Vite: Высокопроизводительная Асинхронная Децентрализованная Платформа для Приложений

Chunming Liu
charles@vite.org

Daniel Wang
daniel@loopring.org

Ming Wu
woo@vite.org

Резюме

Vite - это универсальная децентрализованная платформа приложений, которая отвечает требованиям промышленных приложений: высокая пропускная способность, низкая задержка и возможность масштабируемости с сохранением уровня безопасности. Vite использует структуру DAG ledger и транзакции в лэджерах группируются по аккаунтам. Структура Snapshot Chain в Vite может компенсировать недостатки безопасности DAG ledger. Консенсусный алгоритм HDPoS, благодаря которому запись и подтверждение транзакций асинхронны, обеспечивает высокую производительность и масштабируемость. Виртуальная машина Vite совместима с виртуальной машиной Ethereum, язык смарт-контрактов - расширенная версия Solidity, которая обеспечивает более мощную способность к описанию. Кроме того, важным усовершенствованием в разработке, Vite является внедрение асинхронной архитектуры Event Driven Architecture, которая передает информацию через сообщения между смарт-контрактами, что значительно повышает пропускную способность и масштабируемость системы. Помимо внутренних нативных токенов, Vite поддерживает пользователей для выпуска своих собственных цифровых активов, а также обеспечивает кросс-чейн переводы и обмен на основе Loopring Protocol [1]. Vite реализует распределение ресурсов по квотам и обычным пользователям не нужно платить комиссию за транзакции. Vite также поддерживает планирование контрактов, сервис имен, обновление контрактов, обрезку блоков и другие функции.

1 Введение

1.1 Определение

Vite - это универсальная платформа для децентрализованных приложений (dApp), которая может поддерживать набор смарт-контрактов, каждый из которых является конечной машиной с независимым состоянием и разной операционной логикой, которая может коммуницировать посредством доставки сообщений.

В целом, система является транзакционной конечной машиной. Состояние системы s ∈ Ş также известное как мировое состояние, заключается в состоянии каждого независимого аккаунта. Событие, которое вызывает изменения в статусе аккаунта, называется транзакцией. Более формализованное определение выглядит следующим образом:

Определение 1.1 (Транзакционная машина) *Транзакционная машина 4-х составная: (T, S, g, δ), где T представляет собой набор транзакций, S представляет собой набор состояний, g ∈ S - начальное состояние, также известное как «блок создания», δ: S × T → S является функцией перехода состояния.*

Семантика этой транзакционной конечной машины - дискретная система транзита, которая определяется следующим образом:

Определение 1.2 (Семантика транзакционной машины) *Семантика транзакционной машины (T, S, s0, δ) является дискретной системой транзита: (S, s0, →). → ∈ S × S является транзитной связью.*

В то же время децентрализованная платформа для приложений представляет собой распределенную систему с конечной согласованностью. Через некий консенсусный алгоритм конечное состояние может быть достигнуто между нодами.

В реалистичных сценариях, то, что хранится в смарт-контракте, является набором заполненных данных в децентрализованном приложении, с большим объемом и не может быть передано между нодами. Поэтому нодам необходимо передавать набор транзакций для достижения согласованности в конечном состоянии. Мы организовываем такую группу транзакций в специальные структуры данных, обычно называемые леджерами.

Определение 1.3 (Леджер) *Леджер состоит из набора транзакций с рекурсивно построенным абстрактным типом данных. Он определяется следующим образом:*

$$\begin{cases} l = \Gamma(T_t) \\ l = l_1 + l_2 \end{cases}$$

Среди них Tt ∈ 2T, представляющий собой набор транзакций, Γ ∈ 2T → L, представляет собой функцию построения книги через набор транзакций, L представляет собой набор леджеров, +: L × L → L, представляющий операцию слияния двух субледжеров в один.

Следует отметить, что в таких системах леджеры обычно используются для отображения группы транзакций, вместо структуры. В Bitcoin [2] и Ethereum [3], леджер представляет собой структуру из цепочки блоков, где транзакции глобально упорядочены. Чтобы изменить транзакцию в леджере, нам нужно перестроить субледжер в учетной книге, тем самым увеличивая стоимость вмешательства в транзакцию.

Согласно той же группе транзакций, могут быть построены разные действительные книги, но они отображают разную последовательность транзакций, и это может привести к тому, что система перейдет в другое состояние. Когда это происходит, это обычно называют «форком».

