ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ «МИСИС»

На правах рукописи

ХАММУД Обадах

Модели и алгоритмы автономного распределения данных и управления доступом на базе смарт-контрактов

Специальность 2.3.1 – Системный анализ,
управление и обработка информации, статистика

Диссертация на соискание ученой степени кандидата технических наук

Научный руководитель
к.т.н. Тарханов И.А.

Москва 2024

Содержание

ВВЕДЕНИЕ

Актуальность

Информационные системы обмена данными на основе блокчейн (DApps) становятся все популярнее. По данным DApps Industry Report 2023 количество пользователей таких систем в течении только одного года выросло в 2 раза и достигло 4 миллиона, а капитализация рынка DApps приложений превысила 100 миллиардов долларов. Особое развитие такого рода системы получили в финансовом секторе и индустрии игр. При этом случаи внедрения блокчейн в государственных системах также увеличиваются. Основным преимуществом DApps по сравнению с централизованными системами является то, что они позволяют организовать доверенный обмен информаций между участниками, которые изначально друг другу не доверяют. Но очевидны и проблемы таких систем – это избыточность данных, скорость их обработки, а также в некоторых случаях повышенное энергопотребление. Именно необходимость многократно дублировать данные хранящиеся в блокчейн на всех узлах системы является существенным ограничением для внедрения DApps в реальных секторах экономки.

Требования к обработке данных в современных информационных системах со временем резко возрастают по мере увеличения масштабов этих систем и размера ежедневно генерируемых ими данных. В случае распределенных систем возникает необходимость балансировки нагрузки между узлами системы, чтобы избежать проблем с увеличением нагрузки на отдельные узлы системы, количества данных хранящихся в ней, а также количества узлов, где эти данные могут хранится.

Равномерное распределение данных является сложной задачей, поскольку объекты имеют разные размеры и наличие большого количества разнородных объектов может привести к высоким требованиям к ресурсам всех узлов системы (загрузке процессора, количестве выделяемой для хранения памяти на носителях),

а также к пропускной способности между узлами. Известные научные подходы и распространенные облачные платформы не позволяют балансировать нагрузку без многократного дублирования данных на узлах одновременно с сохранением возможности увеличения узлов системы и количества объектов в ней (свойство масштабируемости системы).

Основным элементом DApps являются смарт-контракты, которые могут выполняться на узлах блокчейна и по сути являются самоисполняющимися программами. Именно с помощью смарт-контрактов возможно организовать балансировку нагрузки без участия пользователя. Однако в научной литературе не описаны подходы, которые решали бы такую задачу. В первую очередь это обусловлено наличием технических ограничений языка программирования смарт-контрактов.

Необходимо отметить, что, как и в любой информационной системе, в DApps нужно управлять доступом к объектам системы. В случае распределённой системы важно тратить на это минимум ресурсов при увеличении количества объектов и узлов в ней, а также сохранить устойчивость от сбоев, т.е. она не должна содержать централизованных компонент. Наиболее распространенные модели контроля доступа в информационных системах (DAC, RBAC) являются гибкими и удобными для администрирования доступа к объектам, но по сути являются централизованными и не подходят для использования в DApps.

В данной работе предлагаются алгоритмы и модели распределения данных, которые позволяют реализовать с их помощью DApps и одновременно обойти противоречие в избыточности данных. Также работа посвящена созданию модели управления доступом к данным, которая минимизирует размер хранилища данных прав доступа к объектам DApps, обеспечивая при этом способность увеличивать количество узлов и общий объем данных системы без ограничений.

Важным аспектом внедрения DApps помимо проблемы избыточности данных является оценка целесообразности внедрения такого рода систем по сравнению с системами централизованной архитектуры. Основным критерием здесь является обеспечение определенного уровня надежности (не хуже чем в

централизованных системах) при минимуме ресурсов. Для корректного сравнения надежности распределенных DApps и централизованных систем необходимо разработать новый подход к расчету надежности, который учитывает особенности работы балансировщика нагрузки, что является одной из задач данной работы.

Рассматриваемый в данной работе класс информационных систем не подходит для задач хранения объектов, требующих непосредственной обработки внутри DApps (прямое изменение данных внутри системы, запись видео непосредственно на диски системы и т.д). Все операции чтения, записи и удаления объектов осуществляются только через балансировщик нагрузки, а размер файла должен быть заранее известен и не должен постепенно изменяться, так как размер файла является важным параметром для выбора местоположения файла.

В работе решается научная задача создания системного подхода для разработки DApps систем, который позволяет избавиться от избыточности данных в распределённой среде при этом обеспечить равномерное распределение данных по узлам системы, необходимую надежность системы (не хуже, чем в централизованных системах).

Целью работы

Целью работы является разработка моделей и алгоритмов управления распределением данных и контроля доступа в децентрализованной информационной системе, которые отличаются от известных тем, что:

- обеспечивают равномерную автономную балансировку нагрузки при распределении данных для любого заданного количества узлов с помощью смарт-контрактов,

- позволяют минимизировать размер хранилища на 25% по сравнению с информационными системами с полным резервным копированием каждого узла и с надежностью в целом не хуже, чем в информационных системах с централизованной архитектурой.

Идея работы заключается в сочетании гибридной архитектуры системы, в которой часть данных хранится в блокчейн, а часть в локальных хранилищах

узлов системы. Управление распределением данных реализуется набором алгоритмов, созданных для автономного выполнения в среде смарт-контрактов блокчейн. Для минимизации ресурсов применяется метод стирающего кодирования (Erasure Coding), а также алгоритмы уменьшения размера данных и кэширования запросов, позволяющие управлять доступом к объектам в распределенной системе.

Задачи:

- Исследование параметров эффективности функционирования известных методов распределения данных, таких как общий размер необходимого дискового пространства для хранения объектов и их резервных копий, возможностей увеличения количества узлов, равномерность распределения данных по узлам, надежность системы в целом.

- Выявление ограничений известных моделей контроля доступа, которые можно применять в распределенных системах, и способов снятия этих ограничений.

- Разработка модели распределения данных, которая уменьшает использование памяти на узлах в условиях динамического изменения топологии сети и учитывает ограничения, накладываемые смарт-контрактами.

- Разработка распределенной модели управления доступом, решающей проблему увеличения количества узлов и объектов хранения для DApps систем без использования централизованных компонентов.

- Разработка модели расчета надежности для оценки предложенной модели в определенном состоянии и сравнения с другими подходами.

Методы исследования включают теоретико-множественные методы системного анализа для разработки предлагаемой модели, принципы распределения вероятностей и теории вероятностей, модели надежности, теорию игр, теорию принятия решений, методы математического моделирования, теоретико-информационный анализ систем хранения и распространения файлов, а также систем контроля доступа. Принципы статистики также используются для оценки полученных результатов и сравнения с другими системами.

Объектом исследования являются распределенные информационные системы, которые решают задачу конфиденциального обмена данными между участниками, которые не доверяют друг другу.

Предметом исследования являются системные показатели эффективности функционирования децентрализованных систем и методы управления распределением данных, а также технологические ограничения применения данных методов при работе с блокчейн.

Научные положения, выносимые на защиту:

1) Разработанная модель распределения данных в DApps и 4 алгоритма балансировки нагрузки позволяют организовать равномерное распределение данных по узлам и уменьшить суммарный объем необходимого места на всех узлах на 25% по сравнению с системами полного резервного копирования.

2) Предложенная модель и алгоритмы контроля доступа позволяют минимизировать размер данных хранящихся в блокчейн и избежать их централизованного хранения, что решает задачу увеличения количества узлов и объектов в системе без ущерба для надежности и производительности.

3) Разработанная модель надежности учитывает автоматическое восстановление данных на основе количества узлов, доступного пространства на узлах и времени передачи данных, что позволяет корректно рассчитывать надежность распределённых систем на основе блокчейн и сравнивать их с системами другой архитектуры.

Новизна научных исследований заключается в следующем:

1) Предложены модель распределения данных и оперирующие этой моделью 4 алгоритма балансировки нагрузки, которые минимизируют избыточность данных и отличаются от известных тем, что гарантируют справедливое распределение данных между узлами, а также обеспечивают возможность автоматического восстановления данных на узлах за счет того, что сочетают метод стирающего кодирования, распределение по виртуальным кластерам и реализацию алгоритмов балансировки в среде смарт-контрактов в автономном режиме.

2) Предложены новая модель и алгоритмы управления доступом в распределенной системе, базирующиеся на известной DAC и RBAC моделях, которые в отличие от известных моделей разграничения доступа минимизируют необходимое пространство для хранения на балансировщике нагрузки и распределяют данные по узлам таким образом, чтобы поддерживать увеличение количества узлов и хранящихся в системе объектов на основе кэширования.

3) Предложена новая модель оценки надежности распределенных систем, основанная на модели оценки надежности для RAID и отличающийся от известных тем, что учитывает возможность восстановления потерянных данных на существующих узлах на основе информации о свободной памяти на узлах, количестве узлов и времени восстановления данных в системе.

Обоснованность и достоверность результатов исследования обеспечиваются: репрезентативностью исходных данных для моделирования поведения системы основанной на данных полученных из реального проекта коммерческой компании; использованием современного программного обеспечения, оборудования и апробированных методик для оценки надежности и моделирования.

Практическая значимость:

Предложенные модели, включая модель распространения данных, а также модель контроля доступа, могут применяться в государственных и корпоративных системах многих отраслей экономики:

1) Для государственных систем предложенные модели могут быть полезны в сценариях, когда необходимо организовать проверку и аудит со стороны государства деятельности компаний, где существуют ограничения по ресурсам и требования к надежности. Например, контроль лицензирования различных видов деятельности (финансы, торговля требующими государственного регулирования товарами, оказание эксклюзивных услуг), сбор налоговых пошлин, предоставление финансовой отчетности для получения льгот и т. д.

2) Предложенные модели и алгоритмы могут быть использованы в

качестве инфраструктуры облачных сервисов для корпоративного обмена данными, которые требуют хранения больших объемов файлов транзакций, изображений, аудио и видео библиотек. Они могут лечь в основу нового поколения распределённых систем для хранения любых информационных объектов или выполнять функции распределённой базы данных.

3) Предложенные модели также могут использоваться в DApps сервисах ориентированных на конечных пользователей. Например, в сервисах управления и предоставления доступа к данным обучения нейронных сетей, NFT системам, в которых хранятся файлы, метаданные файлов или сложные структурированные объекты.

Реализация выводов и рекомендаций работы

Основные положения диссертации прошли апробацию в проектах ООО «РИИТ». Подсистема распределённого хранения подключена к программной платформе «Рубикон», которая обеспечивает защищенный обмен данными между организациями, что подтверждается соответствующим актом внедрения. Зарегистрирована программа ЭВМ «DecStore: балансировщик нагрузки на базе смарт-контракта для распределённого хранения данных», регистрационный номер 2023682504 от 26.10.2023.

Публикации. Материалы диссертации опубликованы в 10 научных работах, в том числе в 3-х рекомендованных ВАК РФ и одна работа из перечня изданий, индексируемых в международных библиографических базах данных Scopus и Web of Science.

Соответствие паспорту специальности.

Соответствует П5 «Разработка специального математического и алгоритмического обеспечения систем анализа, оптимизации, управления, принятия решений, обработки информации и искусственного интеллекта», т.к. одной из задач диссертации является разработка программного и алгоритмического обеспечения системы распределения данных (обработки информации).

Соответствует П9 «Разработка проблемно-ориентированных систем

управления, принятия решений и оптимизации технических объектов», т.к. в диссертации решается задача разработки модели управления доступом в децентрализованной системе.

Соответствует П11 «Методы и алгоритмы прогнозирования и оценки эффективности, качества, надежности функционирования сложных систем управления и их элементов», т.к. в диссертации решается задача оценки надежности функционирования систем на основе блокчейн.

Объем и структура диссертации. Диссертация состоит из введения, 4 глав, заключения, библиографического списка из 136 наименований и представлена на 148 страницах, включая 43 рисунков, 16 таблиц.

1. ПРОБЛЕМЫ ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ РАСПРЕДЕЛЕНИЯ ДАННЫХ И УПРАВЛЕНИЯ ДОСТУПОМ К НИМ В ДЕЦЕНТРАЛИЗОВАННЫХ СИСТЕМАХ

1.1 Информационно-теоретический анализ эффективности распределённых систем хранения данных

В эпоху цифровой трансформации правительствам и крупным компаниям необходимо обмениваться конфиденциальными данными. Этот процесс является частью развития электронного правительства во многих странах, например, когда компании предоставляют государственным органам отчеты для оценки налогов или лицензирования [96] [24]. Крупные корпорации и финансовые учреждения разрабатывают собственные отраслевые решения для распределенного и безопасного обмена данными. Электронные документы составляют для этих корпораций важную часть ежедневного обмена данными B2B в Интернете. Необходимость обмена электронными документами между организациями существует с начала 1970-х годов. В процессе развития концепции EDI (электронного обмена данными) первый стандарт был разработан в 1996 году [34].

Первые программные решения работали по принципу p2p, используя безопасные соединения (VPN и т. д.). Позже появились решения на базе VAN (сетей с добавленной стоимостью), организованных как локальные промежуточные узлы, управляющие передачей документов с использованием различных транспортных протоколов (FTP, AS2 и др.) [103]. Эта конфигурация до сих пор является самой популярной и обычно управляется крупной телекоммуникационной компанией, консорциумом провайдеров или государственными учреждениями (например, Peppol, EDF+) [62, 74, 86].

Большинство корпораций создают собственные сервисы для управления электронными документами и контроля доступа пользователей с помощью централизованных решений. Централизованные решения состоят из централизованного сервера, на котором хранятся необходимые данные, и к которому клиенты могут получить доступ на основе правил, созданных на этом сервере. Этот сервер обрабатывает запросы различных пользователей, обрабатывает их и возвращает ответ на основе параметров запроса. Централизованные решения считаются простыми в развертывании. Они хорошо работают в случаях использования с низкими требованиями к ресурсам. Одним из преимуществ использования централизованных решений является их относительно низкая стоимость, а также тот факт, что ими несложно управлять, поскольку все данные расположены в одном месте. Централизованный сервер без применения технологии резервного копирования приводит к малому требуемому пространству для хранения данных (S_{Output}). Однако централизованные системы относительно более уязвимы к атакам и угрозам, что приводит к критическим последствиям для всей системы [42, 85].

Если сервер взломан или поврежден, данные становятся не доступны (что приводит к низкой надежности - R). Чтобы решить эту проблему, многие компании используют решения резервного копирования (рисунок 1.1), в которых один или несколько серверов резервного копирования используются для дублирования данных, хранящихся на основном сервере. Хотя этот метод может эффективно повысить надежность системы, это означает дополнительные затраты, связанные с дополнительным размером хранилища — S.

Еще одна проблема, от которой страдают централизованные решения, — это масштабируемость. За последние несколько лет объем данных, которыми обмениваются в Интернете, резко увеличился. Ожидается, что к 2025 году будет генерироваться 463 ЭБ данных в день [25]. Поскольку данные компаний со временем растут, для хранения данных может потребоваться использовать более одного физического файлового хранилища. Это означает, что централизованные

решения имеют существенные ограничения, т.к. не могут увеличивать емкость дискового пространства бесконечно (S`).

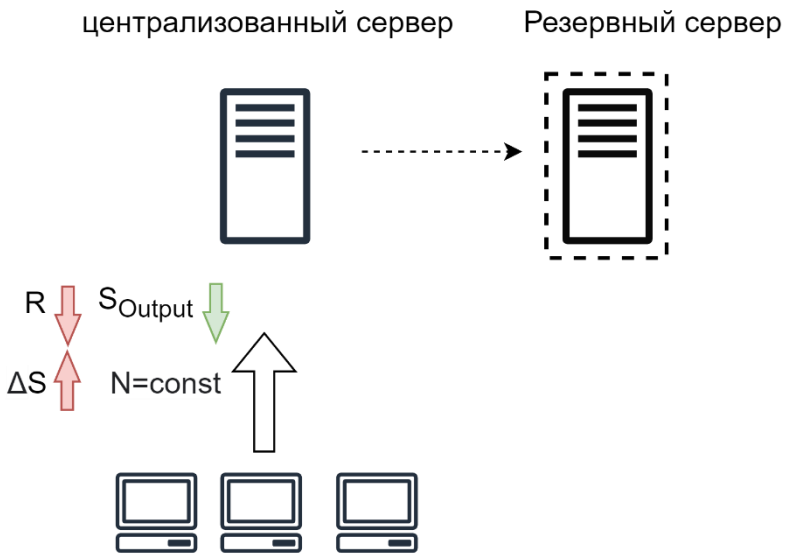


Рисунок 1.1 — Топология сети централизованных серверов

С другой стороны, децентрализованные системы сложны в развертывании и дороги, но в целом они более надежны, несмотря на то, что отдельные компоненты таких систем остаются уязвимыми. Блокчейн рассматривается как одна из перспективных технологий повышения надежности информационных систем (DLT) [92, 116] в целом. Приложения, которые взаимодействуют друг с другом или с конечным пользователем с помощью блокчейна вместо централизованной базы данных, называются DApps [6, 46, 59]. Основным препятствием для внедрения DApps является высокая стоимость, которая существенно снижает количество потенциальных пользователей [59]. Блокчейн хранит данные в виде транзакций в блоках. Если требуется обновить некоторые хранимые данные, новые данные добавляются в новые блоки вместо изменения существующих данных, что гарантирует неизменяемость данных. Историю изменений можно отслеживать. Блокчейн использует механизм, называемый механизмом консенсуса[8, 20, 56], который проверяет транзакции и

предотвращает изменение данных недоверенными пользователями в одном или некоторых узлах блокчейна. Наиболее популярными механизмами консенсуса являются два: Proof of Work и Proof of Stake [18, 38, 116]. Использования консенсуса и особенность хранения блоков делают DApps устойчивыми к компрометации данных.

Многие блокчейн-платформы позволяют запускать на своей стороне программное обеспечение, называемое смарт-контрактом [39, 110, 116], которое можно использовать для управления операциями, связанными с блокчейном. Смарт-контракты можно рассматривать как приложения, которые работают на стороне серверов и контролируют управление данными в системе и то, как пользователи взаимодействуют с данными. Сети блокчейна могут быть общедоступными, где все транзакции видны любому, или частными. Самыми популярными публичными сетями блокчейнов являются Биткойн [82] и Ethereum [27, 125]. Hyperledger Fabric [3, 67, 98] является примером частных сетей блокчейнов. Эти сети и платформы приобрели большую популярность за последние несколько лет, и ожидается, что их рынок вырастет намного больше к 2027 году [53] (рисунок 1.2).

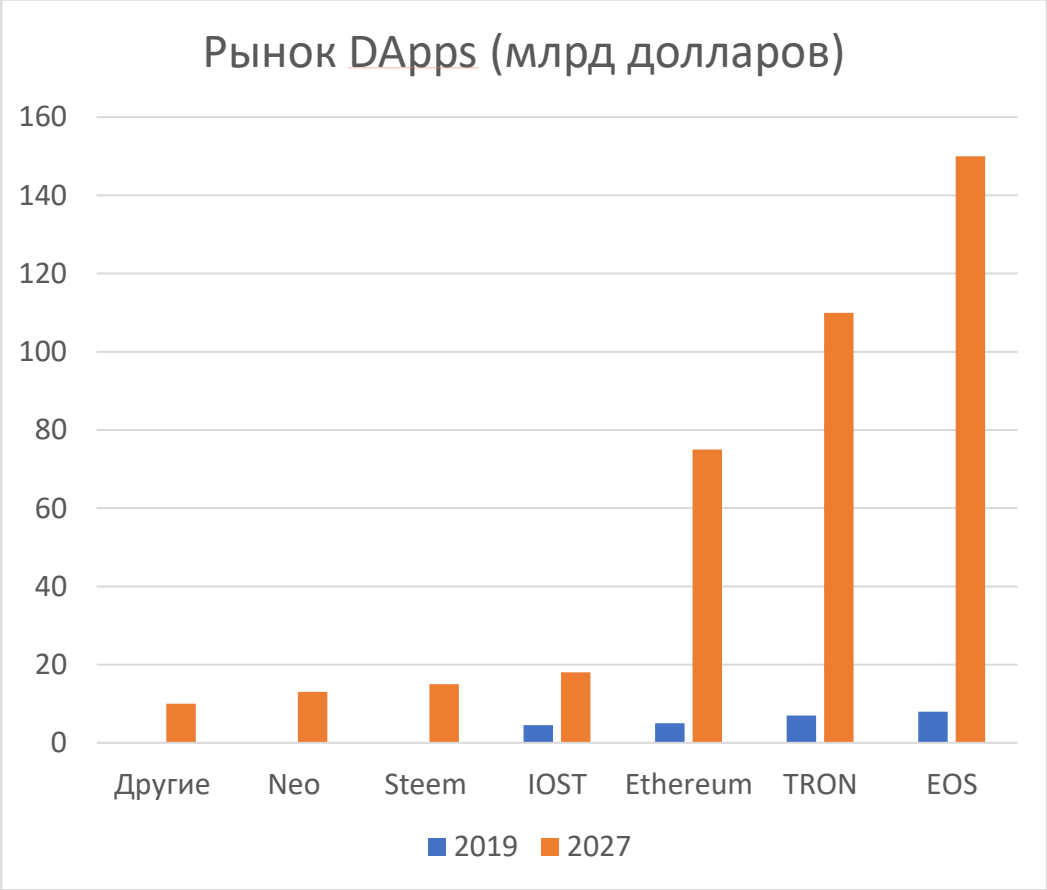


Рисунок 1.2 — Маркет DApps

Блокчейн хранит одни и те же данные на многих узлах, а это означает, что он имеет высокую избыточность (что означает высокое значение S_{Output}), что также приводит к высокой доступности и надежности (R). На рисунке 1.3 показано, как связаны узлы и пользовательские приложения блокчейна. Кроме того, блокчейн не учитывает, что разные узлы имеют разную емкость хранения, а значит, не гарантирует равное распределение данных (ΔS).

Поскольку блокчейн реплицирует данные, он обычно не используется для хранения больших объемов данных, например файлов. Вместо этого для этой задачи обычно используются другие типы распределенных систем. Однако распределенные системы различаются по различным аспектам, таким как производительность, размер хранилища данных, надежность и другие параметры, основанные на разных параметрах, таких как используемая топология и

архитектура, а также связанный с ней метод хранения. По результатам анализа в данном исследовании выделено несколько параметров для оценки эффективности систем хранения [81]. Они перечислены в таблице 1.1.

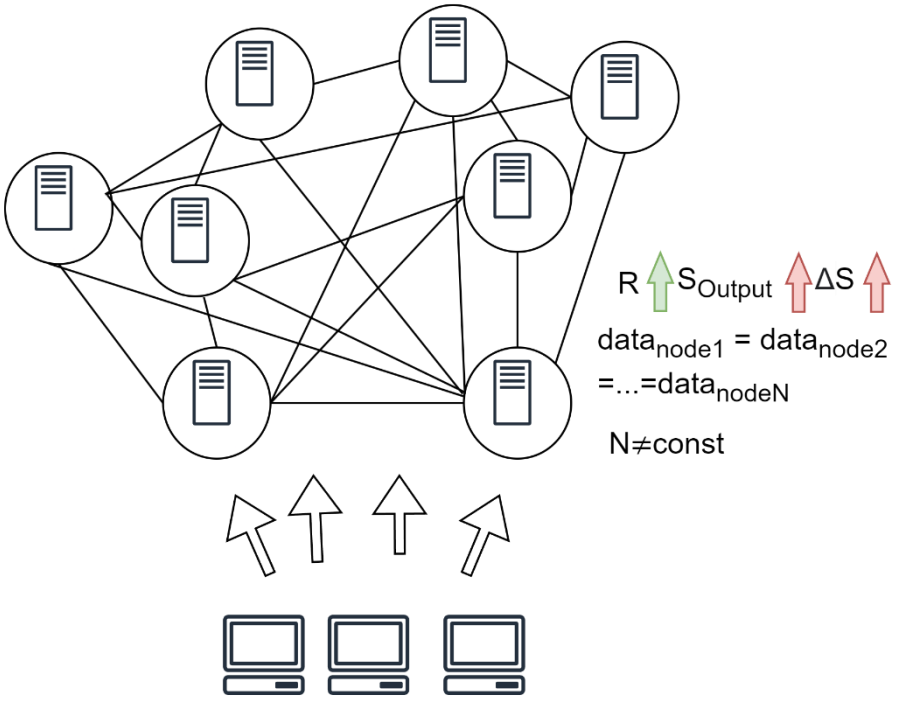


Рисунок 1.3 — Топология сети блокчейна

Таблица 1.1 — Необходимые показатели для анализа систем

Показатель	**Символ**	**Единица измерения**
Множество узлов в системе	$\boldsymbol{N}$	Узел
Надежность системы	R	Процент
Объем свободного места, доступного для хранения объектов	$S`$	Байт
Отклонение значения размера хранилища в системе от равномерного распределения	ΔS	Байт
Размер входных объектов в системе	S_{Input}	Байт
Размер выходных объектов в системе	S_{Output}	Байт
Размер данных контроля доступа	$S_{AccessControl}$	Байт

Размер метаданных	$S_{Metadata}$	Байт
Размер дополнительных данных, используемых для повышения доступности/надежности объектов	$S_{Reducancy}$	Байт

Количество узлов в системе считается важным показателем эффективности системы хранения [94]. Здесь и далее под масштабированием системы мы будем понимать возможность бесконечного увеличения узлов системы и общего доступного для хранения места без изменения других параметров (в первую очередь, производительности и надежности). В идеале системы хранения должны быть динамичными и поддерживать масштабирование, чтобы соответствовать растущим требованиям к хранению данных. Это может быть отражено в изменении количества узлов, поскольку оно не должно быть константном ($N \neq const$), поэтому можно добавлять новые узлы для масштабирования системы и хранения большего количества данных. Надежность системы (R) также должна быть максимально высокой. Большинство систем дублируют данные с помощью серверов резервного копирования для их увеличения, но это приводит к увеличению размера данных в системе (S_{Output}), что приводит к увеличению стоимости. В этом исследовании рассматривается уменьшение размера данных при одновременном повышении надежности всей системы.

Еще одним важным параметром системы является объем свободной памяти в системе (S`). При добавлении нового узла важно, чтобы в новом узле был свободный объем памяти для хранения новых данных, а не просто дублирование существующих данных в системе, как в блокчейне, что делает добавление новых узлов бесполезным с точки зрения увеличения свободной памяти в системе. Важно увеличивать свободное пространство в системе S` при добавлении новых узлов.

При добавлении новых узлов или удалении узлов также важно в любой момент распределять ресурсы между доступными узлами, поскольку не должно быть узла, который полностью используется и рассматривается как узкое место, в

то время как другие узлы имеют небольшую процент хранения. По этой причине оптимальный размер хранилища каждого узла должен быть рассчитан SS_i, который должен учитывать размер хранилища всех узлов и размер хранилища узла i, и на основе этого фактический размер хранилища на узле i (S_i) должно быть как можно ближе к SS_i, что означает $\Delta S_i = (SS_i - S_i) \rightarrow 0$: $\forall i \in \boldsymbol{N}$.

С увеличением количества узлов должно увеличиваться свободное пространство, доступное для хранения новых файлов, а также должна повышаться надежность системы, как на рисунке 1.4.

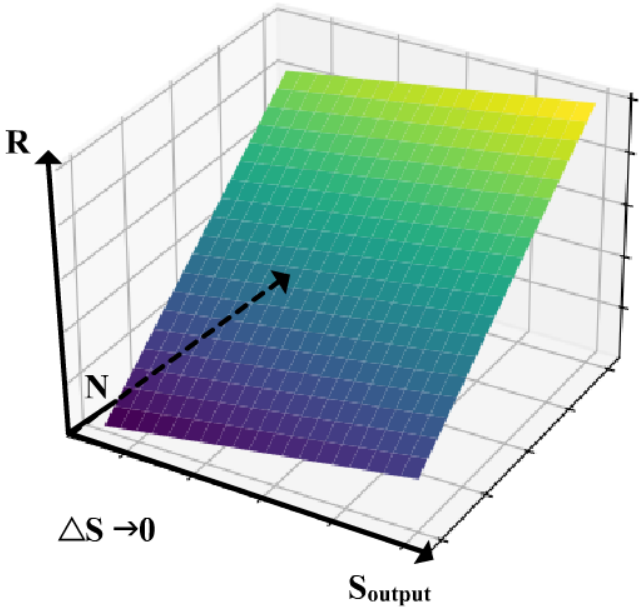


Рисунок 1.4 — Требуемое соотношение между $\boldsymbol{N}, R$ и S_{Output} в распределенных системах хранения

В результате необходимо получить распределенную систему, как на рисунке 1.5, которая сочетает в себе следующие показатели:

- Динамическое количество узлов $\boldsymbol{N}$
- Высокая надежность (R)
- Минимальный размер хранилища (S_{Output})
- Равное распределение данных в системе $\Delta S_i \rightarrow 0$: $\forall i \in \boldsymbol{N}$

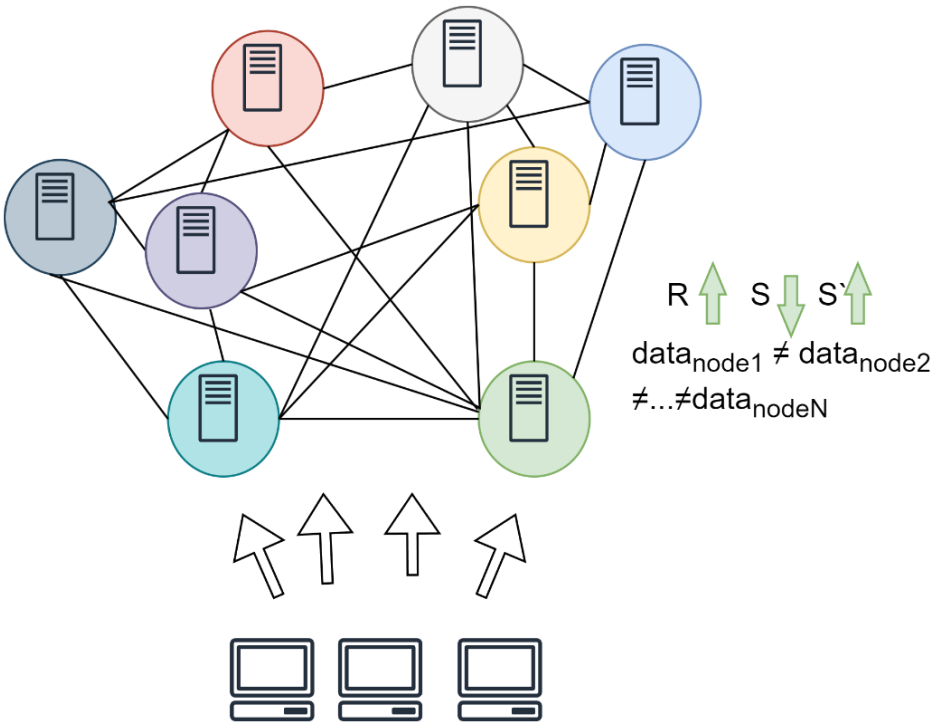


Рисунок 1.5 — Требования к системе хранения

Важно отметить, что наличие централизованного сервера управления данными может привести к снижению надежности системы в целом. Разные системы распределенного хранения имеют разные значения этих показателей, так как это зависит от используемого метода распределения, модели хранения и т. д.

В данной работе алгоритм стирающего кодирования используется предложенной моделью и алгоритмами для минимизации необходимого размера хранилища. RAID-5 и RAID-6 считаются одними из самых популярных реализаций стирающего кодирования в хранилищах данных. Несмотря на то, что они ориентированы на распределение данных по физическим дискам, эту концепцию можно расширить для использования на уровне узлов системы вместо физических дисков, что обеспечивает параллельную систему, что важно в контексте рассмотрения распределенных систем. Существуют и другие классы алгоритмов кодирования, которые могут минимизировать размер данных больше, чем на 25%, а также другие виды RAID. Однако RAID-5 и RAID-6 рассматриваются как базовые реализации, которые можно использовать для

базовых сравнений и они представляют сбалансированное решение между минимизацией размер данных и скоростью их восстановления.

IPFS [9, 30, 61]— это распределенная система обмена файлами P2P (одноранговая сеть). В этой системе содержимое файла хэшируется для создания адреса, который можно использовать для ссылки на этот файл. Он имеет торрент-подобную модель хранения. В этой системе, когда узел добавляет новый файл, этот файл становится доступен другим узлам. Узлы могут выбирать, какой файл сохранить, а какой удалить. Это означает, что файлы не гарантированно будут храниться преимущественно, а это, в свою очередь, означает, что не все файлы будут доступны, и это зависит от количества узлов, которые решили хранить эти файлы. Как показано на рисунке 1.6, если у узла №3 есть файл, а узлы №4 и №6 хотят иметь копию этого файла, они могут найти его местоположение, используя его хэш из распределенной хэш-таблицы (DHT) [1, 117]. Они обнаруживают, что этот файл есть на узле №3, и загружают его с этого узла. Благодаря этому они смогут поделиться этим файлом. Если узел №3 покинет сеть, файл по-прежнему будет доступен через узлы №4 и №6.

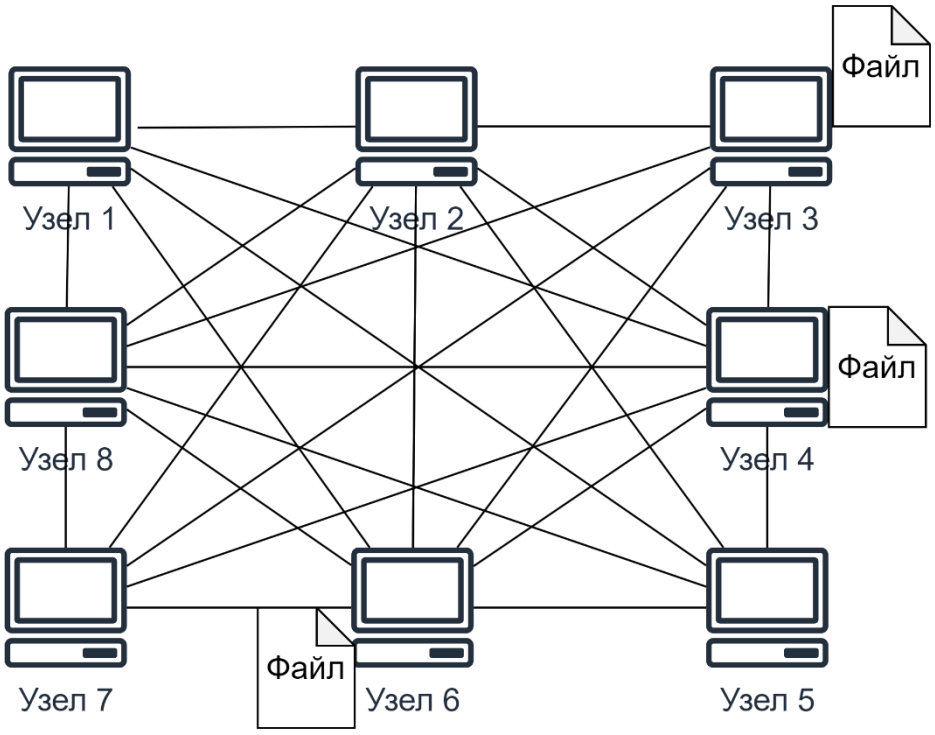


Рисунок 1.6 — Обмен файлами в IPFS

IPFS хранит данные блоками по 256 КБ. Если размер файла превышает 256 КБ, он разбивается на несколько блоков. Когда узел объявляет новый файл, он вычисляет хэш содержимого файла и добавляет его в Ledger. Таким образом, при загрузке файла можно проверить корректность файла, рассчитав хэш после его загрузки. Этот метод хранения файлов подходит для многих приложений, особенно для хранения данных NFT [28, 49, 58].

Однако IPFS не подходит для хранения файлов организациям, так как надежность зависит от того, какие узлы делают копию этого файла. Если узел потерян и никакие другие узлы не запросили его содержимое, все имеющиеся у него файлы считаются потерянными (низкая надежность R).

Storj — распределенная система хранения файлов, имеющая некоторое сходство с IPFS. В этой модели файл разбивается на 80 зашифрованных частей, которые распределяются по нескольким узлам.

