. This using of previous block hash continuous. As a result, the blocks are linked together forming immutable blockchain.

In Bitcoin blockchain, the Bitcoin full nodes keep the copy of the all the blocks starting from the genesis block. The copy of the nodes gets updated whenever new blocks are identified. Whenever a miner in the network finds a valid block, first it updates its blockchain. Then the new block is relayed to other nodes in the network so that other nodes also update their copies. Other nodes on receiving a new block , first they validates the block and then updates the blockchain.

2.1.7 Merkle Tree.

Merkle tree[Mer] is a binary tree data structure in which every non-leaf node is the hash of its children. Merkle tree provides an efficient way of verifying whether the data is included in a binary

tree or not.

In Bitcoin, merkle tree of transactions is calculated and finally, root of the merkle tree is used in the header instead of the whole transaction. Since the hash is of constant length, merkle root is always of constant length thus hashing of the block header is easier and efficient. Merkle root allows us to verify the data and supports to securely verify that the transaction has been accepted by the network. The great advantage of merkle tree in Bitcoin is for lite nodes. They don't need full block with every transaction instead can request just block header and merkle branch from other nodes and recreate the required data.

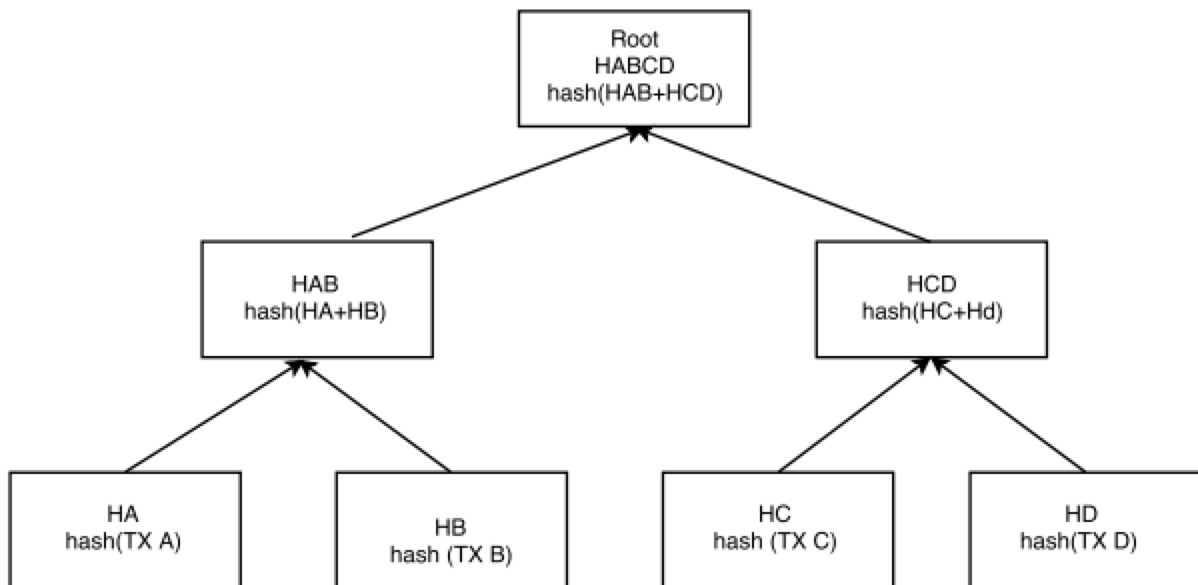

Figure 2.8: Structure of Merkle Tree

2.1.8 Mining and Consensus

Blocks and blockchain are the important factors in Bitcoin. Transactions are aggregated to the blocks and these blocks are added to the previous valid block forming a blockchain. This addition of transactions in a block and linking a block to the previous block is done by the miner and this process is called mining. Mining is a process of verifying transactions and creating a valid block for forming blockchain. This creating of new blocks will create new Bitcoins thus mining is also called

as a process through which new Bitcoins are released by the miner nodes as does the central bank .

In the Bitcoin network, a network consists of several nodes. These nodes may be relay nodes, clients and miners.Miners on receiving transactions from other nodes, stores in memory pool called mem pool. These transactions in mem pool are known as unconfirmed transactions which are yet to be included in blocks. During mining process, miners select some unconfirmed transactions from the mem pool and they verify the transactions. Once the transactions are verified then they create a valid block solving a mathematical puzzle called Proof of Work(PoW)[2.1.8.1]. This block is then added to the blockchain of a local miner and then relayed to the neighboring miners in the network. On receiving the block from another neighbor, they verify the metadata of the block thus verifying the block. Once neighbor verifies the received block, they acknowledge that they lost the race of finding the current height block and adds the newly received block in their blockchain. After that, they again start competing in the race to find the new block of different height.

On average, a new block is identified in every 10 minutes, thus adding transactions to the block and to a blockchain takes an average of 10 minutes. Transactions that are verified and added to a block are considered as confirmed, which gives the permission to the new owner to spend that amount they received through the transaction. The miner who mines the block gets rewards. There are two kinds of rewards owned by the miner who mines the block. New Bitcoins are created with each block creation. This new Bitcoins are obtained by miners as a reward and another reward is transaction fee obtained from all the transactions included in the block. This incentive of getting two rewards motivates the miners to continuously compete in the race of finding a valid block. The valid block is found by the miner by solving a mathematical puzzle using a cryptographic hash. This process is called Proof of Work(PoW).

Just like central bank issues money and money supply is made, similarly, Bitcoin manages the supply of Bitcoin through the creation of new Bitcoin on solving the block by the miner. Whenever a miner solves a block, a new Bitcoin is created which is then rewarded to the winner miner. As earlier mentioned, a new block is created in an average of every 10 minutes and also known that with each block created, new Bitcoin is created. This means that in every 10 minutes, new Bitcoin is created. Currently, 12.5 Bitcoin is generated with each new block found. This creation of new Bitcoin is not constant though, creation of new Bitcoin decrease by half in every four years (every

210000 blocks approx.). e.g. in 2009 when Bitcoin was created, the incentive for new block creation was 50, which decreased to 25 in 2012 and then to 12.5 in 2016. Thus the current rate of creating Bitcoin is 12.5 for each new block creation. Based on this formula of decreasing Bitcoin rate, in 2140 all the Bitcoin will be issued and the incentives will only be based on transaction fees.

Transaction fee[section 2.1.5.2] is the difference between input and output. This difference is also added as the reward to the miner as keep the change. When the rewards from new Bitcoin creation gradually decreases, the rewards from transaction fee will occupy the greater portion. And after 2104, all the new Bitcoin creation will be zero and Bitcoin mining rewards will be cover only by transaction fees. Experts in Bitcoin are concerned about this situation of rewards only through the transaction fee. [CKN16] explains the instability without the block rewards.

2.1.8.1 Proof of Work (PoW)

Miners in Bitcoin are involved in mining for finding the valid block. Valid block finding in Bitcoin involves solving a mathematical puzzle. This mathematical puzzle which needs to be solved by the miners is Proof of Work(PoW). Mathematical Puzzle in block creation is a cryptographic hash. All the miners continuously work for finding valid hash. Any miner which first find the valid hash is the winner and takes all the reward. PoW is based in the cryptographic hash. Hash has the important feature that result cannot be determined in advance and also there is no any pattern in the hash value So there is no certainty that which miner will solve the hash. The network will set the difficulty level of the hash known as hash difficulty[2.1.8.2] and all the miners try to solve this hash puzzle satisfying the difficulty level. This PoW algorithm of finding the correct hash is the backbone of the blockchain.
Let us take an example such that we are creating a 101st block. First, the block header is created by a miner with nonce value[2.1.8.3] as 0 and target difficulty as first 5 bit of the hash as 0. with the nonce value as 0, this becomes the candidate block. Hash for the candidate block is calculated and compared against a target. The candidate block is updated with iterating the nonce value and finding the hash until and unless the hash of the candidate block is less than or equal to target.

$$Validitycondition = Hash(candidateblock) <= Targetdifficulty$$

.

2.1.8.2 Target difficulty

Target difficulty is the field involved in mining. It is set by increasing or decreasing the starting bit of block hash to be 0. For e.g. network can set target difficulty as the first 5 bit of the block hash must be 0. Miner continues its PoW algorithm such that the final hash of the block is less than or equal to the target difficulty. The main aim of the miner is to mine a block such that its hash value is less than target difficulty of the network. Target difficulty in the block header is dynamically set by the network so that the blocks are created in every 10 minutes on average. If the earlier block was created in less than 10 minutes, then the target difficulty is set to more difficult such that next block is more difficult to find. Increasing the difficulty by 1 bit causes a doubling in the time to find a solution so they will take more than 10 minutes to find the next block and in average will be 10 minutes.

