

Secure Voting

CM3203 One Semester Individual Project

Final Report



Scott Hulbert
May 2019

Supervisor: Dr George Theodorakopoulos
Moderator: Dr Yuhua Li

Abstract

The aim of this project is to implement a secure voting application. A voter should be able to vote anonymously, but no unauthorized party should be able to vote on behalf of an authorized voter. The system should be able to properly authorize legitimate voters and reject everybody else.

Voters should also have anonymity within the system to protect their privacy and prevent coercion. No person should be able to modify or delete a vote, nor can they insert a false vote into the system. The system should also ensure its own availability by avoiding a central point of failure.

This application will be accessible via a website that could be used on mobile devices or on regular computers. The website application will be supported by a blockchain back-end to help ensure the above security properties are met.

The primary focus of the project is on this blockchain back-end. The implementation of the web application provided will likely introduce security issues and should be seen as a prototype to be built upon.

Acknowledgements

Thank you to Dr George Theodorakopoulos and his PhD student Peter Davison for their support and guidance on this project.

Thanks to the developers of Hyperledger Fabric for writing brilliant documentation.

Thank you to my family and friends for keeping me sane.

Contents

1	Introduction	6
2	Background	8
2.1	E-voting	8
2.2	Blockchain	8
2.3	Blockchain Voting	11
2.4	Existing Blockchain Voting work	12
2.5	Permissioned Blockchain Frameworks	13
3	Specification, Design and Implementation	15
3.1	Architecture	15
3.1.1	Hyperledger Fabric Blockchain	17
3.1.2	Flask Client Application	25
3.1.3	Python Blockchain Interface	32
3.2	Challenges	33
4	Results and Evaluation	38
4.1	Performance Testing	38
4.1.1	Methodology	38
4.1.2	Results	39
4.2	Error Handling	44
4.3	Security Properties Evaluation	45
5	Future Work	47
5.1	Blockchain Network Ordering Service Improvements	47
5.2	Replacement of Python Blockchain Interface	47
5.3	Race Condition Elimination	48
5.4	Testing	48
5.5	Investigating Other Blockchain Frameworks	49
6	Conclusions	50
7	Reflection	51
7.1	Learning	51

7.2 Project Management	51
Glossary	Error! Bookmark not defined.
Abbreviations	53
References	53
Appendices	54
User Guide	54

Figures

Figure 3-1 – System Architecture Overview	15
Figure 3-2 - Blockchain Network	18
Figure 3-3 - UML Sequence Diagram.....	21
Figure 3-4 - State Transition Diagram of an Election	22
Figure 3-5 - Actors involved in elections.....	22
Figure 3-6 - Structure of an Election.....	23
Figure 3-7 - Actors involved in a registration	23
Figure 3-8 - Structure a register	23
Figure 3-9 - Actors involved in voting	24
Figure 3-10 - Structure of a vote	24
Figure 3-11 - Actors involved in a count.....	25
Figure 3-12 - Structure of a count	25
Figure 3-13 - Election Administrator Homepage	27
Figure 3-14 - Voter homepage	28
Figure 3-15 - Election Creation Form	29
Figure 3-16 - Election Creation Results page.....	30
Figure 3-17 - Election Count Results page.....	30
Figure 3-18 - View Elections page	31
Figure 3-19 - Election Detailed View	31
Figure 4-1 – Actions unimpacted by the number of voters	40
Figure 4-2 - Actions impacted by a large number of voters	42
Figure 4-3 - Actions impacted by number of voters.....	43

1 Introduction

The aim of this project is to implement a secure voting application. This application should allow clients to vote anonymously, but no unauthorized party should be able to vote on behalf of an authorised voter. The implementation could be accessible on desktop or mobile devices via an application or website.

When discussing the security of an application, it is important to define what exactly we mean by security. Security in computer systems is often defined by the implementation of the CIA Triad or Cornerstone [1] security properties. There are many other desirable security properties such as Authentication and Anonymity.

To define what is meant by secure for this project, I aim to implement the following security properties, defined in terms of voters using the system:

1. Confidentiality - No person can tell how another person voted
2. Integrity - No person can modify or delete a vote, nor can they insert a false vote into the system
3. Authentication - Only authorised parties can vote
4. Anonymity - It is not possible to tie someone's identity within the voting system to their real-life identity
5. Availability - Voters should always have access to the system when requested

These properties should also be implemented in terms of those who wish to create elections in the system:

1. Integrity – No unauthorised person can modify an election. No person can delete an election
2. Authentication - Only authorised parties can create and modify elections
3. Availability – Election Administrators should always have access to the system when requested

The intended audience for the project is any organisation or government who wishes to hold an election with a number of participants and proposals or candidates. Due to funding and time limitations, this application is not be intended for use in applications with a very large number of voters such as General Elections. The application may however be suitable for purposes such as Boardroom or Local Elections.

The scope of the project is primarily focused on the system used to securely store election data. A website to access this system will be provided but it is intended for demonstration purposes only and should be seen as a prototype.

The approach taken will be to first, work on the secure system for election data. Data such as votes, elections, voter registers and election results all need to be stored and retrieved on demand. A blockchain infrastructure will be used to help ensure the integrity of the system. This secure system will be operated via a Command Line Interface (CLI).

Once this system has been fully developed and allows for elections to be modelled from start to end, a website will be developed.

This website is intended to be used by two different types of users. Those who will be involved in the creation and management of elections, Election Administrators, and those who will register for and vote in elections, Voters. Both types of users will be able to use the website to view data in the secure system such as the status of and the results an election.

Election Administrators will have various actions available to them such as creating a new election, adding voters to an existing election and the ability to trigger the counting of a finished election. Voters will have actions such as being able to register to vote in an election that has not yet started and voting in an election that is underway.

The assumptions made for this project are the following. Elections will follow the first-past-the-post electoral system where one person gets one vote for a single candidate and the candidate with the highest number of votes wins. The system will not support other electoral systems such as Ranked voting.

First-past-the-post has been chosen because it is the one of the simplest forms of elections to represent. It is also the electoral system used for most of the elections in the UK and is one of the most common electoral systems in the world¹

Important outcomes

- Develop a blockchain infrastructure to store election data
- Implement the previously defined security properties
- Develop a website to provide access to the infrastructure

¹ Source: <https://www.independent.co.uk/news/uk/politics/first-past-the-post-voting-system-uk-what-is-fptp-electoral-proportional-representation-a8623696.html>

2 Background

2.1 E-voting

It is no surprise that in the modern era where nearly everything is done digitally, efforts have been made to digitise elections. E-voting is simply a voting system that uses some electronic means to handle casting and/or counting votes.

E-voting can be split into two main categories:

- Supervised e-voting – votes are cast under the supervision of election officials
- Internet Voting – votes are cast remotely over the internet

The system developed as part of this project could be deployed as either an Internet Voting system, where users connect to a website, which in turn connects to the blockchain. Alternatively, the website and blockchain could be hosted on a government intranet for use in a supervised e-voting system. However, due to the nature of the project delivering a website rather than a dedicated electronic voting machine, an internet voting system is likely more appropriate.

Currently, the world is divided on e-voting. Estonia became the first country to host a local election over the internet in 2005. Estonia also went on to hold the first general election over the internet in 2007. Other countries, such as Canada, have decided against e-voting due to security concerns^[2]. France has also decided against the use of e-voting for citizens abroad due to security concerns².

The UK has recently had an e-voting trial performed by the University of Warwick³. This was not a blockchain-based system but it is an end-to-end verifiable system, allowing voters to verify that their vote has been cast and tallied correctly.

2.2 Blockchain

For the purposes of this project, it is not important to have an in-depth understanding of the underlying technologies that support Blockchain. This section will cover the basics of what Blockchains are, but the focus is on the properties they provide such as Replication and Immutability. This section will also discuss the different types of Blockchain, which one has been chosen for this project and why.

² Source: <https://www.reuters.com/article/us-france-election-cyber-idUSKBN16D233>

³ Source: <https://warwick.ac.uk/fac/sci/dcs/news/?newsItem=8a1785d86a6df044016a96a32ef65ebd>

Blockchain is an example of a Distributed Ledger Technology (DLT)[3]. DLTs are defined as a way to distribute data across a network while ensuring that data is replicated and synchronised among the peers. The peers also have consensus on the state of the data. In a DLT, there is no central authority controlling the peers nor any centralised storage.

In Blockchain, data is stored as a sequence of chronological transactions. The transactions are stored in blocks – each block contains multiple transactions. Each block is linked to the previous block and the next block. Validation of each new block is achieved when consensus is reached by all nodes through some consensus mechanism. In Bitcoin for example, consensus is achieved by Proof-of-Work⁴.

Blockchain can provide[3] :

- Decentralisation – there is no central authority running the network, all nodes are equal to each other
- Replication – each node has a copy of the blockchain
- Transparency – each node can see all records
- Timestamping – transactions are timestamped
- Integrity – once a transaction has been included in a block and that block has been validated, is it nearly impossible for a lone malicious node to change the ledger

Chaincode:

A smart contract defines code that should run when certain conditions are met on the network. For example, when a transaction is proposed to cast a vote, chaincode is triggered to create the object to store the vote and store it on the blockchain. This allows for business logic to be defined at the network level.

Ethereum, one of the most popular public blockchains, has implemented their own smart contract framework called Solidity that is considered to be nearly Turing-complete. Other blockchains such as Hyperledger Fabric provide SDKs for existing languages such as Go and Node.js

This allows for chaincode to be written in a similar fashion to traditional programs and for the blockchain to be used like a database to store and retrieve data. There is no need to have a deep understanding of the supporting technologies behind blockchain to use these SDKs as the details are abstracted away.

⁴ An explanation of proof-of-work can be found at: https://en.bitcoin.it/wiki/Proof_of_work

There are two main types of Blockchain.

Public / Permissionless Blockchain:

Bitcoin and Ethereum are two of the most popular public blockchains. As the name implies, these Blockchains are accessible via the public internet. Anyone with internet access can take part in these networks and perform actions like proposing transactions, inspecting blocks or contributing to consensus.

Private / Permissioned Blockchain:

Hyperledger Fabric and Ripple are two examples of private blockchains. Private blockchains are not publicly accessible. Network administrators dictate who can join the network and what actions they can perform. The network administrators may not belong to a single organisation – there may exist a consortium that manages the network. Note that public, permissioned chains and private, permissionless chains do exist.

Arguably the largest disadvantage of a public blockchain is the use of Proof-Of-Work. Proof-of-work is the reason why the Bitcoin network consumes an incredible amount of power – as of April 2019, the Bitcoin network is consuming almost as much power as countries such as Israel, Greece and Algeria⁵.

The difficulty of the hash puzzles under Proof-of-work is frequently adjusted by the network to keep the time between blocks at an average at 10 minutes. In Bitcoin, to avoid double-spend attacks, it is common practice to wait for a number of new blocks to extend the chain before accepting a transaction as valid – this is known as a confirmation. This is because the probability of a successful double-spend attack decreases exponentially with the number of blocks [4].

Typically, most Bitcoin users will wait for 6 confirmations, but this is not a set rule of the network – some users may choose to wait for more or less confirmations. If the time between blocks averages 10 minutes, this means that it will take an average of an hour before a transaction can be accepted.

