CloudChain Project

Design Overview

Advisor:
Onur Tolga Şehitoğlu
Assistant:
Mustafa Levent Eksert
Team Members:
Berk Yaşar - 2036333
Cem Önem - 2031821
Ekin Dursun - 2098952
Sinan Erdil - 2098994

January 2018

Table of Contents

List of Figures

1 Product Description

CloudChain is a peer-to-peer file sharing platform that has blockchain reinforced consensus functionalities. What makes cloudchain different from usual P2P systems is that via these functionalities, features that traditional P2P systems do not possess due to lack of a trusted main entity (which breaks the P2P concept and may not be favorable) can be implemented. Along with the blockchain basis, CloudChain also provides most prominent two of these:

- **Authentication:** Traditional P2P protocols do not support scenarios for files shared between *only* certain entities. The concept of a user/group does not exist.
- **Negotiations:** Commonly, the only reason that a peer keeps sharing a file is that the peer benefits from the content itself or it being distributed. This benefit could be shifted to other assets with the addition of a blockchain (such as money, extra storage etc.), agreements are possible.

These features are realized by the contracts that run inside the blockchain (see Section 2 for more information about the out-of-system entities that CloudChain interact), that run on a seperate P2P network from the the network that files are shared (more information in Section 3). Basically, the file sharing network is responsible of pushing/pulling files while the contracts confirm, validate and broadcast the details and meta-data of these operations.

2 High Level System View

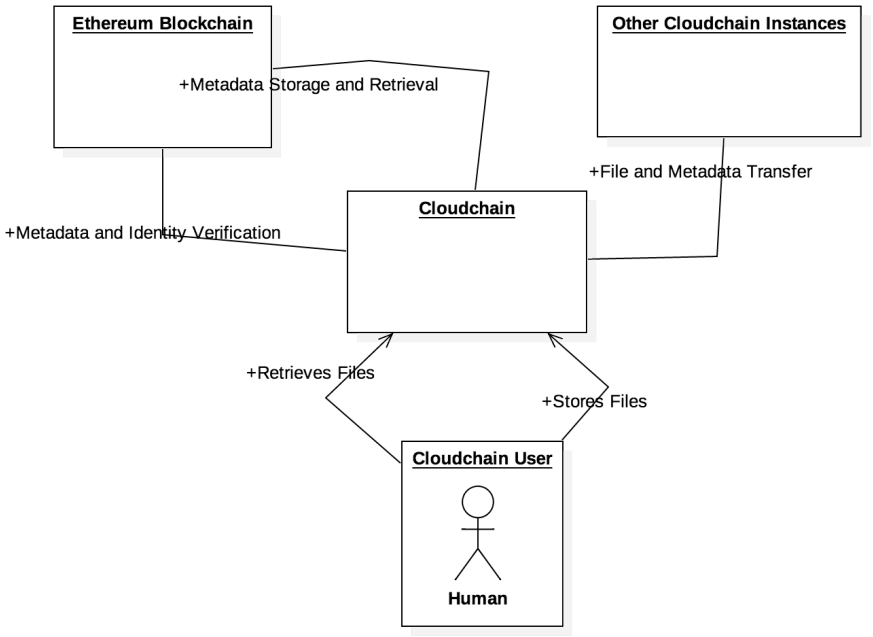

Figure 1: Context Diagram for CloudChain

A context diagram for CloudChain can be seen above. CloudChain consists of a standalone program, called a "CloudChain node" installed on the users computer. A CloudChain node is part of the CloudChain P2P network and communicates with other nodes to store/retrieve files from others. CloudChain nodes also interact with the Ethereum[1] blockchain through the use of an Ethereum client in order to store/retrieve metadata about the files stored in the network and for identity, metadata and signature verification.

