Санкт-Петербургский государственный университет

Фараджов Анар Мурватович

Выпускная квалификационная работа бакалавра

Тестирование и сравнительный анализ DLT решений

Уровень образования: бакалавриат
Направление: 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2015 «Прикладная математика, фундаментальная информатика и программирование»

Научный руководитель:
доктор физ.-мат. наук, профессор,
Богданов А.В.

Рецензент:
кандидат технических наук,
Дик Г. Д.

Санкт-Петербург
2022

Содержание

Введение

Консенсус – это процесс согласования данных в системе, участники которой не могут вполне доверять друг другу. Алгоритмы достижения консенсуса в среде с неизвестным числом неисправных узлов как правило требуют создания некоторого протокола голосования. Так, различают алгоритмы, базирующиеся на выборе лидера посредством “лотереи” (Nakamoto-style consensus), а также BFT (Byzantine fault tolerant) алгоритмы, базирующиеся на много раундовых голосованиях.

На данный момент существует множество алгоритмов консенсуса. Многие из них предполагают дополнительные издержки для участников (Proof-of-Work, Proof-of-Stake, Proof-of-Elapsed-Time и др.). Однако существует ряд алгоритмов, не накладывающих подобных ограничений. Как правило, эти алгоритмы базируются на решении задачи византийских генералов (Byzantine fault tolerance problem), наиболее известным из них является Practical Byzantine Fault Tolerance (PBFT).

В связи с существованием большого разнообразия функционально похожих систем, появляется необходимость, и, что не менее важно, возможность проведения тестирования и сравнения различных DLT платформ между собой в рамках некоторого круга задач, описываемых сценариями тестирования.

Постановка задачи

В рамках данной работы проводится разработка методологии тестирования DLT систем и сравнение эффективности работы двух платформ для формирования распределенных реестров (Distributed Ledger Technology, DLT) на базе PBFT-подобного консенсуса и определения области их применимости.

Первая из рассматриваемых платформ, Hyperledger Sawtooth - корпоративное решение с открытым исходным кодом, имеющее в своем составе несколько готовых алгоритмов консенсуса, в частности, оригинальный алгоритм PBFT. Вторая, BGX/DGT – решение на базе Hyperledger Sawtooth со встроенным алгоритмом F-BFT, который является двухуровневой модификацией PBFT.

Обзор литературы

Относительно алгоритмов консенсуса, применяемых в различных DLT системах традиционно существует широкая дискуссия. Это не удивительно – внедрение технологии распределенных реестров в различные области общественной жизни поднимает новые вопросы в области безопасности и функционирования DLT платформ [1].

Наиболее заметными теоретическими работами в этой области можно назвать представление, формализация и решение задачи византийских генералов [2], а также создание алгоритма Paxos – прародителя современных BFT-базированных алгоритмов консенсуса [3].

Помимо работ Лэмпорта, стоит отдельно отметить работы Барбары Лисков и Мигеля Кастро [4], которые дополняют идею BFT-базированного консенсуса рядом доработок и представляют алгоритм PBFT, который широко используется на практике.

Важной вехой развития идей стало представление Синтией Дуорк и Мони Наором основных концепций алгоритма Proof-of-Work [5], который позже лег в основу сетей Bitcoin и Ethereum.

Наиболее близкой к данной, в отношении предмета и применяемых методик, является работа Женьшунь Ши и соавторов [6]. В данной работе авторы проводят оценку эффективности и стабильности работы платформы Hyperledger Sawtooth.

Глава 1. Теоретические сведения о DLT системах

1.1. Общие сведения о распределенных реестрах

Распределенные реестры – это один из типов распределенных баз данных, использующий технологию блокчейна для хранения транзакций отдельных участников. В таких системах не бывает центрального хранилища данных, что заметно усложняет администрирование подобных сетей, но повышает их надежность. В сущности, требования к подобным реестрам обычно следующие:

• Распределенность: Цепочка блоков распределена между узлами с малым уровнем доверия. В таких сетях база данных может быть шардирована (каждый узел хранит только часть данных), либо на каждом узле может храниться полная реплика всей информации.

• Неизменяемость: база данных представляет собой неизменяемую историю всех транзакций, в которой используются хэши блоков, чтобы упростить обнаружение и предотвращение попыток изменения истории.

• Защищенность: все узлы должны быть идентифицированы, а все транзакции должны быть подписаны.

1.2. CAP-теорема

Впервые упоминания о CAP теореме появились в 2000 году благодаря её создателю, Эрику Брюеру [7], затем она была доказана 2 года спустя в 2002 году. Эта теорема получила широкое освещение, поскольку индустрия начала сталкиваться с проблемой, которая сейчас получила название “Big Data Problem” (“Проблема Больших Данных”). В CAP говорится, что в распределенной системе возможно выбрать только 2 из 3-х свойств, что показано на Рисунке 1:

1) C (consistency) — согласованность. Каждое чтение выдаст самую последнюю запись.

Согласованность – это гарантия того, что одновременное чтение из

разных мест вернет одно и то же значение. То есть система не возвращает устаревшие или противоречивые данные.

2) A (availability) — доступность. Каждый узел (исправный) всегда успешно выполняет запросы (на чтение и запись). Доступность – это способность системы реагировать на обычные операции, так и вероятность сбоев.

3) P (partition tolerance) — устойчивость к распределению. Даже если между узлами нет связи, они продолжают работать независимо друг от друга. Разделение – это число, подверженность ошибкам и производительность независимых компонентов в системе и сети, которая их соединяет. Рассматриваемые системы могут условно быть условно отнесены к CP – типу.

Рисунок 1 – Иллюстрация CAP

1.3. Потенциальные уязвимости распределенных систем

Распределение элементов вычислительной системы на физически отделенные друг от друга узлы часто выполняется для увеличения устойчивости к выходу из строя отдельных составляющих. Выход из строя элементов системы зачастую может быть следствием не только ошибок, допущенных при разработке и эксплуатации, но и

следствием злого умысла. В этой связи наиболее распространенными типами атак на блокчейн-системы являются атака Сивиллы, атака 51% и двойная трата [8].

1.3.1 Атака Сивиллы

Злоумышленник создает новые узлы в одноранговой сети, которые постепенно «окружают» атакуемый узел, принцип проиллюстрирован на Рисунке 2. Так как каждый узел хранит и обновляет собственный «рейтинг» соседних узлов, через некоторое время он больше доверяет тем, кто предоставил ему данные. Создание нового идентификатора узла в одноранговой сети ничего не стоит, поэтому злоумышленник может применять различные стратегии «окружения», создавая новые идентификаторы быстрее, чем их обнаруживают защитники сети. Как только хост получает данные только от хостов, контролируемых злоумышленником, злоумышленник может начать предоставлять хосту ложные данные.

