國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

# 可擴展區塊鏈錢包系統的設計與實作

# Design and Implementation of a Scalable Blockchain Wallet System

詹上潁

Shang-Ying Jhan

指導教授：廖世偉 博士

Advisor: Shih-Wei Liao, Ph.D.

中華民國 106 年 8 月

August, 2016

# 誌謝

在完成這個研究的過程中，得到了許多人的幫助。我要感謝恩冉學長、永宸學長提供了許多寶貴的意見，每次和學長們的討論都讓這個研究推進了一大步。另外我要謝謝 Tammy Liao 幫我的英文潤稿。當然還有帶領我進入區塊鏈研究領域的指導老師廖世偉博士。最後我要感謝我最愛的爸爸、媽媽，沒有你們幫助和支持，我不可能無憂無慮地完成碩士學位。

# 摘要

線上錢包系統在區塊鏈生態系中扮演了一個很重要的角色，它提供使用者簡單、容易上手的操作介面，讓使用者可以忽略區塊鏈背後的技術細節，降低使用基於區塊鏈的電子貨幣系統的進入門檻。

Gcoin 是基於區塊鏈技術所設計出來的電子貨幣系統。因為區塊鏈技術先天上採用分散式、點對點的精神設計而成。現有的 Gcoin 參考實作（本文以下稱 Gcoin Core）並不是針對大量使用者的使用情境設計，所以採用Gcoin Core 為基礎打造的線上錢包系統很難擴展以提供大量使用者使用。

本論文探討如何修改 Gcoin Core，使得使用Gcoin Core 核心程式為基礎打造的線上錢包系統可以簡單、快速的擴展，以提供更多使用者使用，或是承受短時間內大量請求。除了可擴展之外，在本論文所設計的架構之下，還提高了系統的可用性和彈性。

雖然本論文的所提出的設計只有實作在 Gcoin 上，但同樣使用區塊鏈技術的其他電子貨幣系統也一樣可以使用。

# Abstract

Web wallet systems play an important role in the blockchain ecosystem. They provide users with a simple and friendly user interface so that users can ignore the technical details of blockchain technology. Web wallet systems also reduce the barrier to adopt the digital currency system which is based on blockchain technology.

Gcoin is one of the digital currency systems based on blockchain technology. Due to the characteristics of blockchain technology (distributed and peer-to-peer), the existing reference implementation, Gcoin Core forked from Bitcoin Core is not designed for serving a large number of users. So, the web wallet system which uses Gcoin Core as a building block is very hard to scale.

We made some minor modifications to Gcoin Core so that a web wallet system using Gcoin Core as a building block of a web wallet system can be scaled very easily to support a large number of requests and serve more users simultaneously. In addition to scalability, the design and architecture proposed in this thesis can also improve the availability and flexibility of the system.

Although we have only implemented the design and architecture on Gcoin, the design and architecture that proposed in this thesis can be applied to other digital currency systems that are also based on blockchain technology.

# Contents

# List of Figures

# Chapter 1

# Introduction

Web wallet systems play an important role in the blockchain ecosystem. They provide users with a simple and friendly user interface so that users can ignore the technical details of blockchain technology. Web wallet systems also reduce the barrier to adopt the digital currency system which is based on blockchain technology. Gcoin Core is the reference implementation of Gcoin. It provides almost every feature that a node needs in the Gcoin network including a built-in wallet. Using Gcoin Core as a building block to implement a web wallet system is faster and cheaper than building a new one from scratch. However, Gcoin Core is not designed for simultaneously serving a large number of users, and we found that the performance of Gcoin Core becomes the bottleneck of the web wallet system when handling a lot of user requests. This issue leads to delayed response times and bad user experiences.

Gcoin Core's performance issue definitely needed to be fixed, but we did not want to completely abandon Gcoin Core and implement a new wallet system from scratch. We made this design decision due to security and cost concerns. Gcoin Core is open source software overseen by many developers, so using Gcoin Core is more secure. The modules which handle the validity of transactions and communication with other nodes also maintain blockchain data so that it can be reused and we do not have to invest time and effort in reinventing the wheel. This thesis focuses on how to design and implement a scalable architecture for a web wallet system while using Gcoin Core as a building block.

The remainder of the thesis is organized as follows: chapter 2 introduces key features of a wallet system, chapter 3 discusses the problem of just using existing implementation of Gcoin Core, chapter 4 introduces the design and system architecture, chapter 5 goes deep into the implementation detail of the scalable wallet system, chapter 6 shows the experiment results, and in chapter 7, we conclude the thesis and discuss future works.

# Chapter 2

# Background

## 2.1 Transaction and Unspent Transaction Output (UTXO)

Transaction is a data structure that records the transfer of coins from one owner to another. A transaction is composed of inputs and outputs. An output records the receiver and the amount of coins. The previous received output will be recorded as the input in the new transaction, so the input and the previous output are linked together. You must first have an output so that you can use that output as the input of the next transaction in all cases except for in the case of a special type of transaction called the “coinbase” transaction. Coinbase transactions create outputs from thin air the need for an input. In Bitcoin, coinbase transaction will only appear in the first transaction of every block. In Gcoin, issuers can mint their coins through “mint” type transaction. Mint type transactions are also a coinbase transactions.

Unspent transaction output means an output has not yet been used as the input of another transaction. Unspent transaction output is important because it represents the current ownership state of coins on the whole blockchain. The balance of an address is just the sum of the address’s UTXOs. When creating transactions, you need to select some output from the UTXOs to be the input of the new transaction.