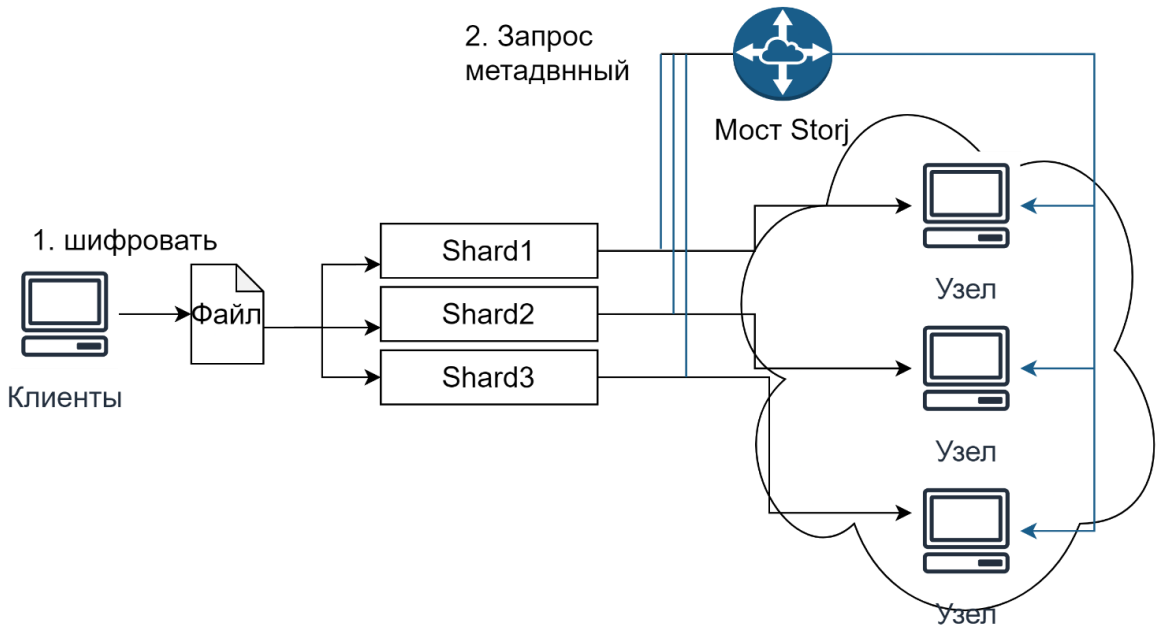


Рисунок 1.7 — Модель распределения данных в Storj

В Storj используется централизованная точка управления, которая называется мостом Storj (рисунок 1.7), а узлы, хранящие файл, получают

вознаграждение в виде монет, называемых монетами Storj. Storj использует централизованные компоненты (мосты Storj). Централизованная точка отказа означает меньшую надежность R. Также, Storj не распределяет данные равномерно по узлам. Распределение зависит от расположения узлов.

DADS [26] — это система развертывания приложений, которая использует IPFS для хранения приложений вместо использования выделенных серверов. В целом система предназначена для хранения различных приложений с разными пользователями и кошельками.

HyperBSA [19, 69]— это решение, которое предлагает кэшировать файлы, к которым обращаются чаще всего, с использованием нового алгоритма кэширования LBN, который управляет пространством памяти с помощью HashMap. Кроме того, он разделяет данные на две категории: состояние и непрерывные данные, а затем обрабатывает каждый тип отдельно. Согласно оригинальному документу HyperBSA, сама система не эффективна в поиске архивных файлов, поэтому в этом решении есть режим подключения к другим решениям для хранения данных, таким как IPFS.

В другом решении [100] обсуждается новый метод хранения и восстановления данных в промышленных блокчейн-сетях, который пытается увеличить скорость восстановления данных, улучшить способ мониторинга данных и т. д. Тем не менее, этот метод не учитывает хранение файлов.

Lustre [4, 13, 52]— это файловая система с открытым исходным кодом на основе кластеров, которая работает в распределенных структурах. Большинство HPFS (высокопроизводительная файловая система) используют файловую систему Lustre. Файловая система Lustre в основном состоит из узлов трех типов (рисунок 1.8):

- Клиент, который использует файловую систему
- Сервер метаданных (MDS)
- Сервер хранения объектов (OSS)

Файловая система Lustre может содержать более одного сервера метаданных, и эти серверы сохраняют данные на устройствах, называемых MDT

(MetadataTarget). Кроме того, файловая система Lustre может содержать один или несколько MDT. MDT содержат все необходимые метаданные, такие как имена файлов, их разрешения, папки и т. д.

Эти серверы управляют устройствами хранения, называемыми OST (Object Storage Target). Файловая система Lustre может иметь один или несколько OSS, и каждый OSS может иметь один или несколько OST. Общая емкость системы хранения представляет собой сумму мест хранения всех OST.

MDS использует как циклический (round-robin), так и взвешенный случайный алгоритм (weighted random algorithm) для распределения объектов.

Пока может существовать более одного MDT, Lustre использует удаленные каталоги DNE (распределенное пространство имен), чтобы назначить каждому MDT часть общих данных MDT. Этот процесс называется DNE phase 1. MDT могут иметь вложенные отношения, поэтому один MDT может представлять часть другого MDT, но это может привести к проблемам, поскольку в случае повреждения MDT все вложенные MDT становятся бесполезными.

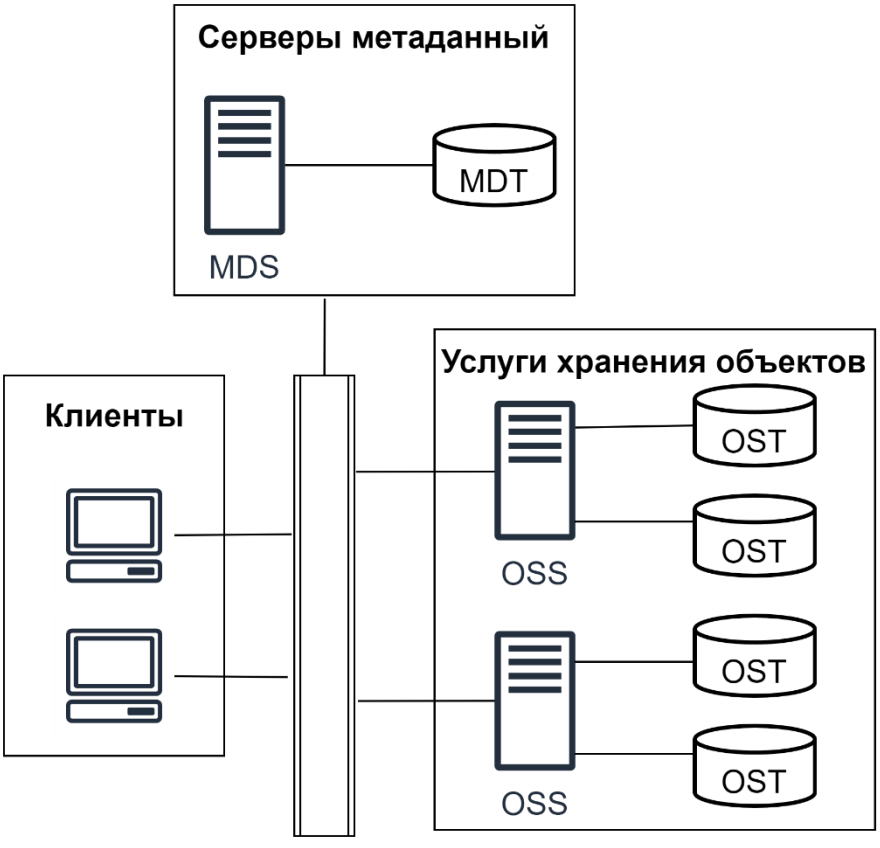


Рисунок 1.8 — Архитектура хранилища Lustre

Метаданные одного каталога могут храниться на нескольких MDT, причем каждый MDT хранит часть этих метаданных. Например, этот каталог содержит несколько файлов, и несколько MDT хранят метаданные файлов, расположенных в этом каталоге, причем каждый MDT хранит часть метаданных файлов. Тем не менее, все метаданные одного файла хранятся в одном и том же MDT.

Серверы MDS требуют больших ресурсов. Согласно официальной документации [57], для MDS-серверов принято выделять 16 ядер и 256 ГБ оперативной памяти. MDS используют Inode bit locks (Inode — это структура данных, которая используется в Unix-подобных системах) для поиска файлов [44]. Inode bit locks работают следующим образом: при запросе файла (пример files/file1) проверяется структура Inode на предмет местоположения папки «files», например, она находится в блоке «B1». Затем проверяется блок «Б1» на чтение содержимого папки и расположения файлов, находящихся внутри. Например, метаданные файла «file1» расположены в блоке B2. Наконец, проверяется блок B2, и это по=показывает, что фактические данные хранятся в блоке B3.

Lustre не подходит для обработки электронных документов, поскольку:

1. Не гарантируется, что файлы будут равномерно распределены по узлам. Распределение файлов зависит от размера файла и количества узлов, и клиент сам должен настроить параметры распространения.

2. Для хранения метаданных требуется центральный сервер с высокими характеристиками, что считается узким местом.

3. Lustre не использует интегрированную модель резервного копирования, поэтому следует использовать другие решения для резервного копирования

WekaFS [113] — еще одна файловая система, предназначенная для работы с распределенными структурами. WekaFS поддерживает операции с высокой пропускной способностью, поскольку изначально была разработана для NVMEeoF. Также WekaFS обладает высокой масштабируемостью. В отличие от

Lustre, WekaFS распределяет метаданные по всем узлам хранения (в то время как другие распределенные файловые системы, такие как Lustre, используют выделенные серверы для размещения метаданных). При этом не будет узких мест, поскольку серверы метаданных влияют на общую производительность системы. В wekaFS метаданные равномерно распределяются по узлам, поэтому нет узла, который считается более важным, чем другие узлы. Способ реализации процессов метаданных запатентован. Кроме того, WekaFS не обеспечивает никаких методов избыточности данных или резервного копирования, а это означает, что его не следует использовать без реализации какого-либо внешнего механизма резервного копирования (низкий R).

Gluster [54, 70, 101] — масштабируемая сетевая файловая система. Это бесплатный проект с открытым исходным кодом. Название Gluster представляет собой комбинацию GNU и Lustre, однако оно не основано на Lustre. Gluster позволяет объединить несколько физических серверов в одну файловую систему, делая ее прозрачной для клиентов. Имеет клиент-серверную архитектуру. На каждом сервере работает определенное приложение (демон glusterfsd), которое позволяет монтировать файловую систему на сервере как том в системе хранения. Клиенты могут подключаться с использованием протокола, созданного для этой задачи, с помощью процесса, называемого «Клиентский процесс Glusterfs» [12]. В Gluster, в отличие от Lustre, нет серверов метаданных, а в отличие от WekaFS метаданные вообще не нужны. Файлы располагаются с использованием эластичного алгоритма хэширования. Благодаря этому алгоритму расположение фрагментов данных можно найти с помощью вычислений без поиска по индексам. Виртуальный том может располагаться в нескольких томах (рисунок 1.9).

В Gluster администраторы должны вручную создавать хранилища, называемые bricks. Gluster не управляет и не балансирует данные или блоки автоматически между узлами.

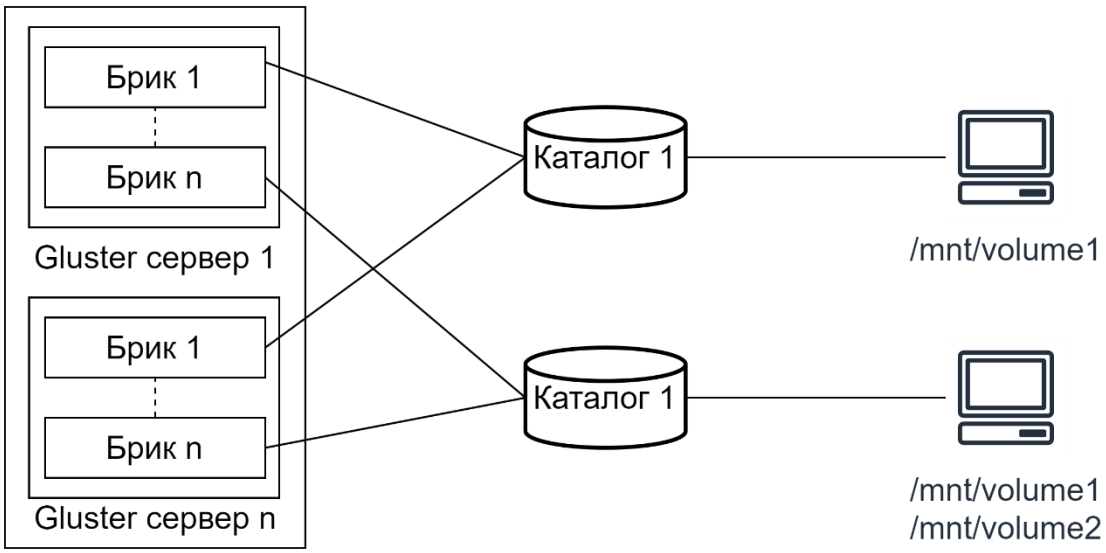


Рисунок 1.9 — Архитектура Gluster

RAID — одна из старейших, но до сих пор используемых технологий хранения данных на нескольких дисках. Несмотря на то, что эта технология предназначена для объединения дисков внутри сервера вместо соединения нескольких серверов или узлов хранения вместе, важно обсудить эту технологию, поскольку для узлов могут использоваться одни и те же концепции хранения данных, касающиеся распределения данных и восстановление. RAID — это сокращение от «Резервный массив независимых дисков». Эта технология объединяет физические диски в логические. RAID имеет несколько типов. RAID-5 [123] и RAID-6 [40, с. 6] считаются самыми популярными. Оба они используют данные четности для восстановления данных. На рисунке 1.10 показана схема хранения данных в RAID-5.

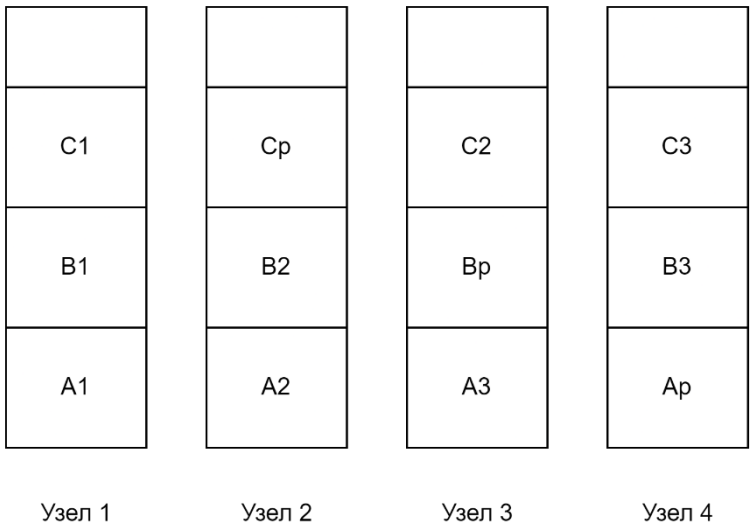


Рисунок 1.10 — Модель распределения данных в RAID-5

RAID имеет централизованную точку управления, называемую RAID-контроллером. Он может быть аппаратным, то есть физическим контроллером, установленным на сервере, или программным, когда на сервере установлено определенное приложение. Однако этот контроллер можно рассматривать как единственную точку отказа (что приводит к снижению R). Кроме того, необходимо, чтобы все диски были одинаковой емкости. Если использовались диски разной емкости, диск с минимальной емкостью определяет доступное дисковое пространство любого другого диска. Кроме того, RAID не поддерживает масштабируемость (N=const). После настройки набора дисков (узлов) нельзя добавлять новые узлы, чтобы распределить данные по большему количеству узлов.

Ceph [13, 112, 118, 121] — это платформа, использующая объектное хранилище и имеющая собственную файловую систему (cephFS) [11, 22]. Он имеет сходство с Lustre, поскольку также использует серверы метаданных (ceph-mds). В cephFs один из серверов метаданных должен находиться в активном режиме, а остальные серверы метаданных — в режиме ожидания. Если активный сервер выходит из строя или перестает работать, один из серверов ремайнинга становится активным, чтобы гарантировать высокую доступность. Можно сделать

вывод, что недостатками использования данного проекта для обработки метаданных являются:

1. Для хранения метаданных требуется центральный сервер с высокими характеристиками, что считается узким местом.

2. Ceph имеет большую сложность, поскольку способ его работы зависит от того, как администраторы создают сеть и разбивают ее на кластеры. Кластеры в Ceph независимы, и выбор, в какой кластер следует добавить новый узел, означает, что его пространство хранения может использоваться только этим кластером, а другие узлы не могут получить доступ к этому кластеру, даже если у них закончится свободное пространство, а это означает, что загрузка балансировка по всем узлам не гарантируется.

HDFS [71, 102, 107, 111] — это файловая система, используемая Hadoop [41, 43]. В этой системе хранения файлы разбиваются на блоки и реплицируются на трех серверах, поэтому в случае сбоя одного из серверов содержимое по-прежнему будет доступно с двух других серверов. HDFS имеет разные типы узлов (рисунок 1.11):

1. Главные узлы (Master Nodes): серверы, используемые для управления распределенным хранилищем. Они называются NameNodes. NameNode содержит метаданные о кластере. Может быть один активный NameNode и резервные узлы имен, поэтому в случае сбоя NameNode один из оставшихся NameNodes становится активным.

2. Рабочие узлы (Worker Nodes): узлы, где фактически хранятся данные. Их также называют DataNodes.

Чтобы обнаружить сбои узлов, каждые три секунды на NameNode отправляется контрольное сообщение. Когда DataNode перестает отправлять контрольные сообщения, сервер NameNode предполагает, что этот DataNode вышел из строя, и отменяет его сопоставление с кластером, поэтому он исключается из будущих операций записи/чтения. Как только сервер NameNode снова получит контрольное сообщение от исключенного DataNode, DataNode автоматически присоединяется к кластеру.

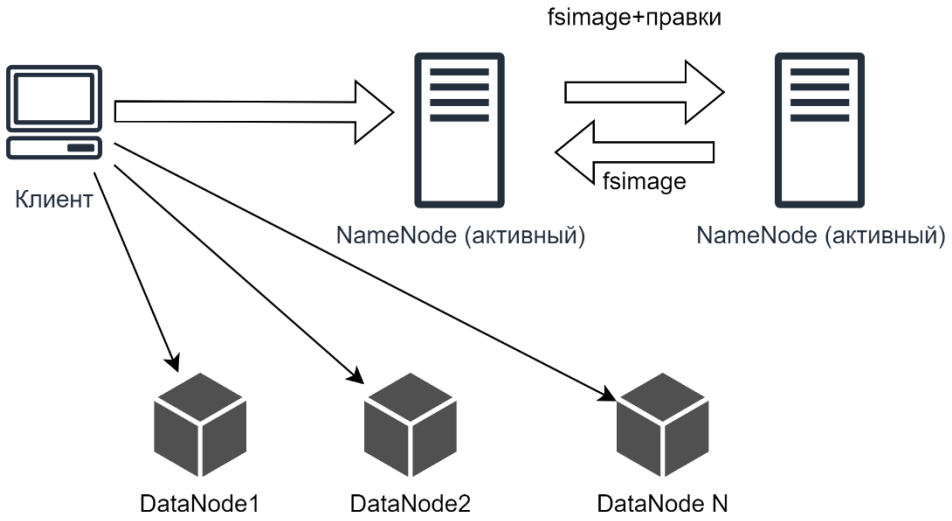


Рисунок 1.11 — Архитектура HDFS

HDFS в основном страдает от двух проблем:

1. Для хранения метаданных требуется центральный сервер с высокими характеристиками, что считается узким местом.

2. Он реплицирует данные на несколько узлов в одном кластере. Если все реплики выйдут из строя, данные будут потеряны навсегда. Кроме того, это означает хранение больших объемов данных, что означает высокую стоимость.

3. HDFS хранит данные блоками по 128 МБ каждый. Это означает, что хранение небольших файлов неэффективно, поскольку много свободного места будет потрачено впустую.

MS Azure [45, 64, 99]считается одним из самых популярных решений для облачных хранилищ и вычислительных услуг. Оно использует центры обработки данных, управляемые Microsoft. MS Azure поддерживает несколько типов данных, таких как хранилище объектов, хранилище файлов и хранилище таблиц. MS Azure имеет балансировщик нагрузки, который распределяет трафик между несколькими виртуальными машинами на серверной части системы и работает на четвертом уровне модели OSI (взаимодействие открытых систем). MS Azure имеет два типа балансировщика нагрузки:

- Публичный балансировщик нагрузки: используется для балансировки трафика в общедоступной сети, поскольку IP-адреса виртуальных машин преобразуются в общедоступные с помощью переадресации портов.
- Частный балансировщик нагрузки: он балансирует трафик в частной виртуальной сети.

Балансировщик нагрузки использует алгоритм, называемый алгоритмом хэширования кортежей. Он хэширует несколько параметров запроса и маршрутизирует трафик на основе результата хэширования. Список параметров, которые учитываются в этом процессе:

- Исходный IP-адрес
- Исходный порт
- IP-адрес назначения
- Порт назначения
- Протокол

Этот метод позволяет распределять трафик между несколькими виртуальными машинами вместо использования одной виртуальной машины для всех запросов. Однако этот метод не гарантирует равномерное распределение трафика по узлам. Если один и тот же пользователь отправляет трафик одного и того же типа с одинаковыми параметрами, каждый раз будет выбираться одна и та же виртуальная машина. Кроме того, балансировщик нагрузки не учитывает другие параметры, такие как размер файла в случае хранения файлов, а это означает, что данные не будут одинаково храниться на нескольких машинах (ΔS_i увеличится). На рисунке 1.12 показано, как работает балансировщик нагрузки в MS Azure.

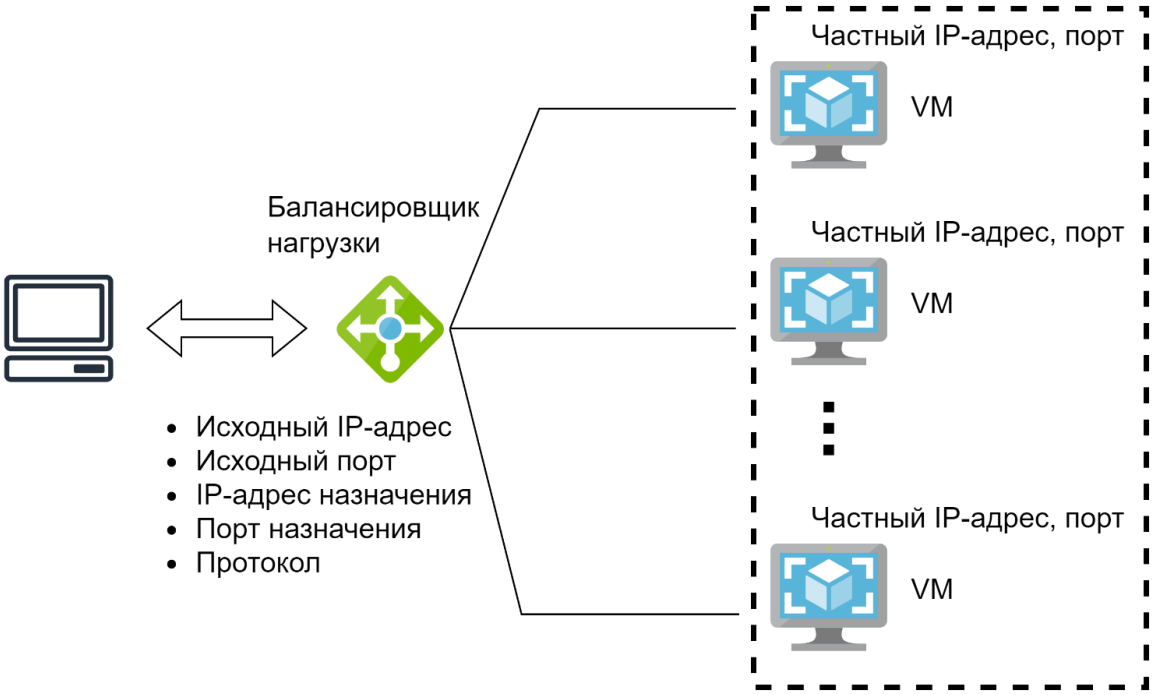


Рисунок 1.12 — Балансировщик нагрузки в MS AZURE

Amazon AWS [23, 32, 78], с другой стороны, распределяет трафик по нескольким целям с помощью сервиса под названием «Эластичная балансировка нагрузки». AWS имеет несколько типов механизмов балансировки нагрузки [33]:

1. Программный балансировщик нагрузки. Он работает на седьмом уровне модели OSI и обрабатывает трафик http и https. Алгоритм распределения данных по умолчанию — циклический. Несмотря на то, что он распределяет данные по узлам/виртуальным машинам, он не гарантирует справедливость распределения, поскольку в некоторых сценариях пользователь может отправлять на хранение файлы разного размера, что приводит к неравномерному распределению файлов в системе, что приводит к неравномерному распределению файлов в системе. означает, что ΔS_i увеличится.
2. Сетевой балансировщик нагрузки, который работает на четвертом уровне OSI и обрабатывает пакеты TCP/UDP. Он имеет алгоритм балансировки нагрузки, аналогичный тому, который используется Microsoft Azure. Информация запроса, включая IP-адрес источника, порт

источника, IP-адрес назначения и порт, используется для распределения трафика между узлами/виртуальными машинами, что не гарантирует равное распределение данных.

3. Балансировщик нагрузки на основе шлюза: работает на третьем уровне OSI. Он обеспечивает соединение между различными виртуальными частными сетями.

В таблице 1.2 представлен обзор известных систем хранения данных, а в таблице 1.3 – обзор обсуждаемой литературы. Знак +/- означает, что свойство поддерживается, но не полностью.

Таблица 1.2 — Анализ проектов хранения и распределения данных

	$\boldsymbol{R \rightarrow \max}$	$\boldsymbol{\lvert N \rvert \neq const}$	$\boldsymbol{S_{Output} \rightarrow \min}$	$\boldsymbol{\Delta S \rightarrow 0}$
HDFS	-	+	-	+
Gluster	+	+ / -	-	+/-
MS Azure	+	+/-	-	+/-
Amazon AWS	+	+/-	-	+/-

Таблица 1.3 — Анализ научных подходов хранения и распределения данных

	$\boldsymbol{R \rightarrow \max}$	$\boldsymbol{\lvert N \rvert \neq const}$	$\boldsymbol{S_{Output} \rightarrow \min}$	$\boldsymbol{\Delta S \rightarrow 0}$
Ceph	+/-	+	-	+
Raid-5	-	-	+	-
Raid-6	-	-	+	-
Storj	-	+	-	-
IPFS	-	+	+/-	-
Lustre	-	+	-	-

Сложность распределения данных

Как обсуждалось в главе о надежности, предоставленная модель представляет алгоритм, который позволяет распределять данные за полиномиальное время для проблемы, которую можно рассматривать как сложную задачу NP, которая обычно решается другими системами с использованием жадного алгоритма. Openstack использует жадный алгоритм для определения наилучшего физического узла для размещения нового экземпляра [47]. Этот процесс требует времени и ресурсов, поскольку проверяет все возможные решения. Hui Zhao и др. [55] предложил решение, которое учитывает использование дискового ввода-вывода и центрального процессора (далее ЦП) для политики размещения виртуальных машин. Вместо использования жадного алгоритма, который оценивает все возможные способы принятия решения о том, куда поместить эту виртуальную машину, они предложили для достижения этой цели использовать алгоритм Форда Фулкерсона. Этот алгоритм имеет сложность O(E*N), где N — количество узлов, а E — количество необходимых итераций. Это исследование не соответствует предлагаемой модели, поскольку:

1. Использование для принятия решения времени ввода-вывода диска для данного узла нестабильно. Можно предположить, что вновь созданная виртуальная машина (или VD) может иметь самый высокие показатели ввода-вывода, но как только хранилище заполнится, она будет потреблять гораздо меньше операций ввода-вывода.

2. Обсуждаемая проблема не требует обработки ЦП (например: на виртуальной машине/диске не выполняются никакие приложения)

В [55] устанавливается порог использования ЦП. Как только порог превышен, виртуальная машина выбирается и перемещается на другой хост на основе дискового ввода-вывода.

Для всех методов, которые зависят от перемещения виртуальной машины в зависимости от использования дискового ввода-вывода, важным ключевым моментом, который следует учитывать, является то, что виртуальная машина может иметь высокий объем дискового ввода-вывода в течение короткого

времени, а другая виртуальная машина после этого имеет высокую нагрузку. поэтому перемещение виртуальных машин не может быть полезным. Напротив, частое перемещение этих виртуальных машин может снизить общую производительность.

Мун-Хён Ким эль др. [63] учитывали несколько параметров, например, если узел был занят в течение длительного времени, время ответа узла, загрузка ЦП, количество виртуальных машин внутри узлов и т. д. Этот алгоритм начинается со списка узлов, которые можно добавить, и затем выполняет несколько тестов и с каждым тестом сокращает список, чтобы в конечном итоге получить нужный узел. Поскольку это решение может в конечном итоге выбрать лучший вариант, этот выбор не гарантируется, поскольку невозможно предсказать направление входящего трафика. Кроме того, алгоритм требует множества итераций, чтобы решить, какой узел выбрать.

Все упомянутые выше решения не рассматривают конкретные сценарии, такие как расширение дисков, когда существующей виртуальной машине следует выделить больше места, а существующего пространства недостаточно на текущем узле. Microsoft Azure просто уничтожает существующий диск и выделяет новый виртуальный диск (есть несколько случаев, когда диск не уничтожается, но он должен находиться в одном из определенных регионов и с конкретными условиями относительно его размера и способа нужно добавить большой размер, иначе диск будет уничтожен и воссоздан заново [36])

1.2 Анализ эффективности использования методов управления доступом на основе блокчейна

Поскольку использование централизованных компонентов снижает надежность системы в целом, необходимо рассмотреть научные подходы,

использующие блокчейн для управления контролем доступа и других метаданных объектов в системах.

Управление контролем доступа считается критически важным компонентом любой системы обмена конфиденциальными данными. Будь то система хранения файлов, веб-сайт социальной сети или любая другая система, обеспечивающая доступ к ее частным ресурсам, необходимо управлять тем, как разные стороны могут получить доступ к этим ресурсам. Таким образом, системе хранения недостаточно обеспечивать возможность хранения файлов. Она необходима для управления и контроля доступа пользователей к его файлам, в противном случае он не может быть применима.

Контроль доступа можно разделить на следующие основные модели [80, 114, 120]:

1. Мандатный контроль доступа (MAC). Это тип контроля доступа, который имеет уровни конфиденциальности. Примером может служить ситуация, когда ресурсы классифицируются по грифу секретности {общие, секретные, совершенно секретные}. Таким образом, пользователи могут получать доступ к ресурсам в зависимости от предоставленного им уровня безопасности.

2. Дискреционный контроль доступа (DAC) [122]: при этом типе контроля доступа владелец или создатель ресурса определяет, кому разрешен доступ к нему. Обычно DAC использует матричную модель доступа [29], которая представляет собой таблицу, описывающую привилегии в стиле субъекта (например: пользователя) и объекта (например: ресурса). В социальных сетях, таких как Facebook [37], пользователь может ограничить количество людей, которые могут получить доступ и просмотреть содержимое его профиля или конкретных сообщений, что рассматривается как пример DAC.

3. Управление доступом на основе ролей (RBAC) [95]. При этом типе управления доступом пользователи имеют разные роли, а доступ к ресурсам осуществляется в зависимости от их роли. Например, на веб-сайтах электронной коммерции некоторые страницы доступны только администраторам, другие страницы могут быть доступны менеджерам и администраторам и т. д.

4. Управление доступом на основе правил [16]. Этот тип управления доступом определяет правила доступа к ресурсам, которые могут быть различными. Например, некоторые банки могут поставить условие, запрещающее обмен валюты в ночное время. Обычно он используется в сочетании с контролем доступа на основе ролей [50].

5. Контроль доступа на основе атрибутов (ABAC) [93]. Он управляет доступом путем назначения политик для различных атрибутов пользователей, ресурсов и среды [106].

Важно отметить, что в современных системах обычно используется комбинация разных типов моделей контроля доступа.

Практически все модели доступа применяются совместно с DAC, поскольку это модель обеспечивает самые простые базовые правила доступа субъект-объект. Например, иногда требуется предоставить конкретному пользователю доступ к определенному ресурсу, не предоставляя разрешения другим пользователям с той же ролью или набором атрибутов. Проблема в том, что реализация DAC может привести к созданию большой матрицы доступа ресурсов и пользователей. В файловых системах хранения, если всего имеется 1 миллион файлов и 1000 пользователей, мы получим миллиард записей, что может быть проблематично при рассмотрении распределенных систем с огромной избыточностью данных, таких как блокчейн. При использовании RBAC требуется хранить меньше данных, поскольку одна политика может охватывать множество файлов. Однако результирующий размер данных может также оказаться большим и непригодным для хранения в блокчейне. Целью данного исследования является разработка распределенной модели управления доступом DAC+RBAC, решающей проблему масштабирования для систем на базе блокчейна без централизованных компонентов.

Существует множество исследований, в которых обсуждается применение контроля доступа в блокчейне. Большинство этих исследований посвящено тому, как реализовать контроль доступа с использованием блокчейна для конкретной области, особенно Интернета вещей [21, 126]. Cheng и др. [21] предлагают

развернуть точки принятия решений по политике (PDP) в блокчейне и хранить точки администрирования политики (PAP) вне блокчейна, чтобы уменьшить размер нагрузки на блокчейн. Однако это исследование не показывает, как хранятся данные вне сети и как они реплицируются или располагаются, чтобы предотвратить потерю данных в случае потери сервера хранения. Таким образом, правила контроля доступа в основном централизованы (вне блокчейна) и доступны децентрализованной системе (блокчейн), которая не является настоящей децентрализованной системой. Кроме того, согласно их решению, когда запросы отправляются в блокчейн, запрос перенаправляется на ресурсы вне блокчейна для каждого отдельного запроса, что может быть неэффективно в крупномасштабных системах.

Maesa и др. [73] предложили гибридный подход, который позволяет хранить политики прямо в блокчейне или ссылаться на него. Новые политики могут быть созданы путем создания новых биткойн-транзакций определенного типа. Для обновления или отзыва доступа после этого создается новая транзакция, которая ссылается на созданную политику, и вместе с ней сохраняются новые политики. Существует также особый тип политик, называемый «Транзакция передачи прав», который позволяет передавать права от пользователя. А пользователю В. Последний тип введенных авторами политик позволяет обмениваться правами между владельцами, поэтому он аналогичен передаче прав от пользователя А к В и прав от В к А одновременно.

Этот подход переписывает политики, чтобы минимизировать объём данных, и сохраняет их сжатую версию в блокчейне. Это решение может хорошо работать с системами доступа ABAC, но оно не подходит для систем DAC, поскольку эти политики основаны на пользователях, а файлы политик могут быть большими.

Paillisse и др. [88] предложили распределенную систему, в которой политики доступа хранятся в блокчейне. Этот подход предлагает расширение групповых политик (GBP) и применим в сети блокчейн на базе Hyperledger Fabric. Архитектура предлагаемого решения подразумевает, что когда организация А хочет получить доступ к ресурсу, контролируемому организацией В, организация

B создает запрос на доступ, содержащий информацию организации A, и отправляет его в смарт-контракт блокчейна, который, в свою очередь, проверяет, имеет ли организация B доступ к необходимому ресурсу или нет, как на рисунке 1.13.

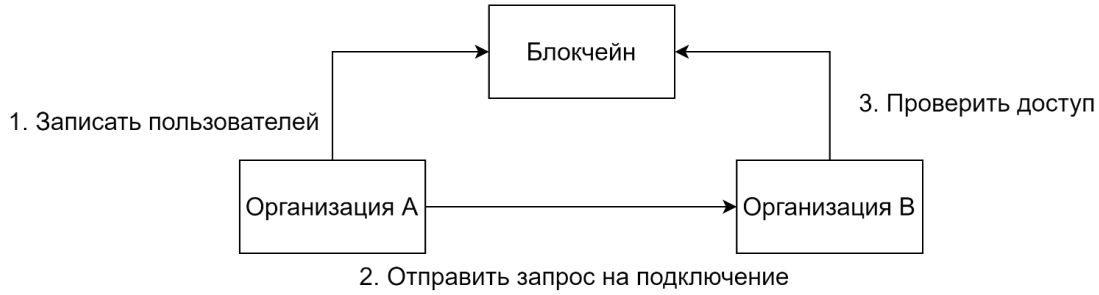


Рисунок 1.13 — Модель распределения данных контроля доступа с использованием GBP

Авторы предположили, что администраторы могут управлять операциями с помощью CLI на основе групповой политики. Однако они предлагают хранить все политики в блокчейне, что невозможно при рассмотрении DAC.

Tuler и др. [109] предложили новую модель контроля доступа к медицинским записям в смарт-контрактах. Решение предлагает хранить необходимые ресурсы вне блокчейна, сохраняя при этом проверку в блокчейне. В соответствии с этим решением можно развернуть несколько смарт-контрактов: смарт-контракт обеспечения соблюдения политик (PEPSC), смарт-контракт принятия решений (PDPSC), смарт-контракт политик (PAPSC) и смарт-контракт информации о политиках (PIPSC). Политики хранятся с использованием модели XACML [75]. На рисунке 1.14 показано, как различные смарт-контракты взаимодействуют для предоставления доступа и проверки информации запроса.

