Санкт-Петербургский государственный университет

Программная инженерия
Кафедра системного программирования

Долголев Филипп Петрович

Разработка системы измерения производительности реализаций blockchain

Бакалаврская работа

Научный руководитель:
ст. преп. Я. А. Кириленко

Рецензент:
руководитель филиала "DSX Technologies" Г. В. Мавчун

Санкт-Петербург
2017

SAINT PETERSBURG STATE UNIVERSITY

Software engineering

Filipp Dolgolev

Blockchain Implementations Benchmarking Tool

Graduation Thesis

Scientific supervisor:
senior lecturer Iakov Kirilenko

Reviewer:
head of branch at "DSX Technologies" George Mavchun

Saint Petersburg
2017

Оглавление

Введение

Технология blockchain представляет собой децентрализованную распределенную базу данных особого вида, обеспечивающую высокую надёжность хранения и добавления данных участниками, не доверяющими друг другу. Начало развитию технологии blockchain положила криптовалюта Bitcoin [3] в 2008 году.

С тех пор всё больше и больше финансовых и технологических компаний обращают внимание на blockchain, разрабатывают свои вариации, проводят эксперименты, внедряют в бизнес, создают консорциумы и вступают в них для совместных исследований этой технологии. Примерами таких консорциумов являются Hyperledger [8] и R3CEV [16]. Интерес проявляют даже государственные структуры. Например, Шведский земельный реестр [13] занимается тестированием blockchain для проведения сделок с землей и недвижимостью. Всё это показывает актуальность технологии blockchain.

За последние годы было представлено множество различных реализаций blockchain: Ethereum [7], Hyperledger Fabric [9], Nxt [15], Multichain [14], Eris [6] и другие. Но при выборе той или иной реализации возникает вопрос, сможет ли она обеспечить определенный уровень производительности для решения конкретной задачи. Более того, поведение даже одной и той же реализации при различных настройках может отличаться, что ставит вопрос о нахождении подходящих параметров.

На момент начала данной работы, в свободном доступе не было найдено инструмента, позволяющего ответить на вышепоставленные вопросы касательно различных реализаций, что послужило причиной создания проекта Blockchain Benchmarking, посвященного изучению производительности различных реализаций blockchain.

В рамках этого проекта были выделены следующие задачи:

- Разработка системы автоматизированного нагрузочного тестирования реализаций blockchain.
- Разработка универсального API для взаимодействия с реализациями blockchain.
- Анализ производительности реализаций blockchain.

Однако стоит отметить, что недавно появилась работа [2], посвященная анализу производительности реализаций blockchain, что также подчёркивает актуальность данной проблемы. При этом, наш подход предоставляет более подробные данные о работе blockchain, на основе которых можно построить множество метрик.

1. Постановка задачи

Целью данной работы является создание системы, позволяющей автоматизированно проводить нагрузочное тестирование реализаций blockchain и собирать данные, которые в дальнейшем можно использовать для анализа производительности и влияющих на неё факторов.

Для достижения этой цели были поставлены следующие задачи.

- Провести анализ существующих реализаций blockchain.
- Разработать независимый от конкретной реализации blockchain метод нагрузочного тестирования для получения данных, позволяющих провести анализ производительности.
- Разработать архитектуру системы нагрузочного тестирования для реализаций blockchain.
- Реализовать прототип системы и предоставить примеры её использования.

2. Обзор

2.1. Blockchain

Технология blockchain в первую очередь ориентирована на обеспечение высокого уровня надёжности хранения и валидации данных среди множества участников, не доверяющих друг другу. Достигается это за счёт децентрализации принятия решения о добавлении новых записей и хранением локальной копии истории множеством участников.

Также, реализации blockchain могут поддерживать работу умных контрактов. Основная идея умных контрактов заключается в следующем: при наступлении определенных условий, автоматически выполняется алгоритм заданый в контракте. Например: при появлении записи о переводе некоторой суммы денежных средств от пользователя A пользователю B, происходит запись о передаче права на владение некоторой недвижимостью от пользователя B пользователю A.

