Permissioned Distributed Ledgers for Land Transactions

D. T. Fernando

Permissioned Distributed Ledgers for Land Transactions

D. T. Fernando
Index No : 14000342

Supervisor: Dr. D. N. Ranasinghe

December 2018

Submitted in partial fulfillment of the requirements of the B.Sc in Computer Science Final Year Project (SCS4124)

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name: D. T. Fernando

……………………………………………

Signature of Candidate   Date:

This is to certify that this dissertation is based on the work of

Ms. D. T. Fernando

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Principle/Co- Supervisor's Name: Dr. D. N. Ranasinghe

……………………………………………

Signature of Supervisor   Date:

Abstract

Considering the inefficiency and ineffectiveness of the current manual land registration systems being practiced in Sri Lanka and the emergence of the concept of blockchain based land registries as a successful replacement for badly kept, mismanaged and/or corrupt land registries from around the world, this research proposes a permissioned distributed land ledger solution for Sri Lanka.

The proposed solution is an island wide unified land ledger which addresses unequal regional land transaction density conditions across the island, as opposed to the present regional ledger system. The final solution presents optimal content for the ledger (extracted from the current folio), has reassigned duties to state validators and has got away with the folio system while ensuring derivation of the pedigree/ folio tree for a land at a given time.

The proposed solution was implemented using Hyperledger Fabric v1.2. It was evaluated for performance on an AWS t2.large instance with 2 vCPUs, 8GiB memory, against the implementation of a regional distributed land ledger, under different land transaction density conditions and failure conditions. The proposed solution records higher throughput, lower latency and tolerance for fail-stop conditions than the regional distributed land ledger. Further, the proposed solution does not show a significant drop of throughput up to two crash failures in production scale deployment.

Preface

Through this research, an island wide, unified permissioned distributed land ledger has been proposed for Sri Lanka. The optimal content included in the final solution was derived based on the content of the current folio. Design of transactions executed against the land ledger was a result of closely studying the Sri Lankan land transaction scenario. Thus, the design of optimal content and transactions is solely based on my own work. The regional distributed land ledger (identified as Abstract Model 1 in the dissertation) was designed to closely map the current manual land transaction scenario in Sri Lanka. Subsequent to identification of possible drawbacks of the regional distributed land ledger, island wide unified land ledger (identified as Abstract Model 2 in the dissertation) was designed by myself, under supervision. The design of regional distributed land ledger or island wide unified land ledger has not been proposed in any other related work. Implementation of the two Abstract Models was performed by myself. Evaluation model for the performance evaluation was devised by referring related performance evaluation studies by myself under supervision.

Acknowledgement

I would like to express my sincere gratitude to my research supervisor, Dr. D. N. Ranasinghe, Senior Lecturer at the University of Colombo School of Computing for the continuous guidance and supervision provided to me throughout the research.

I would also like to extend my sincere gratitude to Dr. M. D. J. S. Goonetillake, Senior Lecturer at the University of Colombo School of Computing and Dr. C. I. Keppitiyagama, Senior Lecturer at the University of Colombo School of Computing for providing feedback on my research proposal and interim evaluation to improve my study. I would also take the opportunity to acknowledge the assistance provided by Dr. H. E. M. H. B. Ekanayake as the final year computer science project coordinator.

This thesis is dedicated to my loving parents who have been an immense support to me throughout this journey of life. It is a great pleasure for me to acknowledge the assistance and contribution of all the people who helped me to successfully complete my research.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AM1	Abstract Model 1
AM2	Abstract Model 2
BFT	Byzantine Fault Tolerance
CFT	Crash Fault Tolerance
DLT	Distributed Ledger Technology
FLP	Fischer, Lynch and Patterson result
IaaS	Infrastructure as a Service
MSP	Membership Service Provider
OSN	Ordering Service Node
PoW	Proof of Work
RLR	Regional Land Registry

Chapter 1 - Introduction

1.1 Background to the Research

Currently, two types of land registration systems are being practiced in Sri Lanka. They are 1) the Deed Registration System and 2) the Title Registration System. Deed Registration System which was introduced during the colonial era has several disadvantages. During deed registration, legal documents are registered rather than title to the property, it does not consider physical existence of the boundaries to the land and it is unable to determine the real economic value to a particular piece of land.

As a remedy to the above mentioned drawbacks of Deed Registration System, Title Registration System was introduced by the Sri Lankan government through the Registration of Title Act no. 21 of 1998 (RTA). Title Registration System is a complicated, expensive and time consuming task which involves a large number of institutional players. However, according to the Implementation Completion and Result (ICPR) report 2007 published by the world bank, the land titling project which was implemented with 5 million US Dollars under the World Bank funds had been unsatisfactory. Subsequent to the failure of Land Titling system and termination of World Bank funding, the titling project continues as “Bimsaviya” national land titling project [1].

However, it is evident that regardless of all the efforts, Sri Lanka does not have a sufficiently effective and efficient administrative framework for land registration. There are many negative implications of the present land registration systems in Sri Lanka, such as existence of a large number of unsolved land disputes, litigation and unclear tenure leading to land encroachment, misuse and disuse of land.

As stated in [1], it is important that current land registration systems as well as newly proposed systems/ strategies should enforce **pragmatic** decisions rather than relying on too standardized, bureaucratic and costly approaches. While exploring for pragmatic approaches taken by other countries in the world in order to improve the efficiency and effectiveness of their land registration systems, it could be observed that some countries in the world have turned their attention towards implementing **blockchain based land registries**. As stated in [2], a badly kept, mismanaged and/or corrupt land registry could be successfully replaced by a blockchain based land registry, because of the added value of cryptographic auditability. However, it could be observed that, not only countries with badly kept land registries (e.g.: Honduras, Ghana) but also countries with already well-functioning land registries (e.g.: Sweden, Georgia, Estonia) are in the process of implementing and deploying blockchain based land registries. Thus, it is worthwhile to explore the suitability of a blockchain based land registry or an equivalent approach to make Sri Lankan land registration system more efficient and effective.

The concept of blockchain emerged with bitcoins. Later, the Distributed Systems community generalized the concept of blockchain to distributed ledgers. As claimed by concepts of Distributed Systems, **blockchain is one type of distributed ledger**. Thus, through this research, it is intended to provide a distributed ledger solution to the Sri Lankan land transaction scenario.

1.2 Justification for the Research

A Distributed Ledger Technology (DLT) network is a collection of interconnected nodes where, each node maintains a copy of the same database, called the **ledger**. In DLT, there is no centralized database which is controlled or administered by a central party that is trusted by every participant. The process of updating the distributed ledger requires exchanging transaction information between nodes, achieving distributed consensus among nodes, followed by adding the validated transaction as a new ledger entry.

If blockchain is the underlying database structure of the ledger, the ledger could be identified as a hash chain over blocks. Thus, during the last step of updating the distributed ledger (‘adding the validated transactions as a new ledger entry’), validated transactions are grouped into blocks and appended to the ledger (i.e. the blockchain).

Distributed Ledgers have several advantages over traditional (centralized) databases. DLT provides a full audit trail of information history, provides accessibility to a common view of information to all nodes at the same time and it is impossible to make unauthorized changes to the distributed ledger. In addition to the general advantages of distributed ledgers, DLT is inherently suitable for implementing a land ledger because it facilitates storing digital records of assets in blocks. When, new information regarding a land asset is created (e.g.: when a new land is registered) as well as when existing information about a land asset changes (e.g.: when the owner of a land changes), new blocks are formed and securely chained to the previous one.

There are two main types of distributed ledgers, namely, unpermissioned (permissionless) distributed ledgers and permissioned distributed ledgers [9]. An unpermissioned DLT network is accessible to anyone, i.e. all participants are public nodes, while a permissioned DLT network contains an authorized consortium of participants. Procedure of obtaining distributed consensus in an unpermissioned DLT network is through “Proof of Work” (PoW) mining, while, in a permissioned DLT network distributed consensus is obtained through validation by a selected subset of ‘trusted validating nodes’. In the Sri Lankan land transaction scenario, Regional Land Registries (RLRs), notaries and surveyors could be identified as trusted validators of land transactions recognized by the government. Thus, it is evident that implementing the distributed ledger solution as a permissioned DLT network is more suitable when similarity with the real scenario is considered.

In addition, validation performed in permissioned DLT networks is more energy-efficient than the resource intensive PoW mining performed in unpermissioned DLT networks. Speed of validation process when updating the ledger and security is higher in permissioned DLT networks than in unpermissioned DLT networks. Apart from that, operational costs of permissioned DLT networks is lower than that of unpermissioned DLT networks. Thus, it could be inferred that a permissioned DLT solution is more suitable for implementing a distributed land ledger for Sri Lanka.

1.3 Research Problem and Research Questions

The aim of this research is to provide a permissioned distributed ledger solution for the Sri Lankan land transaction scenario, subsequent to a systematic performance evaluation of the proposed solution.

Thus, through this research, it is intended to find answers to the following research questions.

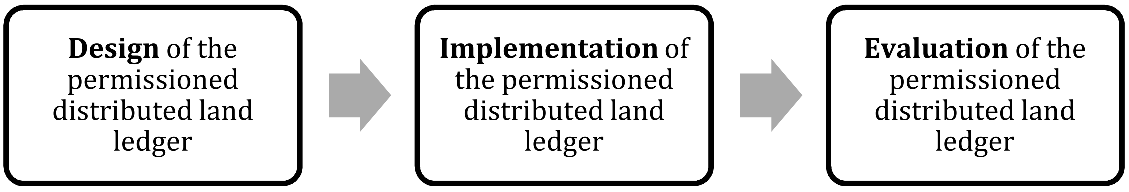
A) What are the capabilities and limitations of adapting an open source solution for implementing a distributed land ledger for Sri Lanka?

B) What is the performance difference between two proposed abstract models of the land ledger under different land transaction density conditions and failure conditions?

C) What are the future prospects and possibilities for implementing a large scale distributed land ledger model for Sri Lanka?

1.4 Methodology

The research methodology followed with the intention of achieving the research aim could be explained in a high-level as follows.


Design of the permissioned distributed land ledger
Implementation of the permissioned distributed land ledger
Evaluation of the permissioned distributed land ledger

As the first step of the research approach, the permissioned distributed land ledger is designed to suit the Sri Lankan land transaction scenario. The design of the distributed land ledger is provided based on features of a generic permissioned DLT platform. During the design phase, optimal land ledger content for the proposed solution is derived from the existing folio and transactions to be performed against the land ledger are devised. During this phase, the ledger solution is designed with the intention of getting away with the folio system and facilitate derivation of the pedigree/ folio tree from the distributed ledger.

As stated in the first research question, during the implementation phase of the research approach, the proposed design of the distributed land ledger would be implemented using an **open source** permissioned DLT platform.

During the last stage; evaluation, the implemented distributed land ledger solution would be evaluated for performance under different land transaction density conditions and failure conditions (fault tolerance).

The term **'Land transaction density'** with respect to a Regional Land Registry (RLR) could be defined as the frequency of land transactions submitted to the particular RLR. RLRs such as Colombo, Galle generally have a higher land transaction density than RLRs such as Hambantota, Tangalle. Considering the **validation policies** of Sri Lankan land transaction scenario and the heterogeneity of land transaction density across RLRs in Sri Lanka, two Abstract Models are designed, implemented and evaluated, with the intention of proposing the most suitable land ledger solution for Sri Lanka. Further, the two proposed Abstract Models are evaluated under different failure conditions of the DLT network, as stated in the second research question.

1.5 Outline of the Dissertation

The dissertation is structured as follows. Chapter two presents a review of the literature on blockchain based land registries in other countries and features of permissioned DLT platforms suitable for implementing the proposed solution. Chapter three presents the design of the distributed land ledger based on features of a generic permissioned DLT platform. Chapter four explains the implementation details of the distributed land ledger solution using Hyperledger Fabric. Chapter five provides details of evaluation performed on the implemented DLT solutions and interpretation of the results obtained. Chapter six provides a conclusion for the thesis with prospects for future work.

1.6 Delimitations of Scope

In this research, “Land transactions”, refer only to change of ownership right of a particular piece of land between two parties, i.e. mortgages etc. would not be considered as “Land transactions”. Proposed Abstract Models of land transactions are designed by only taking Sri Lankan land transaction scenario into consideration. Conversion of public lands to private lands through government land grants would not be handled through the provided solution. Fail-restart model of process failures is not considered during evaluation of the solutions.

1.7 High level architecture of a permissioned DLT solution for land transactions

This section provides an overview of the high level architecture of a general permissioned DLT solution for land transactions, followed by a brief introduction to the internal workings of a distributed land ledger solution.

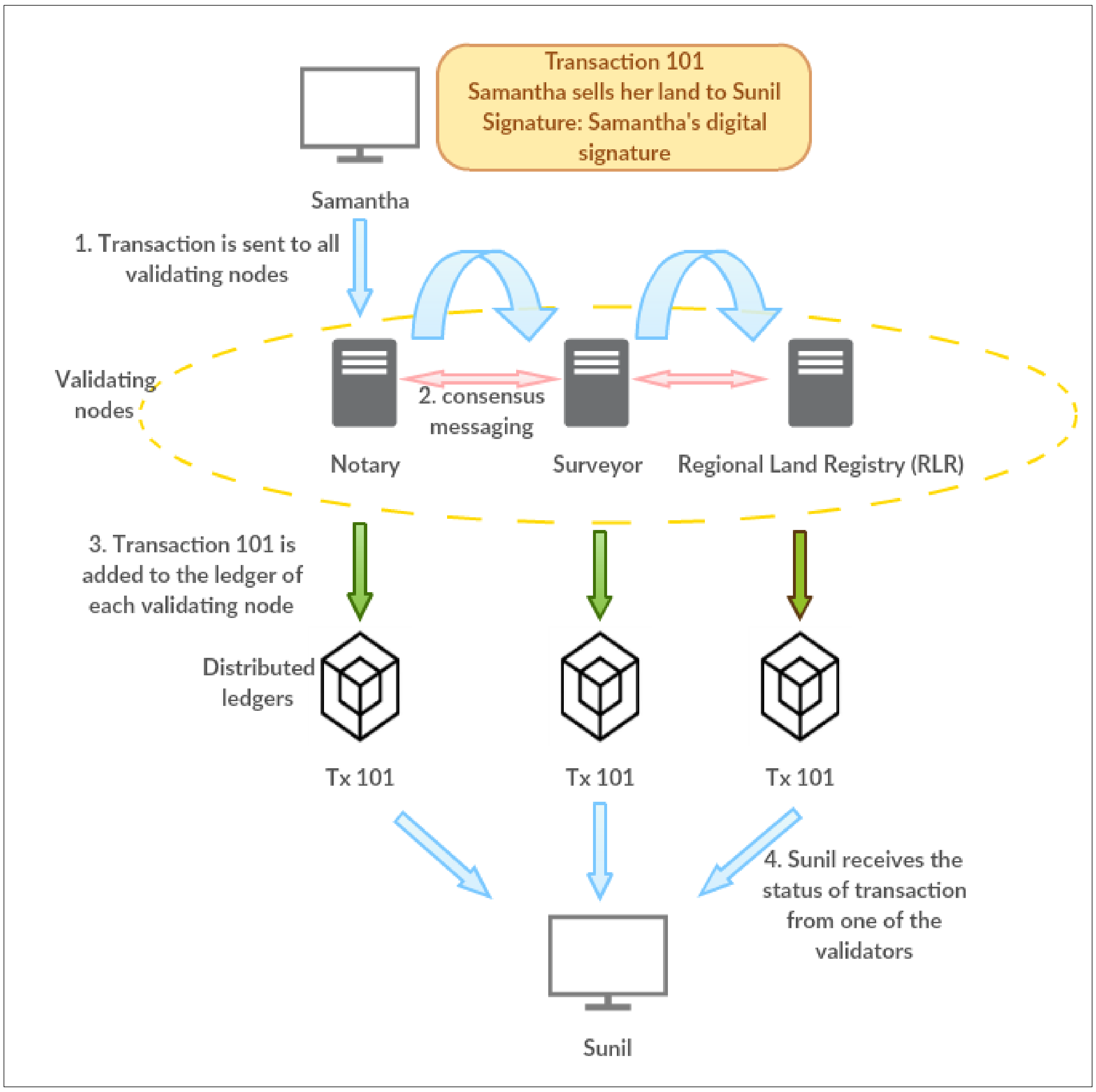

Transaction 101
Samantha sells her land to Sunil
Signature: Samantha's digital signature
Samantha
1. Transaction is sent to all validating nodes
Validating nodes
2. consensus messaging
Notary
Surveyor
Regional Land Registry (RLR)
3. Transaction 101 is added to the ledger of each validating node
Distributed ledgers
Tx 101
Tx 101
Tx 101
4. Sunil receives the status of transaction from one of the validators
Sunil

Figure 1.1: High level architecture of a permissioned DLT solution for land transactions.

Based on Figure 1.1, it could be seen that the seller will sign and submit a transaction proposal to the permissioned DLT network, indicating the sale of her land to buyer. Next, the submitted transaction will be sent to all validating nodes. Subsequent to consensus messaging among validating nodes, if the transaction is identified as valid, it will be added to the ledger of each validating node. Finally, buyer could receive the status of transaction from any of the validators. Notaries, Surveyors and Regional Land Registries (RLRs) who are involved in the validation of land transactions in the current manual system, could be identified as **validators** in the permissioned DLT network as well.

Chapter 2 - Literature Review

2.1 Introduction

In this chapter, a review of the features of blockchain based land registries of other countries is provided. Subsequent to justifying the importance of exploring the suitability of a distributed ledger solution for the Sri Lankan land transaction scenario, the choice of **permissioned** distributed ledgers is justified. Choice of Hyperledger as the permissioned DLT platform for implementation of the solution is presented next. However, it is important to note that the design of solution has been provided to suit a **generic** permissioned DLT platform. Finally, the choice of Hyperledger Fabric for implementation is justified.

2.2 Blockchain based land registries of other countries

As mentioned in Chapter 1, [2] states that, a badly kept, mismanaged and/or corrupt land registry could be successfully replaced by a blockchain based land registry. Nevertheless, it could be observed that, not only countries with unreliable land registries such as Honduras, Ghana, but countries with well-functioning land registries such as Georgia, Sweden, Estonia are also in the process of implementing and deploying blockchain based land registries [3, 4, 5, 6, 7, 8].

Table 2.1: Stakeholders of blockchain based land registries in other countries

Country	Government stakeholder	Blockchain solution provider	Other stakeholders
Georgia	National Agency of Public Registry (NAPR)	“BitFury” (a Bitcoin company)	Renowned Peruvian economist Hernando de Soto
Sweden	“Lantmäteriet”	“ChromaWay” (a blockchain technology company)	“KairosFuture” (an international research firm), “Telia” company
Estonia	Center of Registers and Information Systems (RIK)	Guardtime	-
Chicago’s Cook county	-	Velox.RE (a startup)	Volunteer collaboration of public and private stakeholders
Honduras	Honduran government	Factom	-
Ghana	-	“Bitland” (a nonprofit organization)	-

According to Table 2.1, it could be observed that companies specialized in blockchain technology have undertaken implementation of blockchain based land registries in those countries. In countries where government assistance is extended to the blockchain based land registry project, the blockchain solution is integrated to the digital land records system.

In all the countries listed in Table 2.1, **custom designed** blockchain solutions have been developed. In Georgia, Sweden and Estonia, blockchain solution providers have implemented the custom designed blockchain to suit the title registration process exercised in those countries. In contrast, developers of Cook county’s blockchain based land registry have implemented and deployed a “Blockchain deed protocol”.

However, blockchain based land registries of Sweden, Chicago's Cook county, Honduras and Ghana are **open-source** solutions.

Table 2.2: Type of blockchain technology used by other countries

Country	**Type of blockchain technology**
Georgia	Permissioned blockchain anchored to the Bitcoin blockchain.
Sweden	Permissioned DLT network where trusted parties validate transactions while public could view details in the blockchain using an SSN based ID solution.
Estonia	Public ledger
Chicago's Cook county	A colored coin (a bitcoin token) represents the land asset. Ownership change is recorded on a public ledger (the Bitcoin blockchain).
Honduras	Factom anchored to the Bitcoin blockchain.
Ghana	Blockchain solution based on Bitcoin blockchain technology.

Although there are two main types of distributed ledgers; unpermissioned and permissioned, countries listed in Table 2.2 have implemented their blockchain based land registry solutions <u>not only</u> based on those two types. When implementing their blockchain solutions, some of those countries have taken approaches such as **coloured coins** (an overlay network on Bitcoin blockchain) as well as **cross-chain exchange layer** approaches across public and private blockchains.

In blockchain based land registries of all countries listed above, only the **hash value** of data is embedded in the blockchain, while actual data which is generally large in size and confidential, is kept **off-chain** (in a traditional server). Through this move, content of a land transaction remains irrefutable. Nevertheless, the risk of losing the content remains the same as in a traditional land registry system.

Countries with already well-functioning land registries have moved from reliable manual systems to digitized systems and now towards blockchain based systems, with the intention of increasing efficiency and effectiveness of their land registration process. As a result of their effort, Georgia has been ranked among the top three countries in the world for ease of property registration [3]. As stated in [5], currently over 1 million immovables are recorded in the Estonian e-land register. In Ghana, the “bitland” land registry procedure is executed in addition to the Ghanian Land Commission procedure, with the aim of providing a service to its citizens, companies and farm unions [8]. Thus, through review of literature based on [3-8], it could be concluded that countries with already well-functioning land registries as well as countries with badly-kept, mismanaged and/or corrupt land registries have reaped benefits by implementing and deploying blockchain based land registries. Thus, the importance of exploring the suitability of a distributed ledger solution for the Sri Lankan land transaction scenario is justified.