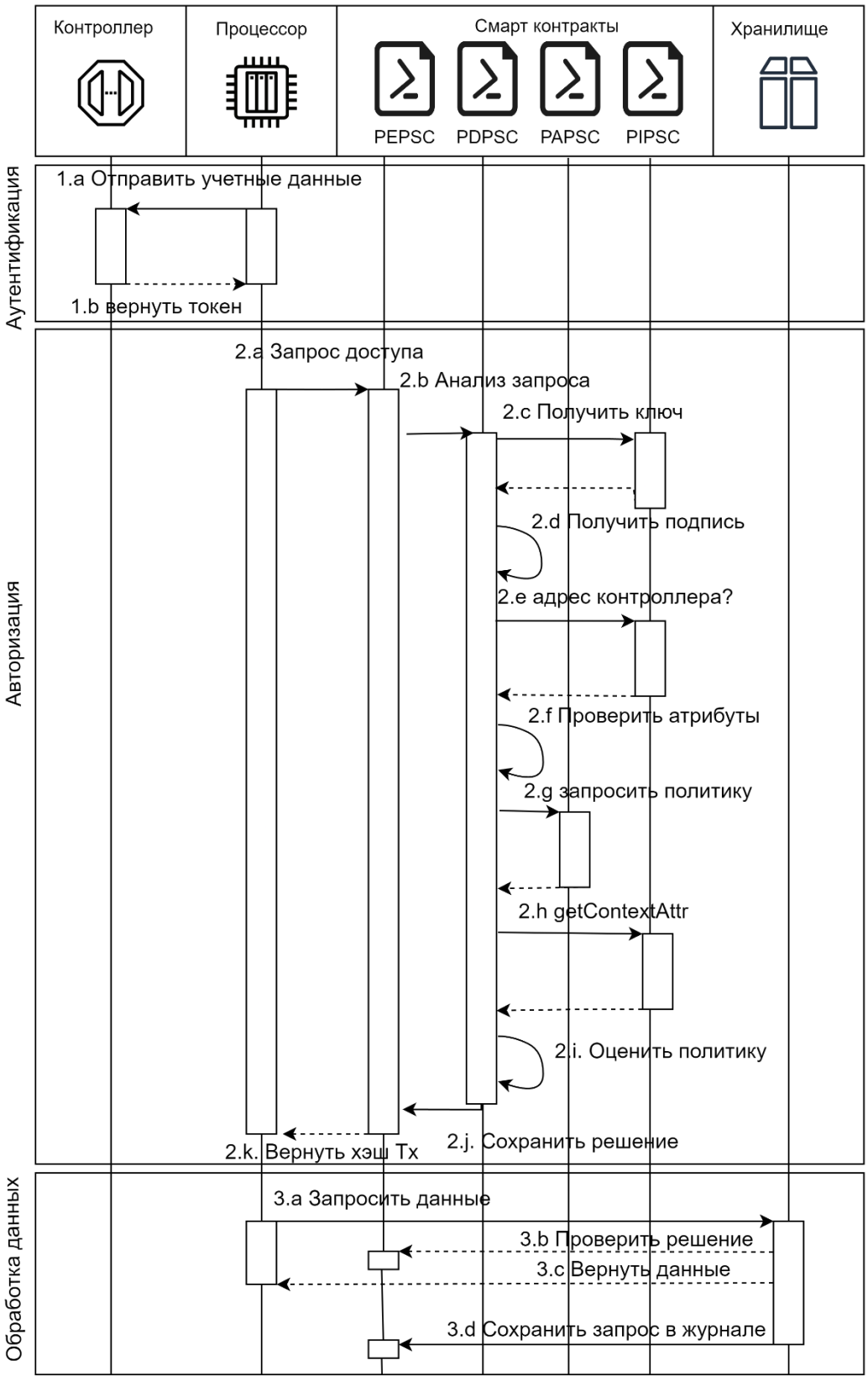


Рисунок 1.14 — Разделение пунктов политики по смарт-контрактам

Этот метод хорошо работает для управления доступом на основе атрибутов, но хранить правила в блокчейне в случае DAC не рекомендуется по причинам, упомянутым выше.

Из проведенного анализа можно сделать вывод, что известные исследования не предлагают модель управления доступом DAC и RBAC на основе блокчейна, которая требует минимального размера хранилища данных в блокчейне и обладает возможностью добавления новых объектов и субъектов в систему без ущерба для ожидаемого уровня безопасности системы или скорости доступа.

Рассмотренные в этой главе проблемы хранения данных контроля доступа можно обобщить для управления метаданными хранящихся в распределенной системе объектов. В случае метаданных также необходима привязка метаданных к объектам и субъектам, необходим быстрый поиск этих метаданных по ряду критериев. Необходимо обеспечить их хранение больше чем на одном узле системы, но при этом уменьшить избыточность данных.

Выводы по главе 1

1. Цифровая трансформация требует новых подходов в проектировании распределённых систем обмена данными и хранения файлов. Требования к хранению данных растут, поскольку обмен данными в последние годы растет в геометрической прогрессии.

2. Централизованные серверы используются многими организациями, поскольку они дешевы и просты в развертывании. Однако централизованные серверы имеют одну или несколько единых точек отказа и не поддерживают масштабируемость (увеличение общего размера хранилища данных без ограничений). Надежность централизованных серверов можно повысить за счет использования серверов резервного копирования, которые дублируют данные,

увеличивая тем самым общую стоимость. Кроме того, серверы резервного копирования не решают проблему масштабируемости, которая считается требованием для организаций, где требуется хранить огромное количество файлов.

3. Блокчейн приобрел популярность за последние несколько лет, поскольку это распределенная система хранения, может организовать обмен данными между участниками с нулевым доверием, благодаря неизменность хранящихся на узлах данных, своим алгоритмам консенсуса. Однако на узлах дублируются данные, что приводит к их избыточности.

4. В результате обзора выделены несколько показателей эффективности систем распределения данных: избыточность данных, равномерность их распределения, надежность и свойство увеличения количество узлов и данных на них (масштабируемость).

5.Распределенные системы хранения различаются по поведению в зависимости от представленной ими модели хранения. Анализ архитектуры известных ИТ-проектов и подходов показал, что каждый из них не может решить одну из трех задач одновременно: избыточность данных, равномерное распределение данных по узлам, высокую надежность при минимизации необходимого места хранения для хранения данных и их резервных копии.

6. Необходимо предложить научный подход к распределению данных в блокчейн, который позволяет равномерно распределять данные по узлам без многократного дублирования. Кроме того, должны выполнятся требования к надежности всей системы.

7. Анализ литературы, посвященной моделям контроля доступа, показал, что существующие исследования не решают проблему контроля доступа на базе DAC и RBAC моделей в DApps, т.к. используемые в этих моделях матрицы доступа субъектов к объектам системы нельзя хранить в блокчейн при увеличении размера системы.

2. МОДЕЛЬ РАСПРЕДЕЛЕНИЯ ДАННЫХ, АЛГОРИТМЫ БАЛАНСИРОВКИ ДАННЫХ И МОДЕЛЬ КОНТРОЛЯ ДОСТУПА К НИМ ДЛЯ МИНИМИЗАЦИИ ОБЩЕГО РАЗМЕРА ДАННЫХ СИСТЕМЫ

2.1 Формализованная постановка задачи оптимизации распределения данных в распределенной среде

Размер используемой системы хранения S_{Output} можно рассчитать следующим образом:

$$S_{Output} = S_{Input} + S_{Reducancy} + S_{Metadata} + S_{AccessControl} \quad (2.1)$$

Однако S_{Input} определяется размером входящего объекта, поэтому его невозможно оптимизировать, поскольку пользователи определяют, какие объекты хранить. По сравнению с входными объектами метаданные и данные контроля доступа считаются небольшими:

$$S_{Metadata} + S_{AccessControl} \ll S_{Input} \quad (2.2)$$

$$S_{Output} \approx S_{Input} + S_{Reducancy} \quad (2.3)$$

С учетом этих параметров систему распределения можно представить как $DistrSystem < S_{Output}, \Delta S, R, \boldsymbol{N} >$, где $\Delta S = \sum_{i=1}^{|N|} |\Delta S_i|$, $\Delta S_i = (SS_i - S_i) \rightarrow 0$: $\forall i \in \boldsymbol{N}$,

$SS_i = \frac{C_i * S}{S + S}$: $\forall i \in N$, C_i— размер хранилища на узле i.Соответственно, цель данной работы может быть определена как результат следующей оптимизации:

$$max_R min_{S_{Output},\,\Delta S}\left(DistrSystem < S_{Output}, \Delta S, R, \boldsymbol{N} >\right) \quad (2.4)$$

2.2 Модели и алгоритмы распределения данных в DApps

Для решения проблемы избыточности данных в DApps, а также сохранение надежности на уровне централизованных систем и сохранения свойства масштабируемости, предлагается новая модель распределения. Для её описания рассмотрим следующие элементы:

- Архитектура системы на базе модели, которая описывает, какие компоненты существуют в системе, как они распределены и как взаимодействуют друг с другом.
- Алгоритмы балансировки данных, которые используются для распределения данных между узлами в любой момент времени, чтобы сделать систему сбалансированной и обеспечить необходимый уровень надежности. Распределение данных оптимизируется во время различных событий, таких как добавление новых узлов хранения или удаление узлов, и т. д.

Архитектура типовой DApps системы состоит из следующих частей (см. Рисунок 2.1)

Рисунок 2.1 — Архитектура DApps системы

Важно отметить, что технология стирающего кодирования [7, 65, 66] адаптирована и развернута в системе хранения. Стирающее кодирование – это тип кодирования, который принимает k входных данных одинаковой длины, которые могут быть потоком данных или сигналов и т. д., и выдает на выходе n сигналов, созданных на основе входных сигналов и функция стирающего кодирования. На основе подмножества входных данных и выходных данных (k`) можно получить все исключенное подмножество входных данных, как показано на рисунке 2.2.

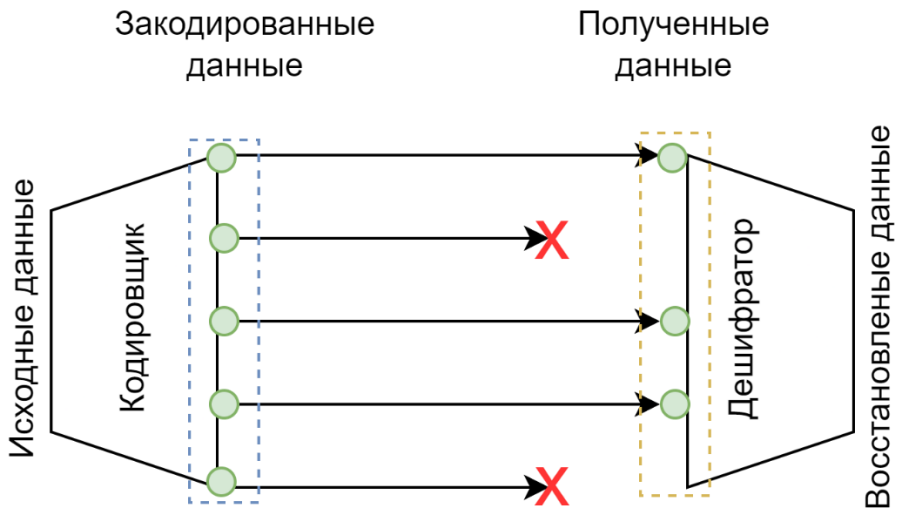


Рисунок 2.2 — Модель кодирования стирающего кодирования

Функция XOR рассматривается как пример функции стирающего кодирования. В XOR, как и в таблице 2.1, видно, что длина выходного сигнала равна длине первого и второго входных сигналов. Если B потеряно, его можно восстановить, используя выход XOR и A. Например, если известно, что A равно 0, а выход равен 1, это означает, что B должно быть равно 1. С другой стороны, если A потеряно, а B все еще существует, можно найти A, используя B, и выход XOR, используя ту же логику. Исходя из этого, можно восстановить либо A, либо B, используя выходные данные функции стирающего кодирования и оставшийся входной сигнал того же размера вместо репликации A и B, что приводит к уменьшению требуемого объема памяти/пропускной способности на 25%.

Таблица 2.1 — Функция XOR

Входы		**Выходы**
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Архитектуру децентрализованной системы хранения, использующей блокчейн для децентрализованного управления, можно представить как на рисунке 2.3.

Существует несколько алгоритмов, которые используют специальные кодировки для минимизации размера данных. RAID-5 и RAID-6 считаются одними из самых популярных реализаций стирающего кодирования. Другие типы RAID не рассматриваются, поскольку они менее популярны или основаны на этих реализациях, таких как комбинация использования стирающего кодирования с чередованием данных по группам дисков (RAID-50).

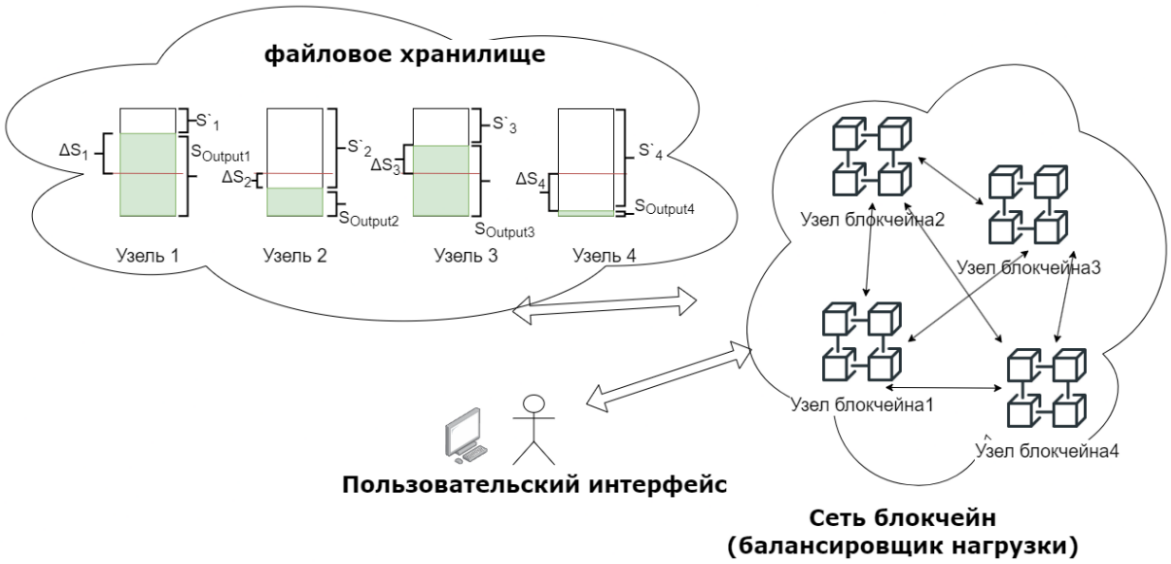


Рисунок 2.3 — Архитектура децентрализованной системы распределения данных на основе блокчейна

1. **Балансировщик нагрузки**: Балансировщик нагрузки обеспечивает необходимые функции управления и распределения данных. Он отвечает за следующие задачи:

a) Управление узлами хранения: управление существующими узлами, контроль процессов присоединения и выхода к узлам и т. д.

b) Управление данными: управление размещением существующих файлов (или объектов) и определение места добавления новых файлов. Предлагаемая модель распределения использует виртуальные объекты хранения, называемые виртуальными кластерами (VC) и виртуальными дисками (VD), которые будут объяснены в пункте 2 «Узлы хранения». Балансировщик нагрузки отвечает за процессы, связанные с этими виртуальными объектами, например за их распространение. Разработано четыре алгоритма распределения данных между узлами.

c) Управление пользователями: добавление и удаление пользователей, а также управление контролем доступа.

Балансировщик нагрузки во многих системах, таких как Lustre и HDFS, представлен одним узлом. Некоторые системы, такие как Ceph, используют несколько резервных узлов для балансировки нагрузки, поэтому, если основной балансировщик нагрузки выходит из строя, один из резервных узлов используется в качестве балансировщика нагрузки. Хотя этот метод помогает избежать единых точек отказа, основной узел балансировки нагрузки по-прежнему считается узким местом, поскольку все пользователи должны подключаться к одному и тому же узлу. В предлагаемой системе блокчейн используется для запуска балансировщика нагрузки в форме смарт-контракта. В этом случае пользователи могут подключиться к любому узлу системы блокчейн, и все узлы синхронизируются из-за алгоритма консенсуса.

2. **Узлы хранения**: Система хранения объектов системы (в обобщенном случае - файлов) состоит из узлов (одноранговая сеть P2P, где все узлы имеют одинаковую роль). Файлы организованы в виртуальные кластеры (VC). В свою

очередь, каждый из которых состоит из трёх виртуальных дисков фиксированного размера (VD). Это сделано для того, чтобы при увеличении VC и файлов в системе на основе рассматриваемой модели время работы алгоритмов управления распределением файлов по виртуальным дискам не зависело от общего количества файлов в системе. Очевидно, что количество VD в системе будет гораздо меньше количество файлов в ней. Важность минимизации сложности работы алгоритмов возрастает, поскольку предлагаемые алгоритмы реализованы на смарт-контрактах. Когда файл загружается пользователем или внешней информационной системой в систему хранения, он разбивается на две части и рассчитывается соответствующий стирающий код. Важно отметить, что размер стирающего кода равен размеру половины файла. В каждом кластере виртуальные диски организованы как индексированный массив. Первый VD используется для хранения первых половин файлов, второй VD для соответствующих вторых половин и третий для хранения стирающего кода. Такая схема распределения данных приводит к минимизации необходимого хранилища резервных данных на 25% по сравнению с системами хранения с одним сервером резервного копирования. На рисунке 2.4 представлена концепция предлагаемой модели. VC6 состоит из 3-х виртуальных дисков. Первый виртуальный диск в VC6 расположен в узле 2, второй виртуальный диск — в узле 6, а последний — в узле 4.

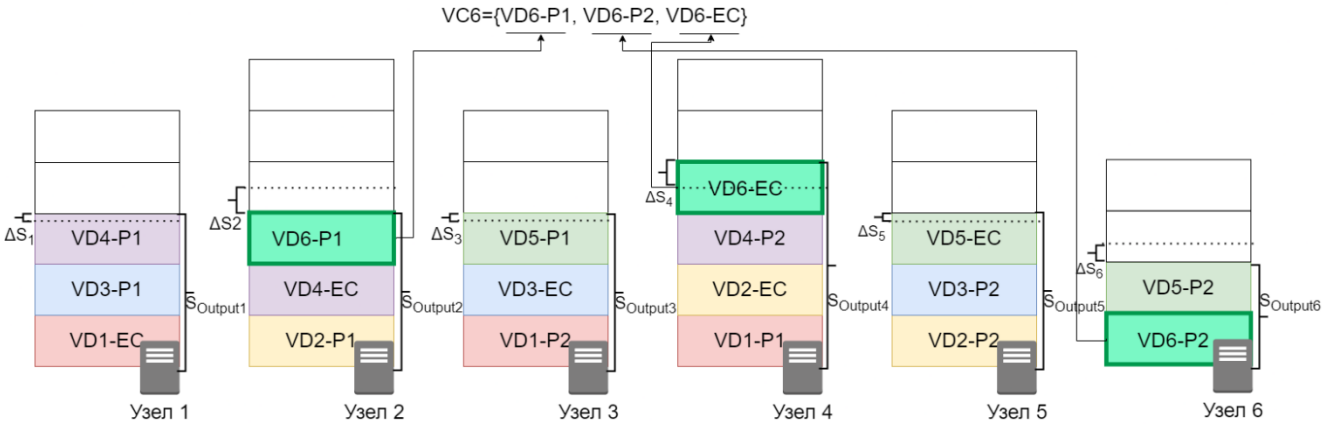


Рисунок 2.4 — Распределение VCs и VDs по узлам хранения

3. **Пользовательский интерфейс:** позволяет пользователю взаимодействовать с системой. Пользовательский интерфейс также отвечает за операции разделения и объединения файлов вместо узлов хранения, чтобы снизить нагрузку на систему.

Необходимые определения для алгоритмов управления данными в таблице 2.2.

Таблица 2.2 — Необходимые определения для модели распределения файлов

$\boldsymbol{VD_S}$	Виртуальный кластер в системе
$\boldsymbol{VD`_S}$	Виртуальный диск в системе
$\boldsymbol{VC}$	Множество виртуальных дисков в системе
$\boldsymbol{VD_{n(i)}}$	Множество пустых слотов виртуальных дисков в системе
$\boldsymbol{VD_{n(i,\,x)}}$	Множество виртуальных кластеров в системе
$\boldsymbol{VD`_{n(i)}}$	Подмножество $\boldsymbol{VD_S}$ расположенных в узле i
$state(d)$	Подмножество $\boldsymbol{VD_S}$ расположенных в узле i, которые имеют состояние x
$\boldsymbol{VD_{index(i,\,j)}}$	Подмножество $\boldsymbol{VD`_S}$ расположенных в узле i.
$\boldsymbol{VD_{S(x)}}$	Функция, возвращающая состояние виртуального диска d. VD представляет одно из состояний hot(горячим), cold(холодным) или normal (обычным)
$vd_{vc,\,p}$	Элемент множества $\boldsymbol{VD_S}$, расположенный в узле i и имеющий порядок j
$\boldsymbol{VC_{n(i)}}$	Подмножество $\boldsymbol{VD_S}$, которые имеют состояние x
$\boldsymbol{VC_i}$	Элемент множества $\boldsymbol{VD_S}$, представляющий vd, принадлежащий виртуальному кластеру vc и имеющий часть p. p может быть 1 для первой части, 2 для второй части, 3 для стирающего кодирования
$fsr(i)$	Подмножество $\boldsymbol{VC}$, которым принадлежат узлы i

vc_d	Элемент множества $\boldsymbol{VC}$ с индексом i
$p(d)$	(Free space to request ratio) функция, возвращающая отношение свободного места к запросу для узла i
$mcvd(x)$	Элемент множества $\boldsymbol{VC}$, который содержит конкретный виртуальный диск d
$MTTF_{system}$	Функция, возвращающая номер части виртуального диска d
$MTTF_{node}$	(Mean count of VDs) – функция, которая возвращает среднее количество VD в состоянии x на узел
$MPRC$	Время наработки на отказ всей системы
P_{SFANF}	Время наработки на отказ узла
$MTTR$	Максимально возможное количество процессов восстановления

2.2.1. Алгоритмы балансировки данных

При работе со смарт-контрактами существует несколько ограничений, которые учитываются при реализации алгоритмов балансировки данных, поскольку эти алгоритмы работают на смарт-контрактах:

1. Ограничения языка программирования: не все языки программирования поддерживаются в сети блокчейн: Многие сети блокчейнов используют язык программирования, специально разработанный для смарт-контрактов, например Solidity.

2. Сложные типы данных и структуры не поддерживаются, что ограничивает возможности работы с переменными. Например, сеть блокчейн Ethereum (Solidity) ограничивает сложность алгоритмов за счет расходования комиссий (Ethereum gas), а блокчейн платформа Hyperledger ограничивает максимальное время выполнения транзакций.

3. Ограниченные библиотеки и функциональность: смарт-контракты имеют ограниченные библиотеки и функциональность по сравнению с программированием общего назначения. Например, Solidity (один из самых популярных языков для написания смарт-контрактов) не поддерживает

модификаторы, наследование, перегрузку функций и операторов, бесконечные циклы, рекурсии и числа с плавающей точкой.

Видно, что не все алгоритмы подходят для работы в рамках смарт-контрактов, особенно алгоритмы, которые включают сложные переходы между состояниями или сложные структуры данных. Алгоритмы высокой сложности не подходят для работы на блокчейне.

Предложенные алгоритмы учитывают несколько ограничений при балансировке данных, в том числе поддержание равного распределения данных на основе свободного пространства узлов и размера самих данных. Также в смарт-контрактах реализуется требование о том, чтобы система не содержала на одном узле двух VD, принадлежащих одному и тому же VC, чтобы сделать возможность восстановления данных в случае отказа узлов и проверка состояния использования VD (горячие – VD, к которым обращаются чаще всего, холодные – VD используются для стирающего кодирования и обычные VD – остальные VD).

Распределение данных и то, как виртуальные диски (и виртуальные кластеры) распределяются по узлам, управляются алгоритмами смарт-контрактов. На это распределение влияют несколько событий, которые влияют на баланс данных в системе (рисунок 2.5). Есть четыре основных события, которые напрямую влияют на это распределение: добавления узла, добавления файла, отказ узла и возобновления узла. Существуют и другие события, которые могут влиять на распределение данных по узлам, но отдельные алгоритмы для этих событий не разрабатывались, так как соответствующие алгоритмы можно сделать на основе 4-х упомянутых выше алгоритмов. К этим событиям относятся запуск системы, удаление файлов и т. д.

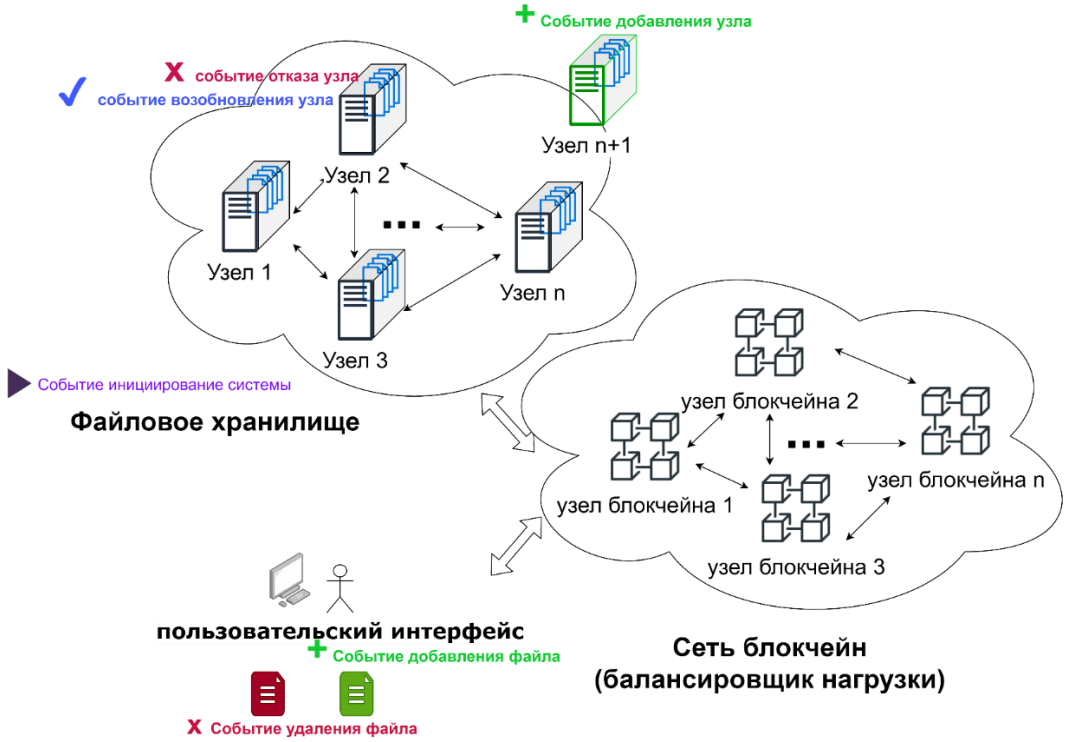


Рисунок 2.5 — События, влияющие на распределение данных в системе

2.2.2. Алгоритм добавления новых узлов (A1)

При добавлении нового узла рассчитывается общее свободное пространство и максимальное пространство для хранения виртуальных устройств. Также необходимо учитывать количество горячих VD (виртуальных устройств, которые используются чаще всего). Горячие VD должны быть распределены поровну. Для определения списка «горячих» VD можно использовать алгоритм Least Recently Used (LRU), где количество слотов для алгоритма LRU определяется как общее количество кластеров, разделенное на количество узлов. Узлы сортируются по количеству «горячих» узлов в порядке убывания. В этом алгоритме мы вычисляется идеальное количество горячих VD в каждом узле по формуле 2.5:

$$mcvd(x) = \frac{\left|\boldsymbol{VD}_{\boldsymbol{S}(x)}\right|}{|\boldsymbol{N}|} \tag{2.5}$$

Состояние диска, которое алгоритм использует вначале — «горячие VDs».

Для каждого узла (n) списка, начиная с первого элемента, выбирается набор горячих устройств исходя из условий, определенных формулой 2.6:

$$\forall\, d \in \boldsymbol{VD_{n(i,x)}}; i \in \{1,2, \ldots |\boldsymbol{N}|\}:$$

$$ChooseVD(d) = \begin{cases} 1 & \boldsymbol{VD_{n(|N|,x)}} \cup \{d\} \le mcvd(x) \\ & \boldsymbol{VD_{n(i,x)}}\backslash\{d\} \ge mcvd(x) \\ & vd_{vc_d,\, p(d)\%3+1} \notin \boldsymbol{VD_{n(|N|,x)}} \\ & vd_{vc_d,\, (p(d)+1)\%3+1} \notin \boldsymbol{VD_{n(|N|,x)}} \\ & |\boldsymbol{VD`_{n(|N|)}}| > 0 \\ 0 & otherwise \end{cases} \quad (2.6)$$

$$\boldsymbol{VD_{n(|N|)}} = \begin{cases} \boldsymbol{VD_{n(|N|)}} \cup \{d\} & ChooseVD(d) = 1 \\ \boldsymbol{VD_{n(|N|)}} & otherwise \end{cases} \quad (2.7)$$

$$\boldsymbol{VD_{n(i)}} = \begin{cases} \boldsymbol{VD_{n(i)}}\backslash\{d\} & ChooseVD(d) = 1 \\ \boldsymbol{VD_{n(i)}} & otherwise \end{cases} \quad (2.8)$$

Эти условия определяют способ выбора виртуального диска, который учитывает количество выбранных виртуальных дисков из каждого узла, количество виртуальных дисков определенного состояния в каждом узле, требование о том, что узел не может иметь два виртуальных диска, принадлежащих одному узлу. том же кластере, чтобы иметь возможность восстановить данные в случае потери данных и т. д.

Процесс повторяется для холодных VD (дисков, содержащих стирающего кодирования, поскольку они используются только для целей восстановления данных), поэтому новый узел получает указанное количество горячих и холодных устройств. Оставшееся необходимое количество требуемых VD достигается выбором VD, которые не являются ни горячими, ни холодными.

Рисунок 2.6 содержит шаги алгоритма.

2.2.3. Алгоритм добавления файлов (A2)

В этом алгоритме балансировщик нагрузки выбирает VC или создает новый на основе размера файла и свободного места в виртуальных кластерах в системе, как в формуле 2.11:

$$\boldsymbol{PVC} = \{c \in C \mid freespace(c) > fileSize\} \tag{2.9}$$

$$g(j) = \min\{freespace(e)\} : \forall\, e \in j \tag{2.10}$$

$$Selected\_VC = \begin{cases} g^{-1}(\boldsymbol{PVC}) & |\boldsymbol{PVC}| > 0 \\ \boldsymbol{VC}_{|\boldsymbol{N}|+\boldsymbol{1}} & else \end{cases} \tag{2.11}$$

Приложение на стороне пользователя разделяет файл и вычисляет соответствующее стирающее кодирование. На рисунке 2.6 представлен полный алгоритм.

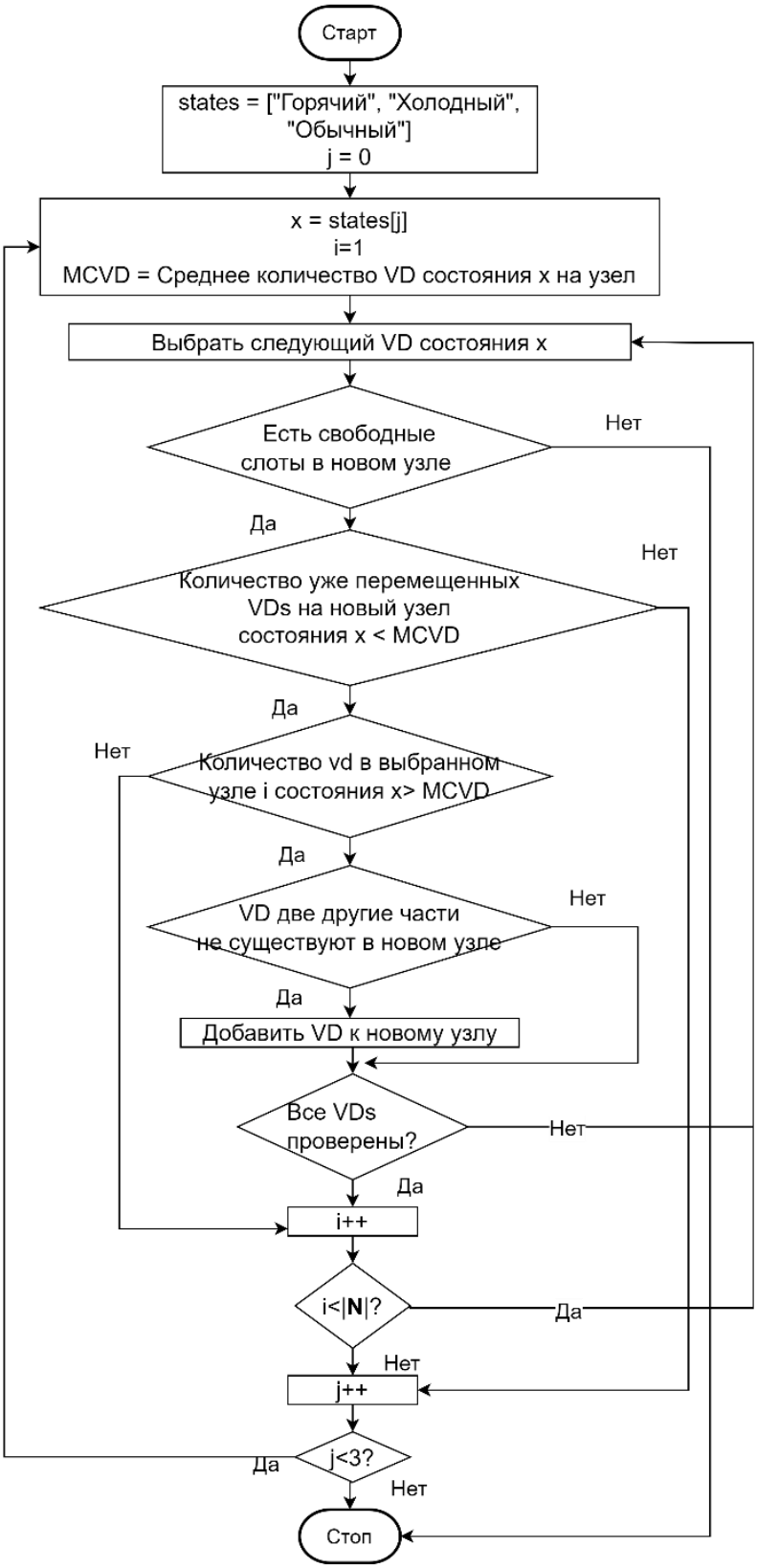


Рисунок 2.6 — Алгоритм добавления узлов (A1)

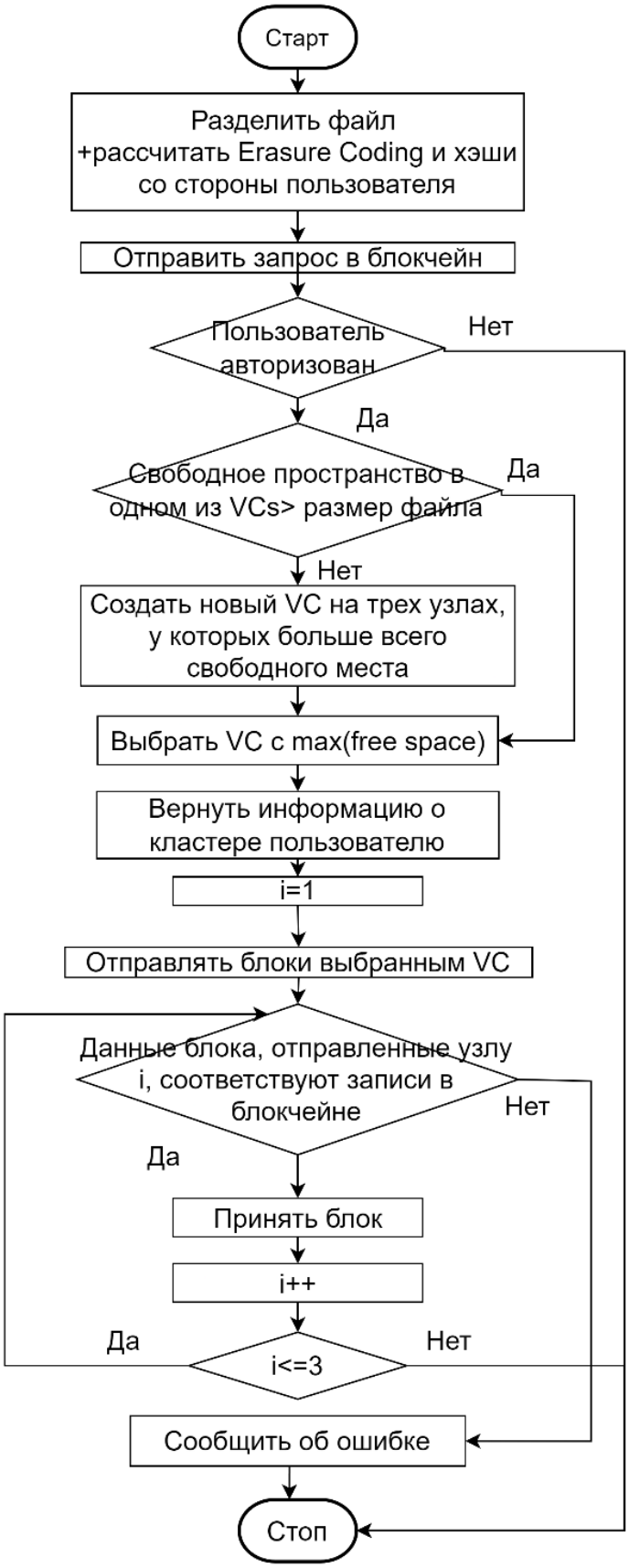


Рисунок 2.7 — Алгоритм добавления файлов (A2)

2.2.4. Алгоритм восстановления потерянных данных (A3)

Этот алгоритм используется для восстановления узлов в случае сбоя узла. Определенные контрольные сообщения регулярно отправляются со всех узлов в балансировщик нагрузки. Эти сообщения показывают, что узел все еще работает. Когда узел не отправляет сообщение в течение определенного количества последовательных интервалов, он попадает в список «автономных устройств». Когда узел недоступен, его виртуальные диски необходимо перестроить в остальные существующие узлы. Когда узел отключается от сети, для восстановления данных требуется восстановление потерянного виртуального диска. Этот процесс имеет некоторые требования:

1) VD не может сосуществовать в одном узле с другим VD, принадлежащим тому же VC.

