Санкт-Петербургский государственный университет

Программная инженерия

Скаредов Сергей Антонович

Реализация протокола электронного голосования на блокчейне Hyperledger Fabric

Выпускная квалификационная работа

Научный руководитель:
доцент кафедры системного программирования, к.ф.-м.н. К. Ю. Романовский

Консультанты:
ст. преп. кафедры системного программирования Я. А. Кириленко
ведущий разработчик «DSX Technologies Russia» Ф. П. Долголев

Рецензент:
технический директор «DSX Technologies Russia», к.ф.-м.н. А. Н. Иванов

Санкт-Петербург
2020

SAINT PETERSBURG STATE UNIVERSITY

Software Engeneering

Sergey Skaredov

Implementation of electronic voting protocol using Hyperledger Fabric

Bachelor's Thesis

Scientific supervisor:
associate professor, PhD Konstantin Romanovsky

Consultants:
senior lecturer Iakov Kirilenko
lead developer at «DSX Technologies Russia» Philipp Dolgolev

Reviewer:
chief technology officer at «DSX Technologies Russia», PhD Alexander Ivanov

Saint Petersburg
2020

Оглавление

Введение

Важные решения в крупных компаниях принимаются на собраниях акционеров. Процесс голосования обычно заключается в формировании, заполнении и последующем подсчёте бюллетеней. Это ресурсоёмкий и трудозатратный процесс, неточности в котором естественны в силу человеческого фактора. Ручная обработка бумажных носителей подверженна ошибкам и возможной фальсификации. В отличие от традиционного, в акционерном голосовании количество голосов участников соответствует количеству акций, которыми они владеют. Голосующий в праве распределить имеющиеся у него голоса между вариантами ответов в соответствии со своими убеждениями.

Электронное голосование — способ упростить процесс коллективного принятия решения. Удалённое участие в бумажном голосовании возможно, но оно требует доставки бюллетеней до конечных голосующих и последующей их обратной отправки. После направления заполненного бюллетеня в организацию, проводящую голосование, участник не имеет возможности удостовериться в том, что его голос дошёл до получателя, не был подделан или вообще был принят. Электронное голосование позволит избавить участников от необходимости личного присутствия на собраниях и сэкономить большое количество рабочего времени по сравнению с затратами на организацию, проведение и участие в очном бумажном голосовании. Однако, как и для бумажного голосования, необходимо выполнение требований защиты от фальсификаций или игнорирования голосов участников, а также обеспечение приватности — никто кроме самого участника не должен узнать о его фактическом волеизъявлении. Это необходимо для исключения какого-либо контроля или давления на голосующего, защиты его от преследования.

Для электронного голосования, как и для задач, в которых необходимо коллегиальное принятие решений, публичная доступность данных и неизменяемость истории операций, всё более популярным

решением становится технология блокчейн — децентрализованная распределённая база данных [11]. Такие страны как Россия [18], Турция [2], Эстония [14] уже применяют или планируют использовать блокчейн для проведения голосований на государственном уровне, что говорит об актуальности подобного решения. Реализация различных процессов с применением данной технологии возможна с помощью создания смарт-контрактов — программ, исполняющихся в блокчейн-сети и изменяющих состояние базы данных в соответствии с необходимыми правилами.

Важным аспектом тайного электронного голосования является возможность участников удостовериться в правильности подсчёта голосов, а также сокрытие фактического распределения голосов от других участников. Чтобы удовлетворять данным требованиям, система голосования может быть реализована с применением криптографических алгоритмов zero-knowledge-proof (доказательство с нулевым разглашением) [16]. Такие алгоритмы активно исследуются и применяются совместно с технологией блокчейн, например, в реализациях криптовалют Zcash [13] и Monero [12].

Компания DSX Technologies[1] совместно с компанией Accenture[2] разработали и запатентовали алгоритм тайного электронного акционерного голосования *E-voting* [3, 4]. Текущая Proof-of-Concept реализация представляет из себя модульную систему, которая обрабатывает запросы пользователей, проверяет корректность распределения голосов в бюллетенях и использует блокчейн-платформу Hyperledger Fabric в качестве способа хранения данных, обмена информацией между участниками в виде сообщений и публикации конечных результатов голосования.

Особенность данного подхода заключается в том, что он имеет слабую зависимость от конкретной реализации технологии блокчейн. Это даёт возможность при необходимости использовать блокчейн-платформу, отвечающую каким-либо специфичным требованиям,

[1] `https://dsxt.uk/`

[2] `https://www.accenture.com/`

минимизируя затраты на модификацию существующего решения. Однако в связи с этим на модули системы накладываются обязательства по валидации входящих сообщений, локальному выстраиванию актуального состояния процесса голосования, а также его синхронизации между собой. Так как блокчейн в данном случае является лишь средством хранения и обмена сообщениями, а проверки корректности сообщений при записи в него отсутствуют, такой подход уязвим к злоумышленному засорению базы данных большим количеством запросов на запись любого рода информации. То есть даже невалидные сообщения записываются в блокчейн, расходуют ограниченные ресурсы памяти системы, но не продвигают процесс голосования.

Использование смарт-контрактов [15] для реализации логики валидации E-voting протокола позволит обрабатывать сообщения участников ещё до записи в базу данных. Это сократит размеры хранимой информации в блокчейне и снизит нагрузку на внешние модули системы. При помощи смарт-контрактов возможно явное отслеживание текущего состояния голосования на уровне блокчейна, что избавит модули системы от необходимости локального дублирования данных и постоянной поддержки и синхронизации собственной версии общего состояния системы.

1. Постановка задачи

Целью данной работы является реализация протокола тайного электронного акционерного голосования на блокчейне Hyperledger Fabric. Для достижения цели были сформулированы следующие задачи:

- Провести обзор предметной области
  - Протокол E-voting
  - Hyperledger Fabric
- Спроектировать архитектуру системы
  - Провести анализ существующего решения
  - Разработать новый подход
  - Провести сравнение подходов
- Реализовать новый подход
  - Реализовать систему смарт-контрактов
  - Провести рефакторинг исходной реализации
- Провести тестирование системы

2. Обзор предметной области

Обзор предметной области включает в себя описание протокола E-voting и блокчен-платформы Hyperledger Fabric. Для реализации протокола необходимо рассмотреть структуру участников процесса, порядок их взаимодействия, а также подход к анонимизации и проверке корректности голосов. В рамках поставленной задачи о реализации протокола с использованием Hyperledger Fabric необходимо рассмотреть процесс работы сети данной блокчейн-платформы, её составные части и предоставляемые платформой возможности по реализации собственной бизнес-логики.

2.1. E-voting

E-voting — протокол тайного электронного акционерного голосования на основе технологии блокчейн. Разработан и запатентован компаниями DSX Technologies и Accenture. Данный подход позволяет организовывать голосования различной структуры и масштаба [3]. Надёжное хранение данных, сокрытие фактических значений голосов участников и в то же время возможность проверки их корректности обеспечиваются неизменяемостью истории транзакций в распределённом реестре и применением криптографических алгоритмов [4].