## 2.2 Gcoin Core

Gcoin Core forked from Bitcoin Core is the reference implementation of Gcoin. It provides almost every feature that a node needs in the Gcoin network. Gcoin Core can communicate with other nodes in the network, validate and maintain blockchain, mine blocks. It also has a built-in wallet. We grouped these features into a network module, blockchain module, miner module, and wallet module, respectively, in this thesis. Gcoin Core stores blocks in binary files called “blk files” on the disk and uses LevelDB to store “block index”, “tx index”, and “UTXO”. It uses BerkeleyDB to store wallet-related data, including private keys and transactions that are related to the private keys managed in the wallet of another database called “wallet.dat”.

## 2.3 Key Management

Gcoin uses elliptic curve digital signature algorithm to sign the transactions. Private key management is one of the key features that a wallet should provide. Gcoin Core uses Berkeley DB to store private keys in a file called “wallet.dat”. Gcoin Core also provides import/export/backup private keys and supports native wallet encryption.

## 2.4 Get Balance Operation

The Get balance operation is another important feature that a wallet should provide. The naivest way to get the balance of an address on blockchain is to just simply scan through the whole blockchain starting from the first block and sum up all the UTXOs of the address, but this is not very efficient so Gcoin Core has found another solution. Gcoin Core also stores transactions that are related to the private keys in wallet.dat. It uses transaction hashes as keys and the transactions as values. Gcoin Core will load the related transactions into wallet.dat and maintain a map which uses the address as the key and a set of transactions that is related to that address as values in memory. Gcoin Core provides an RPC called "getaddressbalance". The "getaddressbalance" RPC finds all the address's related transactions, then filters out unspent outputs from these transactions and calculates the balance from these UTXOs.

## 2.5 Send Coins Operation

Another key feature is the send coins operation. Since transactions are composed of inputs and outputs, we need to find and select a set UTXOs as inputs to create a transaction. Gcoin Core provides an RPC called "sendfrom". The "sendfrom" RPC finds the address's UTXOs from every output of the address's related transactions (this step is almost the same as getaddressbalance), then select the appropriate UTXOs. Finally, the "sendfrom" RPC signs and broadcasts the transactions created from the selected UTXOs.

# Chapter 3

# Scaling Problem

## 3.1 Original Design

Figure 1 is the architecture of a wallet system using Gcoin Core as a building block. Gcoin Core has a built-in wallet module and wallet database, so we simply run a Gcoin Core in the server and use its built-in features -- using the network module to communicate with other nodes in the network, the blockchain module to maintain a blockchain data and UTXO database ant the wallet module to manage private keys and handle users' get balance requests and send coins requests. There is a thin web UI and API layer between users and Gcoin Core. We use this layer to prevent users from directly accessing our Gcoin Core. In this layer, we can also implement some access control or business logic. Since blockchain and Gcoin users only know addresses, we use a database to store the mapping of users and addresses so that we can know which

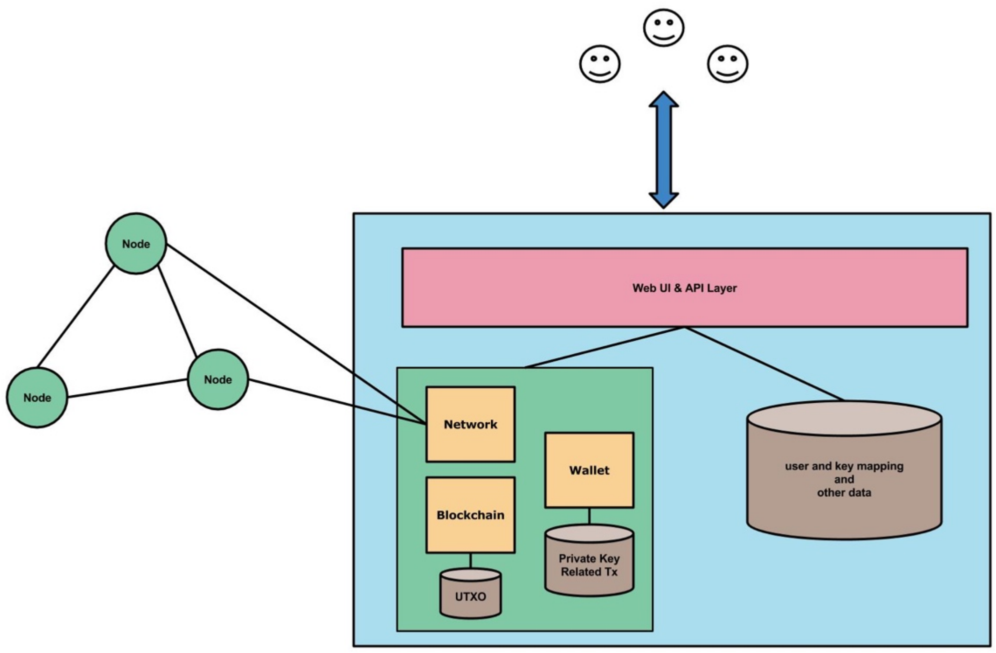

Figure 1: Wallet system using original Gcoin Core

user is the owner of the address. The database also stores users' metadata and data in order to implement business logic onto the web UI and API layer.

In this architecture, every user request that is related to blockchain, such as getting the balance or sending coins, will eventually be handled by the underlying Gcoin Core. When the number of users grows, the load cannot be handled and Gcoin Core cannot respond quickly. User requests have to queue and wait for a while, leading to bad user experience.