2.1.8.3 Nonce

Nonce is the field in the block header which involves in PoW. As earlier explained in PoW[2.1.8.1 hash is calculated by each miner, first setting the nonce field to 0. Whenever the target difficulty is not satisfied by calculated hash, miner needs to again calculate the hash of the block. For this calculation of hash, some value of the block header needs to be changed. This nonce is the value which is changed repeatedly in order to calculate a new hash value of the block. Nonce is a counter used in the header, which the miners manipulate to change the hash value of a block to meet the hash criteria with difficulty satisfaction.Nonce value will start from 0 and continuously increased to create a valid hash.

2.1.9 Decentralized Consensus.

Blockchain is the backbone of the Bitcoin where there is no any centralized authority as in traditional system. It is obvious to have the question on the mind "how can the consensus be reached without trusting anyone". In Bitcoin there is no central authority, but every node have the copy of full blockchain that it can trust thus node doesn't need information from any centralized system. All nodes in the network have the same blockchain achieved through consensus of all nodes in the network. All the nodes in the network reached to the common conclusion and update the blockchain by all the nodes so that the final blockchain is the same. Thus overall process involved in consensus from start of transaction to blockchain formation are:

1. Verification of each transaction by a full node.
2. Combining the transactions in to block with proof of work computation.
3. Verification of block and adding to a chain forming a blockchain.
4. Resolving Forks

2.1.9.1 Verification of each Transaction by FullNode.

As we have discussed in wallet section[2.1.4], wallet is the one which creates the transaction with input from the UTXO and assigns new owner of the transaction as the output. These transactions are then relayed to other nodes. These nodes don't simply relay transaction but first, it verifies the validity of transaction so that only the valid transactions are relayed and invalid are dropped. Every node first verifies the validity of transaction against a long list of criteria. Once the transactions are confirmed valid, then the transaction is placed in a mem pool as unconfirmed transaction before relaying it.

2.1.9.2 Combining Transaction into Blocks.

Among several nodes in the network, some of the nodes are mining nodes which are supposed to solve the computation puzzle to create a block aggregating the transactions. Step by step process of creating a block by miner are mentioned under:

- Constructing Block Header

  First every miner, on receiving transaction validates the transaction and place the transaction to memory pool as an unconfirmed transaction. Later it also relays this transaction to other nodes for propagation in the network. Every Miner then selects some unconfirmed transaction and then create the block header by filling the six fields of block header[2.6] . Let us take an example such that current block in the blockchain is 100. So now we are trying to find 101 block. In such case, header of 101th block will contain the hash of 100th block. The next step is that the transactions are summarized in merkle tree and merkle root is placed in merkle root field. Difficulty Target refers to Proof of Work to make the block valid. And finally, Nonce is initialized to 0.

- Mining the Block with PoW Algorithm

  Now the candidate block is created by miner and miner will run the PoW algorithm to make the block valid. PoW algorithm is the hashing of the candidate block header repeatedly until the hash value matches the difficulty target. Hashing is done continuously changing one parameter of the block header which is nonce. Nonce value will start from 0 and continuously increased to create a valid block. So the same process of PoW is carried by all the miners. The one to find the valid block with less than difficulty target is the winner. This way a new block is created.

2.1.9.3 Verification of Block and adding to a chain forming a Blockchain

Once a valid block is found by a miner. The winner first updates its own local blockchain, removes the transactions from the unconfirmed transaction as they are confirmed now and then propagates the block to other miners. The other miners on receiving the block acknowledge that another miner has won the race of finding the current block. It then verifies the block. Verification of block is easier as it needs an only single hash to verify whereas millions of hash are required to find the one. If the block is valid, it will drop all its PoW for the current block with the information that another miner is the winner. The transactions within the block are removed from an unconfirmed transaction in other blocks as well. Now the receiving miner will add the new block in its blockchain and starts mining the next block starting from creating new block header for the new block with different height.

So in this way, new blocks are created and blocks are linked together to blockchain without trusting any parties in the network. Once the blocks are verified and linked in a blockchain, the blocks are permanent. They are immutable. The more depth the block is in, the more it is secure.

2.1.9.4 Resolving Forks

Normally each block should have a single child however, it is not the case always. When two miners found a block in simultaneous similar time and blocks relayed through different path then at some point, there may be a situation that a parent may have multiple children. such situation is called forks. And even sometimes, valid blocks are found but parents are not found such cases is called

orphan block. Orphan blocks are formed when two blocks are mined in short interval of time and child is received before the parent due to a different path of propagation.

2.2 Pool Mining

Basically, Miner is the special nodes which mine the blocks and create the blockchain. Miners are the backbone of Bitcoin system continuously working on finding valid blocks and building blockchain. Miner will continuously work on finding the mathematical puzzle called PoW to find the valid block. The one which first finds the valid block is the winner and gets the reward. However, the major problem that occurs in mining is that some solo miner may never find the valid block in its entire life as might have a less computational capacity machine. The finding of valid hash is the main operation that occurs in finding a valid block. Since there is no any pattern in finding a valid hash nor there is turn in being the winner of a block, so any node can become winner depending upon their hashing capacity. So there can be a situation such that a solo miner with low computational hash capacity may never become a winner. In such case, there is no any reward for that particular node though it is continuously working on hashing. So to remove such disadvantage in mining, a concept called mining pool was coined.Mining with mining pool is just like a lottery. If a single person buys one ticket, in such case, the winner will take all the winning amount but the individual has less chance of winning. Now if a group of people forms a committee, and this committee buys 100's of tickets in bulk, in this case, the chance of winning the lottery by the committee is higher, however the reward is distributed between the members of committee depending upon shares.

Mining pool is the collection of a miner. It is the sharing of resources over the network to find the valid block with splitting the total reward to the members of the pool based on the contribution of work in finding the valid hash. Even when solving for slow solo miners can take years in finding the valid block, the mining pool with numbers of miners will contribute in finding the valid block by sharing their resources and the rewards obtained will be shared between the contributing members. This way, smaller miner gets a fraction of Bitcoin on a regular basis without waiting for years to generate Bitcoin on its own. In such a way of sharing resources and reward, the variance of miners is lowered in compared to the solo miners.

2.2.1 How Mining Pool Works

Mining pool is the collection of miner in a committee. Mining pool has one leader called pool operator. Only pool operator has the entire blockchain and rest don't have entire blockchain .

The pool operator first creates PoW with lower difficulty than the network and sends it to the miners. Pool operator aggregate transactions, build the candidate block then sends block templates to all the miners in the pool. The miner then uses this block template to mine the block with a lower difficulty. This block with lower difficulty is then returned back to the pool operator. The pool operator then keeps records of the shares submitted. While the miner submits the shares, some shares will have chances of having target difficulty greater than the difficulty of the network. Then the valid share for the network is found. The pool operator than submits the valid hash to the network. The rewards obtained will be shared by all pool members depending upon the shares submitted by the miners in the pool.

2.2.2 Distributed Mining Pool

Most mining pool has the owner called the pool operator. Since the pool operator chooses its own transaction and candidate block with target difficulty. Such pool is the centralized pool. Though the variance of rewards is low, miners need to trust on the pool operator. The miners cannot choose their own transactions, they have to work on the transaction as selected by the pool operator. Centralized pools have the possibility of cheating and have single point of failure. If the pool operator is down, then the whole pool will not work. To remove the centralized pool mining and its demerits, decentralized mining concept was introduced. The first ever known decentralized pool mining is called P2Pool[p2p].

2.2.2.1 P2Pool

P2Pool is a decentralized pool mining without central pool operator. The disadvantages of central pool mining are eliminated by a method of P2Pool. P2Pool implements a similar concept of Bitcoin blockchain to decentralize the pool mining. As Bitcoin uses blockchain, similarly P2Pool mining implements a parallel blockchain called share chain. Blocks with less difficulty are known as share and these shares are chained together to form share chain. New blocks are added to a sharechain in every 30 seconds. Every pool miner has the record of sharechain. Each block in the share chain

records the share of the miners who contributed the work and then also relayed to other miners. When the share blocks achieve the target difficulty of the Bitcoin network, then is propagated and included in the Bitcoin network, rewarding all the contributor the necessary shares. So the pool miners can select their own transaction to create the candidate block without the centralization of pool operator. Share chain uses decentralized consensus share chain mechanism just like Bitcoin's blockchain mechanism.

Disadvantages of P2pool:

- Every mining pool should contain enough space for holding share chain.
- Number of message exchanged between miners in P2Pool in linearly dependent on number of shares in the pool

2.3 Ethereum

Ethereum[But13] is the software platform based on blockchain technology which enables developers to build and deploy the decentralized application. As Stated by Sally Davies, FT Technology Reporter, Bitcoin is to blockchain where email is to internet. Email is just a single application in internet and several other application exists on the internet, Similarly, Bitcoin is just a single blockchain application for peer to peer electronic cash application. This blockchain is not just limited to Bitcoin, thus ethereum acts as a platform for the developers to build new blockchain application. Miners in ethereum use ether unlike Bitcoins in Bitcoin network. Ether is used for transaction fee and services in ethereum network. The average block creation in ethereum is 20 seconds unlike 10 minutes in Bitcoin. The most important feature of ethereum is the smart contract.