Голоса участников в системе представлены объектами **Coin** — абстрактными монетами, которые обладают следующими свойствами:

- Номинал монеты выражает некоторое количество голосов
- Фактическое значение количества голосов зашифровано
- Обладают свойством аддитивности
- Создаются в рамках конкретного голосования и не могут быть использованы в других голосованиях

- Сообщение, определяющее распределение голосов пользователя содержит криптографическое доказательство неотрицательности номинала каждой монеты [8]

Так как фактическое распределение голосов зашифровано, доказательство неотрицательности номиналов монет необходимо, чтобы любой участник голосования имел возможность проверить на корректность волеизъявление любого другого участника, а также корректность финального количества голосов, отданных любому ответу любого вопроса в голосовании. Дело в том, что сумма номиналов распределённых участником монет должна быть эквивалентна номиналу исходной монеты, которой этот участник владел. Нечестный участник с исходной монетой номиналом в 10 акций может попытаться «создать голоса из воздуха», распределив монеты с номиналами 15 и -5 на два ответа. Фактические значения зашифрованы, и никто кроме самого злоумышленника не знает настоящих номиналов, а сумма распределённых монет равна исходной монете. Но при этом доказательство неотрицательности номиналов монет позволит системе уличить такого участника в мошенничестве и посчитать его сообщение с распределением невалидным.

Для проведения голосования протоколом определены следующие сущности-участники, представляемые электронными системами, и их ответственности:

- **Organizer**
  - Публикация описания голосования в блокчейн
  - Первичная эмиссия монет
  - Распределение монет дочерним **Intermediary**
- **Intermediary**
  - Запрос монет у родительского **Intermediary** (опционально)
  - Распределение монет дочерним **Intermediary** и **Voters**

- **Voter**
  - Конечный участник голосования
  - Запрос монет у родительского **Intermediary** (опционально)
  - Публикация голосов в блокчейн как напрямую, так и через родительского **Intermediary**
- **Voting Service**
  - Подсчёт голосов
  - Публикация результатов голосования

Процесс голосования (рис. 1) состоит из 3 последовательных фаз:

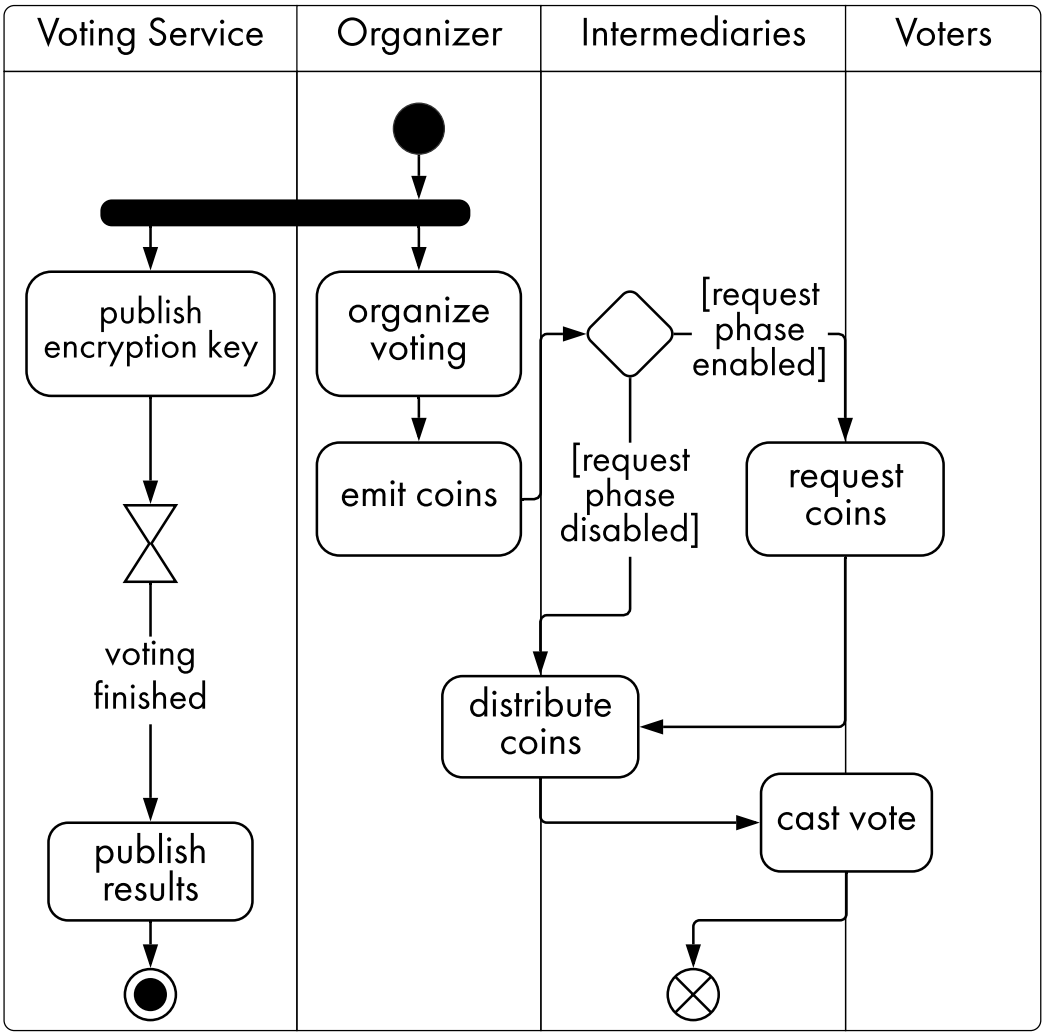

Рис. 1: Процесс голосования

2.1.1. Инициализация голосования

Первый этап заключается в подготовке всех необходимых данных для проведения голосования.

1.2 **Organizer** публикует сообщение о начале нового голосования

1.3 **Voting Service** публикует сообщение с публичным ключём для шифрования конечных голосов

- Соответствующим приватным ключём для расшифровки владеет только **Voting Service**
- Это необходимо для подсчёта финального количества голосов на всех ответах голосования

2.1.2. Распределение монет

Распространение монет для голосования происходит от родительских сущностей к дочерним.

2.1 **Organizer** публикует сообщение о начальной эмиссии монет

- Монеты создаются для каждого вопроса отдельно, так как для разных вопросов может быть различное суммарное количество голосов
- Созданные монеты попадают на кошелёк, которым владеет **Organizer**

2.2 Дочерние сущности **Intermediaries** и **Voters** публикуют сообщения о запросе монет у родительских сущностей (опционально)

- Сообщение содержит адрес для получения монет и необходимое количество голосов по вопросам
- Запрашиваемое количество голосов шифруется открытым ключём родительской сущности

2.3 Родительская сущность **Intermediary** или **Organizer** отправляет монеты с необходимыми номиналами на адрес дочерней сущности

- Перевод содержит информацию о распределении некоторого количества голосов в виде монеты для каждого вопроса
- Перевод содержит доказательство неотрицательности для каждой переводимой монеты

2.1.3. Голосование и публикация результатов

По итогам распределения каждый голосующий получает на свой адрес монеты с суммарным номиналом, соответствующим количеству его акций. Затем начинается этап непосредственного голосования.

3.1 Каждый **Voter** голосует путём отправки сообщения-бюллетеня

- Сообщение содержит информацию о распределении всех голосов учистника в виде монет для каждого ответа
- Сообщение содержит доказательство неотрицательности для каждой переводимой монеты
- Конечные значения голосов шифруются публичным ключём **Voting Service**

3.2 По истечению некоторого периода **Voting Service** расшифровывает, подсчитывает и публикует финальное распределение голосов по ответам

Протокол направлен на корпоративное применение собраниями акционеров и децентрализация, свойственная криптовалютам на основе блокчейна, не является необходимостью. Блокчейн в данном случае выступает средой исполнения некоторой общей логики и механизмом распределённого и надёжного хранения данных с предоставлением возможности аудита происходящих на нём процессов. Если несколько компаний участвуют в организации и проведении голосования, то каждая из них будет привносить свой вклад в обработку и валидацию

транзакций в соответствии с консенсусом, однако наличие доверенной сущности, отвечающей за публикацию результатов голосования в открытом виде, продиктовано самим протоколом.