2.3 Permissioned Distributed Ledger solution for the Sri Lankan land transaction scenario

Through this research, a permissioned distributed ledger solution for the Sri Lankan land transaction scenario has been provided. Choice of permissioned DLT was due to the advantages of permissioned DLT networks over unpermissioned DLT networks as stated in Chapter 1, as well as due to certain other important factors. According to [2], when selecting the type of DLT in order to implement a distributed land ledger solution, it is important for it to be in accordance with the current situation in the country with regards to the content of the land registers. Accordingly, by introducing a distributed land ledger solution, the land registry system of a country cannot be changed from a deed system to a title system or vice versa. Since, majority of divisional secretariat divisions in Sri Lanka follow the deed registration system [1], the solution provided through this research would preserve properties of the deed system. Through analyzing content of current folio system, three main types of validators per a land transaction could be recognized. They are the 1) Regional Land Registrar on behalf of Regional Land Registry, 2) Notary and 3) Surveyor. In the traditional manual system, all three parties need to endorse a transaction, in order for it to be successfully registered. Thus, duty of Regional Land Registrars, notaries and surveyors is analogous to the responsibility of validators in a permissioned DLT network, rather than to the responsibility of miners in an unpermissioned DLT network.

All three types of validators that have been identified are recognized by the Sri Lankan government (i.e. Registrar General's Department). Thus, only authorized validators are permitted to endorse a land transaction in the current manual system. This is analogous to the requirement of permissioned distributed ledgers, where participants of the system require legal identities in real world in order to validate transactions.

Distributed ledger technology is aimed at reducing costs and making the use of trusted 3rd parties such as notaries, surveyors etc. superfluous [2]. However, in the practical Sri Lankan context, it is not pragmatic to eliminate the involvement of trusted third parties. The solution that has been provided, requires each trusted validator to maintain a validating node in the permissioned DLT network. In this distributed land ledger system, scrutinizing of the transaction content will not take place by the validator, manually. Instead, it will be performed by the validating node based on the smart contracts infrastructure of the permissioned distributed land ledger solution. Since each validating node has an updated copy of the land ledger, a validator could always obtain the most up-to-date land registry details through querying.

Further, in an unpermissioned DLT network (such as in Estonia's e-land register), the miners who hold the ledger in their computers for the purpose of performing consensus, have to be provided with incentives [5]. But, since the distributed land ledger solution provided through this research, is a permissioned DLT solution, there is no requirement of providing incentives for the validators. Thus, the choice of permissioned DLT in implementing the distributed ledger solution for the Sri Lankan land transaction scenario, is justifiable.

Table 2.3: Comparison of features of several permissioned DLT platforms [based on references 9, 10 and official websites of permissioned DLT platforms]

Permissioned DLT platform	**Primary application**	**Underlying database structure**	**Cryptocurrency**	**Script language and Turing completeness**	**Open-source /Proprietary**	**Special remarks**
Hyperledger	Generic applications	Blockchain	No native cryptocurrency	Golang: Turing complete	Open-source	
Corda	Financial applications	Non-blockchain	No native cryptocurrency	Java: Turing complete	Open-source	
Ripple	Financial applications	Non-blockchain	Ripple (XRP)	LLVM supported language: Turing complete	Open-source	
Symbiont	Financial applications	Blockchain	Symbiont Coin	Domain specific language for Symbiont	Proprietary	
Tendermint	Generic applications	Blockchain	No native cryptocurrency	Any generic programming language	Open-source	Suffers from a livelock bug. Correctness of protocol is problematic
Kadena	Business applications	Blockchain	No native cryptocurrency	Pact	Proprietary	Protocol is not available for public review
MultiChain	Financial application and multi-currency exchanges	Blockchain	MultiChain	Smart Filters	Open-source	
HydraChain	Extension of Ethereum for creating permissioned DLT networks	Blockchain	ETH, ETC	Python	Proprietary	Correctness of platform is doubtful
Quorum	Financial applications	Blockchain	Quorum	Solidity	Proprietary	Consensus cannot be ensured realistically

2.4 Permissioned DLT platforms

According to research question 1, it is required to implement the distributed land ledger solution for Sri Lanka by adapting an open-source DLT platform. Thus, properties of several permissioned DLT platforms were considered based on [9] and [10], in order to decide on the most suitable DLT platform for implementation.

According to Table 2.3, it is evident that most of the available permissioned DLT platforms have been specifically developed to support financial applications. Some of the DLT platforms listed in the table lack clear explanation of protocol and formal review of properties, thus leaving a question about their correctness. DLT platforms such as Symbiont, Kadena, Quorum and HydraChain had to be eliminated from consideration since it is hard to perform research on proprietary DLT platforms.

Subsequent to comparing features of all available DLT platforms, **Hyperledger** was chosen for the implementation of the distributed land ledger for the Sri Lankan land transaction scenario [11, 12, 13]. Hyperledger is an **open-source**, permissioned DLT platform which facilitates implementation of generic applications and therefore, will never issue a cryptocurrency. Hyperledger project provides high degrees of confidentiality, scalability and security. It facilitates execution of non-deterministic smart contracts and modularity. Although Hyperledger has powered successful prototypes, Proof of Concepts and several production systems, across different industries and use cases (food-safety network [11], etc.), there is no published work on a distributed land registry solution implemented with Hyperledger for Sri Lanka or any other country in the world. Thus, this research would assess the capabilities and limitations of adapting a Hyperledger based solution for implementing a distributed land ledger for Sri Lanka. It is important to note that although Hyperledger was chosen as the permissioned DLT platform used to implement the distributed land ledger solution, the design of the DLT solution has been presented for a generic permissioned DLT platform.

2.4.1 Hyperledger Fabric

Another reason which influenced the choice of Hyperledger is the number of frameworks that Hyperledger provides. Hyperledger provides 5 variants namely, Fabric, Burrow, Indy, Iroha and Sawtooth.

Table 2.4: Brief description of the five Hyperledger frameworks [11]

Hyperledger Framework	**Brief description of the framework**
Fabric	Facilitates development of DLT solutions using special features such as modular architecture, smart contracts called “chaincodes” and channels.
Burrow	Permissionable smart contract machine developed partly to the specifications of Ethereum Virtual Machine.
Indy	Facilitates development of DLT solutions where only the true owner can store, change and revoke identity artifacts on a distributed ledger (self-sovereignty).
Iroha	Facilitates development of DLT solutions having an emphasis on mobile application development.
Sawtooth	Facilitates development of highly scalable DLT solutions with PoET (a consensus protocol similar to PoW), but without high power consumption.

From Table 2.4, based on [11], it could be observed that each Hyperledger framework provides unique features which should be taken into consideration when selecting the most appropriate framework for implementation.

Out of those five variants, Hyperledger Fabric is the most established framework of the Hyperledger project. Fabric is widely used across different industries and use cases. As stated in [14], Hyperledger Fabric could be identified as a **distributed Operating System** for permissioned blockchains. The **extensibility** feature of Fabric allows to run distributed applications consistently across all nodes in the permissioned DLT network. In the context of distributed ledgers, smart contracts function as a type of trusted distributed application. Although early DLT platforms required smart contracts to be written in domain-specific languages (which are prone to programming errors) or rely on cryptocurrencies, Fabric pioneered in producing chaincode (Fabric's smart contracts) using standard, **general purpose programming languages**.

Early DLT platforms followed Order-Execute architecture. During Ordering phase, the underlying consensus protocol orders all the transactions and propagates those to peers. Next, in the Execute phase, all peers execute every transaction sequentially (Sequential execution of transactions limit performance). DLT platforms which follow Order-Execute architecture require all transactions to be **deterministic**. Further, Order-Execute architecture violates confidentiality since every smart contract executes on every peer. In contrast, Hyperledger Fabric follows **Execute-Order-Validate** architecture. Since, transactions are executed by a subset of endorsing peers during Execute phase before the Ordering phase, **non-deterministic** smart contracts are also allowed to be executed. Hyperledger Fabric facilitates execution of transactions on a subset of peers thereby preserving confidentiality as opposed to DLT platforms following order-execute architecture.

Early DLT platforms provided **hard-coded consensus**. There, the trust model is determined by the underlying consensus protocol of the platform. Through the modularity feature of Hyperledger project, Fabric introduced **pluggable consensus** protocols. Thus, DLT solutions provided by Fabric could be tailored for different trust models.

Considering the improved features of Hyperledger Fabric over other permissioned DLT platforms it could be realized that Fabric is a suitable candidate for implementing a permissioned DLT network for the Sri Lankan land transaction scenario. According to [9], features of DLT platforms have to be considered when designing and deploying DLT applications. When considering the requirements of Sri Lankan land transaction scenario, and the features provided by Fabric such as peers, organizations, channel architecture, chaincodes, endorsement policy etc., it could be concluded that Hyperledger Fabric is suitable for implementing the permissioned distributed land ledger for Sri Lanka.

2.5 Summary

In this chapter, a review of the features of blockchain based land registries of other countries was provided. Subsequent to justifying the importance of exploring the suitability of a distributed land ledger solution for Sri Lanka, the choice of permissioned distributed ledgers was justified. Choice of Hyperledger as the permissioned DLT platform for implementation of the solution was presented next. However, it is important to note that the design of solution presented in next chapter, has been provided to suit a generic permissioned DLT platform. Finally, the choice of Hyperledger Fabric for implementation was justified.

Chapter 3 - Design

3.1 Introduction

In this chapter, the design of the DLT solution has been presented for a **generic permissioned DLT platform**. This chapter presents the design of optimal land ledger content followed by the design of transactions against the ledger. The design of two Abstract Models based on validation policies with the intention of addressing the land transaction density variation across RLRs in Sri Lanka has been presented. Finally, the importance of designing a fault tolerant distributed ledger solution has been stated.

3.2 Design of optimal land ledger content

As stated in Chapter 2, according to [2], when selecting the type of DLT technology in implementing a distributed land ledger solution, it is important for it to be in accordance with the current situation in the country with regards to the content of the land registers. The permissioned DLT solution provided through this research project, preserves properties of the deed system which is exercised in majority of divisional secretariat divisions of Sri Lanka.

Current folio system is a centralized system. Details included in a folio could be divided into 2 main sections. They are 1) Fixed details regarding a land and 2) Transaction details. The 2nd section which holds transaction details store one record per each land transaction. Appendix A includes a diagram of the structure of folio.

Section 1 which holds fixed details regarding a land include following details.

- Folio Number
- Location of land
- Boundaries
- Extent
- Name of land
- Plan Number and Date of Plan
- Name of surveyor
- Lot Number

Section 2 which holds transaction details, record following details per each land transaction.

- Deed Number and Date of Deed
- Name of Notary
- Registration stamp duty
- Grantors
- Grantees
- Remarks regarding transaction
- Signature of registrar and the date of signature

It is important to consider the features of a generic permissioned DLT platform when designing and deploying DLT applications. Accordingly, the ledger subsystem which is an integral part of a permissioned DLT platform (with blockchain as the underlying database structure) comprises of,

1. **World state (W)**: Stores the state of the ledger at a given point in time.
2. **Transaction log**: Stores all transactions which have contributed towards current world state in **blockchain B**.

As depicted in figure 3.1, the ledger L comprises of blockchain B and world state W. World state W could be derived from blockchain B.

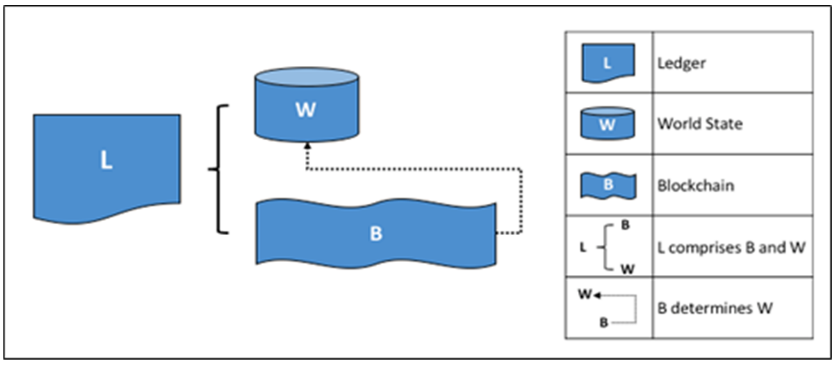


Figure 3.1: Relationship between ledger, world state and blockchain

Thus, it could be observed that world state W is similar to the conjunction of section 1 of folio (which holds fixed details regarding a land) and details of the latest transaction regarding the land in section 2. Further, blockchain B is analogous to past transaction records included in section 2 of the folio. Through this research we have got away with the folio system, by designing a committed ledger which has all the information, embedding folio details.

Although, this research has got away with the folio system, since all transaction details which have contributed towards current world state are available in the blockchain B, it is possible to obtain the **pedigree/ folio tree** which corresponds to a particular land at any given time.

When designing the committed ledger, **optimal ledger content** was extracted from the current folio. Optimal ledger content was finalized by removing redundant details from the current folio and adjusting attributes to suit a permissioned distributed land ledger solution.

Thus, the optimal ledger content included in the provided permissioned distributed land ledger is as follows.

- Land ID
- Location of land
- Boundaries of land (N, E, W, S)
- Extent
- Hash of plan
- Hash of deed
- Registration stamp duty
- Owner
- Remarks regarding transaction
- Parent Land ID

3.2.1 Design of transactions against distributed land ledger

At a given time, when the world state W of a land in the land ledger is queried, latest values corresponding to the above attributes would be returned as follows, i.e. one record per one land.

Land ID	Location	Boundaries	Extent	Hash of plan	Hash of deed	Registration stamp duty	Owner	Remarks regarding transaction	Parent Land ID

If a person who is going to buy the same land, inquires for the pedigree/ folio tree of that land via his notary, the transaction log would return all past transaction records. Each transaction record holds values for these attributes pertaining to the corresponding land transaction, i.e., there would be multiple records per one land, as follows.

Land ID	Location	Boundaries	Extent	Hash of plan	Hash of deed	Registration stamp duty	Owner	Remarks regarding transaction	Parent Land ID

A permissioned distributed land ledger solution for Sri Lanka would facilitate clients to request details regarding a piece of land when the LandID is provided. A query which facilitates retrieving details of all lands of the ledger would be useful for the land registrars at RLRs. In addition to querying, clients would be able to submit two types of transaction proposals to the land ledger. Clients would be able to request for a change of ownership of an existing piece of land, which could be termed as **changeLandOwner** transaction. Furthermore, a client would be able to request to split an existing land and register newly created lands with new owners, updated extents and boundaries. The latter transaction could be termed as **forkLand**.

Blockchain based land registries of all the countries reviewed in Chapter 2, embed only the hash value of data which is generally large in size and confidential, in the blockchain. Actual data is stored off-chain (in a traditional server). The permissioned DLT solution implemented through this research has also included only the hash values of plan and deed in the optimal content of the land ledger.

3.3 Design of two Abstract Models for the SL distributed land ledger

From the above details included in the folio, 3 types of validators could be identified. They are the 1) Regional Land Registrar on behalf of Regional Land Registry, 2) Notary and 3) Surveyor. All 3 types of validators are recognized by the Sri Lankan government (i.e. Registrar General's Department). A Regional Land Registry is identified by the district that it belongs to (One district may have one or more Regional Land Registries). A notary is identified by the Regional Land Registry that he/she is registered with. A surveyor is identified by the district. Official website of the Registrar General's Department (http://www.rgd.gov.lk) includes a list of RLRs identified by district and notaries registered with each RLR. Official website of the Land Survey Council of Sri Lanka (http://www.landsurveycouncil.org) provides a list of licensed surveyors identified by district. In the traditional manual system, all 3 parties need to endorse a transaction pertaining to a particular land, in order for it to be successfully registered.

Consider the land transaction scenario of a land in Sri Lanka explained shortly, in order to figure out the involvement of 3 types of validators when registering a transaction. Suppose a person from Colombo wants to buy a land in Galle which is located in the terrain of Galle Regional Land Registry (RLR). Assume the buyer hires a notary from Colombo to perform all the legal undertakings related to the purchase of land. Notary's responsibilities include certifying the purchase consideration with a written deed until forwarding the deed to the land registrar of Galle RLR for registration. Since the land is located in Galle RLR's territory, the record pertaining to the land is included in the Galle RLR's land ledger. Suppose the buyer hired a surveyor who is registered in the Hambantota district. The surveyor prepares a plan which is annexed to the deed (prepared by the notary) with an affidavit by the surveyor certifying that he has prepared the plan correctly and truthfully. The land registrar in Galle RLR would consider all details and endorsements provided by the notary and the surveyor and provide his endorsement, thus successfully completing registration of the land transaction. Since the notary has been registered with the Colombo RLR in this scenario, a copy of the deed has to be sent to the Colombo RLR for future reference. Thus, it could be observed how the endorsement of all 3 types of validators are required for the successful registration of a land transaction.

When the above scenario is considered, it could be observed that the extent of details accessible by each type of validator varies. Accordingly, each RLR stores details of lands in its territory, including deeds and plans of those lands. Notaries could access details pertaining to lands belonging to a particular RLR through formal inquiry from the relevant RLR. Further, a notary possesses deeds of lands certified by him. Surveyors too could access details of lands belonging to a particular RLR through formal inquiry from the relevant RLR. In addition, a surveyor possesses plans of lands prepared by him. RLRs have copies of deeds pertaining to lands (these lands could belong to other RLRs) certified by notaries registered with the particular RLR.

In the implemented solution, all three types of validators would have access to the optimal ledger content. In addition, RLRs would have access to deeds and annexed plans of lands in their terrain as well as those of lands certified by notaries registered with them. Notaries would have access to deeds certified by them and surveyors would have access to plans prepared by them. This approach has preserved the extent of details accessible by validators in the present traditional land transaction scenario.

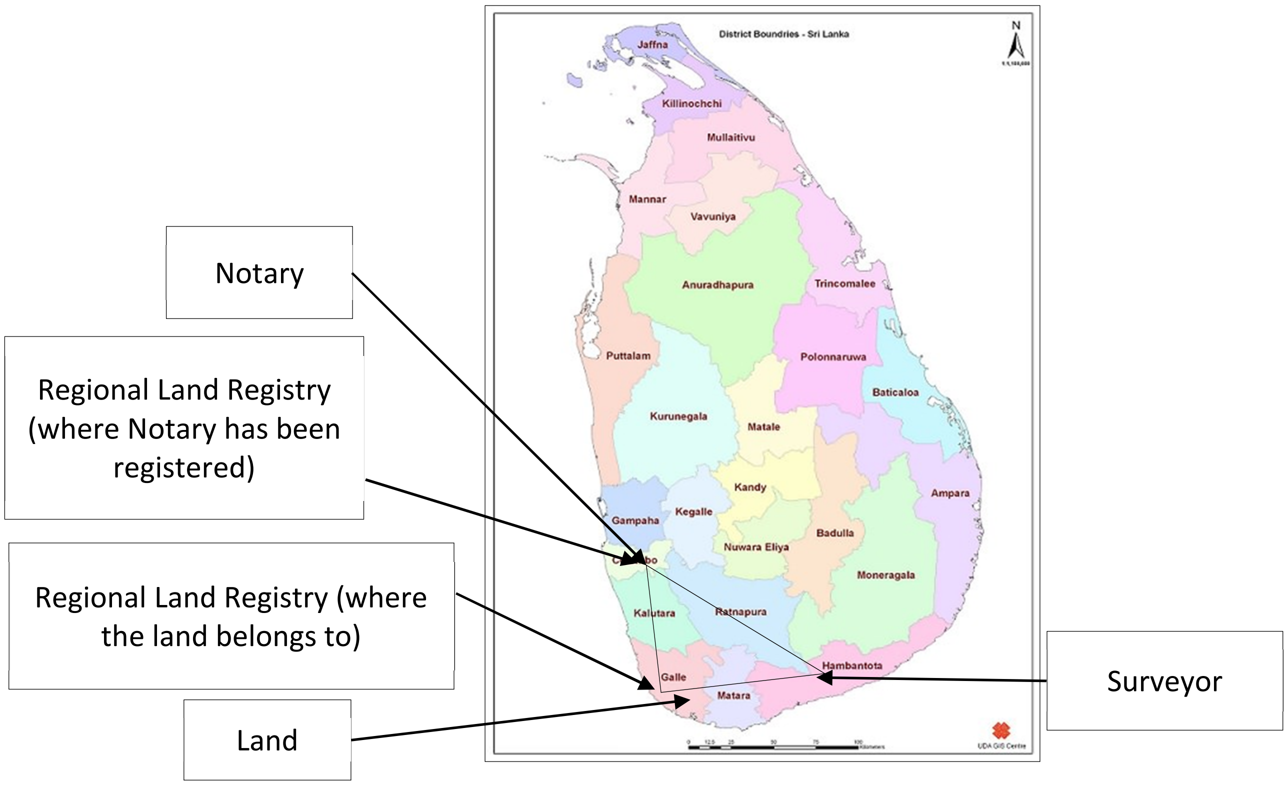


Figure 3.2: Validators involved in endorsing a transaction of a land in Galle.

As required by the second research question of this research project, two abstract models have been provided as permissioned DLT solutions for the Sri Lankan land transaction scenario. The first abstract model; Abstract Model 1 (AM1) was designed such that it closely maps the current manual system. When the previously explained traditional scenario is implemented using a permissioned DLT platform, the same 3 types of validators could be adapted for the validation of a land transaction. In addition to 3 validators; 1) Regional Land Registry where the land belongs to, 2) Notary and 3) Surveyor, the Regional Land Registry where the notary has been registered could also act as a validator in Abstract Model 1. This is because, the RLR where the Notary has been registered with, also possesses transaction details of the land. Thus, **four validators** per a given land transaction could be identified. Since, validators from at most 3 districts are involved in the validation process, a **three district model** (indicated by the triangle in Figure 3.2 for validation could be identified. Thus, it could be concluded that implementation of a three district model is acceptable for the purpose of evaluation.