Определение 1.4 (Форк) *Предположим, что Tt, Tt∈ 𝕋, $T_t \subseteq Tt'$. Если $l = \Gamma_1(T_t)$, $l' = \Gamma_2(T_t')$ и не соответствуют $l \preceq l$, мы можем назвать l и l' форкнутыми (разветвленными) блокчейнами. $\preceq$ представляет собой префиксную связь.*

Согласно семантике транзакционной машины, мы можем легко доказать, что из начального состояния, если леджер не разветвляется, каждая нода будет в конечном счете переходить в тоже состояние. Итак, если получен форкнутый леджер, будет ли он непременно переходить в другое состояние? Это зависит от изначальной логики транзакции в леджере и как леджеры организуют частичные последовательности между транзакциями. В действительности часто существуют некоторые транзакции, которые удовлетворяют коммутативные законы, но из-за проблемы со строением аккаунт, они часто вызывают форки. Когда система запускается из начального состояния, получаются два форкнутых леджера, и они заканчиваются в одинаковых состояниях, мы называем эти два леджер ложными форкнутыми блокчейнами.

Определение 1.5 (Ложный форк) *Начальное состояние $s_0 \in S$, леджер $l_1, l_2 \in L, s_0 \xrightarrow{l_1} s_1, s_0 \xrightarrow{l_2} s_2$. Если $l_1 \neq l_2$ и $s_1 = s_2$, мы вызовем эти два леджера l_1, l_2 как ложные форкнутые леджеры.*

Хорошо продуманный леджер должен свести к минимуму вероятность ложного форка.

Когда случается форк, каждая нода должна выбрать один из нескольких форкнутых леджеров. Чтобы обеспечить согласованность состояния, нодам необходимо использовать один и тот же алгоритм для совершения выбора. Этот алгоритм называется алгоритмом консенсуса.

Определение 1.6 (Алгоритм консенсуса) *Алгоритм консенсуса - это функция, которая получает набор леджеров и возвращает один единственный леджер:*

$$\Phi : 2^L \rightarrow L$$

Алгоритм консенсуса является важной частью конструкции системы. Хороший алгоритм консенсуса должен обладать высокой скоростью конвергенции, чтобы уменьшить влияние консенсуса в разных форках, и иметь высокую способность защищаться от вредоносных атак.

1.2 Текущий прогресс

Ethereum [4] взял на себя инициативу в реализации такой системы. В конструкции Ethereum определение состояния мира это $S = \Sigma^A$, отображение из аккаунта это $a \in A$ и состояние этого аккаунта $\sigma_\alpha \in \Sigma$. Поэтому любое состояние в машине Ethereum является глобальным, что означает, что нода может в любой момент достигнуть статус любого аккаунта в любое время.

Транзитная функция δ Ethereum определена набором программных кодов. Каждая группа кода называется смарт-контрактом. Ethereum определяет полную виртуальную машину Туринга, называемая EVM, набор команд которой называется EVM-кодом. Пользователи могут создавать через язык программирования (похожий на JavaScript) Solidity смарт-контракты, компилировать их в EVM-код и развертывать на Ethereum [5].

Как только смарт-контракт успешно развертан, это является эквивалентом определения – аккаунт контракта α получает функция перехода состояния δ_α. EVM широко используется в таких платформах, но есть и некоторые проблемы. Например, существует недостаток поддержки функции библиотеки и проблемы безопасности.

Структура леджера Ethereum представляет собой цепочку блоков [2] Цепочка блоков состоит из блоков, каждый блок содержит список транзакций, а последний блок обращается к хешу предыдущего блока для формирования цепной структуры.

$$\Gamma(\{t_1, t_2, \ldots | t_1, t_2, \ldots \in T\}) = (\ldots,(t_1, t_2, \ldots)) \quad (1)$$

Наибольшим преимуществом этой структуры является способность эффективно предотвращать появление подделанных транзакций, но так как данная структура поддерживает полный порядок всех транзакций, смена двух транзакционных порядков будут генерировать новый леджер, который имеет более высокую вероятность форка. Фактически, согласно этому определению, состояние пространства транзакционной конечной машины рассматривается как дерево: начальное состояние – это корневая нода, разный порядок транзакций представляет разные пути, а нода листа дерева - конечное состояние. На самом деле состояние большого числа нод листа одинаковы, что приводит к большому числу ложных форков.