2.2. Инструменты нагрузочного тестирования

Существующие инструменты для нагрузочного тестирования, находящиеся в свободном доступе, в силу специфики работы технологии blockchain, не подходят для поиска ответов на поставленные в начале вопросы. На примере JMeter [10] и YCSB [4] были изучены возможности существующих инструментов для нагрузочного тестирования и сделан вывод о сомнительности их применения в контексте blockchain. Основным препятствием для их применения является децентрализованность blockchain, так как эти инструменты полагаются на централизованность тестируемой системы. Проблема возникает в определении времени принятия записи сетью blockchain. Причиной является то, что время запроса на добавление записи напрямую не связано со временем принятия записи сетью blockchain, более того, запись принимается конкретным узлом сети, и единственный способ узнать о появлении новой записи на конкретном узле со стороны пользователя blockchain - сделать запрос к этому узлу.

Из-за того, что существующие инструменты нагрузочного тестирования не предоставляют механизм, с помощью которого можно было бы узнать время принятия записи каждым узлом, применение подобных инструментов в констекте blockchain ограничено областью генерацией нагрузки. Однако, и здесь было решено их не использовать по следующим причинам:

- Затраченное время на их интеграцию в систему оказалось бы не меньше времени написания собственного модуля генерации нагрузки.

- В дальнейшем планируется сделать генерацию нагрузки динамической, для автоматизированного поиска предельного уровня нагрузки, при котором поведение сети blockchain было бы стабильным. Данные инструменты не располагают необходимой функциональностью для решения данной задачи.

3. Анализ реализаций blockchain

Нами были рассмотрены следующие реализации: Bitcoin, Hyperledger Fabric, Ethereum, Nxt, Multichain. Как результат, мы выявили следующие особенности работы рассмотренных реализаций.

3.1. Общий механизм работы

Основная идея добавления новых записей заключается в следующем: формируется новый блок, в который включаются эти записи и хеш предыдущего блока. Хеш каждого блока зависит от записей и различной служебной информации. Далее этот блок рассылается участникам сети, которые должны его провалидировать и, в случае корректности, записать в локальную копию. Таким образом, в процессе работы получается цепочка блоков, содержащих в себе записи. Хранение в каждом блоке хеша предыдущего блока и зависимость хеша от записей обеспечивают целостность и защиту от подмены или удаления записей, т.к. практически невозможно изменить содержимое блока таким образом, чтобы не изменился хеш блока.

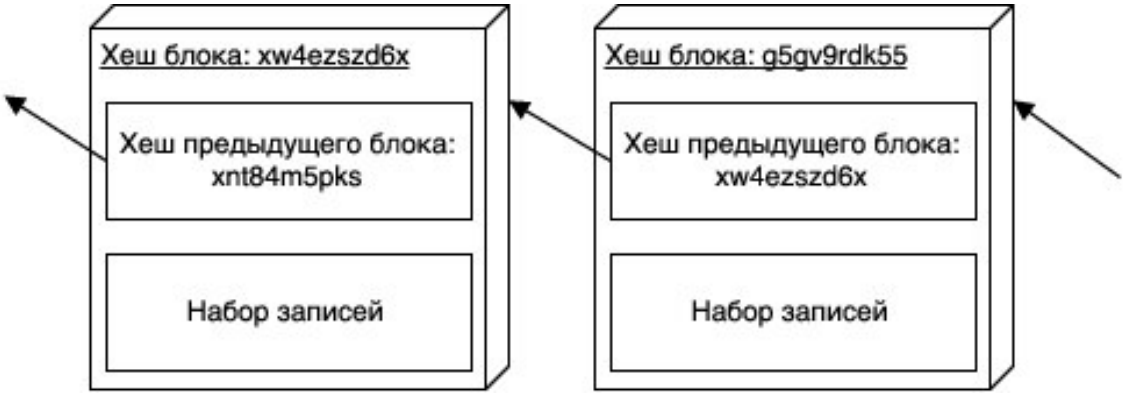

Рис. 1: Цепочка блоков

3.2. Различия

- Механизм добавления новых блоков. В вышеперечисленных реализациях используются вариации следующих трех подходов:
  - Proof-of-work [12]. В этом случае право на публикацию нового блока получает тот участник, который решил определенную трудоемкую задачу и предоставил ответ. Минусом этого способа является требование больших вычислительных мощностей для решения задачи. Такой подход используется в Bitcoin.
  - Proof-of-stake [17]. При использовании этого подохода, вероятность получить право на публикацию блока связана с количеством ресурса на счёте участника, что куда менее затратно по сравнению с proof-of-work, но менее безопасно. Примером использования этого подхода может служить платформа Nxt.
  - Решение задачи византийских генералов [18]. При данном подходе, решение о создании нового блока принимается путем достижения консенсуса некоторым количеством участников на основе алгоритмов решения задачи византийских генералов. Минусом этого подхода является проблема выбора участников для достижения консенсуса. Этот подход используется например в Hyperledger Fabric.
- Структура блока. Отличия заключаются в основном в хранении различной служебной информации.
- Механизм работы умных контрактов. Рассмотренные реализации поддерживают работу умных контрактов, но способ их описания и публикации кардинально отличается.

