Cryptocurrency Simulation

A project report submitted for the pre-final year of Bachelor of Technology in

Computer Science and Engineering

By

Laya Aluri	N150580
Naga Jyothi Lukka	N150687
Karthik Chennupati	N150675
Vamsi Krishna Padamuttam	N150649

Under the Guidance of

Mr. Ramarao,
Faculty, Dept of CSE,
RGUKT - Nuzvid

to

Department of Computer Science and Engineering,
Rajiv Gandhi University Of Knowledge Technologies,
Nuzvid, Krishna, Andhra Pradesh - 521202

Rajiv Gandhi University of Knowledge Technologies
RGUKT-Nuzvid, Krishna Dist - 521202

Certificate of completion

This is to certify that the project title entitled “Cryptocurrency Simulation” is a bonafide work by N150675, N150649, N150580, and N150687 submitted for E3 SEM - 2 mini projects to the Department of Computer Science and Engineering under my guidance in the academic year 2019-2020. In my opinion, this project is worthy of consideration for the awarding of marks in accordance with the department and university rules and regulations.

Date: July 2020
Place:

Head of the Department,
Mr. Kumar Anurupam,
Computer Science & Engineering,
RGUKT - Nuzvid.

Project Supervisor,
Mr. Ramarao
Computer Science & Engineering,
RGUKT - Nuzvid.

Rajiv Gandhi University of Knowledge Technologies
RGUKT-Nuzvid, Krishna Dist - 521202

Certificate of examination

This is to certify that the work entitled, “Cryptocurrency Simulation” is a bonafide work of N150675, N150649, N150580, N150687, and hereby accord our approval of it as a study carried out and presented in a manner required for its acceptance in the third year of Bachelor of Technology for which it has been submitted. This approval does not necessarily endorse or accept every statement made, opinion expressed, or conclusion drawn, as recorded in this thesis. It only signifies the acceptance of this thesis for the purpose for which it has been submitted.

Date: July 2020
Place:

Project Supervisor,
Mr. Ramarao,
Computer Science & Engineering,
RGUKT - Nuzvid.

Project Examiner

Rajiv Gandhi University of Knowledge Technologies
RGUKT-Nuzvid, Krishna Dist - 521202

Declaration

We hereby declare that this mini project entitled “Cryptocurrency Simulation” is a bonafide work of N150675, N150649, N150580, and N150687 submitted in partial fulfillment of the requirement for the award of marks in the pre-final year in the Bachelor of Technology in the Department of Computer Science and Engineering during the academic session December 2019 - April 2020 at RGUKT-Nuzvid.

We also declare that this project is a result of our own effort and has not been copied or imitated in any source. Citations from other sources are mentioned in the references. The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

Date: July 2020
Place:

Karthik Chennupati,
N150675.

Vamsi Krishna Padamuttam,
N150649.

Naga Jyothi Lukka,
N150687.

Laya Aluri,
N150580

Table of Contents

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions such as banks, serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust-based model. Completely non-reversible transactions are not really possible since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Cryptocurrencies are developed for the very same reason, and as of May 2018, there exist over 1,800 cryptocurrency specifications. Even with all the work being done in the field, it is very difficult to understand how cryptocurrencies actually work. We intend to develop a generic cryptocurrency model simulation, which makes it easy to understand how a decentralized cryptocurrency model works.

2. Description

Cryptocurrency is a digital currency in which encryption techniques are used to regulate the generation of units of currency and verify the transfer of funds, operating independently of a central bank. The main strength of cryptocurrencies comes from being decentralized. The decentralized nature of cryptocurrencies comes from the underlying paradigm, ***Blockchain.***
Blockchain was invented by a person (or group of people) using the name Satoshi Nakamoto in 2008 to serve as the public transaction ledger of the

cryptocurrency bitcoin. The identity of Satoshi Nakamoto remains unknown to date. The invention of the blockchain for bitcoin made it the first digital currency to solve the double-spending problem without the need for a trusted authority or central server.