Thus Ethereum is a public, open-source, decentralized platform based on blockchain featuring smart contract. Ethereum allows for building a decentralized application on top of a blockchain.

2.3.1 Origin of Ethereum

Ethereum was first proposed by Vitalik Buterin in 2013 with aim of building and deploying decentralized applications. Buterin first proposed need for scripting language in Bitcoin for application development, however, he couldn't gain agreement. Thus he suggested development of new platform which is named as Ethereum. This project was crowdfunded online.

2.3.2 Smart Contract

Contract is an agreement between parties. This agreement written in code and placed in the blockchain is the smart contract. Smart contract is the self-operating computer program(code) that automatically executes when certain conditions are met. It resides in the blockchain and facilitates the exchange of property, money or any value after it is triggered by the transactions. It is the contract or agreement between parties that is written as code into blockchain at a certain address. Smart contract has its code, its storage and changes its state when triggered by the events. This code executes automatically when triggering events hit the contract. In order to invoke a contract at address @, the user sends a transaction to address @, then the code gets automatically executes and changes its states according to the program of smart contract.

This smart contract can be created by the developers, however they are public. This flexibility of ethereum allowing programmers to write the contract makes ethereum popular for building application. Simple example of a contract is the vending machine where the machine is programmed (contract) in it. Whenever a person inserts some coin in it, the program gets triggered and we can get the item from the machine.

```
if money received == $2.50 && the button pressed is "Diet Coke"
then release Diet__coke
```

2.4 SmartPool

SmartPool[LVJS] is the distributed pool mining which uses smart contract and runs on Ethereum network. The drawbacks of P2Pool mining pool is eliminated by SmartPool

2.4.1 How SmartPool Works.

SmartPool is based on a smart contract which runs on the ethereum network. SmartPool contains two lists in the contract state.

- ClaimList

  It is the list of claims submitted by the miners.

- VerClaimList

  It is the list of claims that is verified valid.

The miners submit the shares in the form of claim. Once the claim is received by the SmartPool, it first places the claims in the ClaimList. Each claim has the definite structure, which has a number of shares submitted by the miner and the root of the augmented merkle tree that helps in verifying the claim. Once the shares are verified, they are then listed in a VerClaimList. For the efficiency and security purpose, claims are submitted to the SmartPool in the form of batches and during verification, all the shares are not verified but only some claims are verified. The verification of only some shares is to enable efficiency. Since only a few submitted shares are verified, the main challenge of verification is to prevent miners from over claiming the number of shares and invalid shares. Once the valid network block is identified, the reward is distributed to all the miners based on the VerClaim list since VerClaim list has the overall information of the claims submitted by the miners.

2.4.1.1 Claim Submission

Instead of sending all the shares by the miners one by one, miners send the shares in the form of batches of shares in a single claim. SmartPool defines a claim structure that contains only a few data. Miners submit the total number of shares and a merkle root of the batch of shares in a field called ShareAugMT[LVJS]. After miners submits shares in batches, SmartPool asks the miners to submit proof to ensure that the share submitted by miners is valid in a structure called ShareProof. Miners send the ShareProof to demonstrate that share has been included in the ShareAugMT.

2.4.1.2 Batching and Probabilistic Verification

Probabilistic verification approach is applied in SmartPool so as to increase efficiency and security. As we know that in ethereum smart contract, smart contract uses some fee in the form of gas. If all the shares are submitted and verified then a lot of amount is spent in the verification process, and when the valid network block is identified, the reward obtained is distributed, however, the expense on smart contract may be higher than the reward obtained. Thus among all the shares submitted only a few shares are verified. This verification of just a few shares ensures efficiency as only a few shares are to be validated which will be a lot faster than validating all the shares. Now the question arises that with just few shares verified, how can the SmartPool ensures that all the

shares are valid? Miners submit shares in a definite structure to ensure probabilistic verification. Moreover, smartpool [LVJS] explains that even cheating miners cannot be benefitted by claiming invalid shares.

2.4.1.2.1 Batching of Shares.

As earlier explained in section[2.4.1], shares are collected, batched then send to the SmartPool. The SmartPool then verify the shares with Probabilistic verification. The major problem with probabilistic verification are

- How shares repeated in a claim verified.
- How duplicate of shares in two different claim verified.

So the solution of this two problems is addressed before the shares are batched. During searching for shares by miners, each miner searches shares in an increasing monotonic order. Counter is used with each share and goes on increasing with new shares found. So when the claim is received by the SmartPool, SmartPool ensures that the duplicate counter value is not present in the claim. Since the verification of claim is done with probabilistic verification, how the unique counter value in the share are verified? To enable probabilistic verification with ensuring unique counter in share, SmartPool uses a data structure called Augmented merkle tree.

2.4.1.2.2 Augmented Merkle Tree.

Merkle tree[Section 2.1.7] is a binary tree with each node is hash of its children. With the merkle root, it is easy to ensure that the share exists in the tree. In SmartPool, we not only want to ensure share exists in the batch but also there are no repeats and ordering of counter is correct. Thus the addition of simple counter in merkle tree forms augmented merkle tree.

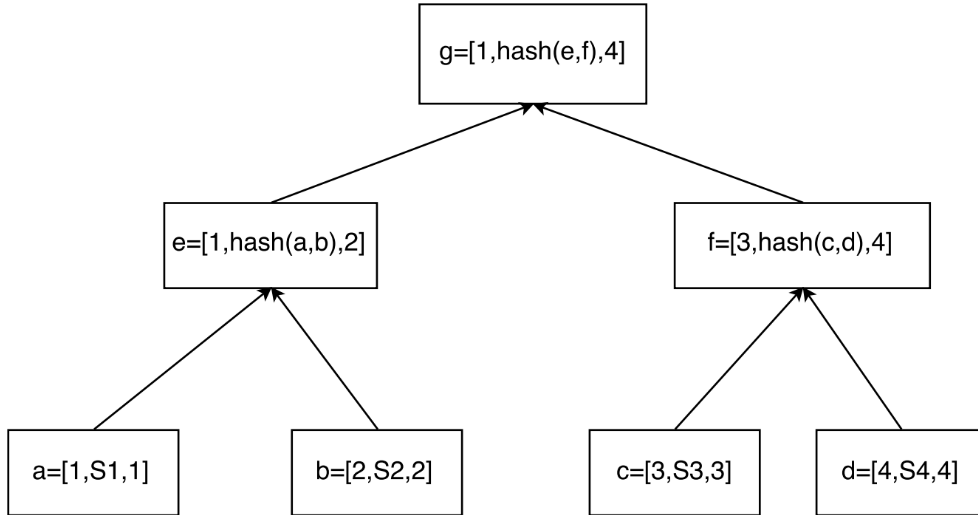

Figure 2.9: Augmented Tree for a list of Shares

A node in Augmented merkle tree has following fields for the node X:

- Minimum(X)
- Hash(leftnode,rightnode)
- Maximum(X)

Figure 2.9 gives an example of augmented merkle tree. The Minimum field of the tree must be always less than Maximum field for all non leaves node. If there are any duplicate shares in the batch, then the augmented merkle tree has a sorting error with min and max principle violated, thus this sorting error helps to detect the duplicate of the shares in the claim. To address the second problem, SmartPool has a latest counter value stored in it, thus when a new claim is received, SmartPool ensures that the recent counter received must be greater than the SmartPool counter. Since every counter is supposed to have a unique counter, this verification of last received claim is smaller than the latest claim ensures that no duplicate claim is submitted in more than one claims.

2.4.1.2.3 Batch Submission with Augmented Merkle Trees.

So after collecting the shares in monotonic order with a counter, miner locally forms augmented merkle tree for all the shares. Miner then submits the number of shares in a claim and the root

node of augmented merkel tree. After receiving claim by the SmartPool, it randomly samples the claim and request ShareProof from the miner. This ShareProof helps in finding duplicate shares in a claim with sorting error. Figure[2.10] shows the sorting error appeared due to duplicate shares. The non-leaf node is not supposed to have same minimum and maximum field for the same share and also the same level node should have an increasing order of the minimum, maximum field in the order going from left node to right node. If there is a duplicate share , the non-leaf node will have imbalance node. During the probabilistic verification, in case sorting error is detected then the whole claim is dropped. So in this way, with the help of augmented merkel tree, duplicate of shares in a claim is detected and verified.