Консенсусный алгоритм Ф, который называется PoW, впервые был предложен в протоколе Bitcoin [2]. Алгоритм PoW полагается на математическую задачу, которая легко поддается проверке, но ее трудно решить. Например, на основе хэш-функции h: N → N, нахождение результата x, чтобы удовлетворить требованию h (T + x) ≥ d, dis заданное число, называемое трудностью, T представляет собой двоичное представление торгового списка, содержащегося в блоке. Каждый блок в цепочке блоков содержит решение таких задач. Добавим сложность всех блоков, которые являются общей сложностью цепочки блоков леджера:

$$D(l) = D\left(\sum_i l_i\right) = \sum_i D(l_i) \quad (2)$$

Поэтому при выборе правильного аккаунта из форков, выбирается форк с наивысшей трудностью:

$$\Phi(l_1, l_2, \ldots, l_n) = l_m, \text{где } m = arg \max_{i \in 1..n}\left(D(l_i)\right) \quad (3)$$

Консенсусный алгоритм PoW обеспечивает лучшую безопасность и хорошо работает в Bitcoin и Ethereum. Однако, в этом алгоритме есть две основные проблемы. Первая – чтобы решить математическую задачу требуется большой количество вычислительных ресурсов, в результате чего тратится электрическая энергия. Вторая - медленная скорость конвергенции алгоритма, что влияет на общую пропускную способность системы. В настоящее время TPS (количество транзакций в секунду) Ethereum составляет всего около 15, что полностью не в состоянии удовлетворить потребности децентрализованных приложений.

1.3 Направление совершенствования

После создания Ethereum сообщество Ethereum и другие подобные проекты начали улучшать систему в различных направлениях. Из абстрактной модели системы, можно улучшить следующие направления:

• Улучшение системы S
• Улучшение транзитной функции δ
• Улучшение структуры леджера Г
• Улучшение консенсусного алгоритма Ф

1.3.1 Улучшение системы

Основная идея улучшения системы заключается в том, чтобы локализовать глобальное состояние мира, каждая нода больше не касается всех транзакций и трансферов, а только поддерживает подмножество всей машины. В этом случае, потенциалы множества S и множества T значительно уменьшены, тем самым улучшая масштабируемость системы. Такие системы используют: Cosmos [6],Aelf [7],PChain и т.д.

По сути, эта схема, основанная на боковых цепях (side chains), жертвует целостностью состояния системы в обмен на масштабируемость. Это делает децентрализацию каждого децентрализованного приложения, работающего в подобных леджерах, ослабленной - история транзакций смарт-контракта больше не сохраняется в каждой ноде во всей сети, а только в части нод. Кроме того, взаимодействие между контрактами станет узким местом такой системы. Например, в Cosmos, взаимодействия в разных Зонах требуют общего центра цепочек (chain Hub) для завершения операции [6].

1.3.2. Улучшение транзитной функции

Основываясь на улучшении EVM, некоторые проекты предоставляют более функциональные языки программирования смарт-контрактов. Например, язык смарт-контрактов Rholang установленный в RChain базируется на основе π калькуляций; смарт-контракт NeoContract в блокчейне NEO может быть расширен с помощью популярных языков программирования, такие как Java, C# и т.д.; EOS программируется с помощью C/C++.

1.3.3 Улучшение структуры леджера

Направлением улучшения структуры леджера является конструкция эквивалентного класса. Линейный леджер с глобальным порядком множественных транзакций улучшен до нелинейного леджера, который записывает только частично упорядоченые связи. Эта нелинейная структура леджера представляет собой DAG (Directed Acyclic Graph – Направленный Ациклический Граф). В настоящее время, Byteball [8], IOTA [9], Nano [10] и другие проекты реализовали функцию шифрования денег, основанной на структуре аккаунта DAG. Некоторые проекты пытаются использовать DAG для реализации смарт-контрактов, но до сих пор, улучшения в этом направлении все еще изучаются.

1.3.4 Улучшение консенсусного алгоритма

Совершенствование алгоритма консенсуса в основном заключается в улучшении пропускной способности системы и основное направление заключается в подавлении генерации ложных форков. Далее мы будем обсуждать, какие факторы задействованы в ложном форке.

