JJLend: a reputation-based smart lending system using Ethereum blockchain network

Jun Zhao, *Electrical and Computer Engineering*
Yuwei Jiao, *David R. Cheriton School of Computer Science*

***Abstract*—Blockchain has emerged as a distributed decentralized transaction ledger on peer to peer network, and has inspired hundreds of innovative applications extending to political, financial and social areas. Virtual currency is regarded as the first generation of blockchain revolution. Inspired by the development pattern of capital and assets in real life, the next step of mature currency should include a secure and stable lending platform. Therefore, we built a new system on Ethereum network, supporting peer-to-peer lending intelligently and improving the usability of virtual currency. Our system, JJLend, allows users to transfer digital currency via network, make borrowing requests or lending offers to the pubic, and collect blockchain ledge data to evaluate users' credit history as well as transaction risks using some machine learning methods. We also provide a web user interface to visualize this process. We believe it is a valuable exploration in blockchain area. The concepts as well as the implementation of JJLend could help with development of digital currency.**

***Index Terms*—blockchain, Ethereum, cryptocurrency, smart contract, machine learning**

I. Introduction

VIRTUAL currency, especially cryptocurrency such as Bitcoin and its derivatives has gained much attention in recent years. These digital currencies is performing their roles not only as fundamental trading currencies, but also as investment assets which are much similar to gold in global market. The price of Bitcoin has skyrocketed recently. Just a few weeks ago the price broke the $11,000 barrier, while three years ago one Bitcoin was worth only $400 USD. Reviewing the history of financial development, individuals tend to increase the value of spare money or asset they have, so there comes banks, stocks and other financial activities and authorities. For those digital asset holders, the absence of such stable and reliable platforms makes it hard to invest for profit. On the other hand, the market demands of such virtual currency cannot be satisfied in time, which would have a negative effect on the currency liquidity.

Bitcoin mainly relies on the blockchain technology, which was first described in the early of 1900s. At the very beginning, blockchain refers to cryptographically secured chain of blocks. Essentially, it is the the first generation of blockchain applications. Later people realize the concept of blockchain could be used to solve more practical problems, for example, voting. For the convenience of fast developing decentralized applications, several blockchain platforms, like Ethereum[1] [2], have introduced the idea of Smart Contracts. These decentralized application platforms enable designers, programmers and developers to focus more on the application itself, instead of underlying network communications.

Therefore, it is believed that after Bitcoin and Smart Contracts, crypto lending is a new revolution on Blockchain[3]. Some lending platforms, like SALT, WeTrust and ETHLend, are trying to challenge the traditional banking institutions with faster, cheaper and more secure services. This is a complex but very interesting problem, and we would like to explore and go deeper into this direction. Particularly, we want to combine machine learning with blockchain technology and create more possibilities. We suggest to design and implement a platform called JJLend, which supports peer-to-peer borrowing and lending to make digital funds available to another. Users could benefit from JJLend because lenders can expect higher interests while borrower can overcome financial difficulties. Besides, the market liquidity is enhanced to some extent. To ensure the scalability and security of such service, we choose Ethereum and take the advantage of its smart contract functionality. We provide convenient interfaces to access the blockchain ledge data, which allows applying advanced machine learning models to evaluate lending risks based on users credit level and transaction history. In general, we provide a reliable, agile and intelligent service to virtual currency users.

In this paper, we will first give some background information about the technology (blockchain) and the platform (Ethereum) that JJLend uses, as well as a few developing lending applications such as ETHLend in section II. In section III, we introduce our system design, features and functionality in detail. Section IV describes the main implementation methodology. We carried out several experiments and evaluate our system in terms of reliability and performance, which is analyzed and discussed in section V. Finally we list future work for JJLend and make several conclusions in section VI and section VII respectively.

II. Related work

A. Blockchain

Blockchain could be defined as "a cryptographically secured chain of blocks"[4][5] to facilitate secure online transactions[6]. Essentially, blockchain can be considered as a decentralized database that is shared by all network members. By storing data across the network, blockchain history records

This paper is the final report for ECE 750: Distributed and Network-Centric Computing course project submitted on December 18th, 2017.

J. Zhao is a second-year graduate student in Electrical and Computer Engineering at University of Waterloo (e-mail: j283zhao@uwaterloo.ca).

Y. Jiao is a second-year graduate student in David R. Cheriton School of Computer Science at University of Waterloo (e-mail: jyuwei@uwaterloo.ca).

cannot be changed and thus blockchain eliminates the risks of network collision and fraud activities[7][8][9]. In blockchain, each block contains a number of transactions and the hash code of its previous block, which forms an abstract chain of blocks. It serves as a transparent ledger which can be shared with the nodes in the network. This decentralized and distributed digital ledger can also reduce the costs to verify and audit transactions compared to centralized copies. The transaction data is verified and updated by miners, then the valid data can be replicated across the whole network. In this way, the blockchain data can be synchronized within the network and controlled by nobody[10].