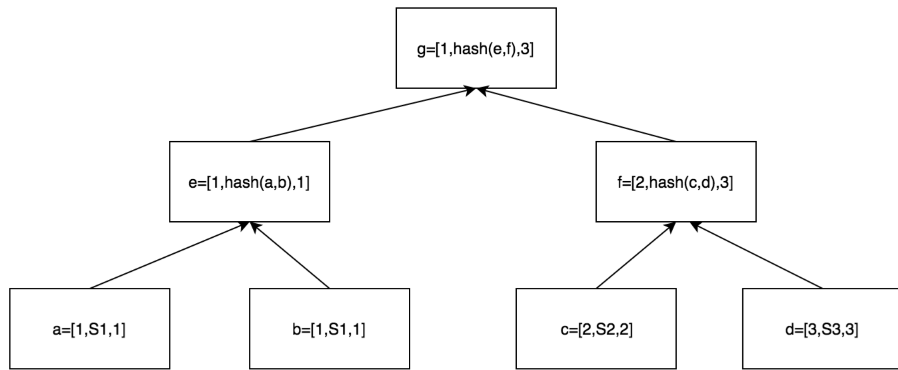

Figure 2.10: Error in Augmented Merkle Tree due to duplicate share

2.5 Transactional Memory

With the increase in popularity of shared-memory model, it is necessary to have the application more concurrent to take advantage of increased computational power provided by the hardware and chip. For the shared memory architecture, traditionally locks are used for designing concurrent data structure. Coarse-grained locking is easy to program but has limited concurrency. On the other hand, Fine-grained locking works better but hard to program. Unfortunately, locks often create a problem of race conditions, deadlocks, priority inversion. So currently projects that are alternatives to locks are gaining momentum. Lock free data structure avoids several problems associated with conventional locking. Transaction memory is undergoing extensive research as an alternatives synchronization mechanism to locks.

Transaction memory is a concurrent method for translating the sequential object into non-blocking. It simplifies the concurrent programming in an atomic way. It attempts to simplify writing concurrent programs without using locks using transaction concept. Transaction Memory(TM) has established itself as an alternative to traditional mutual exclusion primitives such as monitors and locks, which scales poorly and do not compose cleanly. In TM, activities are organized as transactions, analogous to database transaction which is executed atomically. A transaction may commit, which makes transaction's effect appear to take place permanently or may abort, with its effect appear not to have taken place at all. Transaction should follow the following properties:

1. Serializability

   Transactions are said to be executed serially when one transaction never appears to interleaved with the other.

2. Atomicity

   Every transaction makes changes to the shared memory. After its completion transactions may commit with its changes permanent and visible to others or may abort with its effect appear not to have taken place at all.

2.5.1 Types of Transactional Memory

Transactional memory is basically categorized in to two types:

1. Software Transactional Memory(STM)

   It is the transactional programming of synchronization operations in software. It provides transaction memory in a programming language. This TM has software programs to support transaction.

2. Hardware Transactional Memory(HTM)

   It is the transactional programming of synchronization operations in Hardware. This TM may have a modification in cache, bus or processors to support transaction.

Today even the hybrid transactional memory combining the STM and HTM are in research.

2.5.1.1 Non-Blocking Transactional Memory

Transactional memory can be blocking or non-blocking depending upon the use of locks. As the main purpose of the TM is to avoid locks so the non-blocking TM is very useful. Non-blocking TM

should have following feature:

- Lock Free.

  TM is lock free if, at least one of the threads makes progress while running sufficiently long.

- Wait Free.

  TM is Wait free if each process completes an operation after taking finite number of steps. Wait free guarantees that all non-halted processes makes progress.

- Obstruction Free.

  TM is obstruction free if at any point, a single thread executed in isolation.

Chapter 3

Literature Review

3.1 Bitcoin and Other Cryptocurrencies

Bitcoin[Nak09] is a decentralized digital cryptocurrency. However, it is not the first cryptocurrency. The first digital cryptocurrency was proposed by David Chaum in his paper[Cha83]in 1981. Bitcoin is also not the first decentralized digital currency. Wei Dai proposed B-money[Dai18] earlier than Bitcoin as a distributed cash system. Finally, in 2008 , Satoshi Nakamoto proposed Bitcoin with blockchain technology for decentralized consensus protocol in his paper[Nak09] with an old idea with new technology. The consensus in bitcoin is Proof Of Work(PoW).The original idea of Proof of Work was proposed by Dwork et al in their paper[DN] and later Hashcash[Bac02a][Bac02b] by Adam Back. Blockchain is the greatest innovation invented by Nakamoto through the digital cryptocurrency Bitcoin. This blockchain technology used in Bitcoin made it the first digital currency to solve the double spending[Cho17] problem without the need of a centralized server. Blockchain is an ingenious invention, not because it does anything new or performs any new magic tricks but it introduced a mechanism that guarantees to record the history of events in an untrusted network without any source of central authority. Blockchain used the Hashcash algorithm for adding blocks to the chain resulting in forming blockchain. This blockchain technology has been an inspiration to several applications and is under extensive research to identify its uses in other applications apart from cryptocurrencies. After Bitcoin was proposed by Nakamoto with blockchain technology as the backbone of the system, numerous cryptocurrencies were created with different consensus algorithms. There are more than 1543 cryptocurrencies till date and growing[cry18]. According to market, Bitcoin is the largest blockchain network with the highest price per coin followed by Ethereum, Ripple,Bitcoin Cash and Litecoin[cry18]. In 2015 a blockchain based software platform

named Ethereum was proposed by Vitalik Buterin. This project was crowdfunded online. Originally Buterin proposed the need for a scripting language in bitcoin for application development, but he couldn't gain agreement[eth18]. Thus he came up with the new cryptocurrency platform name Ethereum with smart contract[2.3.2] as the scripting language in it.

Bitcoin in 2008 was proposed with the PoW consensus algorithm, and most of the following cryptocurrencies followed the PoW for solving computation puzzles. The major drawback of PoW is that it takes a large amount of energy. As PoW requires an enormous amount of electricity for solving the puzzle, several other alternatives for PoW was coined. In 2012, Sunny King and Scott Nadal proposed the alternatives of PoW as Proof of Stake[kN]. The main aim of this proposal is to eliminate the high consumption of energy by PoW. Proof of stake was proposed for energy efficiency as this protocol is not dependent on energy consumption in the long run but is based on coin age. Peercoin, blackcoin, Nxt were the cryptocurrencies implementing the proof of stake. Later in 2014, Andrew Poelstra wrote a paper[Poe14], which clearly presented that distributed consensus from proof of stake is impossible. In his paper, he demonstrated that proof of stake doesn't work as a replacement of PoW. Bitcoin continues to gather success after its proposal as it is able to eliminate attacks and reach consensus without a third party, but it is also never free from some criticism. One of the concerns on PoW is that in the long run, new Bitcoin generation will stop and mining is only incentivized by the transactional fees[CKN16]. This paper explains that mining rewards only through transaction fees exacerbates instability in the future. So to sustain health in the mining process, a cryptocurrency protocol called proof of activity(PoA) was proposed in the paper[BLM]. This protocol was also criticized the same as both proof of work and proof of stake that too much energy is required and double signing cannot be validated.

Several other consensus protocols were proposed like Proof of Burn, Proof of capacity, proof of elapsed time. The FruitChain consensus protocol was also proposed in [PS17] as a fair blockchain.However, none of these protocols could replace PoW completely. So PoW is still in use in most of the cryptocurrencies for mining process, and double spent is properly handled in bitcoin.

3.2 Pooled Mining

Security is one of the major problems seen in pool mining as it is governed by a single pool operator. Security of pool mining has been analyzed in several previous works[Ros11][GoKC15]. Mining pool

can even hold the block to make it more profitable[CB14]. The threat of transaction censorship in centralized pool mining and the getblock template protocol to overcome this censorship issue were studied in [But] [get18]. Pooled mining[2.2] helps in reducing variance. However, pool mining degrades the concept of decentralization. Also, a centralized mining pool possesses the risk of a single point of failure. So overall the concept of distributed environment is weakened by the mining pool. Currently, 95% of Bitcoin power comes from only 10 mining pools, and 80% of the mining power in ethereum comes from 6 pools[LVJS] . Moreover, when more than half of a network mining power is controlled by a single pool operator, then 51% attack threatens the security of the Nakamoto consensus protocol [Nak09]. Thus to overcome the drawbacks, The bitcoin community proposed the concept of distributed mining named p2pool[p2p].

P2Pool is a decentralized pool mining without a central pool operator. The pool miners can select their own transaction to create the candidate block without the centralization of a pool operator. P2Pool is the first decentralized mining pool. However, it has not gained much more popularity because p2pool is inefficient. The number of messages exchanged between miners is a scalar multiple of the number of shares in the pool[LVJS]. When the share difficulty is low, the number of shares found are high which will increase the message transmitting cost and resources. To reduce the transmitting cost and resources, share difficulty can be increased. However this makes the variance of miners high. Though P2Pool is a decentralized mining pool and reduces the payout variance, its internal operational network remains open to infiltration by attackers[LVJS].