A blockchain is a growing list of records, called blocks, that are linked using cryptographic techniques. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. By design, a blockchain is resistant to modification of the data. It is an open, decentralized ledger that can record transactions between two parties efficiently and in a verifiable and permanent manner.

A blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for inter-node communication and validating new blocks. Once recorded, the data in any given block cannot be altered retroactively without the alteration of all subsequent blocks, which requires consensus of the network majority.

This project is developed in an incremental process, dividing each phase into stages, and adding new requirements on every successive stage. This project leverages instructions from Jetbrains Academy's Blockchain project.

3. Stage 1: Blockchain essentials

Blockchain has a simple interpretation: it's just a chain of blocks. It represents a sequence of data that you can't break in the middle; you can only append new data at the end of it. All the blocks in the blockchain are chained together.

To be called a blockchain, every block must include the hash of the previous block. Other fields of the block are optional and can store various information. The hash of a block is a hash of all fields of a block. So, you can just create a string containing every element of a block and then get the hash of this string. Note that if you change one block in the middle, the hash of this block will also

change. and the next block in the chain would no longer contain the hash of the previous block. Therefore, it’s easy to check that the chain is invalid.

In the first stage, you need to implement such a blockchain. In addition to storing the hash of the previous block, every block should also have a unique identifier. The chain starts with a block whose id = 1. Also, every block should contain a timestamp representing the time the block was created. Since the first block doesn't have a previous one, its hash of the previous block should be 0.

The class Blockchain should have at least two methods: the first one generates a new block in the blockchain and the second one validates the blockchain and returns true if the blockchain is valid. Of course, the Blockchain should store all it's generated blocks. The validation function should validate all the blocks of this blockchain. Also, for hashing blocks, you need to choose a good cryptographic hash function which is impossible to reverse-engineer. Insecure hash functions allow hackers to change the information of the block so that that the hash of the block stays the same, so the hash function must be secure. A good example of a secure hash function is SHA-256.

4. Stage 2: A proof of work concept

The security of our blockchain is pretty low. You can't just change some information in the middle of a blockchain, because the hash of this block will also be changed. And the next block still keeps the old hash value of the previous block. But can't we replace the old hash value with the new hash value so everything will be ok? No, because when you change the value of the previous hash in the block, the hash of this block will also be changed! To fix this, you need to change the value of the previous hash in the block after it. To solve this problem, you need to fix hash values in all the blocks until the last block of the blockchain!

This seems to be a pretty hard task to execute, doesn’t it? If the time it takes to fix the hash value of the previous block is less than time to create a new block,

we suddenly would be fixing blocks faster than the system can create them, and eventually, we will fix them all. The problem is that fixing the hash values is easy to do. The blockchain becomes useless if it is possible to change the information in it.

The solution to this is called ***proof of work***. This means that . The time should depend on the amount of computational work put into it. This way, the hacker must have more computational resources than the rest of the computers of the system put together.

The main goal is that the hash of the block shouldn't be random. It should start with some amount of zeros. To achieve that, the block should contain an additional field: a magic number. Of course, this number should take part in calculating the hash of this block. With one magic number, and with another, the hashes would be totally different even though the other part of the block stays the same. But with the help of probability theory, we can say that there exist some magic numbers, with which the hash of the block starts with some number of zeros. The only way to find one of them is to make random guesses until we found one of them. For a computer, this means that the only way to find the solution is to brute force it: try 1, 2, 3, and so on. The better solution would be to brute force with random numbers, not with the increase from 1 to N where N is the solution.

Obviously, the more zeros you need at the start of the block hash, the harder this task will become. And finally, if the hacker wants to change some information in the middle of the blockchain, the hash of the modified block would be changed and it won't start with zeros, so the hacker would be forced to find another magic number to create a block with a hash which starts with zeros. Note that the hacker must find magic numbers for all of the blocks until the end of the blockchain, which seems like a pretty impossible task, considering that the blockchain will grow faster.

