Ethereum Election Voting Application

Priscilla Imandi	primandi@uh.edu
Abheeshta Reddy Jakka	ajakka@uh.edu
Naresh Kumar S Shanmugasundaram	nsaravanashanmugasundaram@uh.edu
Deepika Konreddy	dkonreddy@uh.edu
Rajeshwari Ravi	rravi@uh.edu

Project Report – Cloud Computing 6376 May 2018

1. INTRODUCTION

This project is focused on the design of a next generation voting system built on the foundation of the blockchain. Voters expect the voting process to be anonymous, but verification of candidate tallies should be possible. Votes should be impossible to tamper with and illegitimate votes should not be counted. All vote counting would be performed in a publicly observable way. In the present scenario, a voter cannot verify if his/her vote is counted appropriately, therefore voters have to trust the system. A similar problem was addressed with digital currency such as Bitcoin and Ethereum. On the blockchain, each group of transactions is hashed together, along with a hash of the previous block, and the entire blockchain would be publicly accessible.

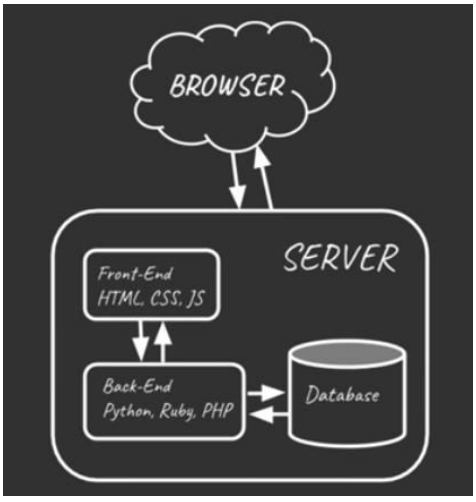
Using a blockchain for digital voting could record both voter and candidate ID, as well as the time. The voter IDs are a public/private keypair, not traceable to a voter's identity. Server are able to keep the votes off the main blockchain for the purposes of keeping the totals hidden until their release. Each vote is verified in a smart contract before being sent to the candidate or ballot measure.

In our system, the smart contract is that a "vote" is given to a candidate if it satisfies certain conditions, such as the amount being cast is equal to exactly one vote, verification that the server concurs that the voter wallet is valid, and that the vote occurred in a valid date range By combining elements of the cryptographic hashes, blockchain, smart contracts, and sidechains, it is possible to build an open, verifiable, and anonymous voting system for the modern world.

1.1 Web Application

The data is stored in the database which resides within the server. This server has a front-end system through which the user communicates with the database via the backend. This entire setup communicates with the client to perform its operations. But the web app has two major disadvantages

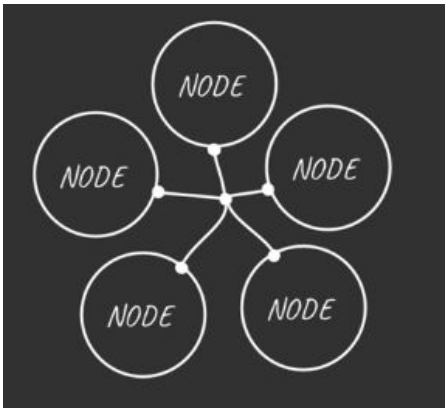
- Votes can change : Data in the database(our votes) can change by the centralized system without our control.
- Election rules can change : The code determining the process can change by the centralized system.



1.2 Decentralized App

Now, to avoid the above mentioned issues we can build a decentralized application. This helps ensure that a vote is counted only once as it will be transparent. The votes are not changed and only the deserving candidates will win the election.

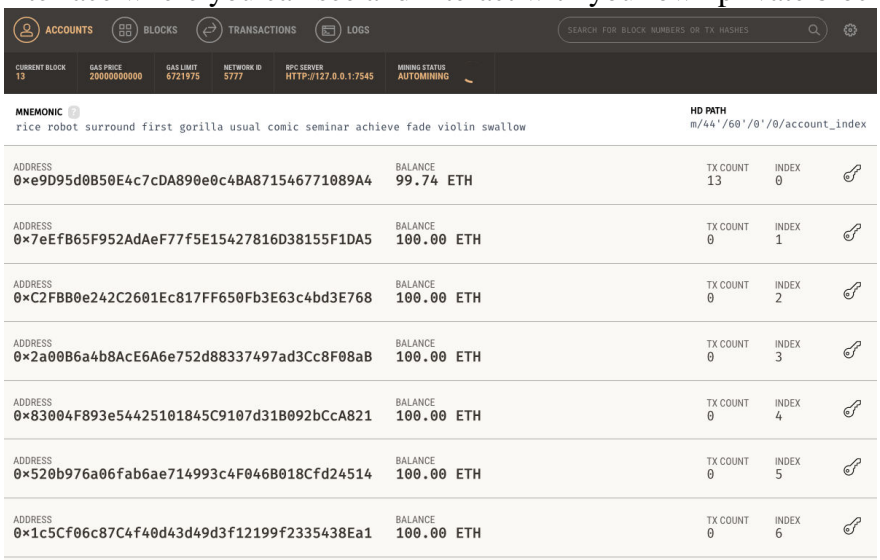
All the data doesn't lie on a single database, instead the data is decentralized and distributed throughout the network on all the nodes. This is a peer to peer network connected to the blockchain. All the data is stored in bundles of records called blocks which make up the public ledger called the block chain. Once a transaction is recorded on this public ledger it is public, secure and cannot be changed. This ensures that our votes are correctly counted. Any transaction has to adhere to smart contracts. A smart contract is a computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract.