2) Данные должны распределяться справедливо, так как процесс восстановления данных должен учитывать все доступные узлы.

Математически эту задачу можно рассматривать как классическую задачу теории игр, поскольку выигрыш может быть достигнут с помощью стратегии (способа распределения данных). Следовательно, при большом количестве VD и узлов, задачу восстановления невозможно решить за полиномиальное время. Раздача VD — это стохастическая игра, так как это повторяющаяся игра, в которой будущие действия зависят от текущего действия. В этом случае совершение действия приводит не к совершенно новой игре, а к игре, в которой множество возможных действий меньше на 1. Игроком в этой игре является система, и она имеет конечное множество узлов, и конечное множество VD. Цель этой игры — справедливо распределить виртуальные диски по узлам с учетом двух упомянутых выше условий. Эту игру можно определить кортежем (Q, N, A, P, R), где:

- Q — конечное множество состояний игры, где в каждом состоянии имеется распределение VD
- N — конечное множество игроков, равное одному (системе)

- $A = A_1 \times \cdots \times A_n$, где A_i — это конечный набор доступных действий (например: перестроить VD5, VD7 и VD15 в узле 1, а VD1, VD6 в узле 2 и т. д.).
- $P: Q \times A \times Q \rightarrow [0,1]$ представляет собой функцию вероятности перехода. P(q, a, q̂) - вероятность перехода из состояния q в состояние q̂ после выполнения действия a.
- $R = r_1, \ldots, r_n$, где r_i: $Q \times A \rightarrow R$ является результирующим выигрышем. Выигрыш в этой задаче может определяться несколькими переменными: Выбор узла для VD, где этот узел имеет другой VD в том же кластере, дает отрицательный выигрыш. Отправка VD на узел, который сохраняет «справедливое» распределение, приводит к положительному результату и так далее. Хотя существует стохастическая игра с одним агентом, эта проблема представляет собой марковский процесс принятия решений. После определения всех возможных состояний выбирается путь, гарантирующий наибольший выигрыш. При большом количестве VD и узлов задачу невозможно решить за полиномиальное время. Эта проблема считается NP-полной проблемой. Реализация такого алгоритма в среде смарт-контрактов нецелесообразна. Вместо этого предлагается новый алгоритм.

Этот алгоритм определяет функцию fsr (Free space to request ratio). При отказе узла список возможных узлов для каждого виртуального диска, существующего на потерянном узле, определяется на основе нескольких факторов, таких как количество доступных пустых слотов виртуального диска на каждом узле и отсутствие конфликта с соответствующим виртуальным диском. На основе этого каждый узел составляет список виртуальных дисков, которые он может принять. fsr — это отношение количества свободных слотов виртуальных дисков к сумме этих запросов. Сложность предлагаемого способа составляет O(n). Этот алгоритм обеспечивает возможность восстановления нескольких последовательных сбоев узлов, пока в узлах не останется свободного места или не останется менее трех узлов.

$$fsr(i) = \frac{\boldsymbol{VD`_{n(i)}}}{\left|\boldsymbol{VD_{n(lost\_node\_id)}}\right| - \left|\boldsymbol{VC_{n(i)} \cap VC_{n(lost\_node\_id)}}\right|} \qquad (2.12)$$

$$hdvc(i) = \left|\boldsymbol{VD}_{\boldsymbol{n(i,hot)}}\right| \quad (2.13)$$

$$m = \max\{fsr(i)\colon\ i = 1..|N|\backslash\{lost\_node\_id\}\} \quad (2.14)$$

$$j = fsr^{-1}(m) \quad (2.15)$$

$$l(j) = \begin{cases} \min\{hdvc(k)\}\colon\ \forall k \in j & state(d) = hot \\ \max\{hdvc(k)\}\colon\ \forall k \in j & else \end{cases} \quad (2.16)$$

$$Selected\_node\_for\_d = l^{-1}(j) \quad (2.17)$$

2.2.5. Алгоритм возобновления узлов (A4)

Если узел, вышедший из строя ранее, возобновляет работу, то вычисляется хэш каждого из его виртуальных дисков и сравнивается с восстановленными. Виртуальные диски, хэш которых не совпадает, заменяются восстановленными. Если хэш совпадает, соответствующая запись в смарт-контракте обновляется. В процессе восстановления, если файлы изменяются на виртуальном диске, создается буфер, который содержит только изменения, и эти изменения объединяются после завершения процесса. Алгоритм описан на рисунке 2.9.

Рисунок 2.8 — Алгоритм восстановления потерянных данных (А3)

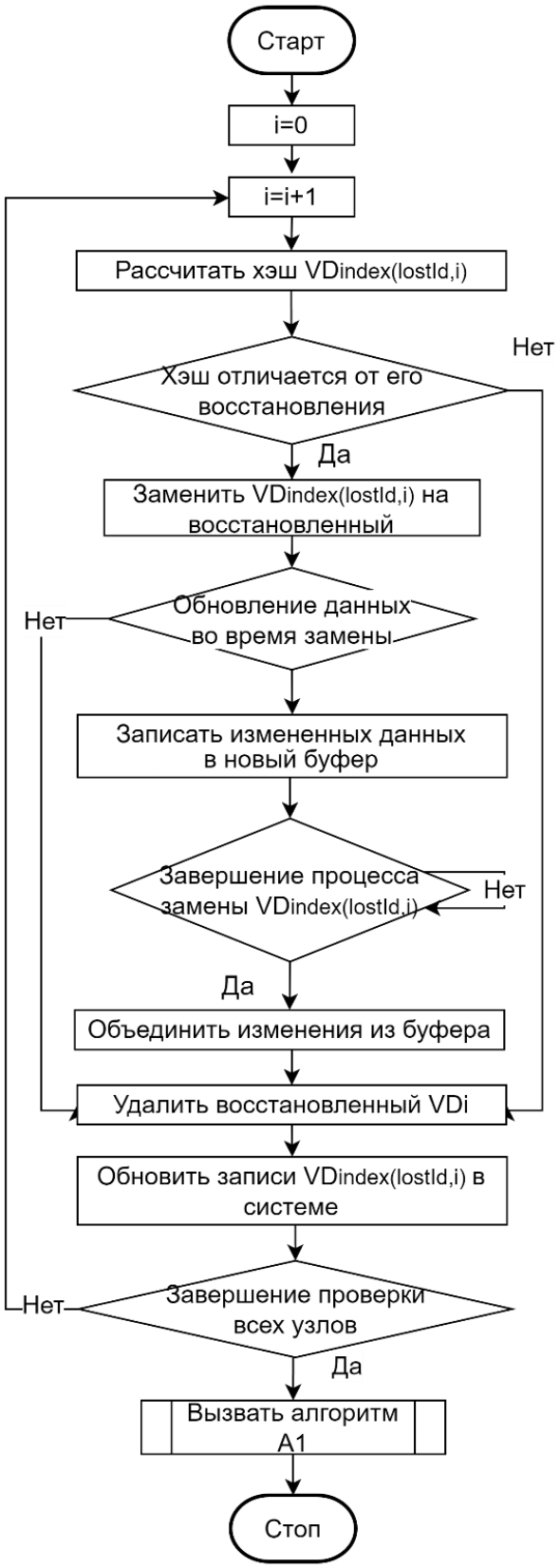


Рисунок 2.9 — Алгоритм возобновления узлов (A4)

Система, в которой реализована модель распределения объектов, далее в этом исследовании будет называться DecStore.

2.3 Модели и алгоритмы контроля доступа в децентрализованной системе хранения данных

Рассмотрим распределенную модель контроля доступа, которая минимизирует размер данных в блокчейне. Эта модель состоит из нескольких алгоритмов и конкретной архитектуры, а также новой техники кэширования. Эта модель также адаптирует особый тип деревьев, называемый деревьями Меркла.

Дерево Меркла [17, 83, 115]— это тип деревьев, который используется для быстрого поиска изменений данных в дереве или проверки принадлежности узла дереву без прохождения всех узлов дерева. Листья дерева Меркла — это хэши необходимых блоков данных или транзакций, как в блокчейне. Это дерево является двоичным, поэтому каждый узел имеет два дочерних узла. Каждый промежуточный узел представляет собой хэш суммы двух его дочерних узлов. На рисунке 2.10 представлен пример дерева Меркла. Например, A,B,C,D могут быть файлами, а листья дерева представляют собой хэши соответствующих файлов. Если файл A изменяется, в результате изменяется его хэш, а это означает, что родительский узел (хэш(A+B)) тоже изменяется. Это означает, что значение корневого узла Меркла изменяется, поскольку оно представляет собой сумму хэшей узлов 1 и 2.

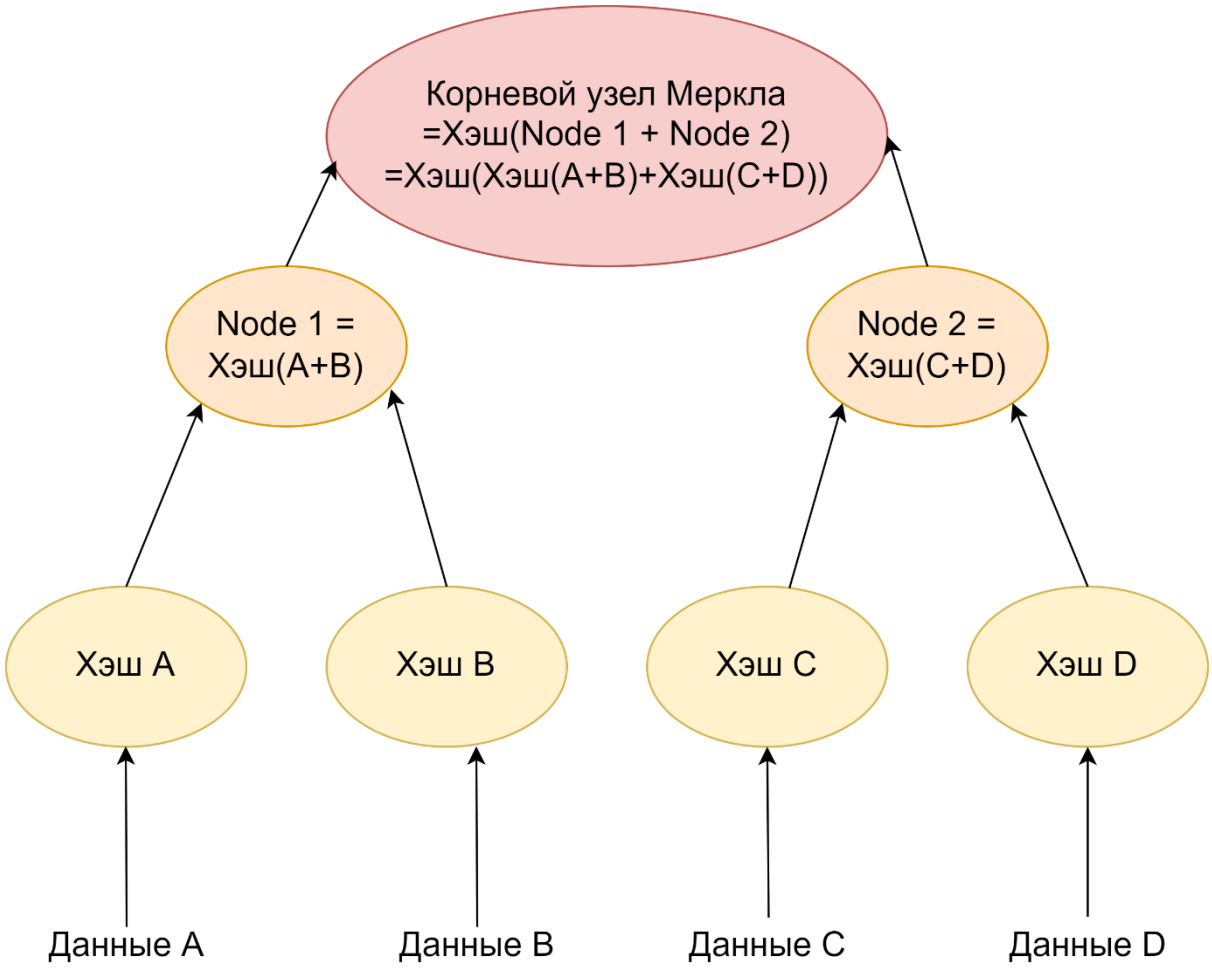


Рисунок 2.10 — Пример дерева Меркла

Кроме того, легко подтвердить, что узел A существует в дереве. Если указано значение узла A, в дополнение к хэшу B и хэшу узла 2, можно подтвердить, что узел A принадлежит дереву, объединив хэш узла A с хэшем B и вычислив хэш этой суммы, который будет быть равным хэшу узла 1, а затем объединить его с хэш-узлом 2 и вычислить хэш суммы узла 1 и узла 2. Полученное значение должно быть равно хэш-значению корня Меркла. Это означает, что для доказательства существования узла в дереве требуется только подмножество дерева. Для проверки принадлежности листа дереву необходимо отправить требуемое значение листа в дополнение к хэшам $\log_2$ количество листьев. Это означает, что это небольшое количество узлов. В этом примере существует 4 листа. Вот почему необходимо был отправить Hash A

вместе с $\log_2 4 = 2$ хэша. Если узел состоит из 1,000,000 записей, помимо необходимого листа требуется только 20 хэшей. Необходимые данные, используемые для доказательства принадлежности узла дереву, называются доказательством Меркла.

2.3.1. Модель контроля доступа

Чтобы создать модель контроля доступа для DecStore, которая поддерживает масштабируемость и требует минимального хранения данных в блокчейне, архитектуру рассмотренной выше модели распределения необходимо изменить следующим образом:

- Каждый виртуальный диск (VD) снабжен блоком, который называется «Компонент хранения разрешений». Для каждого пользователя, имеющего доступ хотя бы к одному файлу в указанном VD, в этом блоке создается дерево. Это дерево представляет файлы, организованные в каталогах, к которым пользователь имеет доступ через DAC. Единицы хранения, расположенные на VD, принадлежащих одному VC, имеют одинаковую копию деревьев.
- Каждый узел хранения имеет одно дерево для каждого пользователя, которое представляет собой комбинацию его деревьев DAC на VD, существующих в этом узле хранения.
- На стороне блокчейна у каждого пользователя хранится 1 хэш, который представляет собой общий хэш корня его дерева Меркла [77]. Если файлов миллион и пользователей 100, будет сохранено только 100 хэшей. Таблица (2.3) показывает, как информация о доступе пользователей хранится в блокчейне. Корневой хэш дерева Меркла в этой таблице представляет собой хэш корня дерева Меркла пользователя DAC.

Таблица 2.3 — Хранение данных контроля доступа DAC в блокчейне

Пользователь	**Хэш корня дерева Меркла**	**Роли**
f970e2767d0cfe75876ea857f92e319b	006d2143154327a64d86a264aea225f3	ADMINISTRATOR
7694f4a66316e53c8cdd9d9954bd611d	76d80224611fc919a5d54f0ff9fba446	DEVELOPER, TEAM-LEADER
…	…	

Также роли хранятся в блокчейне аналогично, как в таблице (2.4). Для каждой роли строится дерево Меркла, в котором файлы, доступные этой роли, рассматриваются как листья дерева.

Таблица 2.4 – Хранение данных контроля доступа RBAC в блокчейне

Роль	**Хэш корня дерева Меркла**	**Узлы**
ADMINISTRATOR	0cc175b9c0f1b6a831c399e269772661	1,4
DEVELOPER	8a8bb7cd343aa2ad99b7d762030857a2	2,3
…	…	

Типовая гибкая система контроля доступа управляется политиками и состоит из [2]:

- Точка администрирования политики (PAP): это компонент, который создает политики доступа.

- Точка информации о политике (PIP): это источник, содержащий информацию о политике, на основе которой пользователю предоставляется или запрещается доступ.

- Точка принятия решения о политике (PDP): это компонент, который решает, предоставлен ли пользователю доступ или нет.

- Точка применения политики (PEP): это компонент, который создает запрос и отправляет его в PDP. PEP собирает необходимую информацию из PIP и других ресурсов, например информацию о запросах пользователей в PDP.

- Точка извлечения политики (PRP): этот компонент не является обязательным. Он используется только тогда, когда имеется несколько PDP и требуется предоставить централизованную точку для отправки или получения политик доступа.

В модели, предложенной в этом исследовании, точка информации о политике (PIP) не хранится на централизованном сервере. Когда PDP (который представлен в системе смарт-контрактом блокчейна) принимает решение, он получает информацию от двух объектов: пользователя и записей, хранящихся в блокчейне. Информация о полном доступе хранится на узлах хранения, и пользователи синхронизируют свои собственные деревья доступа с деревьями, расположенными на узлах хранения. Точка применения политики (PEP), которая также представлена смарт-контрактом, отправляет в PDP доказательство дерева Меркла из запроса пользователя вместе с соответствующим ожидаемым корневым хэшем дерева Меркла.

Таким образом, PIP представлен тремя сущностями:

- Распределенная сеть узлов хранения, которая имеет полные политики и используется только для того, чтобы позволить пользователям синхронизировать свои политики.
- Сторона пользователя, где каждый пользователь хранит свои политики, будь то DAC или RBAC.
- Корневые хэши дерева Меркла как для пользователей, так и для ролей, которые хранятся в блокчейне.

Деревья пользователей состоят из трех типов узлов:

Корень – представляет собой общий хэш дерева Меркла.

Листья: представляют файлы (или каталоги, если политика разрешает пользователю доступ ко всем файлам в каталоге вместо выбора файлов вручную). Они могут иметь необязательное значение «compress». Это будет обсуждаться в разделе «Алгоритм сжатия деревьев».

Промежуточные узлы: они могут быть каталоги или объединяющие узлы. Объединяющий узел — это узел, который объединяет два узла различных типов,

причем любой из этих двух узлов может быть листом, каталогом или объединяющим узлом. Этот тип узлов будет обсуждаться далее в разделе «Алгоритм преобразования дерева в двоичное».

Каждый узел дерева содержит две связанные структуры данных: хэш и значение. Хэш представляет собой хэш значения. В таблице 2.5 представлены значения каждого типа узла.

Таблица 2.5 – Типы узлов в предлагаемом дереве контроля доступа

Тип узла	Структура узла	Пример
Файл	*{* *Name = "File name",* *Type = File,* *Access = "Access type",* *Compressed = "compressed paths",* *Path = "Path"* *}*	*{* *Name = "document.pdf",* *Type = File,* *Access = R,* *Compressed = "",* *Path = "/documents/"* *}*
Каталог	*{* *Name = "Directory name",* *Type = Directory,* *Access = "Access type",* *Compressed = "compressed paths",* *Path = "Path"* *}*	*{* *Name = "documents",* *Type = Directory,* *Path = "/"* *}*
Объединяющий узел	*{* *Type = Combiner,* *Paths = ["paths array"]* *}*	*{* *Type = Combiner,* *Paths = []* *}*

На рисунке 2.11 представлена обновленная архитектура системы с поддержкой контроля доступа. Видно, что деревья ролей распределены по узлам хранения, где один узел хранения не хранит роль и ее копию вместе., а ее копию. Также внутри VD добавляются деревья доступа.

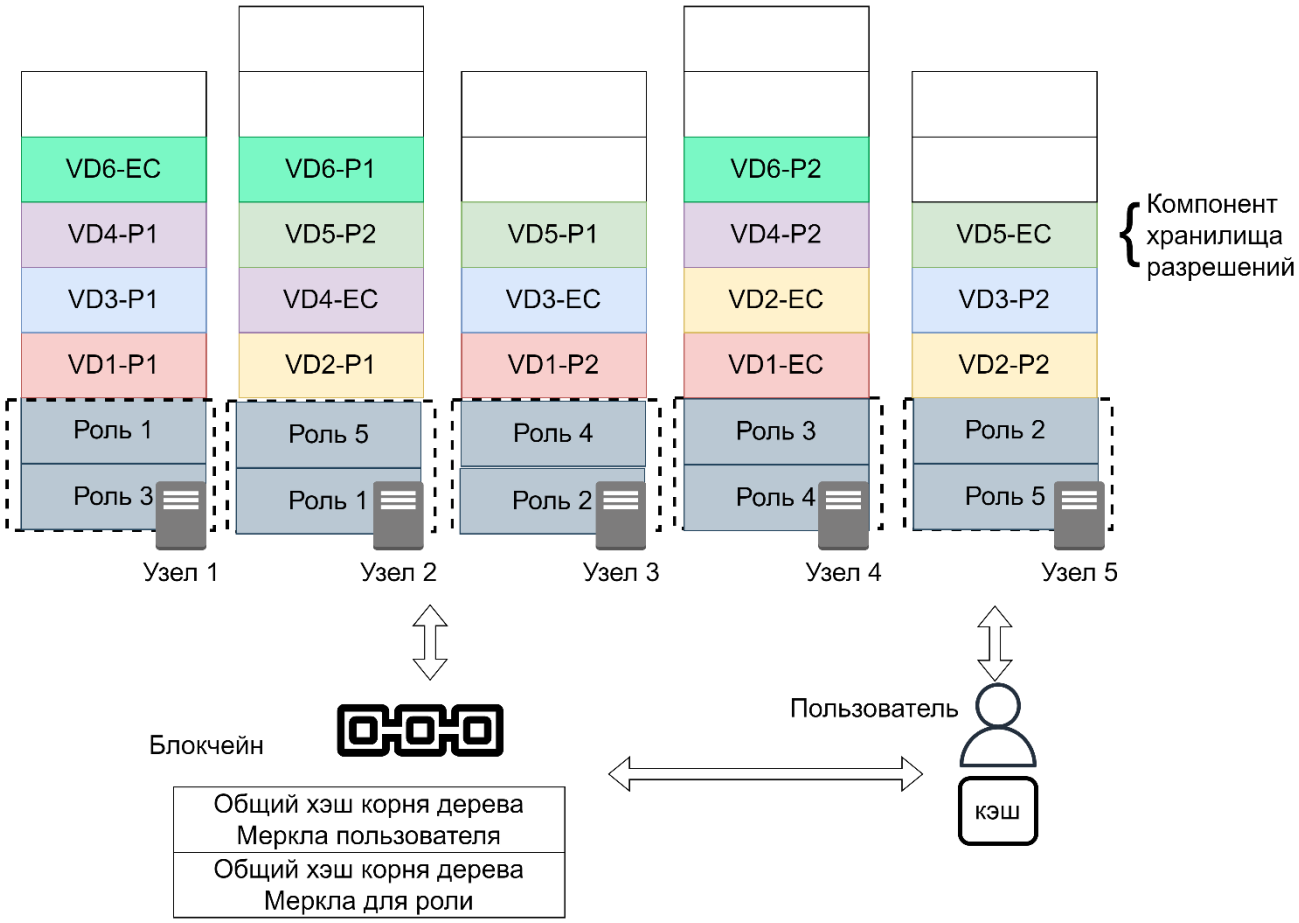


Рисунок 2.11 — Архитектура DecStore с предлагаемой моделью контроля доступа

Опишем основные этапы применения описанной модели:

1. В каждом узле хранения создается дерево для каждого пользователя. В этом узле имеется набор файлов, к которым имеет доступ пользователь, которые существуют на виртуальных дисках с определенным заранее выбранным индексом. Для каждого пользователя все деревья с VD в узле объединяются в

одно дерево, сжимаются и преобразуются в бинарное дерево. Эти деревья используются для DAC.

2. Также каждый узел хранения имеет несколько деревьев ролей. Существует дополнительная копия каждого дерева ролей, расположенная на другом узле. Все узлы хранения имеют одинаковое количество деревьев. Блокчейн распределяет деревья и их копии по узлам хранения.

3. Пользователи синхронизируют свои деревья с существующими на узлах хранения. Сюда входят их личные деревья (DAC) и деревья ролей (RBAC).

4. Пользователи используют свои копии деревьев для доступа к файлам, генерируя доказательство Меркла.

5. Блокчейн обрабатывает процессы, связанные с распространением и восстановлением деревьев в различных ситуациях, чтобы обеспечить балансировку системы и возможность восстановления файлов или автоматического восстановления на другой узел при потере узла хранения.

Далее рассмотрим описанные алгоритмы подробнее.

2.3.2 Алгоритм создания дерева Меркла

Сначала случай DAC будет подробно рассмотрен, а RBAC — в конце этого раздела. Алгоритм создания дерева Меркла состоит из трех основных этапов:

1. В каждом виртуальном диске, выбранном блокчейном, создается дерево для каждого пользователя. Это дерево показывает, к каким файлам на этом виртуальном диске имеет доступ пользователь.

2. Каждый узел объединяет эти деревья в одно дерево, которое показывает, к каким файлам пользователь имеет доступ в этом узле. В процессе объединения файлы из разных деревьев, расположенные в одной папке, объединяются в один узел дерева.

3. После этого дерево сжимается путем объединения узлов дерева, имеющих одного дочернего элемента, в один узел дерева, чтобы минимизировать размер дерева.

4. Полученное дерево преобразуется в двоичное дерево, чтобы минимизировать размер доказательства Меркла.

Информация о доступе пользователей распределяется по узлам хранения с помощью виртуальных дисков. Это означает, что данные реплицируются на виртуальные диски, принадлежащие одному и тому же VC. При создании объединенного дерева Меркла в этом процессе участвует только один из VD каждого VC, чтобы избежать избыточности, а значит, требуется выбрать, какой VD будет участвовать в этом процессе. Выбирать фиксированный VD для всех пользователей нецелесообразно, поскольку это означает большую нагрузку на конкретные VD, тогда как предпочтительнее распределять нагрузку по VD. Назначать набор VD каждому пользователю — не лучшая практика, поскольку это означает дополнительное хранение данных в блокчейне. Выбор VD в этом алгоритме динамический, основанный на формуле (2.18):

$$selectedVdIndex(user) = userId(user)\ mod\ 3 \qquad (2.18)$$

Если полученное число равно 1, это означает, что для построения дерева для данного пользователя выбран только первый VD во всех VCs. Если оно равно 2, это означает, что используется второй VD. Если он равен 0, это означает, что используется VD, который содержит стирающее кодирование. Начальный TreeNode в алгоритме является корневым узлом дерева.

Деревья представляются с помощью множеств множеств, где основным множеством является корень дерева, а элементы этого множества представляют собой дочерние узлы корня (файлы и каталоги), а также являются множествами, представляющими одно и то же понятие. Эти множества (узлы дерева) имеют атрибуты (имя, тип и т. д.). Файлы представлены пустыми множествами с атрибутами. Слова «узлы дерева» и «множества» будут использоваться как взаимозаменяемые. В каждом узле хранения выбираются деревья на основе выбранных VD (на основе выбранного индекса VD), которые будут объединяться

для построения единого дерева на уровне узла хранения. Узел дерева выражается множеством **TreeNode**, а корень дерева выражается как **TreeRootNode**. Первоначально выбранным **TreeNode** в алгоритме является **TreeRootNode**. Атрибут x корневого узла дерева может быть выражен как $\boldsymbol{TreeRootNode^x}$, а первый элемент набора — $\boldsymbol{TreeNode_1}$**.** Это не обязательно означает, что это дерево VC1. Это значит, что это дерево первого VD, хранящегося на узле хранения, который находится в списке выбранных VD. **CombinedTreeRootNode** можно рассчитать, выбрав объединение (∪) каждого **TreeRootNode**, участвующего в этом процессе. Начиная с этого, элементы узла дерева Node объединяются следующим образом:

$$\boldsymbol{TreeNode} = \{C = A \cup B \mid A \in Node\ and\ \exists B \in Node, B \neq A\ and\ A^{path} = B^{path}\} \cup \{A \mid \begin{matrix} A \in Node\ and \nexists B \in Node, \\ B \neq A\ and\ A^{path} = B^{path} \end{matrix}\} \quad (2.19)$$

Это означает, что он может содержать элементы из разных деревьев в одних и тех же каталогах или может быть одним элементом, если в каталоге есть только один файл.

Рисунки (2.12) и (2.13) описывают алгоритм объединения деревьев VD в одно дерево. Пример соединения деревьев представлен на рисунках 2.14 и 2.15.

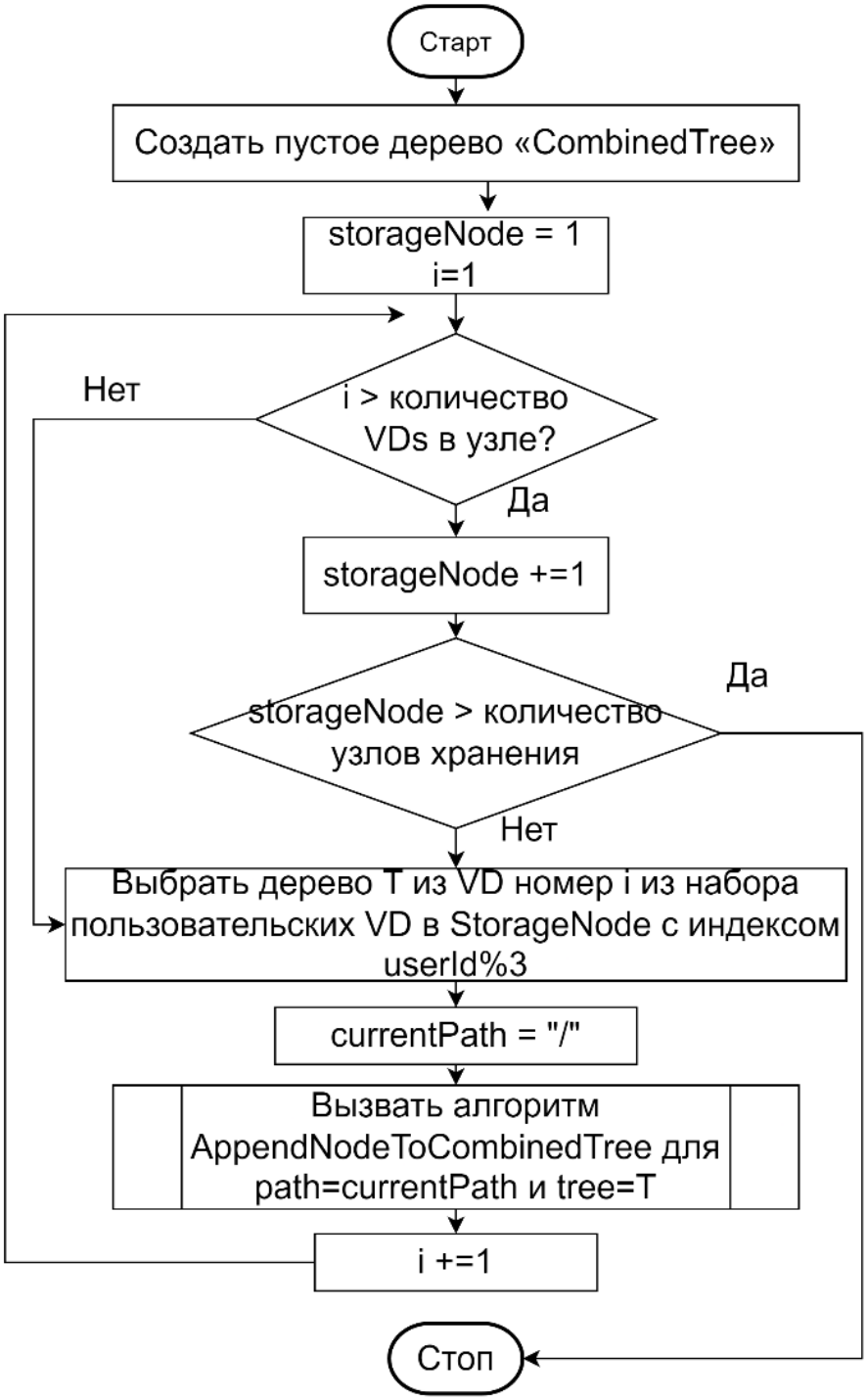


Рисунок 2.12 — Алгоритм объединения деревьев

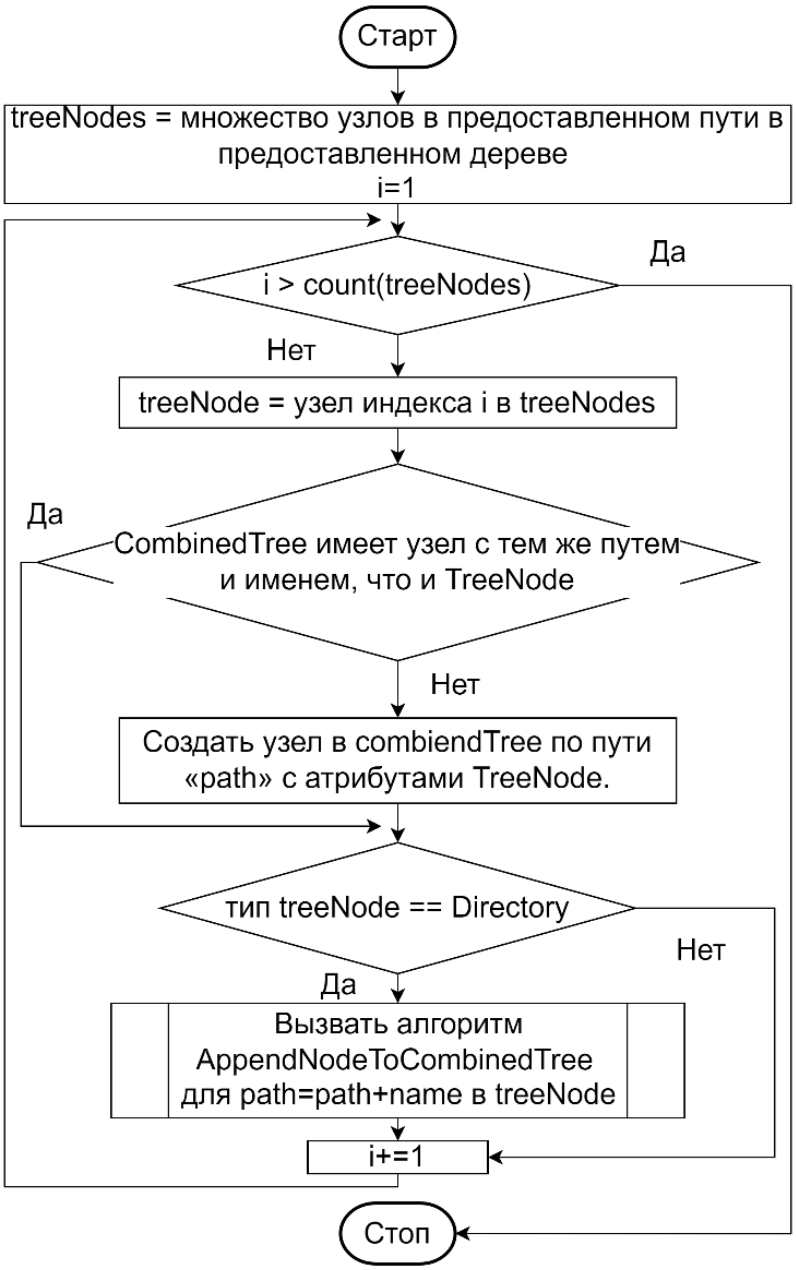


Рисунок 2.13 — Алгоритм AppendNodeToCombinedTree

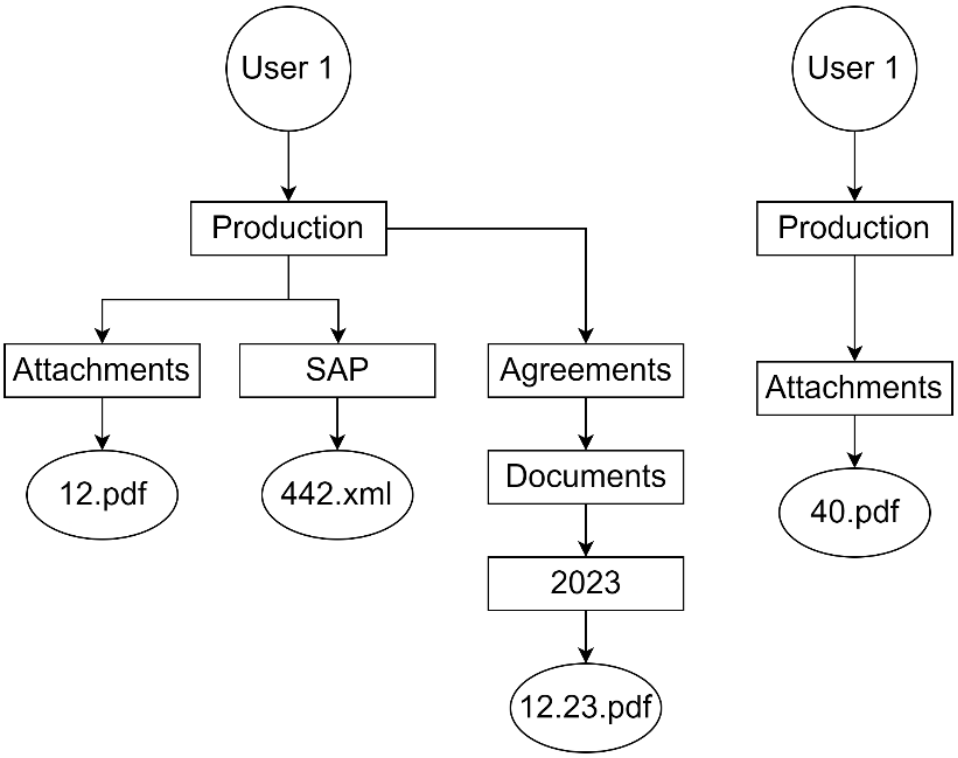


Рисунок 2.14 — Пример деревьев на разных VD (VD3-P2 и VD6-P2) в одном узле хранения

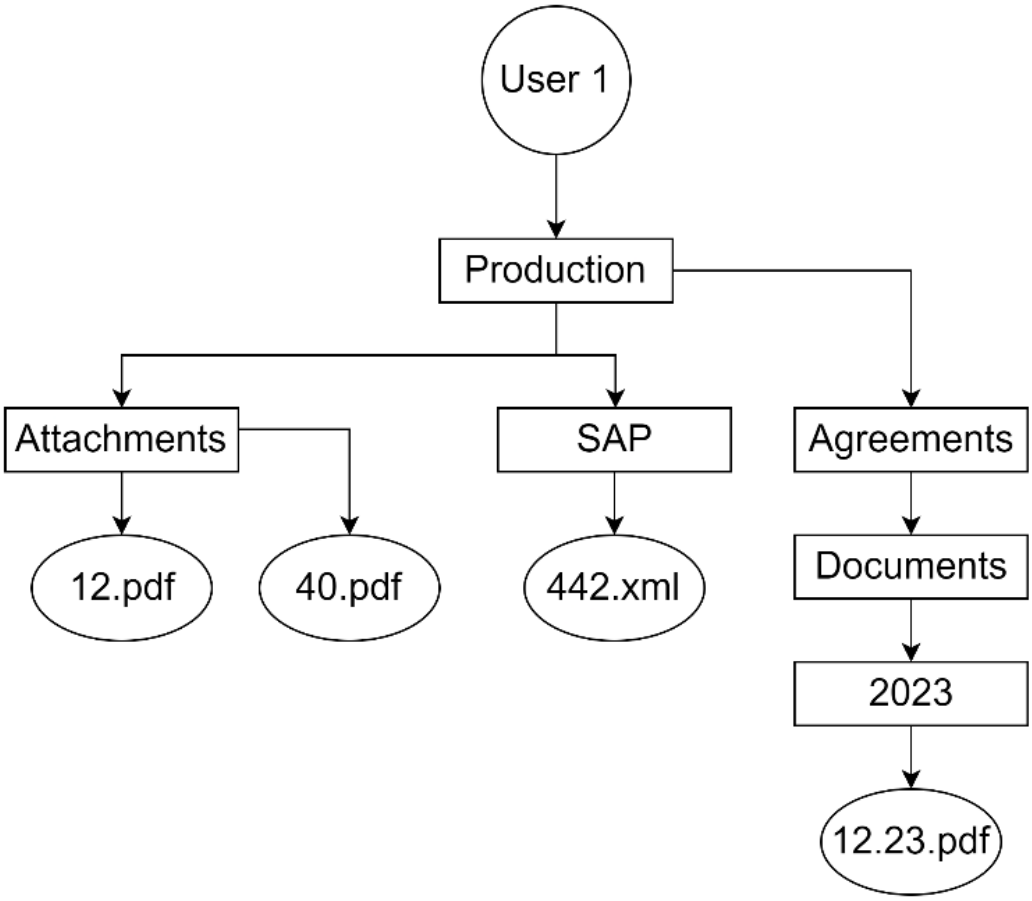


Рисунок 2.15 — Объединенное дерево VD3-P2 и VD6-P2 пользователя 1

Процесс объединения повторяется для каждого узла дерева в списке, и функция соединения выполняется также для всех полученных узлов дерева, пока не останется дочерних узлов. Используемые обозначения показывают, что узлы дерева, имеющие одинаковый атрибут имени, объединяются друг с другом и добавляются в **TreeNode**, а также узлы дерева, имеющие уникальные имена.