При проведение платежей с использованием блокчейн систем, атака Сивиллы может иметь некоторые специфические особенности:

- при получении контроля над участком сети (несколько соседних узлов), злоумышленник может блокировать проведение любых транзакций на атакуемом узле;
- злоумышленник, контролирующий участок сети который включает в себя достаточно большое количество узлов может валидируемые в сети транзакции, а также изменять уже принятые;
- захвативший достаточное количество связанных узлов может имитировать задержки сети, “придерживая” транзакции – такое “придерживание” может быть использоваться для реализации фронтраннинга[9], что ведет к выполнению транзакции злоумышленником раньше пользовател;
- в некоторых случая на основе атаки Сивиллы может быть реализована атака двойной траты. Например, если участник сети держит

достаточно большое количество монет, в случае PoS консенсуса, или при достаточно высоком хэшрейте, в случае PoW консенсуса [10].

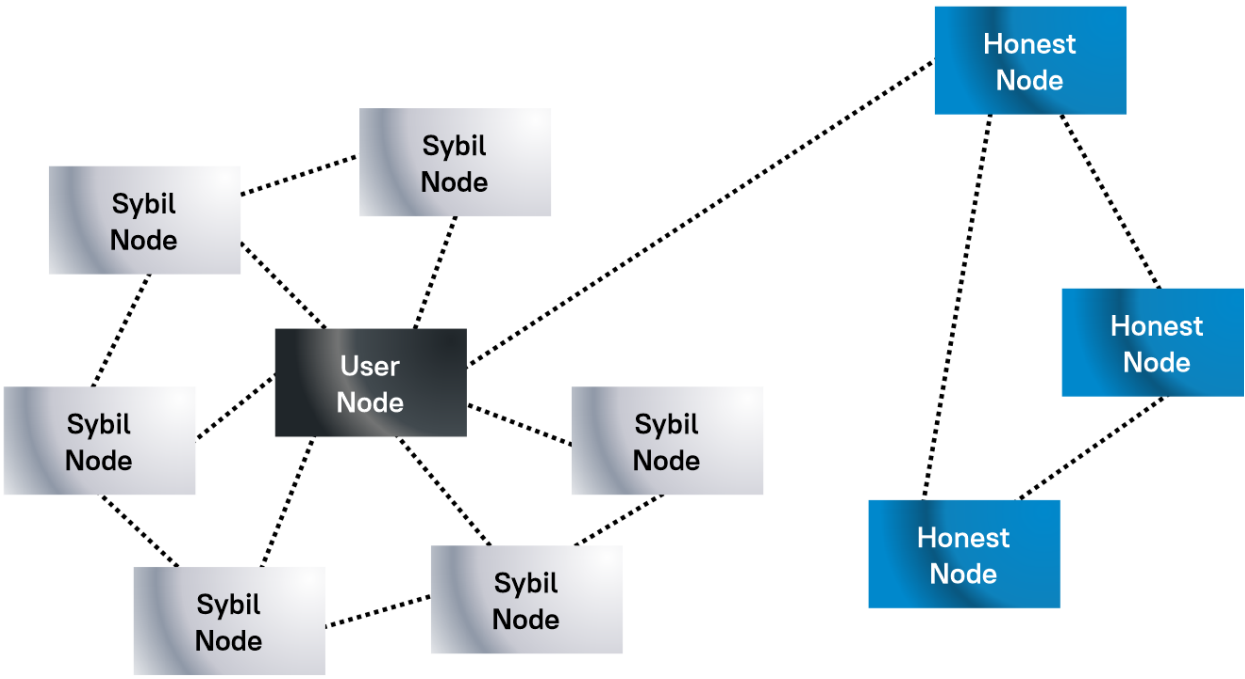

Рисунок 2 – Атака Сивиллы

1.3.2 Атака 51%

Атака 51% — это уязвимость блокчейнов с различными типами консенсусов, смысл которой состоит в захвате злоумышленником более половины всех узлов системы [11].

В зависимости от типа консенсуса, контроль над более чем половиной узлов может давать различные возможности:

- блокировать транзакции пользователей сети, подключающихся к неподконтрольным атакуюшим узлам;
- проводить атаку двойной траты, что позволяет злоумышленнику присвоить средства пользователя;
- разделить цепочку блоков на две конкурирующие ветви, тем самым нарушив консистентность всей системы;
- могут получить комиссию и награду за майнинг блоков и проведение транзакций.

С увеличением доли контролируемых узлов, возможности атакующего могут увеличиваться, так, например, при приобретении контроля над значительно большем количеством узлов, чем 51% сети, злоумышленник может:

- изменять топологию сети;
- изменять и манипулировать транзакциями, находящимися вне генезис-блока, получать дополнительное вознаграждение (путем отката старых блоков и переполучения наград за эти блоки);
- удалять контракты или историю транзакций (за счет отката старых блоков и редактирования списка включаемых транзакций).

Однако сам по себе этот тип атак не позволяет злоумышленникам:

- заполучить приватный ключ или подделать подпись участника;
- заполучить монеты, полученные в результате неправильной работы контракта;
- управлять решениями неподконтрольных валидаторов.

1.3.3 Атака Double spending

Двойная трата — это тип атак на распределенные системы, при котором один и тот же ресурс может быть израсходован дважды. Так, например, при наличии достаточно большой задержки в блокчейн системе и несовершенства механизма подтверждения транзакции, пользователь может потратить больше монет чем находится у него в распоряжении.

С точки зрения управления системой, этот тип атак является наиболее простым в исполнении, но в то же время достаточно опасным, потому для любой блокчейн системы важно иметь механизм противодействия атакам этого типа.

Глава 2. Обзор тестируемых DLT платформ

2.1. Hyperledger Sawtooth

Hyperledger Sawtooth [12] – это корпоративная блокчейн-платформа для создания блокчейн-приложений и распределенных реестров.

Sawtooth упрощает разработку блокчейн-приложений, разделяя ядро системы и пользовательские функции. Разработчики приложений могут определять логику работы приложения используя язык и технологии по своему выбору, без необходимости знать устройство системы целиком. Каждый отдельный узел Sawtooth имеет модульную конструкцию. Эта модульность позволяет совмещать множество уровней бизнес-логики в пределах одного узла. Такая архитектура позволяет пользователям самим определять правила формирования транзакций, присоединения к сети и организации консенсуса.

