Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

Метод хранения данных с возможностью доказательства неправомерного доступа.

Студент группы **ИУ7-83Б**	_____	**П. Г. Пересторонин**
	(Подпись, дата)	(И.О. Фамилия)
Руководитель ВКР	_____	**А. С. Григорьев**
	(Подпись, дата)	(И.О. Фамилия)
Нормоконтролер	_____	_____
	(Подпись, дата)	(И.О. Фамилия)

***2022** г.*

РЕФЕРАТ

Расчетно-пояснительная записка 38 с., 6 рис., 1 табл., X ист., X прил.

КЛЮЧЕВЫЕ СЛОВА

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ВВЕДЕНИЕ

Обеспечение защиты от неправомерного доступа — мера по защите информации. При работе с информацией лицо, имеющее доступ к этой информации, ограничено правами, которые определяют способы взаимодействия с информацией. В качестве примера таких прав можно привести права на доступ, копирование, уничтожение, изменение, блокирование, распространения и предоставления информации.

Задача по обеспечению защиты от неправомерного доступа может решаться на нескольких уровнях:

— Непосредственная защита от неправомерного доступа. Самый желаемый уровень, при котором лицо не имеет возможности превышения прав при работе с информацией. Для чтения, например, может достигаться за счет шифрования данных на диске, где ключ шифрования имеется только у людей, обладающими правами на чтение.

— Возможность устранения последствий неправомерного доступа. Данный уровень может быть использован совместно с уровнем выше, обеспечивая дополнительную защиту. Примерами методов защиты, обеспечивающими описанную возможность, могут послужить резервное копирование и репликация данных.

— Возможность доказательства неправомерного доступа. Данный уровень служит для ответа на вопрос, являются ли данными целостностными. Для обеспечения такой возможности может использоваться, например, расчет контрольных сумм.

Цель работы — разработать метод хранения информации с возможностью доказательства неправомерного доступа на основе движка MergeTree[2] в СУБД ClickHouse[1].

Для достижения поставленной цели требуется решить следующие задачи:

— рассмотреть принципы работы существующих систем хранения данных;

— рассмотреть общую архитектуру СУБД ClickHouse;
— рассмотреть архитектуру движка MergeTree СУБД ClickHouse;
— разработать метод хранения информации с возможностью доказательства несанкционированного доступа;
— провести исследование метода на предмет способов и ограничений применения метода.

1 Аналитическая часть

В данном разделе будут проанализированы существующие решения, а также рассмотрена архитектура СУБД ClickHouse и движка MergeTree.

1.1 Базовые понятия

В данном подразделе будут рассмотрены базовые понятия, необходимые для понимания устройства существующих решений.

1.1.1 Хэш-функция

Хэш-функция[3] — функция, осуществляющая преобразование массива входных данных произвольной длины в выходную битовую строку установленной длины, выполняемое определённым алгоритмом.

Хэш-функции применяются:

— при решении задачи дедубликации;

— при построении идентификаторов;

— при вычислении контрольных сумм;

— при хранении паролей.

Криптографическая стойкость [4] — способность криптографического алгоритма противостоять криптоанализу. Стойким считается алгоритм, успешная атака на который требует от атакующего обладания недостижимым на практике объёмом вычислительных ресурсов либо настолько значительных затрат времени на раскрытие, что к его моменту защищённая информация утратит свою актуальность.

При рассмотрении хэш-функций под алгоритмом подразумевается процесс вычисления значения хэш-функции, а под атакой на алгоритм — решение обратной задачи: нахождение для заданного значения хэш-функции *H1* такого массива входных данных *A1*, что *f(A1) = H1*. Хэш-функцию, которая является стойкой по определению криптографической стойкости по отношению к такой задаче, называют криптостойкой.

Криптостойкие функции также обладают следующим свойством: при на-

личии массива входных данных *A1* и значения хэш-функции для него *f(A1)*, настолько же сложной задачей, что и нахождения обратного значения, является задача нахождения такого отличного от *A1* значения массива входных данных *A2*, для которого верно *f(A1) = f(A2)*. Такие значения *A1* и *A2*, для которых верно равенство *f(A1) = f(A2)*, называются коллизиями.

1.1.2 Блокчейн

Блокчейн[5] — выстроенная по определённым правилам непрерывная последовательная цепочка блоков — элементов, содержащих информацию. В общем случае такая цепочка поддерживает 2 операции:

1) добавление нового элемента в конец цепочки;
2) проверка целостности всей цепочки.

При добавлении блока вычисляется значения хэша от его содержимого и хэша предыдущего блока. Вычисленное значение считается хэшом добавляемого блока.

Пример блокчейна изображен на рисунке 1.

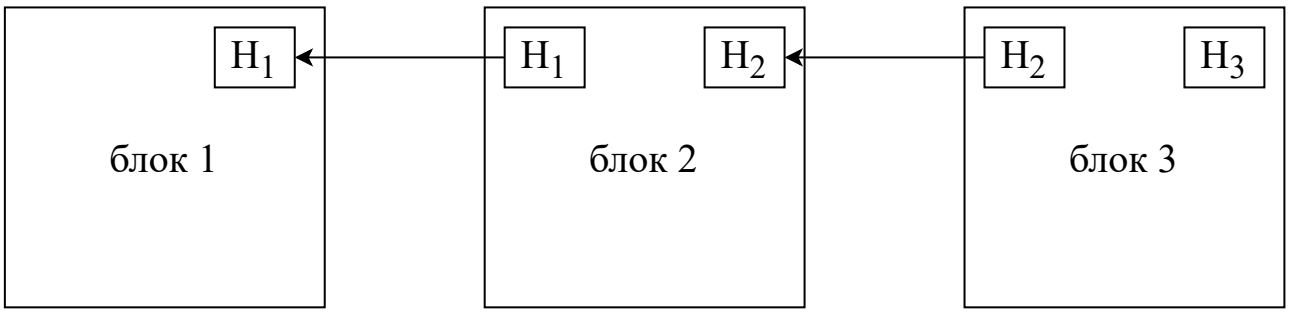

Рисунок 1 – Пример блокчейна

При проверки целостности цепочки выполняются следующие действия:

1) Высчитывается хэш от содержимого первого блока и сравнивается со значением хэша, записанным при добавлении данного блока в цепочку. Если значения не совпадают, то констатируется факт наружения целостности.
2) Для очередного блока вычисляется значение хэша от его содержимого и хэша предыдущего блока. Вычисленное значение сравнивается со значением хэша, записанным при добавлении блока. При несовпадении кон-

статируется факт наружения целостности.