## 3.2 The Naive Scale-Out Solution

One of the naive solutions to scaling the wallet system is adding more resources and increasing the number of nodes. When we have multiple nodes running Gcoin Core, we can dispatch the requests to different Gcoin Core to improve the throughput of the wallet system. Figure 2 shows the architecture of this solution. There are several defects in this naive scale-out solution. This solution dose not really solve the scalability problem. The wallet module of Gcoin Core stores and manages both private key and the transactions related to the private keys its own wallet database. Since Gcoin Core is not designed for large systems, it uses the embedded database, Berkeley DB, instead of a client/server database as the wallet database. When the private keys or the transaction related to the private keys grow in number, querying or maintaining the wallet database becomes inefficient. The wallet database also needs to be flushed frequently to avoid corruption if the Gcoin Core suddenly crashes, but this process becomes time-consuming if the wallet database is very large.

Besides the fact that the wallet module becomes increasingly inefficient as the user base grows, it also needs private keys to perform the send coins operation. Managing private keys among nodes also becomes a problem.

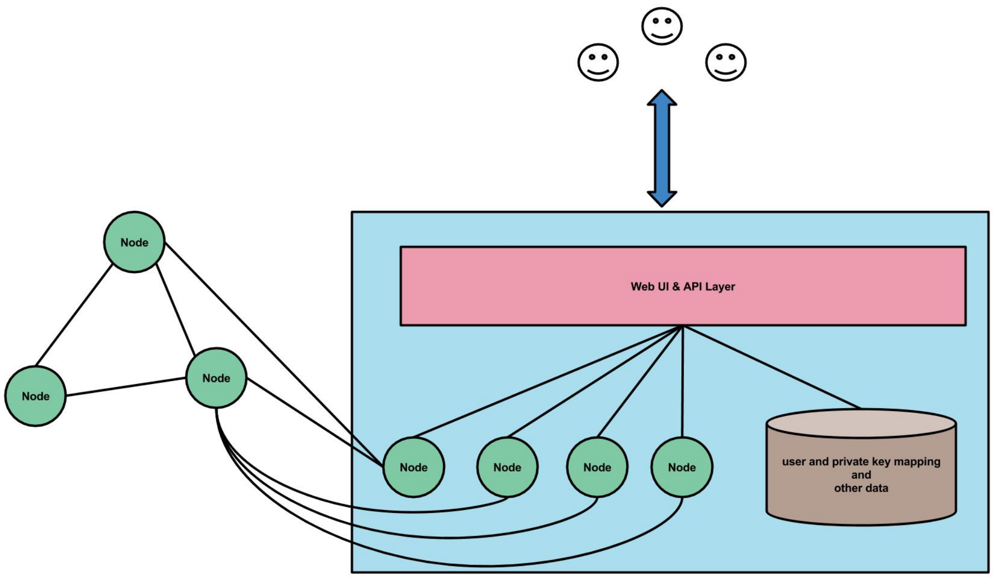

Figure 2: The naive scale out solution

One solution is to divide users and nodes into groups and dispatch user requests to the right node containing the user's private key according to the user's group, but in this way, the load will likely be imbalance among the nodes when the requests are all dispatched to the same nodes due to the same group of users are simultaneously using the wallet system. It would be a waste of resources in the instance that one node is busy while the others remain idle, and as this system is unable to resist node crash, a portion users will not be able to access our wallet system in the case of a node crash. Another solution would be to store a full copy of the private keys in the wallet database in every node in the wallet system, so that users' requests can be randomly dispatched to any of the nodes in the system. But with this way, we need to deal with the problem of syncing private keys. When a new user registers to our system, we need to generate a new private key for the user and sync the private key among nodes.

It is hard to scale the wallet system with either solution. Managing private keys among nodes and using an embedded wallet database make Gcoin Core and the wallet system inefficient and hard to deploy and scale.

# Chapter 4

# Design and Architecture

## 4.1 Overview

In chapter 3, we found that the root cause of the problem is that the wallet module of Gcoin Core maintains private keys and related transactions in the wallet database. Since it stores private keys and related transactions, it is stateful from the perspective of the users. When the node is stateful, it is difficult to scale out. In this thesis, we will propose a new design of Gcoin Core to solve this problem and improve scalability.

In order to make Gcoin Core stateless from the aspect of users, we need to remove the wallet database from Gcoin Core. After removing the wallet database, Gcoin Core will no longer manage the private keys and related transactions. The problem of key management among nodes does not have to be solved inside Gcoin Core, but should be solved from the outside. We will explain how we provide the key management, get balance operation and send coins operation features after removing wallet database from Gcoin Core in the following sections.

## 4.2 Key Management

After removing the wallet database from Gcoin Core, we need to manage private keys ourselves. To retain flexibility, we do not restrict where to store private keys in our architecture. We designed it so that private keys can be stored in anywhere as long as key is stored safely and the user can find the private keys when needing it to sign a transaction.

## 4.3 Get Balance Operation

Since the blockchain module needs to validate the transactions received by the network module from other nodes in the network, blockchain maintains a UTXO database on its own. The balance of an address is simply the sum of an address's UTXOs. We can utilize the UTXO database to implement the get balance operation. However, the trade-off of using UTXO database rather than the wallet database is that the get balance operation will take a longer time to process. The wallet database only maintains transactions related to the private keys in the wallet, and it can be loaded into memory to improve the searching speed, but the UTXO database maintains every UTXO on the blockchain, so it is not suitable to load the whole UTXO database into memory. Although the get balance operation will be slower, we can deploy multiple nodes to handle requests simultaneously to improve the throughput of the wallet system.