2.2. BGX/DGT

Платформа BGX/DGT [13] базируется на фреймворке Hyperledger Sawtooth. Потому оно наследует все основные технологические и архитектурные концепции Sawtooth версии 1.1. Тем не менее, в наиболее существенных деталях BGX/DGT сильно отличается от продукта Hyperledger. В отличие от Sawtooth, BGX/DGT работает на в одноранговой сети, вместо этого он использует федеративную структуру – несколько одноранговых сетей, соединенных узлами верхнего уровня – арбитрами. Так как система использует двухуровневую сетевую топологию, то наиболее общепринятые BFT алгоритмы не подходят в качестве алгоритмов консенсуса, потому BGX/DGT[14] использует алгоритм F-BFT (Federated byzantine fault tolerance). Как и в Sawtooth, узлы в BGX/DGT составляются из нескольких модулей, сообщающихся при помощи протокола TCP. Центральный элемент узла – валидатор, в задачи которого включено:

• Обеспечение взаимодействия между различными модулями, составляющих узел;

• Обеспечение взаимодействия между узлами;

• Аутентификация и проверка идентичности участников консенсуса;

• Шифрование и обеспечение безопасности передаваемой информации.

Другой важный модуль, обеспечивающий функционирование системы – модуль REST-API. Его задача заключается в обеспечении возможности взаимодействия пользователя и системы, формировании интерфейса.

Третьим обязательным модулем является консенсус-модуль, который представлен также отдельным заменяемым компонентом. Помимо трех перечисленных модулей, система поддерживает разные типы процессоров транзакций, реализующих пользовательскую логику и обрабатывающих настройки системы.

• Система хранения базируется на DAG, подобно IOTA, Hedera Hashgraph, Orumesh, DagCoin, Byteball, Nano. Подход BGX отличается от названных систем тем, что позволяет осуществлять голосование в федеративной сетевой структуре с настраиваемой топологией;

• Федеративный подход к голосованию активно используется такими решениями, как Ripple, Stellar. BGX переносит эту идеологию на горизонтально масштабируемым DAG и за счет ротации лидеров в кластерах обеспечивает динамическую топологию, ведущую к решению проблемы реализации безопасности за счет устойчивости сети;

• Система консорциум-базированного консенсуса позволяет сохранить гибкость, не теряя скорости и интероперабельности. Например, Hyperledger Fabric нацелен на частные одноранговые сети и требует формирования специальных вспомогательных структур (sidechain), а решение ICON использует специальный механизм Loopchain Fault Tolerance для взаимодействия с другими сетями. В отличии от данных

решений, BGX реализует динамическую топологию поверх DAG, позволяя достигать высокой степени асинхронности работы сети.

2.3. Описание алгоритма PBFT

Представим модель нашей системы как некоторую машину состояний, которая изображена на Рисунке 3. Каждый узел поддерживает некоторое состояние и набор операций. Обозначим все множество узлов за R и будем нумеровать каждый узел целым числом из отрезка [0; |R|-1]. Для простоты также предположим, что |R| = 3f+1, где f – максимально возможное число неверно работающих узлов (узлов, которые не посылают сообщений или посылают неверные сообщения).

В течение своей работы, узлы меняют свою конфигурацию (каждая такая конфигурация далее называется видом). В одном виде должен быть ровно один узел, называемый лидером (см. Рисунок 2).

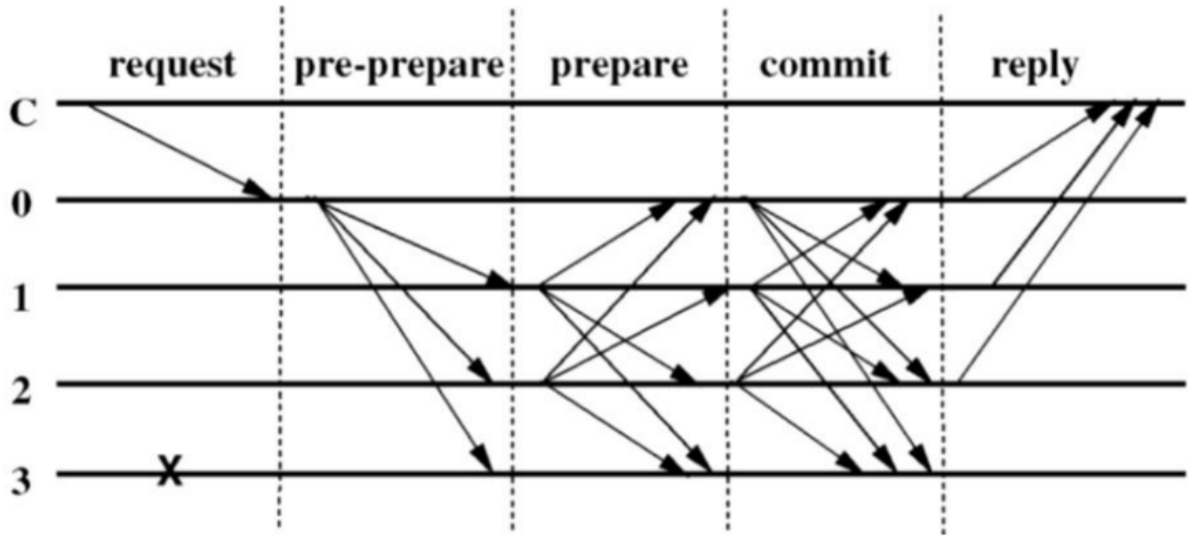

Рисунок 3 – Фазы алгоритма PBFT

Каждый вид v далее мы будем нумеровать последовательно, а смена лидера будет осуществляться по закону p = v mod |R|, где p – номер узла, являющегося лидером.

Сделаем также два предположения, во-первых, машина состояний является детерминистической - выполнение одних и тех же операций при одном и том же первоначальном состоянии возвращает один и тот же

результат, во-вторых, все узлы в начале работы системы находятся в одном состоянии.

Помимо узлов системы введем понятие клиента, посылающего лидеру сообщение <REQUEST, o, t, c>. Здесь, первый операнд означает тип сообщения, второй – операцию, третий – время, четвертый – отправителя. Кроме этого, клиент может принимать сообщения <REPLY, v,t,c,i,r>, где второй операнд указывает на вид, пятый – номер узла, а шестой – результат выполнения операции.

После первоначальной отправки сообщения типа REQUEST, клиент будет ожидать не менее 2f+1 сообщений типа REPLY от разных узлов с совпадающими значениями времени и результатом перед тем как этот результат принять.