There are mainly two types of consensus protocols that govern the blockchain: Proof of Work and Proof of Stake. The Proof of Work consensus protocol is used in the Bitcoin[11][12] and Ethereum blockchain[13][14]. This consensus protocol makes use of puzzles which are difficult to solve, but very easy to verify the solution. In order to create new blocks to the chain, it requires computational power to solve the puzzle[15]. When the majority of CPU power is controlled by honest nodes in the network, Proof of work protocol can make attacks to the blockchain computationally impractical by simply agreeing to accept the longest chain as the correct one[16].

B. Ethereum

In the blockchain context, we are now extending currency and payment transactions to the more complicated applications such as recording and transferring of more complex assets like smart property and smart contracts, which needs a more robust scripting system [10].

Ethereum is a blockchain-based distributed computing platform, designed for smart contract functionality, which is also a Turing-complete programming language for building distributed applications[17]. Ethereum serves as a foundational general-purpose platform to implement the paradigm of a transactional singleton machine with shared-state[18].

Ethereum is so far the most well-known and used framework for smart contracts[19]. It allows to execute scripts in Solidity language[20] either on user-defined private network or public main network (Homestead). There are several important components in Ethereum architecture. One is the virtual machine which provides to set up a runtime environment for smart contracts. Ethereum has two transaction pricing mechanisms: ether and gas. Ether is the crytocurrency used on the platform, while gas measures the computational resources consumed to perform specific transaction operations in the network. Solidity is a smart contract-oriented programming language, which is used on Ethereum to compile and deploy smart contracts on P2P network[21][22].

C. Current Lending Platforms

Before proposing our lending system, we select three typical implementations among currently existing applications, and briefly discuss their system characteristics.

1) SALT: The SALT Lending platform is a membership-based lending system, which enables cryptocurrency holders to carry out cash loans with crypto collateral tokens[23]. The first testing version of SALT Loan was released in March 2017.

2) WeTrust: WeTrust is an autonomous and frictionless collaborative savings, lending and insurance platform. It creates a full-stack alternative financial system utilized Ethereum blockchain and social capital networks. Besides, they provide WeTrust platform SDK for developers to build decentralized applications using WeTrust public data and tools.

3) Ethlend: Ethlend provides crypto lending with confidence of security. It creates reputation based lending consisting of data from decentralized credit tokens and identities[24][25]. Ethlend also takes the advantages of Ethereum with tokenized value and collaterals. Credit risk assessment is one of the most impressive features of Ethlend.

III. Design

In this section, we introduce our lending system features and design details. First, we list the main features and services of our JJLend system to give a general idea of the system functionality. Second, we present the high level design of the system architecture and its components. Third, each component in the lending system will be introduced with more details. Finally, we present some additional features of our JJLend lending system.

A. System features

As a decentralized lending application, JJLend platform mainly provides the following features:

- Users can buy and sell the value token on the JJLend platform using *ether*;
- Users can transfer the value token with other user or organization;
- Users can register their assets for collateral token;
- Borrowers can use their collateral token to apply for a loan;
- Lenders can provide funds for lending premium;
- Lenders can post lending offer on the platform;
- Borrowers can apply for loan posted on the JJLend system;
- Borrowers can earn credit token when paying back the loan in time;
- Borrower's credit will be burn down in case of defaulted loan;

In terms of lending workflow, we distinguish two kinds of lending services:

- Borrower-request-based lending: the lending relationship between the borrower and the lender starts from the borrower's lending request and the loan agreement is reached when the lender accepts the borrowing request.
- Lender-offer-based lending: the lending relationship starts from the lender's lending offer and the loan agreement is reached once the desired borrower is selected by the lender.