They help exchange anything of value in a transparent conflict free way without a middleman.

1.3 Ganache

It is a local blockchain for development purposes. It emulates the blockchain so you can make calls to it without having to actually mine the blocks. The truffle team developed an Electron interface where you can see and interact with your own private blockchain.

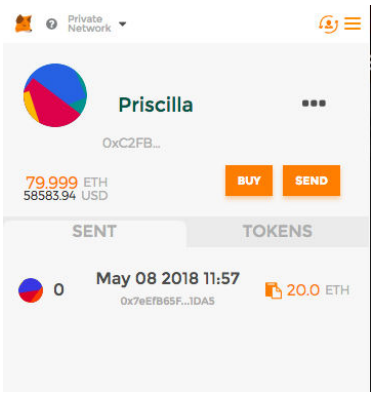


1.4 MetaMask

It is a bridge that allows us to run Ethereum decentralized applications in our web browser.

The Meta Mask browser extension enables browsing of Ethereum blockchain enabled websites.

We can now run the decentralized applications without running the entire Ethereum Node.



2. WORKING OF THE APPLICATION

2.1 Writing a Smart Contract With Solidity

Ethereum allows us to write code that gets executed on Ethereum virtual machine with smart contracts and this where the business logic of our application resides. A smart contract represents a layer that reads and writes the data in ledger and also communicates with other services. In the case of our application it is an agreement that counts the votes and does not allow an account to vote more than once. This voting contract is written in solidity and deployed on the local blockchain. In solidity, a contract works just like a class in java. Whenever the contract is changed or modified, it is given a completely new address to deploy on the local block chain Ganache, it specifies a port number with which we can access the smart contract.

2.2 Deploying a smart Contract

Truffle framework allows to create a decentralized application on the Ethereum network. It provides a set of tools to test and deploy our smart contracts to the blockchain network. We perform migration to deploy our smart contract so that we can interact with the application using truffle console or with the front-end of our application. On deploying a smart contract to network a certain amount of ether will be deducted as a fee. The balance gets deducted whenever the smart contract is deployed to the network. Smart contract address and the initial state of the contract can be retrieved in the truffle console.

2.3 Business Logic

In this project we write a smart contract where 5 Candidates are defined (participate in the elections). The candidate structure is created in such a way that it stores the attributes of the candidates that is, ID, NAME & NUMBER OF VOTES with their appropriate data types. Mapping is done to map the id to the candidate just like a key value pair and every time this is done, state of the contract changes. A function is used to add the candidates to the smart contract. To add the candidates to the smart contract, addCandidate function is used and it is kept private because it must not be accessed by anyone else but the contract. Functions and State Variables in the smart contract must be declared public to access them from the console. A count variable is used to know how many candidates are participating in the voting system. Also, every time a candidate is selected to cast a vote, the number of votes of the candidates is incremented. Before incrementing the vote of a candidate, its id is verified for authenticity.

2.4 Casting A Vote

Any person connected with the blockchain can cast a vote from their own systems. Voters are going to be anyone who has an account in the block chain. In our experiment we can run it locally using Ganache, which provides us with 10 accounts for developer testing with various accounts address. Each address in Ganache can be used to cast a vote from a different person. We can cast the vote using truffle console or we have also created a front-end, which interacts with the smart contracts displaying the candidates name, option to vote and count of votes that each candidate has obtained. Web3 is a library used to access these accounts from the truffle console. Web3 is a client-side application. It is used to connect the html file to the smart contract. Metamask extension in our browser provides us with web3 provider by connecting to our local blockchain instance.

This happens in three steps: Initializing web3, Initializing Contract and Rendering the Contract.

2.5 Testing the Smart Contract

We all know that code on block chain is immutable, if there are any bugs we need to redeploy

our smart contract, which will not have the same state as old contract and to minimize the ether spent for deploying often. Testing our smart contracts is important to ensure that it is bug free. We write various test cases for it. Truffle provides framework like mocha and chai to write tests for our smart contract. We write test cases to check for the number of candidates participating in the electing and whether they are assigned with unique ID. We also need to check for invalid candidates and ensure that an account can vote only one.

3. CHALLENGES

There was an issue with installing Ganache and Truffle on a Machine running Windows Operating System, the versions and dependancies varied and were not consistent with each other. Whereas, the working on a Machine using Mac operating system was smoother without much inconsistencies in versions. There were a lot of dependancies that had to be installed like node, npm and multiple other requirements which increased the complexity of the working.