4. Нагрузочное тестирование реализаций blockchain

4.1. Метод нагрузочного тестирования

Blockchain – распределенная база данных специального вида. У неё (с точки зрения распределенной базы данных) есть две, важных с нашей точки зрения, особенности:

- Основная модель данных – это последовательность записей, которые могут только добавляться.
- Все полноценные узлы сети содержат всю базу целиком (что не характерно для распределенных баз данных).

Именно исходя из этих двух особенностей мы можем предложить следующий метод нагрузочного тестирования blockchain: задавая разную конфигурацию сети и разную нагрузку, фиксировать время распространения записей по узлам сети – набор этих времен будет основным результатом, на основе которого будут вычисляться различные метрики.

Далее, поскольку записи объединяются в блоки, общие для всей сети, нам достаточно отслеживать время появление каждого блока на каждом узле, а по завершении тестирования посмотреть распределение записей по блокам.

Умные контракты в рамках тестирования решено не рассматривать, в основном по причине сильных отличий по взаимодействию с ними в различных реализациях blockchain, и как следствие, сложности обобщения этого взаимодействия.

В результате получаем следующие данные.

- Идентификатор записи – время появления этой записи в blockchain.
- Идентификатор записи – идентификатор блока, в который была включена эта запись.

- Идентификатор блока – время появления этого блока на узле.
- Время – потребление ресурсов компьютеров.

На основе перечисленных данных можно воспроизвести некоторые интересующие метрики, например, время задержки распространения записи на заданный процент узлов или же количество принятых записей в единицу времени.

4.2. Ограничения

Нужно отметить, что общее состояние сети blockchain в некоторые моменты времени может быть неконсистентно, и, спустя время, некоторые последние принятые блоки могут быть признаны невалидными вместе с содержащимися записями.

Наш подход не акцентирует внимание на таких ситуаций и не позволяет собрать данные, необходимые, например, для вычисления времени, в течении которого на некоторых узлах были видны записи, в дальнейшем не принятые сетью blockchain.

В случае возникновения необходимости в получении подобных данных, достаточно будет также сохранять в процессе работы идентификаторы всех записей для каждого блока. Также на данный момент не рассматриваются ситуации, с отключением/подключением узлов blockchain в процессе работы. Вышеперечисленные ситуации могут быть интересны при анализе устойчивости и надёжности blockchain, но в данной работе мы сосредоточены именно на оценке производительности при штатной работе.

4.3. Параметры тестирования

Данный метод тестирования предполагает следующие входные параметры:

- Количество узлов blockchain.

- Количество записей — сколько записей необходимо будет добавить на каждый узел blockchain.

- Распределение времени задержки между записями — тип распределения (нормальное, равномерное, экспоненциальное) времени задержки между публикациями записей на один из узлов и характеризующие его параметры.

- Задержка между запросами к узлу о появлении новых блоков — временная точность, с которой мы хотим наблюдать появление новых блоков. В общем случае узел не может нас уведомлять о появлении блока, поэтому мы вынуждены сами с некоторой периодичностью его опрашивать.

- Распределение размера записей — тип распределения (нормальное, равномерное, экспоненциальное) размера записей в байтах и характеризующие его параметры.

- Количество нагрузочных узлов и степень параллелизма - сколько узлов генерируют нагрузку, и сколько потоков приходится на каждый узел blockchain.

Задание распределений времени задержки между записями и размера записей вместо конкретных фиксированных величин позволит в некоторой степени приблизиться к реальным ситуациям использования blockchain.

4.4. Сравнение с BLOCKBENCH

Ближе к завершению нашей работы мы обнаружили статью [2] от 12 марта, посвященную анализу производительности и поиску узких мест в работе различных реализаций blockchain. В данной статье рассматриваются следующие реализации: Hyperledger Fabric, Ethereum и Parity. Анализируется производительности сети blockchain в целом, а так же эффективность среды выполнения умных контрактов.