3 Overall Design

3.1 Architecture

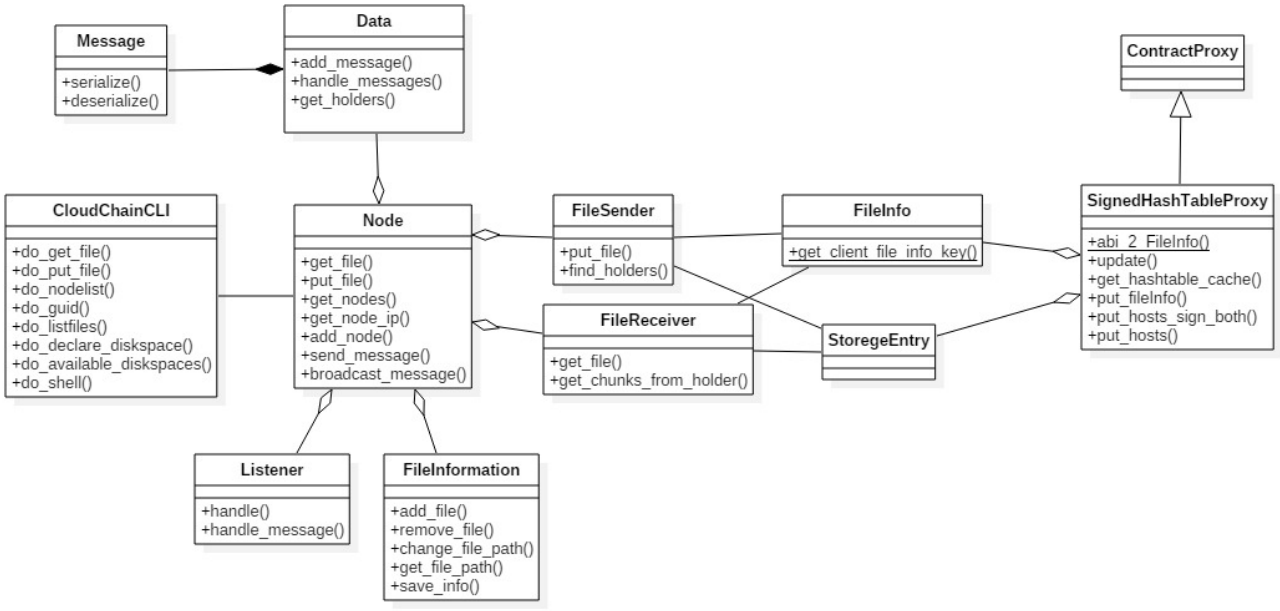

Figure 2: Class Diagram for CloudChain

A CloudChain node consists of the classes which can be seen from the Figure 2 and 3.

When a user starts a CloudChain node, one instance of `Node` and one instance of `CloudChainCLI` are created. `CloudChainCLI` is responsible for delivering user's commands to `Node` by calling its methods.

`Node` class manages all operations based on a CloudChain node. It creates an instance of `Listener`, which listens requests from other nodes and do relevant actions, an instance of `Data` which is basically the list of received messages and an instance of `FileInformation` which maps files on CloudChain with their local paths. Also, in every `get_file` and `put_file` call, one instance of `FileReceiver` and `FileSender` are created respectively.

`Data` holds a list of `Message` instances. `Message` is basic encapsulation of request payloads between clients. It serializes objects to binary data and deserializes binary data to objects.

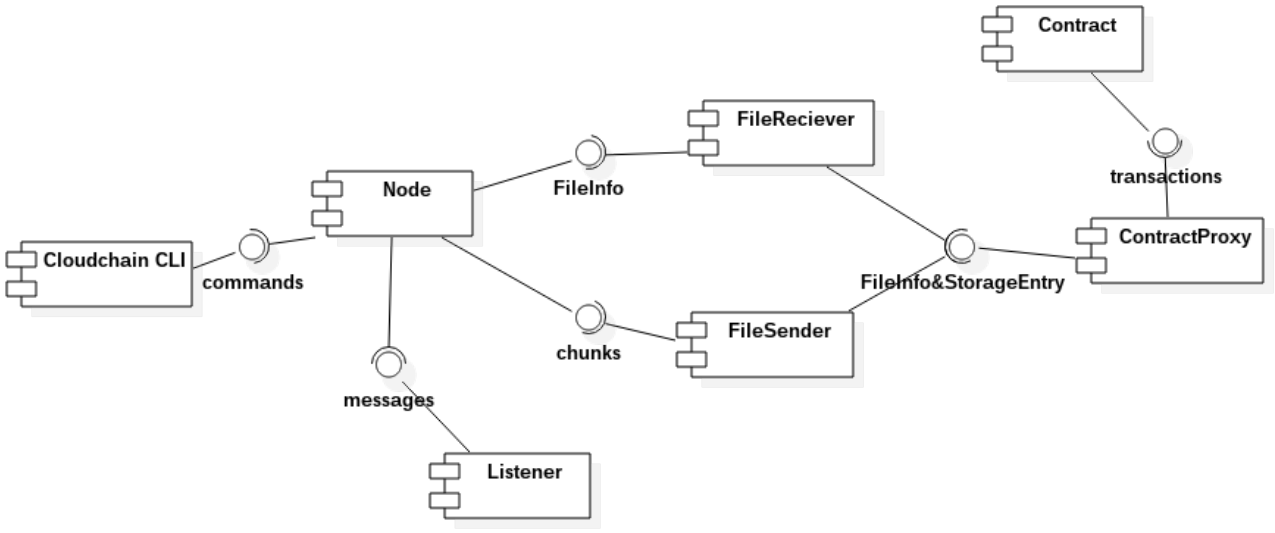

Figure 3: Component Diagram for CloudChain

`FileSender` and `FileReceiver` read from and write to the Ethereum blockchain by the help of `SignedHashTableProxy`, which is a subclass of base class `ContractProxy`. The data coming from the blockchain is encapsulated in `FileInfo` and `StorageEntry` instances.

The `ContractProxy` controls the transactions to and from the blockchain. It caches the metadata table on the blockchain so that unnecessary transactions are avoided. Also it has blocking capabilities to force synchronicity, via *event* mechanisms supplied by Ethereum contracts. The sequence diagram of this process is shown in Figure 4.