При изменении последнего блока достаточно пересчитать и перезаписать значения хэша только для него. Однако при изменении хэша блока, не являющегося последним, потребуется пересчитать значение хэша всех последующих блоков, что в некоторых системах может потребовать больших вычислительных затрат.

1.1.3 Дерево и ориентированный ациклический граф Меркла

Дерево Меркла[6] — двоичное дерево, в листовые вершины которого помещены хеши от блоков данных, а внутренние вершины содержат хеши от сложения значений в дочерних вершинах. В общем случае операция сложения — произвольная функция от 2 аргументов *f(x, y)*, которая может быть несимметричной (например, конкатенация строк). Корневой узел дерева содержит хеш от всего набора данных, то есть такое дерево является однонаправленной хеш-функцией.

Пример дерева меркла можно увидеть на рисунке 3.

Применения дерева Меркла:

— Хранение транзакций в блокчейне криптовалют (например, в Bitcoin; позволяет получить значение хэша всех транзакций в блоке, а также эффективно верифицировать транзакции).

— Для множества значений в среде с ограничением по памяти. В среде хранится только корень дерева. Для проверки принадлежности элемента множеству вместе с элементом требуется передать доказательство Меркла — значения всех хэшей, с которыми суммируется хэш проверяемого элемента на пути из листа в корень.

Ориентированный ациклический граф Меркла[7] — структура данных, представляющая собой граф, строящийся по следующим правилам:

1) Все вершины графа, степень полуисхода[8] которых равна 0, представляют хэши данных.
2) Вычисление значений остальных вершин графа делается в порядке, об-

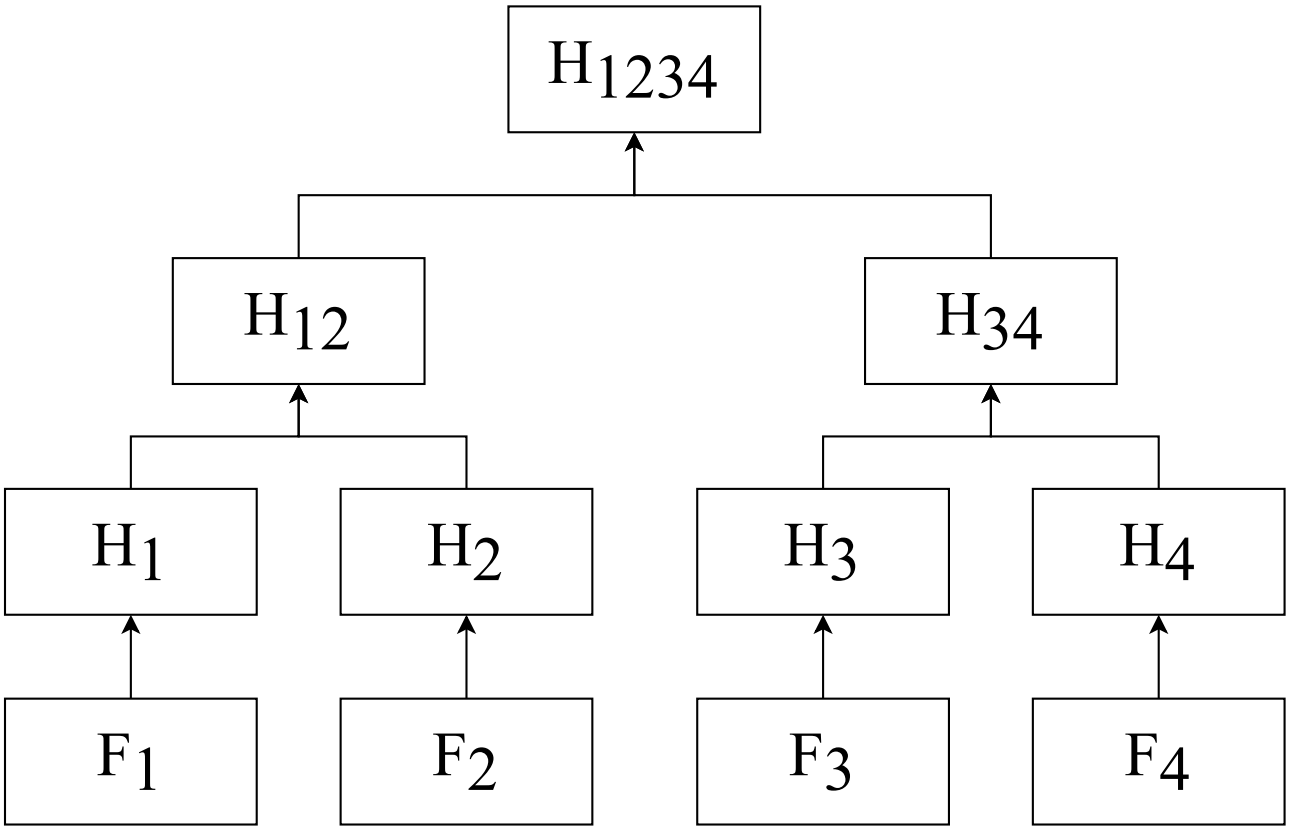

Рисунок 2 – Пример дерева Меркла

ратном топологической сортировке[9]. Хэшом очередной вершины будет значения хэша от суммы вершин, в которые есть дуга из рассматриваемой вершины.

Данная структура данных используется в системе версионного контроля Git[10].

1.2 Существующие решения

В данном подразделе будут описаны существующие системы, решающих задачу обеспечения защиты от неправомерного доступа.

1.2.1 PASIS

PASIS[11] — распределенное хранилище данных, реализующее метод хранения, защищающий от ошибок, связанных с подменой данных определенными узлами системы, называемыми византийскими. Узлы хранилища хранят данные фрагментами и версионируют информацию. Клиенты читают и пишут фрагменты данных.

Клиент пишет данные по фрагментам на N узлов хранилища. При чтении данных клиент обращается к m узлам, которые являются подмножеством узлов, на которые происходила запись, и проверяет валидность полученных фрагментов. Процесс чтения может быть выполнен в несколько итераций, если в результате валидации будет выявлена ошибка. Процесс чтения считается завершенным, если получено m верных фрагментов. Примечательно то, что каждый отдельный фрагмент не несет в себе информации, исходные данные можно восстановить только по m и большему числу фрагментам.