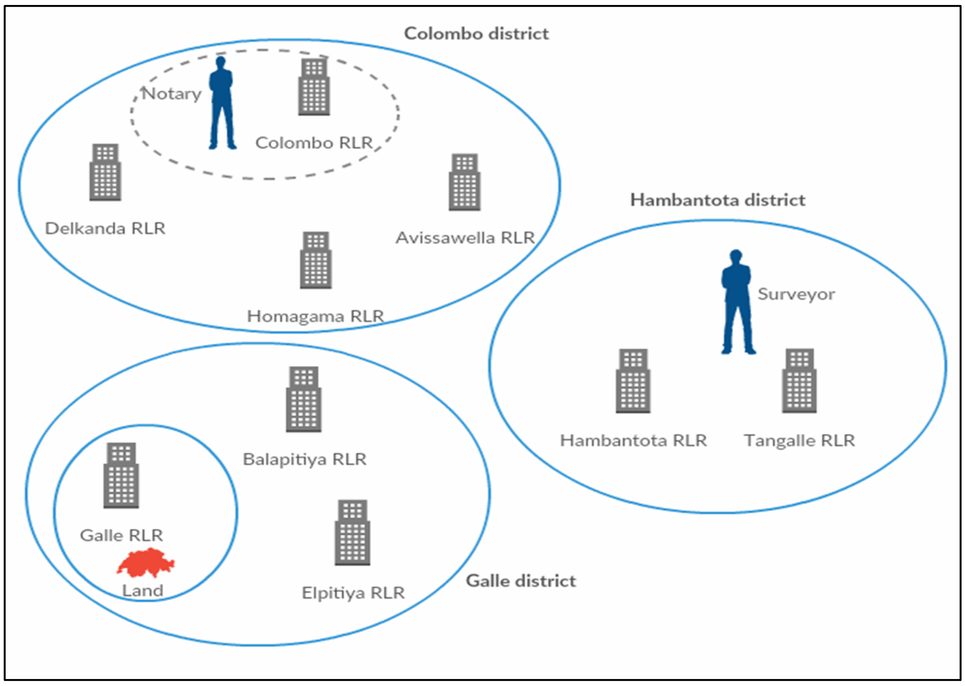


Figure 3.3: Three district model for validation of a land transaction

In the present traditional system, each Regional Land Registry (RLR) maintains a ledger containing only details of lands belonging to that RLR. When this situation is adapted in Abstract Model 1, each RLR holds an independent land ledger of its own lands. Therefore, each RLR would maintain its own land ledger (indicated by Li; i=1..9) as shown in figure 3.4.

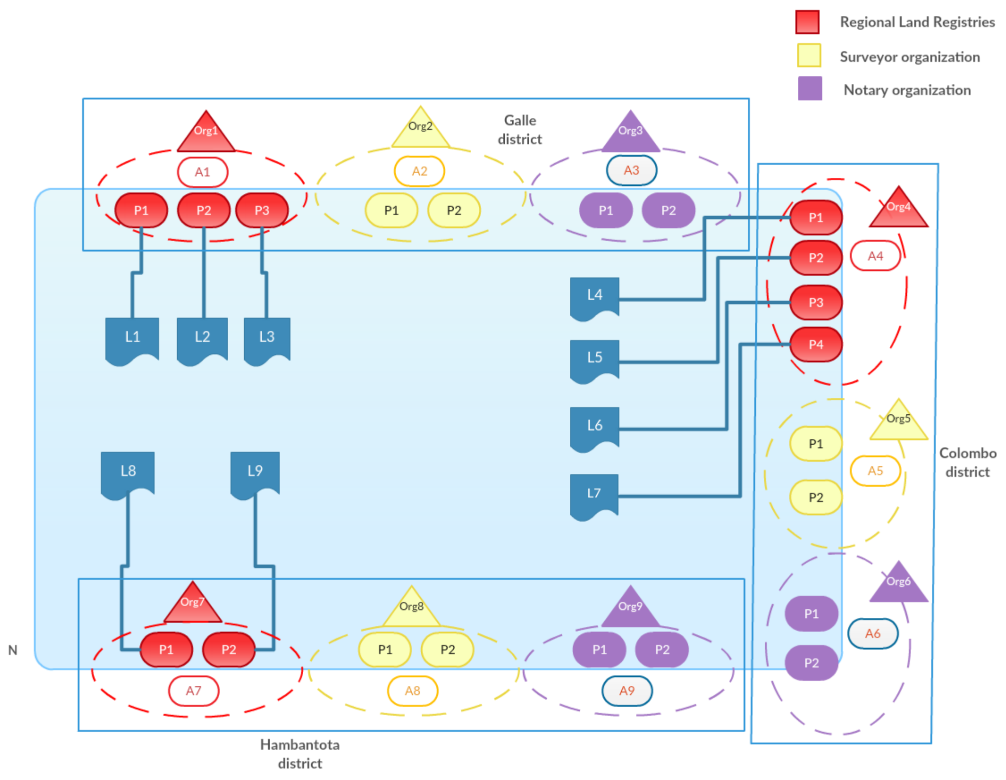


Figure 3.4: Each RLR maintains a ledger containing only details of lands belonging to itself

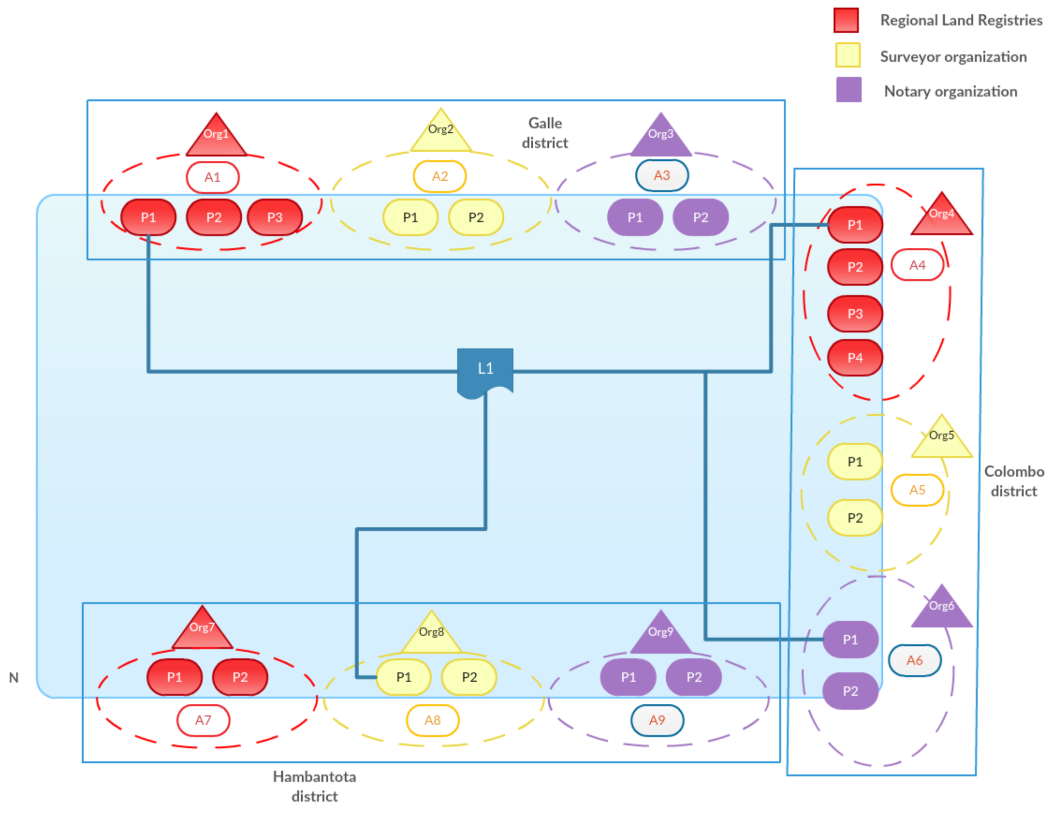


Figure 3.5: Four validators are validating a transaction of a land belonging to Galle RLR

In the real Sri Lankan scenario, each RLR has to endorse all transactions regarding lands in its terrain, submitted for registration. Figure 3.5 depicts how four validators are involved in validating a transaction pertaining to a land belonging to Galle RLR. This requirement, emerges an issue with regards to workload distribution among RLRs. Consider RLRs such as Colombo, Galle which have a high land transaction density (i.e. those RLRs may have a high frequency of land transactions submitted for registration). Validating nodes representing those RLRs may have a high overhead on performing validation of submitted transactions. At the same time, RLRs such as Hambantota, Tangalle would have a low land transaction density (i.e. those RLRs may have a low frequency of land transactions submitted for registration). Thus, when a set of transactions pertaining to lands situated island wide, are submitted to the permissioned DLT network **concurrently**, RLRs with low land transaction density would complete validation earlier than RLRs with high land transaction density. This would reduce the overall transactional throughput of the provided solution. **Transactional throughput** is the rate at which transactions are committed to the distributed land ledger.

Abstract Model 2 (AM2) which is more suitable for a distributed system is proposed as a remedy to the above mentioned drawback of Abstract Model 1. Abstract Model 2 proposes a single land ledger for the entire country. That single land ledger would hold details of all lands island wide. Now, since all RLRs have access to the same land ledger, RLRs having low land transaction densities would be able to validate transactions submitted to RLRs with high land transaction densities, thus **sharing the workload**. Figure 3.6 demonstrates how all RLRs access a single land ledger containing details of all lands across the island. Figure 3.7 depicts how validators are involved in validating a transaction pertaining to a land belonging to Galle RLR. There, since other RLRs could also access the land ledger, it is possible for any other RLR to perform validation on behalf of Galle RLR. It was hypothesized that through this design approach of Abstract Model 2, the overall transactional throughput would increase.

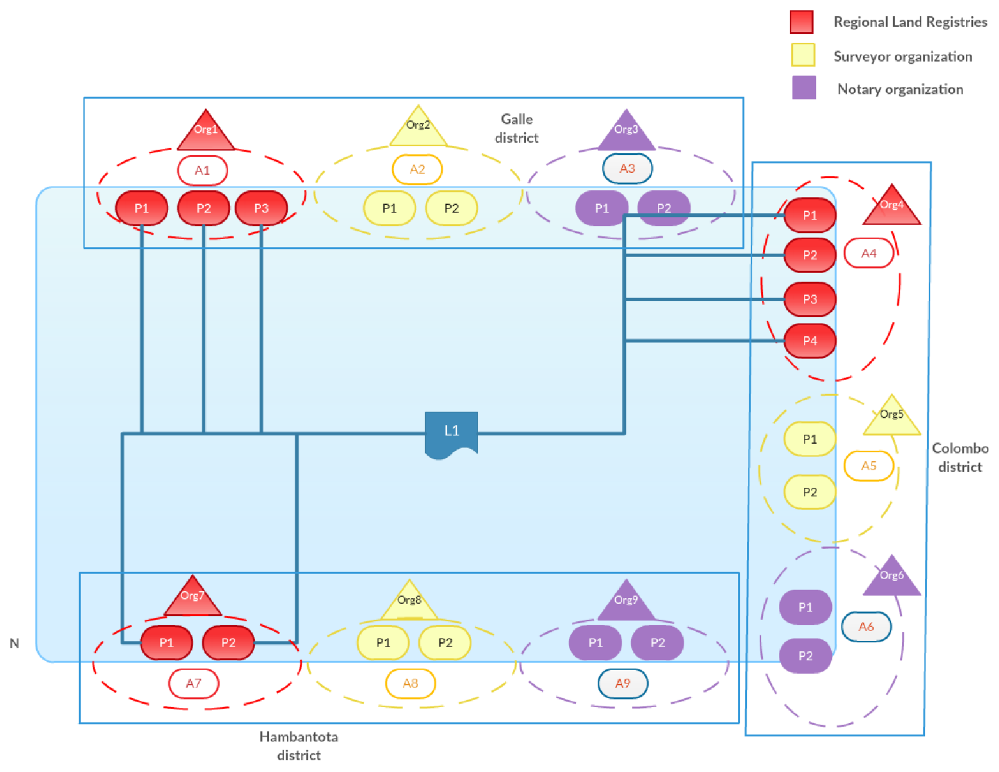


Figure 3.6: All RLR access a single ledger containing details of all lands situated island wide

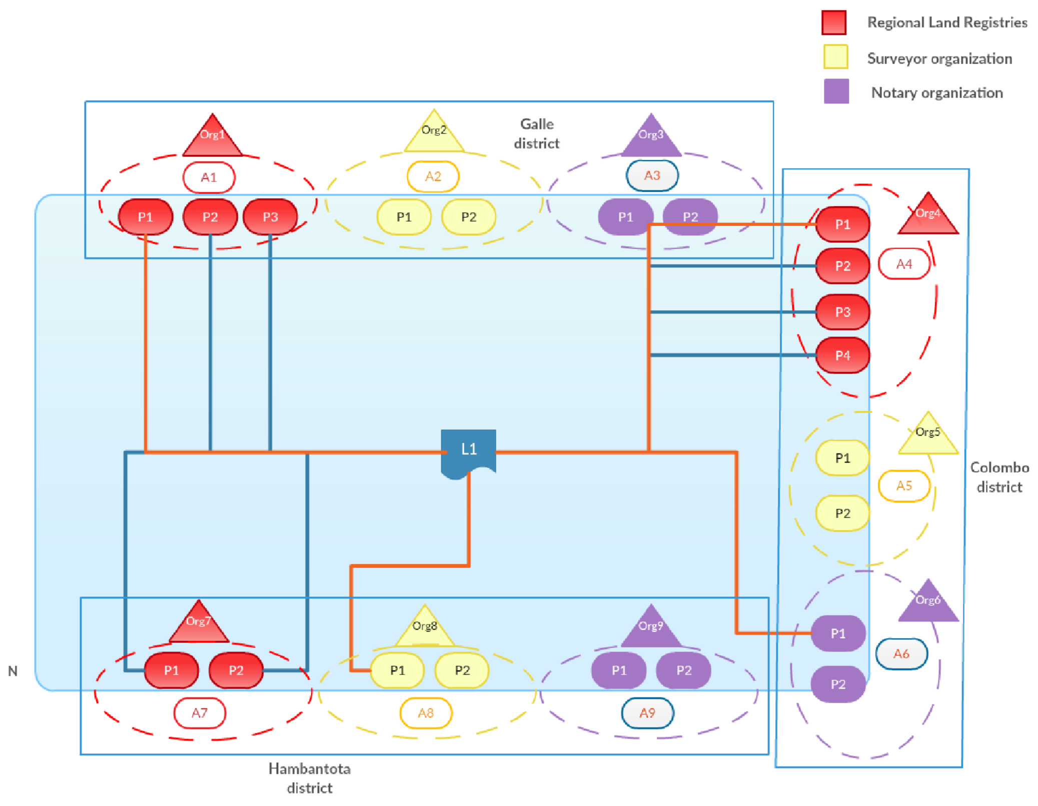


Figure 3.7: Validators are validating a transaction of a land belonging to Galle RLR (AM2)

Therefore, as expected by the second research question, the performance difference between two Abstract models under different land transaction density conditions across RLRs, was evaluated. Thus, this section of the chapter presented the design of 2 abstract models implemented and evaluated through this research project. Table 3.1 provides a summary of the 2 Abstract Models.

Table 3.1: Summary of the two Abstract Models

Abstract Model 1	Abstract Model 2
Each RLR maintains a land ledger containing details of lands belonging only to itself.	All RLRs have access to a single land ledger containing details of all lands situated in the island.
Validation is performed by only those who have originally involved in transactions regarding the land.	Validation is performed by replicated copy holders.
Overall transactional throughput is low due to high overhead at RLRs having higher land transaction density.	Overall transactional throughput is higher than that of Abstract Model 1, by sharing the workload among validating nodes.

As mentioned in Table 3.1, with the introduction of Abstract Model 2, all RLRs in the country gain access to a single land ledger containing details of all lands situated island wide. This is in contrast to Abstract Model 1 (which closely resembles the real Sri Lankan land transaction scenario) where each RLR maintains a land ledger containing details of lands belonging only to itself. In order to prevent possible problems arising due to the deviation from the real Sri Lankan scenario, the implemented solution requires clients to submit **copies of deed and plan** when a transaction is submitted to the DLT network. The hash values of deed and plan are generated by the system before sending the transaction to the validating nodes. The validators would perform validation based on the hash values of deed and plan. As a result of this approach, validators are prohibited from submitting **forged transactions**.

3.4 Design of a fault tolerant distributed land ledger solution

A **consensus protocol** is responsible of determining the order in which entries are appended to the distributed ledger. As stated in Chapter 1, security is higher in permissioned DLT networks over unpermissioned DLT networks. However, individual nodes in a DLT network could **crash** or behave **maliciously** (if computers of validators are taken over by a hacking entity). In environments where network connectivity is uncertain, validating nodes could crash. This could lead to the validating nodes acting against the common goal of reaching consensus. A fault tolerant consensus protocol must be established in the DLT network in order to detect and withstand such process failures.

Distributed Systems theory refers to three types of process failures, namely, 1) crash (fail-stop), 2) crash & reboot and 3) Byzantine crashes. In the fail-stop model, processes can fail by stopping, i.e. a faulty process eventually stops executing the algorithm permanently, whereas in the fail-restart model, a process can resume execution after crashing. Byzantine nodes could be identified as malicious validating nodes. For example, a virus infected validating node may inject blocks containing false or unauthorized transactions into the DLT network.

As stated in [9], the consensus approach which should be chosen when designing a DLT network, should depend on the nature of environment where the DLT solution would be deployed. Accordingly, if the DLT solution is deployed in a **trustworthy environment**, **Crash Fault Tolerance** consensus approach is sufficient. But, if the same solution is to be deployed in a **multi-party use case**, **Byzantine Fault Tolerance** consensus approach should be chosen. This, justifies the choice of Hyperledger which provides pluggable consensus, unlike other permissioned DLT platforms which provide hard-coded consensus. Thus, through plugging the correct fault-tolerant consensus protocol, higher transactional throughput is attainable.

Therefore, as expected by the second research question, the performance difference between two Abstract Models under different failure conditions, was evaluated.

3.5 Summary

In this chapter, the design of the DLT solution for a **generic permissioned DLT platform** was presented. This chapter presented the design of optimal land ledger content followed by the design of transactions against the ledger. Next, the design of two Abstract Models based on validation policies with the intention of addressing the land transaction density variation across RLRs in Sri Lanka was presented. Finally, the importance of designing a fault tolerant distributed ledger solution was presented.

Chapter 4 - Implementation

4.1 Introduction

This chapter provides the most important and relevant, high level implementation details of the distributed land ledger solution, provided using **Hyperledger Fabric version 1.2**. Readers are requested to refer Appendix B for further code listings. Section 4.2 presents implementation details of land ledger and transactions in accordance to the design presented in Chapter 3. Subsequent to a statement on the first research question, implementation details of the two abstract models has been presented. Advanced implementation of production scale DLT networks and facilitation of fault tolerance has been explained finally.

4.2 Implementation of optimal land ledger content and transactions

As justified in Chapter 2, Hyperledger Fabric was used for the implementation of the distributed land ledger solution, whose design was presented in Chapter 3. The ledger subsystem of Hyperledger Fabric is similar to the ledger subsystem of a generic permissioned DLT platform (with blockchain as the underlying database structure) as explained in Chapter 3. Thus, the ledger sub system of Hyperledger Fabric comprises of World state (W) and Transaction log stored in a blockchain (B).

In Chapter 3, optimal ledger content to be included in the distributed land ledger solution was extracted from the current folio. Thus, the optimal content included in the implemented distributed land ledger for Sri Lanka is as follows.

- Land ID
- Location of land
- Boundaries of land
- Extent
- Hash of plan
- Hash of deed
- Registration stamp duty
- Owner
- Remarks regarding transaction
- Parent Land ID

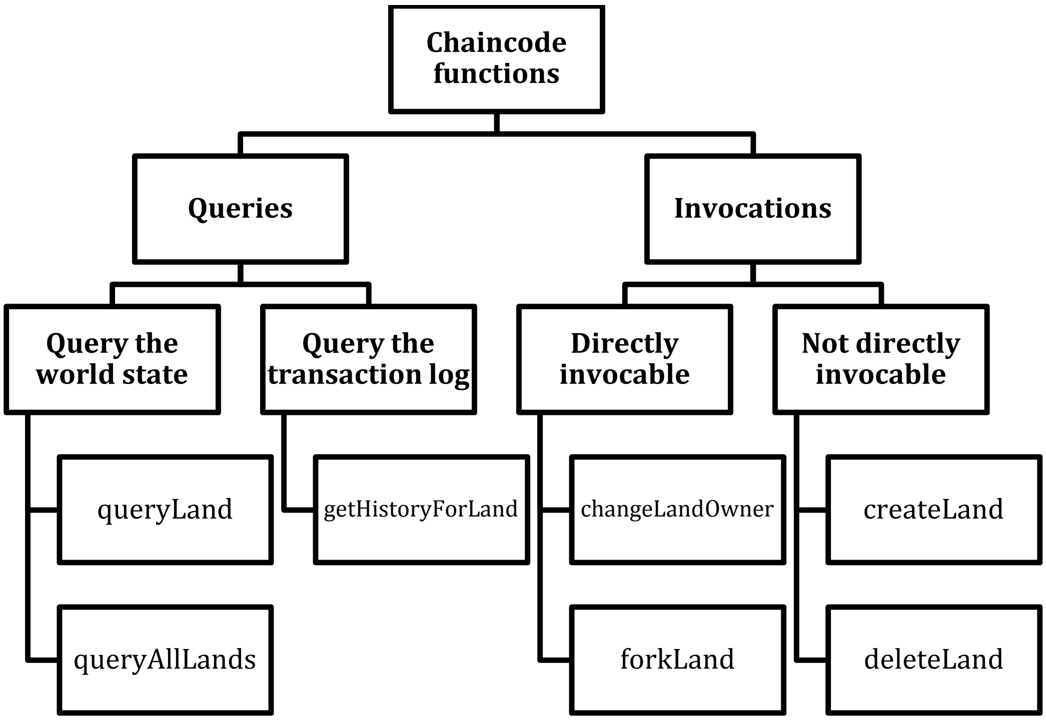
In Hyperledger Fabric, assets are represented as a collection of key-value pairs. During implementation, land assets were modeled as JSON in chaincode. **Chaincode** in Hyperledger is analogous to smart contracts in generic distributed ledger applications. Hyperledger's chaincode typically written in go or NodeJS, is used to define assets and contain the rules for modifying the assets. In this research, chaincode was written in go language. Thus a land asset (declared as a JSON structure) is identified by the LandID which is the **key** and the remaining attributes listed in optimal ledger content comprise the **values**.