## 4.4 Send Coins Operation

In the original Gcoin Core, the send coins operation was done by one RPC with the following step. First, the RPC selects a set of appropriate UTXOs as transaction inputs. Second, it creates the raw transaction structure. Third, the RPC finds the private key in the wallet database then sign the raw transaction. Finally, it broadcasts the transaction to the network. With our design, without the wallet database, the third step cannot be done in Gcoin Core. We decided to divide the send coins operation into two RPCs. The first RPC selects a set of appropriate UTXOs from the UTXO database and returns the raw transaction. We can sign the raw transaction by using a crypto library with the private key and the raw transaction as the input outside Gcoin Core to generate a signed transaction. The second RPC broadcasts the signed transaction to the network.

However, the trade-off is the same as with the get balance operation: using the UTXO database to select UXTOs is slower than using the wallet database.

## 4.4 Summary of the Design

To summarize, we removed the wallet module and wallet database from Gcoin Core and implemented two new RPC, “getbalancestateless” and “createrawtransactionstateless”. With this new Gcoin Core, we can easily scale out our system by deploying more nodes to achieve scalability. We can also deploy some redundant nodes to deal with unexpected server crashes and thus improve availability. Figure 3 shows the new wallet system architecture with just one node, and figure 4 shows the wallet system architecture with redundant nodes. Furthermore, private keys can not only be stored in wallet system’s database, but also be stored in the user’s computer or mobile device as long as there is a crypto library that can provide the transaction signing feature. We can implement an architecture where the user can choose where to store their private keys. Figure 5 is the wallet system architecture that

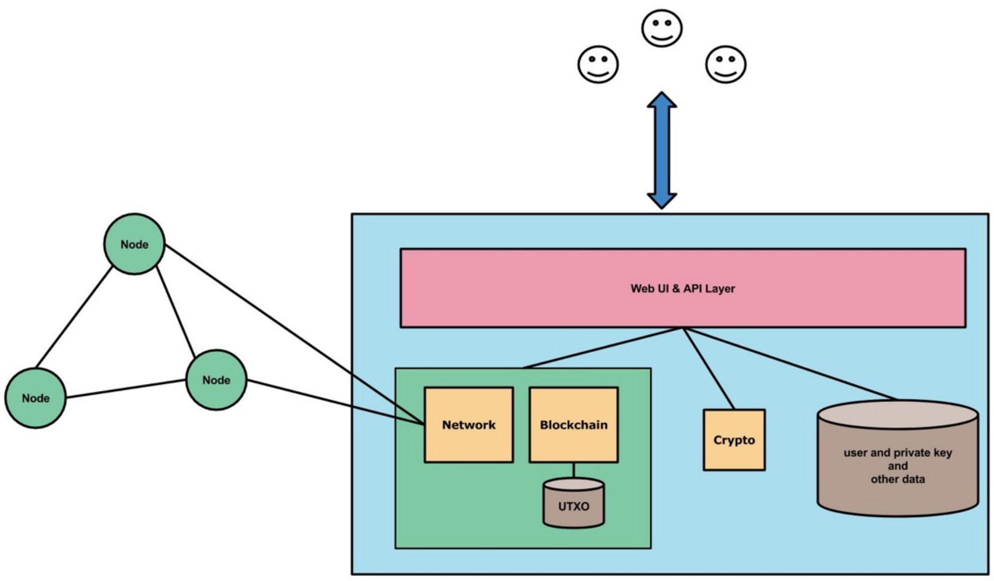

Figure 3: New wallet system architecture with just one node

stores private keys on the client side. This would enhance our wallet system's flexibility, so that we can adapt to different business scenarios.