Also, the metamask extension works for Chrome and using any other browser like Firefox or Safari was not possible.

Connecting the private blockchain and the public blockchain was a huge hassle as it requires a huge amount of space on the local machine making it another hurdle.

Web3 was very difficult to learn as it is new to all of us. Understanding the flow of web3 code has been a real challenge.

4. LIMITATIONS

Complexity: Blockchain technology involves an entirely new vocabulary as the cryptography is made more mainstream. There are several efforts at providing glossaries and indexes that are thorough and easy to understand

Network Size: The blockchain networks require a large number of users(nodes). If a blockchain is not a robust network with a widely distributed grid of nodes, it becomes more difficult to reap the full benefit. Because of this huge size sometimes it is very difficult to identify the root cause of an issue

Transaction Cost and Speed: Bitcoin now has notable transactions costs though it was initially free. Now, if a user has to make a vote then he incurs some transaction cost(ether). So even though the voting is trustworthy, it is costly to the user.

Human Error :If a blockchain is used as a database, the information going into the database needs to be of high quality. The data stored on a blockchain is not inherently trustworthy, so events need to be recorded accurately in the first place.

Unavoidable Security Flaw :There is one notable security flaw in bitcoin and other blockchains: if more than half of the computers working as nodes to service the network tell a lie, the lie will become the truth. This is called a '51% attack' and was highlighted by Satoshi Nakamoto when he launched bitcoin.

5. CONCLUSION

Now we can say with confidence that whomever we are voting to, our vote will be correctly counted as

all the process is transparent. The decentralized app allows us to see the candidates that are present, keep track of how many votes each candidate is receiving, keep a check on the number of times a user can vote(one user/voter can vote only once – avoid double spending). We can now trust our voting process, but at the cost of a small transaction fee(very less ether is consumed as our transaction fee every time a voter votes).

6. FUTURE WORK

At present, this application is very basic and has only the compulsory features like voting only once, transparent voting mechanism etc, but new features like voting period, an analysis mechanism for the votes cast, voter registration system – so that only valid voters can cast the votes, displaying the portfolio and/or agenda of the particular candidate

7. REFERENCES

i) Introducing Ethereum and Solidity[Foundations of Cryptocurrencies and BlockChain Programming for Beginners] – Chris Dannen

ii) he General Theory of Decentralized Applications, DApps – David Johnston, Steven Mason

iii) Solidity Documentation [Release 0.4.22]

iv) Making Smart Contract Smarter, Loi Luu, Duc-Hiep Chu, Prattek Saxena, Aquinas Hobor.

8. APPENDIX

Deploying

```
Priscillas-MacBook-Air:election-master priscilla$ truffle migrate --reset
Using network 'development'.

Running migration: 1_initial_migration.js
  Replacing Migrations...
  ... 0xda41047837e5c2df34be7c27718a2e166e00aef81fceb686239b96b764655ecf
  Migrations: 0x0e7eb48a296b271bf6500ba4fd68dcd26df5d4b5
Saving successful migration to network...
  ... 0x3147baad33cb86f8e3063ffafa27fd71795877f2c19e8e767a6709a7cf2895d6
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Replacing Election...
  ... 0x3569fae8e8555e5d8912abad11996db663712b5ac25e5dc5f6d41d034e8531a0
  Election: 0x96a64c17bab0ede19d587237659e61c794d1d902
Saving successful migration to network...
  ... 0xeb47cee345027d1f3f8abac99f6a65e2e7e9f0006bd68113b5b26a151b1ad39e
Saving artifacts...
Priscillas-MacBook-Air:election-master priscilla$
```

Application UI

Truffle

```
Priscillas-MacBook-Air:election-master priscilla$ npm run dev

> pet-shop@1.0.0 dev /Users/priscilla/Downloads/election-master
> lite-server

** browser-sync config **
{ injectChanges: false,
  files: [ './**/*.{html,htm,css,js}' ],
  watchOptions: { ignored: 'node_modules' },
  server:
   { baseDir: [ './src', './build/contracts' ],
     middleware: [ [Function], [Function] ] } }
[Browsersync] Access URLs:
 --------------------------------------
       Local: http://localhost:3000
    External: http://172.20.20.20:3000
 --------------------------------------
          UI: http://localhost:3001
 UI External: http://172.20.20.20:3001
 --------------------------------------
[Browsersync] Serving files from: ./src
[Browsersync] Serving files from: ./build/contracts
[Browsersync] Watching files...
18.05.08 19:59:37 200 GET /index.html
18.05.08 19:59:37 200 GET /js/app.js
18.05.08 19:59:37 200 GET /js/bootstrap.min.js
18.05.08 19:59:37 200 GET /css/bootstrap.min.css
18.05.08 19:59:37 200 GET /js/web3.min.js
18.05.08 19:59:37 200 GET /js/truffle-contract.js
18.05.08 19:59:38 200 GET /Election.json
```