В рассматриваемой статье, с нашей точки зрения, предложенный метод для тестирования производительности сети blockchain, в сравнении с нашим, обладает следующим недостатком: определение момента окончательного принятия записи возлагается на реализацию blockchain. Например, в случае с Ethereum, полагаются на внутренний параметр, количество принятых блоков после того, в который вошла рассматриваемая запись, а в случае Hyperledger Fabric запись считается подтвержденной в тот момент, как только сгенерирован блок с ней на какомнибудь узле. Более того, время принятия записи берётся единное для всех узлов, и связано оно только со временем генерации соответствующего блока на узле, сгенерировавшем этот блок.

На наш взгляд, такой подход не достаточно точно отражает время добавления новой записи в сети blockchain, к тому же не позволяет получить информацию о распространении блока по узлам. Происходит это потому, что в момент генерации блока узлом, остальная сеть ещё о нём ничего не знает, и соответсвенно, о записях, в него вошедших, и лишь спустя некоторое время, данный блок распространится по сети. Мы считаем что будет более корректно говорить о времени появления блока на некоторой доле узлов сети. Вопрос о конкретной величене этой доли хоть и остаётся открытым, но такое время является куда более приближенным к реальности. Именно поэтому мы фиксируем время появления блоков независимо на каждом узле, и это является преимуществом нашего подхода. На основе данных времен также можно анализировать скорость распространения блоков по сети.

5. Архитектура

С учетом предложенного выше процесса тестирования была разработана и реализована модульная архитектура, где каждый модуль является отдельным java приложением. На данный момент для взаимодействия с blockchain используется реализованная вне рамках этой работы библиотека, предоставляющая универсальный интерфейс к некоторому подмножеству операций нескольких реализаций blockchain.

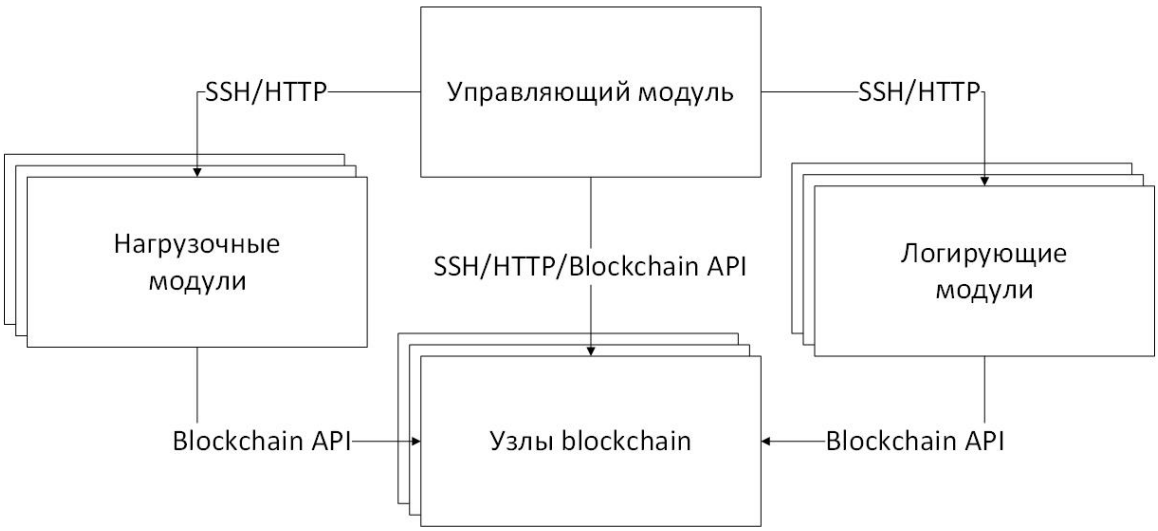

Рис. 2: Архитектура системы

5.1. Управляющий модуль

Занимается инициализацией сети blockchain, запуском и управлением процесса тестирования, сбором полученных данных. Для подготовки окружения на компьютерах и запуска теста используется библиотека JSch [11], которая позволяет вызывать команды и загружать файлы по протоколу ssh. На вход подаётся файл с настройками тестирования, среди которых нужно указать путь к файлу с ip адресами компьютеров, а так же путь к папке, содержащей необходимые файлы и скрипты для инициализации сети blockchain. Содержимое этой папки будет доставлено на указанные компьютеры для последующей инициализации.