2.3.3. Алгоритм сжатия деревьев

После объединения деревьев из выбранных для каждого пользователя VD вызывается алгоритм сжатия. Этот алгоритм необходим для минимизации размера дерева путем сжатия узлов дерева, имеющих одного прямого дочернего элемента, в один, в результате чего в целом получается меньшее дерево. Это означает, что каталоги, содержащие только один объект (файл или каталог), сжимаются в один. Соединение двух узлов дерева в один описывается формулой 2.20:

$$JoinTreeNode(\boldsymbol{TreeNode}) = \begin{Bmatrix} \boldsymbol{TreeNode}_1^{Compress} = \boldsymbol{TreeNode}^{name} \\ +\boldsymbol{TreeNode}^{compress}, \\ \boldsymbol{TreeNode} = \boldsymbol{TreeNode_1} \end{Bmatrix} \quad (2.20)$$

Атрибут «Compressed» на листе дерева представляет путь к сжатым узлам дерева, если они существуют. Атрибуты «Access type» и «Compressed» в случае каталогов используются только тогда, когда политика разрешает пользователю доступ ко всем файлам в папке, поэтому в этом случае каталог является листом. Начиная с корневого узла, алгоритм сжатия работает в модели сверху вниз, как в формуле 2.21.

$$\boldsymbol{ParentTreeNode} = \boldsymbol{TreeRootNode} \quad (2.21)$$

Функция сжатия родительского узла дерева определяется формулой 2.22:

$$Compress(\boldsymbol{ParentTreeNode}) = \forall \boldsymbol{TreeNode} \in \boldsymbol{ParentTreeNode},$$
$$\begin{cases} JoinTreeNode(\boldsymbol{ParentTreeNode}), \\ compress(\boldsymbol{TreeNode}): |\boldsymbol{ParentTreeNode}| = 1 \\ compress(\boldsymbol{TreeNode}): |\boldsymbol{ParentTreeNode}| > 1 \end{cases} \quad (2.22)$$

Алгоритм сжатия описан на рисунке 2.16.

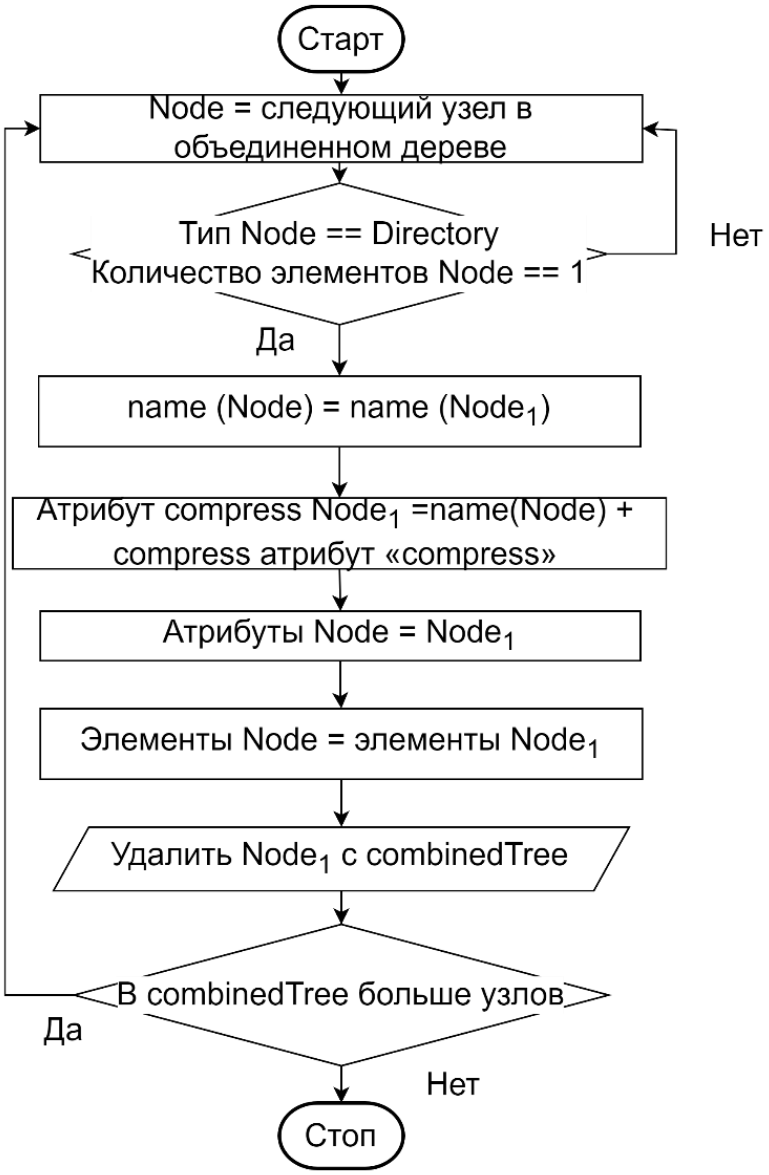


Рисунок 2.16 — Алгоритм сжатия деревьев

Версия сжатого дерева, представленная на рисунке 2.15, показана на рисунке 2.17. Понятно, что атрибут «Compressed» для 12.23.pdf равен Agreements/Documents/2023.

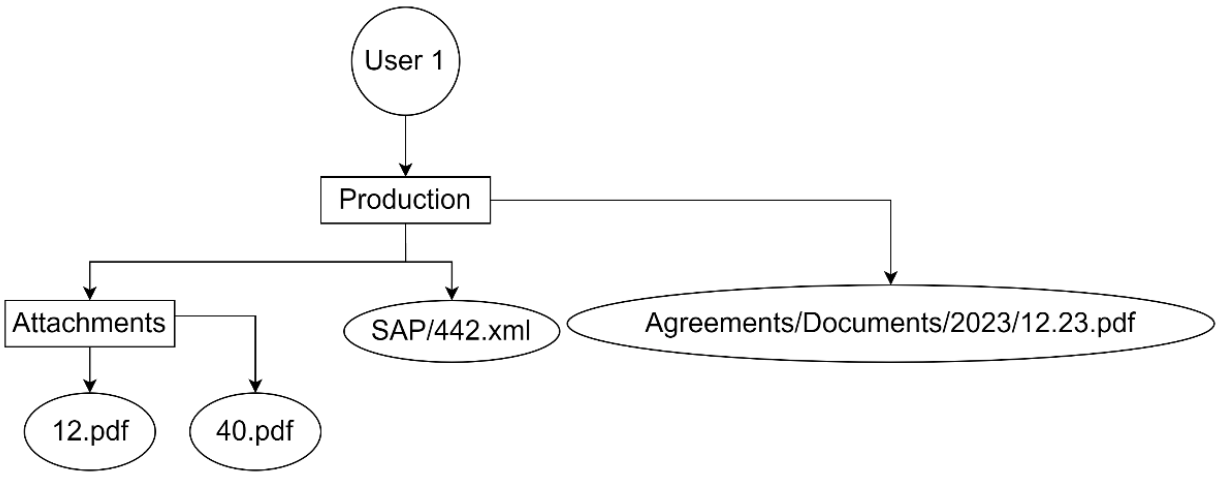


Рисунок 2.17 — Пример сжатого дерева

2.3.4. Построение двоичного дерева Меркла

Следующий шаг — преобразование сжатого дерева в двоичное дерево. Для выполнения этого шага добавляется новый тип промежуточных узлов (объединитель). Объединители используются для объединения дочерних узлов таким образом, что только 2 узла дерева считаются прямыми дочерними узлами. Этот тип узлов имеет атрибут «Paths». Этот атрибут устанавливается, если объединяющий узел имеет каталог в одном из своих прямых потомков, которые не объединяют узлы. На рисунке 2.18 показано, как создаются пути в двоичных деревьях.

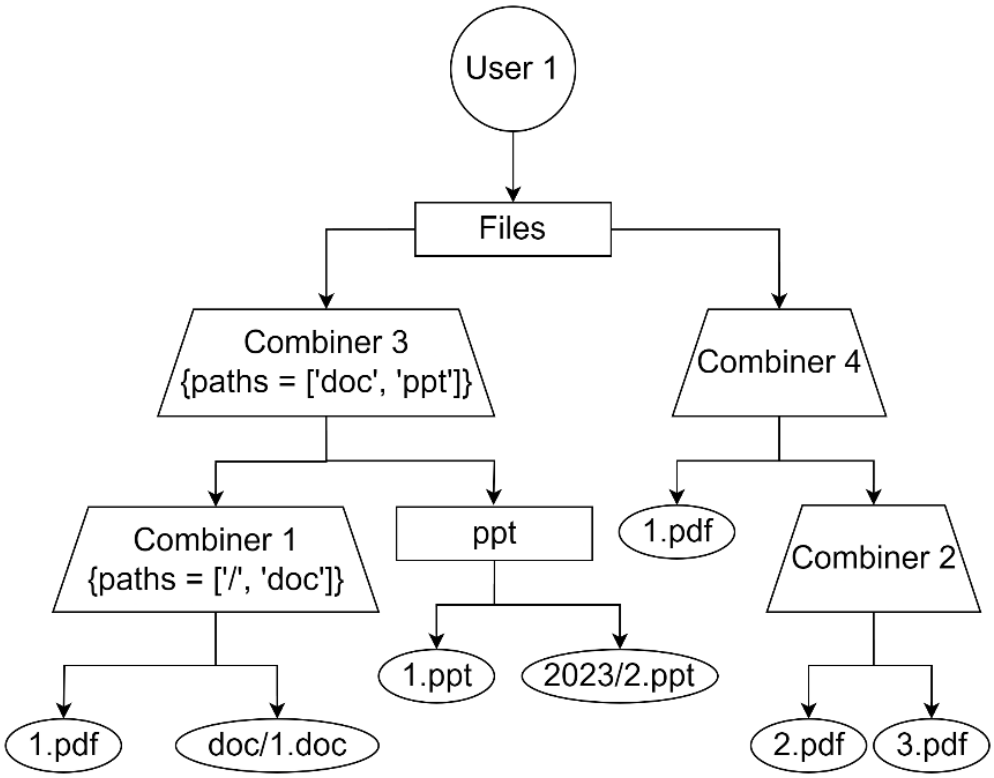


Рисунок 2.18 — Пример использования атрибута Paths

В объединителе 3 значением «Paths» являются doc и ppt. Это связано с тем, что у него есть две прямые папки: doc, который представляет собой «Compressed» значение для узлов doc/1.doc и ppt. Папка 2023 не добавляется в пути, поскольку это дочерний узел ppt, который уже добавлен в пути. При доступе к 2.ppt, учитывая, что полный путь — /files/ppt/2023/2.ppt, Объединитель 3 доступен напрямую, поскольку в путях есть ppt. После этого доступ к 2023/2.pdf осуществляется напрямую.

Преобразование дерева в двоичное дерево важно для уменьшения размера доказательства дерева Меркла. Можно рассматривать пример дерева на рисунке (2. 19), в котором не используется объединение узлов:

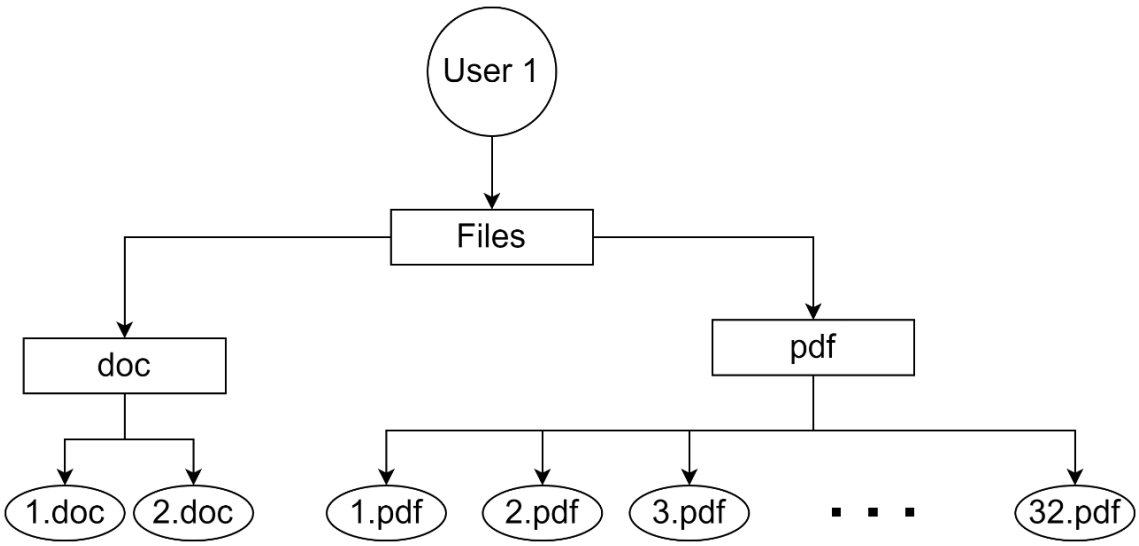


Рисунок 2.19 — Пример дерева, не использующего объединение узлов

Доказательство Меркла для файла 1.pdf требует проверки 33 узлов дерева: 1.pdf + хэши всех файлов pdf + хэш каталога doc. Если преобразовать дерево в бинарное, то потребуется проверка всего 7 узлов: 1.pdf + хэш 2.pdf + хэш 4-х объединителей (объединители 2 + 18+26+30) + хэш каталога doc как на рисунке (2.20). Количество хэшей необходимых узлов дерева в одном каталоге для доказательства Меркла равно $\log_2 number\ of\ tree\ nodes$.

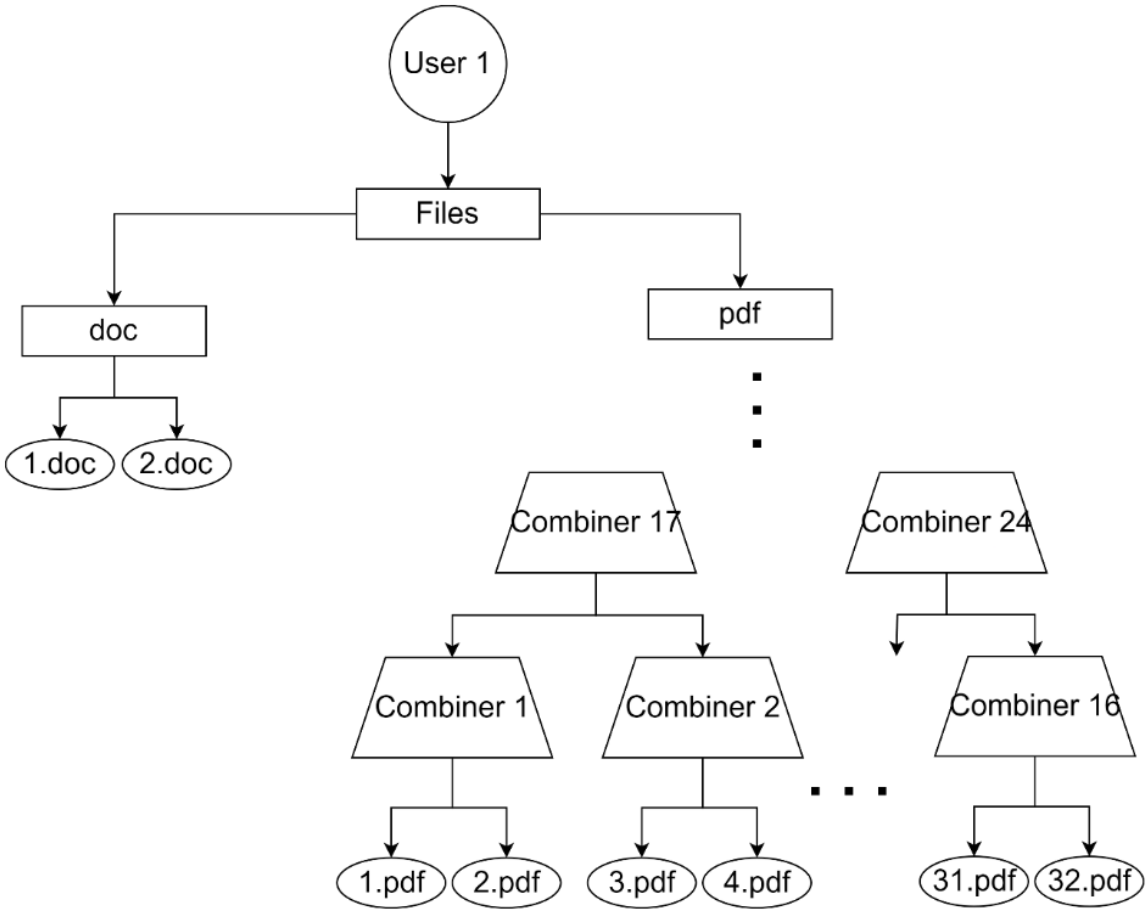


Рисунок 2.20 — Пример дерева, в котором используется объединение узлов

На рисунке 2.21 представлен алгоритм преобразования сжатого дерева в двоичное. Понятно, что каждые 2 дочерних узла любого дерева объединяются в один объединитель, и процесс является рекурсивным до тех пор, пока всего не останется только 2 дочерних узла. Затем алгоритм вызывается для каждого дочернего узла. В этом алгоритме $treeNode_1$ является первым дочерним элементом узла **TreeNode**. Алгоритм сначала вызывается для корневого узла дерева.

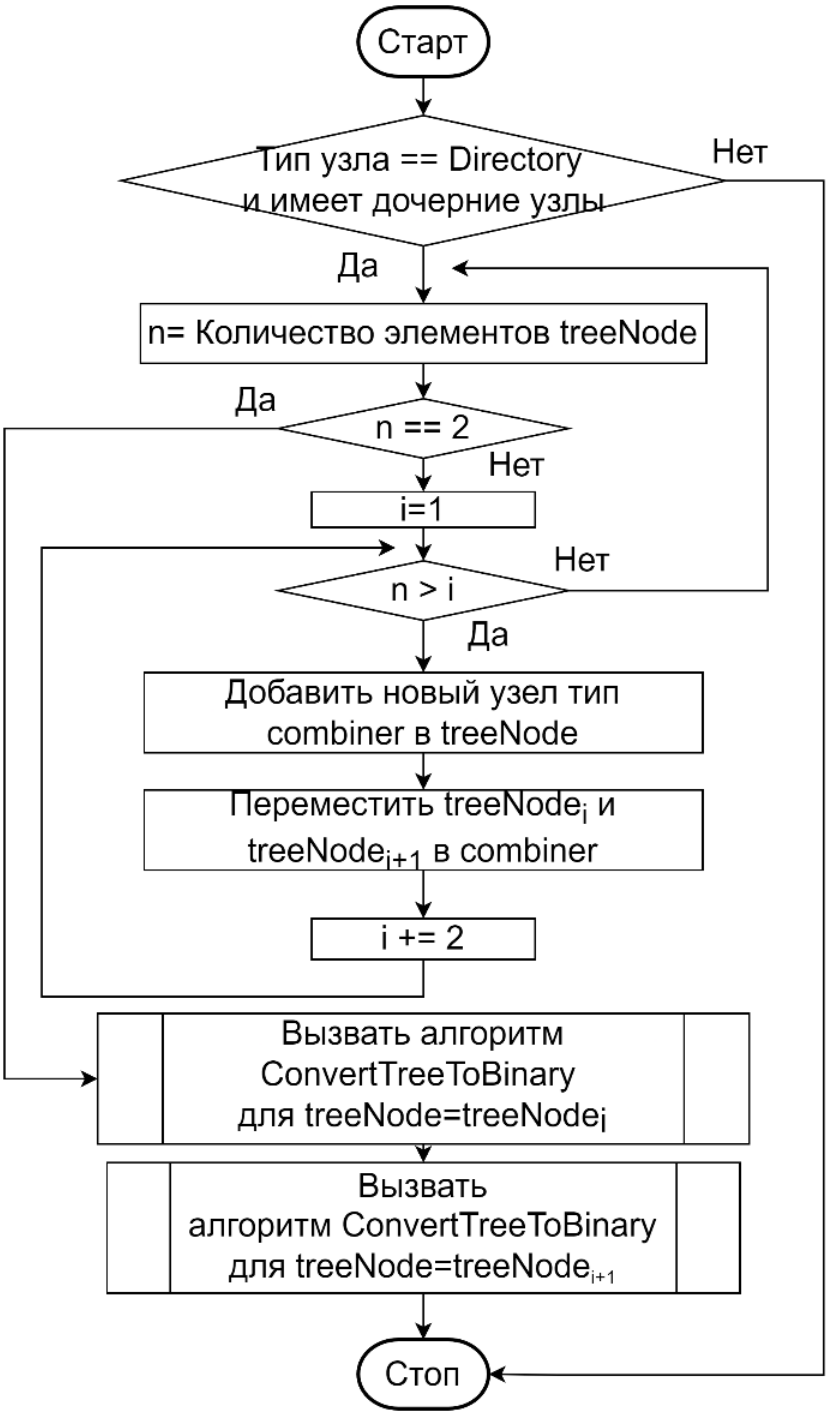


Рисунок 2.21 – Алгоритм преобразования дерева в двоичное

После того как все узлы хранения построят пользовательское дерево, корневые узлы объединяются таким же образом, используя объединение узлов для построения единого дерева Меркла. Хэш полученного корневого узла этого дерева сохраняется для каждого пользователя в блокчейне, как в таблице (2.3).

В случае использования ролей (RBAC) для каждой роли строится дерево. Дерево ролей содержит файлы, к которым политика, связанная с этой ролью, разрешает доступ. В RBAC используются те же узлы дерева и атрибуты, которые предложены для DAC. Роли дублируются и распределяются по узлам хранения, где узел хранения не может хранить роль и ее дублирование. Каждый узел хранения содержит $(|\boldsymbol{Roles}| * 2)/|N|$ ролей, где N — множество узлов хранения.

2.3.5. Алгоритм проверки доступа пользователей

Этот алгоритм используется для проверки того, имеет ли пользователь доступ к файлу или нет. Когда пользователь отправляет запрос на доступ к файлу, будь то запрос на чтение или запись, он отправляет доказательство Меркла вместе с запросом. Тело запроса зависит от метода доступа, используемого для этого файла. Таблица (2.6) показывает заголовок сообщения в случае RBAC и DAC.

Таблица 2.6 — Заголовок запроса доступа к файлу

DAC	RBAC
{ *Access = "DAC",* *MerkleProof = [...],* *fileInfo = {...}* *}*	*{* *Access = "RBAC",* *Role = "Role type",* *MerkleProof = [...],* *fileInfo = {...}* *}*

У каждого пользователя есть копия его личного дерева + копия дерева/деревьев ролей. Первый шаг — найти файл. Этот процесс можно выполнить, используя метод сверху вниз, без поиска всех возможных узлов дерева, как в случае поиска в глубину (DFS) [105], поскольку дерево Меркла построено в

структуре, которая сохраняет файлы организованными в исходной иерархии путей. Доступ к файлам можно получить быстро, выбрав правильные промежуточные узлы. Узлы объединения имеют атрибут «Paths», который содержит имена прямых каталогов, если они существуют. В этом процессе также используется атрибут Compressed, если имеются сжатые узлы.

Как только пользователь выбирает файл из соответствующего дерева, он отправляет информацию о файле вместе с доказательством Меркла в балансировщик нагрузки, представленный смарт-контрактом блокчейна. Смарт-контракт блокчейна сначала проверяет тип доступа и хэширует информацию о файле, а затем вычисляет хэш суммы хэша информации о файле и первого хэша в массиве доказательств Меркла и повторяет процесс для всех хэшей в массиве доказательств Меркла. После этого он сравнивает полученный хэш с сохраненным в таблице (2.3) в случае DAC или с сохраненным хэшем в таблице (2.4) в случае RBAC. Блокчейн может подтвердить личность пользователя на основе авторизации его кошелька.

Пользователь отправляет значение файла без его хэша и массив доказательства Меркла (обязательные узлы дерева) $P = \{P_1, P_2, P_3, .. P_n\}$, где P_1 — необходимое значение информации о файле, а $P_2, P_3, .. P_n$— хэши необходимых узлов дерева для доказательства Меркла.

Предполагая, что hash(x) является хэш-функцией, хэш дерева Меркла из доказательства Меркла можно вычислить с помощью рекурсивной функции $verifyMerkleProof(P, n)$.

$$verifyMerkleProof(P, i) = \begin{cases} 0 & i = 0 \\ hash\big(P_i + verifyMerkleProof(P, i-1)\big) & otherwise \end{cases} \quad (2.23)$$

2.3.6. Алгоритм кеширования локального дерева доступа

В случае DAC, если пользователю предоставляется или отменяется доступ к файлу f, или если изменяется тип доступа (чтение/запись), этот файл добавляется/удаляется/обновляется в дереве доступа на соответствующем виртуальном диске. Изменения распространяются на объединенное дерево и на общий корневой хэш Меркла. Пользователь может обнаружить изменения в своем дереве, сравнив свой общий корневой хэш Меркла с хэшем, хранящимся в блокчейне. Если хэш отличается, то пользователь отправляет запрос всем узлам хранения, содержащий его хэш корня Меркла для узла хранения. Если хэш дерева Меркла отличается от хэша, хранящегося на узле хранения, узел хранения отправляет обратно обновленную версию дерева Меркла узла хранения, и пользователь обновляет свое дерево на основе этого. На рисунке 2.22 показан соответствующий алгоритм.

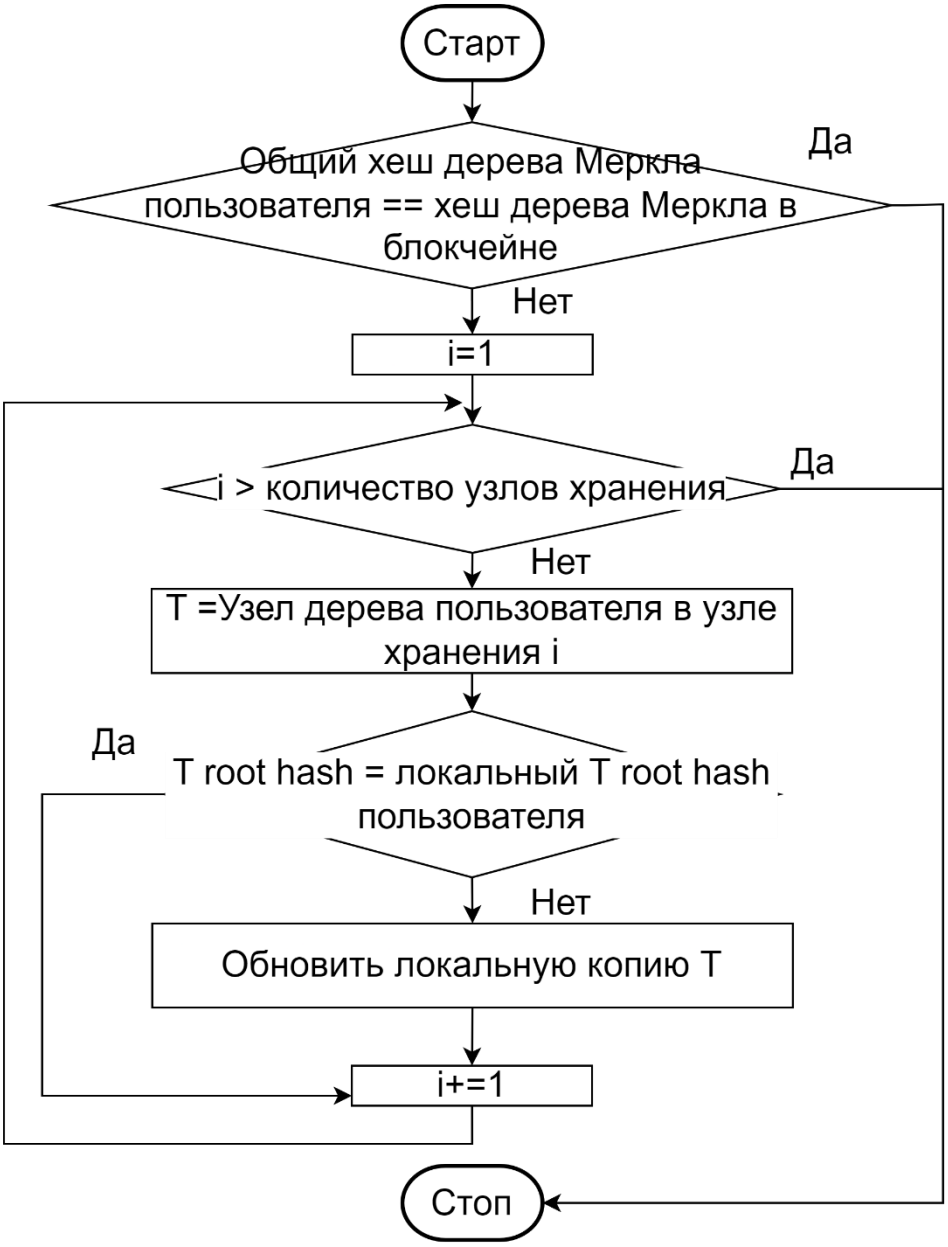


Рисунок 2.22 – Алгоритм кеширования локального дерева доступа

В случае RBAC используется тот же алгоритм. Если роль пользователя изменяется, дерево обновляется, и пользователь обнаруживает это, сравнивая хэш корня дерева Меркла, хранящийся в блокчейне, с его сохраненной версией ролей.

Если пользователю добавляется новая роль, он запрашивает копию дерева Меркла ролей у одного из узлов хранения, на которых оно размещено, и дерево проверяет в блокчейне, есть ли у пользователя эта роль или нет, прежде чем отправить ее обратно.

Если пользователь теряет роль, эта роль удаляется из его записи в блокчейне, и, таким образом, доступ аннулируется.

2.3.7. Алгоритмы, связанные с изменением узлов хранения

При потере узла хранения вызывается алгоритм восстановления узла хранения, описанный выше. Этот алгоритм восстанавливает файлы, используя стирающее кодирование только для восстановления файлов. Как только виртуальный диск восстанавливает все блоки файлов, запрос отправляется на один из узлов хранения, на котором хранятся два других виртуальных диска, принадлежащих тому же кластеру, и получаются соответствующие деревья Меркла для пользователей, которые имеют доступ DAC к файлам в этом виртуальном диске. скопирован на восстановленный VD.

После этого балансировщик нагрузки инициирует восстановление деревьев ролей, которые хранил потерянный узел хранения, выбирая, на какие узлы хранения отправлять деревья, путем расчета количества деревьев ролей в каждом узле хранения $(|\boldsymbol{Roles}| * 2)/|N|$, где $\boldsymbol{Roles}$ — множество всех ролей в системе. Поскольку каждая роль имеет две копии, можно восстановить потерянные деревья ролей на потерянном узле хранения, скопировав их непосредственно с узлов хранения, на которых есть их копии. Целевой узел хранения для каждого дерева ролей не может быть исходным узлом, так как дерево не должно копироваться с узла хранения на тот же узел хранения, чтобы можно было восстановить эту роль в случае потери узла хранения.

Если узел хранения, который был потерян на короткий период времени (например, несколько дней), возобновляет работу, потерянный узел сравнивает хэш каждого имеющегося у него корневого дерева Меркла с созданными копиями. Если хэш корневого дерева Меркла совпадает с копией, записи в

блокчейне обновляются, а лишняя копия удаляется. Если оно не совпадает, дерево копируется из резервной копии, а лишняя копия удаляется.

При добавлении нового узла хранения в систему вызывается алгоритм добавления узла хранения, описанный выше, который перемещает некоторые виртуальные диски в зависимости от конкретных условий на новый узел. Вместе с ними перемещаются деревья соответствующих пользователей. Деревья ролей перемещаются тем же механизмом, что и выбор блокчейна $(|\boldsymbol{Roles}| * 2)/|N|$ уникальные деревья ролей со всех узлов хранения и перемещает их на вновь добавленный узел.

Выводы по главе 2

1. В этой главе представлена новая модель, которая позволяет распределять данные по узлам с минимизацией места хранения. Эта модель является основой архитектуры информационной системы, которая определяет, как различные компоненты взаимодействуют друг с другом. Предлагаемая модель распределения данных предназначена для работы с блокчейн-системами.
2. Представленные алгоритмы балансировки оперируют этой моделью и управляют распределением данных в указанной архитектуре в автономном режиме.
3. Алгоритмы балансировки модели распределения данных выполняются в среде смарт-контрактов, поэтому имеют ряд ограничений.
4. Представленная модель распределения данных уменьшает общий объем требуемого дискового пространства на 25% по сравнению с системами полного резервного копирования за счет использования

метода стирающего кодирования, при этом сохраняя возможность автономного восстановления данных на других узлах.