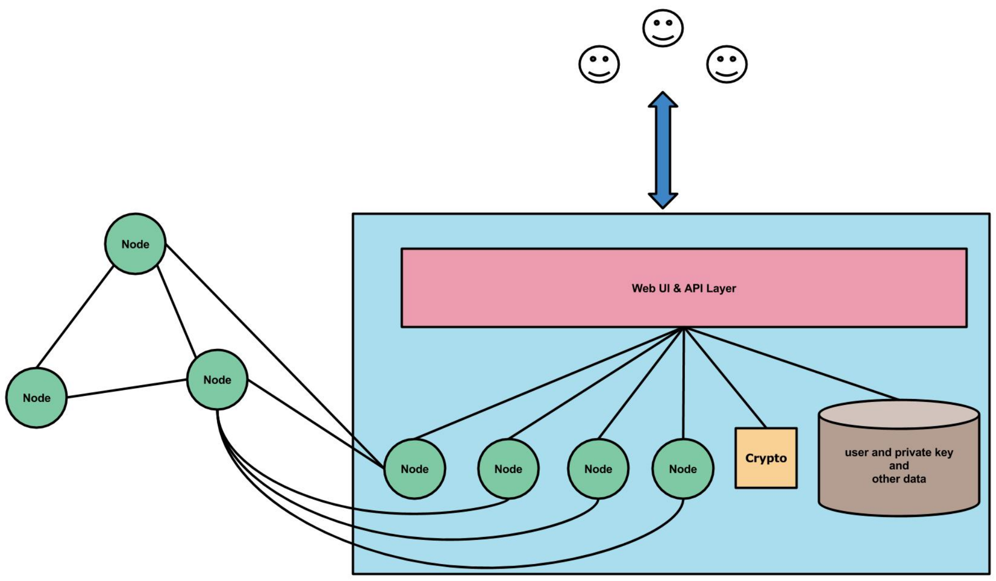

Figure 4: New wallet system architecture with redundant nodes

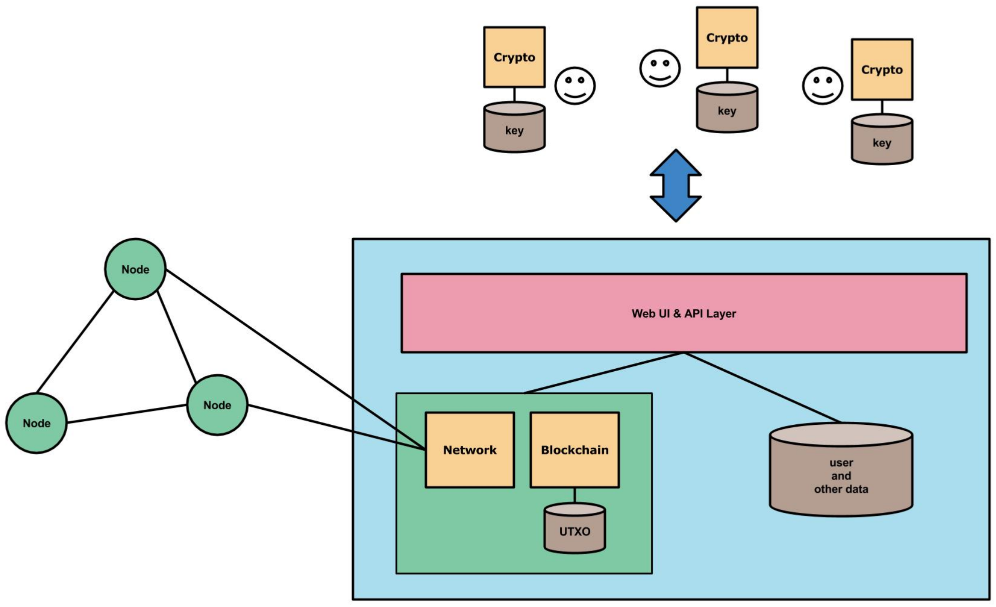

Figure 5: New wallet system architecture with private keys stored on the client side

# Chapter 5

# Implementation Detail

## 5.1 The getaddressbalancestateless RPC

The wallet module stores related transactions in the wallet database. When starting Gcoin Core, it will load all of the related transactions onto a multi-map data structure in memory called "mapWalletAddr" with the address as the key and the transaction and output index pair as the value. We can look up an address's UTXOs from "mapWalletAddr" by using the address as the key. The balance is simply the sum of the UTXOs. The wallet module will update the key-value pairs in "mapWalletAddr" when a new related transaction is added to wallet or a UTXO is spent. The wallet module will also flush the state of "mapWalletAddr" back to the wallet database periodically.

Finding the UTXOs of an address from the UTXO database is not as simple as finding them from "mapWalletAddr". The UTXO database maintained by blockchain module is a key-value database, but it uses transaction hashes as keys and the unspent transaction outputs in that transaction as values. We need to parse the scriptPubKey field in the transaction output to get the recipient's address. If we want to find all the UTXOs of an address, we need to linear search the entire UTXO database. The complexity of linear search is O(n). This is very inefficient and unacceptable when the number of UTXO grows. So, instead, we decided to build an index to avoid linear searching the entire UTXO database. The index uses the addresses as the key and the related transaction hash list as the value. We stored the index in the same database as the UTXOs. With the index, we can get the transaction hash list of an address from the UTXO database, and then we can linear search the output of the transaction hash in that list. The output number of a transaction is normally two, one is the output to the recipient and the other is the change, so with the index, we only need to linear search a small set of UTXOs.

The blockchain module updates the UTXO database in two functions, “ConnectBlock” and “DisconnectBlock”, so we also need to update the index in these two functions. Updating the index slightly slows down the “ConnectBlock” and “DisconnectBlock” functions and the index also requires some disk space, but the additional time and disk space is worth it when comparing it to the time that required to linear search all UTXOs on the blockchain. With the additional index, we can get all the UTXOs of an address from the UTXO database and then compute the balance efficiently.

## 5.2 The createrawtransactionstateless RPC

The implementation of “createrawtransactionstateless” is relatively easy when compared with “getbalancestateless” because we only need to reuse and combine the functions that we already have. We reused the function to find all UTXOs of an address, the use “SelectCoins” function and part of “createrawtransaction” RPC’s code. In addition to the unsigned raw transaction, we also returned the UTXOs used in the raw transaction including the transaction hash, index of outputs, and scriptPubKey. The crypto library needs the scriptPubKey of an output to generate scriptSig.

# Chapter 6

# Experimental Result and Comparison

## 6.1 Environment

### 6.1.1 Hardware

We used two type of google compute engines. Google Compute Engine n1-standard-1, 1 vCPU, 3.75 GB memory, for Gcoin Core and Google Compute Engine n1-highcpu-8, 8 vCPU, 7.2 GB memory, for test client.

### 6.1.2 Blockchain state

We generated 10,000 addresses and 150,000 UTXOs, with an average of 15 UTXOs per address, on blockchain. This simulates a blockchain state where 10,000 users have been using the wallet system for an extended period of time.