As private blockchains only consist of trusted nodes, there is no need to use a consensus mechanism like Proof-of-Work. Other consensus mechanisms such as Tendermint's BFT Consensus allow for consensus without mining[5].

This allows the environmental impact of a private blockchain to be drastically smaller than a public blockchain. Transactions can also be proposed and confirmed much

⁵ Source: <https://digiconomist.net/bitcoin-energy-consumption>

faster. For example, a recent paper showed that with optimisations, Hyperledger Fabric could support up to 20,000 transactions per second[6].

Depending on individual use cases, the fact that public blockchains have no central authority may be a clear positive or negative point but often, there is both advantages and disadvantages to this.

For example, for a cryptocurrency like Bitcoin – it may be desirable to have an international currency that is not subject to exchange rates. The lack of a central authority also protects Bitcoin users from bank fees and bank policies that may limit them. There is also no fear of bank failures and bank collapses.

However, the protection banks provide in the form of fraud detection and prevention is lost. If you are a victim of fraud, a scam or if there is some mistake in the blockchain, there are no laws to protect you. All of this makes Bitcoin very attractive for illegal activity.

A high-profile example is the WannaCry ransomware infection that occurred in 2017. WannaCry encrypted all the files stored on a computer and asked users for a £230 payment in Bitcoin⁶.

For some applications such as supply chain management or asset exchange, a consortium running the blockchain is more suitable than a public chain. In these cases, there is data that should only be accessible to members of those organisations – it should not be hosted as a Permissionless chain on the public internet.

Also, none of the organisations are a clear central authority that should have control over the whole network, so a traditional centralised system may not be suitable as there may be some dispute over who should control this system. In this case, a consortium with members from each organisation could run a private, Permissioned blockchain.

2.3 Blockchain Voting

Blockchain voting systems could be instrumental in dealing with some of the largest issues faced in voting – namely, participation and fraud according to the authors in [7]. In terms of voter fraud, blockchain provides a system to store election data with integrity.

⁶ Source: <https://www.bbc.co.uk/news/technology-39901382>

It is nearly impossible to tamper with the data stored on the blockchain. Identities in the blockchain can be decoupled from real-life identities, allowing for voters to be authenticated to access the blockchain but their actions on the blockchain to remain anonymous. There is no need to track their actions tied to their real-life identity as the integrity of the blockchain is ensured regardless of the actions of a voter.

Blockchains can provide better transparency than a traditional election. The data stored in the blockchain can be open for inspection by anyone – allowing anyone to verify that any given vote was correctly stored and counted on the system. The chaincode used on the blockchain could be made open-source. This may have some risk in exposing potential vulnerabilities in the chaincode but would allow anyone with a technical background to verify the behaviour of the system

2.4 Existing Blockchain Voting work

Researchers from Reykjavik University, Iceland proposed a permissioned blockchain based voting system using one of three different blockchain frameworks [8]. Go-Ethereum (Geth) was chosen as it was considered to be the most secure and developer-friendly of the three frameworks.

The paper proposes similar requirements to those I have proposed in terms of authentication, integrity, anonymity, transparency and confidentiality. I was not aware of this paper while I was designing and implementing the project. One key difference is that I have implemented a web application to make the use of the blockchain application more user-friendly whereas, the paper appears to focus on the blockchain application entirely.

Researchers from The University of Tokyo proposed a permissioned Blockchain based voting system using a blockchain with proof-of-work [9]. The future work section suggests that proof-of-work may not be suitable for a voting system due to nodes being selected according to hash power – elections could be influenced by those with the most computational power. There are also the issues of environmental impact and scaling discussed earlier in this report.

One paper has proposed the use of a permissioned blockchain, Hyperledger Fabric for a decentralised election system [10]. This paper contains a summary of other blockchain voting systems that highlights some of the issues these systems face.

To summarise, most of the existing systems either rely on a central authority in some form or do not scale well due to the use of proof-of-work. This paper combines

approaches from traditional e-voting systems such as blind signatures and zero-knowledge proof with blockchain

2.5 Permissioned Blockchain Frameworks

Based on existing research and the requirements of the voting system, this project will use a permissioned blockchain framework. There are many different frameworks in this area, this report will focus on the most popular.

The Hyperledger Project is a collection of open source blockchain framework and tools. It is maintained by the Linux Foundation and is supported by organisations across many industries⁷ such as:

- Blockchain vendors such as Digital asset
- Technology companies such as IBM, Intel, Cisco, Airbus
- Financial services firms such as J.P Morgan, American Express

Currently, Hyperledger run 6 framework projects:

Fabric is the most active project. It has a modular architecture which allows components such as the consensus algorithm and membership services to be changed easily. Typically, smart contracts (known as chaincode in Fabric) are written in Go but in theory any programming language could be used that implements the Fabric interface for smart contracts.

Burrow is made up of 2 key parts. The first of these is the Ethereum Virtual Machine for running smart contracts. Ethereum is one of the most popular public blockchains. The second is the use of the Tendermint consensus algorithm which allows for consensus without mining, as discussed earlier. Burrow, unlike Fabric, is not considered to be production ready.

Iroha is written with C++ and makes use of a Byzantine Fault Tolerant (BFT) ordering service and consensus algorithm. It also makes use of a PostgreSQL to store the state of the blockchain. It is intended to be fast and performant, allowing for blockchain applications that could run on mobile devices⁸. When the project started, it was not considered production ready⁹ so I did not consider using it.

⁷ Source: <https://www.hyperledger.org/members>

⁸ See: <https://www.hyperledger.org/projects/iroha>

⁹ Iroha became production ready on May 6th 2019

<https://www.hyperledger.org/blog/2019/05/06/welcome-hyperledger-iroha-1-0-flattening-the-dlt-learning-curve>

Sawtooth uses Proof of Elapsed Time for its consensus algorithm. The nodes in Sawtooth are completely decentralized, similar to a public blockchain. It is considered to be production ready.

Indy is focused on distributed identity management. Similar to Iroha, it was not considered production ready¹⁰ when the project started so I did not investigate it further.

Hyperledger also provides various blockchain-related tools such as:

- Caliper: Blockchain benchmarking tool
- Composer: Tools for building blockchain applications and smart contracts for business applications
- Explorer: Web application to explore blocks in the blockchain
- Quilt: Offers an interledger protocol (ILP) - this allows for interoperability between different types on distributed and non-distributed ledgers
- Ursa: A cryptographic library to avoid the duplication of cryptographic work by using existing solutions

All of these tools are not considered to be production ready, so I did not investigate them any further for this project. Perhaps in the future, blockchain development will be made simpler by the existence of these tools when they are fully released.

In order to achieve the aim of implementing a secure voting system, this project will use a private, permissioned blockchain framework. The framework used will be Hyperledger Fabric. This is to avoid the issues discussed with public blockchains, particularly proof-of-work and its scalability – an important point for elections. The project will also provide access to the blockchain system through a website to ensure that it is simple to use.

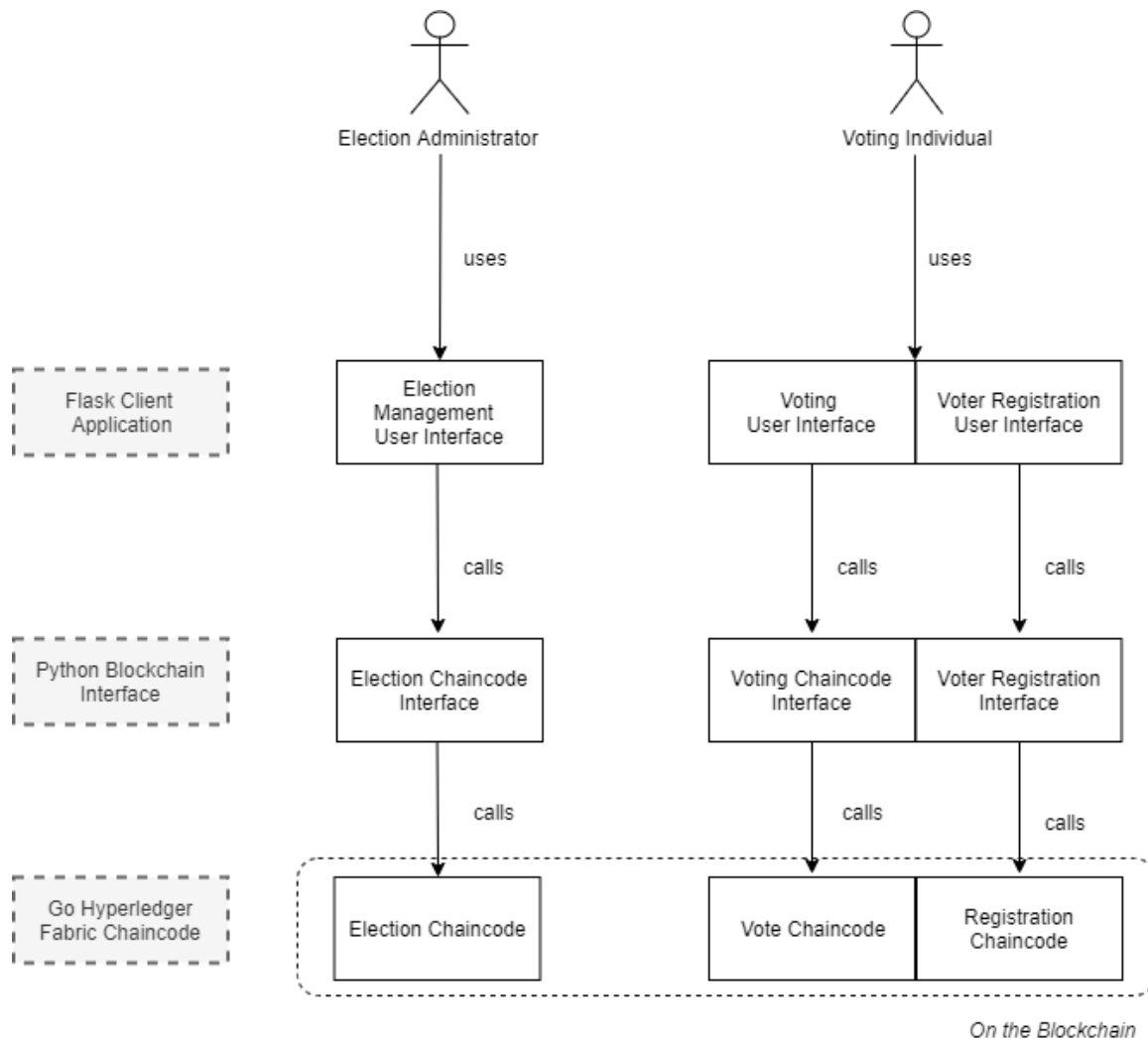
Hyperledger Fabric has been chosen as it is the best documented, most active of the Hyperledger frameworks. Alternatively, Sawtooth could have been chosen as it was the other production-ready framework at the time of selection. I selected Fabric over Sawtooth as it has private data collections and channels for private transactions which Sawtooth lacks.

¹⁰ Indy became production ready on April 10th 2019
<https://www.hyperledger.org/blog/2019/04/10/hyperledger-indy-graduates-to-active-status-joins-fabric-and-sawtooth-as-production-ready-hyperledger-projects>

3 Specification, Design and Implementation

3.1 Architecture

Figure 3-1 – System Architecture Overview



The system has 2 user roles. The first of these is the Election Administrators who use the Election Management Interface to create and manage elections. The Election Management Interface supports the creation of new elections, allows existing elections to be started and finished and the counting of a finished election to be triggered. The second user role is Voting Individuals (Voters) who use the Voting Interface to cast votes in elections and the Voter Registration Interface to register to vote in elections.