5. Предложена новая система контроля доступа, которая может работать с блокчейном в сочетании с локальным хранилищем и механизмом кэширования на стороне пользователей. В этой системе часть данных контроля доступа хранится у самих пользователей, которые запрашивают авторизацию, однако модель использует хэши корня дерева Меркла, хранящиеся в блокчейне, для предотвращения манипулирования данными и минимизация размера данных хранящихся в блокчейн. Данная модель контроля доступа состоит из нескольких алгоритмов, которые позволяют управлять контролем доступа в различных сценариях и уменьшает количество требуемых запросов, а значит снижает нагрузку на узлы хранения. Аналогичную концепцию кеширования можно использовать для хранения и управления метаданными.

3. МОДЕЛЬ ОЦЕНКИ НАДЕЖНОСТИ ПРЕДЛАГАЕМОЙ СИСТЕМЫ

3.1. Классические методы расчета надежности систем хранения данных

В общем случае надежность систем рассчитывается с помощью графика надежности [76], на которых отдельные части системы (подсистемы) обозначаются как компоненты. В этих методах репликации выражаются как параллельный компонент, а компоненты, которые не могут быть дублированы, выражаются как последовательные компоненты. В случае параллельных компонентов системы только один из компонентов требуется для выполнения любого процесса в системе. Это значит, что если один из компонентов утерян, система все равно сможет работать. Как показано на рисунке 3.1, для работы системы требуется только компонент 1 или компонент 2. Если компонент 1 утерян, то работу можно выполнять с использованием компонента 2, и наоборот. Этот подход можно использовать для представления резервных копий, так как используется только основной сервер, а копия работает только в случае сбоя сервера. Надежность в случае выражается формулой 3.1:

$$R = 1 - [(1 - R_1)(1 - R_2) \dots (1 - R_n)] \quad (3.1)$$

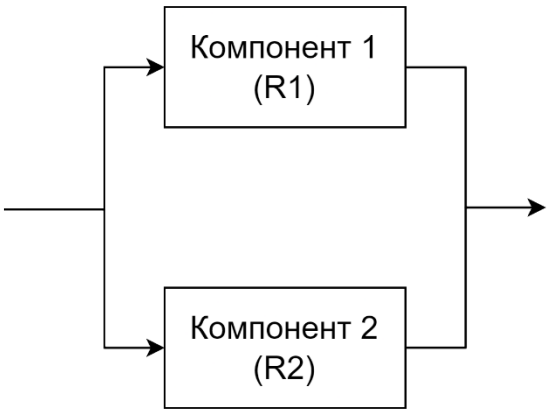


Рисунок 3.1 — Случай «Или» для двух компонентов один из них необходим для выполнения работы

Для сериализованных компонентов системы для функционирования системы необходимы все сериализованные компоненты. Если один из сериализованных компонентов перестает работать, система перестает работать. Как показано на рисунке 3.2, если компонент 1 или компонент 2 утерян, система будет считаться неисправной.

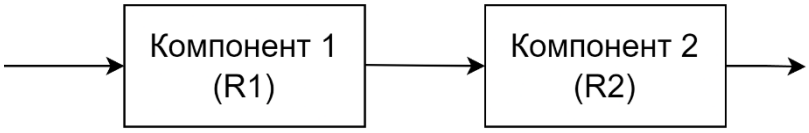


Рисунок 3.2 — Случай «И» для двух компонентов, оба из которых необходимы для выполнения работы

Надежность системы в этом случае выражается надежностью всех отдельных серийных компонентов, как в формуле 3.2:

$$R = R_1 * R_2 * \ldots * R_n \tag{3.2}$$

Система может иметь комбинацию обоих типов компонентов. Если рассмотрим систему, которая имеет один балансировщик нагрузки и один узел

хранения с двумя дубликатами, систему можно представить так, как показано на рисунке 3.3:

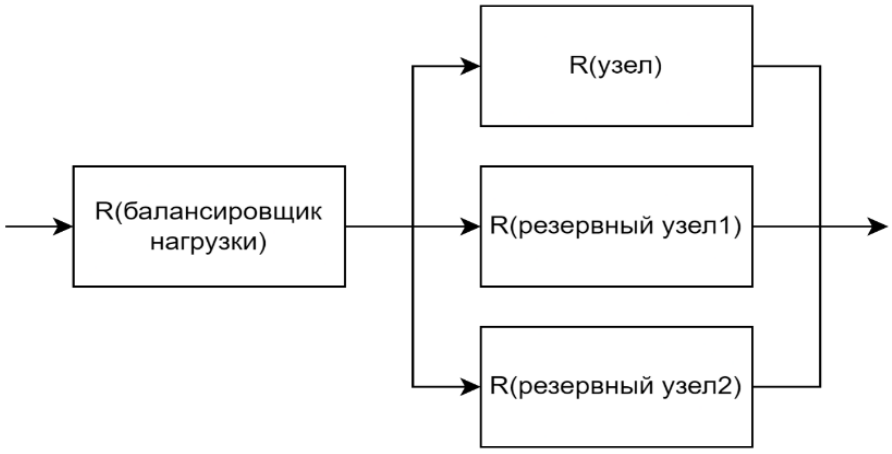


Рисунок 3.3 — Представление системы как с параллельными, так и с последовательными компонентами с использованием графика надежности

После построения графика применяются формулы вычисления надежности. Надежность всех серийных компонентов выражается как умножение надежности этих отдельных компонентов. Надежность системы на рисунке 3.3 выражается как:

$$R = R(\text{балансировщик нагрузки}) * \left[1 - \left(1 - R(\text{узел})\right) * \left(1 - R(\text{резервный узел1})\right) * \left(1 - R(\text{резервный узел2})\right)\right] \quad (3.3)$$

Однако этот метод не может быть применен к предлагаемым моделям, поскольку они имеют возможность восстановления потерянных данных в случае потери узла. В рассмотренной выше модели распределения данных предлагается метод автоматического процесса восстановления. На всех узлах должно быть свободное место для размещения восстановленных файлов с других узлов. В случае потери узла его данные можно восстановить на остальных узлах, если на

этих узлах достаточно свободного места для размещения восстановленных файлов. Этот метод явно зависит от количества узлов N и свободного места, доступного на узлах VD`, что влияет на максимальное количество возможных восстановлений MPRC. Этот метод также учитывает, что узел должен быть восстановлен до следующего возможного сбоя.

Если узел x выходит из строя при восстановлении другого узла y, некоторые данные могут стать невосстановимыми, если и узел x, и узел y имеют виртуальные диски, принадлежащие одному и тому же виртуальному кластеру. Как показано на рисунке 3.4.A, если узел 4 потерян, балансировщик нагрузки определяет, где восстановить потерянные узлы. После этого, как показано на рисунке 3.4.B, балансировщик нагрузки решает, что VD1-P1 (виртуальный диск, содержащий первую часть файлов в первом виртуальном кластере) перестраивается на узле 2, VD2-EC (виртуальный диск стирающего кодирования в виртуальном кластере 2) на узле 6, VD4-P2 на узле 2, VD6-EC на узле 1. После этого данные могут быть доступны и восстановлены в случае повторного сбоя, так как в система из-за отказа еще одного узла. Как и на рисунке 3.4.C, если после этого узел 5 теряется, его виртуальные диски восстанавливаются на остальных доступных узлах с использованием того же механизма (рисунок 3.4.D). Понятно, что, как и в примере, максимальное количество возможных отказов узлов равно двум, поскольку из-за ограничений свободного пространства система может пережить только отказы двух узлов.

Другой метод, используемый RAID, заключается в оценке общего среднего времени до сбоя системы на основе среднего времени восстановления диска / узла и среднего времени до сбоя диска / узла. Формула 3.4 выражает среднее время до отказа:

$$MTTF_{system} = \frac{(MTTF_{Disk})^2}{(D + C * n_G) * (G + C + 1) * MTTR} \quad (3.4)$$

Где:

- $MTTF_{Disk}$ — среднее время отказа дисков

- $MTTR$ — среднее время восстановления
- D — общее количество дисков с данными
- n_G — количество групп дисков
- C — количество проверочных дисков на группу
- G — число. дисков с данными в группе.

Эта модель учитывает количество узлов, но не учитывает свободное пространство, доступное на узлах, поскольку в RAID доступное свободное пространство используется только для основного хранения данных и не используется для целей восстановления. По этой причине максимальное количество возможных восстановлений не может быть рассчитано напрямую с использованием модели надежности RAID.

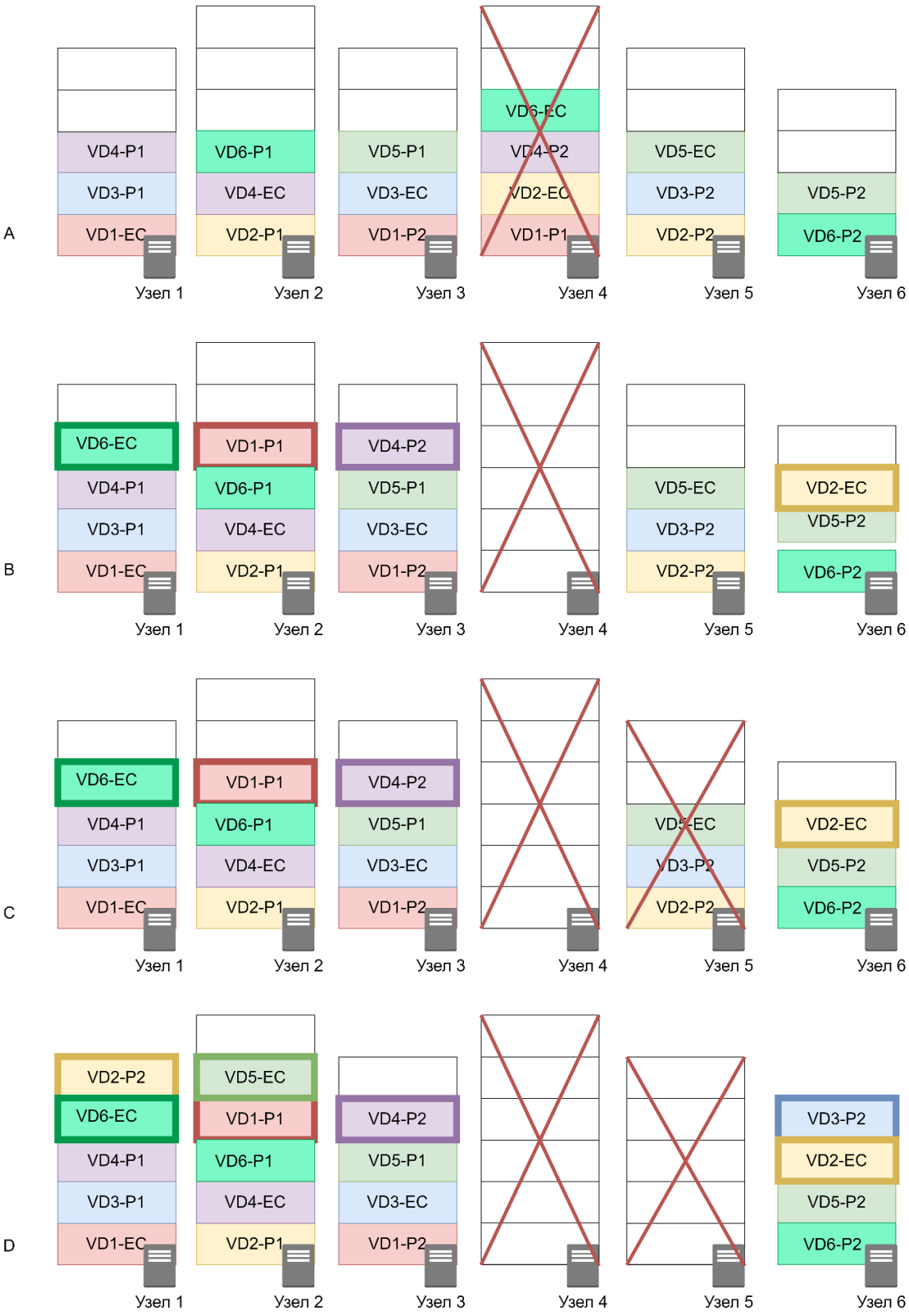


Рисунок 3.4 - Автоматическое восстановление данных в системе

Таким образом, очевидно, что требуется разработать модель надежности предлагаемой системы. Балансировщик нагрузки может размещаться на любом, в

том числе и внешнем узле в блокчейн сети или может быть включен в саму систему хранения (располагаться на тех же серверах, что и виртуальные диски для файлов). Далее рассматривается такая конфигурация системы, где блокчейн и смарт-контракт размещен на каждом сервере, вместе с файлами. В такой конфигурации при выходе из строя одного из узла блокчейна можно подключатся к любой другой, которые содержит всю цепочку данных. Отказ системы произойдет только при выходе из строя всех узлов блокчейн. Следовательно, надежность всей системы зависит от подсистемы хранения файлов. Рассмотрим её подробнее.

3.2. Модель оценки надежности динамического восстановления данных в распределенной сети хранения данных

За основу вычисления надежности была взята известная модель расчета надежности для RAID-5. Для вычисления надежности системы в течении года (8760 часов) нужно вычислить вероятность отказа всей системы. Рассматриваемая надежность системы противоположна вероятности отказа. Итак, это 1-вероятность отказа. Вероятность отказа можно измерить с помощью AFR (Annualized Return Rate) [5, 79].

$$AFR = 1 - e^{\frac{8760}{MTTF_{system}}} \tag{3.5}$$

Поскольку $8760/MTTF_{system} \ll 1$ [5], AFR можно представить как:

$$AFR \approx \frac{8760}{MTTF_{system}} \tag{3.6}$$

Таким образом, R можно рассчитать следующим образом:

$$R = 1 - \frac{8760}{MTTF_{system}} \tag{3.7}$$

Для вычисления надежности нужно вычислить время наработки на отказ всей системы (MTTF – Mean time to fail), обозначим его как $MTTF_{system}$, а $MTTF_{node}$ - время наработки на отказ для конкретного узла. $MTTF_{system}$ и $MTTF_{node}$ измеряются в часах, а R не имеет единицы измерения.

Мы предполагаем, что система выходит из строя, когда часть хранящихся данных безвозвратно потеряна. Т.е. достаточно привести её в состояние, когда будет потерян хотя бы один файл из VD. $MTTF_{system}$ можно рассчитать воспользовавшись формулой из расчета надежности для RAID [6].

$$MTTF_{system} = \frac{MTTF_{node}}{|\boldsymbol{N}|} * \frac{1}{P_{SFANF}} \tag{3.8}$$

где:

- $\boldsymbol{N}$ – множество узлов в системе
- P_{SFANF} (SFANF: System fails after the node has failed) – вероятность, что система откажет после отказа одного узла в системе.

Поскольку DecStore реализует динамическое восстановление на остальных узлах, P_{SFANF} рассчитывается иначе, чем для RAID, так как требуется учитывать возможное время восстановления данных на основе свободного места, оставшегося в системе. Чтобы рассчитать надежность системы, предположим, что произвольный узел вышел из строя и нужно определить вероятность отказа всей системы после потери этого узла. Например, в системе с полным резервным копированием, в которой есть один сервер резервного копирования для каждого узла, система может выйти из строя при условии, что до восстановления первого потерянного узла на новом сервере произойдет потеря ещё одного узла.

Рассмотрим подробнее от чего зависит P_{SFANF}. Например, вероятность потери других узлов в процессе восстановления P_{NFBR} (Node Failure Before the Repairment is finished).

$$P_{NFBR(i)} = \frac{MTTR_i}{MTTF_{node(i)}/|\boldsymbol{N}`|} \tag{3.9}$$

где $\boldsymbol{N}`$ - множество оставшихся узлов, которые участвуют в восстановлении потерянного узла.

Приведем пример для систем, где осуществляется полное резервное копирование, где для каждого узла существует два сервера резервного копирования. В это случае, $\boldsymbol{N}` = 2$ для $i=1$ (если один узел выходит из строя). Если 2 узла выходят из строя ($i=2$), то можно восстановиться с 1 узла, т.е. для $i=2$, $\boldsymbol{N}` = 1$. Необходимо заметить, что в системах полного резервного копирования не важно общее количество узлов. Имеет значение только количество резервных узлов для каждого узла.

Поскольку разные узлы могут иметь разные характеристики, расчеты $P_{NFBR(i)}, MTTR_i, MTTF_{node(i)}$ различаются в зависимости от узла i. Будем считать, что все узлы имеют одинаковые характеристики для простоты вычислений, поэтому вместо этого будем использовать $P_{NFBR}, MTTR, MTTF_{node}$

Практически, $MTTR \ll \frac{MTTF_{node}}{|N`|}$, так что время необходимое для ремонта/замены диска, намного меньше, чем срок службы диска.

Для $MTTR_{node}$ время восстановления узла в часах (mean time to repair a node) системы полного бекапирования вычисляется следующим образом:

$$MTTR_{node} = \frac{S_{node}}{Th * 3600 * R_{rate}} + Tr \tag{3.10}$$

где:

- S_{node} – размер хранилища определённого узла

- Th – пропускная способность узла в Мб/сек. Предположим, что $MTTR_{node}$ одинаково для всех узлов. Так как $MTTF_{node}$ измеряется в часах, то Th необходимо умножить на 3600.
- R_{rate} (rebuild rate) – показывает сколько нужно выделить системных ресурсов для восстановления утраченного диска. Среднее значение 50%
- Tr – время необходимое для замены потерянного узла в часах. Оно равно 0, если используется запасной узел, который уже подключен к сети и сконфигурирован, как основной для любого узла (скопирован из резервного узла в случае систем полного резервирования). Так называемый «горячий» узел.
- т.к. система состоит из множества узлов можно предположить, что общий размер дисков всех узлов одинаковый (S) и можно вычислить обобщенный *MTTR* для всех узлов как:

$$MTTR_{node} = \frac{S}{Th * 3600 * R_{rate}} + Tr \quad (3.11)$$

В предложенной модели, когда узел выходит из строя, её данные перестраиваются на оставшихся узлах.

Время восстановления в предложенной системе:

$$MTTR_{node} = \frac{S}{|\boldsymbol{N}`| * Th * 3600 * R_{rate}} + Td \quad (3.12)$$

где *Td* – время распределения потерянных *VD* по оставшимся узлам в часах. В более общем случае это может быть время восстановления блока данных (dataUnit), файла, части файла фиксированного размер, или как в нашем случае, виртуального диска.

При этом сложность алгоритма распределения $O(|VD_{lostNode}|)$(количество vds в потерянном узле) ,чем меньше количество блоков данных, тем быстрее он работает.

$$Td = \sum_{dataUnit=1}^{|dataUnits|} Td_{dataUnit} \quad (3.13)$$

Где $dataUnits$ – Множество всех блоков данных.

Нужно отметить, что чем меньше размер блока данных мы используем, (например, используем файл вместо виртуального диска), тем больше значение $|dataUnits|$. Следовательно, по формуле (3.13) увеличивается Td, а значит увеличивается $MTTR$ и уменьшается $MTTF$.

Однако, группировка данных в единый большой виртуальный диск не приводит также к увеличению надежности, т.к. достаточно затратно иметь другой такой же узел, который будет иметь необходимое свободное место, что делает применение такого подхода бессмысленным по сравнению с систем полного резервного копирования. Если узел выйдет из строя и этот большой диск необходимо восстановить, потребуется найти узел, способный разместить такой диск, что менее вероятно по сравнению с восстановлением виртуального диска меньшего размера, что означает меньшую надежность.

При использовании виртуальных дисков, а не отдельных файлов, алгоритм восстановления работает намного быстрее, т.к. ему надо вычислить оставшиеся части только для потерянных VD, а не каждого из тысячи или миллиона файлов, которые были на вышедшем из строя узел. Размер VD нужно выбрать так, чтобы он был достаточным для уменьшения параметра $|dataUnits|$ и достаточно малого для удобства размещения на оставшихся узлах.

Для расчета $MTTF_{node}$ предположим, что у каждого узла всего один жесткий диск, тогда мы в последствии сможем более корректно сравнить предложенный подход с другими известными системами хранения. В этом случае, $MTTF_{node}$ определяется производителем конкретного диска. Например, для дисков, которые используются для файловых хранилищ корпоративного уровня $MTTF$ обычно равен 2000000 часов [108].

Предложенная система имеет динамическую модель восстановления, где при выходе узла из строя, все данные на нем могут быть восстановлены на оставшихся узлах *MPRC* раз. *MPRC* (Maximum Possible Recoveries Count) определяется как счетчик максимально возможного количества восстановлений после потери. Процесс восстановления возможен, когда два из следующих условий выполняется:

$$\left| \bigcup_{i=1}^{|N|} \left(\boldsymbol{VD`_{n(i)}} \right) \setminus \{ \boldsymbol{VD`_{n(j)}} \} \right| \geq \left| \boldsymbol{VD_{n(j)}} \right| \; \forall \; 1 \leq j \leq |\boldsymbol{N}| \tag{3.14}$$

$$|\boldsymbol{N}| - 1 \geq \; 3 \tag{3.15}$$

Следовательно, возможно восстановить данные, пока будет достаточно свободного места во всей системе для восстановления любого вышедшего из строя узла и пока будет не меньше 3-х узла в системе.

Важно отметить, что условие (10) включает только свободное место, которое может содержать новые виртуальные диски. Не все свободное место на узлах можно использовать. Представим, что требуется восстановить потерянный *VD* размером 1 Гб. При этом в системе осталось 4 узла на каждом из которых по 512 Мб каждом. Системе не сможет восстановить *VD*, несмотря на то, что сумма свободного места на оставшихся узлах равна 2 Гб. Предполагая, что $\boldsymbol{VD`_S}$ — это набор свободных слотов VD в системе, условие можно переписать следующим образом:

$$\left| \boldsymbol{VD`_S} \setminus \boldsymbol{VD`_{n(j)}} \right| \; \geq \left| \boldsymbol{VD_{n(j)}} \right| \forall \; j \; \in \; \boldsymbol{N} \tag{3.16}$$

Следовательно, $MPRC$ может быть определено следующим образом:

$$MPRC = max\{n = 1..|\boldsymbol{N}| - 2 : F(n) \geq 0\} + 1 \tag{3.17}$$

$$F(n) = F(n-1) - \frac{|\boldsymbol{VD_S}|}{|\boldsymbol{N}| - n + 1} - \frac{F(n-1)}{|\boldsymbol{N}| - n + 1} \tag{3.18}$$

$$F(0) = |\boldsymbol{VD`_S}| \tag{3.19}$$

где

- *F(n)* рекурсивная функция, которая вычисляет свободное место в системе на основе количества вышедших из строя *n* узлов.

MPRC равно максимальному количеству процессов восстановления + 1, потому что даже если больше нет свободного места для повторного восстановления данных, будет возможно восстановить данные на стороне пользователя (данные все еще могут быть доступны).

В системе с полным резервным копированием, если существует *B* резервных серверов для каждого узла произойдет отказ всей системы, если все рабочие и все резервные сервера выйдут из строя до копирования данных на новые сервера. Эту модель можно определить как:

$$P_{SFANF} = \prod_{i=1}^{B} \frac{MTTR_i}{\frac{MTTF_{node}}{B+1-i}} = B! * \prod_{i=1}^{B} \frac{MTTR_i}{MTTF_{node}} \tag{3.20}$$

Тогда $MTTF_{system}$ вычисляется в системах полного резервного копирования как:

$$MTTF_{system} = \frac{MTTF_{node} * B!}{|\boldsymbol{N}|} * \prod_{i=1}^{B} \frac{MTTF_{node}}{MTTR_i} \tag{3.21}$$

В DecStore, вся система выйдет из строя в одном из следующих случаев:

- Если 2 узла выйдут из строя до того времени пока не кончится время необходимое для восстановления одной из вышедших из строя узлов на оставшиеся в системе узлы;
- Если 3 узла выйдут из строя до того пока не пройдет необходимое время для восстановления 2-ух из них на оставшиеся в системе узлы;
- Произойдет выход из строя такого количества узлов, которое равно *MPRC*.

Важно отметить, что вероятность отказа системы в случае *MPRC* = 3 вычисляется следующим образом:

$$P_{SFANF} = \frac{MTTR_1}{MTTF/(|\boldsymbol{N}|-1)} + \left[\frac{MTTR_1 + MTTR_2}{MTTF/(|\boldsymbol{N}|-1)} * \frac{MTTR_1 + MTTR_2}{MTTF/(|\boldsymbol{N}|-2)} - \frac{MTTR_1}{MTTF/(|\boldsymbol{N}|-1)}\right] + \left[\frac{MTTR_1 + MTTR_2+MTTR_3}{MTTF/(|\boldsymbol{N}|-1)} * \frac{MTTR_1 + MTTR_2+MTTR_3}{MTTF/(|\boldsymbol{N}|-2)} * \frac{MTTR_1 + MTTR_2+MTTR_3}{MTTF/(|\boldsymbol{N}|-3)} - \frac{MTTR_1 + MTTR_2}{MTTF/(|\boldsymbol{N}|-1)} * \frac{MTTR_1 + MTTR_2}{MTTF/(|\boldsymbol{N}|-2)}\right] \quad (3.22)$$

Или

$$P_{SFANF} = \left[\frac{MTTR_1 + MTTR_2+MTTR_3}{\frac{MTTF_{node}}{(|\boldsymbol{N}|-1)}} * \frac{MTTR_1 + MTTR_2+MTTR_3}{\frac{MTTF_{node}}{(|\boldsymbol{N}|-2)}} * \frac{MTTR_1 + MTTR_2+MTTR_3}{\frac{MTTF_{node}}{(|\boldsymbol{N}|-3)}}\right] \quad (3.23)$$

Следующая формула представляет обобщенный вид вычисления вероятности отказа всей системы:

$$P_{SFANF} = \left(\frac{MTTR * MPRC}{MTTF_{node}}\right)^{MPRC} * \prod_{i=1}^{MPRC} (|\boldsymbol{N}| - i) \quad (3.24)$$

Следовательно, $MTTF_{system}$ вычисляется в предлагаемой системе как:

$$MTTF_{system} = \left(\frac{MTTF_{node}}{MTTR * MPRC}\right)^{MPRC} * \frac{MTTF_{node}}{\prod_{i=0}^{MPRC}(|\boldsymbol{N}| - i)} \quad (3.25)$$

Предполагая, что все узлы имеют одинаковое количество виртуальных дисков и свободных слотов для виртуальных дисков, надежность предлагаемой системы может быть выше надежности серверов полного резервного копирования при двух условиях:

$$R_{DecStore} > R_{fullBackup} \leftrightarrow \begin{cases} |\boldsymbol{N}| > 3 \\ \forall i \in \boldsymbol{N}: \boldsymbol{VD`_{n(i)}} > \dfrac{2 * \boldsymbol{VD_{n(i)}}}{|\boldsymbol{N}| - 2} \end{cases} \quad (3.26)$$

Выводы по главе 3

1. Для оценки надежности рассмотренной в главе 2 модели распределения необходимо учитывать время восстановления узлов, которое зависит от нескольких параметров, включая количество узлов, свободную память на оставшихся узлах и время, необходимое для восстановления информации между узлами. Важно уметь рассчитать надежность этой модели, чтобы оценить, насколько система надежна по сравнению с другими системами.

3. Текущие модели расчета надежности не могут быть использованы для этой задачи, поскольку они не учитывают возможность динамического

восстановления потерянных данных в оставшихся узлах. В этом процессе используется важный показатель (MPRC), который определяет максимальное количество возможных потерь узлов (допустимое количество восстановлений), до тех пор, пока невозможно будет восстановить потерянные данные.

5. Разработана математическая модель надежности на базе известного подхода оценки надежности для RAID, учитывающая рассмотренные выше параметры, зависящие от вероятности отказов узлов системы.

4. СРАВНИТЕЛЬНАЯ ОЦЕНКА НЕОБХОДИМЫХ РЕСУРСОВ, ПРОИЗВОДИТЕЛЬНОСТИ И НАДЕЖНОСТИ ПРЕДЛАГАЕМОЙ МОДЕЛИ РАСПРЕДЕЛЕНИЯ ДАННЫХ И КОНТРОЛЯ ДОСТУПА

4.1. Выбор инструментария и оценка сокращения места хранения

Производительность

Как обсуждалось в первой главе, существует несколько программных платформ для реализации частного блокчейна. На начальном этапе для проверки эффективности предложенных моделей нужно провести ряд экспериментов, чтобы выбрать платформу. По этой причине в локальной сети был развернут тестовый стенд, включающий сервер-балансировщик, кластер из трех виртуальных файловых машин и сеть 100 Мбит/с.

Цель эксперимента — выбрать подходящую блокчейн платформу, которая работает в соответствии с предложенными требованиями, и проанализировать производительность системы. Для этого была разработана базовая система, которая обрабатывает загрузку и скачивание файлов. При загрузке файла файл разбивается на две половины и рассчитывается стирающее кодирование, а также вычисляется его хэш, а затем отправляется запрос в сеть блокчейна. После этого каждая половина (и стирающее кодирование) загружается на сервер. Этот эксперимент был повторен на трех блокчейн-сетях: Hyperledger, Ethereum Enterprise (Hyperledger Besu) и Ethereum Ropsten (публичная сеть Ethereum).

Hyperledger и Ethereum Enterprise являются более подходящими блокчейн-решениями для реализации нашего подхода по следующим причинам:

- данные платформы предназначены для создания корпоративных решений по модели b2c;
- нет необходимости использовать криптовалюту;
- наличие смарт-контрактов и языка их программирования — Solidity (Ethereum Enterprise) или GoLang (Hyperledger);

- поддержка конфиденциальных транзакций (возможность настройки доступа к определенным транзакциям для определенных пользователей).

Конкретная архитектура блокчейна размещалась в облачном сервисе Яндекса [119]. Для Hyperledger использовался виртуальный сервер (2 ГБ ОЗУ, 2 виртуальных ЦП Intel Xeon Gold 6230, жесткий диск 13 ГБ) с пропускной способностью сети 30 Мбит/с.

Для Ethereum Enterprise использовался клиент Hyperledger Besu на виртуальном сервере (6 ГБ ОЗУ, 2 виртуальных ЦП Intel Xeon Gold 6230, 20 ГБ), пропускная способность сети 30 Мбит/с.

В этой инфраструктуре измерялась общая скорость следующих тестов:

- Загрузить — загрузка из балансировщика 1000 новых документов, включая XML с метаданными и файл размером 10 МБ с произвольным содержимым;
- Загрузить — загрузка 1000 документов (XML + файлы) через балансировщик.

Скорость тестировалась на балансировщике нагрузки, чтобы исключить из измерений задержки клиентского трафика.

Как видно из результатов (рисунок 4.1 и таблица 4.1), обе реализации вполне сравнимы по времени с известными p2p-решениями, работающими по торрент-протоколам. Это подтверждено эксплуатационными тестами [48, 89].

Кроме того, было проведено тестирование с использованием смарт-контракта, размещенного в публичной тестовой сети Ethereum Ropsten.

В случае с Ropsten, хотя загрузка файлов происходит довольно быстро, видно, что загрузка файлов происходит в несколько раз медленнее. Это связано с загруженностью тестовой сети и использованием в ней алгоритма консенсуса PoW, в отличие от Hyperleger Fabric [15] и Hyperledger Besu [10], которые по умолчанию используют менее ресурсоёмкие алгоритмы (PBFT или PoA). Основная проблема общедоступных решений DLT заключается в том, что среднее время обработки транзакций слишком велико [35, 124]. Полученные результаты

показывают, что публичные сети блокчейнов не подходят для создания такой системы.

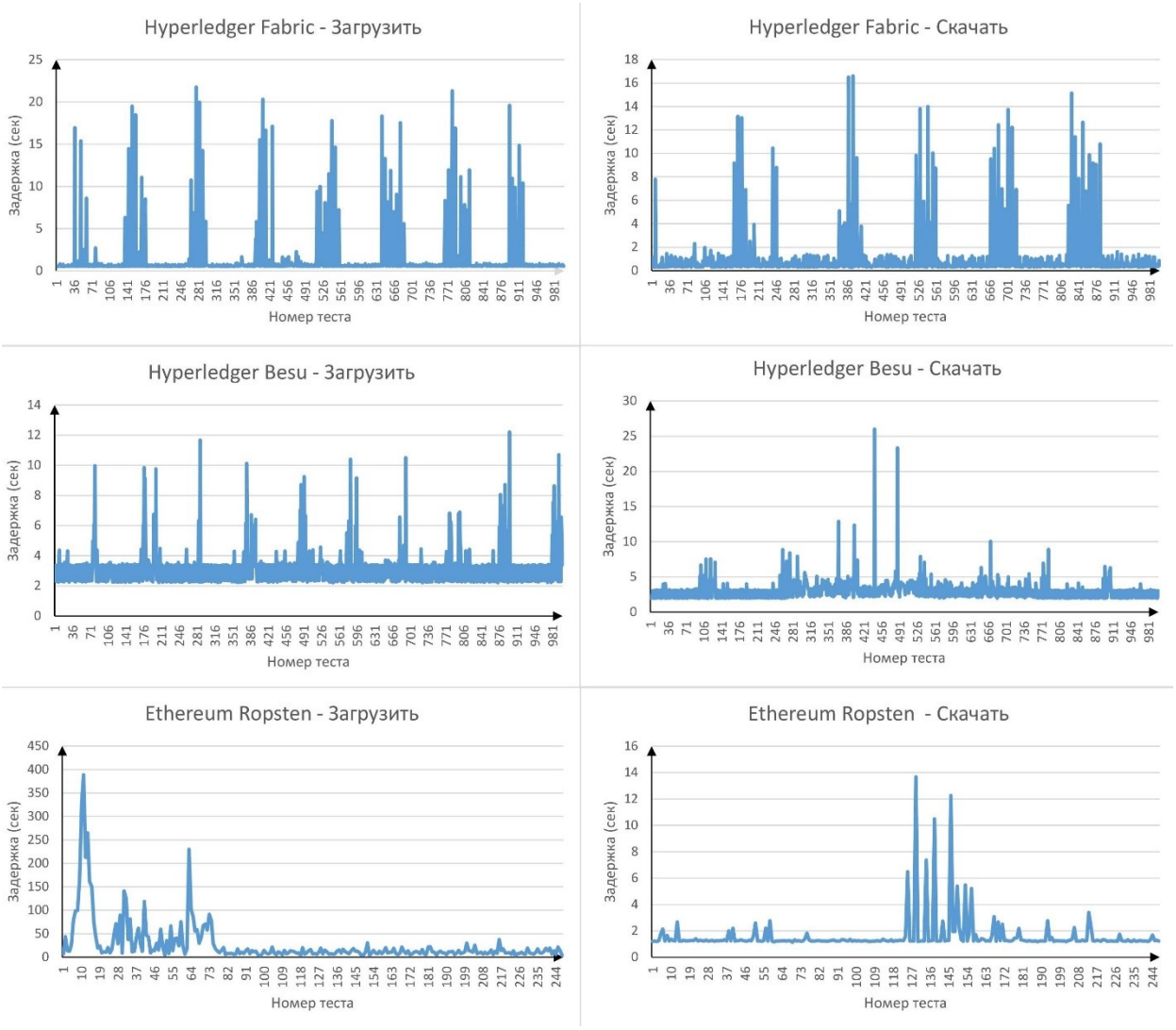


Рисунок 4.1 — Задержка загрузки/скачивания файла в Hyperleger, Ethereum Enterprise(Hyperledger Besu) и Ethereum

Таблица 4.1 — Hyperledger/Hyperledger Besu/Ethereum задержка передачи информации о файле (сек)

		Min	Max	Avg	Median
Hyperledger	Загрузить	0.535	21.762	1.242023	0.585
	Скачать	0.27	16.615	0.927483	0.432

Hyperledger Besu	Загрузить	2.21	12.214	3.203192	3.294
	Скачать	1.937	26.012	2.986525	2.902
Ropsten	Загрузить	3.512	388.917	27.27899	11.935
	Скачать	1.142	13.673	1.597874	1.258

Приведенный эксперимент показал, что такую систему можно быстро масштабировать по горизонтали с использованием недорогих серверов. При этом для хранения файлов 1000 электронных документов достаточно общего объема не более 15 ГБ (файл 10МБ + метаданные xml в DLT). Таким образом, система может хранить до 1’000’000 электронных документов с использованием относительно дешевых узлов общей емкостью 15 ТБ. Для этого можно организовать файловое хранилище на SAS-дисках с IOPS 10 000 об/мин.

Для разработки системы хранения была выбрана Hyperledger Fabric, чтобы провести следующие тесты, так как эта сеть подходит больше всего, исходя из проведенных тестов скорости, а также характеристик, рассмотренных выше. Hyperledger Fabric — это блокчейн-сеть с ограниченным доступом (транзакции не обязательно должны быть публично опубликованы), а ее архитектура поддерживает наличие нескольких организаций. Еще одним преимуществом использования Hyperledger Fabric является то, что его механизм консенсуса намного быстрее и менее ресурсоемок, чем другие общие алгоритмы консенсуса, такие как PoW, используемые в других популярных сетях блокчейнов, таких как Ethereum [26]. Консенсус в Hyperledger Fabric предполагает принятие транзакции на основе количества подписей, полученных среди всех возможных подписей, и добавление принятых транзакций в специальный блок, после чего этот блок проверяется [91].