In order to remove the drawbacks of p2pool mining, Loi Luu et al. proposed a new decentralized pool mining called SmartPool[LVJS]. SmartPool leverages the smart contracts [2.3.2]. Smart contract is a self-executing script which is trackable and irreversible. It was proposed in 1994 by Nick Szabo in [sma94], and it came in to use with blockchain in ethereum. Ethereum is the first cryptocurrency to use smart contract. Several applications are proposed on top of smart contracts. Juels et al. in [JkS] studied the use of smart contract by criminals to support criminal activities. Ramachandran et.al in [RK] explains using smart contracts for secure data provenance management. Loi Luu et. al in [VJL][LCO] studied how smart pool makes bitcoin more venerable and how to make smart contract smarter. [LVJS] proposed a new application on top of smart contract by Luu et al. to enhance security and decentralized mining pools by giving the selection of transactions back to miners. In this Thesis we propose concurrency on top of smartPool so that

verification of shares from the pool is verified quickly and efficiently.

3.3 Parallel Computing

At the time of a market dominated by sequential computing, in 2005 Intel produced history's first parallel computer as the Intel Dual-core processor[dua05]. The Dual core processor has many identical cores in a single processor. This shift to a multicore processor gave immediate rise to the focus on multicore programming solutions[Har02] so as to properly utilize the multicore processor. Experts were focused on programmability rather than performance at the beginning, but the writing of parallel programs utilizing the multicore machines was not that easy. Primitives like locks, semaphores and monitors synchronization mechanism provide a way to coordinate and synchronize threads but they were cumbersome and error-prone.

3.4 Transactional Memory

In 1977, atomic operations in programming was proposed by Lomet[Lom77]with the idea of transactions that exists in databases. Later Knight proposed a way to improve Lisp improving Lomet work[Kni86]. However, it was in 1993 when Herlihy et al. first proposed and showed how to implement transactions in hardware as a means to handle concurrency[HM]. He practically showed how transaction memory simplifies concurrency in multicore machines. TM not only simplifies parallel programming but also solves the thread coordination and synchronization problems in an efficient way. Shavit and Touitou proposed software only transaction memory in 1995 which raised interest in transactional memory dramatically[ST]. Since then, several mechanisms for transactional memory have been proposed like polymorphic contention management[GHP05], lock based blocking system[Enn06][DS06], and composable transactional memory[HMJH05]. Today computers with a multicore parallel architecture are common, and parallel programmability in such computers is a must to enhance efficiency. Transactional memory provides an easy to use parallel programming solution. Transactional memory can be lock based or lockless. Robert Ennals made the first practical use of lock-based STM in 2005[Enn06]. Later Transactional Locker [DSS06][DS06][DS07] and DracoSTM[GC07] systems were proposed. To date, research in open and closed nesting transactional composition are primal in TM.

3.5 Concurrency to Smart Contract

The goal of this thesis is to propose concurrency in a Blockchain based SmartPool. Concurrency is gaining momentum as the approach of choice for replacing sequential execution. With the increase of data, the need of concurrency is growing as well. In today's world, the computation problem is getting more and more complex and if the problem is executed in a serial manner, it is supposed to take years to complete the task, so people are moving to concurrency which could help in completing the same task in less time. In the field of cryptocurrency, Ethereum and Bitcoin are very much in the news. Cryptocurrency is the buzzword in today's world with Bitcoin and Ethereum the most popular cryptocurrencies. Ethereum particularly uses scripts called smart contracts, and several applications are built on top of smart contract and Ethereum.This smart contract manages states, checks credentials and more. These smart contracts throughput is limited if they are executed by transactions in a serial fashion. In the case of Ethereum and smart contract, currently smart contract is executed by miners serially. This drawback of throughput was addressed in a paper [DGHK17] by Herlihy et al. This paper proposes executing the contract code concurrently using Software Transaction Memory(STM). Maurice is the first person to propose an implementation of transaction memory in the smart contract. Earlier several researches were carried out investigating TM methodology for highly concurrent data objects. A transactional support mechanism for non-blocking synchronization was originally proposed by Maurice Herlihy and Moss[HM] which proposed to implement transactional memory by multiprocessor cache coherence protocols. This was based on Hardware transactional memory. Stone et al [SS] proposed a similar concept. Herlihy also proposed implementing concurrent data objects using CAS in [Her93], LL/SC in [IR] , and transactional boosting in [HK]. Later Shavit and Touitou proposed the transactional memory implementation in software in their paper named Software transactional memory [ST]. This paper explained that each transaction should acquire ownership before it makes any change in the shares resources. Updates are only made after a system wide declaration of an update. This process only worked if the shared memory words are known in advance. Scott and Maratha proposed a STM for dynamic objects in [MS04] which even works for dynamic objects without prior knowledge of the shared memory. It can make decisions on the fly about the memory words to be accessed. Fatourou has explained several blocking and non-blocking algorithms in Algorithm techniques in STM design[FIkK15].

Chapter 4

Proposed Solution

4.1 Concurrency in SmartPool

In the SmartPool, shares are created then submitted to the smart contract. The shares are then sampled and verified. All this verification is done in a sequential manner, which degrades the throughput of the SmartPool. This sequential manner of running verifications in SmartPool can be replaced with concurrency using Transaction Memory(TM) for shared memory. Maurice Herlihy in his paper[DGHK17] explains about the implementation of concurrency in smart contract using TM and SmartPool is implemented in smart contract. Thus TM can be implemented in SmartPool for concurrency.

In sequential claim verification , the SmartPool receives the claim in batches, then verifies in a probabilistic verification. During probabilistic verification, some of the samples are taken and verified in a sequential manner. In this proposed algorithm, to support the concurrency, a count variable is added to the smart contract, which counts the number of valid samples. Finally, when all the threads are completed, the total count is compared. The total count is supposed to be the total number of threads spawned which signifies that all the samples are valid. Thus the claim can be added to the verified list.

4.2 Overview of Algorithm

In this proposed algorithm, we have proposed concurrency in SmartPool. The proposed Algorithm contains the following state:

- ClaimList.

  Shares submitted by miners are first stored in a list which is the ClaimList. This list is not the final verified list but is the temporary list of shares which are identified by miners. These Shares are later verified.

- VerClaimList.

  This is the list of shares that are verified by SmartPool. Once the share in ClaimList is verified, the shares in ClaimList is then transferred to VerClaimList.

- MaxCounter.

  This is the value to identify duplicate submission of the same share in different batches. As we know from earlier augmented merkle trees, each share has a minimum and a maximum field in it. Minimum and maximum field is related to the count. Each recently found share has a greater value of the count than an searlier found share. Maxcounter in SmartPool will store the last count value of shares received by SmartPool. Whenever the Pool receives new shares, the new shares are supposed to have a greater count value, and this verification of the recently received count value is checked through the MaxCounter variable. SmartPool will drop the shares as invalid if the received share is less than MaxCounter.

- difficulty.

  This is the target of shares for every miner. A share that is found by the miners must have a hash value greater than or equal to the difficulty to prove itself as valid.

- Counter.

  The counter value is used for updating the number of valid shares. This variable is added to the proposed solution so that the threads can be used for concurrency, and this variable is continuously updated by each thread.

The miners submit the shares in the form of a claim. Instead of sending all the shares by miners one by one, miners send the shares in the form of batches in a single claim. Once the SmartPool receives the batches of shares, shares are stored in ClaimList initially, then the shares from the ClaimList are verified. The submission of claim and verification is done through probabilistic verification. Only some of the samples are verified to increase efficiency and security. The verification of only a few shares increases efficiency. However, there is a risk of fraud such that the same shares are submitted multiple times. In order to address this issue, an augmented merkle Tree structure is

used so that same share is not submitted multiple times with a sorting error identified if any miners try to submit multiple shares. Furthermore, MaxCounter is tracked by the SmartPool so that the current received batch should always have the counter value greater than the latest received batch.

Once the batches of shares are received, it is already ensured that the shares submitted have no duplicate shares, otherwise, it could have been already dropped and that miners would never get involved in mining again. So from the miner point of view, first the valid shares are identified, shares are then batched into a bundle of shares, and an Augmented Merkle tree is created with the appropriate data structure. Once the Merkle Tree shows no sorting error, shares are then submitted to SmartPool as a ShareAugMT.

At the SmartPool end, only a few shares are picked and the SmartPool asks the miners to send ShareProof to demonstrate that the share has been included in the ShareAugMT. The SmartPool then verifies each claim. This verification of shares in SmartPool is done concurrently using Transactional memory in a non-blocking way. The Count state is used to track that the total number of verified shares is equal to the total number of random samples. If the count variable is equal to the total number of random samples, it ensures that all the shares are valid and all the shares are submitted to VerClaimList. Otherwise, the submitted claim is dropped and the miners won't get any rewards.

In the implementation of TM, each random sample is executed in a thread and the counter is updated by the thread. The counter variable is a shared memory and can be accessed by all the threads. Allowing all the threads to access a shared memory is a complex task. All the threads may read, modify, write to the memory at the same time, which causes the share memory to have an unwanted value written in it. It is likely that different threads may access the memory at the same time which could lead to the data inconsistency and race condition. Thus to update the count variable atomically, TM is used.