All users access these interfaces through a web application developed using Flask¹¹. The homepage of the web application gives links to the various actions available to the user.

For example, a voter's homepage has links to all the elections they can vote in and all the elections they are able to register to vote for. For instance, to vote in an election, the user simply clicks the election they wish to vote in on the homepage, this redirects them to a form. Next, they select which candidate they wish to vote for from a dropdown menu and submit the form. Screenshots of the User Interface will be provided in Section 3.1.2

When this form is submitted, the Flask application calls the Python Blockchain Interface. This Interface handles communication between the chaincode and the Flask application. The Docker SDK for Python (a Python library for the Docker Engine API) is used in the Interface to send commands to a Docker container that is the command-line interface (CLI) for Hyperledger Fabric

The commands sent to the CLI for Hyperledger Fabric cause chaincode to be triggered through the `peer` command. The full architecture of the Blockchain network and chaincode will be described in Section 3.1.1.

The chaincode has been written in Go using the Hyperledger Fabric Go SDK. I used the Go SDK despite my lack of familiarity with the language because the majority of the samples provided by Hyperledger are written in Go. Interestingly, Fabric has Node.js and Java SDKs that are officially released whereas the Go SDK has not yet been officially released¹².

Not shown in this diagram is a third user role for testing purposes that allows for elections with a specified number of voters and candidates to be simulated. This user role will be discussed in Section 3.1.2

¹¹ <http://flask.pocoo.org/>

¹² https://hyperledger-fabric.readthedocs.io/en/release-1.4/getting_started.html#hyperledger-fabric-sdks

3.1.1 Hyperledger Fabric Blockchain

Blockchain Network

Figure 3-2 shows the structure of the Hyperledger Fabric blockchain network. There are four nodes in this network, two belonging to Org1 and two belonging to Org2. These two organisations are connected by a channel named mychannel1. This network uses a Solo ordering service.

This network is the Build Your First Network¹³ (byfn) network provided as a sample by Hyperledger.

My supervisor and his PhD student, Peter Davidson, recommended this as a good starting point for Fabric. To be able to run this network, the instructions on the Hyperledger Fabric website should be followed

Part of the future work for this project would be to move away from this sample and build a network with a production-ready ordering service and more nodes for scalability. The impact of using the byfn network will be discussed further in the future work section of this report.

Figure 3-2 shows a simplified view of the blockchain network. In practice, there are other parts of the network such as a Certificate Authority (CA) that authenticates each node as belonging to a given organisation and the ordering service and ledger¹⁴. As these parts of the network are not key to understanding the project, I have excluded them from this diagram for simplicity.

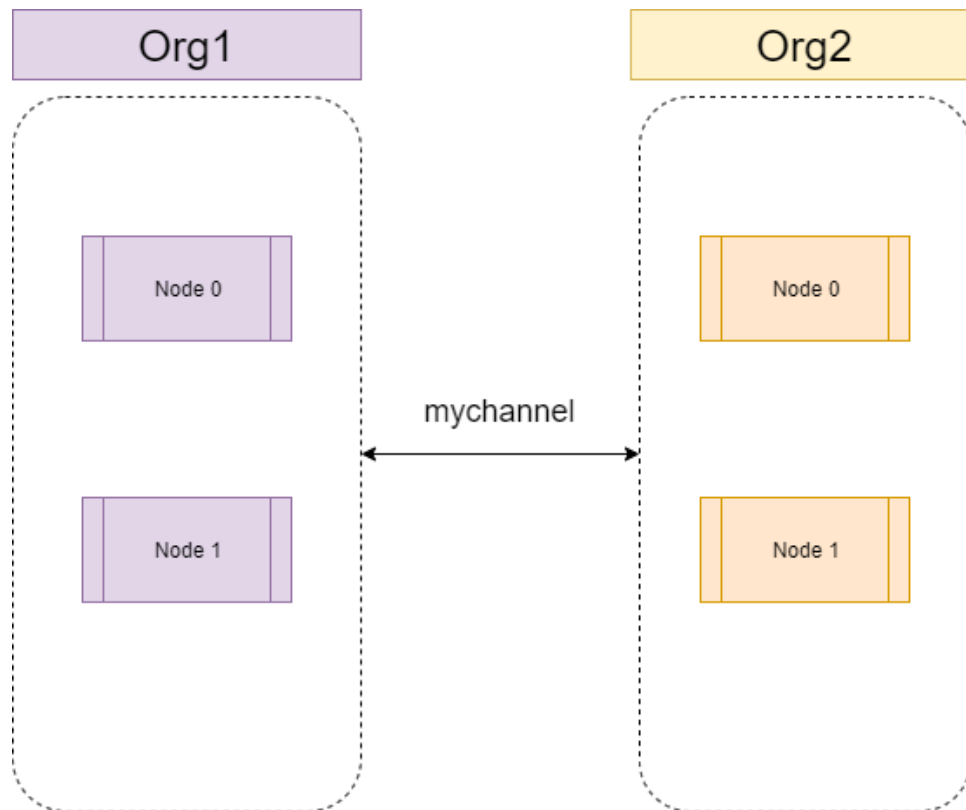
Org1 is the organisation that Election Administrators belong to. This organisation has access to modify and create elections while Org2, the organisations voters belong to, does not.

¹³ Further details can be found at: https://hyperledger-fabric.readthedocs.io/en/release-1.4/build_network.html

Installation instructions for the network can be found at: <https://hyperledger-fabric.readthedocs.io/en/release-1.4/install.html>

¹⁴ See: <https://hyperledger-fabric.readthedocs.io/en/release-1.2/network/network.html>

Figure 3-2 - Blockchain Network



The byfn network uses Docker to create containers. Each node sits in its own container. A fifth container is created and used as a command-line interface for the network, this container is named CLI.

Commands are sent to the network through this CLI container. Commands can be run on the peers by using the 'peer' command on the CLI container. Which peer runs the command is determined by various variables. These variables can be modified to switch between the different peers.

The main command used is the `peer chaincode`¹⁵ command, this command allows for various chaincode operations to be performed such as:

- `peer chaincode install`

This command installs the chaincode on the currently selected peer. A name is given for the chaincode to be referred to in future commands. The `-v`

¹⁵ The documentation for this command can be found at: https://hyperledger-fabric.readthedocs.io/en/release-1.4/private_data_tutorial.html#pd-query-authorized

argument can be used to specify the version of the chaincode, this allows the `install` command to be used to both install the chaincode for the first time and upgrade the chaincode.

- `peer chaincode instantiate`
This command installs the chaincode onto the specified channel. Again, a name is given so the chaincode can be referred to in future commands.
- `peer chaincode invoke`
This command invokes the specified chaincode on a specified channel. Arguments to chaincode functions can be passed in via a JSON string with the `-c` argument. For chaincode transactions involving private data, the data should be passed to the chaincode using the `--transient` argument¹⁶. This data must be in a binary form such as Base64. This command should be used to pass data to store on the blockchain.
- `peer chaincode query`
This command gets the result of a chaincode function and prints it, without creating a transaction. The `invoke` command simply returns a status code and message that indicates whether the transaction was committed successfully. To retrieve data from the blockchain, the `query` command should be used.

In order to install the chaincode on all 4 peers and the channel, several commands have to be executed. Each of these commands takes several seconds. This can make developing and testing chaincode frustrating as small changes can take a long time to test.

To help simplify this process, I spent time during the first couple of weeks of the project writing shell scripts to automate this process and other processes relating to the blockchain network. Writing these scripts was challenging as I was unfamiliar with Unix systems. It is something that I believe was an important part of the initial stages of the project and was a valuable experience.

I will briefly list the purpose of the most important of these shell scripts here:

- `init_network.sh` – this is the most important script, it simply calls `start_network.sh` followed by `install_chaincode.sh` with the `-first-time` flag set

¹⁶ Source: https://hyperledger-fabric.readthedocs.io/en/release-1.4/private_data_tutorial.html#pd-query-authorized

- `start_network.sh` – this script calls the `byfn` script provided by Fabric to bring up the network, it also stops and removes the containers for any old versions of the network that were running before the script was run
- `install_chaincode.sh` – this script installs the chaincode on all four peers and instantiates the chaincode on the channel, this can be used to install the chaincode for the first time on the network or to upgrade the chaincode
- `parameters.sh` – this script simply sets variables for use in other scripts such as the channel name and chaincode name to be used
- `populate_network.sh` – this script runs chaincode `invoke` commands to simulate an election from start to end with 2 voters, this is how I tested the chaincode before the Python Blockchain Interface was developed

Chaincode

The chaincode uses 4 classes to model elections. These classes are: `election`, `register`, `vote` and `count`. The data stored in these classes will be shown in this section. The section will also show how calling chaincode methods changes the data. A UML Sequence diagram will be provided to show how an election is modelled in the system from start to end.

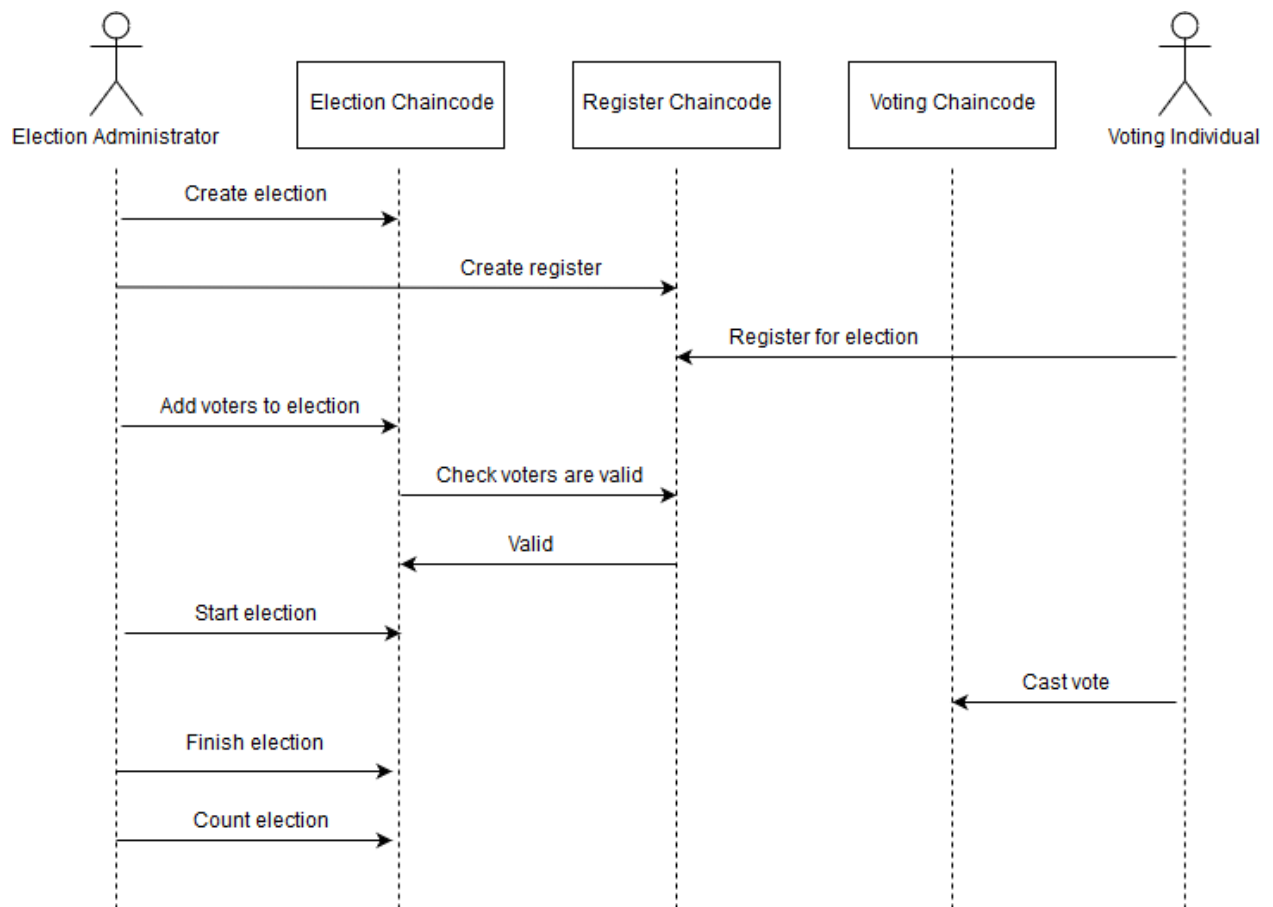
The classes are stored in a CouchDB database in JSON format. The chaincode stores these classes in the databases by marshalling them from Go classes to JSON as necessary. Vice versa, the classes are unmarshalled from JSON to Go classes as necessary when data is retrieved from the database.

