Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Asynchronous Consensus-Free Transaction Systems

Semester Thesis

Shoma Mori

shmori@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Roland Schmid, Jakub Sliwinski
Prof. Dr. Roger Wattenhofer

December 18, 2019

Acknowledgements

I thank Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Abstract

The abstract should be short, stating what you did and what the most important result is. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Contents

Chapter 1

Introduction

In the existing blockchain systems, it is assumed that the consensus problems should be solved to achieve correctness. For instance, Bitcoin uses proof-of-work to get consensus among agents in distributed systems.

However, proof-of-work algorithm takes much cost and therefore prevents scalability. The transaction systems can get popularity if they get scalability improvement.

We introduce Asynchronous Consensus-Free Transaction Systems (ACFTS) aiming to get scalability by removing consensus protocol. ACFTS verifies transactions using agents call servers, which dedicate to manage transactions and keep correctness in the system. The servers put their signatures to prove the correctness of transactions. Importantly, the servers work asynchronously meaning each server does not have to communicate with other servers. In other words, ACFTS does not need consensus.

We implemented ACFTS as a payment system and evaluated the throughput. From the results, we found that the part of the verification of signatures becomes a bottleneck.

Chapter 2

Theory of ACFTS

2.1 Model

The system consists of two kinds of agents called servers and clients. In our system, transactions represent payments from clients to clients. The clients can send cryptocurrency to the other clients or themselves. The servers verify transactions when they receive them and can save them in their memory. Every client can send messages to all servers and all other clients. In the same way, every server can send messages to all clients. However, the servers do not send messages to the other servers.

Each client holds private and public key pairs. The public keys are also called addresses, which are used to represent owners of transactions.

2.2 Protocol

2.2.1 Transaciton

Transactions in our system represent transfers of cryptocurrency from clients to clients. Each transaction consists of one or more inputs and one or more outputs. An input includes one or more outputs and signatures of its owner. In other words, each output can be spent by the client who has a private key connected with the address (i.e. the public key) of the output.

An output includes an address of its owner, amount of cryptocurrency and signatures from servers. An output also has a hash of outputs of the previous transaction. The topology of transactions is DAG. The sum of the amounts of outputs must be the same as the sum of the amounts of inputs.

Since each transaction can have more than one output, outputs can have their "siblings." Each output has an index of the siblings. Therefore, even if one transaction has multiple outputs that have the same addresses, they can be identified by the indexes.

Genesis

All transactions are created from any inputs, but the only genesis is different. The genesis is the first output and all outputs link to genesis as parents. The genesis is created by the system and its address represents the first owner of the cryptocurrency.

2.2.2 Server

A server is an agent who verifies transactions from clients. Every server records all transactions they verified in their database. When a server receives a transaction, firstly, the server check whether the outputs of the received transaction is used or not. If one of them is used, the transaction is not correct and the server sends an error to the client. If not, it verifies a client's signature to check the ownership. Since the signatures are generated from the private key of the client and the address is the same, the server can validate the signature with the address. Then, the server verifies signatures of servers which are also contained in the transaction. We assume that each server knows the public keys of all the other servers here. Importantly, the number of signatures that are verified must be more than two-thirds of all servers (The details will be described in) to use the output. Finally, the server checks whether the sum of the amounts of the outputs is the same as the sum of the amounts of the inputs. When the server completes the verification process without any errors, it makes an own signature from the hash of the outputs and attaches it to respond to the client. The number of signatures for each transaction is only one, which means the signature is created from the entire outputs, not from each output. This reduces the cost to make signatures. Also, the server adds the outputs into their database and updates the status of the inputs to record that they are used and cannot be used anymore.

2.2.3 Client

A client can create new transactions and send them to servers to get signatures. The client manages outputs whose addresses or siblings addresses belong to the client because servers make signatures from the entire outputs in each transaction and therefore the client needs to send the output with the siblings to make it possible for servers to verify the output. When the client sends requests of making a new transaction, it attaches a signature to claim the ownership of the outputs of the transaction.

Consider a transaction from client c_1 to client c_2. First, c_1 sends a request for the transaction with some unused transaction outputs (UTXOs) to all servers and waits for the responses. When c_1 get signatures from more than two-thirds of all servers, c_1 can "use" the transaction. To show the spending of the UTXOs

and make it possible for c_2 to use the new outputs, c_1 sends c_2 the transaction. c_2 can validate the transaction by verifying the signature of servers.

2.2.4 The flow of a payment

In our system, a payment is represented by a verified transaction. The following is a flow of making a payment from client c_1 to c_2.

1. c_1 finds UTXOs which c_1 has ownership of in their local storage.
2. c_1 creates a request for a transaction whose output address is owned by c_2 using the UTXOs including a c_1's signature.
3. c_1 sends the request to all servers and waits for the responses.
4. Servers verify the transaction in the request.
5. If the transaction has no errors, servers create their signatures and send them back as responses.
6. When c_1 gets signatures from more than two-thirds of all servers, c_1 sends the output of the new transaction to c_2.
7. When c_2 receives the output, c_2 verifies the signatures of servers.
8. If the number of valid signatures is more than two-thirds of the number of all servers, the transaction is approved.

2.2.5 Double-spending

In general transaction systems, using the same outputs more than twice, namely, double-spending, is one of the main problems. In our system, it is impossible to make conflicting valid transactions.

Consider a situation where c_1 tries to make two conflicting transactions which are payments to c_2 and c_3. To make a transaction valid, c_1 has to get signatures from more than two-thirds of all servers. However, if a server receives two conflicting transactions, it creates a signature for only one transaction which comes first. In other words, there are no ways to get signatures of both transactions from the same server. Furthermore, it is not possible to send each transction to more than two-thirds of all servers without overlapping. In short, the two conflicting transactions can never be valid at the same time.