So in the overall picture, every miner in the pool is continuously working to find the valid hash. The difficulty of the hash is provided by the SmartPool to the miners, and with the information of the difficulty, each miner works to get the share with a hash less than the difficulty as provided by the SmartPool. Once the miners in the SmartPool find the valid hash, then they submit the

shares in batches with the augmented merkel root. Augmented Merkel Trees have Minimum and Maximum filed in it to verify that the same shares won't be submitted in a single batch. Moreover, SmartPool keeps track of the last submitted shares timestamp so that the newly submitted shares must have a timestamp greater than the latest received share batches. This prevents the miners from submitting the same duplicate share in the new batch. These shares with valid timestamps are then submitted in batches to SmartPool. SmartPool stores these shares in ClaimList. These shares in ClaimList are then verified probabilistically by taking some samples from it. This samples are then verified using TM with the count share variable. This TM will update the counter variable atomically and gives the correct output. Once all the threads complete the transaction, the counter value is compared with the number of random samples. If the number of random sample is equal to the counter value, this suggests that the submitted batch of shares is valid. After then the shares are submitted to VerClaimList. SmartPool also checks whether the submitted random shares is valid block of the network or not. There is a probability that the share can have a difficulty target equal to the network target. In such a case, the share is submitted to the network and the reward is distributed to all the miners as per the VerClaimList.

Algorithm 3: Proposed Algorithm for concurrency in SmartPool

```
  Structure of Claim      : Nsize,ShareAugMt
  Structure of ShareProof: HeaderofShareS_i, AugMkProof
1 Nsize←Number of shares in a claim
2 Nsample←Number of random samples
3 VerClaimList[X]←All verified claims submitted by miner X
4 ClaimList[X]← All unverified claim submitted by miner X
5 MaxCounter[X]← Max counter of the miner X
6 difficulty← minimum difficulty of the share
7 Counter=0/* initialize counter                                   */
8
9 Function Algorithm_for_Executing_in_SmartPool::
      /* Main execution in Smartpool with concurrency              */
10    Accept a Claim
11    for eachclaim do
12        Verify Share counter>maxcounter[X]
13        Verify duplicate submission
14        Create thread with Nsample number
15        for i ← 0 to Nsample do
16            Request ShareProof from Miner
17            Verify ShareProof
18            Verify Minimum Difficulty
19            Update Counter using TM
20        end
21        if Counter == Nsample/* verify that all samples are valid  */
22         then
23            Update VerClaimList[X];
24        end
25        else
26            Reject Claim
27        end
28        if Valid Block then
29            Request Payment
30        end
31    end
```

```
32 Function Algorithm_for_Miners::
       /* execution in Miners                                   */
33     Construct Block Template;
34     Find Valid Shares;
35     Submit Claim;
36     Submit ShareProof;
```

4.3 Tools and Techniques

The concurrent verification of shares can be done with multiple threads. In order to verify that all the shares are verified, a Count variable is added which is updated by each thread on verifying shares. The problem with this count variable is that since it is a shared variable , it may be accessed by more than 1 thread at the same time and the overall count value is incorrectly updated. Thus to make sure that the count variable is updated by a single thread at a time, a lock can be used. However, a lock is a blocking thread mechanism which imposes several problems. The non-blocking way to update a shared variable is by means of TM.

In the implementation of TM, the shared variable which is a counter has a local copy of read set and write set for each thread. There is a global copy of the shared data as well which is accessible to all threads. First the global copy of the shared data is copied to the local read set. Then some operation is performed locally and the value of the data is updated locally in the local write. Before updating the local write to the shared variable by a thread , the local old read copy is compared with the global data to ensure that any other threads haven't changed the content of shared data. If the local and global data are the same for the particular thread, this ensures that no other threads have updated the value, otherwise some other thread must have already changed the value. If any other thread has already changed the shared data, then the global data is copied to the local read, the thread is rolled back to an initial state and the procedure is repeated. The value of the thread write is only committed to the global data if none of the other threads has made changes before. This comparison of local copies with the shared variable and updating can be done through atomic primitives CAS and LL/SC.Thus TM in the counter variable can be implemented by using operations like Compare and Swap(CAS) and Load-Linked/Store-Conditional(LL/SC)[Her93].

4.3.1 Atomicity in Counter with CAS

The compare and swap(CAS) when called as CAS(A,old,new) returns true and updates the value of address A with a new value if old is equal to address value A. Otherwise it returns False and the address value A is copied to the old value. Section 1.2.4.1 gives more detail on CAS.

In our algorithm of concurrency, CAS can be used to update the counter atomically. Each thread is supposed to update the counter value if the shares are valid. For this purpose of updating the counter atomically, each thread will keep a copy of the shared variable counter locally. When the shares are identified valid by a thread, it increases its local count value by 1 and checks if the shared global count variable is already updated by some other threads or not. It checks its earlier local shared count value with the value of the global count. If the earlier local copy of the share value is the same to the global count, it implies none of the threads have updated the counter before it, so it updates the counter. This way of updating the counter atomically can be implemented by CAS.

Algorithm 4: Update Counter using CAS

Data: Counter =0

1 **for** $i \leftarrow$ **to** $Nthreads$ **do**

2   **if** $Sample == Valid$ **then**

3         $CAS(\&Counter, Counter, Counter + 1);$

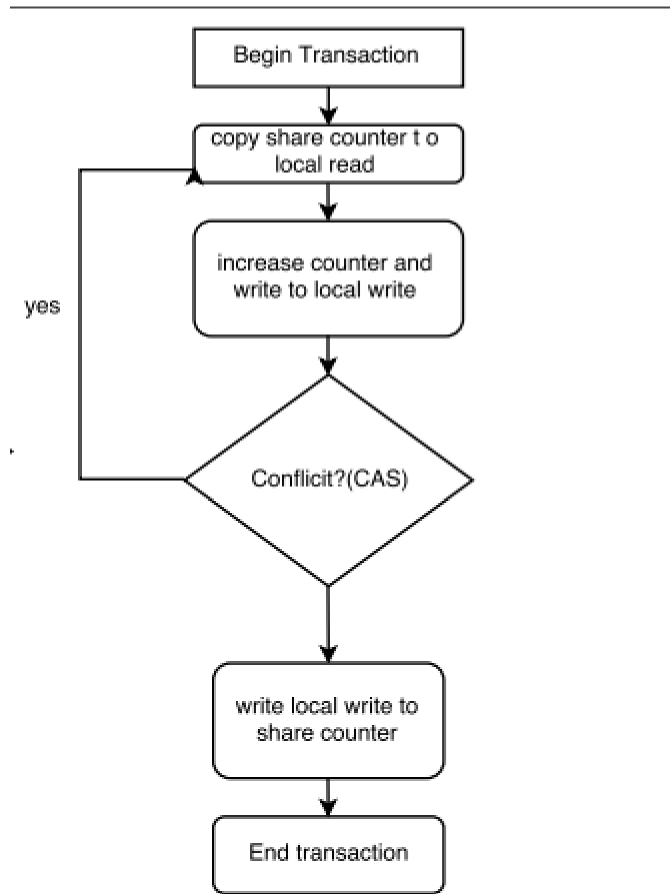

Figure 4.1: CAS flowchart

4.3.2 Atomicity in Counter with LL/SC

Atomicity can also be implemented by using LL/SC. Section[1.2.4] gives more detail on LL/SC. In our algorithm, LL/SC can address our atomicity in the counter. Since the counter needs to be updated atomically without blocking any other threads, LL/SC is a non-blocking primitive for atomicity. Each thread will load the count variable locally and once the share is identified as valid, the local count variable is updated. This local count variable is updated in the global count only if another variable hasn't updated the global count before it.

Algorithm 5: Update Counter using LL/SC

Data: Counter =0

```
1 for i ← to Nthreads do
2     if Sample == Valid then
3         LL(&Counter);
4         SC(&Counter,Counter+1)
```

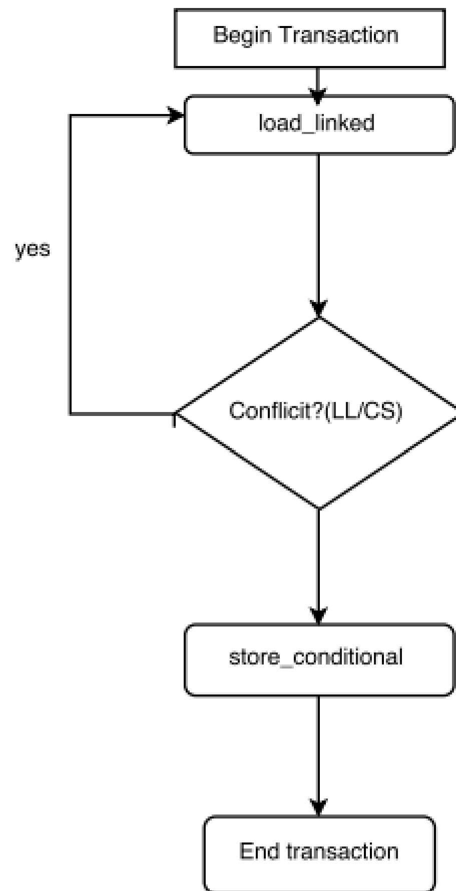

Figure 4.2: LL/SC flowchart

4.3.3 STM for Concurrency for Counter

Transaction memory is the buzzword for the concurrency in shared memory. Several STM techniques have been purposed for concurrency apart from above. First the TM technique as proposed by Herlihy and Moss[HM] can be used for the concurrency control here in smart contracts as well.