CouchDB is a key-value pair database. The JSON objects are stored in the database using a unique identifier. The objects can be retrieved simply by their key, or by using a CouchDB selector¹⁷. Selectors allow for retrieval based on the properties of the object such as the presence of a certain field, combination operators and conditional operators similar to SQL queries¹⁸.

¹⁷ Source: https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html#cdb-query

¹⁸ More details and examples can be found at:
<http://docs.couchdb.org/en/latest/api/database/find.html#selector-syntax>

Figure 3-3 - UML Sequence Diagram



The UML Sequence Diagram shown in Figure 3-3 shows how an election is modelled in the system in terms of how actors interact with the system and how the chaincode interacts with itself. The actions taken by the users, in most cases, must occur in the order shown. For example, the Election Administrator cannot count the election before it has been finished. The effect of each action in the system will be shown later in this section.

I created these diagrams near the start of the project to help plan out my work. I have maintained them as the project has changed. This section will also highlight how the design of the system has changed as I learned more about both Go and Hyperledger Fabric.

Election

The below figures show the `Election` class. This class is used to represent elections in the system. It holds data such as the candidates that can be voted on in the election,

who can take part in the election and when the election can be started and finished. The election is stored in CouchDB using its Election ID as the key.

The election also holds an RSA encryption key so the chaincode can encrypt voter's votes. A private data collection¹⁹ hold the RSA decryption key so the votes can be decrypted for counting – only the Election Administration Organisation has access to this private data collection.

In terms of user roles, elections are created and modified by Election Administrators. Voters can view elections but cannot view the decryption key. Voters are also unable to modify or create elections.

When this diagram was originally created, I had intended to have the start and finish of the election be triggered automatically based on the start date and end date that had been set when the election was created. Upon further investigation, I discovered that it was not possible in Fabric to have the chaincode invoked automatically at a set time.

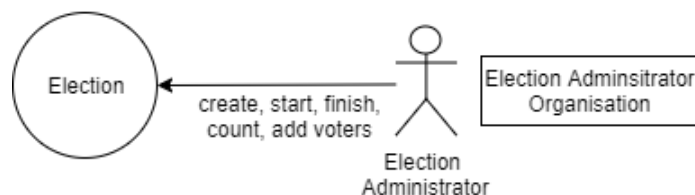
An alternative was to simply set up a cron job on the operating system to invoke the chaincode at the desired time. For this reason, I changed the design to have the start and end of the election triggered manually.

Currently, the Flask frontend does not setup a cron job to handle the automatic start and end of elections. This is because, for development and testing purposes, all the times used are inputted by the user rather than using the system time. This will be discussed further in the Section 3.1.2.

Figure 3-4 - State Transition Diagram of an Election



Figure 3-5 - Actors involved in elections



¹⁹ A private data collection allows information to be kept private from other organisations on the channel, see <https://hyperledger-fabric.readthedocs.io/en/release-1.4/private-data/private-data.html> for details

Figure 3-6 - Structure of an Election



Register

The below figures show the `Register` class. It holds a list of voters who have registered for the election. Each election has an associated register. The register should be created by the election administrator at the same time as the election. The register is stored in CouchDB with the associated Election ID as its key. This class is used to represent voters registering for elections in the system.

The register is used to define which voters can be added to the election. Voters must be in the register for an election before they can be added to it. Election Administrators can add voters manually, this can be done one voter at a time or multiple in a list. When election administrators create the register, it can be created empty or they can pass in voters to be added to the register immediately. Voters can also add themselves to the register.

Figure 3-7 - Actors involved in a registration

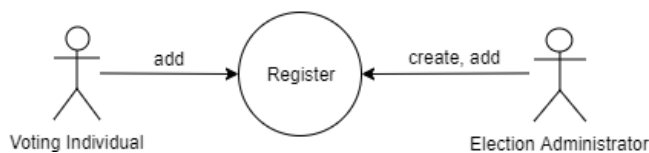
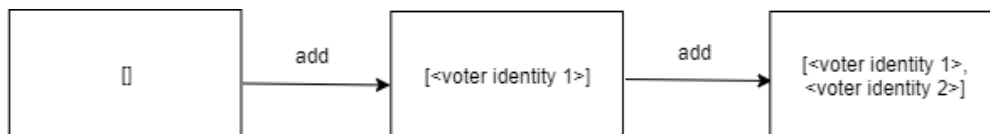


Figure 3-8 - Structure a register



Vote

Voters can only cast a vote in an election that has started (status = started)
Votes are encrypted using RSA Encryption with the Vote Encryption Key stored in the Election. As previously mentioned, only the Election Administrator has access to the private data collection that contains the Vote Decryption Key.

Figure 3-9 - Actors involved in voting

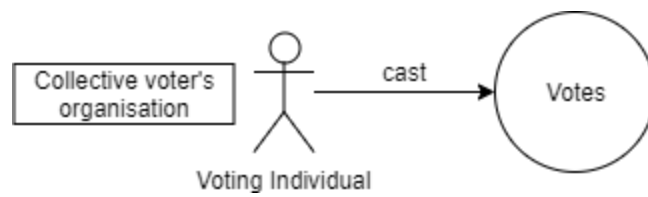


Figure 3-10 - Structure of a vote



Count

The count class is created when the election is counted. It stores the results of the election in a simple dictionary structure. The key of the dictionary is the candidate and the value is the number of votes they received.

The count method creates an instance of the count class. It also changes the status of the election to counted as seen in Figure 3-6. Although count is its own class, it is part of the election chaincode (rather than the count chaincode) as it is directly tied to an election.

Figure 3-11 - Actors involved in a count

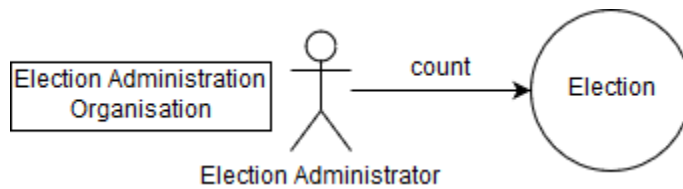
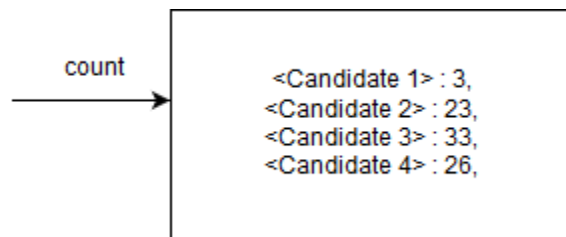


Figure 3-12 - Structure of a count



3.1.2 Flask Client Application

This section will show representative examples of each type of page on the Flask application as well as the home page. There are three types of pages:

- Action pages which cause objects to be created or modified on the blockchain
- Results pages which show the users the results of their actions
- View pages which allow users to query the blockchain to see previously created objects

Homepage

The navigation bar seen in Figure 3-13 persists across all pages of the client application. It provides a link back to the homepage by clicking on the 'Blockchain Voting' text and easy access to the view pages. Also, note the Role and Identity tabs in the navigation bar.

The homepage contains links to the various actions available to the user, this is dependent on the role of the user, as seen in the UML Sequence Diagram in Figure 3-2. Currently, the role of the user is switched by dropdown – users simply select Election Administrator or Voter.

The identity field can be changed by clicking on it and completing the form. This field only appears for the voter, this is because voters have a unique identity that they use to register for and vote in elections. There is no equivalent for the Election Administrator

I had originally intended to implement a login system to authenticate the user. This system would use a username/password to authenticate the user as belonging to a certain role and handle giving voters their identity in the blockchain. It is important for the username for the user and their blockchain identity to be different and for the mapping between the username and identity to be stored securely.

If the mapping was discovered or if they were simply the same value, the anonymity of the blockchain would be lost. Any person could see if another person registered for an election, if they voted and when they voted. This will be further discussed in the Evaluation and Future Work sections of this report but it is suffice to say that this is the main reason why the Flask Client Application should be only be considered for demonstration purposes.

Figure 3-13 Election Administrator Homepage

Blockchain Voting

View: Elections Counts Registers Votes

Role: Election Administrator ▼

This is the web interface to Blockchain Voting.

Create election

Add Voters to Election

testElection3

Start Election

testElection5

Finish Election

testElection6testElection7testElection8

Count Election

testElection9

Figure 3-14 - Voter homepage

Blockchain Voting View: Elections Counts Registers Votes Role: Voter Identity: default

This is the web interface to Blockchain Voting.

Register to vote in Election

testElection2 testElection3 testElection4

Cast Vote

testElection6 testElection7 testElection8

Actions pages

The types of actions available to users can be seen in the UML Sequence Diagram shown in Figure 3-2. The majority of these actions have Action pages that contain a form the user to fill out and submit. These action pages are accessed via links on the homepage. Once the form is submitted, the user is redirected to the results page. An example of a results page will be discussed shortly.

Some of the actions such as counting the election and registering for an election require no additional user input so when the buttons are clicked on the homepage, the user is taken straight to the results page for the action.

Figure 3-14 shows an example of an election creation action page, the election creation form has the fields required to create an election and store it on the blockchain. Once submit is pressed, the information provided in the form will be passed to the relevant method in the Python Blockchain Interface, in this case:

```
create_election(election_id, start_date, end_date, candidate_list).
```

The Python Blockchain Interface then sends a command to the Hyperledger Fabric CLI to create and store an election. The details of this will be shown in Section 3.1.3.

It should be noted that in the current implementation, there is no way for Election Administrators to add voters to the register. This was not missed due to it being challenging to implement, rather it was a simple oversight that there was not time to rectify.