2.2. Hyperledger Fabric

В широком смысле, блокчейн — распределённая база данных, поддерживающаяся сетью узлов-участников, которая позволяет надёжно хранить историю всех операций в сети и обеспечивает строго консистентное изменение данных [11]. Каждый узел хранит и обновляет копию реестра, применяя транзакции, которые были приняты в результате исполнения протокола консенсуса. Транзакции объединяются в блоки, каждый из которых содержит в себе хеш предыдущего блока. Составленная таким образом цепочка блоков защищает исторические данные от изменения нечестными участниками сети.

Hyperledger Fabric — open-source платформа на основе DLT[3], один из продуктов проекта Hyperledger[4], разрабатываемого Linux Foundation. Особенностью данной платформы является направленность на корпоративное применение и создание бизнес-решений консорциумом организаций, вследствие чего функциональность платформы позволяет идентифицировать и наделять различными уровнями доступа отдельных участников сети [7].

Организации, принявшие решение о коммуникации посредством Hyperledger Fabric, способны сконфигурировать сеть под свои нужды, благодаря модульной архитектуре платформы и поддержке смарт-контрактов.

2.2.1. Channel и Ledger

Channel — подмножество сети, ограничивающее область видимости происходящих в нём транзакций и хранящихся в нём данных. На

[3]Distributed Ledger Technology — технология распределённого реестра

[4]`https://www.hyperledger.org/`

интуитивном уровне можно считать, что канал — это приватный блокчейн, доступ к которому предоставляется лишь ограниченному кругу участников.

Ledger — реестр, содержащий все данные в канале. Одному каналу соответствует один реестр. Состоит из двух частей:

- **Blockchain** — неизменяемая история всех транзакций, объединённых в цепочку блоков
- **World State** — база данных текущего состояния системы. Представляет последнее записанное значение для каждого сохранённого состояния в истории транзакций. Читая и изменяя данные, смарт-контракт основывается именно на **World State**

2.2.2. Application и Chaincode

Chaincode — набор смарт-контрактов — программ, реализующих бизнес-логику и исполняемых на узлах сети. Определяет правила изменения состояния сети и предоставляет методы смарт-контрактов для этого. При необходимости записи данных создаёт транзакцию. Запуск *Chaincode* производится установкой его на один или несколько узлов сети и его последующей инициализацией для некоторого канала.

На данный момент Hyperledger Fabric поддерживает несколько языков программирования общего назначения для написания смарт-контрактов — Go[5], JavaScript[6] (Node.js) и Java[7]. Непосредственное исполнение *Chaincode* программ происходит внутри Docker-контейнеров.

Application — пользовательское *off-chain* приложение, то есть находящееся за пределами сети. Отправляет узлам сети запросы на исполнение методов определённого смарт-контракта. Чтение или изменение данных в реестре вызываемыми методами инициируют транзакции. Для взаимодействия с сетью могут использоваться

[5] `https://github.com/hyperledger/fabric-chaincode-go`
[6] `https://github.com/hyperledger/fabric-chaincode-node`
[7] `https://github.com/hyperledger/fabric-chaincode-java`

различные SDK (Go[8], Java[9,10] и другие)

2.2.3. Peer и Ordering Service

Peer — узел сети. Относится к определённой организации и представляет из себя логическую единицу, так как запускается внутри Docker-контейнера и несколько узлов могут существовать на одной физической машине. Узел, присоединившийся к каналу, хранит копию соответствующего реестра и способен создавать Docker-контейнеры с запущенными в них *Chaincode* программами для обработки пользовательских запросов. Получив от приложения запрос на исполнение метода смарт-контракта, узел проверяет, имеет ли данный пользователь на это право. Если всё в порядке, узел запускает исполнение смарт-контракта с переданными ему параметрами. Результат исполнения возвращается приложению для валидации. После успешной проверки результата, приложение формирует и отправляет в **Ordering Service** транзакцию, содержащую все необходимые изменения текущего состояния. Локальная копия реестра узла обновляется после того, как он получит от **Ordering Service** новый блок с упорядоченными транзакциями и проведёт его валидацию.

Ordering Service — набор узлов, ответственных за упорядочивание транзакций и формирование нового блока для каждого канала в сети. Транзакции присылаются клиенсткими приложениями по результатам исполнений смарт-контрактов. Существует возможность конфигурации консенсуса, что ускоряет процесс разработки на платформе. Сформированный блок с хронологически упорядоченными транзакциями распространяется между peer-узлами сети, которые в свою очередь валидируют его и принимают в свою историю.

[8] `https://github.com/hyperledger/fabric-sdk-go`

[9] `https://github.com/hyperledger/fabric-sdk-java`

[10] `https://github.com/hyperledger/fabric-gateway-java`

3. Разработка архитектуры

Архитектура системы разделена на четыре уровня. Уровень *UI* реализует мобильные и веб-приложения для участников голосования. *DLT* уровень представляет собой развёрнутую сеть Hyperledger Fabric, на основе которой происходит процесс голосования. Исходную реализацию системы можно охарактеризовать как *Application-based* подход, так как в данном случае *Application* уровень содержит всю бизнес-логику системы. *Chaincode* уровень в свою очередь предоставляет API для чтения и записи в реестр, осуществляя взаимодействие между *DLT* и *Application* уровнями (рис. 2).

