EtherLend Report

JingChang Sun, Ching-Hui Lin, ZhengYang Song

January 3, 2017

Abstract

We built a *decentralized application* (DApp) prototype based on the rule of *Rotating Savings and Credit Association* (ROSCA), with a combination of technologies such as Solidity, Truffle, Testrpc, Node.js and Android. We write Smart Contracts on the Ethereum platform implementing core functions such as member management, voting and auction. Testrpc works as a Blockchain client for developers so that we do not to spend Ether or wait for the transaction blocks to be mined. Truffle makes the compilation and deployment of Smart Contract written in Solidity easier. The end users only need to interact with the Android client. The Node.js server works as a interface for communications between the Android client and BlockChain client. All the core code is available at Github repository: https://github.com/Sun-Jc/EtherLend.

1 Introduction

A lot of students may need money beyond their ability for a little while. There are also many platforms providing these kind of services. However, the unclear interest rate and amount limit have been a major concern to potential users. We propose the method to combine the traditional *Rotating Saving and Credit Association (ROSCA)*[1][BCL93] with the new *Blockchain* [Swa15] technology. By doing so, all the information (such as members, duration, principal, interest rate) about a particular meeting is stored on the Blockchain, public and unchangeable.

Our app users can check all the available meetings, apply for one based on their relationships, wait for the manager to identify their qualification. Then they can make a suggest or vote for the duration of one round of meeting, hold a second price auction of their interest rate and reveal the result. After that, the users can jsut sit around with everything going as expected.

This report is organized as follows: we first show how the ROSCA mechanism works in Section 2. We introduce the technologies we used one by one: Smart Contract in Section 3, Testrpc and Truffle in Section 4, Node.js server in Section 5, Android client in Section 6. Last but not least, we wrap everything up and show how the final app works in Section 7.

2 ROSCA

The Rotating Saving and Credit Association is a way that close people help each other by forming a small money sharing group. There will be several rounds of meetings where they bid for how much they promise to give back to others in the future. The one gives the highest offer will win the bid and take money from everyone else, with the consequence that he can not participate in later bids but only pay money he has promised. Every one in the group will take turns to take money from others and give back. In the end, someone will gain and someone will lose, with the total amount of money unchanged.

The circumstance may be clearly clarified with a concrete example. Let us take a look at the following table representing a meeting with three people and lasting for three months:

	Alice	Bob	Charlie	Sum
Month 1	+16(3)	-8(2)	-8(1)	0
Month 2	-10	+19(2)	-9(1)	0
Month 3	-10	-10	+20	0
Sum	-4	+1	+3	0

[1] https://en.wikipedia.org/wiki/Rotating_savings_and_credit_association

Say Alice, Bob and Charlie are good friend and they decide to form a ROSA. The principal is 10 dollar. In Month 1, they bid for what they would pay for others in the future. Alice bids 3, Bob bids 2 and Charlie bids 1. Since we use second price auction, so the one with the highest bid wins and pays for the second highest bid. In our case, Alice wins and promises to pay Bob and Charlie 2 in the future. So Bob and Charlie take out 10-2=8 and Alice holds their money. In Month 2, another meeting takes place. Since Alice has won once before, so she has no chance to bid, while Bob bids 2 and Charlie bids 1. So Bob wins and takes 10 from Alice and 9 from Charlie. In month 3, no bidding happens. Charlie just take what Alice and Bob pay.

Note that in the end Alice gives out 4 and Bob takes 1, Charlie take 3. That corresponds to the principle that who uses money earlier, who pays more.

3 Smart Contract

We implement our core functions on the Ethereum [Woo14] platform. In other words, we encode all the rules in smart contracts [B+13] using Solidity. Solidity [LCO+16] is the main stream language to program smart contracts similar to JavaScript, while there are also other niche languages such as Serpent [DAK+15] similar to Python. We program the rules of ROSCA into code, compile it, and deploy the binary to Ethereum platform. There are in total 7 kind of contracts (some of which are abstract). The API between contracts and users are roughly as follows:

- Traits: it contains the membership of all the contracts and defines all the interfaces.
- Service: it is the main service contract, which implements the function of applying for a meeting for a meeting manager. If that succeeds, it will return an address (a unique identifier) of the applied meeting.
- Meeting: a manager can set the recruiting deadline of the meeting, a member can apply for joining an existing meeting, give suggestion on the period and rounds of the meeting, also the manger can set auction time for the meeting.
- Vote: members can vote for the suggestion and decide whether they will accept or deny it.
- Auction: it implements the function of a sealed second price auction, where each one give a bid without revealing it the others. The highest bidder wins and pays for the second highest bidding price.
- Credit Recorder: to add bad credit record of a particular member, in case one break his vowel. Also, we can check the credit score of a particular member.
- Timer: a self-implemented timer to keep track of all the timing events, make sure that every event happens at previously set time.