Данные восстанавливаются из фрагментов с помощью стирающих кодов[12] и проверяются с помощью кросс чексумм. Кросс чексумма — конкатенация хэшей всех N записываемых фрагментов. После восстановления данных в наличии у читателя имеется все N фрагментов, для которых можно рассчитать хэш-сумму и проверить подлинность данных.

Таким образом использование данной системы позволяет защититься от неправомерного удаления данных с части узлов хранилища с возможностью восстановления данных, а также защититься от неправомерного изменения данных.

1.2.2 Криптографические файловые системы

TCFS[13], NCryptFS[14] — примеры криптографических файловых систем. В отличие от обычных файловых систем, данные системы имеют дополнительный слой между виртуальной файловой системой и драйвером файловой системы, который шифрует данные. При такой реализации работа слоя шифрования данных будет прозрачна для пользователя, потому что пользователь работает с виртуальной файловой системой.

TCFS использует стандартные для файловых систем способы подключения в ОС `Linux`: системные вызовы `mount`, `ioctl` [15]. При работе в пределах смонтированной файловой системы все операции записи и чтения проходят через слой шифрования, поэтому данные записываются на устройство только в зашифрованном виде.

NCryptFS обладает похожим принципом работы, однако подключается с помощью собственной команды `attach`, не используя стандартные методы, и имеет возможность создания пространства имен для управления правами групп пользователей на доступ к определенным директориям, а также содержит средства авторизации.

Описанные системы решают задачу неправомерного чтения. При случайной записи данных на устройство будет нарушена их целостность, проверка целостности возлагается на приложения, использующие данные файловые системы, но так как данные хранятся в зашифрованном виде, целостность исходных данных может быть нарушена более значительно. Возможности восстановления данных нет.

1.2.3 OceanStore

OceanStore[16] — безопасная распределенная read-only файловая система.

Данное хранилище хранит данные в одноранговой сети, где узлы соединены между собой по принципу точка-точка. Преимуществом данного хранилища является хранение данных по фрагментам с использованием стирающих кодов. Данный подход, как уже было показано в 1.2.1, позволяет получать исходные данные, записанные как N фрагментов, из любых m фрагментов, где $m < N$.

В данном хранилище используется понятие самопроверяющихся данных[17]: данные адресуются по идентификатору GUID, который является хэшом от содержимого. Данный хэш считается из дерева Меркла, изображенного на рисунке **??**. Каждый фрагмент помимо самих данных содержит в себе также доказательство Меркла. При восстановлении данных проверяется каждый фрагмент по отдельности (с помощью доказательства Меркла), а также все данные целиком. Порядок действий можно описать следующим образом:

1) Получить m фрагментов, проверив у каждого доказательство Меркла.
2) Восстановить N фрагментов с использованием стирающих кодов.
3) Восстановить данные из N фрагментов.

4) Построить дерево Меркла и убедиться в равенстве корня дерева и GUID.

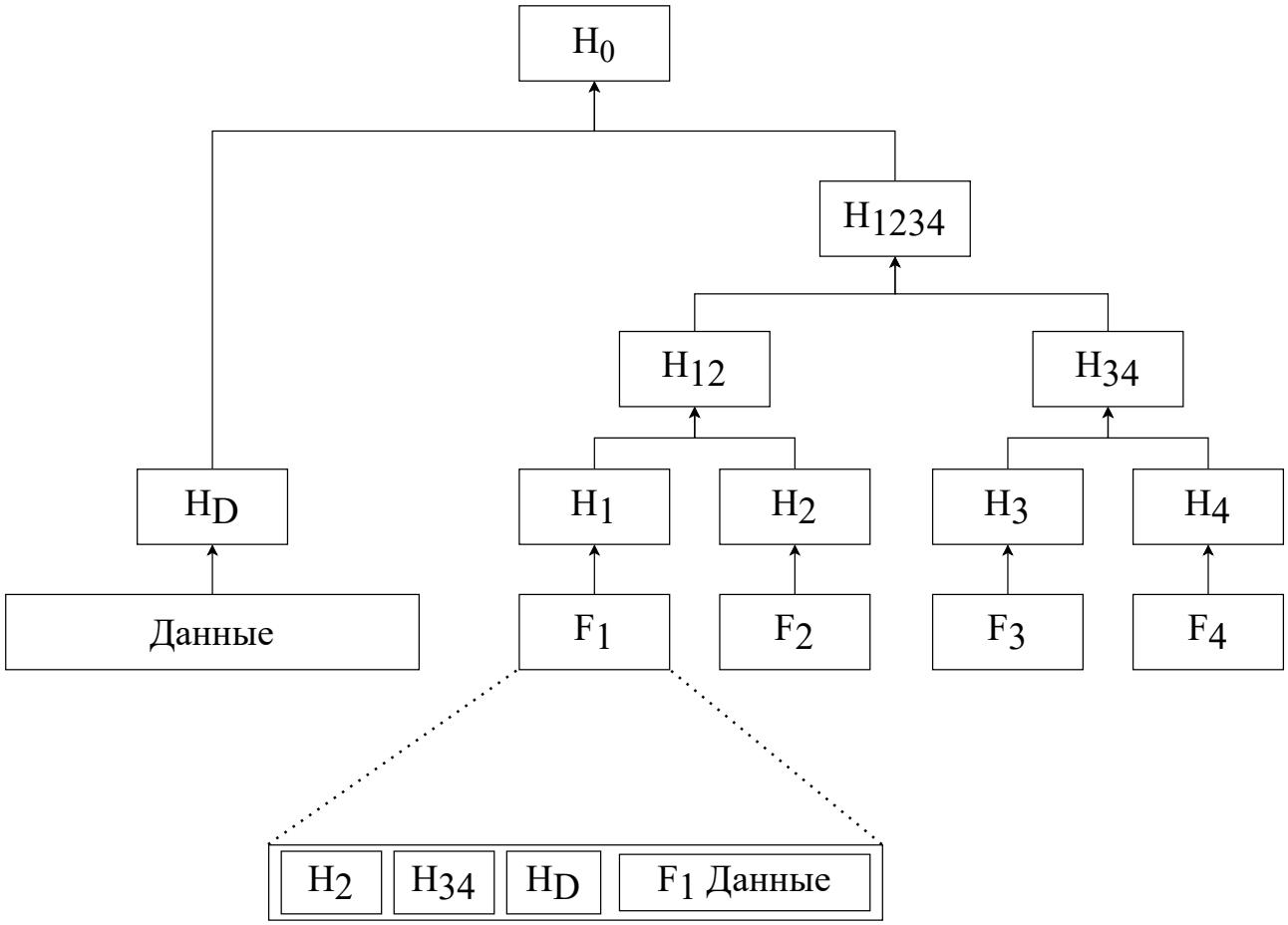

Рисунок 3 – Пример дерева Меркла и фрагментов для OceanStore.