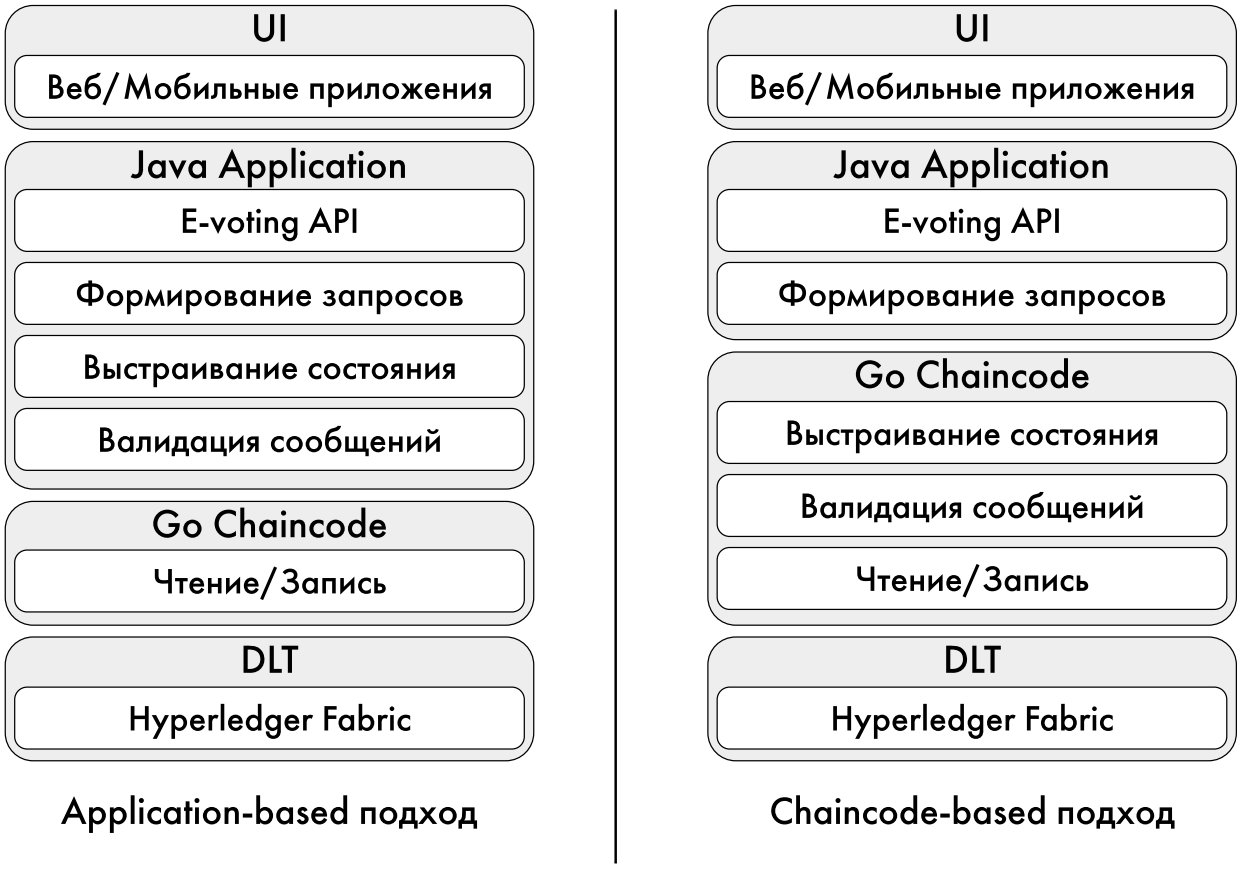

Рис. 2: Подходы реализации E-voting протокола

Перенос логики валидации сообщений и ответственности за поддержание текущего состояния системы на *Chaincode* уровень, используя возможности, предоставляемые платформой Hyperledger Fabric, позволит защитить *DLT* уровень от возможного засорения

некорректными данными, снять с модулей *Application* уровня обязанность по выстраиваиванию локального состояния голосования, а также необходимости синхронизации этих состояний между собой. Такую архитектуру системы можно охарактеризовать как *Chaincode-based* подход.

3.1. Application-based

В исходной реализации (рис. 3) *Application* уровень представлен в виде набора самостоятельных Java-модулей, соответствующих сущностям участников. Основная логика системы вынесена за пределы блокчейн-сети: конструирование исходящих, валидация и обработка входящих сообщений, создание кошельков, распределение монет, построение доказательств и проверка их неотрицательности происходят *off-chain* — на *Application* уровне.

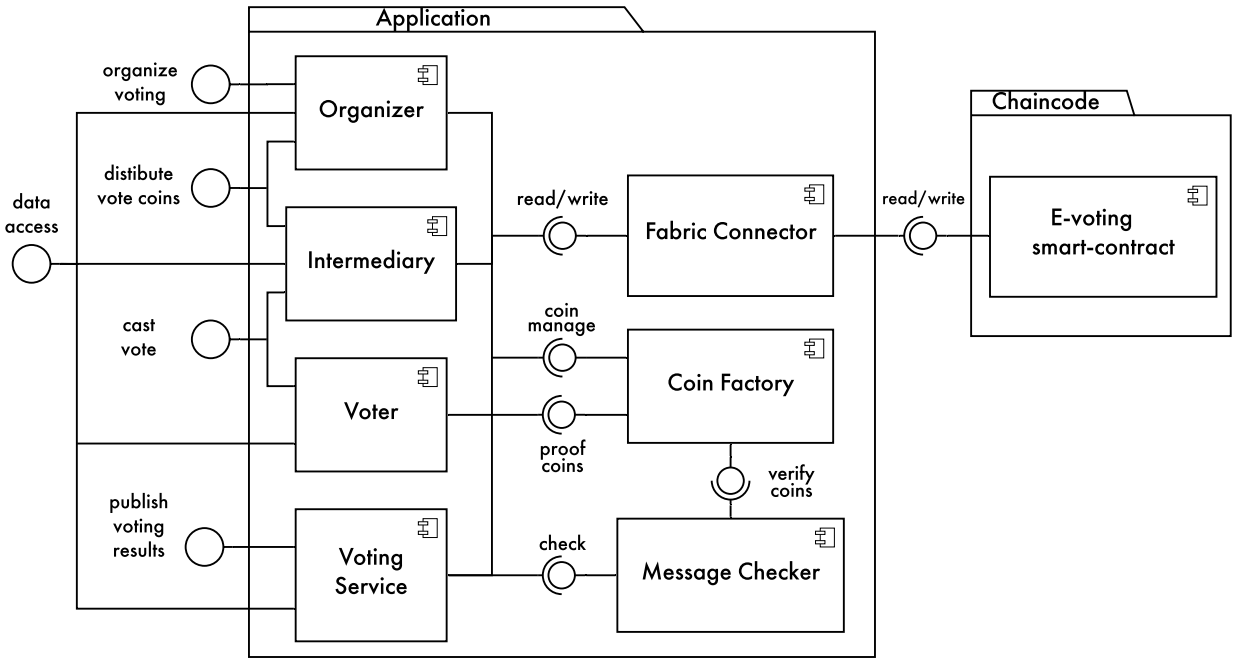

Рис. 3: Архитектура *Application-based* подхода

Модули **Organizer**, **Intermediary**, **Voter** и **Voting Service** реализуют соответствующие пользовательские API и используют для обработки запросов внутренние компоненты:

- **Coin Factory** — реализация логики по работе с монетами

  - Создание монет
  - Арифметические операции над монетами
  - Создание и проверка криптографических доказательств

- **Message Checker** — валидация обнаруженных сообщений
  - Проверка наличия всех необходимых полей
  - Соответствие сообщения текущему голосованию
  - Авторизация отправителя
  - Верификация доказательств неотрицательности монет при помощи **Coin Factory**

- **Fabric Connector** — взаимодействие с *Chaincode* уровнем
  - Сериализация переданных сообщений и их отправка в метод *write* смарт-контракта
  - Периодический опрос смарт-контракта с помощью метода *read* на предмет новых записей в реестре и передача десериализованных сообщений своему клиенту для обработки

Chaincode уровень представляет из себя смарт-контракт для взаимодействия с блокчейн-сетью, реализован на языке Go и предоставляет методы чтения и записи сообщений в реестр. Платформа Hyperledger Fabric использует в качестве реестра *key-value* базу данных *LevelDB*, которая хранит записи в формате <*ключ*, *значение*>. Ключём в данном подходе выступает строка, состоящая из временной метки[11] создания записи и хэша записываемого сообщения. Значением является любое сериализованное в формат *json* сообщение, которое может быть опубликовано участниками в соответствии с протоколом. Однако, проверка корректности записываемого значения на уровне смарт-контракта не производится. Функция чтения принимает на вход

[11]Временная метка — текущее время в милисекундах в формате Unix Timestamp

временную метку и возвращает все сообщения, записанные не раньше переданного параметра времени.