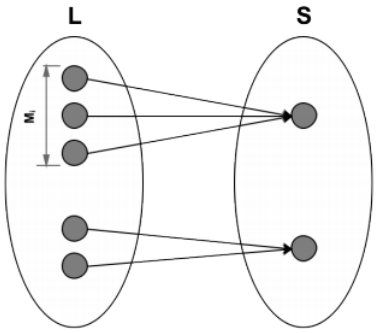

Рисунок 1: Ложный форк

Как показано на рисунке 1, L представляет собой набор всех возможных форкнутых аккаунтов для набора транзакций, а S - совокупность состояний, которые могут быть достигнуты в разных порядках. Согласно определению 1.4, отображение $f: L \rightarrow S$ является сюръективным. А согласно определению 1.5, это отображение не является инъективным. Далее мы вычисляем вероятность ложного форка:

Предположим, что пользователи C имеют право создавать леджеры, $M = |L|$, $N = |S|$, $M_i = |L_i|$, где $L_i = \{l \mid f(l) = s_i, s_i \in S\}$. Вероятность ложного форка следующая:

$$P_{ff} = \sum_{i=1}^{N} \left(\frac{M_i}{M}\right)^C - \frac{1}{M^{C-1}} \qquad (4)$$

Из этой формулы видно, что для уменьшения вероятности ложного форка, есть два пути:

• Установить эквивалентные связи на L наборов леджера, разделить классы эквивалентности на них и построить меньше форкнутых леджеров.

• Ограничить пользователей, которые имеют право создавать леджер, тем самым уменьшая C.

Первый способ - это важное направление в конструкции Vite. Он будет подробно рассмотрено позже. Второй способ был принят многими алгоритмами. В алгоритме PoW, любой пользователь имеет право на создание блока; и PoS алгоритм ограничивает возможность производства блоков, теми кто обладает правами системы; алгоритм DPoS [11] допускает пользователя с правом на создание блока, для дальнейшего ограничения внутри группы агента нод.

В настоящее время посредством усовершенствованного консенсусного алгоритма появились некоторые влиятельные проекты. Например, Кардано использует PoS-алгоритм, названный Ouroboros, и литература [12] дает строгое доказательство связанных признаков алгоритма; алгоритм BFT-DPOS, используемый EOS [13], является вариантом алгоритма DPoS и улучшает пропускную способность системы путем убыстрением производства блоков; алгоритм Qtum [14] также является алгоритмом PoS; алгоритм Каспер, принятый RChain [15] является одним из алгоритмов PoS.

Существуют и другие проекты, которые выдвигают свои собственные предложения по совершенствованию консенсусного алгоритма. NEO [16] использует алгоритм BFT, называемый dBFT, Cosmos [6] использует алгоритм под названием Tendermint [17].

2 Леджеры

2.1 Обзор

Роль леджеров заключается в определении порядка транзакций, и порядок транзакций будет влиять на следующие два аспекты:

- **Согласованность статуса**: поскольку состояние системы - не CRDT (Conflict-free replicated data types - безконфликтные повторяющиеся типы данных) [18], не все транзакции заменяемые, а последовательность выполнения различных транзакций может привести к системе, переходящей в другое состояние.
- **Эффективность хэша:** в леджере транзакция будет упакована в блоки, содержащие хэш, поэтому блоки ссылаются друг на друга. Последовательность транзакций влияет на связность хэша, цитируемого в леджере. Чем больше масштабы этого воздействия, тем больше затраты на фальсификацию транзакций. Это связано с тем, что любое изменение транзакции должно переделать хэш, который прямо или косвенно ссылается на блок транзакции.

  Строение леджера также имеет две основные задачи:

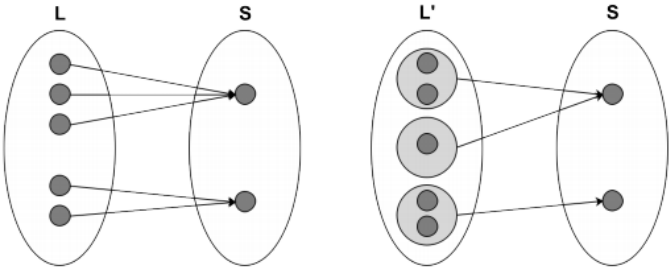

Рисунок 2: Слияние леджера

- **Уменьшение частоты возникновения ложного форка**: как обсуждалось в предыдущем разделе, уменьшение частоты возникновения ложного форка может быть достигнуто путем создания эквивалентного класса и объединения группы аккаунтов, которые приводят систему в одно и то же состояние, к одному аккаунту. Как показано выше, в соответствии с формулой частоты возникновения ложного форка, частота возникновения ложного форка в блокчейне слева:

  $P_{ff} = \left(\frac{3}{5}\right)^C + \left(\frac{2}{5}\right)^C - \frac{1}{5^{C-1}}$; после слияния пространства леджера частота возникновения ложного форка в правой графе:

  $P_{ff} = \left(\frac{2}{3}\right)^C + \left(\frac{1}{3}\right)^C - \frac{1}{3^{C-1}}$. Известно, что когда $C > 1$, $P_{ff}{}' < P_{ff}$. Иными словами, мы должны минимизировать частичной упорядоченной связи между транзакциями и позволить, чтобы больше транзакций было обмененено последовательно.
- **Поддельные доказательства:** когда транзакция t изменяется в леджере *l*, в двух субледжерх $l=l_1+l_2$, субледжер l_1 не затрагивается, а ссылки хэша в субледжере l_2 должна быть перестроены, чтобы сформировать новый валидный леджер $l' = l_1 + l_2'$. Поврежденный субледжер $l_2 = \Gamma(T_2)$, $T_2 = \{x | x \in T, x > t\}$. Таким образом, для увеличения стоимости подделки транзакций необходимо поддерживать частичную упорядоченную связь между транзакциями в максимально возможной степени, чтобы расширить масштаб подделки $|T_2|$.

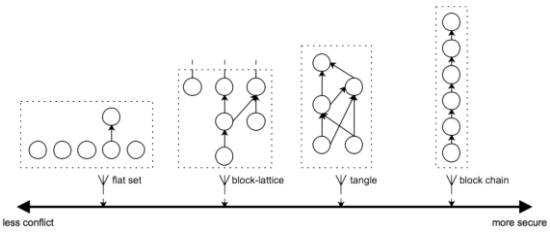

Рисунок 3: Сравнение структуры леджера

Очевидно, что вышеуказанные две цели противоречивы, и при разработке структуры аккаунта необходимо идти на компромиссы. Поскольку обслуживание аккаунта является частичным упорядочиванием между транзакциями, это по существу является частичным упорядоченным множеством (poset) [19], если он представлен диаграммой Хассе (диаграмма Хассе) [20], это DAG по топологии.

На приведенном выше рисунке сравниваются несколько общих структур леджеров, а леджеры слева расположены с менее частичным упорядочиванием. Диаграмма Хассе выглядит плоской и имеет более низкую частоту возникновения ложного форка; блокчейны с правой стороны поддерживают больше частично упорядоченных множеств, а диаграмма Хассе более тонкая и более устойчивая к вмешательству.

На рисунке, самая левая сторона является обобщенной структурой в централизованных системах без каких-либо функций защиты от несанкционированного доступа; самая правая сторона - типичная цепочка блоков леджера с лучшими функциями защиты от несанкционированного доступа; между этими двумя, имеются два леджера DAG, блок-решетчатый аккаунт [10], используемый в Nano (слева); и правая сторона, журнал сплетений [9], используемый в IOTA. С точки зрения характеристик, блок-решетка в меньшей степени поддерживает частично упорядоченные связи и более подходит для структуры учета высокопроизводительных децентрализованных платформ для приложений. Из-за его плохих характеристик связанных с несанкционированным доступом, он может подвергать опасности безопасность, до сих пор никакие другие проекты не применяли эту структуру леджера, кроме Nano.

Для достижения высокой производительности Vite использует структуру леджера DAG. В то же время, вводя дополнительную цепную структуру Snapshot Chain и улучшая консенсусный алгоритм, успешно устраняются недостатки защиты блок-решетки, а также два улучшения будут подробно обсуждаться позже.