Процесс инициализации сети blockchain происходит в два этапа:

- Инициализация начального узла. Это действие необходимо в первую очередь для того, чтобы при инициализации остальных узлов, они могли узнать друг о друге через некоторый общий узел. Кроме этого, на этом узле можно провести первоначальную настройку сети blockchain и создать аккаунты/кошельки, с использованием которых будет проходить добавление записей. Файлы, появившиеся в результате инициализации этого узла, будут переданы остальным узлам для последующей их инициализации.

- Инициализация всех остальных узлов.

Далее происходит запуск процесса тестирования. Для этого будет выполнена загрузка и запуск экземпляров логирующего модуля на все компьютеры с узлами blockchain. На каждый узел blockchain выделяется отдельный экземпляр логирующего модуля для того, чтобы уменьшить минимально возможное значение периода опроса узла. В противном случае возникает вопрос о выборе такого количества экземпляров логирующего модуля, чтобы они за указанный период времени успевали опросить все узлы о появлении новых блоков. Затем на указанное в параметрах количество компьютеров будет загружен и запущен нагрузочный модуль. После этого, управляющий модуль будет ожидать сигнала от экземпляров логирующих и нагрузочных модулей о завершении тестирования. Так же можно задать максимальное время ожидания.

Как только будут получены сигналы о завершении теста или пройдёт максимальное время ожидания, запустится процесс сбора данных, появившихся на компьютерах с логирующими и нагрузочными модулями, а затем для каждого блока в сети blockchain вытащится список вошедших в него записей.

В качестве результата работы будет сформирована папка со следующим содержимым:

- Папка, содержащая для каждого узла blockchain соответствующий файл, в котором указано время запроса на добавление записи и ее идентификатор.

- Папка, содержащая для каждого узла blockchain соответствующий файл, в котором указано время появления каждого блока на этом узле.

- Папка, содержащая для каждого компьютера соответствующий файл, в котором указано время и информация об использованных ресурсах (использование CPU, объем использованной оперативной памяти, объём использованного сетевого трафика).

- Файл, содержащий идентификаторы всех записей в каждом блоке.

5.2. Логирующий модуль

На вход принимает адрес узла, по которому будет узнавать о появлении блоков и период опроса узла. Если при очередном запросе выяснится, что появился новый блок, то для этого блока сохранится время ответа как время появления этого блока на узле. Также на вход принимается время, по истечению которого если не появилось новых блоков или появлялись только пустые блоки (такой случай возможен например в Ethereum), то посылается на управляющий модуль сигнал о завершении работы.

5.3. Нагрузочный модуль

На вход принимается тип распределения размера записей, ip адреса и порты узлов blockchain, количество записей для одного узла blockchain, тип распределения размера задержки между записями, количество потоков для одного узла. Модуль производит записи согласно переданным параметрам, а ответ на каждую запись получает ее идентификатор. Время ответа записывается как время совершения записи в blockchain.

6. Апробация

Данная система была апробирована на следующих реализациях: Hyperledger Fabric, Ethereum. Эти реализации были выбраны из-за наличия большого сообщества, обширной кодовой базы и повышенного к ним интереса со стороны индустрии. Например, запуская 4 узла Etherum и Hyperledger Fabric на 4 экземплярах AWS EC2 [1] t2.mediun (2 ядра, 4гб оперативной памяти), мы получили следующий график:

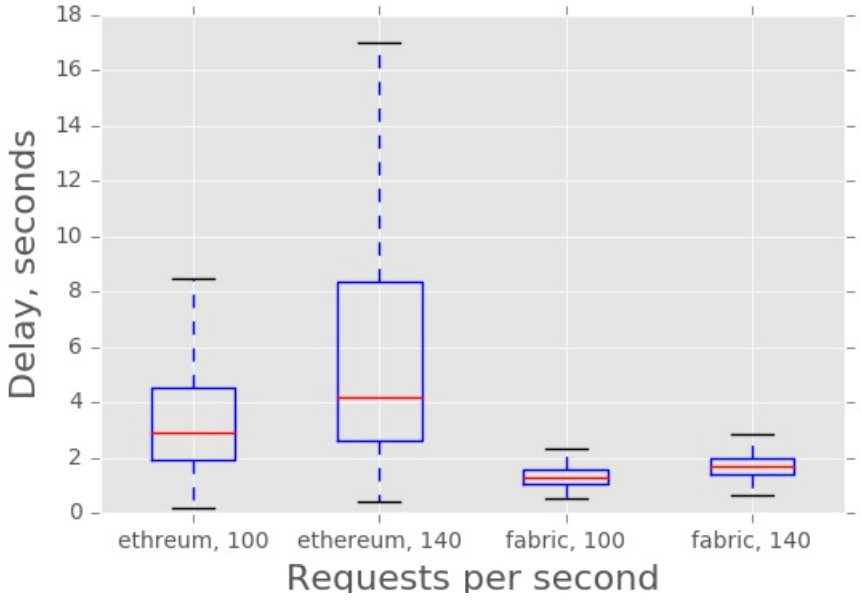

Рис. 3: График распределения времени задержки принятия записи 4 узлами Ethrerum и Hyperledger Fabric в зависимости от нагрузки

Заключение

В рамках данной работы были достигнуты следующие результаты:

- Проведен анализ существующих реализаций blockchain и выделен общий механизм работы этих реализаций.
- Предложен независимый от конкретной реализации blockchain метод нагрузочного тестирования для получения данных, позволяющих провести анализ производительности.
- Разработана архитектура системы для нагрузочного тестирования реализаций blockchain.
- Реализован прототип системы и представлены примеры использования для следующих реализаций blockchain: Hyperledger Fabric, Ethereum.
- Результаты данной работы представлены на конференции SEIM 2017.

В дальнейшем планируются следующие шаги:

- Добавить примеры использования данной системы для следующих реализаций: MultiChain, Bitcoin, Chain [5]
- Расширить систему для изучения устойчивости и надёжности реализаций blockchain
- Автоматизировать поиск предельной нагрузки, при которой сеть в целом остаётся стабильной.

Список литературы

[1] Amazon Web Services, EC2. — 2017. — URL: https://aws.amazon.com/ec2/ (online; accessed: 10.03.2017).

[2] BLOCKBENCH: A Framework for Analyzing Private Blockchains. — 2017. — URL: https://arxiv.org/abs/1703.04057 (online; accessed: 15.05.2017).

[3] Bitcoin. — 2017. — URL: https://www.bitcoin.com/ (online; accessed: 10.03.2017).

[4] Brian F. Cooper Adam Silberstein Erwin Tam Raghu Ramakrishnan Russell Sears. Benchmarking cloud serving systems with YCSB. — 2010. — P. 143–154. — URL: http://dl.acm.org/citation.cfm?id=1807152.

[5] Chain. — 2017. — URL: https://chain.com/ (online; accessed: 10.03.2017).

[6] Eris. — 2017. — URL: https://monax.io/ (online; accessed: 10.03.2017).

[7] Ethereum. — 2017. — URL: https://www.ethereum.org/ (online; accessed: 10.03.2017).

[8] Hyperledger. — 2017. — URL: https://www.hyperledger.org/ (online; accessed: 10.03.2017).

[9] Hyperledger Fabric. — 2017. — URL: https://hyperledger-fabric.readthedocs.io/en/latest/ (online; accessed: 10.03.2017).

[10] JMeter. — 2017. — URL: http://jmeter.apache.org/index.html (online; accessed: 10.03.2017).

[11] JSch. — 2017. — URL: http://www.jcraft.com/jsch/ (online; accessed: 10.03.2017).

[12] Jakobsson Markus, Juels Ari. Proofs of Work and Bread Pudding Protocols. — 1999. — URL: https://www.emc.com/emc-plus/rsa-labs/staff-associates/proofs-of-work-protocols.htm.

[13] Lantmateriet. — 2017. — URL: https://www.lantmateriet.se/en/Nyheter-pa-Lantmateriet/lantmateriet-har-tittat-pa-blockkedjetekniken/ (online; accessed: 10.03.2017).

[14] Multichain. — 2017. — URL: http://www.multichain.com/ (online; accessed: 10.03.2017).

[15] Nxt. — 2017. — URL: https://hyperledger-fabric.readthedocs.io/en/latest/ (online; accessed: 10.03.2017).

[16] R3CEV. — 2017. — URL: http://www.r3cev.com/ (online; accessed: 10.03.2017).

[17] Wikipedia. Proof-of-stake // Википедия, свободная энциклопедия. — 2017. — URL: https://en.wikipedia.org/wiki/Proof-of-stake (online; accessed: 10.03.2017).

[18] Wikipedia. Byzantine fault tolerance // Википедия, свободная энциклопедия. — 2017. — URL: https://en.wikipedia.org/wiki/Byzantine_fault_tolerance (online; accessed: 10.03.2017).