Таким образом, исходная Proof-of-Concept реализация использует *DLT* уровень как средство обмена сообщениями между участниками голосования.

3.2. Chaincode-based

Разрабатываемый в рамках данной работы *Chaincode-based* подход подразумевает реализацию части бизнес-логики системы на *Chaincode* уровне и соответствующий рефакторинг *Application* уровня (рис. 4).

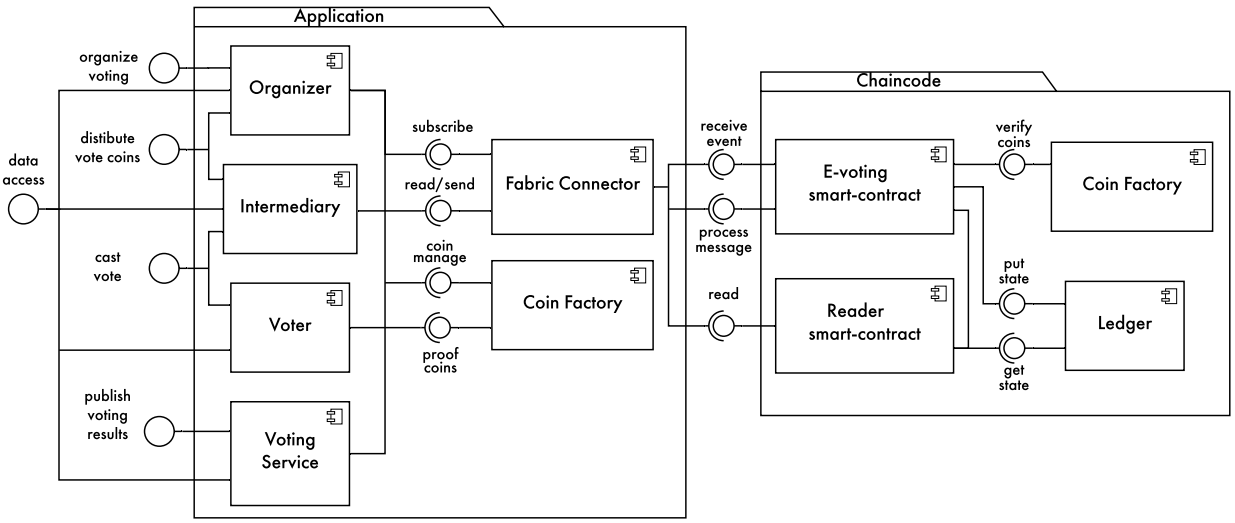

Рис. 4: Архитектура *Chaincode-based* подхода

Уровень *Chaincode* содержит четыре компонента:

- **Ledger** — взаимодействие с реестром
  - Сериализация и запись объектов
  - Чтение и десериализация строкового представления объектов
- **Coin Factory** — реализация логики по работе с монетами
  - Арифметические операции над монетами
  - Проверка криптографических доказательств

- **Reader smart-contract** — API для чтения записей из реестра
- **E-voting smart-contract** — API протокола голосования
  - Каждый метод соответствует определённому типу сообщения, которое могут отправить участники
  - Реализует валидацию сообщений и изменение состояния голосования с последующей записью сообщения в реестр
  - Успешная обработка запроса сопровождается публикацией соответствующего события, которое оповещает всех участников голосования о новой транзакции в системе

Изменения *Application* уровня затрагивают реализации внутренних модулей. Повторяющийся ранее опрос смарт-контракта на предмет новых записей модулем **Fabric Connector** заменяется подпиской клиента на оповещения о событиях, которые публикуются **E-voting** смарт-контрактом, и добавлением в API модуля метода *read*, который принимает идентификатор сообщения, передаёт запрос в смарт-контракт **Reader** и возвращает десериализованный объект соответствующей записи из реестра. Каждый тип сообщения передаётся модулем в соответствующий метод смарт-контракта **E-voting** для валидации и последующего сохранения в реестр. Функциональность модуля **Messages Checker** и используемая им логика верификации криптографических доказательств модуля **Coin Factory** теперь реализованы на *Chaincode* уровне, в связи с чем уровень *Application* больше в них не нуждается.

3.3. Сравнение подходов

Преимуществом *Application-based* подхода является слабая зависимость от конкретной реализации *DLT*. Эта особенность позволяет использовать систему голосования на различных платформах, которые могут удовлетворять каким-либо специфичным требованиям. Для этого достаточно реализовать взаимодействие между Java-модулями и

необходимой платформой. Однако в данном подходе на каждый модуль *Application* уровня накладываются обязательства по локальному выстраиванию и поддержке текущего состояния голосования и балансов на кошельках участников при помощи периодического опроса блокчейна на предмет новых записей.

Переход от «сырого» представления данных в реестре к оперированию сущностями сообщений, кошельков и монет в качестве программных объектов на *Chaincode* уровне позволит явно выстраивать текущее состояние голосования средствами Hyperledger Fabric и отказаться от локального дублирования данных на *Application* уровне. Появится возможность запрашивать из реестра только необходимые данные, что снизит транспортную нагрузку на сеть. Спроектированный таким образом единственный источник истины станет точкой синхронизации общего состояния системы и снимет данное обязательство с *off-chain* модулей.

Реализация логики валидации сообщений на *Chaincode* уровне изменит последовательность выполнения пользовательских запросов. В *Application-based* подходе любое сообщение проходило четыре этапа:

1. Сообщение формируется и отправляется *off-chain* модулем
2. Сообщение записывается в реестр смарт-контрактом
3. Сообщение читается другими *off-chain* модулями
4. Сообщение валидируется *off-chain* модулями и меняет состояние системы в случае корректности

Chaincode-based подход в свою очередь сокращает данный процесс до трёх этапов:

1. Сообщение формируется и отправляется *off-chain* модулем
2. Сообщение валидируется смарт-контрактом и, в случае корректности, меняет состояние системы
3. Сообщение рассылается другим *off-chain* модулям

Таким образом только корректные сообщения, сформированные на *Application* уровне, будут попадать в реестр и тем самым продвигать процесс голосования, обновляя балансы кошельков пользователей и распределение голосов по ответам. Это сократит количество хранимых данных, защитив систему от злоумышленного заполнения блокчейна невалидными сообщениями. Результаты анализа подходов и их сравнения отражены в таблице 1.

Критерий\Подход	*Application-based*	*Chaincode-based*
Модель данных	«Сырые» строки	Программные объекты
Состояние системы	Локальное	Общее
Хранимые данные	Все сообщения	Корректные сообщения
Синхронизация	Индивидуальная	Автоматическая
DLT-зависимость	Слабая	Сильная

Таблица 1: Сравнение подходов

4. Реализация chaincode-based подхода

Разработка системы производилась с использованием языков Java и Go, так как уже имеющаяся кодовая база *Application* уровня на Java не подлежала кардинальным изменениям. Go в свою очередь является языком реализации платформы Hyperledger Fabric и наиболее активно поддерживается сообществом разработчиков среди языков для написания *Chaincode* программ.