Details of all lands which everyone can agree on, is included in the initLedger() function of the chaincode. In Hyperledger each DLT node is represented by a Docker container. Subsequent to **installing** chaincode on validating nodes (identified as endorsing peers), the chaincode has to be **instantiated**. During instantiation, a separate Docker container for each peer's chaincode is started and the initLedger() function of the chaincode is invoked. Invocation of the initLedger() function leads to initializing the key value pairs associated with the chaincode. This could be identified as the **genesis block** which marks the beginning of the 'history of transactions'.

During instantiation, in addition to invoking the initLedger() function, the **endorsement policy** is also passed as an argument. Endorsement policy specifies which validators or how many of them need to endorse a transaction proposal, based on rules for modifying assets in chaincode.

Hyperledger Fabric provides two types of state databases; LevelDB and CouchDB, out of which CouchDB was used during implementation. Since data pertaining to land assets were modeled as JSON in chaincode and CouchDB was used as the state database, complex rich queries could be implemented to query against the data values in chaincode containers, using the CouchDB JSON query language. Implemented chaincode, includes 2 queries namely; **queryLand** and **queryAllLands**. queryLand checks whether the land by the requested LandID exists and if so, returns the latest values for the attributes corresponding to the LandID. queryAllLands would return details of all lands in the land registry. Application clients can perform read-only queries. However, responses of those queries are not submitted as transactions to the ordering service. Therefore, history of queries is not recorded in the transaction log.

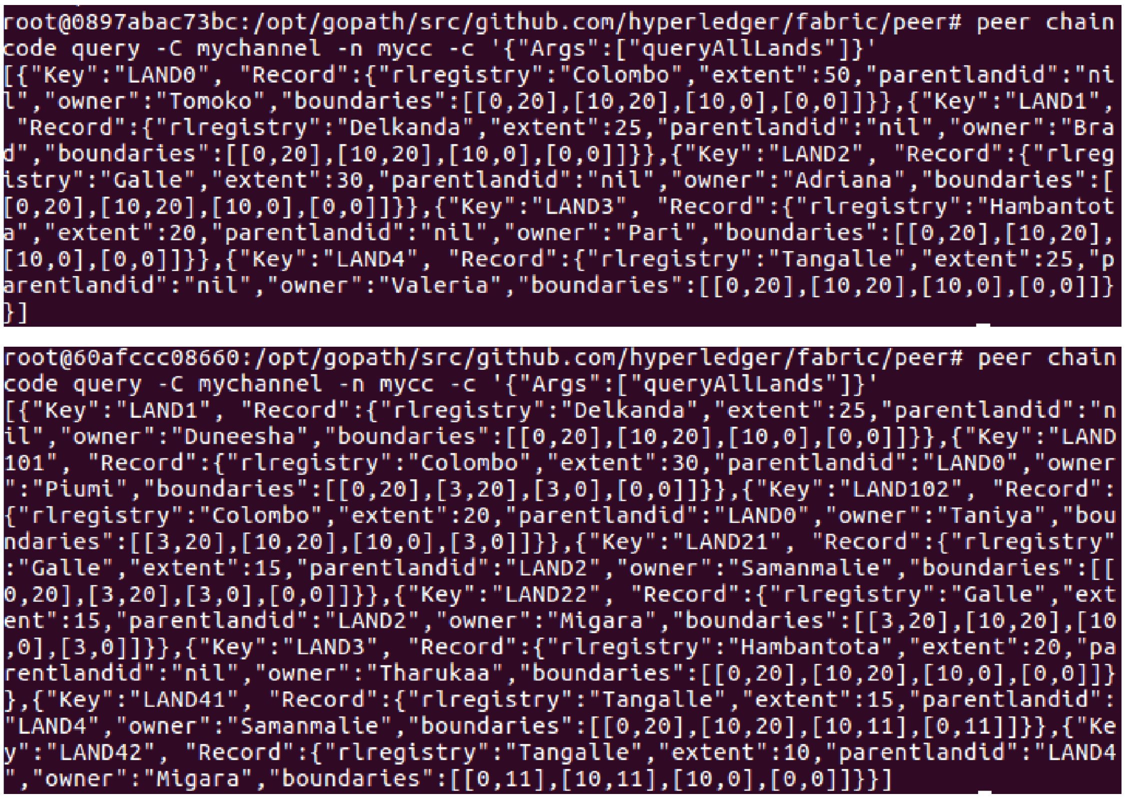
In addition to reading key-value pairs in the chaincode container, it is possible to alter values corresponding to keys through invoke functions. Implemented chaincode, includes 4 invocation functions. They are **changeLandOwner, forkLand, createLand and deleteLand**. Out of those four invocation functions, changeLandOwner and forkLand could be invoked directly by the application client, while createLand and deleteLand cannot be directly invoked.



In a changeLandOwner transaction, the client requests to change the owner of the land corresponding to the LandID mentioned in the transaction proposal. The client is requested to submit copies of deed and plan along with the transaction proposal. If the land identified by the LandID exists, and the hash values of deed and plan confirm with the existing values in the ledger, the value of the owner attribute would be changed from seller's name to buyer's name. changeLandOwner is a transaction which is smaller than forkLand transaction explained next.

In a forkLand transaction, the client requests to split a land into two or more lands and register those lands as new lands with new owners. Thus, the implemented forkLand transaction invokes deleteLand and createLand transactions. deleteLand transaction deletes the original land from ledger, if the land identified by the landID exists. createLand transaction will be invoked n times to create n number of new lands as requested by a valid transaction proposal. During createLand transaction, the LandID of the original land (which was deleted) is assigned to the ParentLandID of newly created lands and the values of Owner & Extent attributes of each new land are updated. **Before invoking deleteLand and createLand transactions**, forkLand transaction checks whether the sum of the Extents of the new lands is consistent with the Extent of original land and it also checks whether boundaries of newly created lands overlap with each other. It is important to note that deleteLand and createLand transactions are not directly invocable by client applications. Thus, it could be observed that forkLand is a transaction which is larger in size than changeLandOwner transaction.

Figure 4.1 demonstrates how, the implemented solution ensures the consistency of the land ledger, before and after executing a set of transactions. Readers who are interested on **validation** aspects of implemented transactions are advised to refer Appendix A.

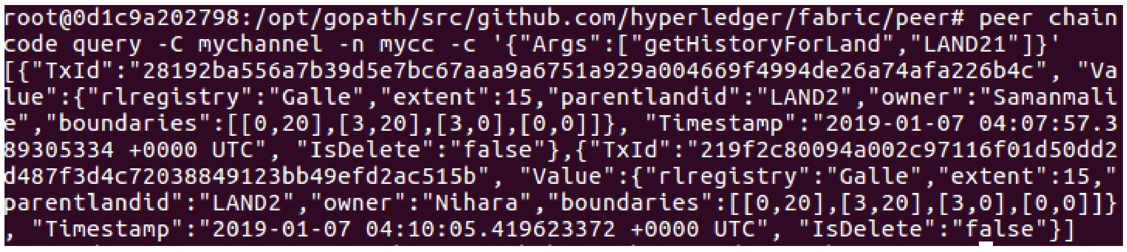


```
root@0897abac73bc:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chain
code query -C mychannel -n mycc -c '{"Args":["queryAllLands"]}'
[{"Key":"LAND0", "Record":{"rlregistry":"Colombo","extent":50,"parentlandid":"ni
l","owner":"Tomoko","boundaries":[[0,20],[10,20],[10,0],[0,0]]}},{"Key":"LAND1",
 "Record":{"rlregistry":"Delkanda","extent":25,"parentlandid":"nil","owner":"Bra
d","boundaries":[[0,20],[10,20],[10,0],[0,0]]}},{"Key":"LAND2", "Record":{"rlreg
istry":"Galle","extent":30,"parentlandid":"nil","owner":"Adriana","boundaries":[
[0,20],[10,20],[10,0],[0,0]]}},{"Key":"LAND3", "Record":{"rlregistry":"Hambantot
a","extent":20,"parentlandid":"nil","owner":"Pari","boundaries":[[0,20],[10,20],
[10,0],[0,0]]}},{"Key":"LAND4", "Record":{"rlregistry":"Tangalle","extent":25,"p
arentlandid":"nil","owner":"Valeria","boundaries":[[0,20],[10,20],[10,0],[0,0]]}
}]
```

```
root@60afccc08660:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chain
code query -C mychannel -n mycc -c '{"Args":["queryAllLands"]}'
[{"Key":"LAND1", "Record":{"rlregistry":"Delkanda","extent":25,"parentlandid":"n
il","owner":"Duneesha","boundaries":[[0,20],[10,20],[10,0],[0,0]]}},{"Key":"LAND
101", "Record":{"rlregistry":"Colombo","extent":30,"parentlandid":"LAND0","owner
":"Piumi","boundaries":[[0,20],[3,20],[3,0],[0,0]]}},{"Key":"LAND102", "Record":
{"rlregistry":"Colombo","extent":20,"parentlandid":"LAND0","owner":"Taniya","bou
ndaries":[[3,20],[10,20],[10,0],[3,0]]}},{"Key":"LAND21", "Record":{"rlregistry"
:"Galle","extent":15,"parentlandid":"LAND2","owner":"Samanmalie","boundaries":[[
0,20],[3,20],[3,0],[0,0]]}},{"Key":"LAND22", "Record":{"rlregistry":"Galle","ext
ent":15,"parentlandid":"LAND2","owner":"Migara","boundaries":[[3,20],[10,20],[10
,0],[3,0]]}},{"Key":"LAND3", "Record":{"rlregistry":"Hambantota","extent":20,"pa
rentlandid":"nil","owner":"Tharukaa","boundaries":[[0,20],[10,20],[10,0],[0,0]]}
},{"Key":"LAND41", "Record":{"rlregistry":"Tangalle","extent":15,"parentlandid":
"LAND4","owner":"Samanmalie","boundaries":[[0,20],[10,20],[10,11],[0,11]]}},{"Ke
y":"LAND42", "Record":{"rlregistry":"Tangalle","extent":10,"parentlandid":"LAND4
","owner":"Migara","boundaries":[[0,11],[10,11],[10,0],[0,0]]}}]
```

Figure 4.1: Consistency of the land ledger, before and after executing a set of transactions.

Although this implementation based on Hyperledger has got away with the folio system, since all transaction details which have contributed towards current world state are available in the transaction log (blockchain B), it is possible to obtain the pedigree/folio tree which corresponds to a land identified by the LandID, at any given time. The chaincode function getHistoryForLand could be invoked by the application client to obtain the pedigree/folio tree corresponding to the requested land. Hyperledger Fabric facilitates querying historical data (concept of **data provenance**), through the chaincode API function GetHistoryForKey which will return the history of values for a key. GetHistoryForLand chaincode function was implemented based on GetHistoryForKey chaincode API function.



```
root@0d1c9a202798:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chain
code query -C mychannel -n mycc -c '{"Args":["getHistoryForLand","LAND21"]}'
[{"TxId":"28192ba556a7b39d5e7bc67aaa9a6751a929a004669f4994de26a74afa226b4c", "Va
lue":{"rlregistry":"Galle","extent":15,"parentlandid":"LAND2","owner":"Samanmali
e","boundaries":[[0,20],[3,20],[3,0],[0,0]]}, "Timestamp":"2019-01-07 04:07:57.3
89305334 +0000 UTC", "IsDelete":"false"},{"TxId":"219f2c80094a002c97116f01d50dd2
d487f3d4c72038849123bb49efd2ac515b", "Value":{"rlregistry":"Galle","extent":15,"
parentlandid":"LAND2","owner":"Nihara","boundaries":[[0,20],[3,20],[3,0],[0,0]]}
, "Timestamp":"2019-01-07 04:10:05.419623372 +0000 UTC", "IsDelete":"false"}]
```

Figure 4.2: Derivation of pedigree/folio tree pertaining to a land

4.2.1 Statement on Research Question 1

Through the implementation of the distributed land ledger for Sri Lanka, using Hyperledger Fabric, this research has been able to provide implementations for the queries (queryLand and queryAllLands) and transaction invocations (changeLandOwner and forkLand) as required by the design in Chapter 3. Most importantly, we have been able to get away with the folio system while facilitating the ability to obtain the pedigree/folio tree for a land through the getHistoryForLand chaincode function. Since, it is obligatory for the client to submit copies of deed and plan along with the transaction invocations, submission of forged transactions have been prevented. Thus, at this point it could be concluded that the capabilities of adapting Hyperledger Fabric as an open source solution for implementing a distributed land ledger for Sri Lanka is at a high level.

However, a single limitation when implementing the distributed land ledger solution was identified. The absence of an existing algorithm for checking overlapping boundaries for non-rectangular shaped lands was identified as a limitation.

Therefore, the implemented solution performs boundary checking for rectangular shaped lands only (The algorithm implemented for boundary checking is included in Appendix A). Above identified limitation is **excluded** as a limitation of adapting Hyperledger for implementing a distributed land ledger for Sri Lanka, for the reason that, it is possible to improve Hyperledger's chaincode, once an algorithm for boundary checking of non-rectangular shaped lands is available.

4.3 Implementation of the two Abstract Models

As mentioned in Chapter 3, through this research project the design of two Abstract Models for the distributed land ledger have been proposed. They are Abstract Model 1 (AM1) and Abstract Model2 (AM2). Table 1 provides a summary of the 2 Abstract Models.

Table 4.1: Summary of the two Abstract Models

Abstract Model 1	Abstract Model 2
Closely maps the current manual Sri Lankan land transaction scenario.	More suitable for a distributed setting of the Sri Lankan scenario, with regards to land transaction density variation across RLRs.
Each RLR maintains a land ledger containing details of lands belonging only to itself.	All RLRs have access to a single land ledger containing details of all lands situated in the island.
Validation is performed by only those who have originally involved in transactions regarding the land.	Validation is performed by replicated copy holders.
Overall transactional throughput is low due to high overhead at RLRs having higher land transaction density.	Overall transactional throughput is higher than that of Abstract Model 1, by sharing the workload among validating nodes.

Individual validators of the permissioned DLT networks corresponding to both Abstract Models were implemented as **peers** in Hyperledger Fabric. Hyperledger Fabric creates a Docker container for each peer in the DLT network. Peers belonging to one trust domain is identified as members of a single **organization**. Peers within the same organization trust each other, but do not trust peers belonging to other trust organizations. As required by the design of two Abstract Models, three types of organizations (RLR organization, Notary organization, Surveyor organization) were implemented. As it could be seen in Figure 4.3, the three district model that was implemented has all three organizations per each district.

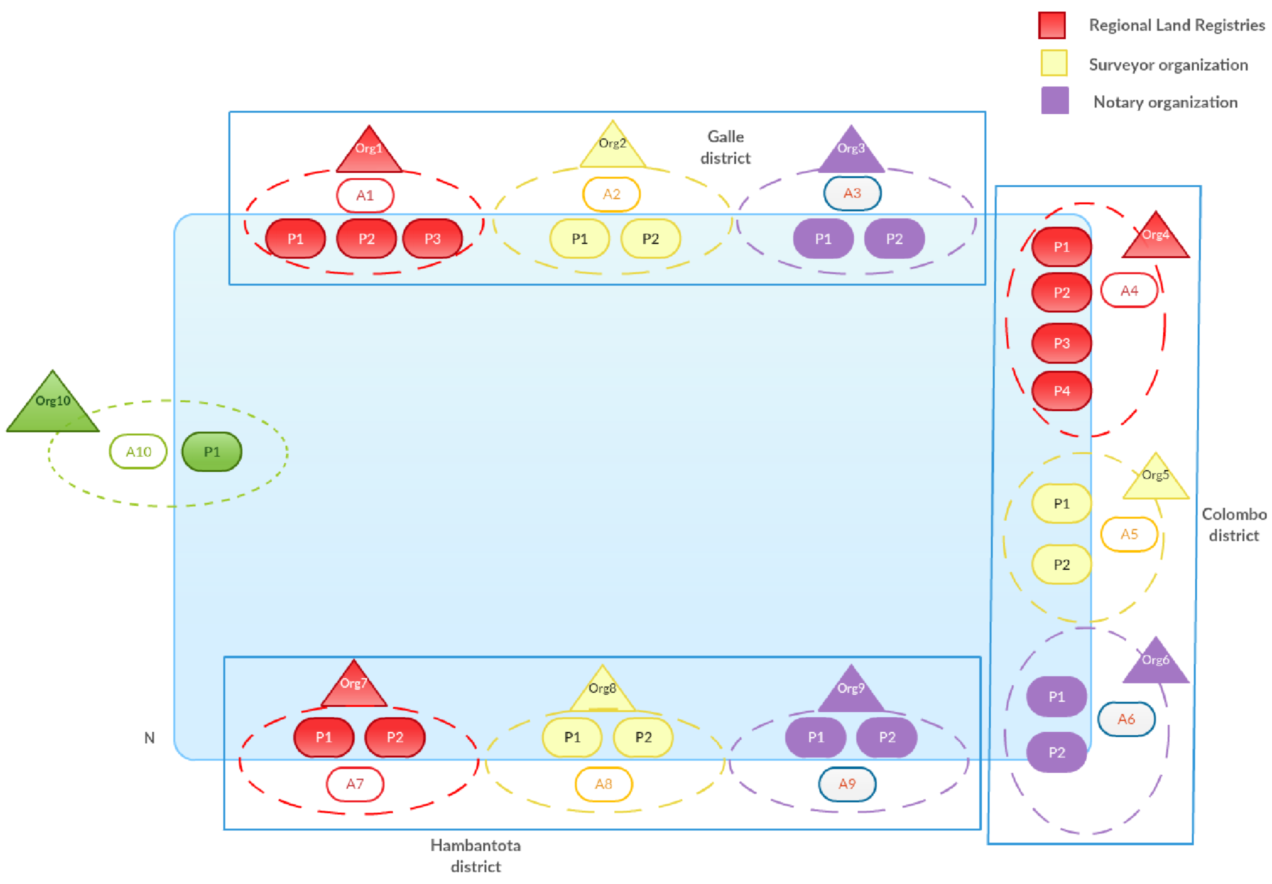


Figure 4.3: Architecture of the Three District Model

The number of peers and organizations for both Abstract Models is the same. Thus the crypto-config.yaml file which contains the configuration of the network, indicating the organizations and which peers belong to which organization is the same for both Abstract Models. Configtxgen tool consumes crypto-config.yaml file in order to provide the required configuration artifacts. As depicted in Figure 4.3, the orderer organization (org 10 in green colour) is an important organization which performs the ordering phase in the underlying Execute-Order-Validate architecture of Hyperledger Fabric.

Cryptogen tool consumes configtx.yaml and issues node credentials (X509 certificates) for the nine organizations, orderer organization and application clients. A generic permissioned DLT platform comprises of a membership manager which manages access of members to the DLT network. In Hyperledger Fabric, an MSP (Membership Service Provider) maintains identities of all nodes including clients, peers and Ordering Service Nodes (OSNs) issued by cryptogen tool, for the purpose of **authentication**. In addition, configtx.yaml includes the anchor peers (which facilitate cross organization communication) for each organization. First peer of each organization has been identified as its anchor peer during implementation. Thus, it could be observed that the role of the MSP is analogous to the role of Registrar General's Department which recognizes validators.

The distinction between AM1 and AM2 is, in AM1, each RLR maintains a land ledger containing details of lands belonging only to itself while in AM2, all RLRs have access to a single land ledger containing details of all lands situated in the island. Hyperledger Fabric's **channel architecture** was exploited in implementing the above mentioned distinction between two Abstract Models. A channel partitions the state of the Fabric DLT network. A single channel can maintain a separate ledger which is embedded in a distinct chaincode which is shared by all peers connected to the channel. In AM1, **each** RLR owns one channel, one chaincode for that channel and thus one land ledger (lands belonging to that RLR only). In contrast, AM2 contains a single channel to which all RLRs are connected, one chaincode for that channel and thus one land ledger representing all lands across the island.