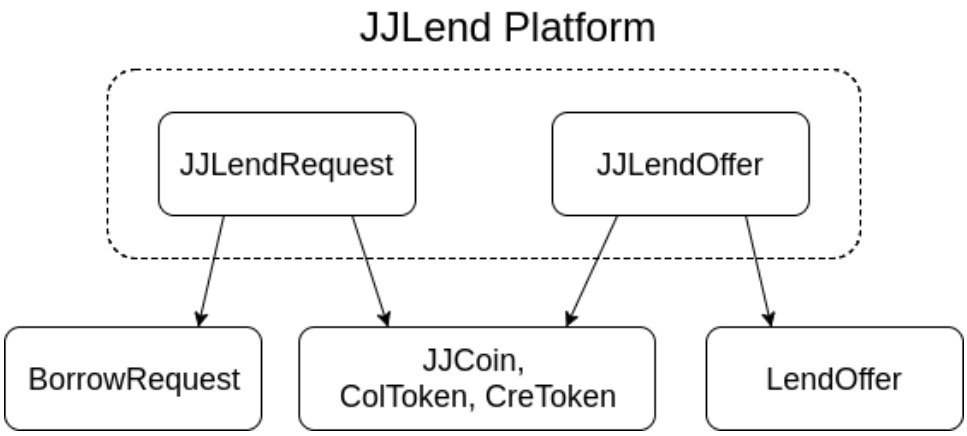

Fig. 1. JJLend architecture

B. JJLend architecture

Based on the system features and lending services, we construct our JJLend platform by combining two components: JJLendRequest and JJLendOffer. The overall architecture is presented in Figure 1. On the higher level of the platform, JJLendRequest subsystem processes the Borrower-request-based lending requests, while the JJLendOffer subsystem processes the Lender-offer-based lending requests. On the Ethereum platform, the contract code and data needs to be first mined into the contract creation block for further execution. The block size is limited by the block *Gaslimit*. Therefore, we choose to separate the two lending workflows to make the platform more scalable and easier to extend horizontally.

On the lower level of the system, the JJLendRequest and JJLendOffer components depend on some other basic components. Common used components are JJCoin, ColToken and CreToken, which provide value token, collateral token and credit token transactional services. Additionally, JJLendRequest subsystem makes use of BorrowRequest component to keep track of the contract information related to Borrower-request-based lending, while JJLendOffer subsystem depends on LendOffer component to process the Lender-offer-based lending requests.

C. Component design

1) Basic digital tokens: We create three types digital tokens in our Ethereum blockchain network to support the upper level lending services.

- JJCoin
  It is the native crypto-currency used for lending in our network, which is analogous to the ether used on the Ethereum platform. The name of this value token, JJ, mainly comes from the WatIAM userids of the group members.
- ColToken
  This token stands for Collateral Token, which represents the value as a collateral for borrowing. Each collateral token can correspond to an object in reality, like a car, a computer or even certain amount of USD. It can be considered as the projection from real world to our virtual network. It serves as a borrowers pledge of specific property to a lender, to ensure repayment of a loan.
- CreToken
  This token stands for Credit Token. In order to handle reputation-based lending, CreToken is used to quantitatively evaluate the borrower's reputation based on the previous lending history on JJLend. Credit token cannot be traded or transferred to other user. For each repaid loan, the borrower is rewarded by the CreToken, which can be used to attract the lenders with less credit risk.

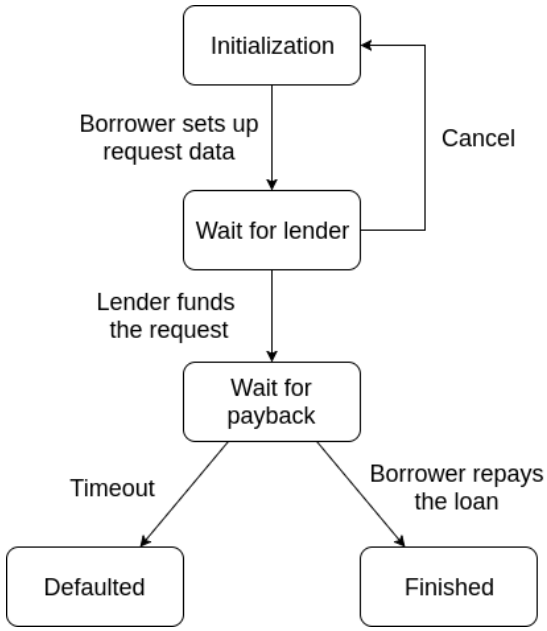

Fig. 2. Borrower-request-based lending workflow

2) Borrower-request-based lending components: As the subsystem of the JJLend platform, the JJLendRequest component provides borrower-request-based lending service allowing the borrowers to place collateralized loan requests on the network. The main workflow is shown in Figure 2. Some key steps are listed as follows:

1) The borrower creates a new loan request by setting up

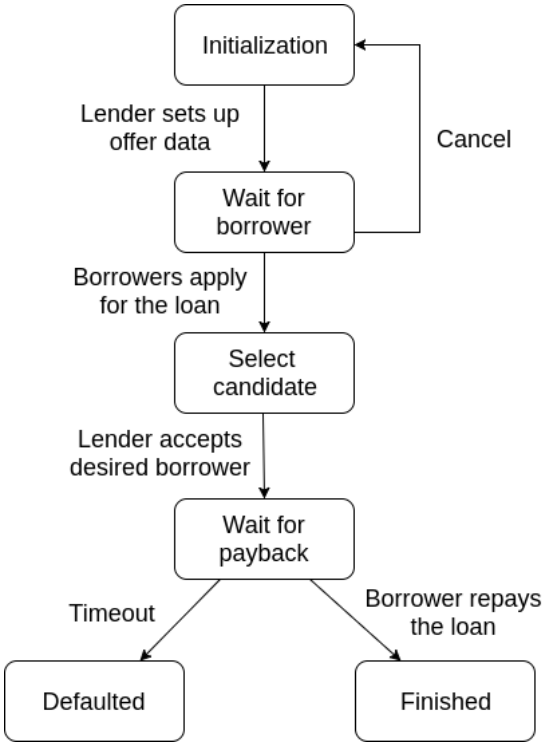

Fig. 3. Lender-offer-based lending workflow

the loan data, including the amount of JJCoin, loan period, premium rate etc.
2) The borrower transfers valid ColToken to the newly created loan request as a collateral in case of default.
3) The borrower broadcasts the loan request to the blockchain network and waits for the lender to fund this request.
4) Once the lender approves the loan request and sends specific amount of JJCoin to the borrower, the agreement between the borrower and the lender is reached. Then the borrower can make use of the loan during the loan contract period.
5) By the end of the loan period, we can have two possible scenarios:
   - If the borrower repays the loan in time, it can gain certain amount of CreToken and the lender gets back its JJCoin and premium;
   - Otherwise, the lender gets the loan collateral and burns the borrower's credit balance.

3) Lender-offer-based lending components: As for an alternative for the borrower-request-based lending, JJLendOffer subsystem provides lender-offer-based lending service where the lender can place the loan offer for the borrowers, which provide flexibility for the lender. The main workflow is shown in Figure 3. Some key steps are described as follows:

1) The lender creates an offer including loan data such as how much JJCoin the lender is willing to lend, required premium, which kind of ColToken the lender is willing to accept as collateral.
2) The lender deploys the offer on the network and wait for the borrower to apply for this offer.
3) Multiple borrowers can apply for the loan offered by the lender and present their collateral token for borrowing.
4) Based on the lending history and credit balance of the borrowing candidates, the lender select the desired borrower. Once the lender accepts the borrower's application, the loan agreement can be reached.
5) By the end of the loan period, we will have the same scenarios as described in the borrower-request-based lending.

TABLE I
ETHEREUM PLATFORM TOOLS

	Tool	Version
Client	go-ethereum	1.7.3
Language	Solidity	0.4.18
Framework	Truffle	4.0.1
IDE	Remix	online
Browser extension	MetaMask	3.13.3

D. Additional features

In order to improve the user experience, JJLend platform contains a web application user interface, which includes features such as active request dashboard, view history, account settings etc. The active request dashboard allows users to browse all requests waiting for borrower or lender and make transactions. History panel lists all borrowing and lending activities for user, where user can also start a new request or cancel a pending request. The account settings page can present more lending details such as CreToken value, account balance and ColToken status.

In addition to customized user interface, JJLend system also provides interfaces allowing users to collect activity logs on the blockchain and perform risks estimation using machine learning techniques. The first stage of this analyzing pipeline has been completed. We are able to automatically generate training data on the blockchain and download the transaction data from the Ethereum network for further machine learning analysis.

IV. IMPLEMENTATION

Our lending system is implemented on the Ethereum blockchain platform. Related tools used in the project are shown in Table I.

We choose to use *go-ethereum*, usually called *geth*, as our Ethereum client, which is one of the most popular clients used in the Ethereum development community. As for the contract development, we use the contract-oriented language, *Solidity*, to implement our smart contracts. In order to deploy and test the smart contracts on different Ethereum networks such as *Ganache* and *testrpc*, the *Truffle* framework and the online IDE *Remix* are used in our project. Finally, we use *MetaMask* to connect the browser and Ethereum network for visualization.

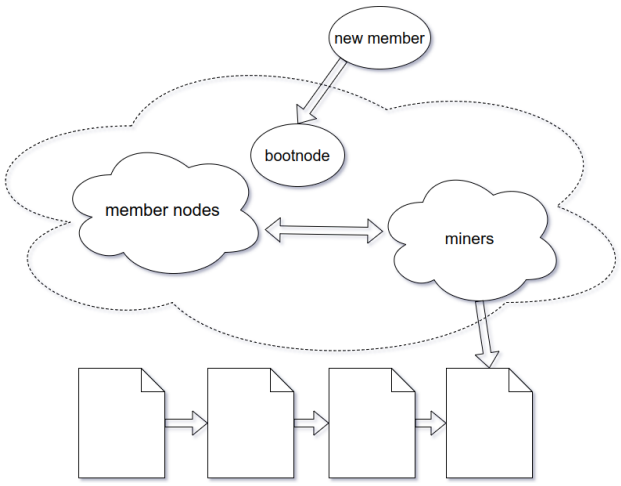

Fig. 4. Ethereum private network

A. Ethereum networks

Before deploying the smart contracts, we need to build up the Ethereum network. We choose to develop and test our decentralized application on a private network (2-5 nodes). This is because the smart contracts deployment, transactions and calls in the public Ethereum network do have a gas cost, paid by real ether, which is the value token of the Ethereum blockchain. As of June 2017, one ether is worth more than $400, which increases by 5000% since the beginning of the year. Besides, the first stated production release (Ethereum Homestead) still has a number of components that are not easy to integrate and use. Therefore, we set up a private test network to avoid real public use difficulties and simulate the blockchain mining environment.