В рамках реализации предложенного подхода предстояло решить следующие задачи:

- Спроектировать модель данных на *Chaincode* уровне, включающую
  - Различные типы сообщений, определённые на *Application* уровне
  - Тип пользовательских кошельков, содержащих голоса участников и заполненные бюллетени
- Разработать способ хранения различных типов в *key-value* базе данных платформы Hyperledger Fabric
- Реализовать логику валидации для каждого типа сообщения в системе в соответствии с протоколом
  - Учесть возможности и ограничения программной модели смарт-контрактов платформы Hyperledger Fabric

4.1. Модель данных

Платформа Hyperledger Fabric позволяет хранить данные в реестре в формате <*ключ*, *значение*> и предоставляет два метода для чтения данных из базы: значение по ключу и диапазон значений по заданным начальному и конечному ключу. Итерация по значениям во втором случае происходит в соответствии с лексикографическим порядком ключей. Чтобы иметь возможность получать из базы

данных необходимые записи, а также необходимые диапазоны записей, было принято решение определить набор сущностей (рис. 5) и соответствующие композитные ключи (таб. 2).

Объект	Тип	Подтип	ID голосования	ID объекта
Уникальное сообщение	*«message»*	*«notification»* *«emission»* *«result»*	+	–
Повторяющееся сообщение	*«message»*	*«request»* *«distribution»* *«vote»*	+	<message ID>
Кошелёк	*«wallet»*	–	+	<wallet address>

Таблица 2: Устройство композитных ключей

Одним из типов записи в базе данных являются сообщения. Каждое сообщение, аналогично кошельку, относится к определённому голосованию. Также каждое сообщение содержит уникальный идентификатор — хеш от содержимого сообщения, цифровую подпись и публичный ключ отправителя, соответствующий подписи. Для каждого вида сообщений определён его подтип, который входит в состав композитного ключа, и соответствующие подтипу сообщения необходимые поля.

Для отслеживания текущего статуса голосования на *Chaincode* уровне необходимо определить формат кошелька пользователя. Прежде всего кошлёк относится к определённому голосованию, в связи с чем он должен содержать уникальный идентификатор голосования в системе. Также необходимо иметь возможность определять принадлежность кошелька конкретному участнику голосования, чтобы предоставлять ему возможность распоряжаться находящимися на кошельке монетами при их распределении между дочерними сущностями, если таковые имеются, или при публикации распределения голосов. Данное требование выполняется определением публичного ключа пользователя в качестве адреса кошелька. Таким образом пользователь модет подтвердить владение кошельком

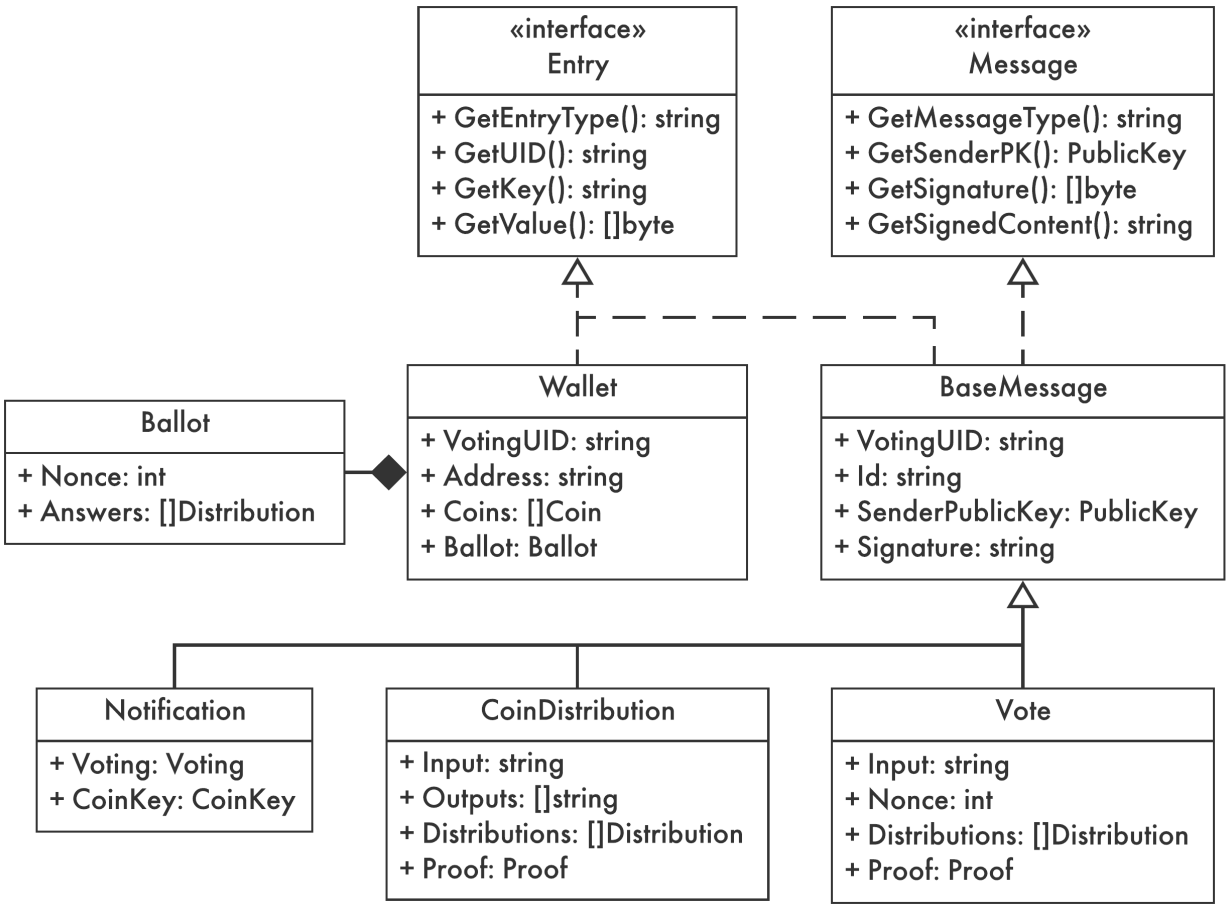

Рис. 5: Модель данных *Chaincode* уровня

и содержащимися на нём средствами, подписывая каждое своё сообщение, содержащее распределение монет, соответствующим приватным ключём. Подпись проверяется на *Chaincode* уровне и в случае, когда она корректна, распределение изменяет общее состояние системы. Протокол E-voting предусматривает возможность многократной публикации распределения голосов участником до завершения голосования. В связи с этим требованием было решено включить в структуру кошелька объект, который содержит актуальное распределение монет по ответам и порядковый номер последнего голосующего сообщения. Это позволит заменять предыдущее волеизъявление пользователя на новое, не пересчитывая промежуточные результаты голосования для каждого запроса.

4.2. E-voting смарт-контракт

Программная модель Hyperledger Fabric определяет исполняемые *Chaincode* программы как набор смарт-контрактов. Смарт-контракт в свою очередь определяется как набор методов, каждый из которых принимает объект контекста транзакции и возвращает необходимые типы данных. Контекст представляет из себя набор данных о текущей транзакции, информации о вызывающем смарт-контракт клиенте, а также предоставляет разработчику доступ к основному API платформы для произведения операций чтения, записи, публикации событий и вызова других смарт-контрактов.

Для каждого типа сообщения, которое может быть опубликовано пользователем посредством *Application* уровня, смарт-контракт **E-voting** определяет метод, реализующий специфичную для данного типа сообщения логику валидации и соответствующего обновления состояния голосования и кошельков участников (таб. 3). Сообщения, непрошедшие валидацию, не сохраняются в реестр и не изменяют состояния системы, а вызывающему клиенту возвращается соответствующая ошибка.