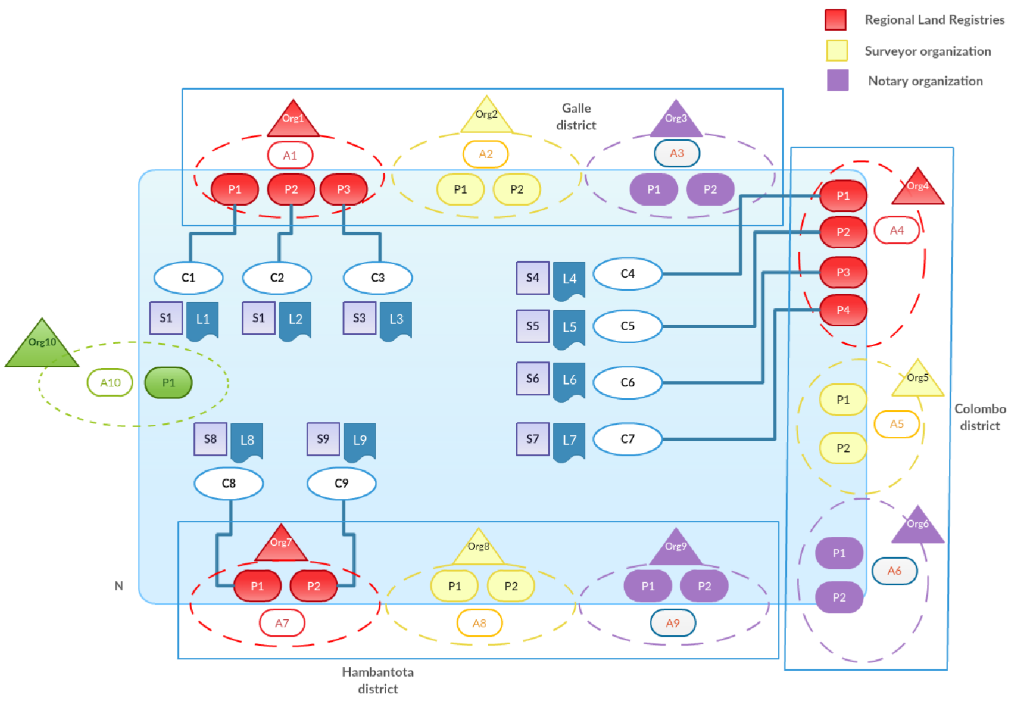


Figure 4.4: Channel architecture of the Three District Model in Abstract Model 1

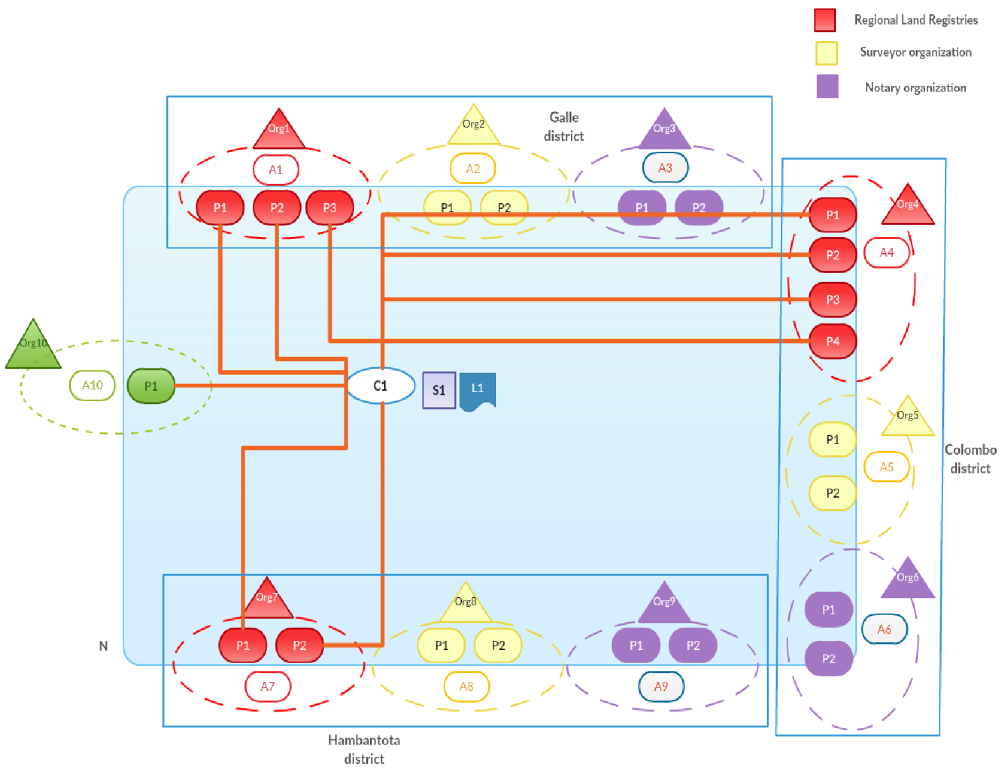


Figure 4.5: Channel architecture of the Three District Model in Abstract Model 2

Through figures 4.4 and 4.5, it could be observed that AM1 contains nine channels while AM2 contains only one channel, in the implemented Three District Model. The organizations which could access each channel is declared in configtx.yaml, since a channel requires its peers to be authenticated in order to provide access to the ledger.

As stated in Chapter 3, four validators were identified per a given land transaction for AM1. In AM1, it is compulsory for the RLR that the land belongs to, to endorse the transaction. A notary and a surveyor along with the RLR where the notary has been registered with are the remaining validators. In AM2, if the RLR that the land belongs to is overloaded with requests for validation, a RLR with low land transaction density could perform validation on behalf of the RLR with high land transaction density. As mentioned in the beginning of this chapter, endorsement policy specifies which validators or how many of them need to endorse a transaction proposal. Thus the two endorsement policies for the two Abstract Models are depicted as follows.

- AM1: AND (**org1**, OR(org1, org4, org7), OR(org2, org5, org8), OR(org3, org6, org9))

- AM2: AND (OR(org1, org4, org7), OR(org2, org5, org8), OR(org3, org6, org9))

As stated in Chapter 2, Hyperledger Fabric follows execute-order-validate architecture unlike other permissioned DLT platforms which follow order-execute architecture. During **Execute** phase, the transaction proposal submitted by the client is sent to endorsing peers as required by the invocation. The set of endorsing peers in the implemented solution is only a **subset of four peers**. Each endorsing peer executes the transaction against the ledger in peer-chaincode-container, based on chaincode logic and provides endorsements only after ensuring correctness of the transaction. Next, the endorsed transactions are sent to the Ordering phase. During **Ordering** phase, the underlying consensus protocol of the ordering service, produces an ordered sequence of endorsed transactions as blocks. In the next phase; **Validation** phase, the blocks are broadcast to all peers. There, each peer ***validates*** the change of ledger state due to endorsed transactions and ***update*** their ledgers in a deterministic order. Hyperledger Fabric's transaction processing protocol [7] is depicted in figure.

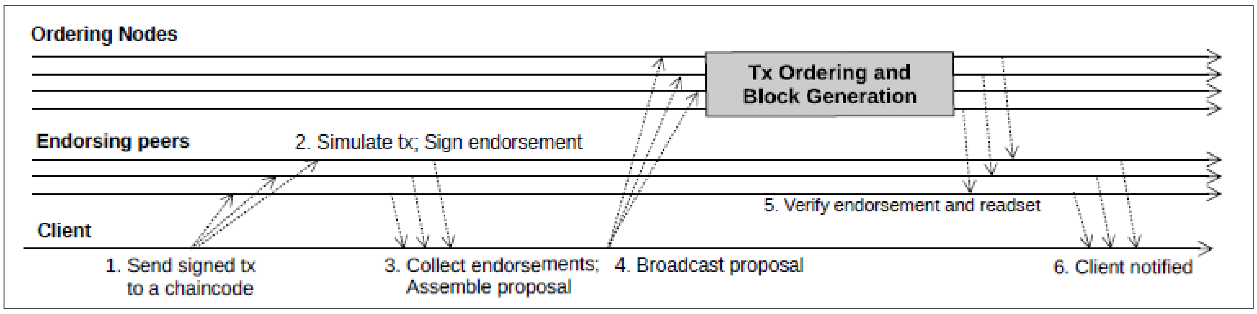

Ordering Nodes
Tx Ordering and Block Generation
Endorsing peers
2. Simulate tx; Sign endorsement
5. Verify endorsement and readset
Client
1. Send signed tx to a chaincode
3. Collect endorsements; Assemble proposal
4. Broadcast proposal
6. Client notified

Figure 4.6: Hyperledger Fabric's transaction processing protocol [7]

Thus, it is clear that, the introduction of AM2 as a remedy to the drawback caused by AM1 when there is a variation of land transaction density, across RLRs **improves only upon** the Execute phase of the Execute-Order-Validate architecture. When a set of land transactions depicting the real Sri Lankan scenario is submitted to AM1 and AM2 simultaneously, AM2 completes Execute phase earlier than AM1, due to workload distribution among RLRs in AM2.

Apart from the low overall transactional throughput of AM1 over AM2, implementation of AM1 is limited by the memory constraints of the implementation environment. As stated previously, a peer-chaincode container is created per each channel that the peer is connected to. Values for Total in Table 4.2 depicts that AM1 has created **five times** more Docker containers than AM2 due to its nine channels.

Table 4.2: Docker containers created by the two Abstract Models

Type of Docker container	Abstract Model 1	Abstract Model 2
Peer containers	21	21
Peer-chaincode containers	189	21
Total	210	42

During large scale implementation of the two Abstract Models by porting the two DLT networks to a cloud instance, implementation of AM1 is limited by the memory capacity of the instance. However, in production scale deployment, this issue could be resolved, since the peer container and peer-chaincode containers belonging to each peer are created at the peer nodes. Therefore, it is important to precisely state the storage specifications recommended for each peer node in AM1. In conclusion, AM1 requires the creation of more Docker containers than AM2.

4.4 Implementation of production scale DLT networks and facilitation of fault tolerance

Implementation details stated up to the previous section were based on Solo ordering service which is a single-node implementation. Although Solo ordering service is recommended for development and testing of DLT networks, it does not suffice in real production. This is because of, the inability of one ordering node to withstand crash faults.

Hyperledger Fabric facilitates plugging of three types of consensus protocols as its ordering service. They are 1) Solo ordering service, 2) Kafka-based ordering service and 3) BFT-SMaRT odering service. Table 4.3 provides a comparison of each ordering service with respect to multiple aspects, based on [15, 16, 17].

Table 4.3: Comparison of pluggable ordering services for Hyperledger Fabric [15,16,17]

	Solo ordering service	**Kafka-based ordering service**	**BFT-SMaRT ordering service**
Components	• Centralised non-replicated ordering service	• Decentralised, replicated ordering service. • Contains Apache Kafka cluster & Zookeeper ensemble	• Decentralised, replicated ordering service. • Ordering cluster contains 3f+1 nodes; f=no.of Byzantine crashes, and a set of frontends
Usage	For testing systems	At production level environments which are trustworthy, nevertheless nodes are prone to crashing	At production level environments which are untrustworthy as well as prone to crashing
Advantages	Requires few hardware resources	Robust	• Withstand both crash faults and Byzantine faults in an untrustworthy environment. • Could be configured to tolerate only crash faults in a trustworthy environment.
Disadvantages	Single point of failure	Withstand crash faults only	Not officially declared as the BFT ordering service of Hyperledger Fabric.
Fault tolerance capability	None	Crash Fault Tolerance (CFT)	Both Crash Fault Tolerance (CFT) and Byzantine Fault Tolerance (BFT)

4.4.1 Implementation of distributed land ledger with Kafka-based ordering service

Kafka-based ordering service consists of Hyperledger Fabric Ordering Service Nodes (OSNs), an Apache Kafka cluster and a Zookeeper ensemble. In Kafka-based ordering service, OSNs depend on Kafka brokers while Kafka cluster depends on Zookeeper ensemble. Apache Kafka which is a distributed commit log uses Zookeeper's metadata consistency protocol, in order to ensure tolerance of crash faults [16]. Kafka-based ordering service uses Zookeeper ensemble for storing metadata. Kafka cluster handles replicated data which is named as in-sync replicas (ISR). While Zookeeper ensemble executes a quorum system to ensure consistency of metadata, Kafka cluster requires all members of ISR to respond.

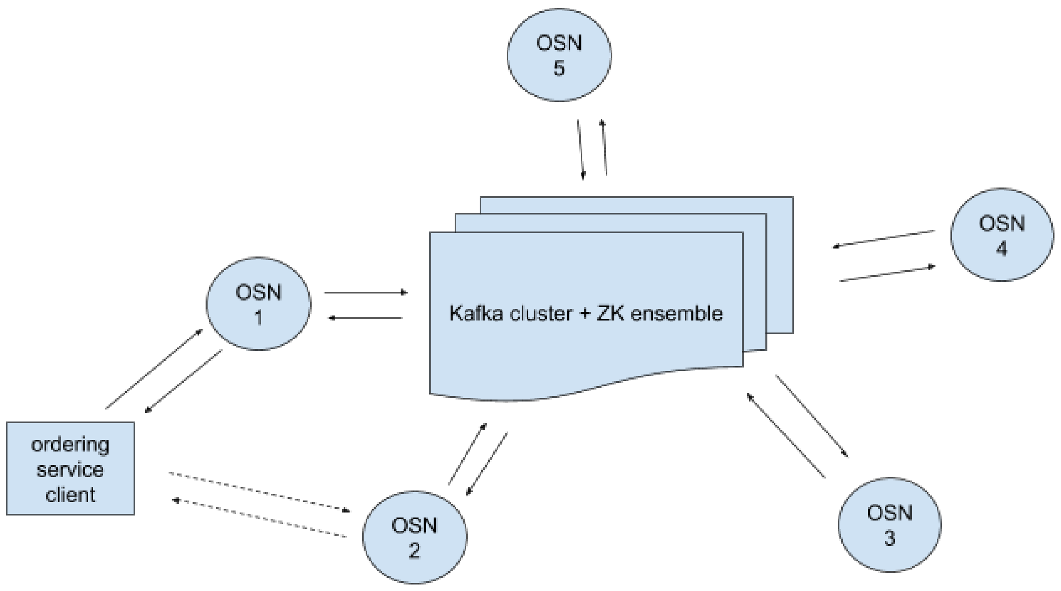


Figure 4.7: An ordering service, consisting of 5 Ordering Service Nodes (OSNs), and a Kafka cluster [15]. The ordering service client can connect to multiple OSNs.

In order to tolerate f crashes, the minimum number of in-sync replicas (M) should be f+1; M>1. N is the default replication factor, which is defined as the minimum number of Kafka brokers that should be alive. It is important that M<N and N<K; where K is the total number of Kafka brokers. Table 4.4 contains configuration of Kafka-based

ordering service to evaluate performance up to two crashes for the distributed land ledger solution.

Table 4.4: Configuration of Kafka-based ordering service tolerating up to two crashes

No. of crashes	M	N	K	Z (No.of Zookeeper nodes)	No.of OSNs
1	2	3	4	3	2
2	3	4	7	3	2

Based on [16], it could be deduced that through the use of ISRs and quorum system, Fabric has tried to overcome FLP result, while liveness is not 100% guaranteed.

During implementation, Kafka-based ordering service's nodes were granted identities and thus access to channels through crypto-config.yaml (where K, Z and OSNs were declared), configtx.yaml and kafka-base.yaml (where M,N were defined).

4.4.2 Implementation of distributed land ledger with BFT-SMaRT ordering service

BFT-SMaRT ordering service consists of an ordering cluster having 3f+1 nodes; f is the number of Byzantine crashes and a set of frontends [17]. Client applications cannot directly access the ordering service. Therefore, frontends relay envelopes on behalf of the client to the orderers. Subsequent to receiving ordered blocks generated by the ordering service, frontends relay those to peers for validation. Figure 4.8 depicts the architecture of BFT-SMaRT ordering service. Malicious clients are identified by invalid transactions added to the ledger (However, these transactions are prevented from being execute on the ledger).

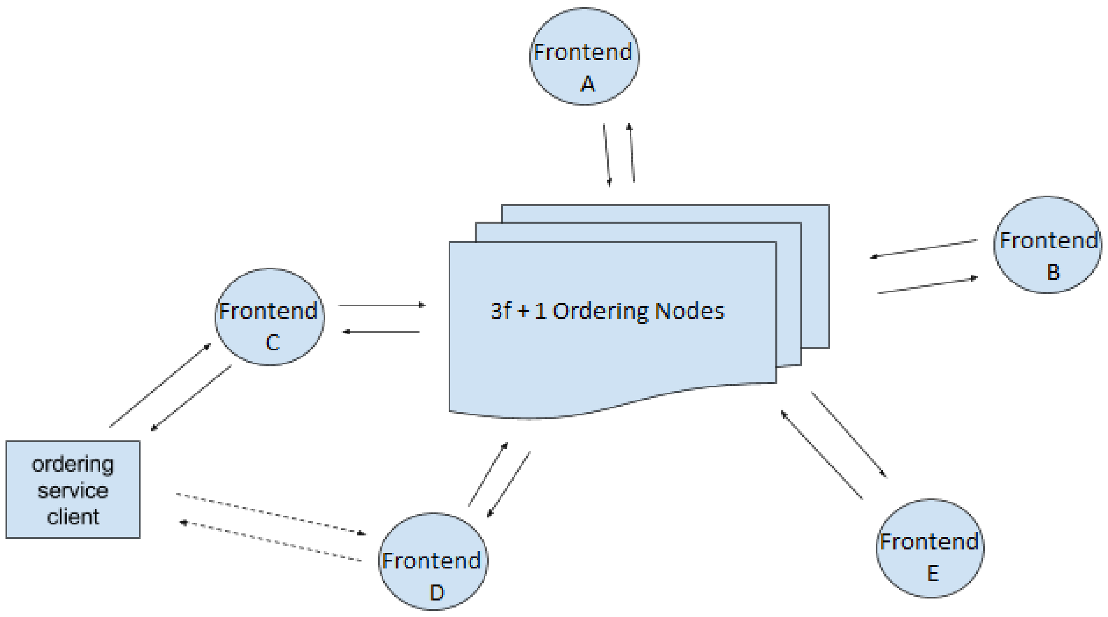


Figure 4.8: BFT-SMaRT ordering service, consisting of 5 frontends, and 3f+1 ordering nodes

Since, BFT-SMaRT ordering service has not been officially announced as the BFT ordering service of Hyperledger Fabric, implementation and testing had to be performed under limitations. Authors of the research work referred to by [17] state that Fabric's codebase is less suitable for BFT-SMaRT like ordering services, since Fabric produces a stream of envelopes unlike a stream of blocks required for BFT. Evaluation result obtained for BFT-SMaRT ordering service (performed under limitations) is included in Appendix A.

4.5 Summary

This chapter provided the most important and relevant, high level implementation details of the distributed land ledger solution, provided using **Hyperledger Fabric version 1.2**. Section 4.2 presented implementation details of land ledger and transactions. Subsequent to a statement on the first research question, implementation details of the two abstract models were presented. Advanced implementation of production scale DLT networks and facilitation of fault tolerance were explained finally.

Chapter 5 - Results and Evaluation

5.1 Introduction

The main purpose of performing evaluation was to provide answers to the second research question. A statement on the first research question was provided in section 4.2.1, focusing on the implemented optimal land ledger content and the implemented transactions. This chapter present a performance evaluation of the two abstract models based on heterogeneous land transaction density conditions across RLRs and failure conditions. Evaluation model followed in this chapter was devised based on [18, 19], while sub evaluations have been performed to verify speculations made during main evaluation process.

Implementation in Chapter 4 as well as evaluation was performed based on Hyperledger Fabric version 1.2. Implementation and evaluation of the two Abstract Models were performed on an AWS t2.large instance (64bit X86, 2 vCPUs, 8GiB memory) with Ubuntu 18.04.1, with Docker version 18.06.0 and Docker Compose version 1.21.2. Figure 5.1 shows the test flow for Hyperledger Fabric v1.2 followed during evaluation.

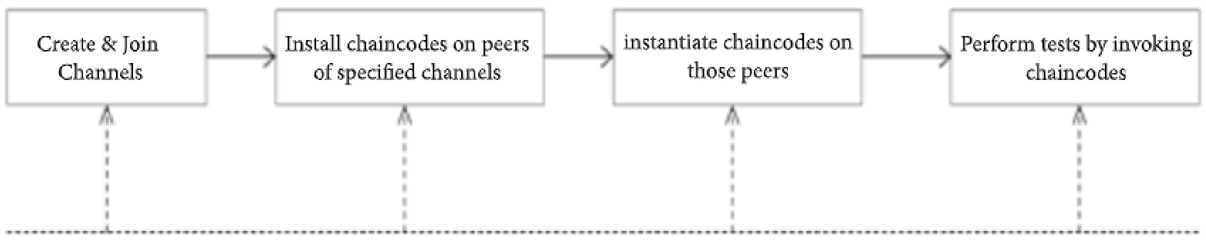

Create & Join Channels
Install chaincodes on peers of specified channels
instantiate chaincodes on those peers
Perform tests by invoking chaincodes

Figure 5.1: Test flow for Hyperledger Fabric v1.2 followed during evaluation

The Three District Model implemented in Chapter 4 was used for evaluation and thus community size is a constant across both AM1 and AM2. Both Abstract Models consist of nine organizations providing a community of twenty-one peers.

Evaluation has been performed for a scenario where one RLR has a higher land transaction density than others. Sets of simultaneous transactions of varying size were submitted to the DLT network throughout the evaluation process. Within a set of transactions, the ratio between changeLandOwner transaction invocations and forkLand transaction invocations is 3:2, unless stated otherwise. It was ensured that the execution of submitted transactions was not interrupted by queries.

Primary performance metrics of Hyperledger Fabric as stated in [18] are,
Throughput:- Rate at which transactions are committed to the ledger. i.e. the rate at which transactions complete the Execute-Order-Validate phase.
Latency:- Time taken from application sending the transaction proposal to the transaction commit.

Throughput and Latency were evaluated against the number of transactions which are submitted to the DLT network simultaneously. Results obtained for throughput and latency (in the form of timestamps) were averaged over three rounds and thus the presented results of throughput and latency are average throughput and average latency respectively.