Данная система считается read-only, потому что изменение данных без изменения идентификатора невозможно, так как идентификатор создается на базе содержимого. Таким образом, например, для обновления версии каких-то данных, старая версия остается неизменной, при этом появляется новая версия с новы содержимым и GUID.

Данное хранилище, несмотря на внутренее устройство, поддерживает возможность иерархической структуры данных. Для этого требуется в некоторых объектах, которые не являются листьями дерева иерархии данных, хранить хэши дочерних вершин. GUID корня иерархической структуры известен.

На примере иерархической структуры видна сложность изменения данных: изменение элемента дерева приводит к необходимости изменения (замены на новый элемент) всех вершин-предков.

Также как и хранилище из раздела 1.2.1, данная система защищена от частичного удаления данных, обладает возможностью восстановления, а также защищена от неправомерного изменения данных.

1.2.4 Git

Git[10] — система контроля версий. Данная система не решает задачу защиты от неправомерного доступа непосредественно, однако делает это косвенно: при нарушении целостности данных система перестанет работать, что будет являться доказательством неправомерного доступа. Внутреннее представление Git — хранилище, адресующее по содержанию. Это означает, что файлы, которыми оперирует Git в рамках системной файловой системы, хранятся в виде ассоциативного массива, доступ в котором осуществляется по ключу. Данная система называется Object Store. В качестве ключа файла выступает хэш от его содержимого, подобно GUID в 1.2.3.

Иерархическую структуру файлов Git поддерживает в виде ориентированного ациклического графа Меркла за счет наличия 3 основных типов объектов:

1) blob — содержит информацию о конкретной версии файла. В ациклическом графе Меркла файлы этого типа всегда обладают нулевой степенью полуисхода, то есть их хэш составлен на основе содержимого файла. Этим же хэшом данные файлы адресуются в упомянутом выше ассоциативном массиве.
2) tree — содержит информацию о конкретной версии директории. Данный объект ”содержит в себе”объекты (больше 0), которые могут быть типа blob или tree. Он обладает отличной от обычной функцией суммирования: вместо простой конкатенации хэшей вершин, в которые ведут дуги, в функции суммирования также фигурируют имена файлов дочерних вершин (реальные имена, не хэши), а также режимы доступа к данным файлам.
3) commit — содержит информацию о коммите — фиксированном состоя-

нии системы. commit обладает единичной степенью полуисхода и единственная его дуга ведет в объект типа tree, соотвествующей корневой директории репозитория. Объект типа commit хранит информацию об авторе изменений, связанных с фиксацией, об имени, хэше и режиме доступа корневой директории и **информацию о хэше родительского коммита** — то есть предыдущей зафиксированной версии. Адресуется коммит значением хэш функции от его содержимого. Схема, по которой описанный коммит связан с предыдущим в точности соответствует схеме работы с блокчейном.

На рисунке 4 схематично представлен пример графа для 3 основных типов объектов.

Изменение внутренней структуры в процессе изменения данных происходит:

1) Изначально Git не будет учитывать изменения в рабочей директории, то есть при изменении файла у системы будет иметься копия файла из рабочей директории, которая не будет изменена.
2) Чтобы система учла изменения, требуется добавить измененные файлы в индекс. Данное действие приведет к изменению с точки зрения внешнего интерфейса, с точки зрения внутренней системы это приведет к созданию нового объекта типа blob в Object Store, а также виртуального объекта типа tree (данный объект не будет виден в Object Store на этом этапе). Возможное состояние системы на этом этапе изображено на рисунке 5.
3) При фиксации изменений виртуальный объект типа tree, соответствующий корню рабочей директории для новых версий файлов, добавленных в индекс, становится реальным, то есть создается объект в Object Store, а также создается объект типа commit, который ссылается (содержит хэш в качестве значения) на вновь созданный объект типа tree. Возможное состояние после всех проделанных действий изображено на рисунке 6.