Figure 3-15 - Election Creation Form

The screenshot shows the 'Election Creation Form' in the 'Blockchain Voting' application. The user is logged in as an 'Election Administrator'. The form includes the following fields and controls:

- Election ID:** A text input field containing 'isprojectgood'.
- Start Date:** A date picker showing '04/27/2019'.
- Start Time:** A time picker showing '09:00 AM'.
- End Date:** A date picker showing '05/04/2019'.
- End Time:** A time picker showing '09:00 AM'.
- Candidate List:** A section with the instruction 'Please list candidates/proposals to be voted for/on in the election'. It contains two text input fields: one with 'yes' and one with 'no'. Each field has a trash icon to its right.
- Add Candidate:** A button to add new candidates.
- Submit:** A blue button to submit the form.

Results page

As the name implies, the results pages are not directly accessible. The user is redirected to a results page after performing an action. The results page contains links to view the objects that were created or modified as a result of the user's action.

Headings split the results page into New / Modified sections. The New section simply links to newly created objects. The modified section contains text descriptions that tell the user what fields of the object has been modified for clarity. An example of this can be seen in Figure 3-17.

Figure 3-16 - Election Creation Results page

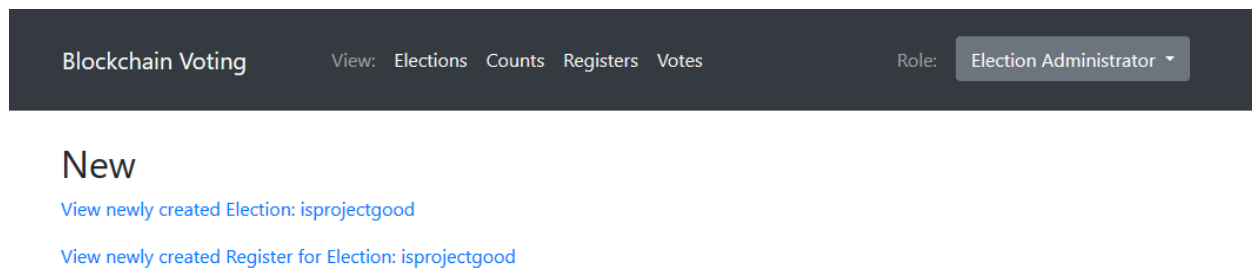
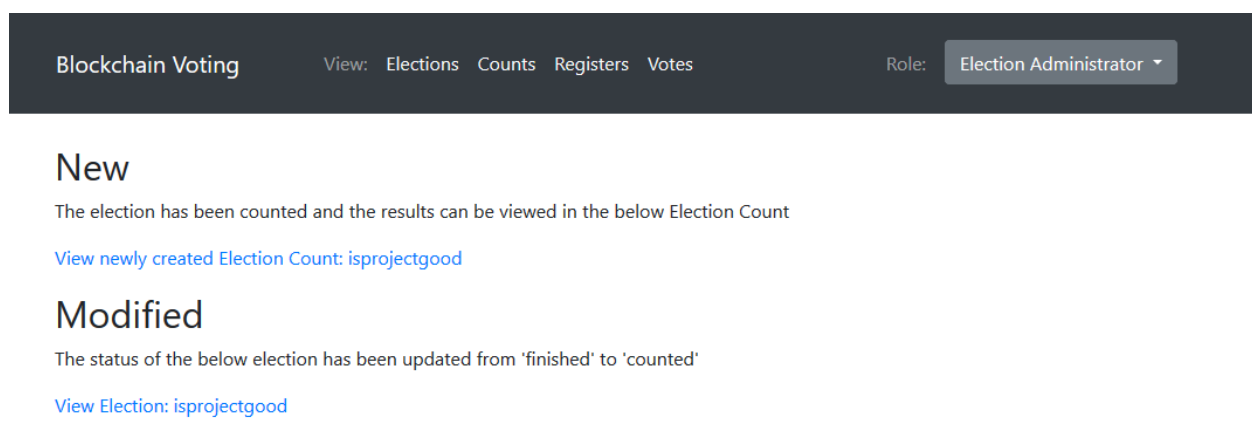


Figure 3-17 - Election Count Results page



View

View pages can be split into two categories, list view and detailed view. The view pages accessible from the navigation bar show all objects of that type, a list view. This view allows us to see some basic information about the object such as the status of an election or the number of voters registered.

For example, Figure 3-18 shows the page after clicking the 'Elections' link in the navigation bar. The ID field of the election is a link to a detailed view of the election. The detailed view contains all the fields for the Election as seen as in Figure 3-6.

Both Election Administrators and Voters have full access to both types of view pages. This allows both parties to verify that actions are being correctly recorded in the system.

Blockchain Voting

View:
[Elections](#)
[Counts](#)
[Registers](#)
[Votes](#)

Role:

Voter

Identity:

default

Elections

Showing 3 elections:

ID	Status	Start Date	End Date
exampleElection	started	28/04/2019 09:00:00	05/05/2019 09:00:00
isprojectgood	counted	27/04/2019 09:00:00	04/05/2019 09:00:00
sampleElection	created	27/04/2019 09:00:00	30/04/2019 09:00:00

Blockchain Voting

View: Elections Counts Registers Votes

Role: Election Administrator

Election: isprojectgood

[View count](#)
[View all elections](#)

ID	isprojectgood
Status	counted
Authority	Org1MSP
Start date	27/04/2019 09:00:00
End date	04/05/2019 09:00:00
Started at	28/04/2019 09:00:00
Finished at	10/05/2019 09:00:00
Voters	Jeff, default
Candidates	yes, no
Encryption key	MIIBKgKCAQEAw4FZWING7PuXh0wHrAFB+UHuQn5nNhFYTsWSjXUvqq+XIklKC/7oEC4wTkXyQsFbXyLqL2epo2asF1h3cE/a9TiC2K8yC1ZDV/2U7/WQnkFcr1SrJWJnsDRgbDakhJwV5mUhn14gP8CnHVUg56kmey851DDp2cURfHfpRgw0FT5k+wheQQFQpMW3z5dpyTd0ETCDY1z9JLaYt6LboGGXCAA+GjprR/kn2C0cj/KHrC0rRbMqLJU4yJrynYf0VQAD87JySutzbwIDAQAB

Simulator User Role

There also exists a third user role, simulator. As this role would likely not be present in a production version of the system, I have not included it in the discussions of the implementation thus far. This role can be accessed in the same manner as the other roles, through the drop-down selection in the navigation bar present on every page. As the name implies, this role allows the user to simulate elections with an inputted number of voters and candidates.

After the simulation has completed, the user is redirected to a simulation results page which shows all the objects created as a result of a simulation. These objects are prefixed with “sim-“ to make it clear they are simulated objects rather than part of any real election. This prefix is used to exclude any simulated objects from the list view pages.

The purpose of this role is primarily for development purposes in order to easily demonstrate the system working without switching between user roles and identities. The results page lists in order what objects were created and modified. This is to help a user to get to grips with the flow of an election in the system.

It could be argued that this user role does have some use to the organisation that wishes to deploy the system as it provides a simple method to verify that the network is functioning correctly. The same chaincode methods are called that would be called in a real election, so if the simulation functions, then a real election should as well. The simulation would also not impact real users experience with the system. As previously mentioned, simulation objects are excluded from the list view pages.

3.1.3 Python Blockchain Interface

Recall the `populate_network` script from Section 3.1.1. This script calls each chaincode function in the order described in the UML Sequence Diagram in Figure 3-3.

The Python Blockchain Interface provides Python methods to invoke every chaincode method. As described in Section 3.1, this is done using the Docker SDK for Python to send commands to the Hyperledger Fabric CLI Container. Each method in this class takes in some form of input and uses the `run_cmd` method to pass the input from the Python code to the CLI container.

The `run_cmd` method either runs a `peer chaincode invoke` or `peer chaincode query` command with the arguments passed into `run_cmd`. The `cmd` argument of `run_cmd` determines whether the `invoke` or `query` command is used. The difference between these was explained in Section 3.1.1. If an exception occurs in the chaincode, then the Python Blockchain Interface raises a `ChaincodeError` exception. The output from the CLI container is returned as a string from this method.

The Python Blockchain Interface was the most impacted by the time constraints of the project. Originally, I had planned to make use of the Hyperledger Fabric SDK for Node.JS. This would allow the web application to directly communicate with the chaincode, without the use of Python. The use of the command-line interface introduces some issues when dealing with large elections as the data becomes too large to be passed in a single command-line argument.

Specifically, if the election administrator is adding all the voters to the election register via the `add_voters_to_register` method or adding voters to the election via the `add_voters_to_election` method, only around 1000 voters can be passed in at a time. If more are passed in, the limit for the length of command-line arguments is reached. This means for large elections, this method needs to be called multiple times.

The impact of this will be further discussed in the Evaluation section of the report.

If the Node.js SDK was used, this would likely not be an issue. Alternatively, the Python SDK could be investigated. This would allow the main structure of the interface to be kept, with the internals of the methods swapped out. This would likely require the least rework to the project.

3.2 Challenges

This section will describe the problems I have faced while implementing the system and how they were dealt with.

Go

I had never used Go before starting this project. I had hoped to pick up the language with relative ease but this was not the case. Last year, I worked for BT as a Software Developer for my industrial placement. During that year, I picked up many new languages such as C# and adopted a Behaviour Driven Development (BDD) approach.

I found C# relatively easy to pick up as syntactically, it is very similar to Java which I had some experience with. I felt my placement really strengthened my skills as a developer and I became much more confident with programming in general.

Before starting the project, I was not concerned about learning a new language, Go, for chaincode development because of this experience. However, I did find the language harder to pick up than I was anticipating. This is mostly because Go's syntax is most similar to C, a language I am not familiar with. It also has some unique syntax such as the declaration syntax²⁰. Go also has some concepts that I have not come across in other languages such as pointers.

By the later weeks of the project, I did start to feel more comfortable with Go. I have used a range of packages and language features in the chaincode. However, there is still much of the language that I am unsure of the best practices for. My approach on learning Go could have been better.

Understanding Hyperledger Fabric

As someone with no experience in blockchain, it took some time for me to understand the mechanics of Hyperledger Fabric. I spent the first two or three weeks of the project just reading through the documentation and making notes to ensure I understood how it worked.

The documentation is very well written and is easy to follow but the beginning sections are theoretical, it takes some time before any code is reached. This was definitely an important stage of the project as I feel if I dived straight into code, I would have been overwhelmed with trying to learn too many new things at once.

Chaincode Development

As briefly mentioned earlier, it can be very time consuming to develop and test chaincode. First, the docker containers for the network must be brought up with takes several minutes. Then, the chaincode must be installed on all the nodes of the blockchain and the channel before it can be tested. This typically takes several minutes and can make debugging very time consuming. Every change to the chaincode requires it to be reinstalled.

I cannot express how frustrating this was at times. For example, imagine a scenario where the chaincode was throwing an error. You are unsure of the likely cause and

²⁰ Details on how Go's declaration syntax differs from other languages can be found here: <https://blog.golang.org/gos-declaration-syntax>

wanted to perform some basic steps like printing out the values of some variables to ensure they were correctly set.

First, you make the changes to the chaincode, you reinstall it to the whole network. Minutes later, you run the script and discover that all those variables are correctly set and that is not the issue. You think of another potential cause of error, maybe a function call is resulting in an error. You add in a print statement before and after the function call. You reinstall the chaincode, this takes several minutes.