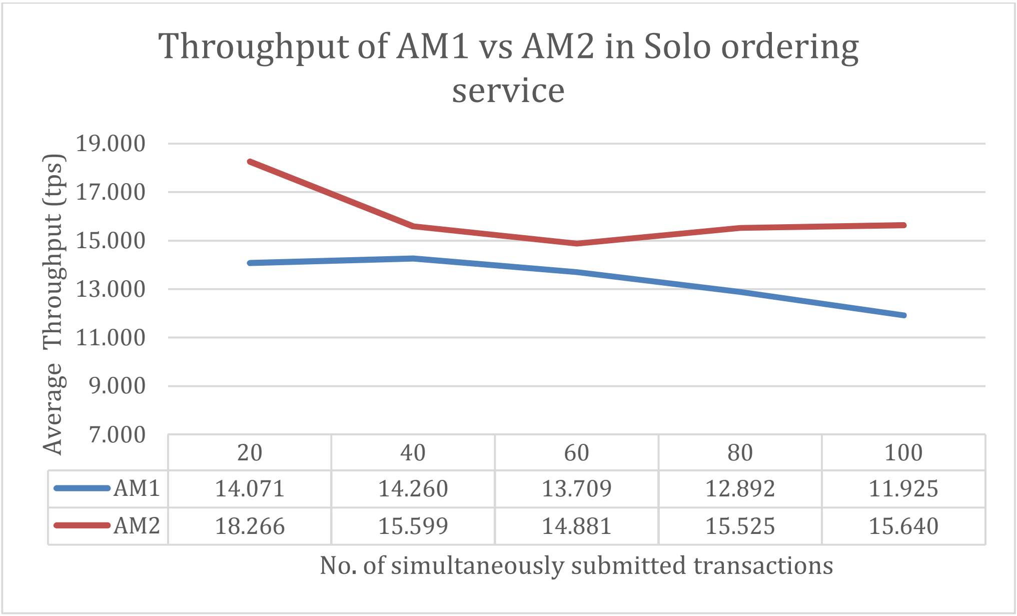
- Throughput= (number of simultaneously submitted transactions)/ (execution end time- execution start time)

5.2 Evaluation of the two Abstract Models

As stated in [14], Fabric being a complex distributed system, its performance depends on the choice of distributed application and transaction size, ordering service, consensus protocol, network parameters, topology of the nodes in the network, number of nodes and channels, further configuration parameters and network dynamics. During evaluation of the two Abstract Models, the dependence on performance on topology of nodes has been evaluated.

5.2.1 Evaluation of the two Abstract Models in Solo ordering service

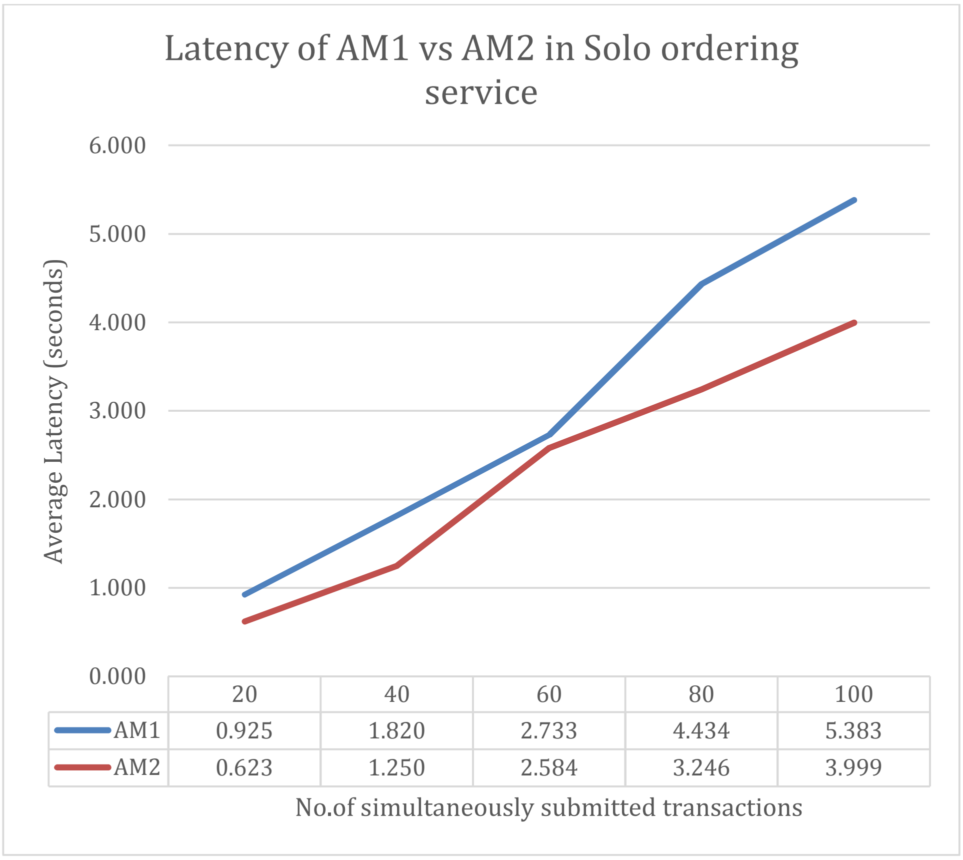
AM1 and AM2 were evaluated for throughput and latency in Solo ordering service.



	20	40	60	80	100
AM1	14.071	14.260	13.709	12.892	11.925
AM2	18.266	15.599	14.881	15.525	15.640

Figure 5.2: Throughput of AM1 vs AM2 in Solo ordering service

It could be observed that the throughput of AM2 is higher than the throughput of AM1 for all evaluated workloads. The throughput of both models decrease when load increases, because of the bottleneck at ordering and validation phases. Since Fabric deploys a queuing system, with a high load, the waiting time increases exponentially and hence throughput decreases [19]. The gain of AM2 is clearly visible when the load is greater than 60 where the throughput increase and tries to stabilize. The validation overhead on RLR with high land transaction density in AM1 causes its throughput to decrease linearly after a workload of 60.



	20	40	60	80	100
AM1	0.925	1.820	2.733	4.434	5.383
AM2	0.623	1.250	2.584	3.246	3.999

Figure 5.3: Latency of AM1 vs AM2 in Solo ordering service

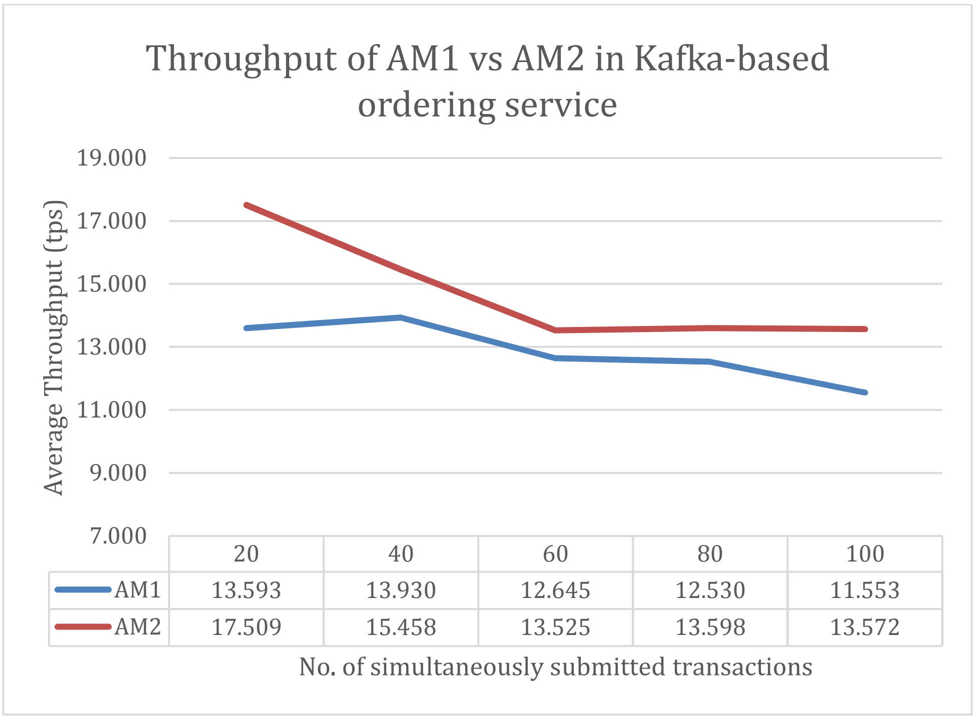
Latency of AM2 is lesser than that of AM1, as hypothesized. Although the latencies of both models increase with load, latency of AM1 is always higher than that of AM2. It could be observed that the rate of increase in latency of AM1 is higher than that of AM2. AM1 has a higher latency due to the bottleneck of validation at the RLR with a higher land transaction density.

However, it should be noted that the approach taken by AM2, only improves upon execute phase of Execute-Order-Validate architecture as explained in Chapter 4. When transaction proposals are submitted simultaneously to the two Abstract Models, AM2 completes Execute phase earlier than AM1. This is because, workload is shared among RLRs in AM2, unlike in AM1 where there is high endorsement overhead on RLRs with high land transaction density. During Order and Validation phases, both models are subject to ordering and validation bottlenecks. In conclusion, AM2 has a higher throughput and a lower latency than AM1 for the evaluated workloads.

5.2.2 Evaluation of the two Abstract Models in Kafka-based ordering service

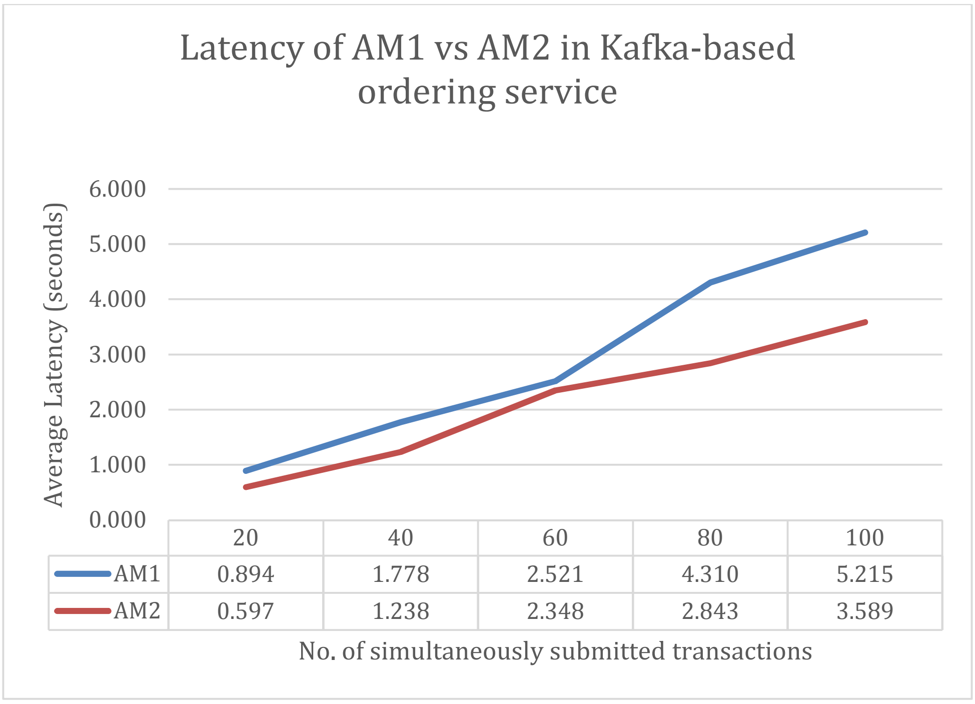
AM1 and AM2 were evaluated for throughput and latency in Kafka-based ordering service.

Specifications of Kafka-based ordering service: - 2 orderer nodes, 4 Kafka brokers, 3 Zookeeper nodes.



	20	40	60	80	100
AM1	13.593	13.930	12.645	12.530	11.553
AM2	17.509	15.458	13.525	13.598	13.572

Figure 5.4: Throughput of AM1 vs AM2 in Kafka-based ordering service



	20	40	60	80	100
AM1	0.894	1.778	2.521	4.310	5.215
AM2	0.597	1.238	2.348	2.843	3.589

Figure 5.5: Latency of AM1 vs AM2 in Kafka-based ordering service

Interpretation of throughput and latency graphs in section 5.2.1 is valid for graphs figures 5.4 and 5.5 as well. However, the average throughput for both AM1 and AM2 in Kafka-based ordering service is slightly less than the corresponding values evaluated under Solo ordering service, due to the **tradeoff with Crash Fault Tolerance (CFT)**.

5.2.3 Evaluation of the two Abstract Models for changeLandOwner transactions

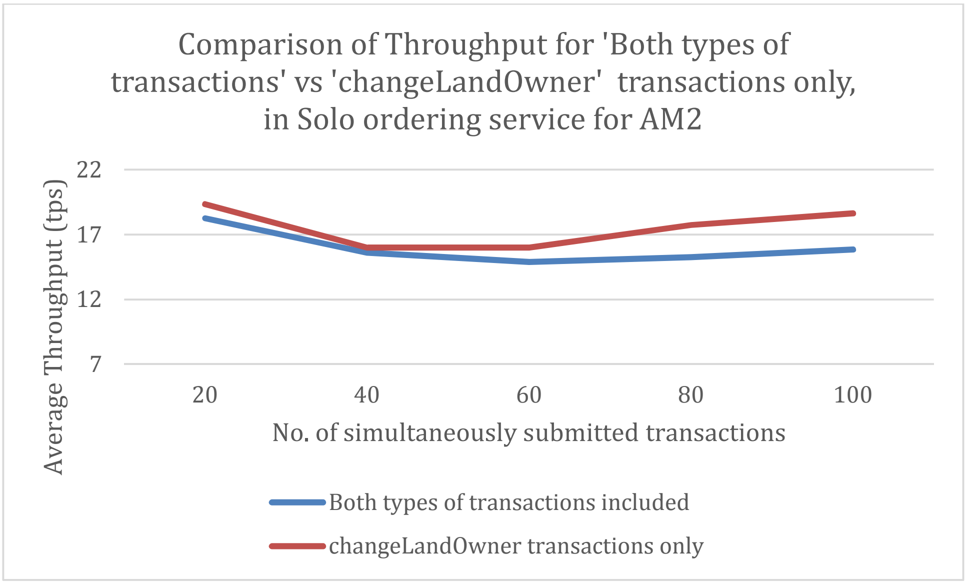
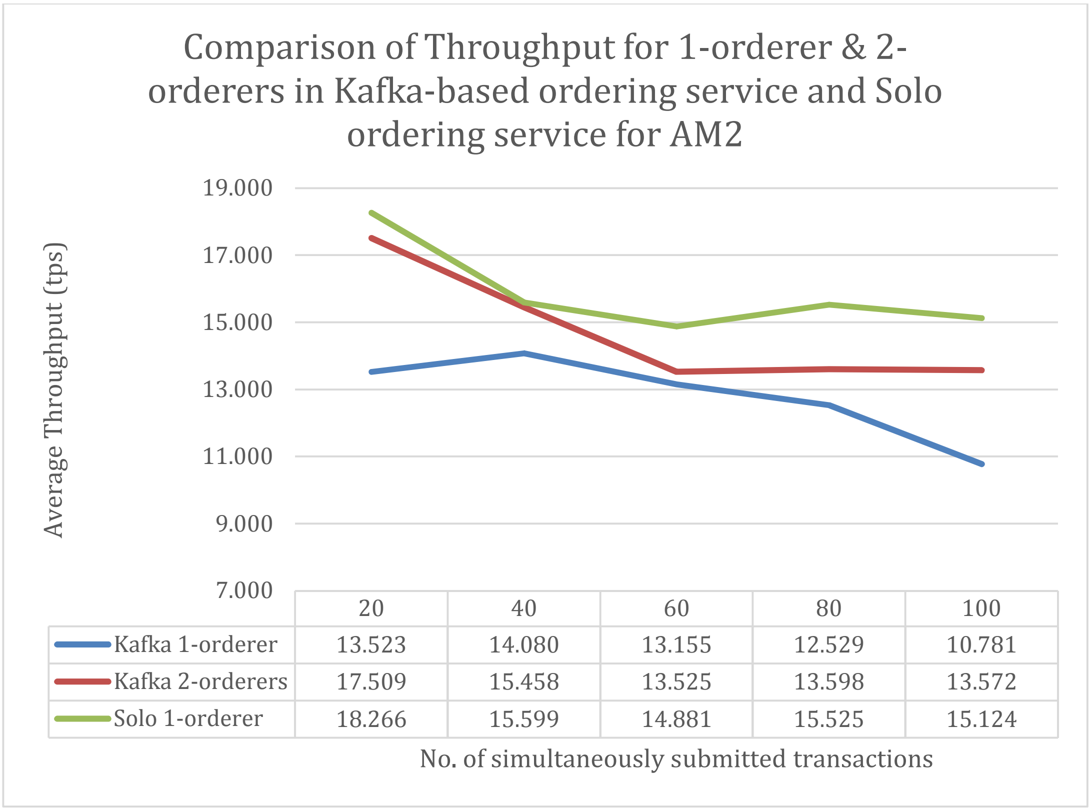


Figure 5.6: Comparison of Throughput for 'Both types of transactions' vs 'changeLandOwner' transactions only, in Solo ordering service for AM2

It could be observed that the average throughput of both Abstract Models is generally at a lower value. As mentioned earlier, since the performance of Fabric depends on many features other than topology of nodes in the DLT network, it was decided to repeat evaluation in section 5.2.1 for a set of **small sized transactions**. Through this attempt, it was required to observe whether the size of land transactions have an effect on the low overall throughput. Thus evaluation in section 5.2.1 was repeated by submitting transactions comprising only of **changeLandOwner** transactions. It could be observed through Figure 5.6, that the throughput is higher than that of graph in Figure 5.2 especially when load increases. Thus it could be inferred that the size of land transactions has an effect on the low overall throughput.

5.2.4 Evaluation of the bottleneck at the ordering service (during ordering phase)

Since it was speculated that the ordering service bottleneck could be one reason for the decreasing throughput in sections 5.2.1 and 5.2.2 over increasing workload, it was decided to evaluate the bottleneck at the ordering service, for Solo ordering service, and 1-orderer & 2-orderers in Kafka based ordering service.



	20	40	60	80	100
Kafka 1-orderer	13.523	14.080	13.155	12.529	10.781
Kafka 2-orderers	17.509	15.458	13.525	13.598	13.572
Solo 1-orderer	18.266	15.599	14.881	15.525	15.124

Figure 5.7: Comparison of Throughput for 1-orderer & 2-orderers in Kafka-based ordering service and Solo ordering service.

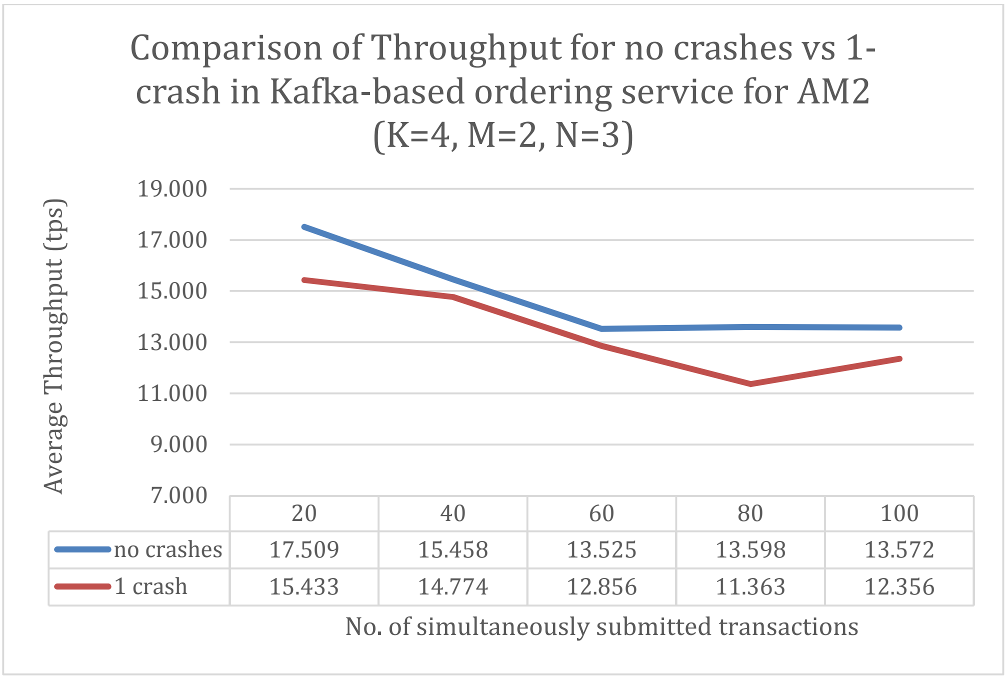
The throughput of AM2, which has a higher overall throughput than AM1, was evaluated in this section. As it could be observed, in Figure 5.7, the throughput curve for AM2 is the highest for Solo which has 1-orderer. From among the throughput curves for Kafka-based ordering services, throughput for 2-orderers is higher than that for 1-orderer for all evaluated workloads. Solo has highest throughput in all cases because it **does not facilitate any kind of fault tolerance**. Therefore, it could be concluded that the bottleneck at the ordering service has an effect on the decreasing throughput with increasing workload.

5.3 Evaluation for Crash Fault Tolerance (CFT)

Fault tolerance of a Fabric network could be evaluated at two levels. They are at, 1) the level of validating nodes and 2) the level of ordering service.

Fault tolerance capabilities of the two Abstract Models at the level of validating nodes (endorsing peers), is governed by their respective endorsement policies. In AM1, it is compulsory for the RLR that the land belongs to, to endorse all land transactions in its territory. **If the node representing that RLR is crashed, none of those transactions could be endorsed and thus committed to the ledger.** But, with AM2 even if the RLR node is crashed, any other RLR could validate the transaction/s on behalf of the original RLR. Therefore, with the correct choice of endorsement policy CFT of validating nodes could be ensured. Thus, it is evident that AM2 is more suitable for implementing a distributed land ledger for Sri Lanka when fault tolerance at the level of validating nodes is considered. Thus we could provide an answer for the second research question, by stating that, AM2 ensures robustness, among the two Abstract Models under crash failures.

Hyperledger Fabric mainly focuses on crash fault tolerance at the level of ordering service, i.e. crashing of Kafka brokers of the Kafka-based ordering service. Since AM2 is capable of tolerating crash faults at the level of validating nodes, crash fault tolerance at the level of ordering service was evaluated for AM2. Based on Table 4.4 in Chapter 4, in order to compare the CFT of Kafka-based ordering service for no crashes against one crash, the model with K=4, M=2, N=3, Z=3 and no. of OSNs=2 was selected.