4 Ethereum Client

All the smart contracts are supposed to be compiled and deployed on the Blockchain supported by Ethereum. Once deployed, the contract will occupy a particular unique address. Then users can interact with the contract through Blockchain client and the deployed address, for example, send some Ether to the contract or call some functions in the contract. Ether is the cryptocurrency used by the Ethereum community. Existing Ethereum client includes go-thereum implemented in Go language, Parity in Rust, pyethapp in Python and so on [2]. We choose Testrpc [3] as our Ethereum client while developing with the following reasons:

- It does not cost real Ether when you run your smart contract in Testrpc, naturally you need to pay one Ether for one step of the execution of your program.
- You do not need to wait for the blocks containing the transactions to be mined by the miners in Testrpc, naturally it will take 3-5 minutes for a brand new block to be mined and appended to the real Blockchain.

[2] http://ethdocs.org/en/latest/ethereum-clients/choosing-a-client.html
[3] https://github.com/ethereumjs/testrpc

The brilliant work Truffle [4] makes the compilation and deployment of a smart contract into just a simple line of command, i.e. 'truffle compile' and 'truffle migrate'. The existence of Testrpc and Truffle makes our life much easier.

5 Node.js Server

Due to the lack of a light client, which means we can only load the whole Blockchain to our local machine if we want to access the data stored in it. While it might not be too hard for a laptop, an Android phone can not afford the storage space and processing power for doing this. Our way around is to add a interface between the Android client and Ethereum client, named the Node.js server. We use the router package 'express' [CHHR14] to map HTTP request of URLs to executions of functions, and encapsulate the result into a JSON object. We use the web3 [5] interface of JavaScript to communicate with our Ethereum client. We adopt the RESTful principle [RR08] when designing our service APIs. A full list of what we have implemented is as follows:

- /accounts: return all the current available accounts
- /balance/account_id: return the balance of the particular account
- /service: return the deployed service contract address of EtherLend
- /meetings: return all the current available meetings
- /events/meeting_address: return all the events past associated with the meeting address
- /apply/manager_id: apply for a new meeting as a manager, return the new meeting address
- /set/meeting_address/manager_id/recruit_end/duration
- /join/meeting_address/member_id
- /accept/meeting_address/member_id/manager_id
- /suggest/meeting_address/member_id/duration/amount
- /vote/meeting_address/member_id/aye
- /bid/meeting_address/member_id/round/bid/amount
- /reveal/meeting_address/member_id/round/amount

A simple visit of `host:port/events/0xaf8c445d426eb6cf0c6def9d15898509b7816baa` in browser will give you what a brand new meeting state should look like:


```
{
    "events": [
        {
            "logIndex": 0,
            "transactionIndex": 0,
            "transactionHash": "0x256c644c18c36ca43322f483b9a40b9210369f479773dd498ae
            "blockHash": "0x77bd43204bc6f3b148cbb087ab19d2d6878c0e5504f8326f12aa083fb8
            "blockNumber": 864,
            "address": "0xaf8c445d426eb6cf0c6def9d15898509b7816baa",
            "type": "mined",
            "event": "Established",
            "args": {
                "startTime": "1482823535",
                "manager": "0x428ae56b16688dd6852e34e84759fa385382bbc3"
            }
        }
    ]
}
```


[4] https://github.com/ConsenSys/truffle
[5] https://github.com/ethereum/web3.js/

6 Android Client

The Android app is what the end users can really operate. We bind listeners to the buttons, which will send requests to the API services we mentioned in Section 5, and wait for the JSON response. The UI are all under the guidance of material design [6]. You can see the video demo to get a better comprehension of what you can do with the Android client.

7 Conclusion

All in all, we realize the core functions of membership management, voting process and second price sealed auction on the Ethereum platform. Users can interact with an Android client to complete all the processes needed as a member of one ROSA. This is composed of technologies as Solidity, Truffle, Node.js and Android. All the code, Android apk and demo video are available on our GitHub repository `https://github.com/Sun-Jc/EtherLend`. Future works may include:

- Utilize the SPV approach (Ethereum mobile) rather than a intermittent Node.js server
- Strengthen accounts security, employ users' private keys more intelligently
- Build a solid and reliable credit system to avoid bad debt

References

[B+13] Vitalik Buterin et al. Ethereum white paper, 2013.

[BCL93] Timothy Besley, Stephen Coate, and Glenn Loury. The economics of rotating savings and credit associations. *The American Economic Review*, pages 792–810, 1993.

[CHHR14] Mike Cantelon, Marc Harter, TJ Holowaychuk, and Nathan Rajlich. *Node. js in Action.* Manning, 2014.

[DAK+15] Kevin Delmolino, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi. A programmer's guide to ethereum and serpent, 2015.

[LCO+16] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 254–269. ACM, 2016.

[RR08] Leonard Richardson and Sam Ruby. *RESTful web services.* " O'Reilly Media, Inc.", 2008.

[Swa15] Melanie Swan. *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.

[Woo14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

[6] https://developer.android.com/design/material/index.html