In this case, only one or neither transaction is confirmed.

Chapter 3

Implementation

In this chapter, we describe the system in terms of implementation.

3.1 Structure

Servers and clients can communicate through the HTTP protocol. Clients can send HTTP requests with JSON format. Servers keep waiting for HTTP requests from clients. Also, clients can make communication through the HTTP protocol to receive UTXOs which are related to themselves. Therefore, clients also keep waiting for HTTP requests at the same time as making requests of new transactions.

Both servers and clients have their database to record transaction outputs.

Genesis is initially recorded in a client that has the genesis and all servers.

3.2 Documentations

3.2.1 Signature

ACFTS uses public key cryptography to show ownership of outputs and completion of verification. The implementation adopts the Elliptic Curve Digital Signature Algorithm (ECDSA) for key pairs and verification processes.

Clients create a hash of UTXOs with SHA256 and sign on it with a private key which is generated with ECDSA when creating a new transaction. Servers verify the signature with the public key of clients and sign on the UTXOs with a private key of the servers. The receiver of the transaction can verify the signatures of servers with public keys of servers.

UTXO hash

A UTXO have an address, a hash of outputs of the previous transaction and an index of siblings. We can show that UTXOs can be identified with these keys.

Hash values become the same deterministically if the inputs are the identical. The input of the hash includes a hash of two transactions ago. In other words, every hash of outputs has information of all transactions which precedes the outputs. From the genesis, each output has an index even if the previous transaction is the same. Therefore, every output has diffrent hash values without collisions of the hash function.

3.2.2 Change transaction

When a client tries to create a request for a transaction, the client finds UTXOs until the sum of the amounts becomes larger than the amount the client wants to send. If the sum exceeds the expected amount, the clients make a change transaction to make both ends meet.

3.2.3 Cluster

In the implementation, multiple client addresses can be managed in one database. We call the set of addresses cluster. When creating transactions among one cluster, it is not necessary to send UTXOs because they can refer to through the shared memory. In some sense, a cluster is a wallet and transactions in one cluster represent sorting out UTXOs. When creating transactions among different clusters, it is required to send the UTXOs, which is a real payment.

In the initialization process, each cluster exchange their addresses and therefore can decide to which cluster they should send UTXOs when creating new transactions.

Chapter 4

Experiment

We evaluated the throughputs of our systems by experiments using some scenarios of transactions. The system is implemented in Go and was benchmarked in a local environment. We do not assume network delay.

We also profiled the system to find bottlenecks aiming for improving the throughputs.

4.1 Environments

- The number of servers: 4
- The number of clusters: 2 ($cs0$ and $cs1$)
- The number of clients in each cluseter: 4
  ($\{ct0, ct1, ct2, ct3\} \in cs0$ and $\{ct4, ct5, ct6, ct7\} \in cs1$)
- The genesis: $amount = 1000000$, $owner = cs0$

We executed the following five diffrent scenarios. Each arrow means transfer of one amount of cryptocurrency.

- Scenario1: $ct0 \to ct1$
- Scenario2: $ct0 \to ct1$, $ct1 \to ct0$
- Scenario3: $ct0 \to ct1$, $ct1 \to ct0$, $ct2 \to ct3$, $ct3 \to ct2$
- Scenario4: $random \to random$ ($rondom \in cs0$)
- Scenario5: $ct0 \to ct4$

Note that sender and receiver are chosen from $cs0$ with equal probability in scenario4.

4.2 Results

Trials in tables means that execution of a set of transctions in one scenario counts one trial. For example, in scenario1, when trials is 10, transfer from $ct0$ to $ct1$ is executed 10 times. On the other hand, in scenario2, when when trials is 10, transfers from $ct0$ to $ct1$ and $ct1$ to $ct0$ are executed 10 times respectively.

Table 4.1: Scenario1

trials [tx]	speed [tx/s]
10	4.30
100	5.25
1000	4.37
10000	3.10

Table 4.2: Scenario2

trials [tx]	speed [tx/s]
10	5.11
100	5.56
1000	4.82
10000	1.52

Table 4.3: Scenario3

trials [tx]	speed [tx/s]
10	5.35
100	5.51
1000	5.06
10000	1.98

Table 4.4: Scenario4

trials [tx]	speed [tx/s]
10	6.63
100	6.37
500	6.17
1000	5.84
10000	2.67

Table 4.5: Scenario5

trials [transaction]	speed [transaction/s]
10	3.10
100	3.13
1000	4.54
10000	2.52

4.3 Discussion

Chapter 5

Conclusion

foofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoo

Todo: This is a TODO annotation.

Theorem 5.1 (First Theorem)**.** *This is our first theorem.*

Proof. And this is the proof of the first theorem with a complicated formula and a reference to Theorem 5.1. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

$$\frac{\mathrm{d}}{\mathrm{d}x}\arctan(\sin(x^2)) = -2 \cdot \frac{\cos(x^2)x}{-2 + (\cos(x^2))^2} \tag{5.1}$$

□

And here we cite some external documents [1, 2]. An example of an included graphic can be found in Figure 5.1. Note that in LaTeX, “quotes” do not use the usual double quote characters.

Figure 5.1: This is an example graphic.

Bibliography

[1] A. One and A. Two, "A theoretical work on computer science," in *30th Symposium on Comparative Irrelevance, Somewhere, Some Country*, Jun. 1999.

[2] A. One and A. Two, "A theoretical work on computer science," in *30th Symposium on Comparative Irrelevance, Somewhere, Some Country*, Jun. 1999.

Appendix A

First Appendix Chapter Title