В случае, если лидер не разошлет полученный от клиента REQUEST остальным узлам, он будет заподозрен в неверной работе, что повлечет смену вида.

Каждый узел содержит лог сообщений, в котором указаны все сообщения, принятые данным узлом.

Как только лидер p получает сообщение m от клиента, он рассылает всем узлам полученный запрос, который обрабатывается в три фазы.

Первая фаза - pre-prepare. При которой лидер выбирает число n и рассылает сообщение содержания <PRE-PREPARE ,v,n,d>, где d – подпись клиента. Следует отметить, что сам запрос на этом этапе не рассылается.

Смысл данного сообщения состоит именно в том, чтобы выбрать n и указать v, которому он принадлежит, чтобы упорядочить все полученные от клиента запросы по n. Это позволяет оптимизировать расходы на транспортировку сообщений.

Получивший данное сообщение узел принимает его если выполнены следующие условия:

• подпись лидера и подпись клиента верна;

• сообщение из того же вида v, в котором сейчас находится узел;

• раньше не было получено таких же сообщений с теми же v и n;

• n содержится в заранее выбранном отрезке [h;H].

Если узел i принимает данное сообщение, он входит в новую фазу prepare, в которой рассылает подтверждающие сообщения <PREPARE, v, n, d, i> всем остальным узла, а также добавляет полученное pre-prepare и отправленное prepare сообщения в лог. В ином случае узел не делает ничего.

Узел, получающий сообщение PREPARE, включает его в свой лог в случае, если v, d и подпись верны, а n содержится в отрезке [h; H].

Будем говорить, что предикат prepared(m,v,n,i) выполняется тогда и только тогда, когда в логе узла содержится REQUEST, PRE-PREPARE с тем же значением v в котором сейчас находится данный узел и для того же клиента m, от которого был получен REQUEST, а кроме того, было получено не менее 2f верных PREPARE сообщений от остальных узлов.

Фазы pre-prepare и prepare позволяют добиться того, что верно работающие узлы согласуют порядок полученных сообщений REQUEST внутри одного вида. Более точно, наличие этих фаз позволяет установить верность следующего утверждения:

Если prepared(m,v,n,i) верно, то любое prepared(m’,v,n,j), такое, что $D(m') \neq D(m)$, где D – подпись узла.

Это утверждение верно по той причине, что в данном предположении хотя бы f+1 узел прислал сообщение типа PREPARE или PRE-PREPARE для данных m, v, n. А значит, для что для того чтобы prepared(m’,v,n,j) выполнялось, необходимо, чтобы один из этих узлов прислал сообщение PREPARE с одинаковыми m, n и v дважды, но по предположению в системе не более f неверно работающих узлов.

Узел i рассылает сообщения типа <COMMIT, v,n,m,i> сразу как только предикат prepared(m,v,n,i) оказывается выполнен, начиная таким образом фазу commit. Получившие его узлы записывают сообщение в лог если выполнены вышеозначенные условия.

Будем обозначать состояние как committed(m,v,n) тогда и только тогда, когда prepared(m,v,n,i) верно для всех i из некоторого подмножества R мощности f+1, и committed-local(m,v,n,i) тогда и только тогда, когда верно prepared(m,v,n,i) и узел получил 2f+1 сообщений типа COMMIT с верными подписями, и с соответствующими полученному ранее PRE-PREPARE, значениями v и n.

Следует заметить, что верность утверждения committed-local(m,v,n,i) влечет за собой выполнение committed(m,v,n).

После выполнения committed(m,v,n), узлы выполняют запрос и посылают клиенту результат.

Наличие последней фазы позволяет убедиться в том, что все исправные узлы будут выполнять запросы в одинаковом порядке, так как алгоритм не предполагает последовательное получение запросов.

2.4. Описание алгоритма F-BFT

1. Во время инициализации сети выбираются узлы-арбитры и узлы-лидеры. Эти роли могут совпадать в одном узле (см. Рисунок 4).

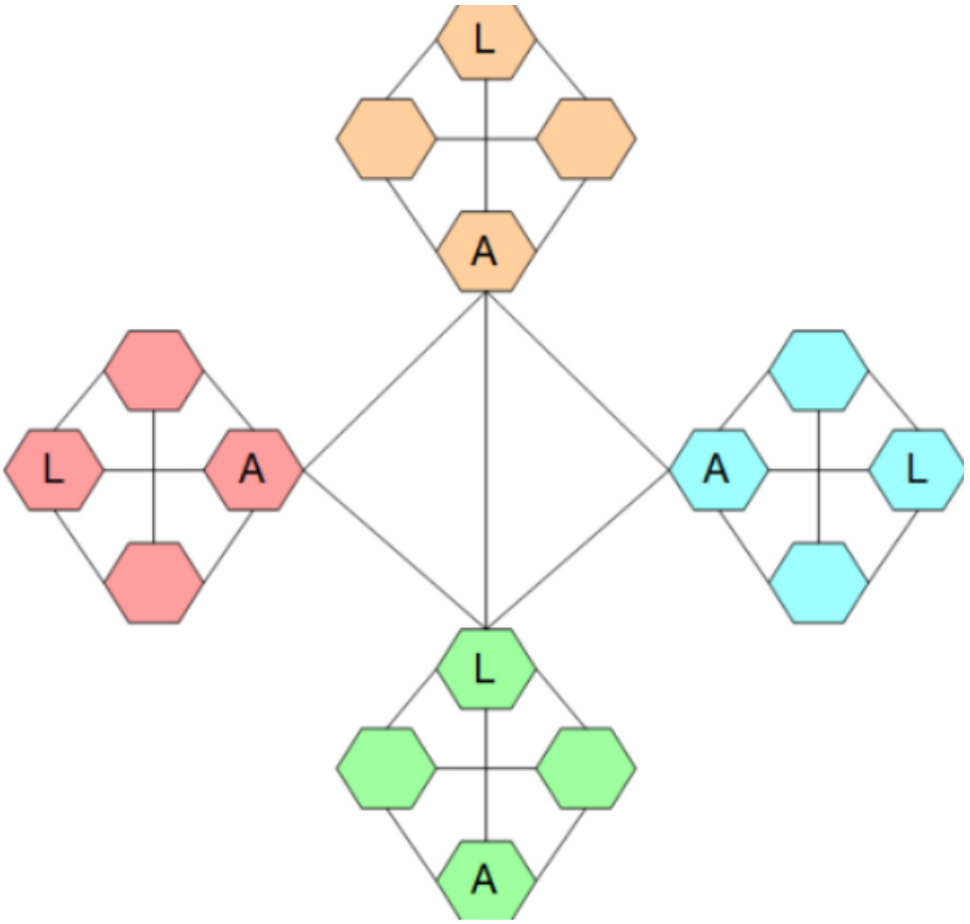

Рисунок 4 – Стадия 1

2. Допустим, какой-то узел решил отправить транзакцию. Блок, содержащий транзакцию принимается консенсусом в кластере которому принадлежит узел-инициатор, но все внешние узлы также получают новый 13 блок на проверку. После проверки корректности они застывают в ожидании разрешения от арбитра их кластера на принятие этого блока (см. Рисунок 5).