Herlihy's proposed TM is a simple extension to the multiprocessor cache coherence protocol. Like the way consistency in cache and main memory is achieved, the same coherence protocol can be implemented for achieving concurrency in smart contacts as well.

STM proposed by Shavit and Touitou requires the pre-knowledge of all the memory that the transaction accesses. Since this thesis proposal has just the single count variable which is static, thus Shavit and Touitou [ST] proposed STM can be implemented for concurrency in SmartPool. This algorithm makes updates to the count variable only after a system wide declaration of its update intention. Such a declaration makes other transactions aware that some other transaction is about to make updates to the particular shared object. This algorithm uses the concept of ownership. The transaction declares that it is taking ownership of the shared variable and declares itself as the owner. The declaration of ownership is done through storing of references by transaction. After taking ownership, the transaction makes updates to the object, then releases its ownership. This process of acquiring and releasing of ownership is done atomically using CAS or LL/CS.

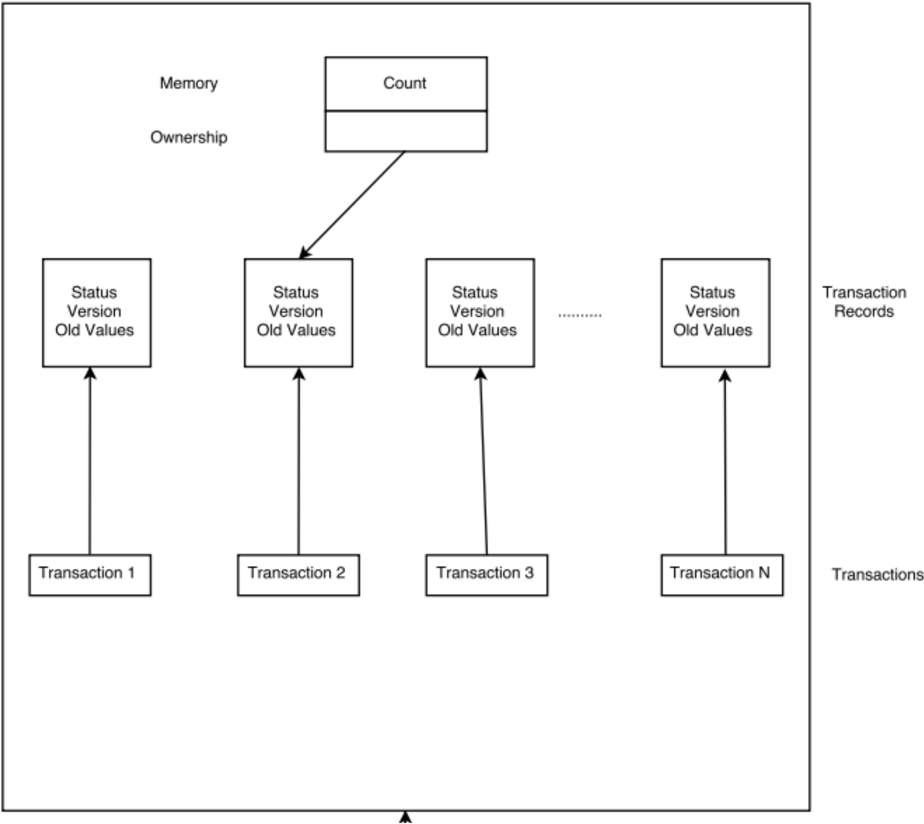

Figure 4.3: counter with STM

4.3.3.1 STM Algorithm with Ownership for Atomic Count

Algorithm 6: Start Transaction

Status: None,Success,Failure

```
1 Start_Transaction(Dataset)::
2     Initialize(trans_i,Dataset);
3     Transaction(trans_i,trans_i.version,True);
4     Trans_i.executing = False;
5     Trans_i.version++;
6     if Trans_i.success == Success then
7         return Success;
8     else
9         return Failure;
```

Algorithm 7: Transaction

```
1  transaction(Trans,version,Isinitiator):
2      AcquireOwnership(trans,version);
3      status = LL(trans.status);
4      if status == None then
5          if version != trans.version then
6              return
7          SC(tran.status,(success,0));
8      status = LL(tran.status);
9      if status == Success then
10         RecordOldValues(tran,version);
11         newvalues = calcnewvalues(stat.oldvalues);
12         updatememory(stat,version,newvalues);
13         releaseownership(tran,version);
14     else
15         ReleaseOwnership(tran,version);
16         if Isinitiator then
17             ConflicitTrans = Ownership[ConflicitAddr];
18             if ConflicitTrans == Null then
19                 return
20             else
21                 ConflicitVersion = ConflicitTrans.version;
22                 if ConflicitTrans.executing then
23                     Transaction(Conflicittrans,ConflicitVersion,False);
```

Algorithm 8: Ownership

```
1  AcquireOwnership(trans,version):
2      location = trans.add[count];
3      status = LL(trans.status);
4      if status != None then
5          return
6      Owner = LL(Ownership[trans.add[count]]);
7      if trans.version != version then
8          return
9      if Owner == trans then
10         break
11     if Owner == null then
12         if SC(trans.status,(null,0)) then
13             if SC(Ownership[location],trans) then
14                 break
15     else
16         if SC(trans.status,(Failure)) then
17             return
18 ReleaseOwnership(trans,version):
19     location = trans.add[count];
20     if LL(ownership[location]==trans then
21         if trans.version != version then
22             return
23         SC(ownership[location],null);
```

Algorithm 9: Memory Access

```
1  RecordOldValue(trans,version):
2      location = trans.add[count];
3      oldvalue = LL(trans.oldvvalue[location]);
4      if trans.version != version then
5          return
6      SC(trans.oldvalue[location],memory[location]);
7  UpdateMemory(trans,version,newvalues):
8      location = trans.add[count];
9      ; oldvalue = LL(memory[location]);
10     if trans.version != newvalues[count] then
11         return
12     if oldvalue != newvalues[count] then
13         SC(memory[location,newvalues);
```

4.3.3.2 Overview of STM Algorithm with ownership for atomic count

The above STM Algorithm with ownership for the atomic count in section[4.3.3.1] is a modification of Shavit and Touitou's paper[ST] with single shared memory "Count". This algorithm requires all shared memory locations to be known in advance. Knowledge of shared memory in advance allows us to acquire ownership of the count shared variable. Once the transaction acquires ownership, it can modify the shared memory, then it releases ownership for other threads. Each transaction records the necessary data structure that is required to acquire the ownership and execute it. The data structure of each transaction follows:

```
memory address
oldvalue
status
version
bool executing
```

memory address holds the address of the memory locations accessed by the transaction. *old*

values records the content of the memory when the transaction acquired ownership. *status,version and executing* are used to track the state of the transaction.The shared memory word "Count "has a corresponding ownership that tracks which transaction owns the shared memory location. After acquiring ownership , the transaction tries to update the memory. However, before updating the memory, the transaction first loads the status, checks the version and then only updates if the version of the transaction is the same. The atomicity is ensured by the store-conditional operation.

```
LL(trans.status)
........
If(version!=trans.version)
return
SC(trans.status,newval
```

The Start_Transaction routine Algorithm [6]starts the transaction. The Dataset parameter has the shared memory address that can be accessed by the transaction. The transactional object is created and the Transaction routine[7] is called to acquire ownership and execute the transaction.

The routine AcquireOwnership and ReleaseOwnership Algorithm[8] is called by the Transaction routine . The AcquireOwnership routine acquires ownership of the memory if the shared memory location is not acquired by any other transaction. ReleaseOwnership releases the ownership of the shared memory location.

The RecordOldValue routine Algorithm[9] records the value of the shared memory. It is only invoked if the shared memory is acquired by the transaction.

4.4 Correctness

Lemma 4.4.1 *Validation of random shares in SmartPool is Lock Free.*

Random shares are validated using concurrency with multiple threads in it. Threads are coordinated to update the count shared memory only if the shares are valid which is checked with a difficulty verification and if shares are linked to earlier transactions or not. For the coordination of threads, locks are not used but shares in each thread are validated as a transaction. The transaction updates the count for sure or will start from the beginning if it fails, thus validation of random shares is Lock free.