Все алгоритмы были реализованы в виде чейнкода в Hyperlager Fabric с использованием языка Go. Помимо запуска цепного кода, узлы блокчейна также запускают собственный API, позволяющий взаимодействовать с пользователями и узлами хранения. Этот API был написан на JS (Node.js [84]). Виртуальные диски были реализованы как тома Docker [31], где виртуальные диски, предназначенные

для конкретной компании, доступны с помощью изолированного контейнера. Узлы хранения также используют собственный API, чтобы иметь возможность взаимодействовать с узлами блокчейна и пользователями. Этот API был реализован с использованием PHP (Laravel [68]). Пользовательский интерфейс был реализован с использованием JS (Electron [14]).

Реализация прототипа DApps системы для проверки модели включала в себя следующие элементы:

- Модель распределения виртуальных дисков (включающий реализацию рассмотренных выше алгоритмов распределения данных), который позволяет сократить используемое пространство в 0,25 раза, имеет более высокую надежность, чем другие системы (при определенных условиях) и использует блокчейн для дополнительного доверия к всю систему в целом, а также предоставление возможности принимать централизованные решения в децентрализованной инфраструктуре.
- Чейнкод Hyperledger (который запускает предложенные алгоритмы с использованием языка Go) + API для взаимодействия с чейнкодом.
- Веб-сервис, работающий на узлах, для управления файлами на узле + API для взаимодействия с другими узлами.
- Прототип пользовательского интерфейса.

Для оценки производительности системы были проведены тесты на трех виртуальных машинах, представляющих собой узлы хранения, а также еще на одной виртуальной машине, представляющей собой балансировщик нагрузки. Каждая виртуальная машина имеет 3 ГБ ОЗУ, 1 ядро ЦП, жесткий диск емкостью 20 ГБ и работает под управлением Lubuntu [72]. Каждый тест повторялся 100 раз и отмечалось среднее значение времени, необходимого для выполнения требуемого задания. Эти тесты включали загрузку файла, отключение одного из узлов для запуска функции восстановления и измерение времени, необходимого для восстановления узла на остальных узлах, а также повторное включение этого узла для запуска функции возобновления. Важно отметить, что время, необходимое для обнаружения потери узла, не учитывается, поскольку оно

зависит от конфигурации системы. Время измерялось начиная с вызова алгоритмов восстановления или возобновления работы. Результаты показаны в Таблице 4.2.

Для проведения эксперимента был сформулирован результат базового сценария тестирования информационной системы, состоящей из 3 узлов хранения, которая будет масштабироваться. Это сценарий использования реальной системы ООО «РИИТ» в крупной финансовой организации. Целевые показатели для тестирования были получены из этой системы.

Первый тест заключался в измерении времени, необходимого для добавления узла, поэтому всего было использовано 4 узла хранения. Добавление узла не заняло много времени, так как в системе был один VD, и никаких операций по ребалансировке не потребовалось. Добавление файла размером 10 МБ занимает немного больше времени, так как помимо обновления записей в блокчейне требуется разбить файл и загрузить его блоки. Восстановление узла заняло около одной минуты (для 100 файлов, то есть 1 ГБ), и это связано с тем, что к блокчейну было сделано меньше запросов по сравнению с добавлением файлов по отдельности. Время возобновления работы узла составило всего 1062 мс, поскольку после отключения узла файлы не были изменены, поэтому обновлялись только записи в блокчейне, а восстановленный виртуальный диск был удален.

Таблица 4.2 — Среднее время тестирования DecStore

Название теста	**Требуемое время (мс)**	**Целевые показатели (мс)**
Добавление узла	974	5000
Добавление одного файла	1512	3000
Восстановление данных с потерянных узлов	65 041	600 000
Возобновление узла	1062	60 000

Целевые показатели получены из эксплуатируемой заказчиком системы с централизованной архитектурой, которую разработало ООО «РИИТ». В системе работает одновременно до 1500 пользователей и она состоит из 10 сетевых узлов. задержка операций в тесте этими тестами может быть незаметна для конечных пользователей при условии наличия быстрых дисков и достаточной пропускной способности в инфраструктуре, на которой развернут DecStore. На основе проведенных тестов видно, что все основные операции выполняются относительно быстро по сравнению с ожидаемыми. Важно отметить, что для теста использовались виртуальные узлы, поэтому время, необходимое для передачи сетевого трафика, не учитывается. Необходим отметить, что минимальные технические характеристики узлов достаточно низкие (с использованием virtual box, 2 ГБ ОЗУ, 1 процессор, ОС LUbuntu), а это значит, что систему можно построить с минимальными затратами на инфраструктуру.

Процент сокращения места для распределения данных

На основе эксперимента удалось сформулировать правило, определяющее процент сокращения пространства при внедрении систему по сравнению с системами полного резервного копирования который состоит из B резервных копий:

$$Storage\ reduction\ percentage = 100 * \left(1 - \frac{\frac{3}{2}}{1 + B}\right)\% \qquad (4.1)$$

По сравнению с одним сервером резервного копирования требуемое пространство будет уменьшено на:

$$Storage\ reduction\ percentage = 100 * \left(1 - \frac{\frac{3}{2}}{1 + 1}\right)\% = 25\% \qquad (4.2)$$

Таким образом, предлагаемая система использует на 25% меньше места для хранения по сравнению с одним сервером резервного копирования, при этом на 50% меньше места при использовании 2 резервных серверов. Рассмотрим следующий пример.

В системе полного резервного копирования, состоящей из 1 резервного сервера на каждый сервер, создается одна копия каждого объекта. Таким образом, для хранения файла размером 10 МБ потребуется 20 МБ. В предлагаемой системе файл разбивается на две половины (5 + 5) и рассчитывается стирающий код (5 МБ), таким образом, всего требуется 15 МБ по сравнению с системой полного резервного копирования с одним уровнем резервного копирования (на 25% меньше). В системе полного резервного копирования с двумя серверами резервного копирования требуется 30 МБ (что означает, что предлагаемой системе требуется на 50 % меньше).

Далее в разделе 4.3 будет рассмотрен сценарий работы реальной системы и описанное в данном разделе утверждение в формуле 4.2 будет подтверждено.

4.2. Оценка эффективности предлагаемой модели децентрализованного контроля доступа

Измерения задержек, связанные с процессами контроля доступа

Для оценки временных характеристик DApps, реализующих предложенные модель и алгоритмы контроля доступа, был разработан ряд тестов эмуляции работы системы контроля доступа (DAC+RBAC) в разных архитектурах. Тесты были написаны на C++ на фреймворке QT, а деревья контроля доступа были представлены двумя разными способами: файлами SQLite и файлами JSON, поскольку технологии реализации влияют на результаты производительности.

Тестовое окружение было развернуто под управлением Ubuntu 22.04 LTS на процессоре Core i7-8575u и 16 ГБ оперативной памяти.

В начальном эксперименте была поставлена цель оценить время создания объединенного дерева с точки зрения времени и места для хранения, а также сделать выбор технологий для его хранения и обработки. Эксперимент включал создание с нуля двух пользовательских деревьев DAC, состоящих из 500 новых файлов каждое, для создания объединенного дерева. Для создания хэшей узлов использовалась хэш-функция SHA-256. Тест проводился автоматически 100 раз. С такой единовременной нагрузкой сталкивается информационная система, где одновременно работает 1000 пользователей со средней степенью интенсивности работы с файлами (системы внешнего и внутреннего документооборота, CRM системы и т.д.) Среднее время этого процесса составило 9,48 секунды в случае использования SQLite и 0,55 секунды в случае файлов JSON. Общий размер дерева в случае SQLite составляет 249,9 КБ, а при использовании JSON — 552 КБ. Реализация включала объединение 2 деревьев в одно, сжатие и преобразование в двоичный формат, чтобы минимизировать размер пользовательского запроса. Реализация включала объединение 2 деревьев в одно, сжатие и преобразование в двоичный формат, она доступна здесь [51].

В реальных приложениях DAC, вероятно, намного меньше, чем 1000 файлов на узел, поскольку доступ обычно предоставляется на основе ролей, а также путем применения политик к каталогам и их подкаталогам вместо выбора отдельных файлов один за другим, что означает, что размер получившихся деревьев должен быть еще меньше.

Чтобы дать оценку модели контроля доступа и сравнить с другими возможными реализациями, для тестирования были рассмотрены архитектуры 3-х систем:

1- Централизованная система: для ее реализации использовалась PHP-инфраструктура Laravel, поскольку она считается популярной платформой для реализации API. MySQL использовался для хранения политик доступа.

2. Blockchain Hyperledger: данные хранятся непосредственно в блокчейне, а не на внешнем ресурсе для хранения политик. Docker использовался для узлов Hyperledger.

3- DecStore: реализация модели контроля доступа, предложенной в этом исследовании.

Было создано 1000 файловых записей, а также 1000 пользователей. Всем пользователям был предоставлен доступ ко всем файлам (1000*1000). Было выполнено три тестовых случая:

1- Предоставление нового разрешения файлу: которое выражает скорость добавления нового правила доступа для пользователя x к файлу y (1x1).

2- Полный отзыв доступа пользователя: это означает отзыв всех типов доступа, предоставленных пользователю x, к любому файлу в системе (1x1000).

3- Проверка доступа: это таймер, необходимый системе для проверки наличия у пользователя x доступа к файлу y (1x1).

Процессы, необходимые для каждого случая, перечислены в таблице 4.3. в таблице 4.4 представлены результаты тестов.

Таблица 4.3 — Необходимые процессы для тестирования производительности моделей контроля доступа

Тест	**DecStore**	**Блокчейн**	**Централизованный сервер**
Предоставление нового разрешения файлу	- Добавление доступа к дереву пользователей на VD - Добавление доступа к	- Добавление записи в Ledger.	- Добавление записи доступа в БД

	объединенному дереву пользователя - Запуск сжатия + преобразования дерева в двоичные алгоритмы. - Обновление корня дерева MerkleTree в блокчейне		
Полный запрет доступа пользователя	-Удаление деревьев пользователя с VD (2 дерева) - Удаление объединенного дерева - Обновление записи пользователя в блокчейне.	- Обновление записи пользователя в блокчейне.	- Удаление записи доступа из БД
Проверка доступа	Проверка доказательства Меркла	Сравнение совпадений сохраненных значений в Ledger	Проверка наличия записи в БД

Таблица 4.4 — Результат тестирования скорости процессов контроля доступа

Тест (измеряется в мс)	**DecStore**		**Блокчейн**	**Централизованный сервер (целевые показатели)**
	JSON	**SQLite**		
Предоставление нового разрешения файлу	157	190	100	60
Полный запрет доступа пользователя	330	330	300	60
Проверка доступа	130	140	100	10

По результатам тестов производительности можно сделать вывод, что скорость предлагаемой системы в различных операциях несущественно меньше по сравнению с другими системами, что не является критичным, так как в отличие от остальных архитектур DecStore поддерживает масштабируемость. Сделан вывод, что файлы JSON могут быть хорошим способом представления деревьев. Однако производительность может отличаться при реализации с использованием другого стека технологий.

Предлагаемая модель контроля доступа использует узлы хранения для хранения всех данных управления доступом, в то время как пользователи имеют копию своих собственных данных, связанных с доступом, что можно рассматривать как метод кэширования. Без этой техники кэширования осуществляется 4 запроса, т.к. пользователь отправляет запрос на доступ к файлу, затем запрос отправляется в блокчейн, после чего пересылается на узел хранения, который содержит информацию о доступе, после этого ответ отправляется обратно на узел блокчейна, и только после этого клиенту. При использовании кэширования необходимое количество запросов сокращается до 2 вместо 4. Предположим, что каждый пользователь имеет доступ к 1000 файлам и права доступа к 3 файлам меняются в неделю. Также предположим, что пользователь

обращается к 10 файлам в день и ко всем файлам у него уже есть доступ. Это означает, что ежедневно отправляется 20 запросов + 3 операции синхронизации, выполняемые в неделю, которые включают только затронутый узел хранения, на котором размещено только затронутое объединенное дерево. Это сокращение означает гораздо меньшую нагрузку на узлы хранения, учитывая, что система может содержать тысячи пользователей. Что касается стороны блокчейна, соответствующие запросы (20,000) распределяются между узлами блокчейна, поскольку они играют равную роль в этом процессе.

Пользователь также может быстро найти свойства файла, поскольку промежуточные узлы дерева представляют папки, а узлы объединения имеют список прямых папок, которые они содержат, поэтому файл можно найти непосредственно, вместо проверки всех узлов один за другим.

При добавлении нового доступа к объединенному дереву пользователя запуск алгоритма сжатия и преобразование дерева в двоичный файл может выполняться быстрее, чем при создании с нуля, так как дерево может вообще не требовать сжатия, а результат запуска алгоритма преобразования преобразование дерева в двоичный формат может привести к добавлению нескольких узлов без изменения большинства узлов, поскольку оно уже находится в двоичной форме.

Однако предлагаемая система имеет некоторые ограничения:

- Пользователи должны регулярно синхронизировать свои деревья, чтобы получить доступ к системе. Однако синхронизация не означает копирование всех деревьев пользователей. Изменения можно обнаружить путем сравнения общего хэша с тем, который хранится в блокчейне. Если хэш отличается, узел с измененным деревом можно определить по его хэшу.

- Частые изменения правил доступа означают частую синхронизацию изменений и запуск связанных алгоритмов. Однако доступ можно назначать по каталогам вместо выбора отдельных файлов, а в некоторых случаях вместо DAC можно использовать RBAC, что позволяет эффективно минимизировать количество требуемых процессов.

Предлагаемая модель была разработана с учетом требований производительности, безопасности, масштабируемости и доступности:

Производительность

• Введена технология кэширования для уменьшения количества запросов к внешнему хранилищу.

• Когда пользователь запрашивает файл, файл может быть быстро найден, поскольку узлы файлов и каталогов хранятся в схеме, которая соответствует их абсолютным путям в модели сверху вниз.

• Изменения в любом узле дерева могут быть обнаружены немедленно, поскольку изменяется хэш, и в результате изменяется хэш корневого узла.

• При предоставлении или отзыве пользователю доступа к новому доступу к файлу или папке один узел хранения обновляет свое дерево, а другие узлы такое изменение не затрагивает.

• При полном удалении пользователя удаляются его деревья, что считается быстрым, поскольку деревья пользователей изолированы и могут быть представлены одним файлом на каждом виртуальном диске, что намного быстрее, чем поиск файлов в системе и обновление списка разрешено пользователям в этих файлах по одному.

Безопасность

Корневой хэш дерева Меркла в блокчейне изменяется всякий раз, когда доступ к файлу предоставляется или отзывается. Несмотря на то, что у пользователя есть своя версия дерева, он не может манипулировать ею, добавляя новый узел дерева или изменяя тип доступа к листу. Если пользователь меняет тип доступа к листу, его хэш меняется, и когда запрос отправляется в блокчейн, блокчейн хэширует это значение и объединяет его с другими хэшами, как описано в алгоритме проверки доступа пользователей. Это приведет к другому общему

хэшу, что приведет к невозможности доступа пользователя к ресурсу, что можно рассматривать как безопасную модель в сети с нулевым доверием.

Масштабируемость

Предлагаемая модель контроля доступа обладает высокой масштабируемостью, поскольку:

1. На стороне блокчейна для каждого пользователя и роли корневого дерева Merkle хранится только хэш корневого дерева Merkle, вместо хранения матрицы контроля доступа пользователей к файлам. Это означает, что требуется гораздо меньше места для хранения.

2. Что касается хранилища узлов, DecStore гарантирует, что система будет оставаться сбалансированной при добавлении или потере узла хранения, перемещая виртуальные диски в рамках определенных алгоритмов, а также деревьев ролей. Деревья доступа пользователей перемещаются вместе с их виртуальными дисками, поэтому они распределяются по узлам хранения, независимо от размера системы.

3. Кэш поддерживается на стороне пользователя, что означает меньшую нагрузку на систему.

Доступность

• Система использует алгоритмы, позволяющие иметь две копии ролей и три копии частных деревьев пользователей. В случае потери узла хранения его деревья копируются из резервных копий в другие узлы хранения, чтобы сохранить данные доступными в случае потери другого узла.

• Если по какой-либо причине пользовательская копия дерева Меркла утеряна, то в следующий раз, когда он отправит запрос на синхронизацию, все деревья со всех узлов будут считаться несинхронизированными, и пользователь получит копии всех из них, а итоговое дерево после этого будет восстановлен.

Предлагаемая система контроля доступа была разработана с учетом совместимости с предлагаемой моделью хранения файлов. Однако эту работу можно адаптировать для других типов систем, которые используют блокчейн и автономное хранилище для хранения данных любого типа.

4.3. Оценка размера выходных объектов и распределения объектов

В предлагаемой модели распределения используется стирающий код для уменьшения избыточности данных. RAID-5 и RAID-6 — одни из наиболее часто используемых RAID-технологий, которые также используют стирающий код разными способами, отличными от предлагаемых модели и алгоритмов. Некоторые другие обсуждаемые технологии можно рассматривать как системы полного резервного копирования с несколькими резервными копиями с точки зрения надежности, поскольку они ориентированы на то, как узлы хранения и их репликации группируются и управляются.

Произведено моделирование расчета размер выходных объектов в системе (S_{Output}) и отклонение значения размера хранилища от равномерного распределения (ΔS) в сравнении с другими известными подходами: RAID-5, RAID-6 и системы с полным резервным копированием, где используется 1, 2 и 3 копии (ПРК-1, ПРК-2, ПРК-3 соответственно), что покрывает все случаи применения информационных систем, где коэффициент оперативной готовности должен быть не менее 0,99. Для расчета использовались следующие параметры емкости дисковых носителей = {2 TB, 3TB, 2TB, 3TB …} (где первый узел в системе имеет емкость 2 ТБ, второй — 3 ТБ и т. д. Узлы различаются по емкости, чтобы затруднить равномерное распределение данных), R_{rate} = 50%, Th = 30MB/s, Количество узлов варьируется от 4 до 13. Предполагается, что размер S_{input} равен 50% емкости системы. Т.к. $S_{metadata}$ и $S_{accesscontrol}$ не существенно, с точки зрения общего объема данных, то дальше ($S_{output} = S_{input} + S_{redunduncy}$). Также допустим, что размер VD для DecStore = 20GB.

В DecStore данные распределяются по узлам поровну, с небольшим ΔS, вызванным резервированием VD фиксированного размера. В RAID файлы распределяются поровну и распределяются по узлам, без учета того, что узлы с большей емкостью должны хранить больше объектов, что приводит к высокому ΔS. Системы полного резервного копирования не имеют встроенного метода распределения данных, поэтому предположим, что они также одинаково хранят файлы на всех узлах, а резервные копии дублируют их. Однако не все данные могут быть сохранены в предлагаемом сценарии. Например, в случае 2 резервных копий при наличии 4 узла $S_{input} = 50\% * (2 + 3 + 2 + 3)$ТБ $= 5$ТБ, а емкость системы 10ТБ. При этом для системы резервного копирования с двумя серверами резервного копирования требуется 15 ТБ для хранения данных, что превышает доступное пространство. С двумя резервными копиями можно хранить только 2 ТБ, а четвертый узел остается непригодным для использования, так как у него нет резервных копий.

В DecStore размер данных для резервного копирования = 0.75 * размер входных данных.

В RAID-5: $S_{Output} = \frac{S_{Input}}{|\boldsymbol{N}|-1} * |\boldsymbol{N}|$

В RAID-6: $S_{Output} = \frac{S_{Input}}{|\boldsymbol{N}|-2} * |\boldsymbol{N}|$

и ПРК-1 размер данных для резервного копирования = размер входных данных.

В ПРК-2 размер данных для резервного копирования = 2*размер входных данных.

В ПРК-3 размер данных для резервного копирования = 3*размер входных данных.

Если система состоит из узлов оптимальное распределение по этим узлам при использовании 75% общего объема хранилища составляет 0.75*2 = 1,5 ТБ для узлов 1 и 3, 0.75*3 = 2.25 ТБ для узлов 2 и 4. В DecStore, используются VDs распределяются как следующе: 75VD, 113VD, 75VD, 112VD, и это значит по размерам 1.5ТБ, 2.26ТБ, 1,5ТБ, 2.24ТБ. И так, $\Delta S = 0 + 0.01 + 0 + 0.01 = 0.02$ТБ.

При этом, в RAID-5 каждый узел содержит $\frac{S_{Input}}{|N|-1} = 1.667$ТБ, и $S_{Output} = 6.667$. Оптимальный размер данных для узлов 1и3: $\frac{S_{Output}}{2+3+2+3} * 2\text{ТБ} = 1.3333\text{ТБ}$. Оптимальный размер данных для узлов 1и3: $\frac{S_{Output}}{2+3+2+3} * 3\text{ТБ} = 2\text{ТБ}$

$$\Delta S = 0.333 + 0.333 + 0.333 + 0.333 = 1{,}333\text{ТБ}$$

Используя тот же принцип, S_{Output} и ΔS можно рассчитать для остальных систем. результаты расчетов представлены в таблицах (4.5) и (4.6).

Таблица 4.5 — Размер выходных объектов (S_{Output})(ТБ)

|N|	4	5	6	7	8	9	10	11	12	13
DecStore	7.5	9	11.25	12.75	15	16.5	18.75	20.25	22.5	24
RAID-5	6.67	7.5	9	9.92	11.43	12.37	13.89	14.85	16.36	17.34
RAID-6	10	10	11.25	11.9	13.34	14.14	15.62	16.5	18	18.9
ПРК-1	10	12	15	17	20	22	25	27	30	32
ПРК-2	15	18	22.5	25.5	30	33	37.5	40.5	45	48
ПРК-3	20	24	30	34	40	44	50	54	60	64

Таблица 4.6 — отклонение значения размера хранилища от равномерного распределения (ΔS)(ТБ)

|N|	4	5	6	7	8	9	10	11	12
DecStore	0.02	0.02	0.03	0.03	0.04	0.04	0.05	0.05	0.06
RAID-5	1.34	1.5	1.8	2	2.28	2.25	2.78	3	3.27
RAID-6	2	2	2.25	2.4	2.67	2.85	3.125	3.34	3.6
ПРК-1	0	2	0	2	0	2	0	2	0
ПРК-2	4	6	0	2	5	1	4	6	0
ПРК-3	2	4	7	9	0	2	5	7	2

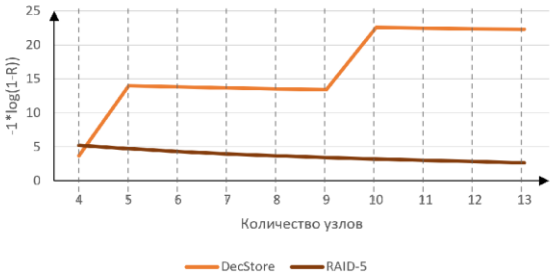
Рассмотрим сценарий реальной системы обмена данными ООО «РИИТ». Рассматриваемая система автоматизирует обмен данными между 5-ю

компаниями. Каждая компания хранит 100,000 файлов, размер каждого файла составляет 10 МБ. Итак, в основном каждая компания хранит 1 ТБ. Всего при использовании резервных копий требуется хранить 10 ТБ данных. При использовании DecStore для всей системы требуется 7.5 ТБ.

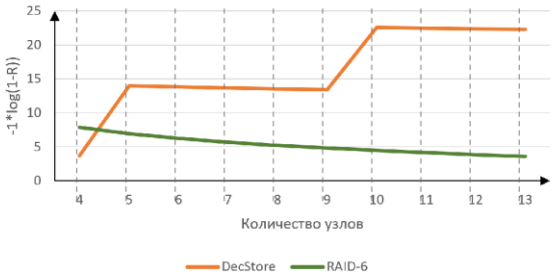
4.4. Оценка надежности предлагаемой модели распределения данных

С помощью математической модели расчета надежности произведено моделирование расчета надежности системы в сравнении с другими известными подходами по тому же сценарию, описанному выше. Так как результаты надежности близки к 1 (100%), то используется преобразование $-1 * \log(1 - R)$.

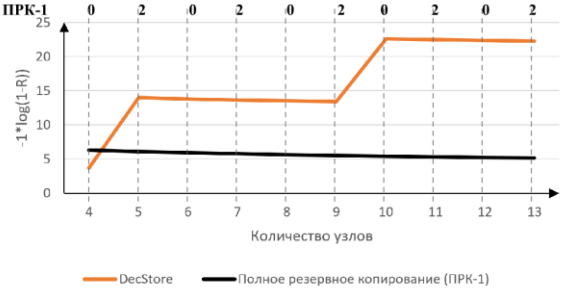
S_{Output} (ТБ)											
	DecStore	7.5	9	11.25	12.75	15	16.5	18.75	20.25	22.5	24
	RAID-5	6.67	7.5	9	9.92	11.43	12.37	13.89	14.85	16.36	17.34
△S (ТБ)	DecStore	0.02	0.02	0.03	0.03	0.04	0.04	0.05	0.05	0.06	0.06
	RAID-5	1.34	1.5	1.8	2	2.28	2.25	2.78	3	3.27	3.5



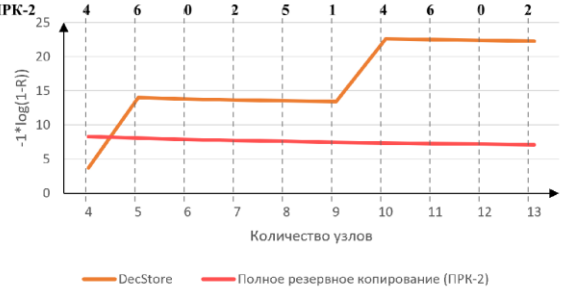
	DecStore	7.5	9	11.25	12.75	15	16.5	18.75	20.25	22.5	24
	RAID-6	10	10	11.25	11.9	13.34	14.14	15.62	16.5	18	18.9
	DecStore	0.02	0.02	0.03	0.03	0.04	0.04	0.05	0.05	0.06	0.06
	RAID-6	2	2	2.25	2.4	2.67	2.85	3.125	3.34	3.6	3.8



S_{Output} (ТБ)											
	DecStore	7.5	9	11.25	12.75	15	16.5	18.75	20.25	22.5	24
	ПРК-1	10	12	15	17	20	22	25	27	30	32
△S (ТБ)	DecStore	0.02	0.02	0.03	0.03	0.04	0.04	0.05	0.05	0.06	0.06
	ПРК-1	0	2	0	2	0	2	0	2	0	2



	DecStore	7.5	9	11.25	12.75	15	16.5	18.75	20.25	22.5	24
	ПРК-2	15	18	22.5	25.5	30	33	37.5	40.5	45	48
	DecStore	0.02	0.02	0.03	0.03	0.04	0.04	0.05	0.05	0.06	0.06
	ПРК-2	4	6	0	2	5	1	4	6	0	2



S_{Output} (ТБ)											
	DecStore	7.5	9	11.25	12.75	15	16.5	18.75	20.25	22.5	24
	ПРК-3	20	24	30	34	40	44	50	54	60	64
△S (ТБ)	DecStore	0.02	0.02	0.03	0.03	0.04	0.04	0.05	0.05	0.06	0.06
	ПРК-3	2	4	7	9	0	2	5	7	2	4

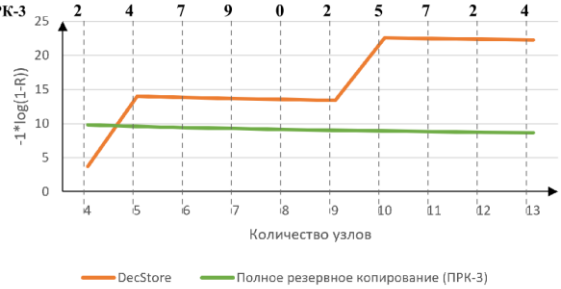


Рисунок 4.2 – Результаты расчета $R, S_{Output}, \Delta S$ для разных моделей распределения данных

В этих сценариях моделирования расчета надежности, рассматривается случай, когда в системе нет замены потерянным узлам. Например, в RAID-5 P_{SFANF} равен 1 после потери одного узла, поскольку потеря еще одного узла означает безвозвратную потерю данных.

Учитывая общую надежность с учетом общих метаданных и контроля доступа, надежность DecStore не меняется, поскольку данные распределяются по узлам хранения и используют один и тот же метод восстановления данных (данные контроля доступа обрабатываются как другие объекты). Однако для других решений, использующих централизованный сервер для метаданных и контроля доступа, надежность снижается, поскольку зависит от единой точки отказа, и диаграмма надежности выглядит следующим образом:

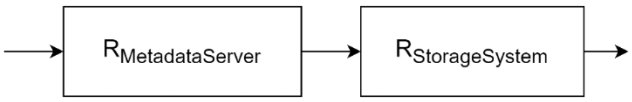


Рисунок 4.3 – Диаграмма надежности систем хранения данных с сервером централизованного управления

Исходя из этого, надежность этих систем можно рассчитать по следующей формуле:

$$R = R_{MetadataServer} * R_{StorageSystem} \quad (4.3)$$

Суммарная надежность систем для узлов от 4 до 13 представлена в таблице 4.7.

Таблица 4.7 – Полная надежность систем с учетом контроля доступа и метаданных

	DecStore	RAID-5	RAID-6	ПРК-1	ПРК-2	ПРК-3
4	97.49	98.7	99.3	99.1	99.3	99.3

5	99.99991	98.4	99.2	99.07	99.3	99.3
6	99.9999	97.9	99.1	99.07	99.3	99.3
7	99.9998	97.4	98.9	98.9	99.3	99.3
8	99.9998	96.7	98.7	98.9	99.3	99.3
9	99.9998	96.09	98.4	98.8	99.3	99.3
10	99.99999998	95.2	98.1	98.8	99.3	99.3
11	99.99999998	94.4	97.7	98.8	99.3	99.3
12	99.99999998	93.5	97.1	98.7	99.3	99.3
13	99.99999997	92.4	96.5	98.7	99.2	99.3

Из графика и таблиц видно, что надежность DecStore повышается, когда система растет и увеличивает количество используемых серверов. В случае традиционных технологий, таких как RAID-5 и RAID-6, с ростом количества узлов надежность становится меньше, т.к. вероятность выхода из строя 2-х или 3-х узлов одновременно увеличивается.

Обсуждение эффективности применения моделей: предлагаемая модель позволяет немедленно восстанавливать потерянные данные на оставшихся узлах таким образом, что ее можно повторить несколько раз (множественное восстановление). Этот процесс происходит автоматически, и не возникает задержек из-за замены недостающих узлов, которая может занять часы или даже дни. При восстановлении потерянного узла его данные автоматически восстанавливаются и распределяются по остальным узлам. Восстановление потерянных узлов занимает относительно небольшое количество времени, поскольку потерянные данные восстанавливаются на остальных узлах, этот процесс распределяется по узлам одновременно. Кроме того, вероятность повторного сбоя при восстановлении потерянного узла невелика.

4.5. Технологические ограничения модели распределения и алгоритмов балансировки и модели контроля доступа

Существуют некоторые ограничения и требования, которые следует учитывать при реализации модели распределения данных, алгоритмов балансировки, модели контроля доступа в DApps:

- Количество узлов должно быть более 3 узлов. Чем больше узлов, тем выше надежность.
- Процент использования хранилища узлов: системные узлы не могут быть использованы полностью, чтобы иметь достаточно места для восстановления данных при отказе узлов.
- Максимальный размер файла определяется емкостью виртуального диска * 2, поскольку файлы разделяются. Если размер виртуального диска был установлен на 20 ГБ, максимальный размер файла составит 40 ГБ.
- Рассматриваемый в данной работе класс информационных систем, для которых подходят предложенные модели и алгоритмы, не подходит для задач хранения объектов, требующих непосредственной обработки внутри DApps (прямое изменение данных внутри системы, запись видео непосредственно на диски системы и т.д.). Все операции чтения, записи и удаления объектов должны осуществляться только через балансировщик нагрузки.

Выводы по главе 4

1. Были проведены тесты для выбора подходящей сети блокчейна для поставленной задачи в качестве балансировщика нагрузки. Результат тестов показал, что использование наиболее известных блокчейн-платформ вполне приемлемо для конечных пользователей. Реализация предложенного подхода в публичных сетях, таких как Ethereum, затруднена из-за непредсказуемой скорости обработки транзакций и избыточных вычислений при достижении консенсуса PoW алгоритмами. Согласно полученным результатам, Hyperledger Fabric можно рассматривать как подходящую блокчейн-сеть для представленных моделей,

поскольку она работает лучше по сравнению с другими сетями, поддерживает смарт-контракты и не требует криптовалюты.

2. Был проведен ряд экспериментов для оценки производительности предлагаемой системы хранения и алгоритмов балансировки, включая добавление файла, добавление узла, восстановление потерянного узла и повторное присоединение потерянного узла к системе. Результаты показывают, что производительность предлагаемой системы хранения считается более высокой по сравнению с ожидаемыми целевыми показателями.

3. Проведено моделирование применения описанных моделей в сравнении с RAID-5, RAID-6, полным резервным копированием, которые состоят из резервных копий от 1 до 3, Показано, что показатели S_{Output} и ΔS при этом ниже.

3. Проведено сравнение надежности системы с другими системами хранения, включая RAID-5, RAID-6 и системами полного резервного копирования с 1, 2 и 3 резервными копиями. Было проиллюстрировано, насколько надежность предлагаемой системы может превзойти эти системы при росте количества узлов хранения и учете достаточного количества свободной памяти на узлах.

4. Были обсуждены некоторые аспекты предлагаемой системы хранения, состоящей из модели и алгоритмов, касающиеся производительности, масштабируемости, требуемой памяти и предотвращения узких мест. Приведены утверждения, как предложенная модель может помочь оптимизировать указанные параметры.

5. Было проведено несколько тестов для оценки производительности и требуемого размера хранилища соответствующей предложенной модели контроля доступа. Показано, что эта модель требует небольшого объема памяти на узлах хранения и минимальной памяти для хранения данных в блокчейне, при этом скорость её работы сопоставима с централизованными моделями контроля доступа. Также обсуждались некоторые аспекты этой модели контроля доступа, касающиеся производительности, доступности и масштабируемости.

6. Систему, которая использует предложенные модели распределения и контроля доступа имеет ряд дополнительных преимуществ, которые позволяют увеличивать количество узлов и количества объектов на них (масштабирование). При добавлении нового узла часть данных перемещается на новый узел, чтобы поддерживать баланс системы. В некоторых системах, таких как системы полного резервного копирования или Gluster, невозможно добавить ни одного узла. Кроме того, система обеспечивает вертикальную масштабируемость. При обновлении памяти хранилища узла вызывается Алгоритм (A1), который предполагает, что обновляемый узел является новым узлом, и часть перемещения данных уже завершена, то есть перемещается только необходимый объем данных с других узлов на этот узел для достижения уровня баланса системы. Кроме того, узлы хранения в DecStore могут иметь разную емкость хранения, в отличие от других систем, таких как RAID.

7. DecStore можно использовать для решения множества прикладных задач. Потенциальные задачи применения DecStore в корпоративных системах:

- Системы, требующие хранения больших объемов файлов транзакций, например файлы SAP XML [97].
- Библиотеки изображений, аудио и видео, предназначенные для продюсерских компаний, таких как Sony Pictures [104].
- Замена облачных сервисов, таких как Amazon AWS.

DecStore также можно использовать в сервисах для конечных пользователей:

- Системы, предлагающие множество наборов данных для приложений нейронных сетей, такие как Kaggle [60].
- NFT-системы, хранящие метаданные файлов (или даже файлы целиком), такие как OpenSea [87].
- Замена систем хранения файлов, таких как Google Drive [90].

Для государственных систем DecStore может быть полезен в сценариях, когда необходимо организовать проверку и аудит со стороны государства деятельности компаний, где существуют ограничения по ресурсам и требования к

надежности. Например, контроль лицензирования различных видов деятельности (финансы, торговля товарами, требующих регулирования государством, оказание эксклюзивных услуг), сбор налоговых пошлин, предоставление финансовой отчетности для получения льгот и т.д.

ЗАКЛЮЧЕНИЕ

В диссертационной работе решена важная научная и практическая задача внедрения DApps приложений - уменьшение избыточности циркулирующих в системе данных без потери надежности. В диссертационной работе был проведен комплексный анализ изучения методов и проектов распределения данных по набору параметров. Было установлено, что не существует научных подходов, которые решают задачу уменьшения избыточности данных без ущерба для надежности в распределенных системах. Анализ известных моделей контроля доступа в распределённых системах показал, что задача минимизации размера данных доступа, хранящийся на стороне блокчейна, на сегодняшний день не решается и для этого используются централизованные решения, что, в свою очередь, не позволяет увеличивать количество узлов и объем данных в системе.

Получены следующие научные результаты:

1) Разработана модель распределения данных в блокчейн и на узлах DApps систем, а также 4 алгоритма балансировки данных, которые оперируют этой моделью. Алгоритмы позволяют распределять данные по узлам автономно (без участия пользователя) и таким образом равномерно балансировать нагрузку. При этом балансировщик нагрузки работает в среде смарт-контрактов, что накладывает ограничения на вычислительную сложность алгоритмов. Модель распределения данных и алгоритмы балансировки позволяют уменьшить общее место хранения данных на 25% относительно систем с полным резервным копированием. Данное утверждение было подтверждено экспериментально при помощи реализации сценария поведения системы на 10 узлах и 1500 пользователей.