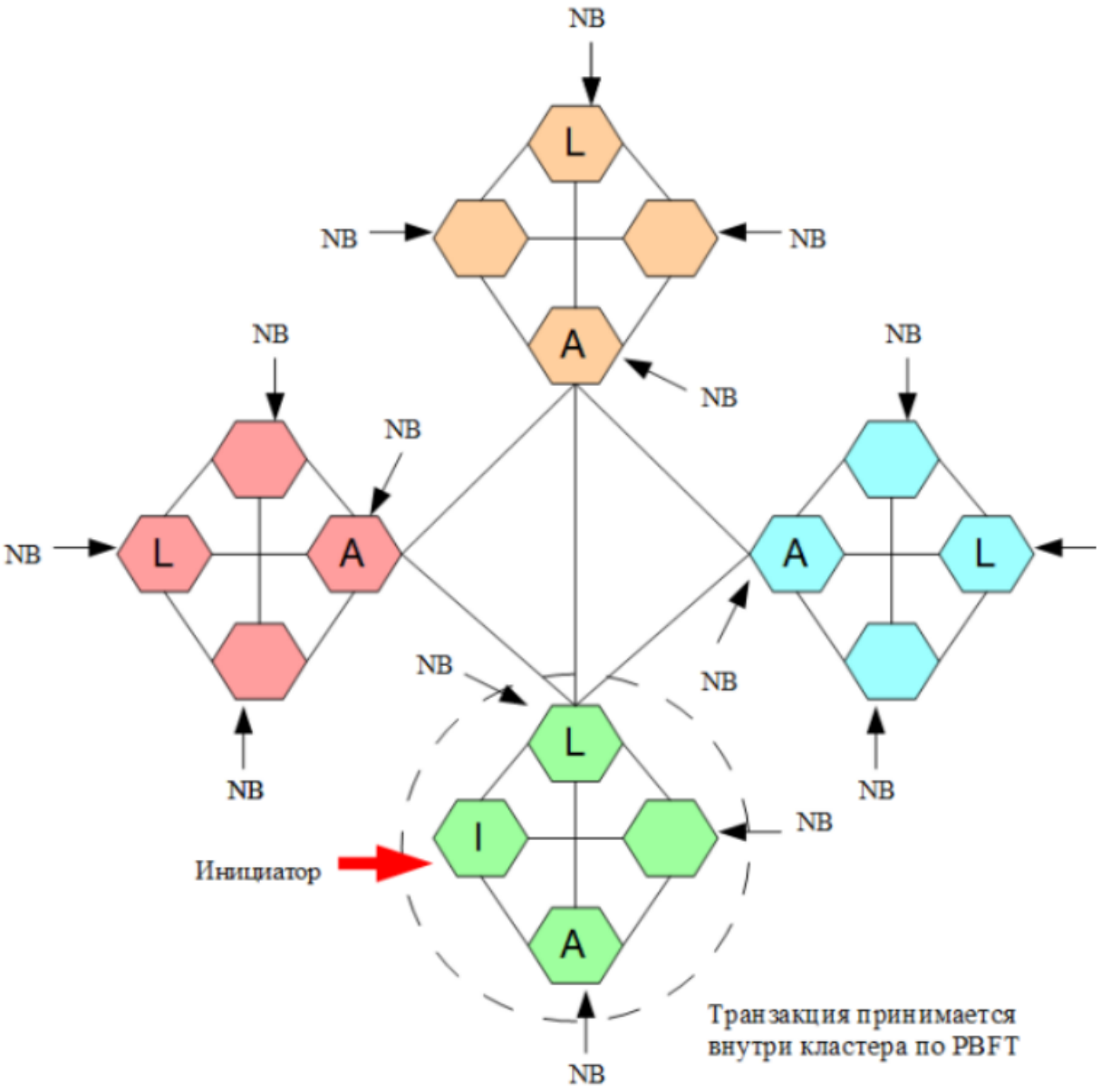

Рисунок 5 – Стадия 2

3. После принятия блока в кластере, лидер отправляет сообщение о консенсусе арбитрам (см. Рисунок 6).