Неправомерный доступ подразумевает возможность доступа на запись к

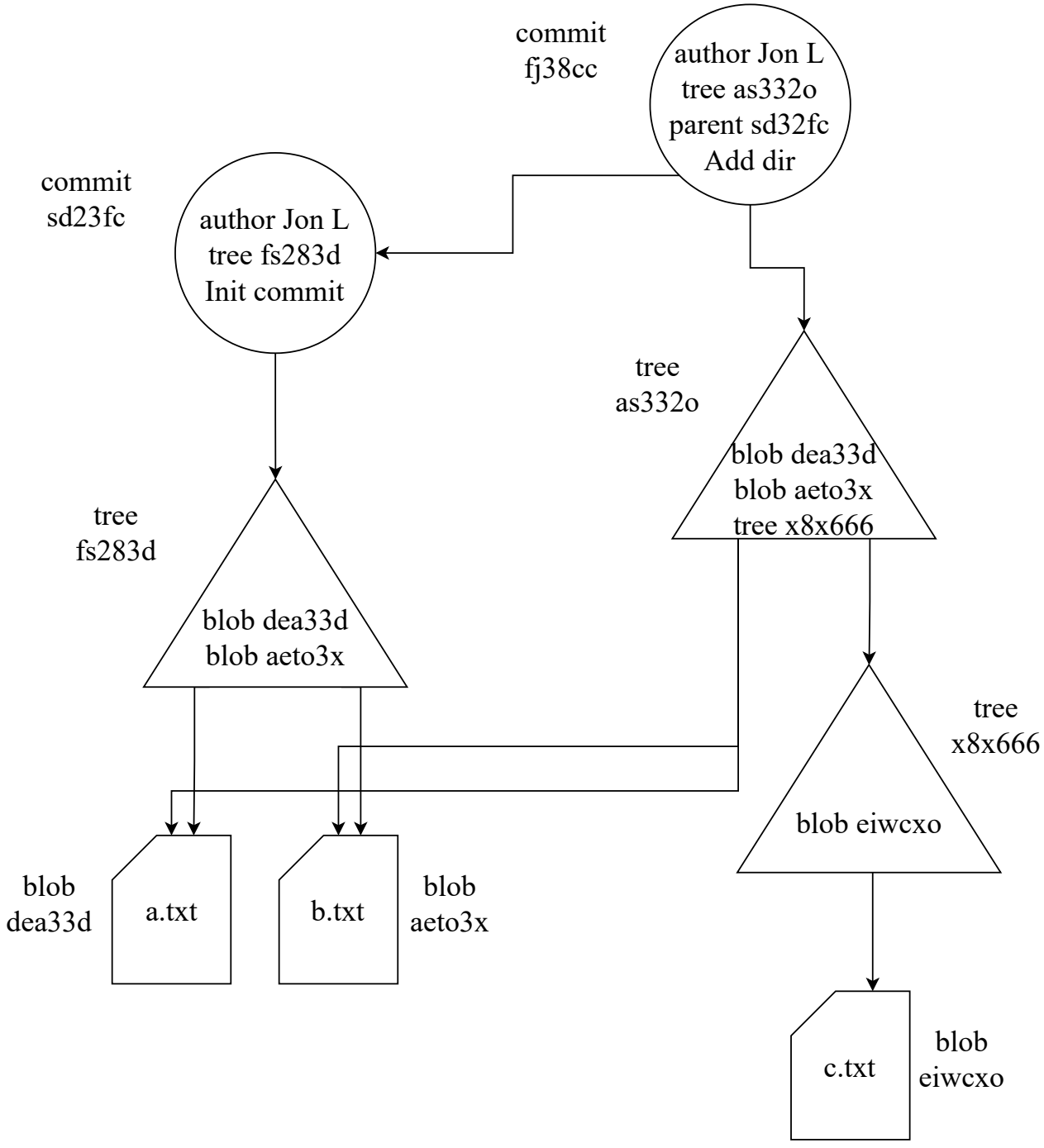

Рисунок 4 – Пример графа Меркла для 3 основных типов объектов

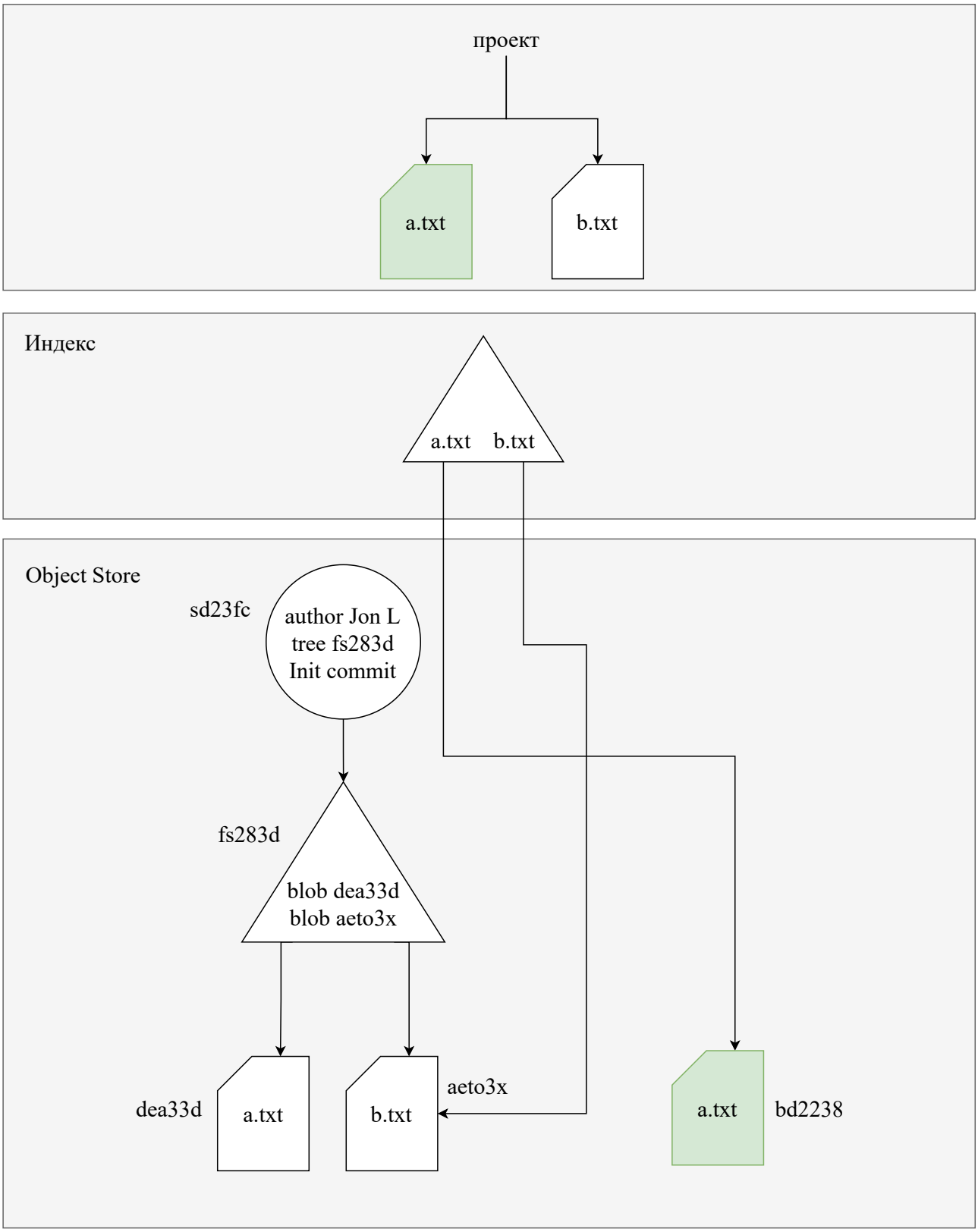

Рисунок 5 – Возможное состояние системы контроля версий Git при добавлении нового файла в индекс.