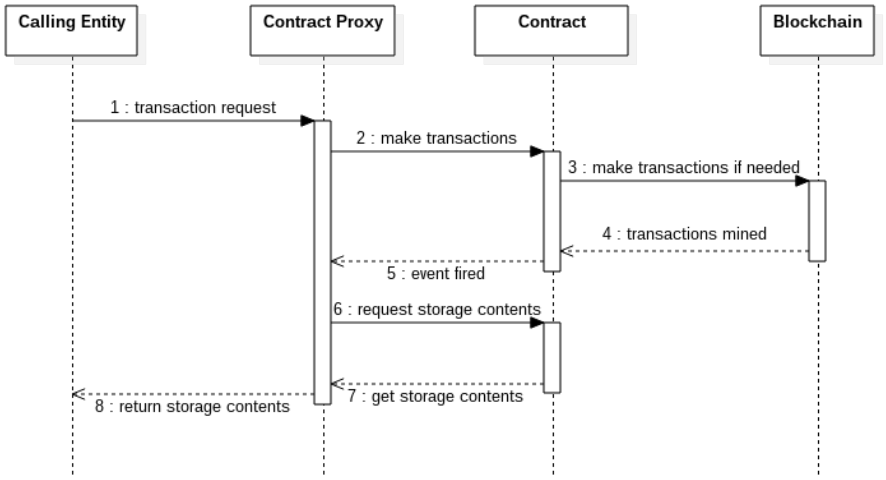

Figure 4: Sequence Diagram for the Blocking Mechanism

After a transaction request is made to the proxy, the proxy blocks the call until the transaction is mined if requested. There are events coupled in the contract for each storage altering function

of the contract. For example, the `SignedHashTable` contract has an event coupled to the function for registering entries to the table. After the transaction is made with the contract, the verification takes some time due to the nature of the blockchain (mining, distrubution of blocks in the network etc.). After the validation of the transaction, the contract fires an event which is listened by the `ContractProxy`. Upon receiving the event, `ContractProxy` pulls the modified area in the storage, updates the cache, and returns the transaction result.

3.2 Deployment

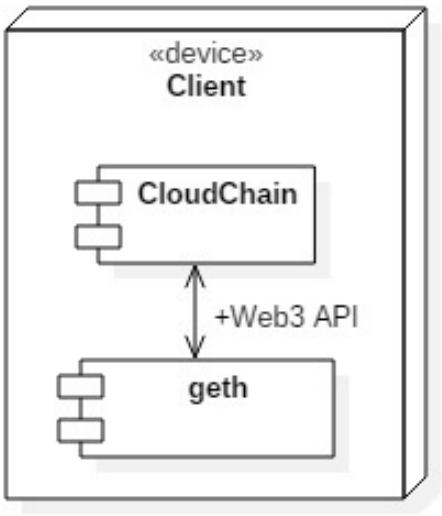

Figure 5: Deployment Diagram for CloudChain

As mentioned before, CloudChain is distributed. So, there are no servers. The only device on which CloudChain is deployed is client PC.

Two programs needs to be deployed to devices: CloudChain and its dependency geth[2]. geth is the official client for interacting with Ethereum network. It allows the user to make and examine the transactions, also to do mining.CloudChain interacts with geth via Web3 API, specifically its Python wrapper web3.py[3].

4 Alternative Design Options

4.1 Custom Blockchain

Implementing a custom blockchain technology as opposed to using an existing smart contract platform like Ethereum was an option and it could have several advantages such as:

- Being domain specific and not necessarily being Turing-complete could reduce the attack surface for security bugs.
- Data storage could better be optimized for specific needs of CloudChain, making the blockchain layer more scalable.

However, we chose not to implement our own blockchain for reasons such as:

- Implementing a blockchain itself is a difficult task that requires a lot of effort and we wanted to focus our effort not on building another blockchain but building a robust file storage network which uses blockchain for some of its core functionality.
- By not being a developer but being a user of the blockchain technology, we could still benefit from the technological improvements in the space without needing to expend extra development effort, provided we keep a simple and clear interface between CloudChain and the blockchain.

4.2 Partial File Storage

Currently, when a file is stored in multiple hosts, the file is stored in its entirety in each of the hosts. An alternative option would be to make it so that it is possible to store a fraction of the chunks in a single host and spread the file across multiple hosts. However, this required us to store a mapping between each host and chunk hash on the blockchain, which would significantly increase the size fo the data on the blockchain, which is why we discarded this model for a simpler one.

References

[1] Ethereum, a blockchain framework. Online: https://github.com/ethereum/wiki/wiki/White-Paper (Retrieved on 01.20.2018)

[2] Ethereum Go Client, go-ethereum (geth). Online: https://github.com/ethereum/go-ethereum (Retrieved on 01.20.2018)

[3] Web3.py, an RPC interface for geth. Online: https://github.com/ethereum/web3.py (Retrieved on 01.20.2018)