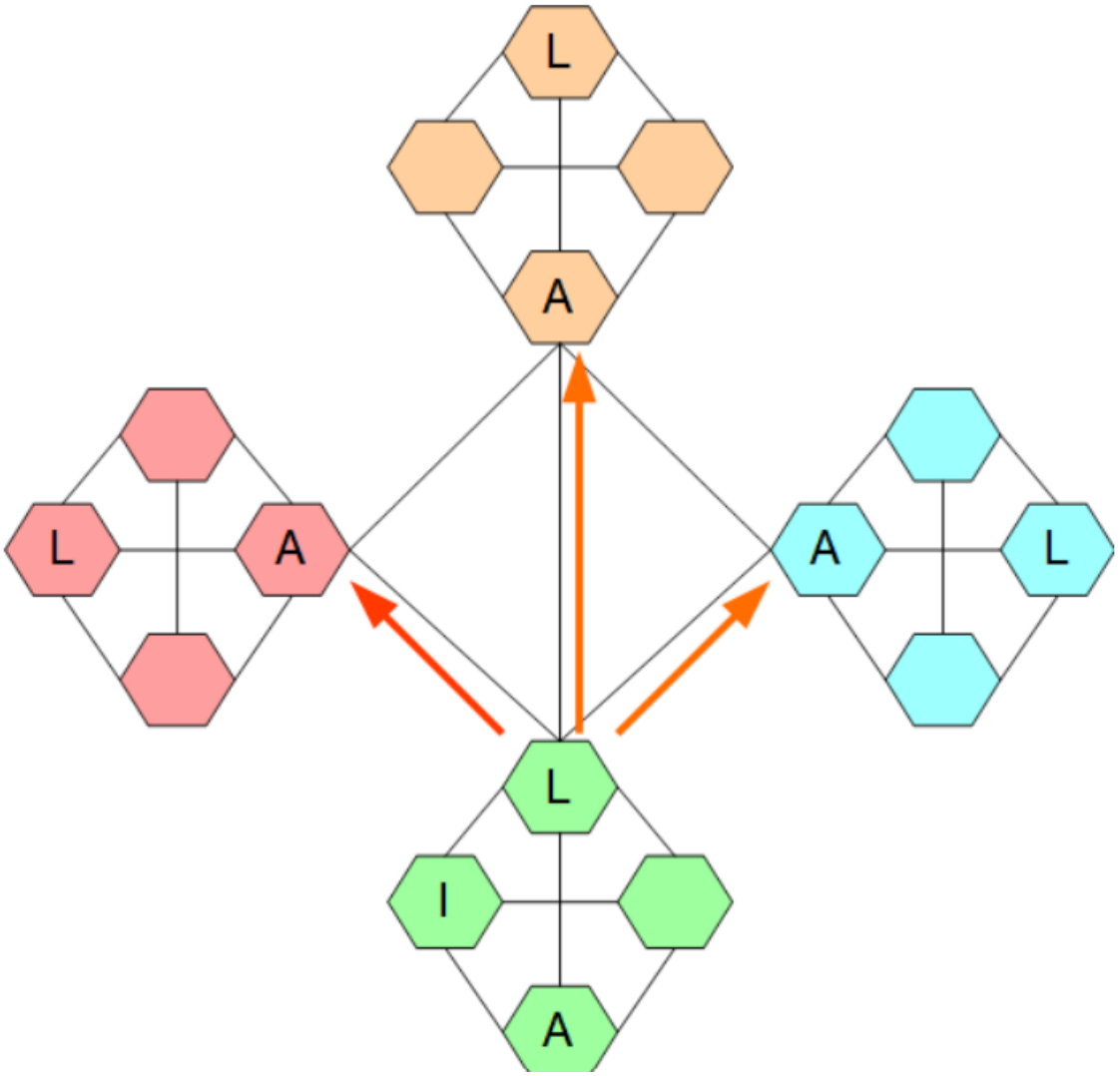

Рисунок 6 – Стадия 3

4. Если арбитры считают, что блок удовлетворяет всем требованиям, они принимают его в свой реестр и отправляют разрешения ожидающим узлам своего кластера (см. Рисунок 7)

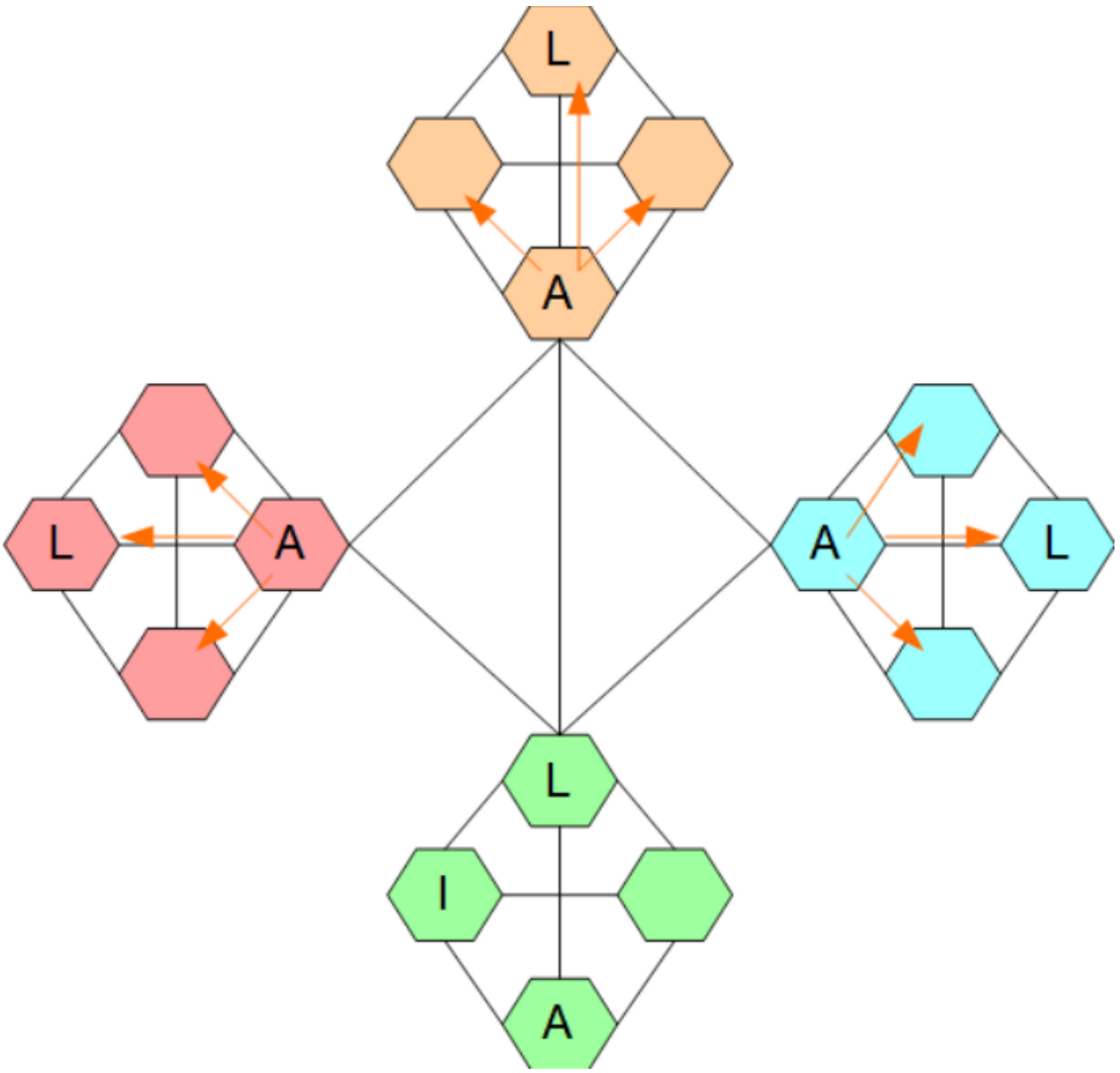

Рисунок 7 – Стадия 4

Глава 3. Тестирование

2.1. Методика тестирования

Набор тестов описывает четыре различные ситуации, которые могут возникнуть во время эксплуатации системы, и не включает преднамеренные атаки на криптографическую защиту или получения злоумышленником контроля над одним или несколькими узлами сети:

1. Проведение отдельных транзакций. Во время этого теста транзакции выполнялись по одной с каждого узла. Каждая следующая транзакция отправлялась только после получения подтверждения о прохождении предыдущей.