For our own private Ethereum network, we set up 1 bootnode and 2-5 member nodes, up to 5 of which can be miner nodes. As a new node joins the Ethereum network for the first time, it will connect to the predefined bootnode using its *enode* address. The bootnode can help the new member node to find other nodes and start to synchronize the blockchain data.

As for mining in the private network, we set relatively small value for the *difficulty* option (currently using 40) in the genesis block to facilitate block mining. In this way, we are able to use CPU instances to perform mining operations and produce a stable stream of blocks at reasonable short time intervals. Our private network environment is illustrated in Figure 4. We mainly take the following steps to set up our private network:

1) Defining the genesis state of our network;
2) Creating the bootstrap node;
3) Starting up the member nodes;
4) Running the private miners.

Other than our own private network, we also make use of the *Ganache* and *testrpc* networks to facilitate the smart contract development and visualization. Each network has its own characteristics and serves as *web3* provider instance to deploy smart contracts and mine blockchain transactions. *Ganache* is part of the *Truffle* Suite, and it provides convenient visualization interface to see how the dapp affects the underlying blockchain. *testrpc* is a *Node.js* based Ethereum development tool, which uses *ethereumjs* to simulate full client behavior and accelerate transaction execution. The main network configurations are presented in Table II.

TABLE II
ETHEREUM NETWORKS

Network	Port	Network ID	Gas Limit
private	8101	1024	90000000
testrpc	8545	8888	4712388
Ganache	7545	5777	6721975

B. Smart contracts

The core of *JJLend* system consists of smart contracts. We use *Solidity* to implement our contracts. The *Truffle* framework is used to deploy and test the smart contracts on the Ethereum networks. We mainly have seven smart contracts in our lending system:

- JJCoin.sol
- CreToken.sol
- ColToken.sol
- BorrowRequest.sol
- JJLendRequest.sol
- LendOffer.sol
- JJLendOffer.sol

We will introduce and discuss the key implementation details in the development of these smart contracts.

1) JJCoin and CreToken: The *JJCoin* and *CreToken* contracts both extend the *Token* contract (Figure 5), which implements the *ERC20 Token Standard* interface. The *ERC20 Token Standard* is a set of 6 functions and 2 events that enable interoperability across multiple interfaces and distributed applications. From the decentralized point of view, *ERC20* mainly specify:

- how tokens can be transferred by its owner
- how tokens can be transferred on behalf of the owner
- how to access information about the token
- events about the token

By following the *ERC20 Token Standard*, we can ensure the transaction effectiveness and security in the decentralized environment. More specifically, *JJCoin* has more advanced operations like *buy* and *sell* to exchange *JJCoin* tokens with the use of *ether*, while *CreToken* overrides the *transfer* and *transferFrom* functions making the token non-transferable. In addition, *CreToken* provides interface to issue and burn down user's credit tokens, which would be useful in case of loan default.

2) ColToken: Unlike the *JJCoin* and *CreToken*, *ColToken* represents objects in the real world, and it needs an administrative agent to manage the complex relation between the virtual collateral token and the corresponding object in reality. It is relatively complicated to implement this IoT (Internet of Things) token in a fully decentralized way. So for simplicity's sake, we retain the centralized role (token creator)

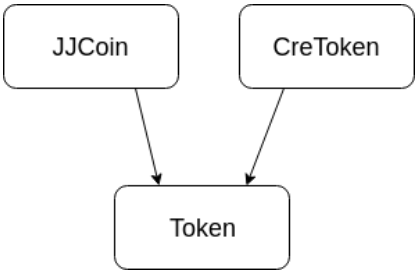

Fig. 5. JJCoin and CreToken implementation

in the contract to perform administrative operations such as registration and other token ownership management. However, the user transactional operations of collateral tokens are still fully decentralized as we adopt the *ERC20 Token Standard* interface to transfer the token.

3) JJLendRequest and BorrowRequest: The *JJLendRequest* provides borrower-request-based lending service. In this subsystem, we keep track of every account address related to the borrow requests, including borrower and lender information. For each account, we store its whole borrowing request history in the *mapping* Solidity data type, which would be recorded in the underlying blockchain. For each borrowing request, we create a *BorrowRequest* contract object to manage and keep track of the following loan states in the request contract life cycle:

- Init: empty loan request;
- Canceled: request is canceled by borrower before funded;
- WaitingForLender: loan data has been set up, including borrow amount, payback amount, expired period etc.
- WaitingForPayback: lender accepts the request and provides funding;
- Expired: loan is defaulted;
- Finished: loan is repaid in time.

All the request contract states are automatically updated whenever appropriate transactional operations are performed. The whole process is fully decentralized on the Ethereum blockchain without any trusted third party. Additionally, important transactional events, such as request creation, request acceptance and loan repaid, are fully recorded as the log information in the blockchain. This log information can be further used to estimate lending risk using machine learning techniques.