You find the print statement you put after the function is not present in the output. Brilliant. You have narrowed down the problem to that function. You check the variables in the function are set correctly. You reinstall the chaincode, this takes several minutes. Ah, finally you found the error is a variable is null that should not be null due to a small typo! You look at the time. It has taken an hour to fix this one issue.

Contrast this to traditional development you simply could have inserted a breakpoint and stepped through the code to find the issue within minutes.

This made the chaincode development take much longer than I had originally envisioned in the initial plan. I had estimated that by half way point, I would have finished the chaincode and would be in a position to spend more time on this report and the web application.

Near the end of the project, I discovered that Hyperledger Fabric does provide a 'dev' mode that is intended to speed up this process²¹. As I discovered this so close to the end of the project, I did not investigate it but this likely would have development much less frustrating.

Scale of Elections

This is similar to the Chaincode Development point. When users act in the blockchain system, their actions are recorded as transactions. This takes time because transactions must be endorsed and verified by the other peers on the system. Thankfully, this is a matter of seconds rather minutes.

However, this means that testing the chaincode with large elections with 1000-10000 voters can be very time consuming, as each voter casting their vote takes a few seconds. This was particularly time consuming when I was facing issues with large elections due to various issues like the maximum length of command line arguments being reached and timeouts.

²¹ <https://hyperledger-fabric.readthedocs.io/en/release-1.4/peer-chaincode-devmode.html>

I did improve the speed at which simulated elections take place by making use of the multiprocessing module in the Python Blockchain Interface to allow votes to be cast by multiple voters in parallel. This was a significant time reduction, but very large elections still take around an hour to simulate. Perhaps this could have been further improved by simulating elections on a more powerful machine.

Race conditions

One issue I have faced during the project that I have been unable to directly resolve is a race condition that occurs when two transactions modify the same object on the blockchain. I first encountered this issue when I was recording the votes that had been cast. I had originally intended to record votes in 2 parts.

The first part of the vote would be the vote as it is now. The second part would be a list containing the candidate portion of all votes. This would be to enable the counting of votes but just retrieving a single object from the blockchain, rather than every vote. I thought this would give better performance for counting the results of an election.

After I had completed the chaincode, I noticed that the votes were not being tallied correctly. Eventually, I realised that because the votes were being cast in parallel, multiple transactions were modifying the list of votes concurrently.

I had expected Hyperledger Fabric be able to handle this and process the transactions correctly such that the vote list was accurate but it seems this is not the case. Upon further research²², I found these concurrent transactions should be throwing a chaincode error at some point but for whatever reason, this wasn't happening in my code.

To solve this problem, I reworked the voting to just create one object and the counting part to retrieve all vote objects with the specified election ID. This avoided the issues with concurrent transactions as they were now creating distinct objects rather than modifying the same object.

However, this problem still occurs with the registration chaincode. The same solution could be applied the register, having each voter register themselves in a separate object. This would not be challenging to implement as it would be very similar to the vote objects. I have simply not completed this due to time constraints.

²² Details: <https://medium.com/wearetheledger/hyperledger-fabric-concurrency-really-eccd901e4040>

An alternate solution could be add some sort of queue system to the Python Blockchain Interface to stop multiple transactions of the same type occurring at the same time. However, the best solution would be to eliminate the problem from the chaincode itself and support a higher volume of transactions.

Testing

As mentioned earlier in this section, during my industrial placement last year, I used a Behaviour Driven Development approach to tasks. I found this, at first, very challenging to adapt to but after a while, it became second nature and helped me to write concise, correct code.

I had hoped to apply what I had learned to follow a Test-Driven Development (TDD) approach to writing the code for this project. However, due to my lack of familiarity of Go, I decided that this was not feasible for the chaincode with the timespan on the project.

It would have been great benefit to have tests to aid in the development of the project, avoiding situations as described in the Chaincode Development section and to verify the correctness of the code. It is particularly relevant for a voting application to have verifiable correctness.

However, with my lack of knowledge about Go and the Hyperledger Fabric SDK, I felt it would have been too much to learn all at once and could have stunted my progress on the project. It would have been challenging to write comprehensive tests for a language where I am unsure of the best practices.

I am also unsure of the best way to write and run tests for chaincode. It appears that there are a few options. The Hyperledger Fabric SDK does provide a `MockStub` class for unit testing chaincode, but it is not covered in the documentation. There is also a library named `cckit` that extends the `MockStub` class and claims that the default implementation is incomplete²³.

²³ "Uncompleted testing tools (MockStub)" - <https://github.com/s7techlab/cckit>

4 Results and Evaluation

4.1 Performance Testing

4.1.1 Methodology

This section will evaluate the performance of the election system. Specifically, the performance of the Hyperledger Fabric Blockchain chaincode described in Section 3.1.1 will be evaluated.

The performance will be evaluated by measuring the amount of time various chaincode operations take and investigating how this scales with the numbers of voters in the election. The actions that will be evaluated are the ones present in the UML Sequence Diagram in Figure 3-3. Each operation will be performed 20 times and the results averaged to account for any variance in terms of other processes running on the operating system.

The sequence of actions mostly follows the UML Sequence Diagram in Figure 3-3. The main divergence from the Sequence Diagram is that the election administrator is registering the voters in bulk rather than the voters registering themselves.

The main reason for this is avoid the simulation taking a very long time to execute as that is the part of the simulation that takes longest to execute is the voter's casting their votes. Voter's registering themselves for elections one by one would take a comparable time to execute.

In order to time the actions, the time taken between when the Python Blockchain Interface runs a command in the CLI and when a response is received will be measured and saved to a file. This file will later be used to graph the results using the Python library Matplotlib.

Elections with 10, 100 and 1000 voters will be included. I will also include times for an election with 10,000 voters but this will not be repeated 10 times due to the extreme length of time this would take as described in Section 3.2 under Scale of Elections. The 10,000 times should be taken with a pinch of salt compared to the others for this reason, but I believe it is worth including them for greater context on how the system scales.

I ran 10 voters, 100 then 1000 voters all 10 times each. I then restarted the blockchain network and ran in the reverse order. I reversed the order because I thought that the

more objects stored on the blockchain, the slower the network could potentially become. I also wanted a larger dataset to work with to obtain more meaningful results. After this, I restarted the network a final time and ran a 10,000 voter election.

For an unknown reason, the first chaincode invocation after the network has been started takes a very long time, around 30 seconds. Perhaps Fabric does not start CouchDB until it is used in the chaincode. I have considered this to be an outlier and removed one instance of this occurring from the data, namely in three calls to `InitElection`.

4.1.2 Results

Before running this test, my expectations were that operations that did not depend on voters would have a very similar execution time across all elections. For example, creating the election, starting and finishing the election and casting votes.

I also expected that for elections with less than the maximum command line argument length number of voters²⁴, operations such as creating and populating the voter register, and adding the voters to the register would take roughly the same amount of time. However, for 10,000 voters these operations would take a significantly longer time as 10 transactions, each containing 1000 voters would have to be used in place of 1 up to 1000 voter transaction containing all voters.

The time taken to count the election should scale the worst of the actions as this is of course the most dependent on the number of voters. All the votes for the election must be retrieved in order for the election to be counted, the more voters, the more votes to be retrieved, the longer database queries take to complete.

All results are being presented with a logarithmic scale on the x axis representing the number of voters with four data points at 10, 100, 1000 and 10000 voters. The y axis shows the time taken to complete the action in seconds on a linear scale.

When interpreting these results, it is important to remember that the data for 10, 100, 1000 voters is the average of 10 iterations but 10,000 is from a single iteration due to the length of time an election of the scale takes to simulate.

²⁴ As mentioned previously, this was around 1000 voters in my experience but there could be more or less before the limit is reached depending on the length of voter identities. The voter identities were around 12 characters during this testing as they were simply given the identity voter—00001 through to voter--10000.

Figure 4-1 – Actions unimpacted by the number of voters

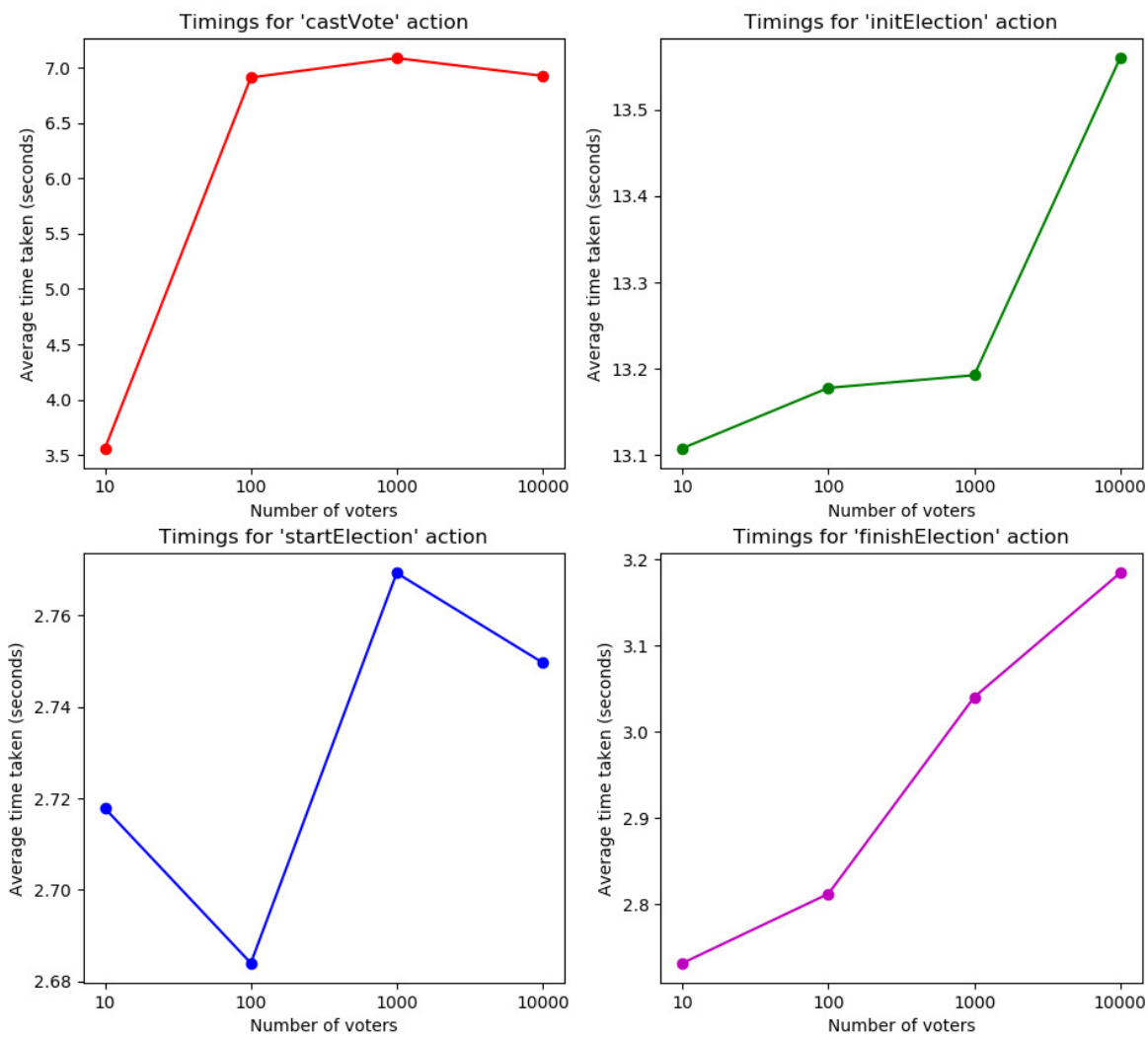


Figure 4-1 shows the actions that are relatively unaffected by the number of voters.