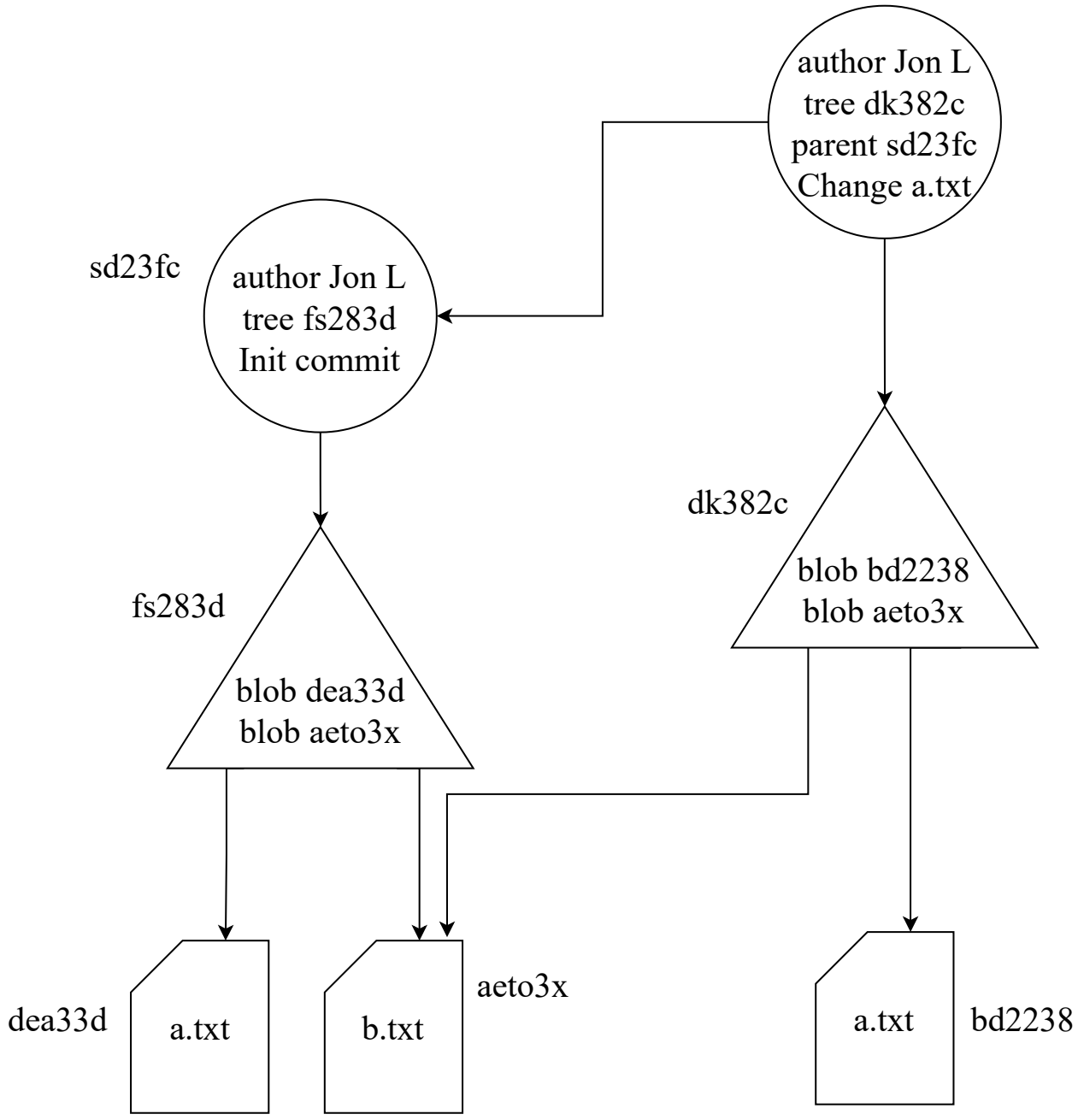

Рисунок 6 – Возможное состояние системы контроля версий Git после фиксирования изменений.

внутренней структуре, изменения в которой происходят исключительно засчет добавления новых объектов.

Возможно 3 варианта:

1) Изменение данных в рамках объекта. Имя файла в системе Git — его хэш и любое изменение содержимого приведет к несовпадению рассчитываемого хэша и имени файла, что приведет к ошибке и обнаружению нарушения в данных.
2) Удаление объекта. Такая атака будет обнаружена в момент, когда произойдет попытка обращения к удаленному файлу. Он не будет найден в Object Store и будет ошибка.
3) Добавление объекта. Добавление объекта не будет обнаружено, однако и данные, добавленные с новым объектом не будут использованы ввиду того, что никакой другой объект не будет ссылаться на вновь созданный. Если добавить коммит в конце цепочки, то появляется возможность посчитать хэш вместе с хэшем предыдущего commit объекта, однако подобный формат работы похож на стандартный и может быть осуществлен извне.

1.2.5 Bitcoin

Bitcoin[5] — одноранговая децентрализованная электронная денежная система.

Для данной системы недопустимы сценарии, при которых данные теряются и приводится только доказательство неправомерного доступа к данным. Ввиду того, что данная система является распределенной, у нее нет единого состояния, что усложняет получение неправомерного доступа к данным и их изменение или повреждение без возможности восстановления.

Транзакция — операция передачи биткойнов с одного адреса на другой. Транзакция осуществляется следующим образом:

1) Владелец биткойн-адреса (который связан с публичным ключом) использует любую совершенную транзакцию, в которой адресом назначения является данный адрес.
2) Создается новая транзакция, которая содержит информацию о наборе адресов и соответсвующем количестве биткоинов, которые требуется перевести в рамках транзакции.
3) Данная транзакция подписывается приватным ключом владельца кошелька, с которого происходит передача биткоинов.

Таким образом невозможность неправомерного изменения транзакции обеспечивается за счет PKI[18].

Транзакция является составным элементом блока. Блок — составленный по определенным правилам набор транзакций.

Блок имеет определенную структуру, в которую входят:

— магическое число;
— размер блока;
— заголовок блока;
— счетчик транзакций;
— набор транзакций.

Заголовок блока состоит из следующих полей:

— версия блока;
— хэш заголовка предыдущего блока;
— хэш корня дерева Меркла, составленного из транзакций, входящих в данный блок;
— текущее время;
— целевое значение хэша, ниже которого должен быть хэш от заголовка;
— число *Nonce*, которого подбирается таким образом, чтобы хэш от заголовка был меньше целевого значения.