## 6.2 Architecture

The experiment environment system architecture is shown in Figure 6. A test client (n1-highcpu-8) is sending a lot of requests to our system. A NGINX (n1-standard-1) server as reverse proxy to dispatch the requests to underlying Gcoin Core (n1-standard-1). We implemented a simple API layer between Gcoin Core and NGNIX. The API Layer just received an http request and translated the request into RPC to Gcoin Core.

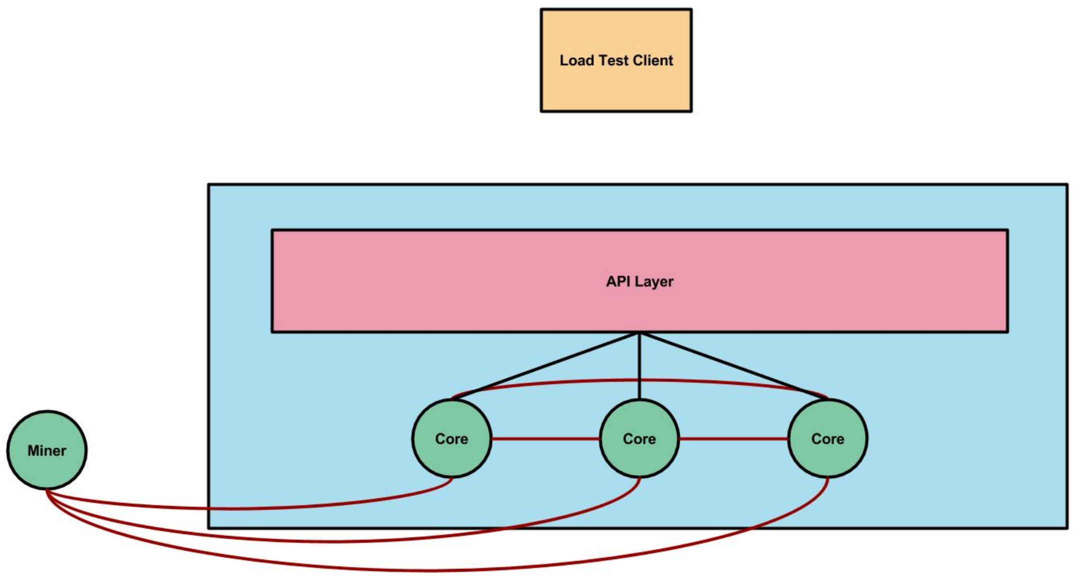

Figure 6: The test environment architecture

## 6.3 Result and Discussion

We ran tests to measure the throughput and average response time of both the original and the new wallet system. We only ran the tests on the original wallet system with one node, since it's not scalable, but we ran the tests on the new wallet system with different node numbers to test whether or not our new design and architecture really works and is better than the original wallet system. Figure 6 and Figure 7 show the results of doing 10,000 get balance operations in both the original and new wallet system. Figure 8 and Figure 9 show the results of doing 1000 send coins operations in both the original and new wallet system. Note that the send coins operation in the original wallet system uses just one RPC while in the new wallet system, it uses two. In the original wallet system, the signing transaction operation is included in RPC, but in the new wallet system, the signing transaction operation is done in client. The results of the original wallet system you see in Figure 8 and Figure 9 has already had the time spent signing transaction subtracted.

In Figure 8 and Figure 9, we see that when the node number is just one, regardless of the total required time or the average response time, as expected, the new Gcoin Core takes longer than the original Gcoin Core to perform the get balance operation. We did not cache the whole UTXO database in RAM, so finding UTXOs required some disk I/O which caused it to take a longer time.

However, when the number in the new wallet system is increased, the advantage of a stateless becomes apparent. It is natural that when more nodes are performing the get balance operation simultaneously, the total required time and average response time will reduce.

Figure N and Figure N show that even with just one node, the new wallet system is better than the original one. We analyzed the source code and the behavior of the original Gcoin Core and found that there are critical sections and a standalone thread that periodically checks whether or not there are updates in the wallet database and flushes the data cache in memory to the wallet database when there is an update in wallet database. Critical sections make it so that send coins operations cannot be executed in parallel while the standalone thread also block the send coins operation when flushing the database. In our new design, we divided the send coins operation into two RPCs, "createrawtransactionstateless" and "sendrawtransaction". There are no critical sections in "createrawtransactionstateless", since "createrawtransactionstateless" will not change the state of blockchain. Although there are still critical sections in "sendrawtransactions", the part of "createrawtransactionstateless" can be executed in parallel. This allows the send coins operations to be executed like a pipeline and the critical sections to be more fine grain. Therefore, in our new wallet system, even just with one node, the performance is better than the original wallet system.

By comparing the results of the get balance operation to the send coins operation, we found that the effects of increasing the node number on the get balance operation are significant, but for send coins operation, the results are not significant. This is due to the fundamental limit of Gcoin. Gcoin is a peer-to-peer and distributed digital currency, so every node in the network will validate transactions. Current implementation of GCoin Core validates the transactions that it receives and stores them into a memory pool. The transactions will stay in the memory pool until they are collected into a block. The send coins operations can dispatch to different nodes in the new wallet system, but the nodes will broadcast the transactions to other nodes in the network. The total transactions that a node needs to handle eventually ends up being the same and will not decrease even if the node number increases. Our new wallet system allow one node to create transactions in its own memory pool quickly, but it is still necessary for the the nodes to validate every transaction, and this is limited by the maximum transactions per second of Gcoin Core.