The action to create the election, 'initElection' only has a 0.4 second difference between an election with 10 voters and 10,000 voters. The shape of the graph is slightly misleading as it appears to indicate a large increase in execution time compared to smaller elections. This difference can be attributed to variance, recall that the other data points on this graph are an average of 10 iterations whereas the 10,000 voters is using a single election. Recall that at this stage, there are no voters in the election so creating an election is always the same operation regardless of the number of voters because the voters are added in later operations.

The action to start the election, aptly named 'startElection' also shows a very small difference between the smallest election of 10 voters and the largest of 10,000 voters with only around 0.3 seconds difference between them. In this graph, we can see that the election with 100 voters was the fastest and the slowest was 1000 with roughly a 0.9 second difference between them. This shows that the small time difference is likely variance due to elements like background processes on the host machine rather than the time taken being directly correlated to the number of voters.

The action to finish the election, 'finishElection' appears to take a linear shape with the time taken increasing in respect to the number of voters. Again, the time difference is very small with a difference of roughly 0.6 seconds between the highest and lowest data points. I believe the shape of this graph is a coincidence with this specific dataset, if the experiment was run again, it is likely a different shape graph would be obtained. This is due to the very small difference in the time taken.

The final action shown in this graph is the action to cast votes, 'castVote'. The time taken is very consistent hovering around 7 seconds for 100, 1000 and 10000 voters. However, for 10 voters the time taken is roughly half this at 3.5 seconds. This results may appear strange at first but I believe this difference is likely down to the Python code used to simulate elections rather than the chaincode itself.

As mentioned in Section 3.2, Scale of Elections, I made use of the Python multiprocessing library to allow votes to be cast in parallel to speed up the overall time taken to simulate large elections. This library creates additional processes to perform a specified function, in this case the cast_vote function in the Python Blockchain Interface. After some simple trial and error, I found using 20 processes gave the best performance.

If there are only 10 voters in the election, only 10 processes will be used. Perhaps the overhead associated with 20 processes in large elections is slowing down the host computer enough that blockchain operations are taking longer. I suspect that if the network was being hosted on a separate machine, this difference would be eliminated.

Regardless of all of this, it is clear from the graph that the number of voters is not directly correlated with an increase in time taken to cast a vote.

Figure 4-2 - Actions impacted by a large number of voters

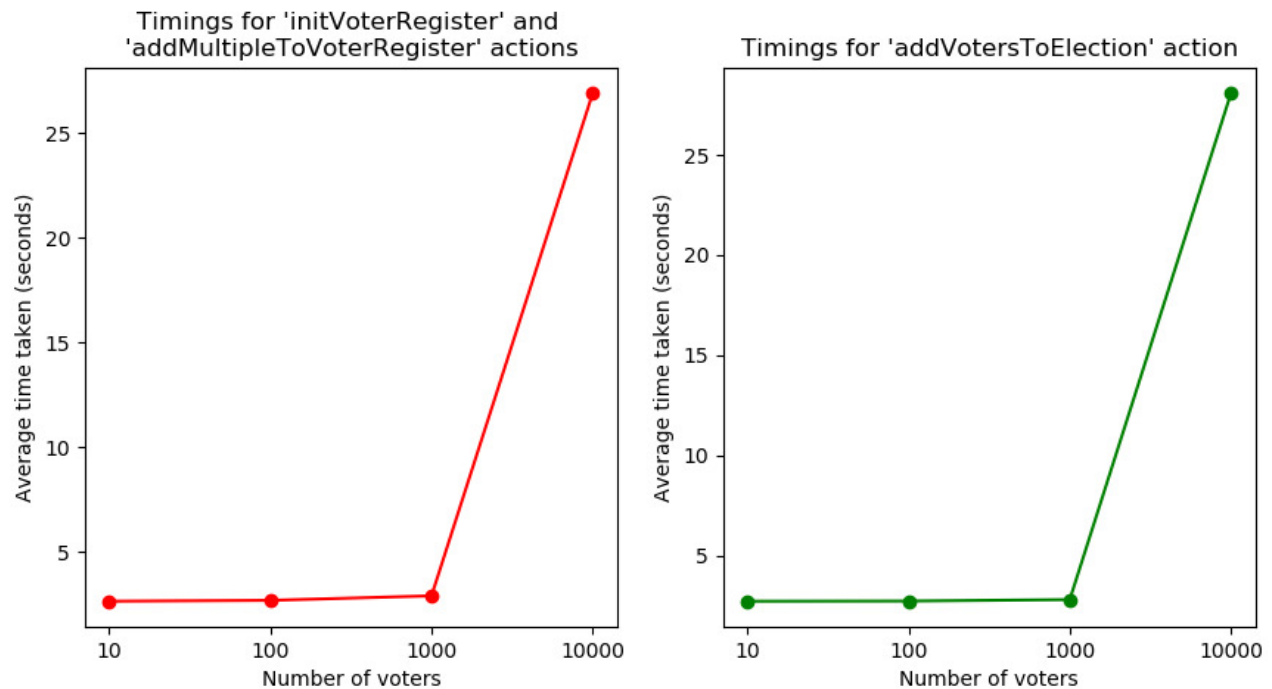


Figure 4.2 shows the actions that are impacted by a large number of voters, over 1000. These actions are impacted due to the maximum length of command line arguments being reached, requiring input to be broken up into multiple transactions.

The first graph in this figure shows the time taken to complete the combination of two actions, creating the voter register ('initVoterRegister') and adding voters to the register ('addMultipleToVoterRegister'). As previously mentioned, for this testing, the election administrator is adding to the voters to the register in bulk rather than the voters registering themselves.

For elections with around 1000 voters or under, all the voters in the election can be added to the register when it is created in 'initVoterRegister', this means a complete voter register can be created in a single transaction. However, for an election with 10,000 voters, 10 transactions must be submitted. One transaction with 1000 voters in 'initVoterRegister' and nine transactions with 1000 voters each in the 'addMultipleToVoterRegister' method.

The impact of this is that the smaller elections all take a very similar amount of time to execute, around 2 seconds while the 10,000 voters election takes exponentially longer at around 29 seconds.

The same holds true for the 'addVotersToElection' action as seen in the other graph. The timings are very similar as it is the same user input of voters being passed to both functions.

As previously mentioned, this is an issue that could be resolved by replacing the Python Blockchain Interface to no longer use the Hyperledger Fabric CLI and instead using the SDK to invoke the chaincode. This could allow for more data to be passed in a single transaction which would greatly improve the performance of these actions.

Figure 4-3 - Actions impacted by number of voters

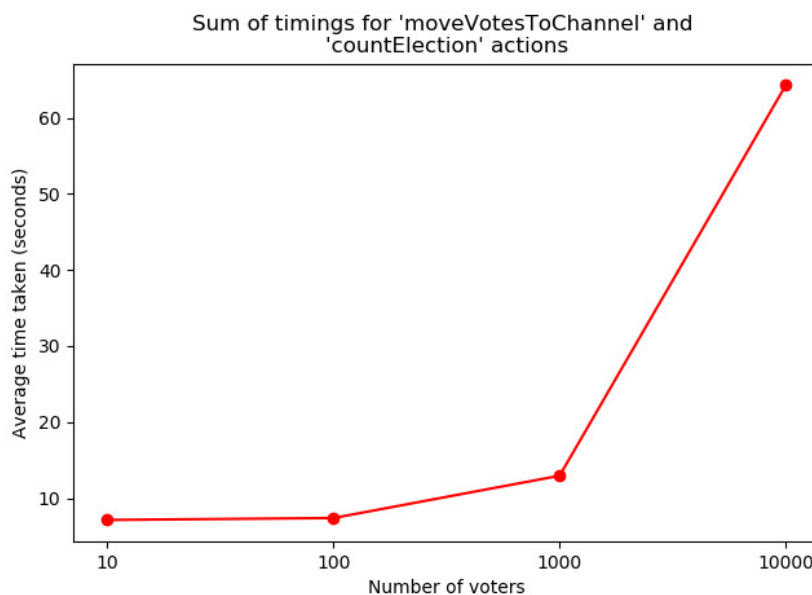


Figure 4-3 shows the action that is the most impacted by the number of voters, the counting of the election. Note that this is the sum of the time it takes to perform two actions, `moveVotesToChannel` is simply a pre-requisite action required to count the election. This is an implementation detail, due to the use of private data collections, it is to read all votes and write the election results to the blockchain in a single transaction. Reads and writes must be split into different transactions.

As the number of voters increases, the time taken to count the election also increases linearly. This is because, as previously mentioned, every vote object must be retrieved to count the election.

This time could potentially be improved. The documentation for Fabric recommends the creation and use of a CouchDB index for better performance with queries. I did attempt

to define and use a query for the count operation but I am unsure if it is functioning correctly and due to time constraints, was unable to fully explore this. This could lead to a very large increase in the performance of this action, which currently, is the worst scaling action in the system.

4.2 Error Handling

As mentioned in Section 3.2 Challenges, chaincode development is challenging and it can be difficult to unit test. For this reason, this project lacks any kind of unit testing. However, throughout development I have performed manual tests to ensure the correct operation of the system.

While I have not produced formal test cases or automated unit tests, I am confident in the correctness of the system. There are many checks in the chaincode to throw errors to ensure the correct operation of the system. For example, logically, an election should not be able to be finished before it has started. Attempting to finish an election that is not in the started state would throw a chaincode error.

Start dates must be before end dates to give another example. An attempt to start an election at a time before the start date would also result in an error.

It would be valuable future work to write unit tests to test all these unhappy paths, as well as the happy paths of the system. This would help gain further confidence in the system, which is very important in this domain. As mentioned in Section 3.2, the `cckit` library may be suitable for this.

The Flask Client Application described in Section 3.1.2 does not feature robust error handling. It instead sends all requests to the chaincode and relies on the chaincode throwing an error for invalid input. In many cases, such as the date checking described above, this check could happen inside the Flask Client Application before the chaincode is called to avoid unnecessary calls to the chaincode.

Calling the chaincode unnecessarily is time-consuming and redirects the user to an error page – if the Flask Client Application was made more robust, the user experience would be much improved.

The reason why the Flask Client Application lacks robustness is primarily due to time constraints and it being a low priority to implement. It would not have been challenging to implement as it would have been simply repeating work I did in the chaincode in another programming language. However, it is certainly a weakness of the project.