4) JJLendOffer and LendOffer: As for the lender-offer-based lending, the parallel subsystem *JJLendOffer* is implemented to provide this service. Similar to the *JJLendRequest* subsystem, *JJLendOffer* also uses the *mapping* data type to store lending offer history for each related account address. The *LendOffer* contract is used to manage different loan states. The only difference in implementation is that the lender can possibly receive multiple borrower applications for funding. Thus, we need to store all the borrowing candidate information, including the borrower address and the corresponding provided collateral in the *LendOffer* contract object. Finally, the lender needs to accept one candidate as borrower to reach lending agreement.

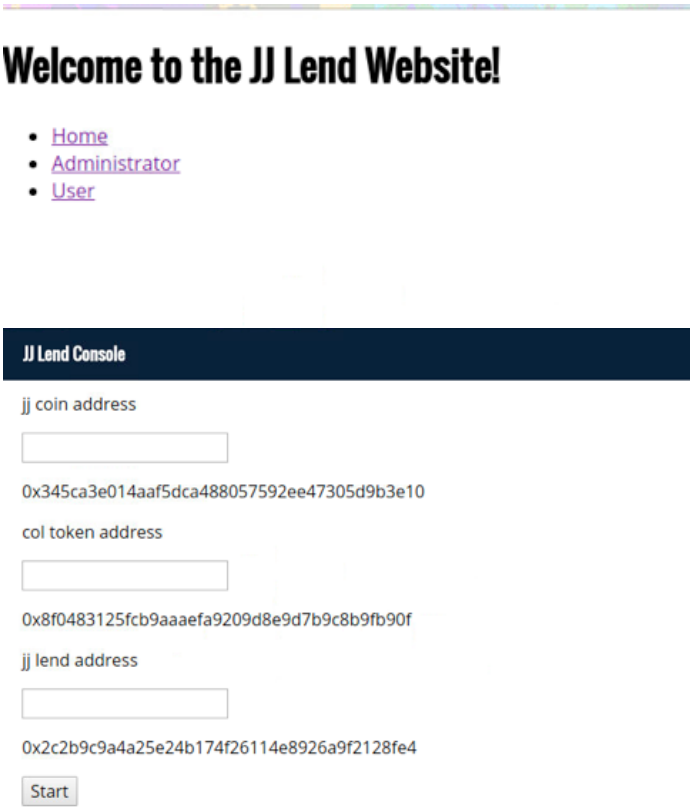

Fig. 6. User Interface of JJLend

C. User Interface

At the beginning, we use *Remix* to compile, debug and run. Remix is a web IDE for solidity and has an integrated debugger and testing environment.

It is easy to compile the smart contract code and create an instance on blockchain using Remix. However, the user interface is not friendly for developing an application because it gets complicated to switch between accounts, contracts and methods. For the ease of use of our lending application, we develop a front-end console page, helping JJLend administrator and users to operate.

Basically, we use *React* framework to build the user interface and interact with back-end blockchain network. *React* makes it easier to create interactive web components without the pain dealing with *DOM*. It is well supported by Ethereum now. Besides, in order to get authorization from Ethereum accounts in browser, we use *MetaMask* to control all testing accounts and have options to accept or reject incoming transaction requests. A few steps to build and run customized web-based blockchain application are as follows:

1) start the blockchain network and create contract instances.
2) use *npm* to compile and build *React* webpages.
3) import and unlock testing accounts to *MetaMask* with Mnemonic seed.
4) interact with the user interface and choose to accept or reject transactions.

There are three main components of our front end: *Home*, *Admin* and *User* (see Fig. 6). We use *React* router to direct between URLs. In the home page, we have some basic information about JJLend and provide with options to go with admin or user portal. The admin portal is designed for JJLend administrators, and here they could choose to cooperate with different bank and collateral authorities. The default setting is to use *JJCoin* and *CreToken* in our system. In the user page, we would like to allow users to view all available requests, make borrowing or lending actions as well as view their own history. Since it is quite heavy tasks to fully develop the UI, we only completed the feature to view all active requests. Continuously improving the web UI of our lending application would be planed in our next step.

D. Data Retrieval

In order to estimate lending risk and user's credit, we take the advantage of machine learning to make JJLend intelligent. Generally, there are three steps to apply machine learning to the system.

1) Collecting data from blockchain transactions;
2) Cleaning data, selecting features and training machine learning models;
3) Integrating results from machine learning models into the lending system.

For now, we completed the first stage. We use Ethereum *event*, which allows convenient EVM logging facilities. We wrote some scripts to generate data automatically, and we manage to get these logging data from the Ethereum network with use of the Truffle framework.

V. Experiments

In order to evaluate our proposed lending system, we design a set of experiments to mock several lending scenarios, including borrower-request-based and lender-offer-based lending. In the *Truffle* framework, we wrote several test scripts in JavaScript to automatically test all the lending scenarios and evaluate the system performance.