Метод	Тип сообщения	Валидация			Обновление кошельков
		Подпись	Роль	ZKP[12]	
organize	*notification*	+	Organizer	–	–
pubkey	*encryption key*	+	Voting Service	–	–
emit	*emission*	+	Organizer	–	+
request	*request*	+	Intermediary Voter	–	–
distribute	*distribution*	+	Organizer Intermediary	+	+
vote	*vote*	+	Intermediary Voter	+	+
finalize	*result*	+	Voting Service	–	–

Таблица 3: Методы смарт-контракта **E-voting**

[12]Zero-Knowledge-Proof — доказательство неотрицательности номиналов монет

4.3. Контекст и обработчики

Ключевой особенностью контекста транзакции является возможность его кастомизации, то есть определению разработчиком более специфичного типа контекста для конкретной задачи. Контекст **E-voting** смарт-контракта был определён как контейнер для объекта интерфейса **Message** (рис. 5), таким образом на каждом этапе обработки запроса предоставляя доступ к обрабатываемому сообщению.

Помимо возможности реализации специфичных для смарт-контракта методов и кастомизации контекста транзакции, программная модель Hyperledger Fabric позволяет определить для каждого смарт-контракта набор обработчиков, исполняемых вне зависимости от конкретного вызываемого метода смарт-контракта. Обработка каждого запроса начинается с формирования контекста транзакции и происходит в три этапа, между которыми контекст сохраняет своё состояние:

- Вызов определённого для смарт-контракта обработчика $BeforeTx$
- Исполнение вызываемого клиентом метода смарт-контракта
- Вызов определённого для смарт-контракта обработчика $AfterTx$

В случае возникновения ошибки на любом из этапов обработка запроса прерывается и ошибка возвращается клиенту. В случае вызова несуществующего метода смарт-контракта исполнение передаётся обработчику $UnknownTx$, реализация которого также может быть определена разработчиком, и для **E-voting** смарт-контракта представляет из себя логирование данного запроса и возвращения ошибки вызывающему клиенту.

Такая программная модель в случае смарт-контракта **E-voting** позволяет обобщить те части логики обработки сообщений, которые не зависят от его конкретного типа, и вынести их в реализации обработчиков:

- *BeforeTx*
  - Десериализует переданное сообщение из *json* формата
  - Проверяет соответствие вызываемой функции типу переданного сообщения
  - Проверяет цифровую подпись сообщения
  - Записывает объект сообщения в контекст транзакции
- *AfterTx*
  - Сохраняет сообщение из контекста в реестр с соответствующим композитным ключём
  - Публикует событие, соответствующее типу сообщения

5. Тестирование системы

Целью тестирования является проверка работоспособности системы, реализованной в рамках предложенного подхода, сравнение с исходной реализацией и выявление способов дальнейшего её улучшения. Под работоспособностью в данном случае подразумевается способность системы корректно функционировать в рамках проведения голосования, приближенного к реальному миру.

5.1. Нагрузочное тестирование

Валидация сообщений — часть логики системы, которая в ходе работы была перемещена с *Application* на *Chaincode* уровень. В связи с чем для сравнения подходов был проведён анализ быстродействия данной функциональности в обеих реализациях. Тестирование *Application* уровня, написанного на Java, производилось с использованием инструмента JMH[13] — популярного бенчмарк-фреймворка для языков семейства JVM. Тестирование *Chaincode* уровня, реализованного на Go, производилось средствами стандартной библиотеки языка, поддерживающей возможность написания бенчмарк-тестов. Все замеры производились на рабочей станции MacBook Pro (15-inch, 2019) со следующими характеристиками:

- Операционная система: macOS Mojave v10.14.6
- ЦПУ: Intel Core i7, 2.6 GHz, 6 Cores, 12 Logical processors
- ОЗУ: 32 GB 2400 MHz DDR4
- Java: AdoptOpenJDK 8 (v1.8.0_242, x86_64)
- Go: go1.14.2 (darwin/amd64)
- Hyperledger Fabric: v2.1.0 (commit: 1bdf97537)

[13]Java Microbenchmark Harness — `https://openjdk.java.net/projects/code-tools/jmh/`

Конфигурация голосования была выбрана на основе годового общего собрания акционеров Московской биржи от 28.04.2020 [17], которое проходило в заочной форме с применением системы электронного голосования НРД[14]. Голосование для тестирования состояло из 30 вопросов по три ответа для каждого.

Для каждого метода было проведено порядка 1000 запусков тестов, в каждом из которых подсчитывалось среднее время выполнения запроса в течение десяти секунд, что является стандартной конфигурацией для инструмента JMH и настраиваемым параметром для бенчмарк-библиотеки языка Go. Вычисленные интервалы быстродействия реализаций с доверительной вероятностью 0.99 по результатам замеров приведены в таблице 4.

Метод	Application (ms/op)	Chaincode (ms/op)
organize	0.468 ± 0.010	0.477 ± 0.001
emit	0.150 ± 0.002	0.310 ± 0.001
distribute	150.167 ± 0.702	82.826 ± 0.353
vote	601.631 ± 2.746	268.063 ± 1.041
finalize	117.593 ± 2.402	148.902 ± 0.863

Таблица 4: Сравнение быстродействия логики валидации

Резализация логики валидации сообщений на *Chaincode* уровне содержит дополнительные необходимые операции, которые отсутствуют в исходной реализации *Application-based* подхода. Такими операциями являются чтение из реестра данных, необходимых для валидации сообщения, запись обработанного сообщения в реестр и сопуствующие сериализация и десериализация сообщений. Однако, учитывая, что методы *organize*, *emit* и *finalize* исполняются единожды для каждого голосования, незначительное увеличение времени их работы не играет существенной роли для общей работоспособности системы.

Большую часть времени выполнения методов *distribute*, *vote* и *finalize* занимают арифметические операции над монетами, которые представлены точками на эллиптической кривой, и верификация

[14]НРД — Национальный расчётный депозитарий

криптографических доказательств. Применение инструментов конкурентного исполнения функций, которые предоставляет язык Go, позволили сократить время выполнения данных операций в более чем два раза по сравнению с их последовательной реализацией в рамках того же *Chaincode-based* подхода. Методы *distribute* и *vote* составляют основную нагрузку на систему в рамках голосования, так как их количество линейно зависит от количества участников процесса. Таким образом полученные результаты говорят о том, что реализованная в рамках *Chaincode-based* подхода логика валидаци работает быстрее, чем в существующей реализации *Application-based* подхода. Стоит отметить, что сравнение нового подхода производилось с исходной версией *Application-based* системы, в которой аналогичные оптимизации времени исполнения арифметических и криптографических операций реализованы не были.