It's said that the block is proved if it has a hash which starts with some number of zeros. The information inside it is impossible to change even though the information itself is open and easy to edit in the text editor. The result of the edit is a changed hash of the block, no longer containing zeros at the start, so this

block suddenly becomes unproved after the edit. And since the blockchain must consist of only proved blocks, the whole blockchain becomes invalid. This is the power of the proof of work concept.

In this stage, you need to improve the blockchain. It should generate new blocks only with hashes that start with N zeros. The number N should be input from the keyboard. Also, the blockchain should be saved to the file after each block. At the start of the program, you should check if a blockchain exists on the hard drive, load it, check if it is valid, and then continue to create blocks. You may want to use serialization to do that.

5. Stage 3: Miner Mania

The blockchain itself shouldn't create new blocks. The blockchain just keeps the chain valid and accepts the new blocks from outside. In the outside world, there are a lot of computers that try to create a new block. All they do is search for a magic number to create a block whose hash starts with some zeros. The first computer to do so is a winner, the blockchain accepts this new block, and then all these computers try to find a magic number for the next block.

There is a special word for this: ***mining***. The process of mining blocks is hard work for computers, as the process of mining minerals in real-life is hard work. Computers that perform this task are called miners.

Note that if there are more miners, the new blocks will be mined faster. But the problem is that we want to create new blocks with a stable frequency. For this reason, the blockchain should regulate the number N: the number of zeros at the start of a hash of the new block. If suddenly there are so many miners that the new block is created in a matter of seconds, the complexity of the next block should be increased by increasing the number N. On the other hand, if there are so few miners that process of creating a new block takes longer than a minute, the number N should be lowered.

In this stage, you should create a lot of threads with miners, and every one of them should contain the same blockchain. The miners should mine new blocks and the blockchain should regulate the number N. The blockchain should check the validity of the incoming block (ensure that the previous hash equals the hash of the last block of the blockchain and the hash of this new block starts with N zeros). At the start, the number N equals 0 and should be increased by 1 / decreased by 1 / stays the same after the creation of the new block based on the time of its creation.

6. Stage 4: You've got a message

For now, we are mining blocks to create a blockchain, but just the blockchain itself is not particularly useful. The most useful information in the blockchain is the data that every block stores. The information can be anything. Let's create a simple chat based on the blockchain. If this blockchain works on the internet, it would be a world-wide chat. Everyone can add a line to this blockchain, but no one can edit it afterward. Every message would be visible to anyone.

In this stage, you need to upgrade the blockchain. A block should contain messages that the blockchain received during the creation of the previous block. When the block was created, all new messages should become a part of the new block, and all the miners should start to search for a magic number for this block. New messages, which were sent after this moment, shouldn't be included in this new block. Don't forget about thread synchronization as there is a lot of shared data.

You don't need any network connections as this is only a simulation of the blockchain. Use single blockchain and different clients that can send the message to the blockchain just invocating one method of the blockchain.

So, the algorithm of adding messages is the following:

1. The first block doesn't contain any messages. Miners should find the magic number of this block.

2. During the search of the current block, the users can send messages to the blockchain. The blockchain should keep them in a list until miners find a magic number and a new block would be created.
3. After the creation of the new block, all new messages that were sent during the creation should be included in a new block and deleted from the list.
4. After that, no more changes should be made to this block apart of the magic number. All new messages should be included in a list for the next block. The algorithm repeats from step 2.

7. Stage 5: Matters of security

How safe is your messaging system at the moment? Anyone can add a message to the blockchain. But can anyone impersonate you and send a message using your name? Without encryption, this is totally possible. There needs to be a method to verify that it is actually you who sent this message. Note that the registration/authorization method is bad because there is no server to check for a valid login/password pair. And if there is, it can be cracked by hackers who can steal your password. There needs to be a whole new level of security.