2) Реализация модели контроля доступа позволяет хранить только часть данных о доступе субъектов к объектам в блокчейн и таким образом минимизировать общий объем данных в реализуемой DApps системе. Новые алгоритмы создания дерева Меркла, сжатия дерева и кеширования оперируют данными этой модели и обеспечивают доступ к данным DApps в сети с нулевым

доверием. Экспериментально показано, что при росте размера системы скорость обработки основных операций сравнима с централизованными системами.

3) Разработана модель надежности, которая учитывает возможности автоматического восстановления данных на других узлах системы, в то время как известные модели надежности эту возможность не рассматривают. Было проведено моделирование оценки надежности рассматриваемой системы в сравнении с системами другой архитектуры и показано, что надежность рассматриваемой системы может превосходить надежность других систем при увеличении количества узлов и наличии достаточного свободного места на узлах для восстановления данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. A survey of DHT security techniques | ACM Computing Surveys A survey of DHT security techniques | ACM Computing Surveys [Электронный ресурс]. URL: https://dl.acm.org/doi/10.1145/1883612.1883615 (дата обращения: 02.01.2024).

2. Adams C. Authorization architecture // Encyclopedia of Cryptography and Security. под ред. H. C. A. van Tilborg, Boston, MA. Springer US, 2005. – С. 23–27. – DOI. 10.1007/0-387-23483-7_18.

3. Androulaki E. [и др.]. Hyperledger fabric: a distributed operating system for permissioned blockchains // Proceedings of the Thirteenth EuroSys Conference. Porto Portugal. ACM, 2018. – С. 1–15. – DOI. 10.1145/3190508.3190538.

4. Arefin S.E. Auditing Lustre file system / S. E. Arefin, 2023. – DOI. 10.48550/arXiv.2302.14824.

5. Arslan S.S. Durability and Availability of Erasure-Coded Storage Systems with Concurrent Maintenance // 2023.

6. Aufiero S., Ibba G., Bartolucci S., Destefanis G., Neykova R., Ortu M. The network structure of smart contracts in Ethereum dApps / S. Aufiero, G. Ibba, S. Bartolucci, G. Destefanis, R. Neykova, M. Ortu, 2023.

7. Balaji S., Muralee Krishnan N.K., Vajha M., Ramkumar V., Sasidharan B., Kumar P. Erasure coding for distributed storage: an overview // Science China Information Sciences. 2018. – Т. 61. – DOI. 10.1007/s11432-018-9482-6.

8. Bashir I. Blockchain Consensus // Blockchain Consensus : An Introduction to Classical, Blockchain, and Quantum Consensus Protocols. под ред. I. Bashir, Berkeley, CA. Apress, 2022. – С. 207–257. – DOI. 10.1007/978-1-4842-8179-6_5.

9. Benet J. IPFS - Content Addressed, Versioned, P2P File System // 2014.

10. Besu documentation Besu documentation [Электронный ресурс]. URL: https://besu.hyperledger.org/ (дата обращения: 01.01.2024).

11. Borges G., Crosby S., Boland L. CephFS: a new generation storage platform for Australian high energy physics // Journal of Physics: Conference Series. 2017. – Т. 898. – №. 6. – С. 062015. – DOI. 10.1088/1742-6596/898/6/062015.

12. Boyer E., Broomfield M., Perrotti T. GlusterFS One Storage Server to Rule Them All.

13. Braam P.J. Lustre: A Scalable, High-Performance File System – С. 13.

14. Build cross-platform desktop apps with JavaScript, HTML, and CSS | Electron Build cross-platform desktop apps with JavaScript, HTML, and CSS | Electron [Электронный ресурс]. URL: https://electronjs.org/ru/ (дата обращения: 15.04.2023).

15. Cachin C. Architecture of the Hyperledger Blockchain Fabric 2016.

16. Carminati B., Ferrari E., Perego A. Rule-Based Access Control for Social Networks // On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops. под ред. R. Meersman, Z. Tari, P. Herrero, Berlin, Heidelberg. Springer, 2006. – С. 1734–1744. – DOI. 10.1007/11915072_80.

17. Chandran R. Pros and Cons of Merkle Tree 2023. – С. 649–653. – DOI. 10.52458/978-81-955020-5-9-61.

18. Chen M. Comparison on Proof of Work Versus Proof of Stake and Analysis on Why Ethereum Converted to Proof of Stake 2023. – Т. AEMPS Vol.12. – С. 200–204. – DOI. 10.54254/2754-1169/12/20230624.

19. Chen X., Zhang K., Liang X., Qiu W., Zhang Z., Tu D. HyperBSA: A High-Performance Consortium Blockchain Storage Architecture for Massive Data // IEEE Access. 2020. – Т. 8. – С. 178402–178413. – DOI. 10.1109/ACCESS.2020.3027610.

20. Chenchev I. Classification of the DLT Consensus Algorithms with Focus on Blockchain // Intelligent Sustainable Systems. под ред. A. K. Nagar, D. Singh Jat, D. K. Mishra, A. Joshi, Singapore. Springer Nature, 2023. – С. 731–740. – DOI. 10.1007/978-981-19-7663-6_68.

21. Cheng C., Yan B., Wang G. The blockchain based access control scheme for the Internet of Things // Procedia Computer Science. 2022. – Т. 202. – С. 342–347. – DOI. 10.1016/j.procs.2022.04.046.

22. Chiusole A., Cozzini S., Ster D. van der, Lamanna M., Giuliani G. An I/O Analysis of HPC Workloads on CephFS and Lustre // High Performance Computing. под ред. M. Weiland, G. Juckeland, S. Alam, H. Jagode, Cham. Springer International Publishing, 2019. – C. 300–316. – DOI. 10.1007/978-3-030-34356-9_24.

23. Clinton D., Piper B. Introduction to Cloud Computing and AWS // AWS Certified Solutions Architect Study Guide: Associate SAA-C02 Exam. Wiley, 2021. – C. 1–19.

24. Cooley A. Examining Development of E-Government in Russia and China: A Comparative Approach // International Journal of Public Administration. 2017. – C. 899–908. – DOI. 10.1080/01900692.2017.1300915.

25. Craft G. Big Data, Both Friend And Foe: The Intersection Of Privacy And Trade On The Transatlantic Stage // University of Miami International and Comparative Law Review. 2022. – Т. 29. – №. 2. – С. 99.

26. DADS: Decentralized (Mobile) Applications Deployment System Using Blockchain : Secured Decentralized Applications Store | IEEE Conference Publication | IEEE Xplore DADS: Decentralized (Mobile) Applications Deployment System Using Blockchain : Secured Decentralized Applications Store | IEEE Conference Publication | IEEE Xplore [Электронный ресурс]. URL: https://ieeexplore.ieee.org/document/9232506 (дата обращения: 01.11.2023).

27. Dannen C. Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners / C. Dannen, Berkeley, CA. Apress, 2017. [Электронный ресурс]. URL: https://link.springer.com/10.1007/978-1-4842-2535-6 (дата обращения: 04.10.2023). – DOI. 10.1007/978-1-4842-2535-6.

28. Dholi A., Mandawade P., Pagare N., Bhangale K., Nalge P. Blockchain Application Using NFT Marketplace // International Journal of Advanced Research in Science, Communication and Technology. 2023. – С. 444–451. – DOI. 10.48175/IJARSCT-13162.

29. Discretionary-Access Control and the Access-Matrix Model Discretionary-Access Control and the Access-Matrix Model // Access Control Systems: Security, Identity Management and Trust Models. под ред. M. Benantar, Boston, MA. Springer US, 2006. – С. 147–167. – DOI. 10.1007/0-387-27716-1_5.

30. Doan T.V., Psaras Y., Ott J., Bajpai V. Toward Decentralized Cloud Storage With IPFS: Opportunities, Challenges, and Future Considerations // IEEE Internet Computing. 2022. – Т. 26. – №. 6. – С. 7–15. – DOI. 10.1109/MIC.2022.3209804.

31. Docker: Accelerated, Containerized Application Development Docker: Accelerated, Containerized Application Development [Электронный ресурс]. URL: https://www.docker.com/ (дата обращения: 15.04.2023).

32. Dubey P., Kumar Tiwari A., Raja R. Amazon Web Services: the Definitive Guide for Beginners and Advanced Users / P. Dubey, A. Kumar Tiwari, R. Raja, BENTHAM SCIENCE PUBLISHERS, 2023. 115–131 – С. [Электронный ресурс]. URL: https://www.eurekaselect.com/222322/volume/1 (дата обращения: 03.01.2024). – DOI. 10.2174/97898151658211230101.

33. Dubey P., Tiwari A., Raja R. Load Balancing and Auto Scaling in AWS 2023. – С. 79–96. – DOI. 10.2174/9789815165821123010006.

34. ELECTRONIC DATA INTERCHANGE (EDI) ELECTRONIC DATA INTERCHANGE (EDI) [Электронный ресурс]. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub161-2.pdf (дата обращения: 27.12.2022).

35. Ethereum Average Block Time Ethereum Average Block Time [Электронный ресурс]. URL: https://ycharts.com/indicators/ethereum_average_block_time (дата обращения: 01.01.2024).

36. Expand virtual hard disks attached to a Windows VM in an Azure - Azure Virtual Machines Expand virtual hard disks attached to a Windows VM in an Azure - Azure Virtual Machines [Электронный ресурс]. URL: https://docs.microsoft.com/en-us/azure/virtual-machines/windows/expand-os-disk (дата обращения: 23.04.2022).

37. Facebook Facebook // Facebook [Электронный ресурс]. URL: https://www.facebook.com (дата обращения: 01.08.2023).

38. Gans J. Proof of Work Versus Proof of Stake // The Economics of Blockchain Consensus: Exploring the Key Tradeoffs in Blockchain Design. под ред. J. Gans, Cham. Springer International Publishing, 2023. – С. 69–83. – DOI. 10.1007/978-3-031-33083-4_5.

39. Ghorashi N., Rahimi M., Sirous R., Javan R. The Intersection of Radiology With Blockchain and Smart Contracts: A Perspective // Cureus. 2023. – T. 15. – DOI. 10.7759/cureus.46941.

40. Gilroy M., Irvine J., Atkinson R. RAID 6 Hardware Acceleration // ACM Transactions on Embedded Computing Systems. 2011. – T. 10. – №. 4. – C. 43:1-43:17. – DOI. 10.1145/2043662.2043667.

41. Giri P., Sharma G. Apache Hadoop Architecture, Applications, and Hadoop Distributed File System // Semiconductor Science and Information Devices. 2022. – T. 4. – C. 14. – DOI. 10.30564/ssid.v4i1.4619.

42. Golosova J., Romanovs A. The Advantages and Disadvantages of the Blockchain Technology // 2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE). Vilnius. IEEE, 2018. – C. 1–6. – DOI. 10.1109/AIEEE.2018.8592253.

43. Gomathy C.K. THE STUDY OF HADOOP ECOSYSTEM // INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT. 2022. – T. 06. – DOI. 10.55041/IJSREM16965.

44. González-Domínguez J., Bolón-Canedo V., Freire B., Touriño J. Parallel feature selection for distributed-memory clusters // Information Sciences. 2019. – T. 496. – C. 399–409. – DOI. 10.1016/j.ins.2019.01.050.

45. Gorman B.L. Azure Storage Ecosystem: Overview and Development with Azure Blob Storage // Developing Solutions for Microsoft Azure Certification Companion: Hands-on Preparation and Practice for Exam AZ-204. под ред. B. L. Gorman, Berkeley, CA. Apress, 2023. – C. 3–41. – DOI. 10.1007/978-1-4842-9300-3_1.

46. Gray G.R. Distributed Applications (dApps) // Blockchain Technology for Managers. под ред. G. R. Gray, Cham. Springer International Publishing, 2021. – C. 89–97. – DOI. 10.1007/978-3-030-85716-5_8.

47. Guo J., Zhu Z.-M., Zhou X., Zhang G.-X. An instances placement algorithm based on disk I/O load for big data in private cloud / J. Guo, Z.-M. Zhu, X. Zhou, G.-X. Zhang, 2012. 287 – C. – DOI. 10.1109/ICWAMTIP.2012.6413495.

48. Guo L., Chen S., Xiao Z., Tan E., Ding X., Zhang X. A performance study of BitTorrent-like peer-to-peer systems // IEEE Journal on Selected Areas in Communications. 2007. – Т. 25. – №. 1. – С. 155–169. – DOI. 10.1109/JSAC.2007.070116.

49. Gupta M. Reviewing the Relationship Between Blockchain and NFT With World Famous NFT Market Places // Scientific Journal of Metaverse and Blockchain Technologies. 2023. – Т. 1. – №. 1. – С. 1–8. – DOI. 10.36676/sjmbt.v1i1.01.

50. Gupta P., Stoller S.D., Xu Z. Abductive Analysis of Administrative Policies in Rule-Based Access Control // IEEE Transactions on Dependable and Secure Computing. 2014. – Т. 11. – №. 5. – С. 412–424. – DOI. 10.1109/TDSC.2013.42.

51. Hammoud O. DistributedFileSystem [Электронный ресурс]. URL: https://github.com/Obadah-H/DistributedFileSystem/blob/735131ac2cf1b89e54fbfb25425294b40a7015d2/jupyter/nodes_recovery_algorithm.ipynb (дата обращения: 15.08.2022).

52. Han R., Gatla O., Zheng M., Cao J., Zhang D., Dai D., Chen Y., Cook J. A Study of Failure Recovery and Logging of High-Performance Parallel File Systems // ACM Transactions on Storage. 2022. – Т. 18. – DOI. 10.1145/3483447.

53. https://www.emergenresearch.com E.R. DApps Market Share | Decentralized application Industry Trend by 2027 [Электронный ресурс]. URL: https://www.emergenresearch.com/industry-report/dapps-market (дата обращения: 12.12.2023).

54. Huang W.-C., Lai C.-C., Lin C.-A., Liu C.-M. File System Allocation in Cloud Storage Services with GlusterFS and Lustre // 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity). 2015. – С. 1167–1170. – DOI. 10.1109/SmartCity.2015.230.

55. Hui Zhao, Zheng Q., Weizhan Zhang, Chen Y., Yunhui Huang Virtual machine placement based on the VM performance models in cloud // 2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC). Nanjing, China. IEEE, 2015. – С. 1–8. – DOI. 10.1109/PCCC.2015.7410296.

56. Hussein Z., Salama M.A., El-Rahman S.A. Evolution of blockchain consensus algorithms: a review on the latest milestones of blockchain consensus algorithms // Cybersecurity. 2023. – Т. 6. – №. 1. – С. 30. – DOI. 10.1186/s42400-023-00163-y.

57. Introduction to Lustre* Architecture Introduction to Lustre* Architecture Lustre* systems and network administration, [Электронный ресурс]. URL: https://wiki.lustre.org/images/6/64/LustreArchitecture-v4.pdf.

58. Jain S.M. IPFS and NFTs // A Brief Introduction to Web3: Decentralized Web Fundamentals for App Development. под ред. S. M. Jain, Berkeley, CA. Apress, 2023. – С. 147–165. – DOI. 10.1007/978-1-4842-8975-4_7.

59. Johnston D. The General Theory of Decentralized Applications, Dapps [Электронный ресурс]. URL: https://github.com/DavidJohnstonCEO/DecentralizedApplications (дата обращения: 15.04.2023).

60. Kaggle Kaggle: Your Machine Learning and Data Science Community [Электронный ресурс]. URL: https://www.kaggle.com/ (дата обращения: 01.01.2024).

61. Kaur M., Gupta S., Kumar D., Raboaca M., Goyal S.B., Verma C. IPFS: An Off-Chain Storage Solution for Blockchain 2023. – С. 513–525. – DOI. 10.1007/978-981-19-9876-8_39.

62. Kemp B., Olivan J. European data format ‘plus’ (EDF+), an EDF alike standard format for the exchange of physiological data // Clinical Neurophysiology. 2003. – Т. 114. – №. 9. – С. 1755–1761. – DOI. 10.1016/S1388-2457(03)00123-8.

63. Kim M., Lee J.-Y., Shah S., Kim T.-H., Noh S.-Y. Min-max exclusive virtual machine placement in cloud computing for scientific data environment // Journal of Cloud Computing. 2021. – Т. 10. – DOI. 10.1186/s13677-020-00221-7.

64. Kingsley M.S. Microsoft Azure // Cloud Technologies and Services : Theoretical Concepts and Practical Applications. под ред. M. S. Kingsley, Cham. Springer International Publishing, 2024. – С. 127–141. – DOI. 10.1007/978-3-031-33669-0_7.

65. Kralevska K. Applied Erasure Coding in Networks and Distributed Storage 2016. – DOI. 10.13140/RG.2.2.33445.19683/3.

66. Krishnan M. Erasure Coding for Big Data // Advanced Computing and Communications. 2019. – DOI. 10.34048/2019.1.F1.

67. Krstić M., Krstić L. Hyperledger frameworks with a special focus on Hyperledger Fabric // Vojnotehnicki glasnik. 2020. – T. 68. – C. 639–663. – DOI. 10.5937/vojtehg68-26206.

68. Laravel - The PHP Framework For Web Artisans Laravel - The PHP Framework For Web Artisans [Электронный ресурс]. URL: https://laravel.com/ (дата обращения: 15.04.2023).

69. Lee C.-D., Choi B.-J., Park K.-S. Design and evaluation of a block encryption algorithm using dynamic-key mechanism // Future Generation Computer Systems. 2004. – T. 20. – №. 2. – C. 327–338. – DOI. 10.1016/S0167-739X(03)00148-1.

70. Liang X. Analysis on Non-Center Cloud Storage Architecture of Gluster // Applied Mechanics and Materials. 2014. – T. 587–589. – C. 2346–2349. – DOI. 10.4028/www.scientific.net/AMM.587-589.2346.

71. Liu Y., Zhang X., Liu B., Zhao X. The research and analysis of efficiency of hardware usage base on HDFS // Cluster Computing. 2022. – T. 25. – №. 5. – C. 3719–3732. – DOI. 10.1007/s10586-022-03597-0.

72. Lubuntu – The official Lubuntu home Lubuntu – The official Lubuntu home [Электронный ресурс]. URL: https://lubuntu.me/ (дата обращения: 01.01.2024).

73. Maesa D., Mori P., Ricci L. Blockchain Based Access Control / D. Maesa, P. Mori, L. Ricci, 2017. 206 – C. – DOI. 10.1007/978-3-319-59665-5_15.

74. Martins J., Barroso J., Gonçalves R., Sousa A., Bacelar M., Paredes H. Transforming e-Procurement Platforms for PEPPOL and WCAG 2.0 Compliance // Information Science and Applications. под ред. K. J. Kim, Berlin, Heidelberg. Springer, 2015. – C. 973–980. – DOI. 10.1007/978-3-662-46578-3_116.

75. Masi M., Pugliese R., Tiezzi F. Formalisation and Implementation of the XACML Access Control Mechanism // Engineering Secure Software and Systems. под ред. G. Barthe, B. Livshits, R. Scandariato, Berlin, Heidelberg. Springer, 2012. – C. 60–74. – DOI. 10.1007/978-3-642-28166-2_7.

76. Menčík J. Concise Reliability for Engineers / J. Menčík, InTech, 2016. [Электронный ресурс]. URL: http://www.intechopen.com/books/concise-reliability-for-engineers (дата обращения: 03.01.2024). – DOI. 10.5772/62009.

77. Merkle Tree: A Fundamental Component of Blockchains | IEEE Conference Publication | IEEE Xplore Merkle Tree: A Fundamental Component of Blockchains | IEEE Conference Publication | IEEE Xplore [Электронный ресурс]. URL: https://ieeexplore.ieee.org/document/9588047 (дата обращения: 30.09.2023).

78. Mishra P. Advanced AWS Services // Cloud Computing with AWS: Everything You Need to Know to be an AWS Cloud Practitioner. под ред. P. Mishra, Berkeley, CA. Apress, 2023. – С. 247–277. – DOI. 10.1007/978-1-4842-9172-6_9.

79. MTTF – What hard drive reliability really means MTTF – What hard drive reliability really means // EMEA Region – Toshiba Storage Solutions [Электронный ресурс]. URL: https://www.toshiba-storage.com/trends-technology/mttf-what-hard-drive-reliability-really-means/ (дата обращения: 27.06.2023).

80. Mudarri T., Abdo S., Al-Rabeei S. SECURITY FUNDAMENTALS: ACCESS CONTROL MODELS // Interdisciplinarity in theory and practice. 2015.

81. Nadiminti K., Assuncao M., Buyya R. Distributed Systems and Recent Innovations: Challenges and Benefits // InfoNet Magazine. 2006. – Т. 16.

82. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System // Cryptography Mailing list at https://metzdowd.com. 2009.

83. Nguyen T.T.T., Le Thi H.A., Doan X.V. Optimizing Merkle Tree Structure for Blockchain Transactions by a DC Programming Approach // Computational Collective Intelligence. под ред. N. T. Nguyen, J. Botzheim, L. Gulyás, M. Núñez, J. Treur, G. Vossen, A. Kozierkiewicz, Cham. Springer Nature Switzerland, 2023. – С. 405–417. – DOI. 10.1007/978-3-031-41456-5_31.

84. Node.js Node.js // Node.js [Электронный ресурс]. URL: https://nodejs.org/ (дата обращения: 15.04.2023).

85. Nycz M., Hajder M., Nienajadlo S. Methods for increasing security of web servers // Annales Universitatis Mariae Curie-Sklodowska, sectio AI – Informatica. 2017. – Т. 16. – №. 2. – С. 39. – DOI. 10.17951/ai.2016.16.2.39.

86. Ølnes J. PEPPOL – Experience from Four Years Work on eSignature Interoperability // ISSE 2012 Securing Electronic Business Processes: Highlights of the Information Security Solutions Europe 2012 Conference. под ред. H. Reimer, N. Pohlmann, W. Schneider, Wiesbaden. Springer Fachmedien, 2012. – С. 282–295. – DOI. 10.1007/978-3-658-00333-3_27.

87. OpenSea OpenSea, the largest NFT marketplace // OpenSea [Электронный ресурс]. URL: https://opensea.io/ (дата обращения: 01.01.2024).

88. Paillisse J., Subira J., Lopez A., Rodriguez-Natal A., Ermagan V., Maino F., Cabellos A. Distributed Access Control with Blockchain // ICC 2019 - 2019 IEEE International Conference on Communications (ICC). 2019. – С. 1–6. – DOI. 10.1109/ICC.2019.8761995.

89. Pawar M.K., Patil P., Sharma M., Chalageri M. Secure and Scalable Decentralized Supply Chain Management Using Ethereum and IPFS Platform // 2021 International Conference on Intelligent Technologies (CONIT). 2021. – С. 1–5. – DOI. 10.1109/CONIT51480.2021.9498537.

90. Personal Cloud Storage & File Sharing Platform - Google Personal Cloud Storage & File Sharing Platform - Google [Электронный ресурс]. URL: https://www.google.com/drive/ (дата обращения: 01.01.2024).

91. Piao X., Ding H., Song H. Performance Analysis of Endorsement in Hyperledger Fabric Concerning Endorsement Policies // Electronics. 2023. – Т. 12. – №. 20. – С. 4322. – DOI. 10.3390/electronics12204322.

92. Pinna A., Tonelli R., Mostarda L. Special Issue - Blockchain and DLT Interoperability / A. Pinna, R. Tonelli, L. Mostarda, 2024.

93. Quan S., Zhao Y., Helil N. Attribute-Based Access Control Policy Generation Approach from Access Logs Based on the CatBoost // Computing and Informatics. 2023. – Т. 42. – С. 615–650. – DOI. 10.31577/cai_2023_3_615.

94. Rao K.K., Hafner J.L., Golding R.A. Reliability for Networked Storage Nodes // International Conference on Dependable Systems and Networks (DSN’06). Philadelphia, PA, USA. IEEE, 2006. – С. 237–248. – DOI. 10.1109/DSN.2006.61.

95. Rostami G. Role-based Access Control (RBAC) Authorization in Kubernetes // Journal of ICT Standardization. 2023. – DOI. 10.13052/jicts2245-800X.1132.

96. Santa R., Macdonald J., Ferrer M. The role of trust in e-Government effectiveness, operational effectiveness and user satisfaction: Lessons from Saudi Arabia in e-G2B // Government Information Quarterly. 2018. – T. 36. – DOI. 10.1016/j.giq.2018.10.007.

97. SAP Software Solutions | Business Applications and Technology SAP Software Solutions | Business Applications and Technology // SAP [Электронный ресурс]. URL: https://www.sap.com/index.html (дата обращения: 01.01.2024).

98. Sato T., Shimosawa T., Kondo Y., Nishijima N. Toward Fully-Decentralized System With Hyperledger Fabric // IEICE Communications Express. 2023. – T. 12. – DOI. 10.1587/comex.2023XBL0011.

99. Savill J. Azure Stack // Microsoft Azure Infrastructure Services for Architects: Designing Cloud Solutions. Wiley, 2020. – C. 281–296. – DOI. 10.1002/9781119596608.ch8.

100. Secure Data Storage and Recovery in Industrial Blockchain Network Environments | IEEE Journals & Magazine | IEEE Xplore Secure Data Storage and Recovery in Industrial Blockchain Network Environments | IEEE Journals & Magazine | IEEE Xplore [Электронный ресурс]. URL: https://ieeexplore.ieee.org/document/8957108 (дата обращения: 01.11.2023).

101. Selvaganesan M., Liazudeen M.A. An Insight about GlusterFS and Its Enforcement Techniques // 2016 International Conference on Cloud Computing Research and Innovations (ICCCRI). Singapore, Singapore. IEEE, 2016. – C. 120–127. – DOI. 10.1109/ICCCRI.2016.26.

102. Singh N. HADOOP DISTRIBUTED FILE SYSTEM (HDFS) / N. Singh, 2023.

103. Smith K. Implications of value added network services // Data Processing. 1985. – T. 27. – №. 6. – C. 41–45. – DOI. 10.1016/0011-684X(85)90272-2.

104. SONY PICTURES AUDIO LIBRARY | Sony Pictures Entertainment SONY PICTURES AUDIO LIBRARY | Sony Pictures Entertainment [Электронный ресурс]. URL: https://www.sonypictures.com/corp/music_library.html (дата обращения: 01.01.2024).

105. Tarjan R. Depth-first search and linear graph algorithms // 12th Annual Symposium on Switching and Automata Theory (swat 1971). 1971. – C. 114–121. – DOI. 10.1109/SWAT.1971.10.

106. Tarkhanov I. Extension of access control policy in secure role-based workflow model / I. Tarkhanov, 2016. 1 – C. – DOI. 10.1109/ICAICT.2016.7991691.

107. Tian L., Sun Y., Yang L. Overview of Storage Architecture and Strategy of HDFS 2022. – DOI. 10.3233/ATDE220098.

108. Toshiba MG05ACA800x SERIES ENTERPRISE CAPACITY Toshiba MG05ACA800x SERIES ENTERPRISE CAPACITY [Электронный ресурс]. URL: https://toshiba.semicon-storage.com/content/dam/toshiba-ss-v3/master/en/storage/product-archive/eHDD-MG05ACAxxx_product-overview_EOL.pdf (дата обращения: 26.12.2022).

109. Tuler De Oliveira M., Reis L.H.A., Verginadis Y., Mattos D.M.F., Olabarriaga S.D. SmartAccess: Attribute-Based Access Control System for Medical Records Based on Smart Contracts // IEEE Access. 2022. – T. 10. – C. 117836–117854. – DOI. 10.1109/ACCESS.2022.3217201.

110. Wang H., Zhang Q. Research on Blockchain-Based Smart Contract Technology // Smart Computing and Communication. под ред. M. Qiu, Z. Lu, C. Zhang, Cham. Springer Nature Switzerland, 2023. – C. 515–524. – DOI. 10.1007/978-3-031-28124-2_49.

111. Wang X., Su J. Research of Distributed Data Store Based on HDFS // 2013 International Conference on Computational and Information Sciences. Shiyang, China. IEEE, 2013. – C. 1457–1459. – DOI. 10.1109/ICCIS.2013.384.

112. Weil S., Brandt S., Miller E., Long D., Maltzahn C. Ceph: A Scalable, High-Performance Distributed File System. / S. Weil, S. Brandt, E. Miller, D. Long, C. Maltzahn, 2006. 307 – C.

113. WekaFS Architecture White Paper WekaFS Architecture White Paper // WEKA [Электронный ресурс]. URL: https://www.weka.io/wp-content/uploads/files/2017/12/Architectural_WhitePaper-W02R6WP201812-1.pdf (дата обращения: 18.02.2024).

114. Wesselius J., Rooij M. de Permissions and Access Control // Pro Exchange Administration: Understanding On-premises and Hybrid Exchange Deployments. под ред. J. Wesselius, M. de Rooij, Berkeley, CA. Apress, 2023. – С. 677–733. – DOI. 10.1007/978-1-4842-9591-5_10.

115. Winarski T. Merkle Tree Storage of Big Data // 2022.

116. Wu K., Ma Y., Huang G., Liu X. A first look at blockchain-based decentralized applications // Software: Practice and Experience. 2019. – Т. 51. – DOI. 10.1002/spe.2751.

117. Xie J., Li Z., Jin J., Zhang B., Hua Y. Research on Blockchain Storage Extension Based on DHT // Proceedings of the 4th International Conference on Big Data Technologies. New York, NY, USA. Association for Computing Machinery, 2022. – С. 79–85. – DOI. 10.1145/3490322.3490335.

118. Xu J. Case Study: Ceph // Block Trace Analysis and Storage System Optimization: A Practical Approach with MATLAB/Python Tools. под ред. J. Xu, Berkeley, CA. Apress, 2018. – С. 209–227. – DOI. 10.1007/978-1-4842-3928-5_9.

119. Yandex Cloud — надежное облако для вашего бизнеса Yandex Cloud — надежное облако для вашего бизнеса [Электронный ресурс]. URL: https://cloud.yandex.ru/ (дата обращения: 01.01.2024).

120. Yang hao, Yu J., Tang S., Hu Y. Overview of security access control mechanisms / hao Yang, J. Yu, S. Tang, Y. Hu, 2023. 41 – С. – DOI. 10.1117/12.2691877.

121. Yang C.-T., Chen C.-J., Chen T.-Y. Implementation of Ceph Storage with Big Data for Performance Comparison // Information Science and Applications 2017. под ред. K. Kim, N. Joukov, Singapore. Springer, 2017. – С. 625–633. – DOI. 10.1007/978-981-10-4154-9_72.

122. Yang J., Tan M.-S., Ding L. Discretionary Access Control Method to Protect Blockchain Privacy // Cyberspace Data and Intelligence, and Cyber-Living, Syndrome, and Health. под ред. H. Ning, Singapore. Springer, 2019. – С. 161–174. – DOI. 10.1007/978-981-15-1922-2_11.

123. Zhang G., Zheng W., Li K. Rethinking RAID-5 Data Layout for Better Scalability // IEEE Transactions on Computers. 2014. – Т. 63. – №. 11. – С. 2816–2828. – DOI. 10.1109/TC.2013.143.

124. Zhang L., Lee B., Ye Y., Qiao Y. Ethereum Transaction Performance Evaluation Using Test-Nets // Euro-Par 2019: Parallel Processing Workshops. под ред. U. Schwardmann [и др.]., Cham. Springer International Publishing, 2020. – С. 179–190. – DOI. 10.1007/978-3-030-48340-1_14.

125. Zhang W., Anand T. Ethereum Architecture and Overview // Blockchain and Ethereum Smart Contract Solution Development: Dapp Programming with Solidity. под ред. W. Zhang, T. Anand, Berkeley, CA. Apress, 2022. – С. 209–244. – DOI. 10.1007/978-1-4842-8164-2_6.

126. Zhang Y., Kasahara S., Shen Y., Jiang X., Wan J. Smart Contract-Based Access Control for the Internet of Things // IEEE Internet of Things Journal. 2019. – Т. 6. – №. 2. – С. 1594–1605. – DOI. 10.1109/JIOT.2018.2847705.

Основные положения диссертационной работы изложены в следующих опубликованных работах

В перечне, рекомендованном ВАК Минобрнауки России

127. Хаммуд О., Тарханов И. DAC модель распределенного управления доступом на основе блокчейна // Системы высокой доступности. 2024. – DOI. 10.18127/j20729472-202401-05.

128. Хаммуд О., Тарханов И.А. Анализ Надежности Распределенной Системы Хранения Файлов на Основе Блокчейн // Труды Института Системного Анализа Российской Академии Наук. 2023. – DOI. 10.14357/20790279230402.

129. Хаммуд О., Тарханов И.А. Методология Хранения Электронных Документов в Децентрализованных Блокчейн Приложениях (Dapps) // Труды Института Системного Анализа Российской Академии Наук. 2021. – DOI. 10.14357/20790279210205.

В других изданиях:

130. Hammoud O., Tarkhanov I. A method of data synchronization with Ethereum blockchain // Artificial societies. 2020. – Т. 15. – №. 3. – С. 9. – DOI. 10.18254/S207751800010671-7.

131. Hammoud O., Tarkhanov I.A. DecStore: a Blockchain-based File Storage System With Autoscaling // 2023 24th International Arab Conference on Information Technology (ACIT). Ajman, United Arab Emirates. IEEE, 2023. – С. 1–6. – DOI. 10.1109/ACIT58888.2023.10453678.

132. Hammoud O., Tarkhanov I.A. Estimating the Reliability of a DApps-Based Files Storage System // 2023 International Russian Automation Conference (RusAutoCon). 2023. – С. 82–87. – DOI. 10.1109/RusAutoCon58002.2023.10272732.

133. Hammoud O., Tarkhanov I.A. A Novel Blockchain-Integrated Distributed Data Storage Model with Built-in Load Balancing // 2022 IEEE 16th International Conference on Application of Information and Communication Technologies (AICT). 2022. – С. 1–6. – DOI. 10.1109/AICT55583.2022.10013548.

134. Hammoud O. Review of metadata storing and searching methods in DLT and distributed files systems // Law & Digital Technologies. 2022. – Т. 2. – №. 1. – С. 35. – DOI. 10.18254/S278229070018060-2.

135. Hammoud O., Tarkhanov I., Kosmarski A. An Architecture for Distributed Electronic Documents Storage in Decentralized Blockchain B2B Applications // Computers. 2021. – Т. 10. – №. 11. – С. 142. – DOI. 10.3390/computers10110142.

136. Hammoud O. Modeling the Reliability of Distributed Data Storage Systems // Artificial societies. 2024. – Т. 19. – №. 2. – С. 0. – DOI. 10.18254/S207751800031356-0.

ПРИЛОЖЕНИЕ А

(обязательное)

РОССИЙСКАЯ ФЕДЕРАЦИЯ

RU2023682504

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

ГОСУДАРСТВЕННАЯ РЕГИСТРАЦИЯ ПРОГРАММЫ ДЛЯ ЭВМ

Номер регистрации (свидетельства): **2023682504** Дата регистрации: **26.10.2023** Номер и дата поступления заявки: **2023665313 18.07.2023** Дата публикации и номер бюллетеня: **26.10.2023 Бюл. № 11** Контактные реквизиты: **+79265262371 ivanzerg@mail.ru**	Автор(ы): **Тарханов Иван Александрович (RU),** **Хаммуд Обадах (SY)** Правообладатель(и): **Тарханов Иван Александрович (RU),** **Хаммуд Обадах (SY)**

Название программы для ЭВМ:
DecStore: балансировщик нагрузки на базе смарт-контракта для распределённого хранения файлов

Реферат:
Балансировщик нагрузки реализован на chaincode Hyperledger Fabric и решает задачи управления узлами сети, процессами присоединения и выхода узлов из системы, а также задачи управления размещением существующих файлов, определение места добавления новых файлов. Модель хранения DecStore использует виртуальные кластеры и виртуальные диски. Балансировщик нагрузки отвечает за управление этими виртуальными объектами путем их распределения, управления расположением новых виртуальных дисков. Балансировщик нагрузки обрабатывает ситуации распространения и восстановление файлов, а также операции, связанные с узлами, с использованием 4-х алгоритмов. Выполняется в среде блокчейн платформы Hyperledger Fabric. ОС: Ubuntu Linux 16.04.

Язык программирования: Go

Объем программы для ЭВМ: 40 КБ

ПРИЛОЖЕНИЕ В

(обязательное)

РИИТ
РУССКИЕ ИННОВАЦИОННЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

ОБЩЕСТВО
С ОГРАНИЧЕННОЙ
ОТВЕТСТВЕННОСТЬЮ
«РИИТ»

127005 г. Москва, ул. Сущевская д.21
e-mail: info@riit-tech.ru
www.riit-tech.ru

ФГБОУ ВО НИТУ«МИСиС»

119049, Москва, Ленинский пр-т, 4

«5» августа 2024 г. № И/3/И.

Справка о внедрении результатов диссертационной работы

Настоящей справкой подтверждается, что аспирант НИТУ МИСиС Хаммуд Обадах разработал прототип распределенной системы хранения файлов с использованием технологии блокчейн в период с 06.2023 по 05.2024 г.

Данный прототип успешно интегрирован в программное обеспечение документированного обмена файлами Рубикон.ДОФ (реестровая запись в реестре Российского ПО №5572 от 24.06.2019), что позволит создавать информационные системы для участников с нулевым доверием друг к другу, при этом экономить до 25% дискового пространства по сравнению с системами полного резервного копирования.

Аспирант принимал непосредственное участие в применении данной разработки и проведении нагрузочного тестирования на 1000 пользователей.

Генеральный директор Л.М. Арлазаров