Блоки образуют блокчейн — в заголовке очередного блока содержится хэш предыдущего блока, с помощью которого блоки собираются в цепь. Из одного блока может исходить несколько других блоков, которые содержат первый, как предшествующий. Действующей цепочкой считается наиболее длинная. В этом смысл концепции, называемой proof-of-work: большинство пользователей берут наиболее длинную цепочку блоков и коллективно пытаются подобрать требуемое значение *Nonce*, которое бы удовлетворяло условию целевого значения, тем самым концентруя вычислительные мощности и не давая другим цепочкам стать длиннее текущей. Наиболее быстро будет рассчитываться та цепочка, для расчета которой используется наибольшее количество мощностей.

Сложность расчета значений *Nonce* контролируется за счет изменения целевого значения хэша. Можно дать вероятностную оценку сложности вычисления нужного значения хэша в количестве попыток, исходя из следующих условий:

— значение хэш функции является случайным значением;
— число нулей в бинарном представлении целевого значения равняется числу M.

Тогда вероятность появления такого значения хэша, которое будет меньше целевого значения, равна вероятности выпадения M первых нулей при расчете

хэша и рассчитывается по формуле 1.

$$P = \frac{1}{2^M} \tag{1}$$

С учетом независимости расчетов для различных $Nonce$ и формулы 1, можно рассчитать количество попыток, требуемых совершить в среднем для получения одного удовлетворяющего числа по формуле 2.

$$N = \frac{1}{\frac{1}{2^M}} = 2^M \tag{2}$$

Наиболее известный способ атаки на системы, работающие по концепции proof-of-work — атака 51%. Суть атаки заключается в том, что атакующий владеет большинством вычислительных мощностей и делает следующие действия:

1) Сделать блок, который будет содержать некоторую транзакцию T_1.
2) Дождаться, пока система, на счет которой мы перевели биткоины, будет считать транзакцию T_1 завершившейся.
3) Взять блок, предшествующий блоку из пункта 1, и начать делать новую цепочку без транзакции T_1 до тех пор, пока она не станет действующей.

Такая атака позволяет неправомерно изменить данные.

1.2.6 Выводы

Сравнение рассмотренных методов хранения данных с возможностью защиты от неправомерного доступа можно увидеть в таблице **??**. Обозначения:

— НД — неправомерное действие;
— КФС — криптографические файловые системы;
— У — удаление;
— И — изменение;
— ЧУ — частичное удаление;
— ЧИ — частичное изменение.

Полученные результаты можно свести к следующему:

— Защита от чтения возможна в системе, в которой определенная группа людей обладает некоторым уникальным знанием (ключ в КФС).
— Восстановление данных после частичного изменения или удаления возможно в распределенных системах, использующих стирающий код, в связи с тем, что фрагменты хранятся в разных местах и допускается неправомерный доступ в части мест.
— Исключает возможность полного неправомерного изменения только Bitcoin, потому что во всех рассмотренных системах количество узлов, в которых хранятся данные, конечно и задано наперед (хоть и в теории может быть очень велико), в то время как в Bitcoin все узлы системы содержат одно состояние.
— Возможностью доказательства неправомерного изменения обладают все системы кроме КФС, используя для этого контрольные суммы и хэши. Однако также как в случае удаления, в локальной системе без шифрования нарушитель может восстановить целостность цепочки блоков при изменении внутреннего состояния Git.
— Возможностью доказательства неправомерного удаления обладают 2 системы, в основе которых лежит блокчейн. Однако если данные не шифруются никаким образом, то в случае с Git человек, нарушающий права, способен повторить действия самой системы, и произвести изменения, поддержав целостность цепочки блоков.

Таблица 1 – Методы хранения и обеспечиваемая защита.

НД	**PASIS**	**КФС**	**OceanStore**	**Git**	**Bitcoin**
чтение (исключение)	-	+	-	-	-
ЧУ или ЧИ (восстановление)	+	-	+	-	-
изменение (исключение)	-	-	-	-	+
изменение (доказательство)	+	-	+	+/-	+
удаление (доказательство)	-	-	-	+/-	+

1.3 Архитектура СУБД ClickHouse

В данном разделе будет рассмотрена архитектура СУБД ClickHouse.

1.3.1 Основные концепции

ClickHouse — полноценная колоночная СУБД. Данные хранятся в колонках, а в процессе обработки - в массивах. По возможности операции выполняются на массивах, а не на индивидуальных значениях. Это называется «векторизованное выполнения запросов», и помогает снизить стоимость фактической обработки данных.

Для представления фрагментов столбцов в памяти используется интерфейс `IColumn`. Интерфейс предоставляет вспомогательные методы для реализации различных реляционных операторов. Почти все операции иммутабельные: они не изменяют оригинальных колонок, а создают новую с измененными значениями.

Различные реализации `IColumn` (например, `ColumnUInt8` или `ColumnString`) отвечают за распределение данных колонки в памяти. Для колонок целочисленного типа это один смежный массив, такой как `std::vector`. Для колонок типа String и Array это два вектора: один для всех элементов массивов, располагающихся смежно, второй для хранения смещения до начала каждого массива. Также существует реализация `ColumnConst`, которая несмотря на то, что хранит только одно значение в памяти, обладает поведением фрагмента колонки.

Можно работать и с индивидуальными значениями. Для представления индивидуальных значений используется Поле (`Field`). `Field` — размеченное объединение `UInt64`, `Int64`, `Float64`, `String` и `Array`. `Field` не несет в себе достаточно информации о конкретном типе данных в таблице (например, `UInt8`, `UInt16`, `UInt32` и `UInt64` в `Field` представлены как `UInt64`).

Различные функции на колонках могут быть реализованы обобщенным, неэффективным путем, используя `IColumn` методы для извлечения значений `Field`, или специальным путем, используя знания о внутреннем распределе-

ние данных в памяти в конкретной реализации IColumn. Для этого функции приводят IColumn к конкретному типу и работают напрямую с его внутренним представлением.

IDataType отвечает за сериализацию и десериализацию — чтение и запись фрагментов колонок или индивидуальных значений в бинарном или текстовом формате. IDataType точно соответствует типам данных в таблицах.

IDataType и IColumn слабо связаны друг с другом. Различные типы данных могут быть представлены в памяти с помощью одной реализации IColumn (например, и DataTypeUInt32, и DataTypeDateTime в памяти представлены как ColumnUInt32 или ColumnConstUInt32).

IDataType хранит только метаданные (например, DataTypeFixedString хранит только N — фиксированный размер строки).