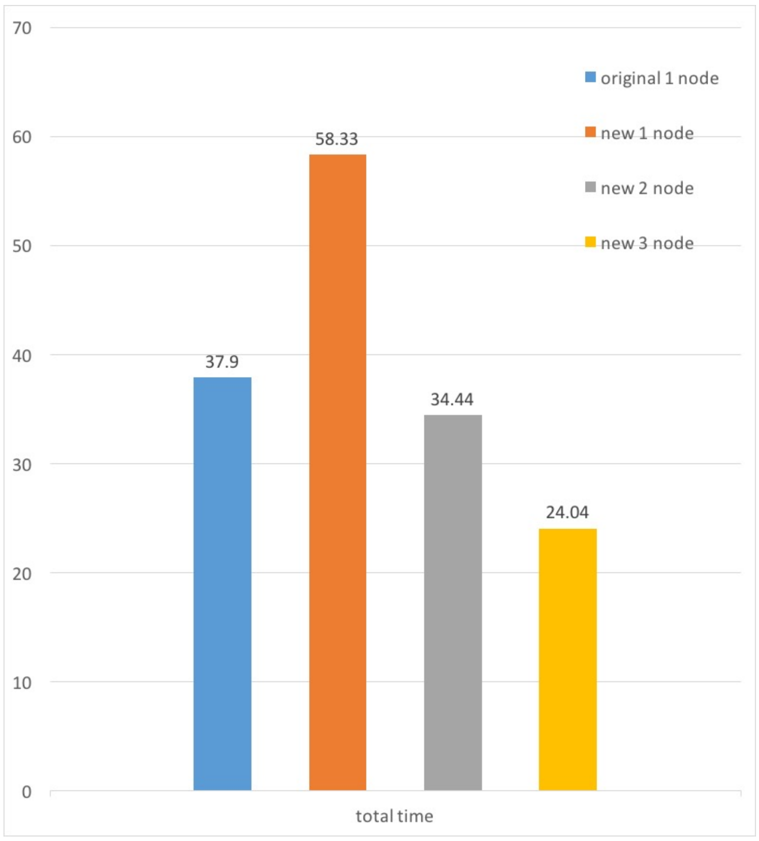

Figure 7: The total time of 10,000 get balance operations

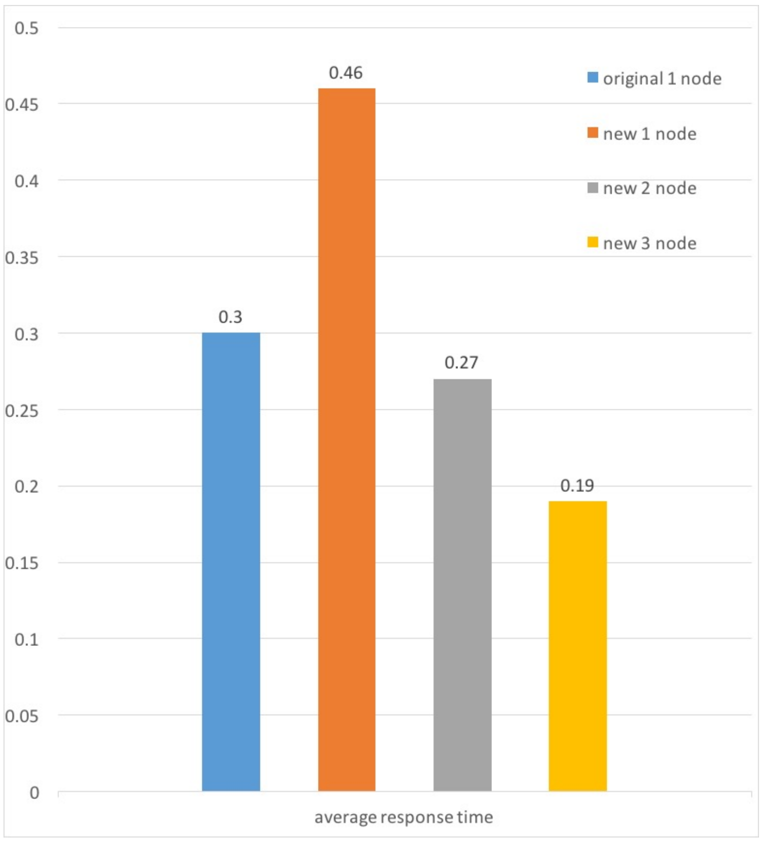

Figure 8: The average response time of 10,000 get balance operations

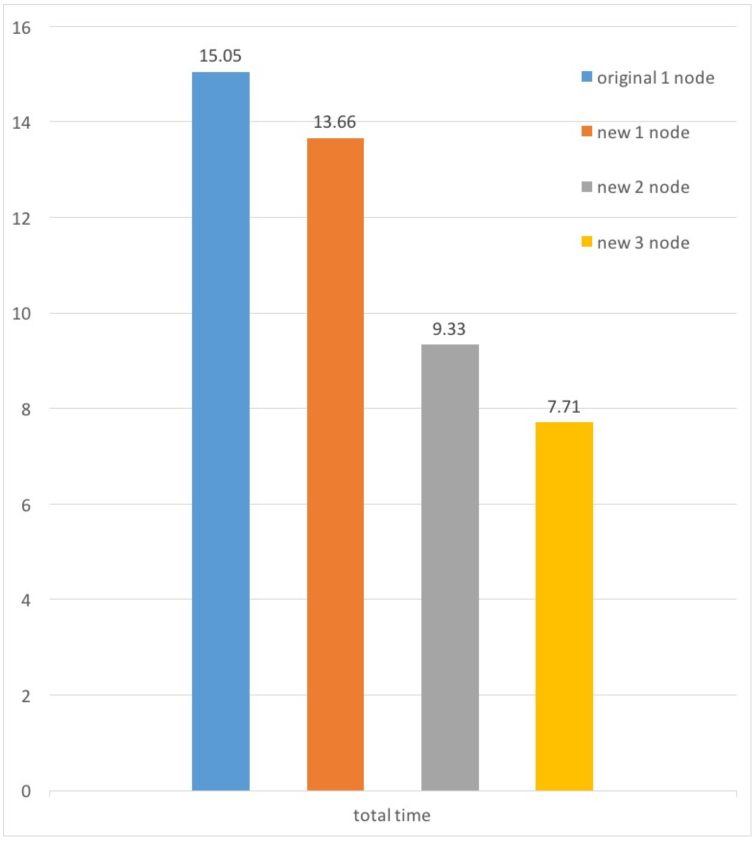

Figure 9: The total time of 1,000 send coins operations

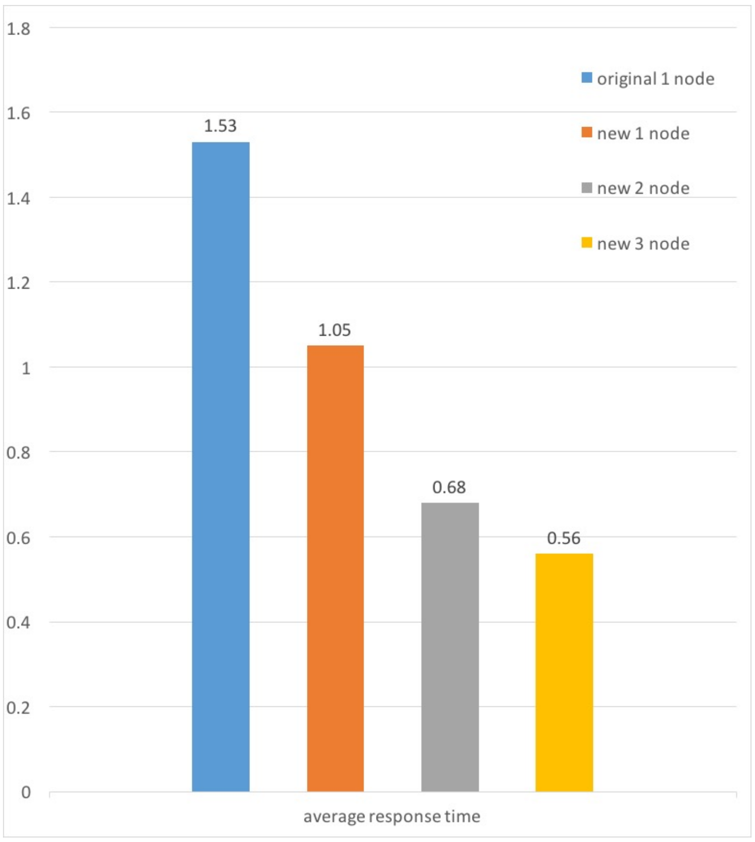

Figure 10: The average response of 1,000 send coins operations

# Chapter 7

# Conclusion and Future work

## 7.1 Conclusion

We introduced the stateless concept in the design of the Gcoin Core and architecture for a large-scale blockchain wallet system so that the nodes in the wallet system can be deployed and scaled very easily. The experiment results prove that the design and architecture proposed in this thesis really works.

## 7.2 Future Work

There are still some synchronizing issues that need to be solved in our wallet system. We proposed some naive solutions in this thesis, but they need to be further discussed and verified.

### 7.2.1 Zero confirmation issue

The current implementation relies on at least one confirmations, but if we want to support zero confirmations in our wallet system, there will be issues. When a user requests a send coins operation followed by a get balance operation and the requests are dispatched to different nodes in our wallet system, so users will see different UTXO sets if the underlying Gcoin Core has not yet been synced.