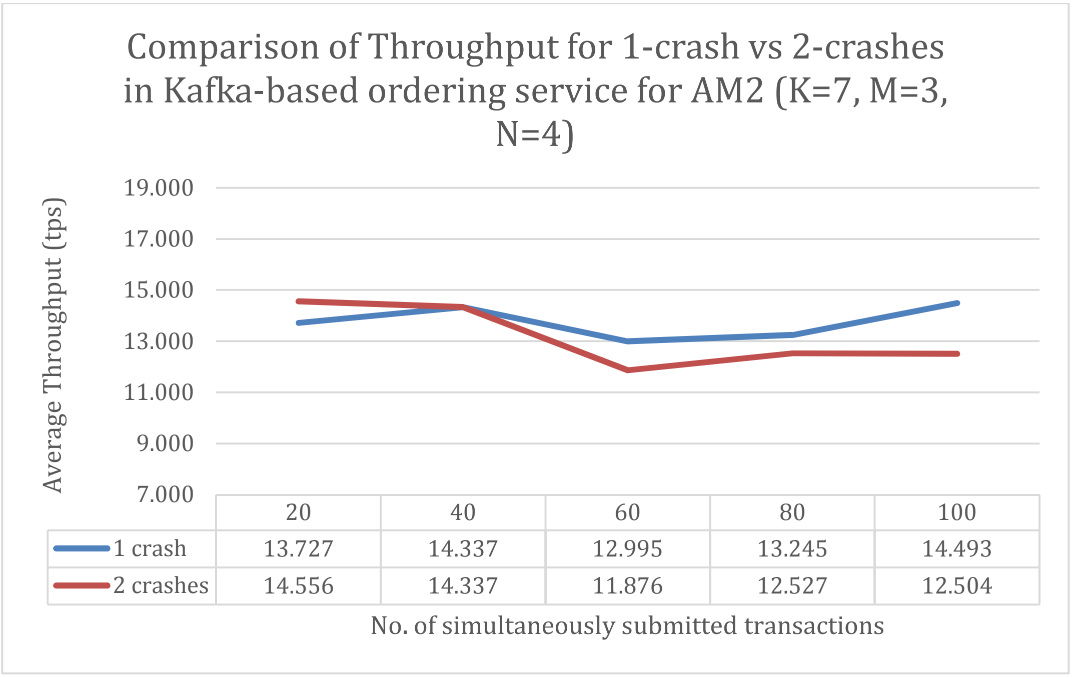


	20	40	60	80	100
no crashes	17.509	15.458	13.525	13.598	13.572
1 crash	15.433	14.774	12.856	11.363	12.356

Figure 5.8: Comparison of Throughput for no crashes vs 1-crash in Kafka-based ordering service for AM2 (K=4, M=2, N=3)

It could be observed that average throughput is higher when there are no crashes than when there is one crash. However, Kafka-based ordering service has ensured that there is **no significant drop** in throughput when one crash has occurred.

In order to compare the throughput when there is one crash against when there are two crashes, the model with K=7, M=3, N=4, Z=3 and no.of OSNs=2 was selected.



	20	40	60	80	100
1 crash	13.727	14.337	12.995	13.245	14.493
2 crashes	14.556	14.337	11.876	12.527	12.504

Figure 5.9: Comparison of Throughput for 1-crash vs 2-crashes in Kafka-based ordering service for AM2 (K=7, M=3, N=4)

Similar to graph in Figure 5.8, it could be observed that Kafka-based ordering service has ensured that there is no significant drop in throughput when two crashes have occurred.

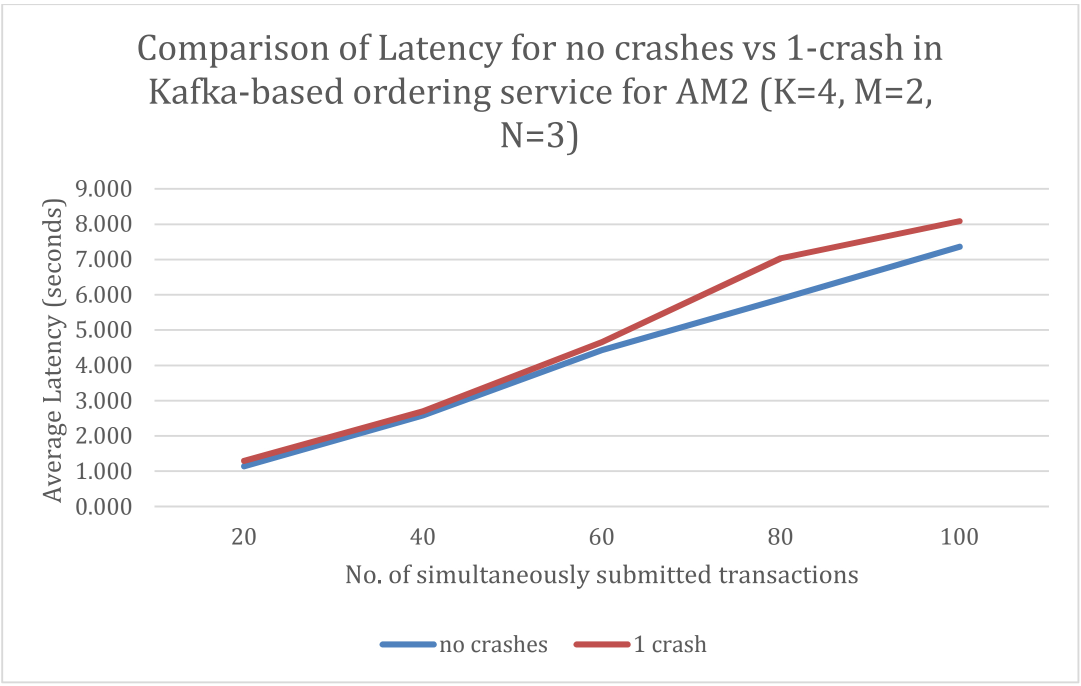


Figure 5.10: Comparison of Latency for no crashes vs 1-crash in Kafka based ordering service for AM2 (K=4, M=2, N=3)

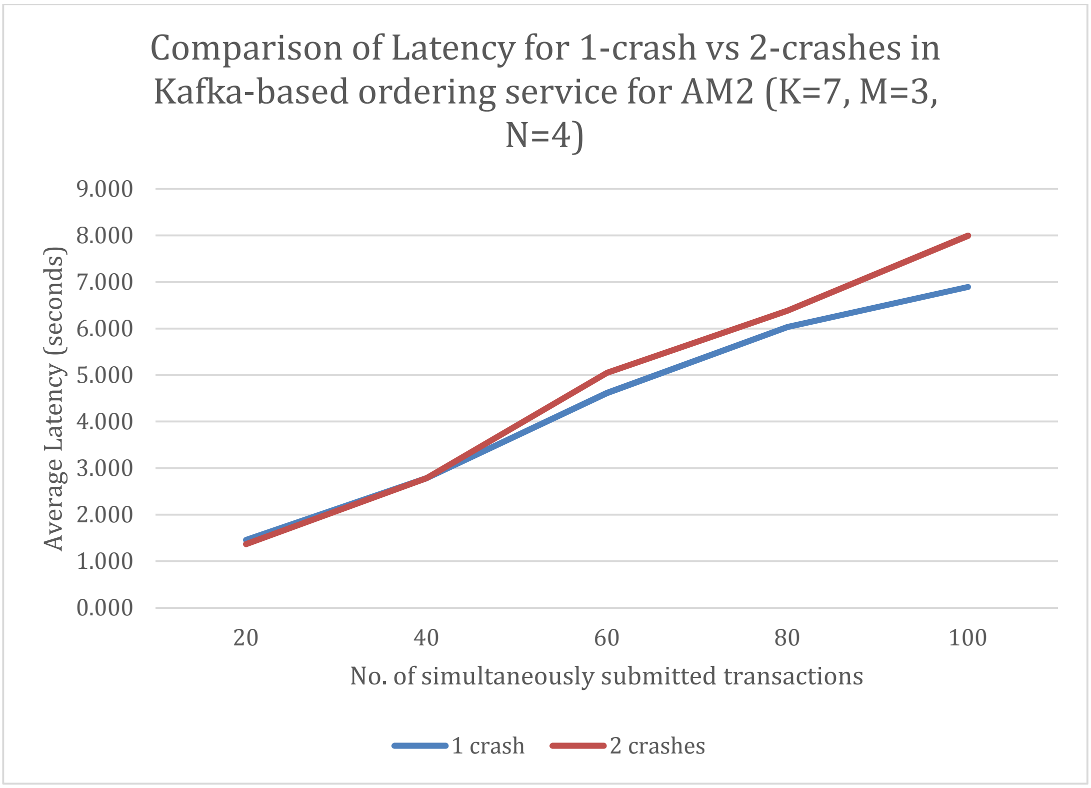


Figure 5.11: Comparison of Latency for 1-crash vs 2-crashes in Kafka-based ordering service for AM2 (K=7, M=3, N=4)

Figures 5.10 and 5.11 include latency graphs for the corresponding throughput graphs included in figures 5.9 and 5.10. It could be observed that the latency is higher when one crash occurs, especially when workload increases.

5.4 Summary

Through the evaluation performed in this chapter, it could be inferred that the average throughput of AM2 is higher than AM1 for heterogeneous land transaction density conditions, as hypothesized. Average latency of AM2 is lesser than that of AM1. When the simultaneously offered workload increases over 60, average throughput of AM2 increases and stabilizes, while the average throughput of AM1 linearly decreases. Similarly, the rate of increase of latency in AM1 is higher than that of AM2. It could be verified that one reason for the low overall throughput of both Abstract Models is, the size of land transactions. Further, it was observed that the throughput of both Abstract Models decrease with increasing workload due to the ordering service bottleneck.

It could be confirmed that AM2 is highly robust to crash faults at the level of validating nodes, unlike AM1. It was identified that Kafka-based ordering service provided by Hyperledger Fabric ensures that there is no significant drop of throughput in AM2 for a given configuration when one crash failure occurs.

As explained in the previous chapter, implementation and evaluation of BFT-SMaRT ordering service was performed under strict limitations and the result of evaluation is included in Appendix A.

Chapter 6 - Conclusions

6.1 Introduction

The aim of this research was to provide a permissioned distributed ledger solution for the Sri Lankan land transaction scenario, in order to overcome the inefficiency and ineffectiveness of the current manual land registration systems in practice. Through this research, a permissioned distributed land ledger for Sri Lanka was designed, implemented using Hyperledger Fabric and evaluated for performance in terms of transaction throughput and latency.

During designing of the DLT solution, optimal ledger content was extracted from the current folio and transactions to be performed against the ledger were devised based on the real Sri Lankan land transaction scenario. As expected initially, we could successfully get away with the folio system while embedding folio details in the land ledger. With the intention of proposing the most suitable DLT solution, two abstract models were designed, such that AM1 closely resembles the current manual Sri Lankan land transaction scenario, while AM2 is more suitable for a distributed setting of the Sri Lankan situation, with regards to land transaction density variation across RLRs.

Subsequently, the two Abstract Models of the land ledger were implemented using Hyperledger Fabric and both were evaluated for performance in terms of transactional throughput and latency under different land transaction density conditions and failure conditions.

6.2 Conclusions about research questions

When the capabilities and limitations of adapting an open source solution for implementing a distributed land ledger for Sri Lanka is considered, the conclusion could be presented along two aspects. Since the implementation of the distributed land ledger for Sri Lanka, using Hyperledger Fabric, has been able to provide all the queries and transaction invocations as suggested by the design in Chapter 3, it could be stated that the capabilities of Hyperledger Fabric is high. Most importantly, since we could get away with the folio system, while facilitating derivation of pedigree/ folio tree from Fabric's transaction log, the fact that Hyperledger has high capabilities is confirmed. Turning towards the other aspect, concerning Abstract Models, Fabric's concept of organizations, channel architecture and endorsement policies helped implement the real Sri Lankan validation policies. Thus it could be concluded that features provided by Hyperledger Fabric are ideal in implementing a distributed land ledger solution for Sri Lanka.

Before considering the performance difference between two Abstract Models under different land transaction density conditions and failure conditions, based on observations made during implementation, it could be stated that AM2 is better than AM1, in terms of the number of Docker containers created. As it could be interpreted in Chapter 4, AM2 performs better than AM1, due to its higher throughput & lower latency under heterogeneous land transaction density conditions and tolerance of crash faults of RLRs unlike AM1. Further, it was identified that Kafka-based ordering service provided by Hyperledger Fabric ensures that there is no significant drop of throughput in AM2 for a given configuration when one crash failure occurs. Thus, it could be concluded that AM2 is ahead of performance than AM1 under different land transaction density conditions and failure conditions.

The next section on limitations and further work would present future prospects and possibilities for implementing a large scale distributed land ledger model for Sri Lanka.

6.4 Limitations and Implications for further research

When providing a conclusion (opinion) on the third research question, the limitations identified during this research, possible causes and **remedies** to overcome those limitations could be considered as future prospects for implementing a large scale distributed land ledger model for Sri Lanka.

In Chapter 5, it was verified that one reason for the overall low throughput of both Abstract Models is the size of land transactions. Therefore, when implementing a large scale distributed land ledger, it is recommended to minimize the size of land transaction invocations as much as possible.

AM2 was evaluated for CFT under Kafka-based ordering service. During large scale implementation, further testing for both CFT and BFT is recommended. However, the evaluation models used for testing CFT and BFT in this research, could be improved upon to achieve the above requirement.

Due to the Execute-Order-Validate architecture of Hyperledger Fabric, the expected performance gain through AM2 was not attainable. However, it is recommended that the proposed solution be implemented and evaluated on a permissioned DLT network with similar features to Fabric but with Order-Execute architecture, which is expected to provide a higher performance enhancement than what Hyperledger Fabric provided for AM2.

Since the DLT network was created as a network of Docker containers on a single instance, the CPU power is divided among all peer nodes. Since it could be speculated as another reason for the low overall throughput, a *resource allocation evaluation* could be performed as future work. During production scale deployment of the proposed solution, it is important to provide higher CPU power (by allocating multiple CPU cores) on peer nodes [18].

During evaluation, a set of transactions were submitted simultaneously to the DLT network. However, arrival rates of transactions in real world production systems would be following certain distributions. Thus, a workload generator which generates concurrent transactional proposals, while maintain the land transactions density variation of the real Sri Lankan scenario, is suggested as essential future work, when evaluating a large scale DLT solution.

In a real world setup, nodes in the DLT network would be geographically distributed. Although during evaluation we assumed that the network is not a bottleneck, it is important to evaluate the effect of network latencies on throughput as future work.

In Honduras, due to the frequent facing of power outages, security problems and high cost of electricity, the blockchain based land registry application has been deployed on the infrastructure provided by a cloud service provider [7]. It is recommended that a large scale implementation of the distributed land ledger solution for Sri Lanka also be deployed on an IaaS platform. If the above recommended deployment is opted to in the future, a performance evaluation should be performed on instances in a **datacenter**.

Although Hyperledger has moved a step forward, by moving away from hardcoded consensus like other permissioned DLT platforms [14], a **presumption** of the trust model of the deploying environment is required. Nevertheless, if a cross fault tolerant consensus protocol (e.g.: XFT) is pluggable with Hyperledger, a presumption of the trust model would not be required. Development of a pluggable cross fault tolerant consensus protocol for Hyperledger Fabric, would be possible future work.

References

[1] T. Perera, "Implementing Land Registration Systems in Sri Lanka: Being Pragmatic," *Sri Lankan Journal of Real Estate,* vol. 4, pp. 74-96, 2011.

[2] J. Vos, "Blockchain-based Land Registry: Panacea, Illusion or something in between?," in *IPRA/CINDER congress, European Land Registry Association (ELRA)*, Dubai, 2017.

[3] R. Pipan, "Bitfury," 7 February 2016. [Online]. Available: https://bitfury.com/content/downloads/the_bitfury_group_republic_of_georgia_expand_blockchain_pilot_2_7_16.pdf. [Accessed 7 January 2019].

[4] Lantmateriet (The Swedish Mapping cadastre and land registration authority), Telia Company, Chromaway and Kairos Future, "The Land Registry in the blockchain," Kairos Future, 2016.

[5] "Estonian blockchain technology," [Online]. Available: https://e-estonia.com/wp-content/uploads/faq-a4-v02-blockchain.pdf. [Accessed 7 January 2019].

[6] J. Mirkovic, "Blockchain Cook County — Distributed Ledgers for Land Records," The Illinois Blockchain Initiative, Illinois, 2017.

[7] J. C. Collindres, M. Regan and G. P. Panting, "Using Blockchain to Secure Honduran Land Titles," Fundacion Eleutra, Honduras, 2016.

[8] "Real Estate Land Title Registration in Ghana," 27 March 2016. [Online]. Available: http://bitlandglobal.com/. [Accessed 7 January 2019].

[9] "Whitepaper on Distributed Ledger Technology. Volume 1," Hong Kong Monetary Authority, Hong Kong, November, 2016.

[10] C. Cachin and M. Vukolic, "Blockchain Consensus Protocols in the Wild," in *31st International Symposium on Distributed Computing (DISC 2017)*, Vienna, Austria, 2017.

[11] "An Introduction to Hyperledger," The Hyperledger White Paper Working Group, 2018.

[12] "Hyperledger Archtecture, Volume 1 Introduction to Hypereldger Business Blockchain Desing Philosophy and Consensus," Hypereldger Architecture Working Group, 2017.

[13] "Hyperledger Architecture, Volume 2 Smart Contracts," Hyperledger Architecture Working Group, 2017.

[14] E. Androulaki, C. Cachin, C. Ferris and S. Muralidharan, *Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains,* arXiv:1801.10228v2 [cs.DC] 17 Apr 2018, 2018.

[15] "Bringing up a Kafka-based Ordering Service," 2017. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.2/kafka.html. [Accessed 7 January 2019].

[16] F. Junqueira and N. Narkhede, "Distributed Consensus Reloaded: Apache ZooKeeper and Replication in Apache Kafka," Confluent, 27 August 2015. [Online]. Available: https://www.confluent.io/blog/distributed-consensus-reloaded-apache-zookeeper-and-replication-in-kafka/. [Accessed 7 January 2019].

[17] J. Sousa, A. Bessani and M. Vukoli´c, "A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform," in *The 48th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2018)*, Luxembourg City, Luxembourg, 2018.

[18] P. Thakkar, S. Nathan and B. Viswanathan, "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform," in *arXiv:1805.11390v1 [cs.DC] 29 May 2018*, 2018.

[19] Q. Nasir, I. Qasse, M. A. Talib and A. B. Nas, "Performance Analysis of Hyperledger Fabric Platforms," *Security and Communication Networks,* vol. 2018, p. 14, 2018.

Appendix A: Diagrams

Diagram on the structure of the Folio

- Folio No.
- Location of land
- Boundaries
- Extent
- Name of land
- Plan No. and Date
- Name of surveyor
- Lot No.

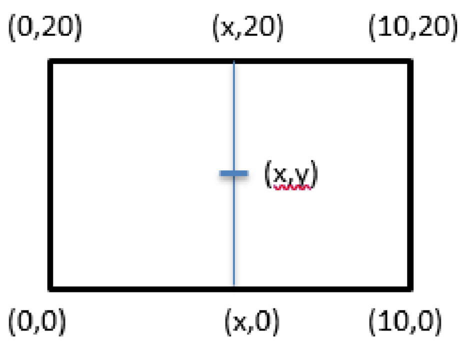
Fixed details regarding a land

No. and date of deed	Name of Notary	Registration stampduty	Grantors	Grantees	Remarks regarding transaction	Signature of registrar and date of signature
One record per each land transaction						

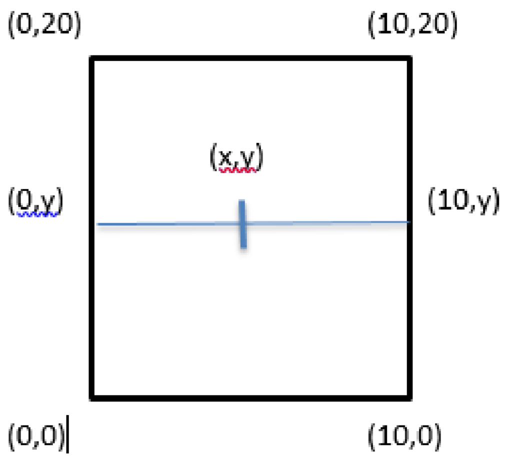
Logic used to check overlapping boundaries during forkLand transaction

Boundary checking logic for forkLand transaction on splitting a land into two pieces is explained below. Consider a vertical division of a rectangular land. Coordinates of the center of the vertical line has to be provided as input to the system. The chaincode checks whether that coordinate lies within the boundaries of original land (indicated by the coordinates of the four corners). If so, two new lands are registered with boundaries calculated by the chaincode.