A. Borrower-request-based lending

For the borrower-request-based lending, we consider two typical scenarios: one is the successfully repaid loan, and the other is the defaulted loan. These two lending scenarios are described as follows:

- Scenario 1: the borrower requests to borrow 1000 JJCoin using his car as collateral. He promises to pay back 1100 JJCoin in 10 second (testing loan period). The lender accepts the borrower's request and lends him 1000 JJCoin. Finally, the borrower successfully repays the loan in time and gets back his collateral.
- Scenario 2: the borrower use his car as collateral to borrower 1000 JJCoin and promise to pay back 1100 JJCoin in 1 second. But he fails to repay the loan in time, so the lender gets the load collateral and burns the borrower's credit balance.

TABLE III
Average execution runtime of different lending experiments

Network	Scenario 1	Scenario 2	Scenario 3	Scenario 4
private	34.56s	27.73s	42.88s	40.29s
testrpc	0.952s	0.871s	1.223s	1.128s
Ganache	1.418	1.217s	1.717s	1.603s

We use automatic testing scripts to perform the transaction operations on behalf of the borrower and the lender. The JJLend system can pass all the test cases on three Ethereum networks where we deploy our smart contracts. The average execution runtime results (average of 10 executions, not including waiting time) are summarized in Table III.

Based on our experimental results, *testrpc* network has the best performance and the *Ganache* network is slightly slower that *testrpc* as it provides blockchain visualization interface. When it comes to the real mining environment, the execution runtime of test cases becomes much slower than the development networks. In our private network, we use 3 mining nodes with low mining *difficulty* value (set as 40), and the performance is much lower than the development blockchain networks.

B. Lender-offer-based lending

For the lender-offer-based lending, we also consider two typical scenarios: one is the successfully repaid loan, and the other is the defaulted loan. These two lending scenarios are described as follows:

- Scenario 3: the lender posts a lending offer of 1000 JJCoin and requires the borrower to pay back 1100 JJCokn within a loan period of 10 second (testing loan period). Two borrowers, A and B, apply for this loan offer, and the lender selects borrower B because his collateral is more valuable than that of borrower A. Finally, borrower B successfully repays the loan in time and gets back his collateral.
- Scenario 4: similar to Scenario 3, the lender posts the same lending offer on JJLend and selects borrower B. But borrower B fails to repay the loan in time, so the lender gets the load collateral and burns the credit balance of borrower B.

For the above scenarios, JJLend system can pass all the test cases on three Ethereum networks. The average execution runtime results (average of 10 executions, not including waiting time) of the above lending scenarios are summarized in Table III. The execution runtime are almost proportional to the number of transaction operations. Scenario 3 consumes more time than other lending scenarios as it has more transactions such as candidate selection and JJCoin transfer, while Scenario 2 has less execution time than other cases.

C. System performance

In order to evaluate JJLend system performance, we randomly choose two accounts to act as borrower and lender to perform borrower-request-based lending operations, which

TABLE IV
JJLEND SYSTEM THROUGHPUT OF REQUEST LENDING ON DIFFERENT ETHEREUM NETWORKS

Network	Throughput (/min)
private	2.36
testrpc	44.5
Ganache	38.1

is similar to Scenario 1. The performance test consists of 20 borrower-request-based lending operations. We run the performance test on different Ethereum network and obtain the system throughput in Table IV. Our system has best throughput performance on *testrpc* network. We try to increase the number of mining nodes in the private network, but the throughput performance can only be slightly improved.

D. Blockchain data retrieval for credit risk analysis

Finally, we also perform some experiments to collect lending transaction data for further credit risk analysis using machine learning techniques. One interesting scenario is that there are two users in the system keep borrowing and lending in turn and increase both CreToken balances. In this case, CreToken would become distrustful because everyone can find a way to earn CreTokens.

In order to distinguish this cheating behavior on CreToken balance, we define two types of borrowing and lending patterns to prepare the training data:

1) User A and B keep borrowing and lending with each other in turn for ten rounds;
2) Two users are randomly chosen out of ten users to act as borrower and lender to perform lending transactions, and this random process is repeated for fifty rounds.

After generating the experimental data on the Ethereum blockchain, we can use some scripts to collect the transaction information, including loan amount, duration, borrower/lender identity and request status, from the network and store the data to local file. We expect to continue with machine learning methods and classify these two transaction patterns so that we can evaluate user's credit differently.

VI. FUTURE WORK

A. Credit risk assessment using machine learning techniques

Currently, JJLend can provide interface to collect transaction data on the blockchain network. The next step to evaluate lending risk and user credit is to apply machine learning techniques to predict user's credit reliability. Several possible machine learning models are under consideration:

- Random forest
- Gradient boosted tree
- Supported vector machine
- Neural network

We plan to use Spark ML package for both training and prediction because of its built-in and ease-of-use features with Spark. Some other machine learning libraries, such as scikit-learn[26] and xgboost[27][28] might be useful depending on requirements in the future. With the evaluation results, our system would rank each request to meet various risk and interest needs from both lenders and borrowers.