Although this issue is not critical and will not lead to double spending, it will confuse users. We propose to let the client side handle this issue. Clients can keep transactions for a short time and detect whether or not the issue occurred, then wait for the Gcoin Core to finish syncing.

### 7.2.2 Chain fork issue

The chain fork issue is an inherited characteristic of blockchain technology. This issue is the same as the zero confirmation issue--they are all caused by the temporarily inconsistent state of nodes among our wallet system. We propose placing a gateway node in our wallet system, so that the blocks and transactions that broadcast into our wallet system will be the same. When a chain fork happens in the network, all the nodes in our wallet system will choose the same candidate chain. How to design an efficient high performance and high availability gateway node is left as future work.

# Bibliography

[1] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

[2] Antonopoulos, A. M. (2016). *Mastering bitcoin: Unlocking digital cryptocurrencies*. S.l.: O'Reilly Media.

[3] Bitcoin Developer Documentation. (n.d.). Retrieved August 14, 2016, from https://bitcoin.org/en/developer-documentation

[4] Gcoin White Paper. (n.d.). Retrieved August 14, 2016, from https://github.com/OpenNetworking/gcoin-community/wiki/Gcoin-white-paper-English

[5] Gcoin open source project. (n.d.). Retrieved August 14, 2016, from https://github.com/OpenNetworking/gcoin-community