Horizontal division



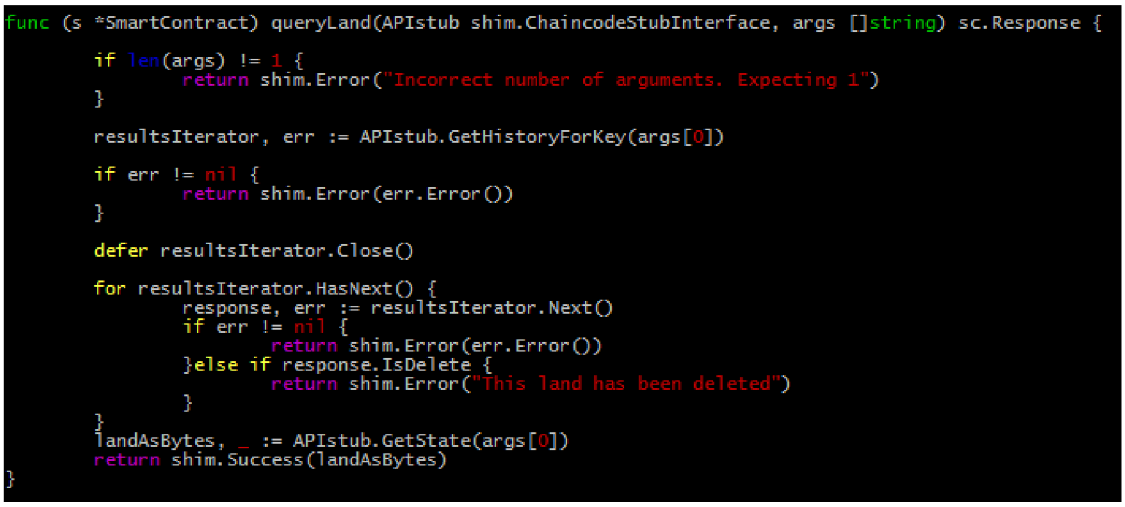
Vertical division



Appendix B: Code Listings

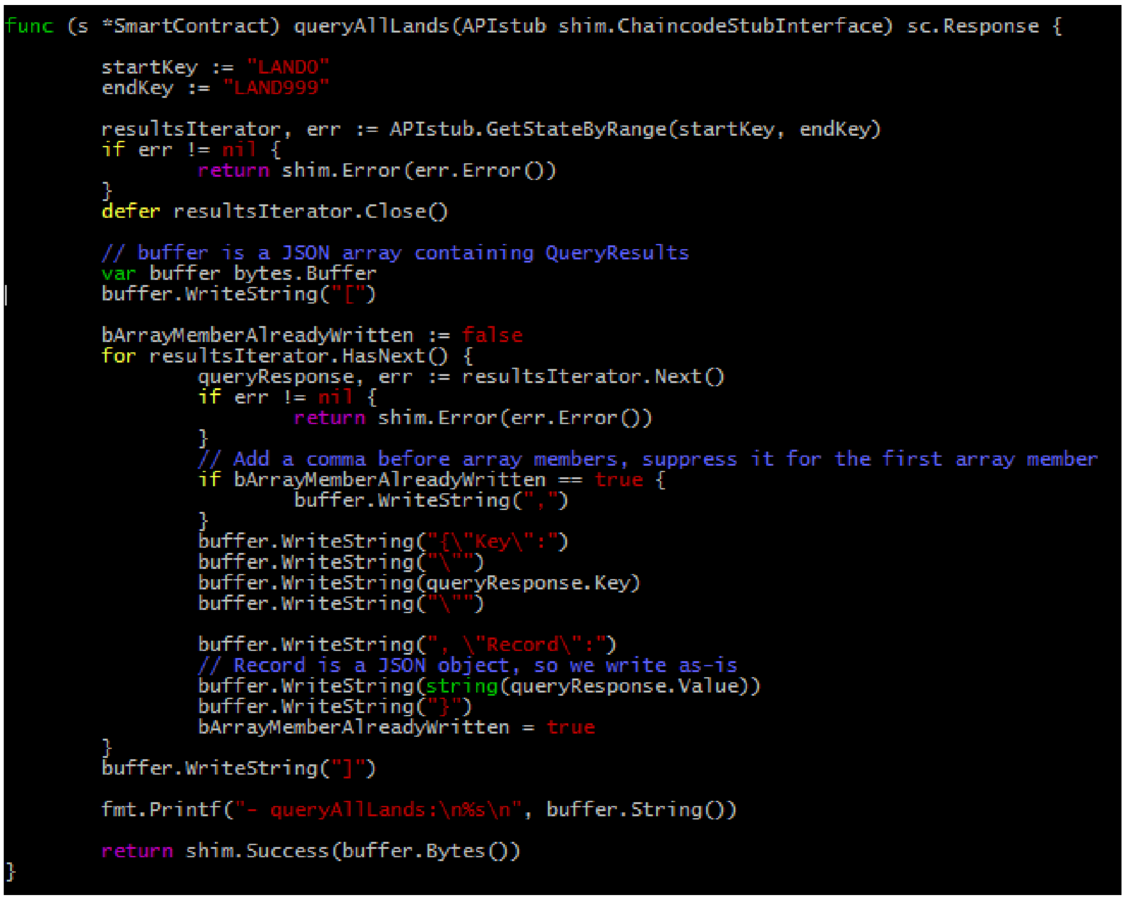
Implementation of chaincode functions

Implementation of queryLand chaincode query



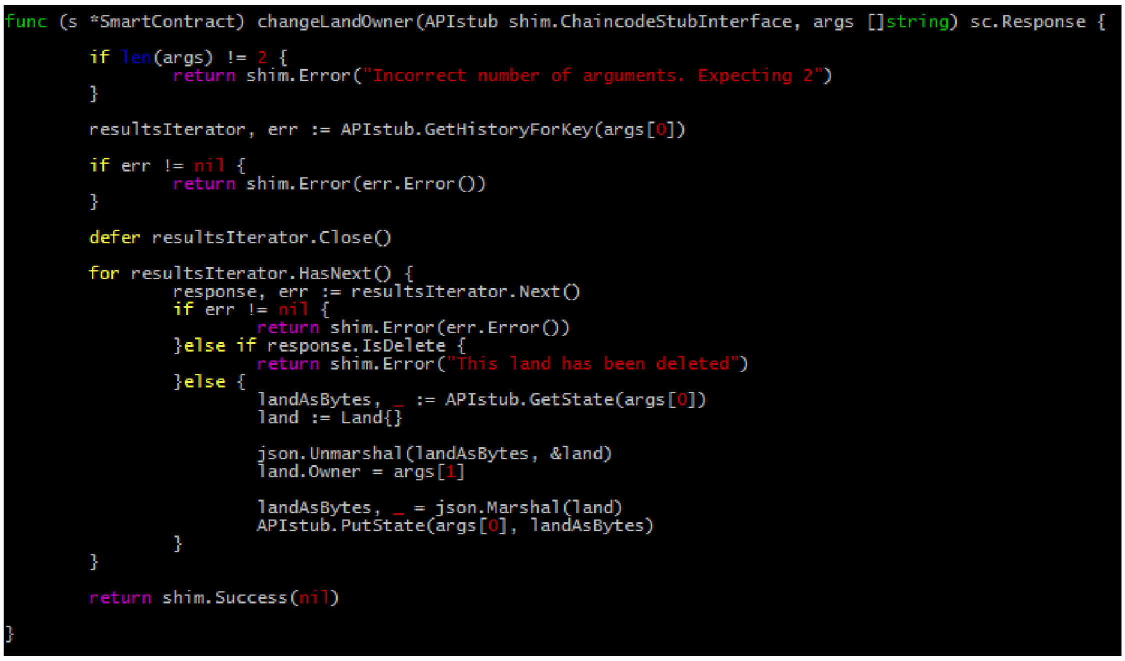
```
func (s *SmartContract) queryLand(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

        if len(args) != 1 {
                return shim.Error("Incorrect number of arguments. Expecting 1")
        }

        resultsIterator, err := APIstub.GetHistoryForKey(args[0])

        if err != nil {
                return shim.Error(err.Error())
        }

        defer resultsIterator.Close()

        for resultsIterator.HasNext() {
                response, err := resultsIterator.Next()
                if err != nil {
                        return shim.Error(err.Error())
                }else if response.IsDelete {
                        return shim.Error("This land has been deleted")
                }
        }
        landAsBytes, _ := APIstub.GetState(args[0])
        return shim.Success(landAsBytes)
}
```

Implementation of queryAllLands chaincode query



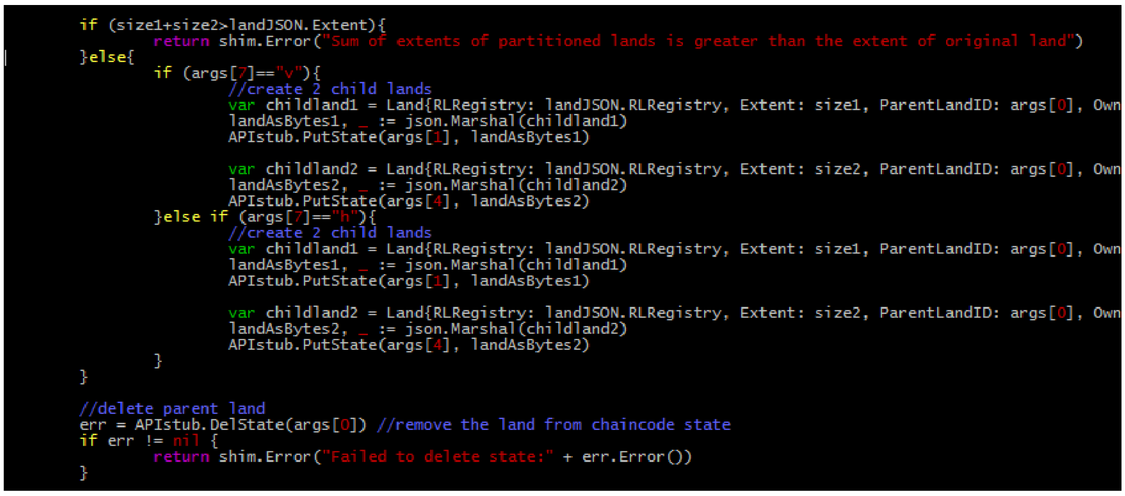
```go
func (s *SmartContract) queryAllLands(APIstub shim.ChaincodeStubInterface) sc.Response {

	startKey := "LAND0"
	endKey := "LAND999"

	resultsIterator, err := APIstub.GetStateByRange(startKey, endKey)
	if err != nil {
		return shim.Error(err.Error())
	}
	defer resultsIterator.Close()

	// buffer is a JSON array containing QueryResults
	var buffer bytes.Buffer
	buffer.WriteString("[")

	bArrayMemberAlreadyWritten := false
	for resultsIterator.HasNext() {
		queryResponse, err := resultsIterator.Next()
		if err != nil {
			return shim.Error(err.Error())
		}
		// Add a comma before array members, suppress it for the first array member
		if bArrayMemberAlreadyWritten == true {
			buffer.WriteString(",")
		}
		buffer.WriteString("{\"Key\":")
		buffer.WriteString("\"")
		buffer.WriteString(queryResponse.Key)
		buffer.WriteString("\"")

		buffer.WriteString(", \"Record\":")
		// Record is a JSON object, so we write as-is
		buffer.WriteString(string(queryResponse.Value))
		buffer.WriteString("}")
		bArrayMemberAlreadyWritten = true
	}
	buffer.WriteString("]")

	fmt.Printf("- queryAllLands:\n%s\n", buffer.String())

	return shim.Success(buffer.Bytes())
}
```

<u>Implementation of changeLandOwner chaincode invocation transaction</u>



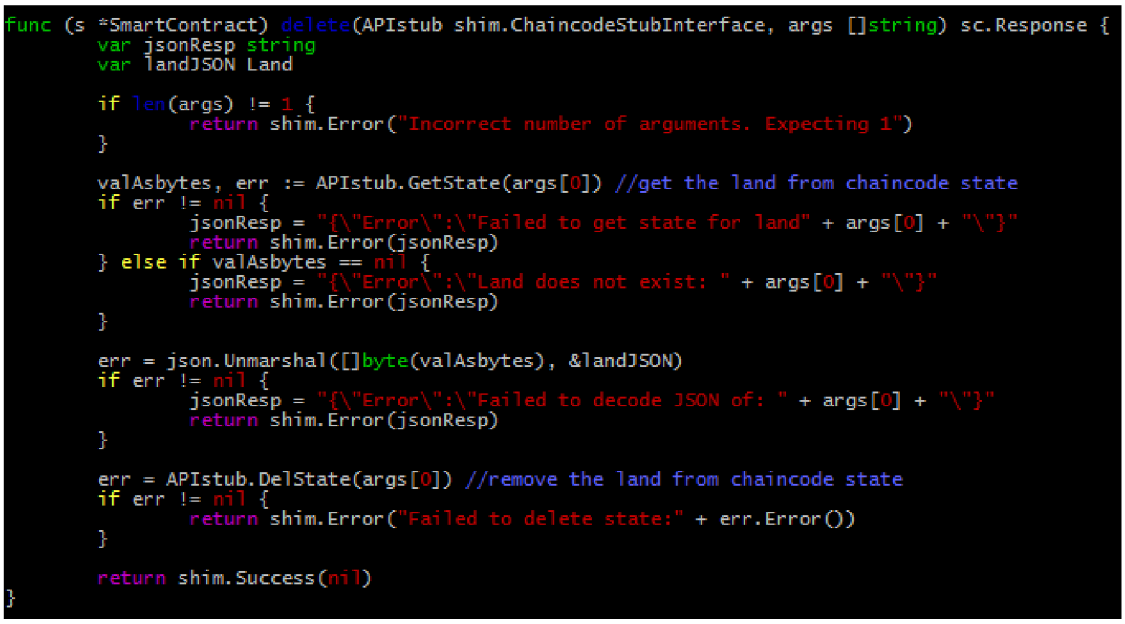
```go
func (s *SmartContract) changeLandOwner(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

	if len(args) != 2 {
		return shim.Error("Incorrect number of arguments. Expecting 2")
	}

	resultsIterator, err := APIstub.GetHistoryForKey(args[0])

	if err != nil {
		return shim.Error(err.Error())
	}

	defer resultsIterator.Close()

	for resultsIterator.HasNext() {
		response, err := resultsIterator.Next()
		if err != nil {
			return shim.Error(err.Error())
		}else if response.IsDelete {
			return shim.Error("This land has been deleted")
		}else {
			landAsBytes, _ := APIstub.GetState(args[0])
			land := Land{}

			json.Unmarshal(landAsBytes, &land)
			land.Owner = args[1]

			landAsBytes, _ = json.Marshal(land)
			APIstub.PutState(args[0], landAsBytes)
		}
	}

	return shim.Success(nil)

}
```

Part of Implementation of forkLand chaincode invocation transaction depicting the invocation of createLand and deleteLand invocations as well as checking for overlapping boundaries and extent consistency



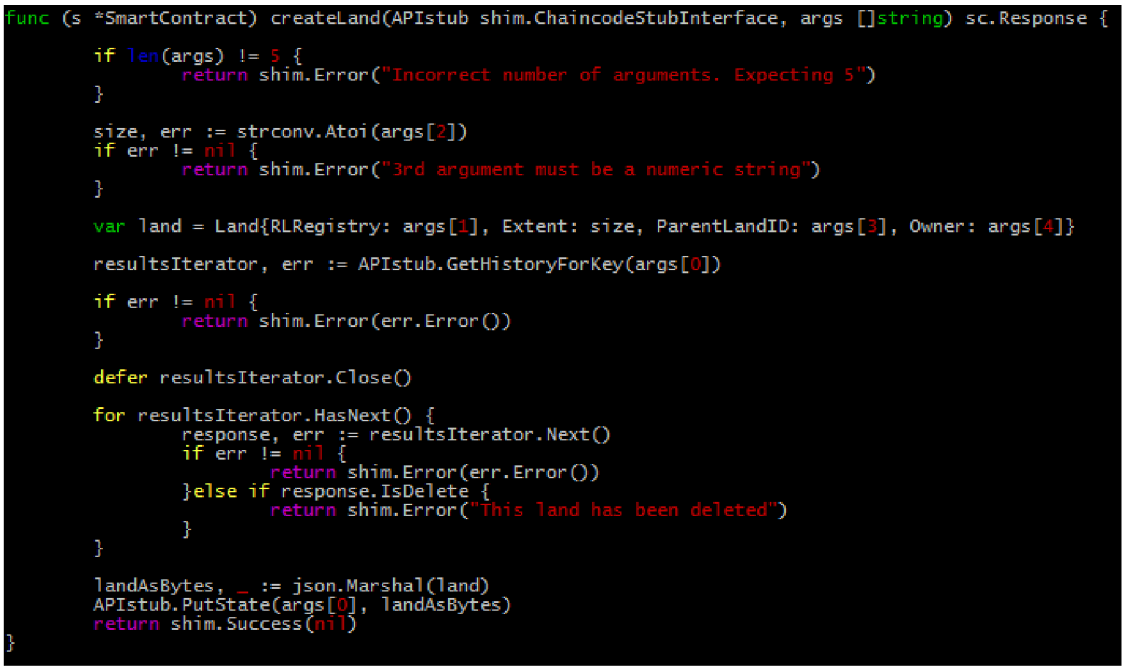
```
        if (size1+size2>landJSON.Extent){
                return shim.Error("Sum of extents of partitioned lands is greater than the extent of original land")
        }else{
                if (args[7]=="v"){
                        //create 2 child lands
                        var childland1 = Land{RLRegistry: landJSON.RLRegistry, Extent: size1, ParentLandID: args[0], Own
                        landAsBytes1, _ := json.Marshal(childland1)
                        APIstub.PutState(args[1], landAsBytes1)

                        var childland2 = Land{RLRegistry: landJSON.RLRegistry, Extent: size2, ParentLandID: args[0], Own
                        landAsBytes2, _ := json.Marshal(childland2)
                        APIstub.PutState(args[4], landAsBytes2)
                }else if (args[7]=="h"){
                        //create 2 child lands
                        var childland1 = Land{RLRegistry: landJSON.RLRegistry, Extent: size1, ParentLandID: args[0], Own
                        landAsBytes1, _ := json.Marshal(childland1)
                        APIstub.PutState(args[1], landAsBytes1)

                        var childland2 = Land{RLRegistry: landJSON.RLRegistry, Extent: size2, ParentLandID: args[0], Own
                        landAsBytes2, _ := json.Marshal(childland2)
                        APIstub.PutState(args[4], landAsBytes2)
                }
        }

        //delete parent land
        err = APIstub.DelState(args[0]) //remove the land from chaincode state
        if err != nil {
                return shim.Error("Failed to delete state:" + err.Error())
        }
```

Implementation of deleteLand chaincode invocation transaction



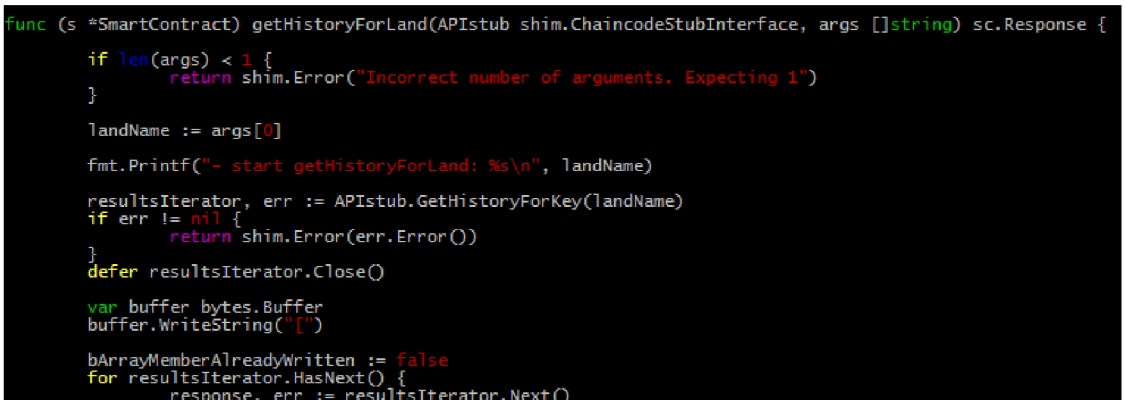
```
func (s *SmartContract) delete(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {
        var jsonResp string
        var landJSON Land

        if len(args) != 1 {
                return shim.Error("Incorrect number of arguments. Expecting 1")
        }

        valAsbytes, err := APIstub.GetState(args[0]) //get the land from chaincode state
        if err != nil {
                jsonResp = "{\"Error\":\"Failed to get state for land" + args[0] + "\"}"
                return shim.Error(jsonResp)
        } else if valAsbytes == nil {
                jsonResp = "{\"Error\":\"Land does not exist: " + args[0] + "\"}"
                return shim.Error(jsonResp)
        }

        err = json.Unmarshal([]byte(valAsbytes), &landJSON)
        if err != nil {
                jsonResp = "{\"Error\":\"Failed to decode JSON of: " + args[0] + "\"}"
                return shim.Error(jsonResp)
        }

        err = APIstub.DelState(args[0]) //remove the land from chaincode state
        if err != nil {
                return shim.Error("Failed to delete state:" + err.Error())
        }

        return shim.Success(nil)
}
```

Implementation of createLand chaincode invocation transaction



```
func (s *SmartContract) createLand(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

        if len(args) != 5 {
                return shim.Error("Incorrect number of arguments. Expecting 5")
        }

        size, err := strconv.Atoi(args[2])
        if err != nil {
                return shim.Error("3rd argument must be a numeric string")
        }

        var land = Land{RLRegistry: args[1], Extent: size, ParentLandID: args[3], Owner: args[4]}

        resultsIterator, err := APIstub.GetHistoryForKey(args[0])

        if err != nil {
                return shim.Error(err.Error())
        }

        defer resultsIterator.Close()

        for resultsIterator.HasNext() {
                response, err := resultsIterator.Next()
                if err != nil {
                        return shim.Error(err.Error())
                }else if response.IsDelete {
                        return shim.Error("This land has been deleted")
                }
        }

        landAsBytes, _ := json.Marshal(land)
        APIstub.PutState(args[0], landAsBytes)
        return shim.Success(nil)
}
```

Part of the code demonstrating the usage of GetHistoryForKey chaincode API function in getHistoryForLand chaincode query



```
func (s *SmartContract) getHistoryForLand(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

        if len(args) < 1 {
                return shim.Error("Incorrect number of arguments. Expecting 1")
        }

        landName := args[0]

        fmt.Printf("- start getHistoryForLand: %s\n", landName)

        resultsIterator, err := APIstub.GetHistoryForKey(landName)
        if err != nil {
                return shim.Error(err.Error())
        }
        defer resultsIterator.Close()

        var buffer bytes.Buffer
        buffer.WriteString("[")

        bArrayMemberAlreadyWritten := false
        for resultsIterator.HasNext() {
                response, err := resultsIterator.Next()
```