Block это контейнер, который представляет фрагмент таблицы в памяти. Это набор троек — (IColumn, IDataType,). В процессе выполнения запроса, данные обрабатываются блоками. Если имеется Block, значит имеются данные (в объекте IColumn), информация о типе (в IDataType) и имя колонки. Блоки создаются для всех обработанных фрагментов данных.

1.3.2 Движок MergeTree

Основная идея движка MergeTree заключается в том, что данные записываются кусками в своем изначальном виде, а затем данные куски объединяются в фоновом режиме. Данных подход используется для исключения большого количества перезаписи данных для сохранения требуемого порядка элементов.

Таблица состоит из кусков данных, отсортированных по первичному ключу. При вставке в таблицу создаются отдельные куски данных, каждый из которых лексикографически отсортирован по первичному ключу.

Данные, относящиеся к разным партициям, разбиваются на разные куски. В фоновом режиме ClickHouse выполняет слияния (merge) кусков данных для более эффективного хранения. Куски, относящиеся к разным партициям не объединяются. Механизм слияния не гарантирует, что все строки с одинаковым

первичным ключом окажутся в одном куске.

Каждый кусок данных логически делится на гранулы. Гранула — минимальный неделимый набор данных, который ClickHouse считывает при выборке данных. ClickHouse не разбивает строки и значения и гранула всегда содержит целое число строк. Первая строка гранулы помечается значением первичного ключа для этой строки (засечка). Для каждого куска данных ClickHouse создаёт файл с засечками (индексный файл). Для каждого столбца, независимо от того, входит он в первичный ключ или нет, ClickHouse также сохраняет эти же засечки. Засечки используются для поиска данных напрямую в файлах столбцов.

Размер гранул оганичен настройками движка по количеству и байтам. Количество строк в грануле лежит в диапазоне не больше максимально заданного значения, в зависимости от размера строк. Размер гранулы может превышать максимально заданный размер в байтах в том случае, когда размер единственной строки в грануле превышает значение настройки. В этом случае, размер гранулы равен размеру строки.

TODO: написать подробнее про движок

1.3.3 Специфика выполнения операций над данными в MergeTree

Здесь надо написать про альтеры и про мержи (слияния).

1.4 Метод защиты от неправомерного доступа в движке MergeTree

Здесь надо описать свой метод в 2 моментах

1.4.1 Хранимые данные и связывание блоков

1.4.2 Пересчет хэшей во время изменений и слияний

2 Конструкторская часть

2.1 Алгоритм добавления нового блока в хранилище

2.2 Алгоритм слияния блоков

2.3 Алгоритм корректировки при изменении данных

3 Технологическая часть

3.1 Код получившейся структуры

3.2 Код добавления нового блока

3.3 Код слияния блоков

3.4 Код корректировки при изменении данных

4 Исследовательская часть

В рамках исследование планируется проверить, насколько сложно взломать систему без наличия и с наличием шифрования в системе.

4.1 Обход механизма защиты при отсутствии шифровании диска

4.2 Невозможность обхода защиты при наличии шифрования диска

Вывод

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. СУБД ClickHouse [Электронный ресурс]. – Режим доступа: https://clickhouse.com/ свободный – (03.09.2021).

2. MergeTree движок СУБД ClickHouse [Электронный ресурс]. – Режим доступа: https://clickhouse.com/docs/ru/engines/table-engines/mergetree-family/mergetree/ свободный – (05.09.2021).

3. Брюс Шнайер. «Прикладная криптография. 2-е издание. Протоколы, алгоритмы и исходные тексты на языке С».

4. Мао В. «Современная криптография: Теория и практика» — М.: Вильямс, 2005. — 768 с. — ISBN 5-8459-0847-7.

5. Bitcoin: A Peer-to-Peer Electronic Cash System [Электронный ресурс]. – Режим доступа: https://bitcoin.org/bitcoin.pdf свободный – (15.09.2021).

6. R.C. Merkle, ”Protocols for public key cryptosystems,”In Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, pages 122-133, April 1980.

7. Merkle Directed Acyclic Graphs (DAGs) [Электронный ресурс]. – Режим доступа: https://docs.ipfs.io/concepts/merkle-dag/ свободный – (17.09.2021).

8. Дистель, Рейнхард (2005), ”Graph Theory (3rd ed.)”, Berlin, New York: Springer-Verlag, ISBN 978-3-540-26183-4.

9. Левитин А. В. Глава 5. Метод уменьшения размера задачи: Топологическая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 220—224. — 576 с. — ISBN 978-5-8459-0987-9

10. Git Internals — Git Objects [Электронный ресурс]. – Режим доступа: https://git-scm.com/book/en/v2/Git-Internals-Git-Objects свободный – (28.09.2021).

11. G. Goodson, J. Wylie, G. Ganger, and M. Reiter, ”Efficient Byzantine-tolerant Erasure-coded Storage,” Proceedings of the International Conference on Dependable Systems and Networks (DSN-2004). Florence, Italy, 2004. Supercedes Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-03-104, December 2003

12. Dave K. Kythe, Prem K. Kythe. Algebraic and Stochastic Coding Theory. — 1-е изд. — CRC Press, 2012. — С. 375—395. — 512 с. — ISBN 978-1439881811.

13. G.Cattaneo, L. Catuogno, A. Del Sorbo, and P. Persiano, ”A Transparent Cryptographic File System for Unix,” In Proceedings of the USENIX Annual Technical Conference, Freenix Track. Boston, MA, 2001.

14. C. Wright, M. Martino, and E. Zadok. “NcryptFS: A Secure and Convenient Cryptographic File System,” In Proceedings of the USENIX Conference General Track, San Antonio, TX, June 2003.

15. Уорд Б. Внутреннее устройство Linux. — СПб.: Питер, 2016. — 384 с. — ISBN 978-5-496-01952-1.

16. J. Kubiatowitz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, ”Oceanstore: An Architecture for Global-Scale Persistent Storage,” In Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, Nov 2000.

17. H. Weatherspoon, C. Wells, and J. Kubiatowicz, “Naming and Integrity: Self-Verifying Data in Peer-to-Peer Systems,” In Proceedings of the International Workshop on Future Directions in Distributed Computing (FuDiCo 2002), June 2002.

18. Полянская О. Ю., Горбатов В. С. Инфраструктуры открытых ключей. Учебное пособие., Москва, 2007. ISBN 978-5-94774-602-0

ПРИЛОЖЕНИЕ А