2. Проведение потоков транзакций. Транзакции отправлялись наборами по 100 с разных узлов в разных кластерах (через один узел кластера одновременно).

3. Проведение одновременных транзакций из разных кластеров. Как и во втором тесте, транзакции отправлялись наборами по 100 с разных узлов в разных кластерах, но на этот раз потоки были синхронизированы.

4. Тест совершения одновременных транзакций II. Набор из 100 транзакций был отправлен одновременно с каждого узла. Тесты были выполнены в следующих конфигурациях: 1. Один кластер из 6 узлов, 2. Два кластера по 6 узлов, 3. Два кластера по 6 узлов и один из 3, 4. Два кластера по 6 узлов и два по 3, 5. Два кластера по 6 узлов и три по 3.

Следует иметь в виду, что первый и второй кластеры имели по 6 узлов, а третий, четвертый и пятый состояли только из 3 — эта конфигурация была статической и не менялась во время тестирования.

Объяснение того, почему для проведения тестов были выбраны только эти пять конфигураций, связано с несколькими техническими проблемами BGX / DGT, которые налагают ограничения на масштабируемость сети. Вторая причина заключается в том, что сети такого размера было достаточно, чтобы увидеть различия между BGX / DGT и бенчмарком, и установить наиболее важные недостатки

рассматриваемой версии продукта (Kawartha). Для унификации терминологии мы предполагаем, что сеть под управлением сети-бенчмарка также имеет федеративную структуру, поэтому в данных терминах первые шесть узлов будут принадлежать первому кластеру, вторые шесть — второму кластеру, 13, 14, 15 узлов — третьему кластеру. А последние 6 узлов составят четвертый и пятый кластеры соответственно. Бенчмарк и BGX/DGT сравнивались по двум основным показателям: количество подтвержденных транзакций в секунду и средний объем сетевого трафика на валидаторе за транзакцию:

$$CTPS = transactions\ commited\ /\ time\ elapsed\ , \quad (1)$$

$$VONT = traffic\ volume\ /\ transactions\ commited. \quad (2)$$

Где, transactions commited – количество транзакций транзакций, включенных в базы данных на всех узлах,

time elapsed - время, прошедшее с момента отправки первой транзакции до момента подтверждения всех транзакций в тесте.

Объем входящего и исходящего трафика измерялся за время от посылания транзакции до ее подтверждения со стороны валидатора.

Кроме того были проверены реакции BGX/DGT на атаку 51% и Double spending. Проверка на подверженность атакам 51% проводилась с помощью встроенных средств BGX/DGT. Проверка на double spending проводилась следующим образом: сначала создавался bgt кошелек на n токенов, потом посылались две одновременные транзакции на снятие средств суммарно более чем на n токенов, причем каждая транзакция имеет объем не более n токенов.

2.2. Тестовая среда

Аппаратное обеспечение тестовой среды:

- OS: Fedora 28
- CPU: Intel(R) Xeon(R) CPU E5-2690 v4, 2.60GHz x56
- RAM: 255 GB

Виртуализация и моделирование тестовых сценариев проводились средствами Docker[15] и Docker-compose[16]. Измерения проводились с помощью Docker stats, vnStat[17] и встроенными средствами Sawtooth. Состав типичного узла BGX/DGT:

- валидатор
- процессор транзакций
- оболочка
- компонент консенсуса
- процессор настроек
- модуль REST API

Каждый узел запускался в отдельном docker-контейнере. Для сравнения аналогичные тесты были проведены для Hyperledger Sawtooth 1.2.6[18] на алгоритме PBFT, так как этот алгоритм является наиболее близким к F-BFT из всего семейства BFT.

2.3. Результаты тестирования

Согласно результатам (см. Таблицу 1 и Рисунок 8), система BGX/DGT, по сравнению с Hyperledger Sawtooth, является более масштабируемой с точки зрения эффективности потребляемых ресурсов и быстродействия при выполнении одиночных транзакций, но гораздо менее приспособленной к увеличению рабочей нагрузки.

Также BGX/DGT показала себя устойчивой к атакам Double spending, но, в зависимости от конфигурации сети, неустойчивой к атаке 51%. Последнее объясняется прежде всего кластерной топологией сети – чтобы атака прошла, необходимо чтобы было подменено более половины от общего числа кластеров, причем для подмены кластера необходимо чтобы чтобы неверно работали не менее половины от числа узлов в данном кластере. Таким образом, выполнимость атаки 51% зависит от распределения поврежденных узлов внутри каждого кластера.

Таблица 1 – Результаты тестирования

Кол-во узлов	**Sawtooth PBFT**			**BGX/DGT**		
Тест 1. Совершение единичных транзакций						
	CTPS, tps	**VONT, kb/tr**	**CR**	**CTPS, tps**	**VONT, kb/tr**	**CR**
6	0,85	81,2	1	0,62	266	1
15	0,8	168	1	1,20	162,7	1
12	0,83	212,8	1	1,07	207	1
18	0,62	293,1	1	1,09	234,9	1
21	0,87	340	1	1,18	206	1
Тест 2. Совершение потоков транзакций						
6	0,74	66	1	1,23	180,8	1
15	1,85	109,2	1	1,49	185	0,57
12	2,43	121,6	1	1,72	142,6	0,49
18	4,54	108,8	1	1,09	209	0,36
21	5,55	115	1	0,81	232,7	0,28
Тест 3. Совершение одновременных транзакций						
12	0,8	109,2	1	-	-	0
15	3,13	110	1	-	-	0
18	3,8	123,4	1	-	-	0
21	4,76	112,6	1	-	-	0
Тест 4. Совершение одновременных транзакций						
6	0,87	42,5	1	-	-	0
15	1,18	101,6	1	-	-	0
12	2,78	75,8	1	-	-	0
18	3,45	103,8	1	-	-	0