Asymmetric cryptography solves this problem. With this, you can sign the message and let the signature be a special part of the message. You can generate a pair of keys: a public key and a private key. The message should be signed with a private key. And anyone can verify that the message and the signature pair is valid using a public key. The private key should be only on your computer, so no one from the internet can steal it. If you think that someone can steal your computer to get the private key, you can delete it from the computer and keep it in your head—that would be an example of maximum safety!

Now there is another problem. A hacker can't just take any message and sign it like it is your message, but he can take an already signed message and paste it into the blockchain again; the signature of this message stays the same, doesn’t it? For this reason, all messages should contain a unique identifier, and all these identifiers should be in ascending order in the blockchain.

To get a unique identifier you should implement a method in the Blockchain class that always returns different numbers in the ascending order starting from number 1.

In this stage, you need to upgrade the messages. The message should include the text of the message, the signature of this message, a unique identifier (remember to include a unique identifier when creating a signature), and a public key so everyone can check that this message is valid. Don't forget to check every message when checking that the blockchain is valid! The blockchain should reject the messages with identifiers less than the maximum identifier in the block in which miners are looking for the magic number. Also, when validating the blockchain you should check that every message has an identifier greater than the maximum identifier of the previous block.

8. Stage 6: Local currency

Today, the most common application of blockchains is cryptocurrencies. A cryptocurrency's blockchain contains a list of transactions: everyone can see the transactions but no one is able to change them. In addition, no one can send a transaction as another person; this is possible using digital signatures. You have actually implemented all of this functionality in the previous stages.

A miner who creates a new block should be awarded some virtual money, for example, 100 virtual coins. This can be remembered in the blockchain if the block stores information about the miner who created this block. Of course, this message also should be proved, so the miner adds this information to the blockchain before starting a search for a magic number.

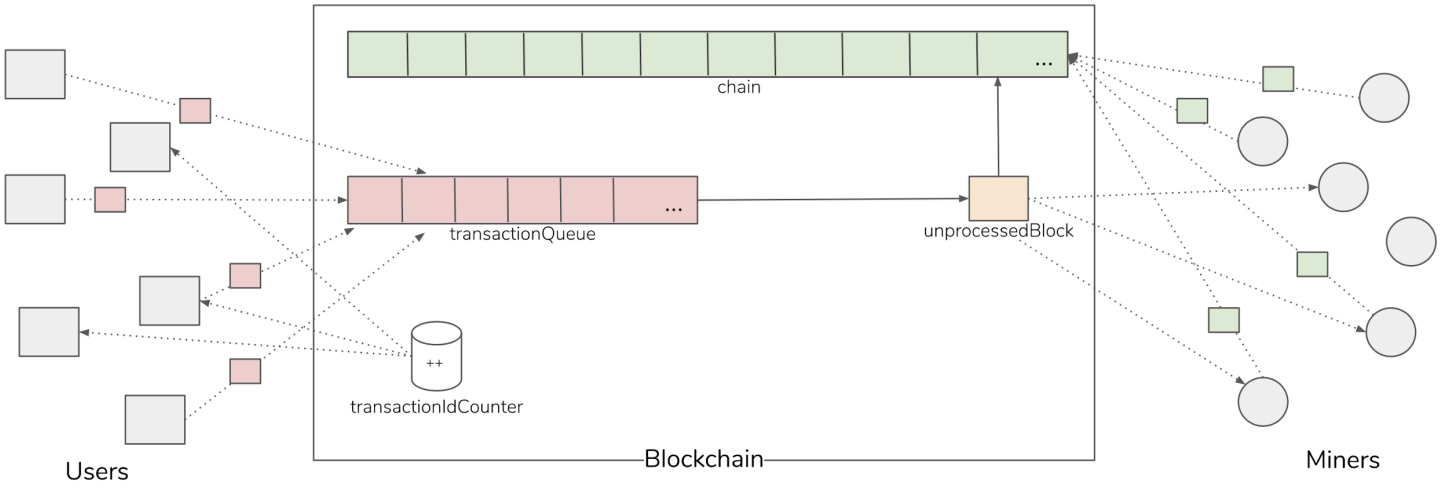
After that, a miner can spend these 100 virtual coins by giving them to someone else. In the real world, he can buy things and pay for them using these virtual coins instead of real money. These virtual coins go to the company that sells the things, and the company can pay salaries with these virtual coins. The circulation of these coins starts here and suddenly virtual coins become more popular than real money!