4.3 Security Properties Evaluation

In the introduction this report, I identified the following as desirable security properties for the system:

1. Confidentiality - No person can tell how another person voted
2. Integrity - No person can modify or delete, nor can they insert a false vote into the system
3. Authentication - Only authorised parties can vote
4. Anonymity - It is not possible to tie someone's identity within the voting system to their real-life identity
5. Availability – Voters should always have access to the system when requested

6. Integrity – No unauthorised person can modify an election. No person can delete an election
7. Authentication - Only authorised parties can create and modify elections
8. Availability – Election Administrators should always have access to the system when requested

The Hyperledger Fabric Blockchain that has been developed does implement most of these properties. For example, Integrity in terms of both the Voters and Election Administrators has been achieved as it is not possible to modify or delete a vote in the system, nor delete elections. Unauthorised parties also cannot modify elections.

The fact Hyperledger Fabric is a private, permissioned blockchain framework provides authentication – access to the system is governed by which organisation someone belongs to and the Election Administrators operating the blockchain control who they give access to which organisation.

The distributed nature of Hyperledger Fabric also provides availability of the system – a single node is not a single point of failure the system can continue operate without it. However, the current network, making use of the build your first network script does not provide availability as there is a single point of failure.

All the nodes are hosted on a docker container on a single machine, if the host machine was comprised then the blockchain network would be as well. This is not a large weakness of the system though as this specific network setup is only intended for development purposes and would be replaced in a production version of the system.

Whether confidentiality has been achieved is debatable. I believe the system has achieved confidentiality between voters. No voter can decrypt another voter's vote and see how they voted, even after the election has finished. However, Election Administrators are able to obtain the decryption key. Again, this is arguably not a large weakness of the project, as in paper elections, all votes are plaintext but anonymous.

Regarding anonymity, the blockchain system on its own does not provide it. An identity for within the blockchain system is needed for voters to vote. It is up to the Election Administrators to provide a system to securely store these identities and tie them to some sort of authentication system such as a username/password system or biometrics.

The Flask Client Application provided does not implement anonymity currently but the system could be extended to support this.

5 Future Work

5.1 Blockchain Network Ordering Service Improvements

As briefly mentioned in Section 3.1.1, the byfn network uses the Solo ordering service by default. As the name implies, in this ordering service, a single node is responsible for ordering. As it is a single node, it is not fault tolerant and this ordering service should not be used for a production environment. It is mostly intended to be used for prototyping and testing.

Whilst I was working on the project, the only alternative to Solo was Kafka. Kafka is a Crash Fault Tolerant (CFT). Kafka is known to be challenging to set up and manage so I did not explore it for the project due to time constraints.

On the 11th April 2019, a new version of Hyperledger Fabric, v1.41, was released. This release included an implementation of a new CFT ordering service called Raft. Raft is said to be easier to set up and manage compared to Kafka.

Changing the network to use Raft or Kafka rather than Solo would be an important piece of future work that would help the system get closer to being production ready. Had Raft been released when I started the project, I would have investigated using it rather than Solo.

5.2 Replacement of Python Blockchain Interface

As mentioned throughout the report, the current implementation of the Python Blockchain Interface is not ideal.

Specifically, elections with over 1000 voters are greatly affected by the current implementation. As discussed in Section 4.1, Performance Testing, some operations that could usually take place in a single transaction such as adding the voters to an election, must happen over multiple transactions due to the maximum length for command-line arguments being reached.

Another weakness of the current implementation is the fact that all commands are ran on Node 0, Org1. Ideally, Election Administrator commands should be balanced between the two Org1 nodes and Voter commands balanced between the two Org2 nodes.

This could be a security issue as the voter commands are being run on a peer that has access to create and modify elections. Perhaps it would be best to separate the Interface into two parts, one for Voter commands and for Election Administrator commands to isolate them from each other.

Replacing the Python Blockchain Interface to either use the Python SDK or moving the interaction with the Blockchain to use the Node.js SDK are both valid options. This would drastically improve performance for larger elections and likely bring performance benefits to other actions as well, as the code can speak directly to the blockchain without the need to send a command to a container.

5.3 Race Condition Elimination

As mentioned in Section 3.2, there is currently a race condition that occurs when two voters register for an election at the same time. A high priority piece of future work would be to rework the voter register system to eliminate this race condition.

This could be done by changing voter registration from a single object for an election to each voter creating a unique object. This would require some rework as this object would require a unique id of some sort, the Election ID and the voter's identity. The unique ID could simply be a combination of the Election ID and voter's identity, similar to how the vote IDs are generated.

This rework would be quite similar to how votes are recorded and stored in the system. The relevant chaincode functions could be used as a base for this work and modified to have the fields described above.

5.4 Testing

Again, as mentioned in Section 3.2, the project currently lacks unit tests. If the system was ever going to be used in a production environment, unit tests would be critical to verify the correctness of the system. In the domain of e-voting, failure of the system in any way could be very damaging to not only this project, but to confidence in all e-voting systems.

As previously mentioned, Hyperledger Fabric SDK does provide a `MockStub` class for unit testing chaincode. There is also a library named `cckit`. These appear to be the best two options to investigate for being to unit test the chaincode.

5.5 Investigating Other Blockchain Frameworks

When this project was started, Hyperledger only had two production ready frameworks, Fabric and Sawtooth. Since that time, two more frameworks, Iroha and Indy have become production ready. Some of the properties of these frameworks such as the distributed identity management of Indy and the performance of Iroha could be very valuable for the use case of voting.

It would be worthwhile to spend time investigating these frameworks and see if some of the weaknesses of this project in terms of performance and identity management could be solved by using either of these frameworks instead of, or in tandem with Hyperledger Fabric.

6 Conclusions

In conclusion, this project aimed to deliver a secure system to store election data. The security of the system was measured by several properties such as confidentiality, integrity and authentication.

I have delivered a blockchain system that implements the majority of these properties in full or partially. The main property missing from the project is voter anonymity. Voters need to be given an identity for use within the blockchain that only they can access but cannot be tied to them. I have not delivered a way to achieve this but have described how it could be done through a username/password system or biometrics.

Performance testing was done on the system and the results showed a system that performed well for elections with around 1000 voters. Larger elections are possible, but take significantly longer. The reason for this is the reliance on the command-line interface for Hyperledger Fabric to deliver commands to the network rather than the use of an SDK. The use of an SDK to invoke the chaincode is a key piece of future work for the project to improve performance.

I have also delivered a web application to act as user interface for the blockchain system. While this application is not production ready, I believe it is valuable for demonstrating the system and could easily be expanded upon in the future to be more secure.

Thank you for taking the time to read this report. This project has been a long and arduous task but overall, I have enjoyed it.

7 Reflection

7.1 Learning

I have of course learnt many new technologies over the course of this project. Namely, I have used Go, Flask, Hyperledger Fabric and CouchDB all for the first time. I also gained some further experience with Python.

More generally, I have learnt a lot about lower level programming through Go. This is something I didn't have much experience with before this project so it was valuable to have some exposure to, rather than learning yet another high level language.

I also learnt a lot about blockchains and other DLTs. I learnt about how they work, more specifically how Hyperledger Fabric works and compares to other blockchains. I found out a lot about the state of Blockchain industry in terms of who is involved and what projects are out there. It was eye opening to learn that there is so much more to blockchains than crypto-currency.

This is the first long-form report I have written with references, a table of figures etc. I have learnt about finding reliable sources and how to structure a report like this.

7.2 Project Management

I believe my project management skills for this project were fairly poor. I did not track the time I had spent on the project nor did I use any kind of issue or task tracking software.

I was always aware of what my next task was and what my overall progress was on the project, but it would have been a much better approach to formally record this instead of keeping it all in my head. This poor management has not hindered my overall progress I feel, but it has likely made it more stressful than it could have been.

I found myself often stressing about how much time I was spending on the project and whether I was achieving enough. Some days, I would push myself very hard and make a lot of progress but at the cost of burning myself out and making no progress for the next couple of days.

If I had spent a consistent amount of time on the project each day and tracked this time, recording what tasks I spent time on as well as allocating scheduled break days, this

could have helped keep the stress levels lower and I could have made some more progress.

In my initial plan, I suggested that I might use a Kanban board to identify and keep track of the progress on tasks. Once I had started on the project and the scope was initially fairly small, I felt this wasn't necessary as I was completing one simple task at a time. However, as the scope of the project grew this could have been a fantastic way to track tasks.

My experience working on this project has taught me that I find this self-driven, independent style of working very difficult. I found myself much less stressed while working as a Software Developer for BT as I was being managed and given direction. At the same time, I did enjoy the freedom of being able to steer the project and focus on what interested me.

For future work like this, that is fairly unstructured and self-driven, I would definitely make more use of something like a Kanban board to keep track of tasks. Even if these tasks aren't strictly software development related, they can still be modelled using a Kanban board. Alternatively, GitHub features an issue tracker that would also be suitable.

Abbreviations

DLT – Distributed Ledger Technology

CLI – Command Line Interface

References

- [1] R. B. Lee, “Security Basics for Computer Architects,” in *Synthesis Lectures on Computer Architecture*, vol. 8, no. 4, 2013, pp. 1–3.
- [2] P. of New Brunswick, *A pathway to an inclusive democracy* New Brunswick Commission on Electoral Reform. 2017.
- [3] S. Nascimento, A. Pólvara, and J. S. Lourenço, “#Blockchain4EU : blockchain for industrial transformations.,” Luxembourg: Publications Office, 2018, pp. 13–19.
- [4] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” in www.bitcoin.org, 2013.
- [5] J. Kwon, “TenderMint : Consensus without Mining,” 2014.
- [6] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, “FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second,” 2019.
- [7] N. Kshetri and J. Voas, “Blockchain-Enabled E-Voting,” *IEEE Softw.*, vol. 35, no. 4, pp. 95–99, 2018.
- [8] F. P. Hjalmarsson, G. K. Hreioarsson, M. Hamdaqa, and G. Hjalmtýsson, “Blockchain-Based E-Voting System,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, vol. 2018-, pp. 983–986.
- [9] C. K. Adiputra, R. Hjort, and H. Sato, “A Proposal of Blockchain-Based Electronic Voting System,” in *2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, 2018, pp. 22–27.
- [10] B. Yu et al., “Platform-Independent Secure Blockchain-Based Voting System,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 11060, pp. 369–386.

Appendices

User Guide

1. Obtain the prerequisites for Hyperledger Fabric by following the steps on this page: <https://hyperledger-fabric.readthedocs.io/en/release-1.4/prereqs.html>
2. Follow the steps at this page to obtain the Docker images and examples for Hyperledger Fabric: <https://hyperledger-fabric.readthedocs.io/en/release-1.4/install.html>
3. Unzip the code submission for this project found on PATS into the chaincode folder of the fabric-samples repository downloaded in step 2
4. Navigate to the /scripts/ folder and execute the init_network.sh script in a terminal emulator such as Git Bash on a Windows system. This script will take some time to execute. Once init_network.sh has finished, the blockchain network will be up with the chaincode installed.
5. Navigate to the /scripts/Python/ folder and run the following command:
 Unix: source venv/bin/activate
 Windows: source venv/Scripts/activate
 This activates the Python Virtual Environment²⁵, avoiding any issues with missing dependencies.
6. Run server.py to start the Flask Client Application. A web server will begin to run at <http://localhost:8000> . It may take approximately 30 seconds to load the homepage for the first time as the first command sent to the blockchain network takes a long time to execute.
7. Optionally, see the populate_network.sh script in the /scripts/ folder to see how to send commands to the Blockchain through the cli container.

²⁵ An explanation of Python Virtual Environments can be found in the documentation at: <https://docs.python.org/3/tutorial/venv.html>