21	3,85	129,9	0,95	-	-	0

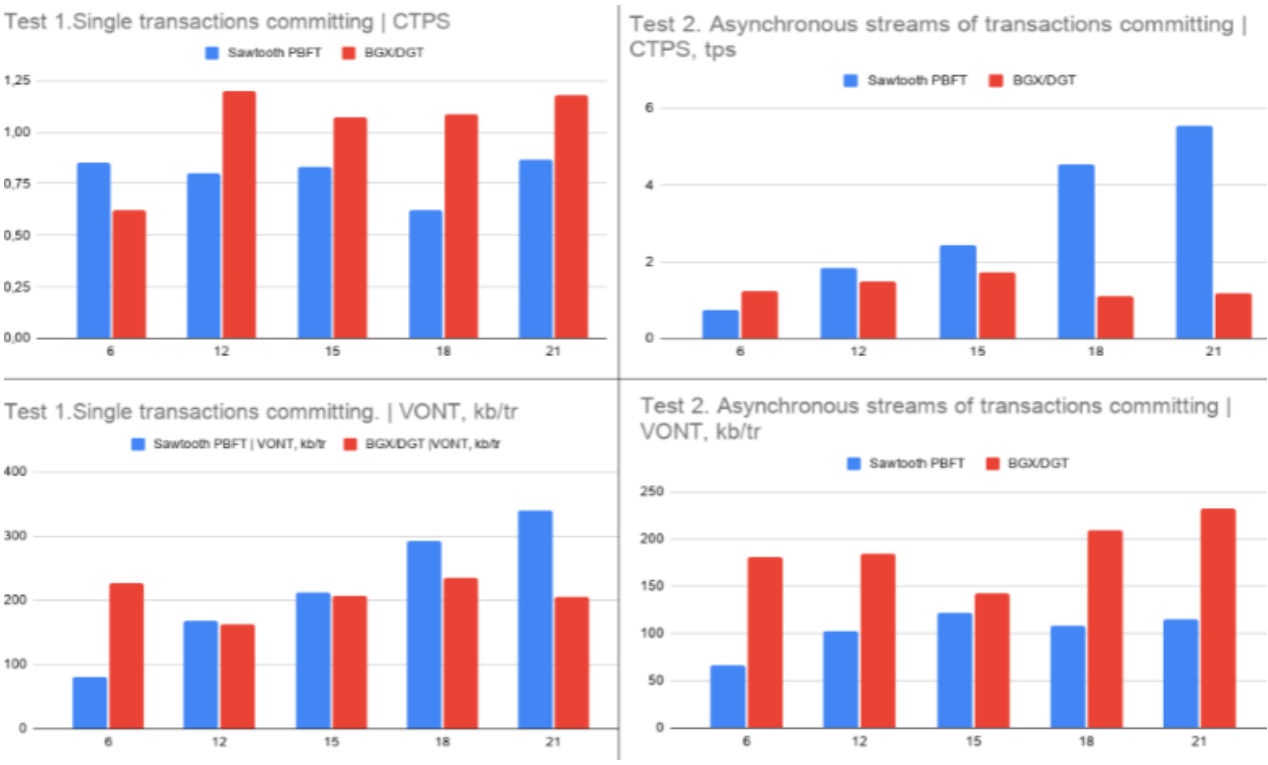

Рисунок 8 – Динамика результатов тестирования

Заключение

В ходе работы была разработана методология тестирования однотипных, в смысле принципов функционирования используемых алгоритмов консенсуса, DLT решений. Разработка методологии включала в себя обзор рассматриваемых систем, формирование тестовых сценариев с учетом особенностей функционирования тестируемых платформ, вывод метрик разносторонне характеризующих эффективность работы системы.

В работе были рассмотрены различные алгоритмы консенсуса и их практические реализации, а также различные виды атак на консенсусы.

Было проведено тестирование двух схожих решений на основе BFT-базированного консенсуса и измерены метрики производительности в различных сценариях работы, на основе чего дана оценка применимости решений в различных типах систем.

Список использованных источников

1. Decentralised Finance: Defining the future of finance [Электронный ресурс]: URL: https://www.pwc.ch/en/insights/digital/defi-defining-the-future-of-finance.html (дата обращения: 16.04.2022)

2. Lamport L., Shostak R., Pease M. The Byzantine generals problem / ACM Transactions on Programming Languages and Systems, 1982. Vol. 4. №6.

3. Lamport, Leslie. Time, Clocks and the Ordering of Events in a Distributed System // Communications of the ACM : journal, 1978. Vol. 21, №7. P. 558–565.

4. Castro M., Liskov B. Practical byzantine fault tolerance and proactive recovery // ACM Transactions on Computer Systems, 2002. Vol. 20, №4.

5. Dwork C.; Naor M. Pricing via Processing, Or, Combatting Junk Mail, Advances in Cryptology. // CRYPTO'92: Lecture Notes in Computer Science, 1993. No. 740. P. 139–147.

6. Shi Z., Zhou H., Hu Y., Jayachander S., Laat C., Zhao Z. Operating Permissioned Blockchain in Clouds: A Performance Study of Hyperledger Sawtooth. // 18th International Symposium on Parallel and Distributed Computing (ISPDC), 2019. P. 50-57.

7. Gilbert S., Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. // SIGACT News, 2019. Vol. 33. №2. P. 51–59.

8.Querci D., Hailes S. Sybil Attacks Against Mobile Users: Friends and Foes to the Rescue // Proceedings IEEE INFOCOM, 2010. P. 1–5.

9. Eskandari, S., Moosavi, S., Clark, J. SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. // Lecture Notes in Computer Science, 2020. Vol 11599.

10. Swathi P., Modi C., Patel D. Preventing Sybil Attack in Blockchain using Distributed Behavior Monitoring of Miners. // 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2019. P. 1-6.

11. Rao, Thanmayee. (2019). 51% Attacks on Cryptocurrencies: A Case Study.

12. Hyperledger Sawtooth documentation [Электронный ресурс]: URL: https://sawtooth.hyperledger.org/docs/core/releases/1.2.6 (дата обращения: 16.04.2022)

13. Introduction to Sawtooth PBFT [Электронный ресурс]: URL: https://sawtooth.hyperledger.org/ docs/pbft/releases/ latest/introduction-to-sawtooth-pbft.html (дата обращения: 16.04.2022)

14. Репозиторий BGX/DGT [Электронный ресурс]: URL: https://github.com/DGT-Network/DGT-Kawartha (дата обращения: 16.04.2022)

15. Docker Engine overview [Электронный ресурс]: URL: https://docs.docker.com/engine

16. Overview of Docker Compose [Электронный ресурс]: URL: https://docs.docker.com/compose

17. vnStat on Wikipedia [Электронный ресурс]: URL: https://en.wikipedia.org/wiki/VnStat

18. Репозиторий Hyperledger Sawtooth [Электронный ресурс]: URL: https://github.com/ hyperledger/sawtooth-core