Lemma 4.4.2 *Validation of random shares in SmartPool is Wait free.*

All the threads in the verification process will complete their task of updating the count variable in a finite time. If the share is valid then the particular thread running that process will update the count value, or if the share is invalid, it will end the particular thread. In case the share is valid but cannot update the count variable as it is currently being used by another, the current transaction is dropped and again starts from the beginning. Any thread will never run for infinite time , Thus validation of random shares in SmartPool in Wait free.

Lemma 4.4.3 *Validation of random shares in SmartPool is Obstruction free*

The validation of shares in SmartPool is obstruction free as all the threads executing shares are executed in isolation. Any process can complete the assigned task even if all the other processes are stopped. Each thread carrying the validation of shares as a task completes the assigned task in finite time without obstruction in isolation. Threads in SmartPool which have partially completed tasks are rolled back.

Lemma 4.4.4 *Validation of random shares in SmartPool is Non-blocking.*

A non-blocking algorithm must follow the properties of being lock free, wait free and obstruction free. Since the random share validation follows all three properties of lock free, wait free and obstruction free, thus validation of random shares is Non-Blocking.

Lemma 4.4.5 *Validation of random shares in SmartPool is Serializable.*

Transactions are said to be serializable when one transaction never appears interleaved with others. In this proposed algorithm of SmartPool, each thread is lock free, obstruction free with executing the task without obstructing the operation of another thread. All the threads execute in a manner that is serializable. None of the threads in the verification of shares have to wait for other threads to execute the task. Thus validation of random shares is serializable.

Lemma 4.4.6 *Validation of random shares in SmartPool is Atomic.*

A transaction is said to be atomic if the changes are likely to commit permanently and visible to others or abort with its affect appearing not to have taken place at all. Here in implementation of validation of shares, the count variable is modified by all the threads. However they never execute at the same time. Whenever a thread had already made an update to the global count, the count variable is rolled back and starts from the beginning. The update to the global count is only

possible if none of the threads have already made changes to the common global count variable. Thus this ensures that the count variable is updated permanently with its effect visible to everyone or it aborts the task to its original state and starts from the beginning.

Lemma 4.4.7 *Validation of random shares in SmartPool is Transactional memory.*

Since the validation of random shares in SmartPool is serializable and atomic, so it is transactional memory.

Chapter 5

Conclusion and Future work

We have proposed concurrency in blockchain based SmartPool using Transactional memory. Earlier proposed SmartPool is decentralized pool mining in which uses smart contract however the throughput could be more enhanced using concurrency. Thus we proposed concurrency using Transactional Memory. CAS,LL/SC and STM techniques can be used to increased throughput in SmartPool. This SmartPool can be used in the bitcoin as well. Thus this technique of concurrency is applicable in both ethereum and bitcoin.

We proposed algorithm for concurrency in SmartPool theoretically. Practically implementing SmartPool and implementing CAS,LL/SC and STM and comparing this techniques for concurrency can be interesting future work. It could be interesting to run the concurrency technique in different cores machines and benchmarking the output

Bibliography

[Bac] Adam Back. A partial hash collision based postage scheme,http://www.hashcash.org/papers/announce.txt.

[Bac02a] Adam Back. Hashcash - a denial of service counter-measure. 2002.

[Bac02b] Adam Back. Hashcash - amortizable publicly auditable cost-functions. 2002.

[BLM] Iddo Bentov, Charles Lee, and Alex Mizrahi. Proof of activity: Extending bitcoin's proof of work via proof of stake.

[But] Vitalik Buterin. The problem of censorship,https://blog.ethereum.org/2015/06/06/the-problem-of-censorship/.

[But13] Vitalik Buterin. A next generation smart contract & decentralized application platform. 2013.

[CB14] Nicolas T. Courtois and Lear Bahack. On subversive miner strategies and block withholding attack in bitcoin digital currency. 2014.

[Cha83] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. 1983.

[Cho17] Usman W. Chohan. The double-spending problem and cryptocurrencies. 2017.

[CKN16] Miles Carlsten, Harry Kalodner, and Arvind Narayanan. On the instability of bitcoin without the block reward. 2016.

[cry18] Cryptocurrency market capitalizations,https://coinmarketcap.com/all/views/all/, March 2018.

[Dai18] Wei Dai. B-money,http://www.weidai.com/bmoney.txt, 1918.

[DGHK17] Thomas Dickerson, Paul Gazzillo, Maurice Herlihy, and Eric Koskinen. Adding concurrency to smart contracts. 2017.

[DN] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail.

[dou] Double spend in digital currency.

[DS06] David Dice and Nir Shavit. What really makes transactions faster? *ACM SIGPLAN Workshop on Transactional Computing*, 2006.

[DS07] David Dice and Nir Shavit. Understanding tradeoffs in software transactional memory. Technical report, 2007.

[DSS06] David Dice, Ori Shalev, and Nir Shavit. Transactional locking ii. Technical report, 2006.

[dua05] Dual core era begins, pc makers start selling intel-based pcs: Intel dual-core processor-powered pc systems first to market. Technical report, Intel Corporation,, 2005.

[Enn06] Robert Ennals. Software transactional memory should not be obstruction-free. Technical report, Intel Research Tech Report, 2006.

[eth18] Ethereum,https://en.wikipedia.org/wiki/ethereum, 2018.

[FIkK15] Panagiota Fatourou, Mykhailo Iaremko, Eleni kanellou, and Eleftherios Kosmas. Algorithmic techniques in stm design. 2015.

[GC07] Justin Gottschlich and Daniel A. Connors. Dracostm: A practical c++ approach to software transactional memory. 2007.

[get18] Bitcoin wiki,https://en.bitcoin.it/wiki/getblocktemplate, 2018.

[GHP05] Guerraoui, Maurice Herlihy, and Pochan. Polymorphic contention management. *DISC: International Symposium on Distributed Computing*, 2005.

[GoKC15] Arthur Gervais, Ghassan o. Karame, and Vedran Capkun. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. 2015.

[Har02] Tim Harris. Concurrent programming for dummies (and smart people too). 2002.

[Her93] Maurice Herlihy. A methodology for implementing highly concurrent data structure. pages 197–206, 1993.

[HK] Maurice Herlihy and Eric Koskinen. Transactional boosting: A methodology for highly concurrent transactional objects.

[HM] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structure.

[HMJH05] Tim Harris, Simon Marlow, Simon L. Peyton Jones, and Maurice Herlihy. Composable memory transactions. *ACM*, 2005.

[IR] Amos Israeli and Lihu Rappoport. Disjoint-access-parallel implementation of strong shared memory.

[JkS] Ari Juels, Ahmed kosba, and Elaine Shi. The ring of gyges: Investigating the future of criminal smart contracts.

[kN] Sunny king and Scott Nadal. Ppcoin:peer-to-peer crypto-currency with proof-of-stake.

[Kni86] Thomas F. Knight. An architecture for mostly functional languages. *Proceedings of the 1986 ACM conference on LISP and functional programming*, 1986.

[LCO] Loi Luu, Duc-Hiep Chu, and Hrishi Olickel. Making smart contracts smarter.

[Lom77] D.B. Lomet. Process structuring, synchronization, and recovery using atomic actions. *In Proceedings of an ACM conference on Language design for reliable software*, pages 128–137, 1977.

[LVJS] Loi Luu, YAron Velner, JasonTeutsch, and Prateek Saxena. Smartpool:practical decentralized pooled mining.

[Mer] Ralph Merkle. Method of providing digital signatures. *1979*.

[mom14] How a mining monopoly can attack bitcoin,http://hackingdistributed.com/2014/06/16/how-a-mining-monopoly-can-attack-bitcoin/, 2014.

[MS04] Virendra J. Marathe and Michael L. Scott. A qualitative survey of modern software transactional memory system. 2004.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009.

[p2p] P2pool:decentralized pool mining,http://p2pool.org/.

[Poe14] Andrew Poelstra. On stake and consensus. 2014.

[PS17] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. 2017.

[RK] Aravind Ramachandran and Dr Murat Kantarcioglu. Using blockchain and smart contracts for secure data provenance management.

[Ros11] Meni Rosenfeld. Analysis of bitcoin pooled mining reward systems. 2011.

[sma94] Smart contracts: Building blocks for digital markets, 1994.

[SS] J. M. Stone and H. S. Stone. Multiple reservations and the oklahoma update.

[ST] Nir Shavit and Dan Touitou. Software transaction memory.

[VJL] YAron Velner, JasonTeutsch, and Loi Luu. Smart contracts make bitcoin mining pools vulnerable.

Curriculum Vitae

Graduate College

University of Nevada, Las Vegas

Laxmi Kadariya

Degrees:

Master of Science in Computer Science 2018

University of Nevada Las Vegas

Thesis Title: Concurrency in Blockchain Based SmartPool with Transaction Memory

Thesis Examination Committee:

Chairperson, Dr. Ajoy k. Datta, Ph.D.

Committee Member, Dr. Laxmi Gewali, Ph.D.

Committee Member, Dr. John Minor, Ph.D.

Graduate Faculty Representative, Dr. Emma E. Regentova, Ph.D.