B. Crowdlending service

One of the good-to-have features under consideration is crowdlending[29]. As described in the borrower-request-based lending and lender-offer-based lending, one loan request can be funded by one lender, and one lending offer can accept one desired borrower. Crowdlending allows multiple lenders to fund a single loan request to share risk with other lenders. The difficult point to implement this service is the collateral sharing problem. In case of defaulted loan, the loan collateral is owned by multiple lenders. A fair and effective mechanism to share the collateral will be our next step to consider to provide this useful service.

VII. CONCLUSION

We developed a fully decentralized lending application named JJLend on Ethereum platform. JJLend makes it easier for users to borrow and lend cryptocurrency without trusting third party. Our lending system cooperates with banking and collateral services to provide secure and reliable digital currency transactions. In this project, we build our own digital currency, JJCoin, which could exchange with *ether*. And we also build a collateral service to support our lending platform. With use of smart contracts to define transaction activities, it is easy to add new features to extend JJLend platform.

This is our first attempt to develop the decentralized application. We believe it is a valuable exploration in blockchain area and the implementation of JJLend platform could help with development of smart lending system.

ACKNOWLEDGMENT

The authors would like to thank Professor W. Golab for giving well-organized lectures, designing practical assignments and providing instructive suggestions throughout the course.

REFERENCES

[1] "Ethereum project." [Online]. Available: https://www.ethereum.org/
[2] "ethereum." [Online]. Available: https://github.com/ethereum
[3] E. Heilman, F. Baldimtsi, and S. Goldberg, "Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 43–60.
[4] R. Wattenhofer, *The science of the blockchain*. CreateSpace Independent Publishing Platform, 2016.
[5] "Blockchain," Dec 2017. [Online]. Available: https://en.wikipedia.org/wiki/Blockchain
[6] G. Zyskind, O. Nathan, *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *Security and Privacy Workshops (SPW), 2015 IEEE*. IEEE, 2015, pp. 180–184.
[7] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *International Workshop on Open Problems in Network Security*. Springer, 2015, pp. 112–125.
[8] A. Wright and P. De Filippi, "Decentralized blockchain technology and the rise of lex cryptographia," 2015.
[9] A. Kiayias and G. Panagiotakos, "Speed-security tradeoffs in blockchain protocols." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1019, 2015.
[10] M. Swan, *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[11] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to betterhow to make bitcoin a better currency," in *International Conference on Financial Cryptography and Data Security*. Springer, 2012, pp. 399–414.
[12] D. Yermack, "Is bitcoin a real currency? an economic appraisal," in *Handbook of digital currency*. Elsevier, 2015, pp. 31–43.
[13] Y. Hirai, "Defining the ethereum virtual machine for interactive theorem provers," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 520–535.
[14] H. Diedrich, "Ethereum: Blockchains, digital assets, smart contracts, decentralized autonomous organizations," 2016.
[15] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *International Conference on Principles of Security and Trust*. Springer, 2017, pp. 164–186.
[16] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[17] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, 2014.
[18] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.
[19] A. Mavridou and A. Laszka, "Designing secure ethereum smart contracts: A finite state machine based approach," *arXiv preprint arXiv:1711.09327*, 2017.
[20] C. Dannen, "Solidity programming," in *Introducing Ethereum and Solidity*. Springer, 2017, pp. 69–88.
[21] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*. IEEE, 2001, pp. 101–102.
[22] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*. IEEE, 2001, pp. 99–100.
[23] E. Duffield and K. Hagan, "Darkcoin: Peertopeer cryptocurrency with anonymous blockchain transactions and an improved proofofwork system," *Mar-2014 [Online]. Available: https://www. dash. org/wpcontent/uploads/2014/09/DarkcoinWhitepaper. pdf.[Accessed: 21-Sep-2016]*, 2014.
[24] D. Carboni, "Feedback based reputation on top of the bitcoin blockchain," *arXiv preprint arXiv:1502.01504*, 2015.
[25] S.-w. ZHENG and L. FAN, "Credit model based on p2p electronic cash system bitcoin [j]," *Information Security and Communications Privacy*, vol. 3, p. 040, 2012.
[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.
[27] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016, pp. 785–794.
[28] T. Chen, T. He, M. Benesty, *et al.*, "Xgboost: extreme gradient boosting," *R package version 0.4-2*, pp. 1–4, 2015.
[29] A. Ordanini, L. Miceli, M. Pizzetti, and A. Parasuraman, "Crowd-funding: transforming customers into investors through innovative service platforms," *Journal of service management*, vol. 22, no. 4, pp. 443–470, 2011.