To check how many coins a person has, you need to check all of his transactions and all of the transactions to him, assuming that the person started with zero virtual coins. The transaction should be rejected when the person tries to spend more money than he has at the moment. Create a special method that returns how many coins the person has.

In this stage, you need to implement transactions like this instead of text messages like in the previous stage. For testing reasons you can assume that everyone starts with 100 virtual coins, not 0. But as described above, all the money of the blockchain is initially awarded for creating blocks of the blockchain.

9. Final Model

The final simulation model is developed as shown in the image below:




```
public int getTransactionId();
public boolean addTransaction(Transaction transaction);
private Block createBlock();
public boolean submitBlock(Block block);
public Block getUnprocessedBlock();
```

The system consists of two types of users as in the real-time use case: *Users* and *Miners*.
Users' behavior is modeled after the fact that users will use the system to make any money related transactions and check their balance every now and then. So, Users will be sleeping for a random amount of time (with the minimum being 1ms) and will be performing transactions until they are interrupted.

Miners are also Users of the system, but with an additional capability of mining blocks and submitting them to the blockchain. The blockchain then verifies the validity of the block and adds it to the chain. This is not exactly how a decentralized cryptocurrency application works, in that there will be no centralized authority verifying the blocks. Instead, the majority consensus of the network will be considered in deciding whether to accept a block and continue in that direction or to reject it. Anyways, since we're developing the model as a standalone application, this approach should work fine enough to understand how mining works and blockchain is ***secure by design***.

Users and Miners are modeled as Threads, and they will continue to do their job until interrupted. Since Blockchain was not modeled as a Tread (which if modeled would give rise to a lot of performance issues), it is the responsibility of users to add the transaction into Blockchain, and miners are responsible for mining and verifying the mined block and adding it into the Blockchain.

- Users will create a transaction and add it to the `transactionQueue` in the Blockchain. Before adding a transaction into the `transactionQueue`, a transaction must be authenticated (which is implemented using Digital Signatures), and the transaction must not contain more than the user's balance at that point.
- The balance of a user in the system will be calculated by analyzing all the transactions in the system (both mined and unmined). This introduces a lot of concurrency issues, if not properly synchronized.
- Once the transactions are added to the `transactionQueue`, miners will be able to create a block (which is placed inside `unprocessedBlock`) out

of those transactions and will be mining the block. When there are no blocks to mine, the miners will be sleeping for a random amount of time.
- The blocks mined by the miners must be validated against the mining constraint and for any tampering in the data. Upon success, it will be added to the Blockchain. The mining constraint for the next block will be updated depending on the time took for mining this block.
- When observed carefully, creating a block from `transactionQueue` seems more of a producer and consumer problem. So, producers (Users) must be waking up consumers (Miners) when required (i.e., The user should create the block and place it inside `unprocessedBlock`). That way, when a miner sees a block to be mined after coming from sleep, it starts mining the block.

Creating blocks out of `transactionQueue`, while users are still adding transactions, and writing to `unprocessedBlock` while someone else is reading it introduces a lot of concurrency issues, which synchronized with a mutual exclusion policy, degrades performance. An elegant solution like read-write locks is required.

The model is developed in Java and the entire source code is open-sourced at Github, by tagging the project at every stage.

10. Conclusion

It has been such a pleasure and challenge working with blockchain, and we've learned a lot from working on this project, especially Java Security and Concurrency APIs. The resulting model helped a few of my peers understand cryptocurrencies and blockchain in a more friendly way, for which we are proud of.