5.2. Имитация голосования

Проверка работоспособности системы проходила при помощи имитации активности участников голосования Московской биржи. 1659 акционеров, принявших участие в годовом общем собрании, проголосовали электронно через сервис Национального расчётного депозитария с 8 по 28 апреля 2020 года [1]. В связи с тем, что E-voting система предоставляет возможность пользователям публиковать свои голоса, обращаясь к соответствующим родительским **Intermediary** модулям, отпадает необходимость в запуске персональных **Voter** модулей для каждого участника. Учитывая эти данные, конфигурация тестируемой системы была подобрана следующим образом:

- Голосование: 30 вопросов с тремя возможными ответами
- Участники: 1500 акционеров
- Application уровень:
  - Один модуль **Organizer**

  - Один модуль **Voting Service**
  - Три модуля **Intermediary**, каждый из которых обслуживает треть участников голосования

- Hyperledger Fabric:
  - Три **Ordering Service** узла, работающих в соответствии с консенсусом Raft [9, 10]
  - Шесть **Peer** узлов с установленными *E-voting chaincode*

После этапа инициализации голосования, происходит распределение монет от **Organizer** его дочерним **Intermediary**. Далее начинается этап распределения монет участникам голосования и обработка голосов, распределённых ими по ответам. Получая от пользователя запрос на публикацию голосов, **Intermediary** сначала отправляет на *Chaincode* уровень запрос *distribute*, который создаст для участника кошелёк с необходимым количеством монет. После этого **Intermediary** от лица пользователя формирует и отправляет в смарт-контракт запрос *vote*, окончательно публикуя волеизъявление участника. Активность участников голосования имитировалась генерацией отложенных запросов к соответствующим **Intermediary** модулям. Реализованная таким образом имитация голосования позволила успешно протестировать систему под постоянной нагрузкой на приближенном к реальному миру сценарии использования.

По итогам тестового запуска процесса голосования было опубликовано и обработано 1503 сообщения типа *distribute* и 1500 сообщений типа *vote*. В рамках описанной конфигурации размер одного сообщения типа *distribute* в среднем составил 12 килобайт, 8 килобайт из которых составляет размер криптографического доказательства, в то время как размер одного *vote* сообщения в среднем составил 52 килобайта с криптографическим доказательством размером 16 килобайт. Так как сообщения данных типов занимают большую часть ресурсов памяти блокчейна, возможным улучшением системы в будущем может стать замена текущей реализации

zero-knowledge-proof доказательств неотрицательности монет на основе алгоритма Borromean ring signature [8] алгоритмом Bulletproofs [6], одной из отличительных черт которого является возможность агрегирования нескольких доказательств с небольшим размером итогового результата [5].

Заключение

В рамках данной работы были достигнуты следующие результаты:

- Проведён обзор предметной области
  - Изучен протокол тайного электронного голосования E-voting
  - Изучена платформа Hyperledger Fabric
- Спроектирована архитектура системы
  - Проведён анализ существующего *Application-based* решения
  - Разработан новый *Chaincode-based* подход
  - Проведено сравнение подходов
- Реализован новый подход
  - Реализована система смарт-контрактов и логика валидации сообщений протокола E-voting
  - Проведён рефакторинг исходной реализации
- Проведено тестирование системы

Список литературы

[1] 99% акционеров Московской биржи, участвовавших в годовом Общем собрании акционеров, проголосовали электронно через НРД | О собраниях. — 2020. — Access mode: https://www.e-vote.ru/ru/news/meet/index.php?id29=634686#bid2252 (online; accessed: 20.05.2020).

[2] Blockchain-Based Electronic Voting System for Elections in Turkey / R. Bulut, A. Kantarcı, S. Keskin, Ş. Bahtiyar // 2019 4th International Conference on Computer Science and Engineering (UBMK) / IEEE. — 2019. — P. 183–188. — Access mode: https://doi.org/10.1109/UBMK.2019.8907102.

[3] Ivanov Aleksandr Nikolaevich, Kazennov Aleksei Vladimirovich, Mavchun Georgii Valerievich et al. Blockchain-based cryptologic ballot organization. — 2019. — Oct. 15. — US Patent 10,445,965.

[4] Ivanov Aleksandr Nikolaevich, Kazennov Aleksei Vladimirovich, Mavchun Georgii Valerievich et al. Blockchain-based cryptologic ballot verification. — 2019. — Aug. 20. — US Patent 10,388,097.

[5] Bulletproofs: Faster Rangeproofs and Much More. — 2018. — Access mode: https://blockstream.com/2018/02/21/en-bulletproofs-faster-rangeproofs-and-much-more/ (online; accessed: 25.05.2020).

[6] Bulletproofs: Short Proofs for Confidential Transactions and More / B. Bünz, J. Bootle, D. Boneh et al. // 2018 IEEE Symposium on Security and Privacy (SP). — 2018. — P. 315–334.

[7] Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains / Elli Androulaki, Artem Barger, Vita Bortnikov et al. // Proceedings of the Thirteenth EuroSys Conference. — EuroSys '18. — Porto, Portugal : ACM, 2018. — P. 30:1–30:15. — Access mode: http://doi.acm.org/10.1145/3190508.3190538.

[8] Maxwell Gregory, Poelstra Andrew. Borromean ring signatures // Accessed: Jun. — 2015. — Vol. 8. — P. 2019. — Access mode: https://github.com/Blockstream/borromean_paper/raw/master/borromean_draft_0.01_9ade1e49.pdf.

[9] Ongaro Diego, Ousterhout John. In Search of an Understandable Consensus Algorithm // Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference. — USENIX ATC'14. — USA : USENIX Association, 2014. — P. 305–320. — Access mode: https://dl.acm.org/doi/10.5555/2643634.2643666.

[10] The Ordering Service — Raft. — 2020. — Access mode: https://hyperledger-fabric.readthedocs.io/en/release-2.1/orderer/ordering_service.html#raft (online; accessed: 11.05.2020).

[11] An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends / Z. Zheng, S. Xie, H. Dai et al. // 2017 IEEE International Congress on Big Data (BigData Congress). — 2017. — P. 557–564. — Access mode: https://ieeexplore.ieee.org/document/8029379.

[12] Practical Range Proof for Cryptocurrency Monero with Provable Security / Kang Li, Rupeng Yang, Man Ho Au, Qiuliang Xu // Information and Communications Security / Ed. by Sihan Qing, Chris Mitchell, Liqun Chen, Dongmei Liu. — Cham : Springer International Publishing, 2018. — P. 255–262. — Access mode: https://doi.org/10.1007/978-3-319-89500-0_23.

[13] Zcash protocol specification version 2020.1.3 (2020) / D Hopwood, S Bowe, T Hornby, N Wilcox // GitHub: San Francisco, CA, USA. — 2019. — Access mode: https://github.com/zcash/zips/blob/master/protocol/protocol.pdf.

[14] e-Estonia — i-Voting. — 2020. — Access mode: https://e-estonia.

com/solutions/e-governance/i-voting/ (online; accessed: 12.05.2020).

[15] An overview on smart contracts: Challenges, advances and platforms / Zibin Zheng, Shaoan Xie, Hong-Ning Dai et al. // Future Generation Computer Systems. — 2020. — Vol. 105. — P. 475–491. — Access mode: http://www.sciencedirect.com/science/article/pii/S0167739X19316280.

[16] A survey on zero knowledge range proofs and applications / Eduardoa Morais, Tommy Koens, Cees van Wijk, Aleksei Koren // SN Applied Sciences. — 2019. — Vol. 1, no. 8. — P. 946. — Access mode: https://doi.org/10.1007/s42452-019-0989-z.

[17] Московская Биржа - Материалы к годовому Общему собранию акционеров ПАО Московская Биржа 2020. — 2020. — Access mode: https://www.moex.com/a7168 (online; accessed: 20.05.2020).

[18] Электронные выборы на блокчейн в Москве депутатов в Московскую городскую Думу 2019 / Мосгордума / Проекты / Сайт Москвы. — 2019. — Access mode: https://www.mos.ru/city/projects/blockchain-vybory/ (online; accessed: 12.05.2020).