2.2 Предварительное ограничение

Во-первых, давайте взглянем на предварительное условие использования этой структуры леджера для модели конечной машины. Эта структура является, по существу, комбинацией всей конечной машины как набора независимых машин, каждый аккаунт соответствует независимой конечной машине, и каждая транзакция влияет только на состояние аккаунта. В леджере все транзакции группируются по аккаунтам и организуются в цепочку транзакций в одном аккаунте. Поэтому мы имеем следующие ограничения на состояние системы S и транзакцию T в Vite:

Определение 2.1 (Ограничение степени свободы) *состояние системы $s \in S$, представляет собой вектор $s = (s_1, s_2, ..., s_n)$, образованный состоянием s_i каждого аккаунта. Для $\forall t_i \in T$, после выполнения транзакции t_i, передача состояния системы выглядит следующим образом: $(s_1', ..., s_i', ..., s_n') = \sigma(t_i, (s_1, ..., s_i, ..., s_n))$, должны встречаться: $s_j' = s_j$, $j \neq i$. Это ограничение называется ограничением одной степени свободы для транзакции.*

Интуитивно, транзакция с одной степенью свободы изменяет только состояние учетной записи (аккаунта), не затрагивая статус других учетных записей в системе. В многомерном пространстве, где находится вектор пространства состояний, выполняется транзакция, и состояние системы перемещается только вдоль направления, параллельного оси координат. Обратите внимание, что это определение более строгое, чем определение транзакции в Bitcoin, Ethereum и других моделях. Транзакция в биткойне изменит состояние двух учетных записей - отправителя и получателя; в Ethereum -может изменить состояние более двух учетных записей посредством вызова сообщения.

При этом ограничении связь между транзакциями может быть упрощена. Любая из двух транзакций является либо ортогональной, либо параллельной. Это обеспечивает условия для группировки транзакций по учетным записям. Вот пример для иллюстрации:

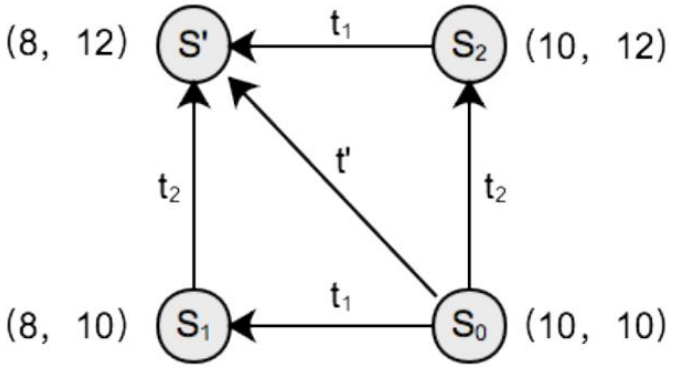

Рисунок 4: Одна степень свободы торговли и промежуточного состояния

Как показано на рисунке выше, предположим, что у Алисы и Боба 10 долларов США соответственно. Исходным состоянием системы является s_0 = (10, 10). Когда Алиса хочет передать 2 доллара США Бобу, в модели Bitcoin и Ethereum, транзакция t', может заставить систему перейти непосредственно в конечное состояние: $s_0 \xrightarrow{t'} s''$.

В определении Vite транзакция t' также изменила статус двух учетных записей Алисы и Боба, что не соответствуют принципу одной степени свободы. Поэтому транзакция должна быть разделена на две транзакции:

1) А транзакция t_1, которая представляет собой передачу 2 долларов США Алисой;

2) А транзакция t_2, которая представляет собой получение 2 долларов США Бобом.

Таким образом, от начального состояния до конечного состояния s' могут быть два разных пути $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} s'$ и $S_o \xrightarrow{t_2} s_2 \xrightarrow{t_1} s'$. Эти два пути соответственно пройдены через промежуточные состояния s_1 и s_2, и эти два промежуточных состояния являются отображением конечного состояния s' в двух измерениях учетной записи. Другими словами, если вы заботитесь только о состоянии одной из учетных записей, вам нужно выполнить только все транзакции, которые соответствуют учетной записи, и не нужно выполнять транзакции других учетных записей.

Затем мы определим, как разделить транзакции в Ethereum в одну степень свободы транзакций, необходимых Vite:

Определение 2.2 (Разделение транзакций) *Разделение транзакции со степенью свободы больше 1 на набор транзакций с одной степенью свободы, называется Разделение Транзакций. Перемещаемую транзакцию можно разделить на отправленную транзакцию и принимаемую транзакцию; транзакция, вызывающая контракт, может быть разделена на транзакцию запроса контракта и транзакцию ответа на контракт; вызов в рамках каждого контракта может быть разделен на транзакцию запроса контракта и транзакцию ответа контракта.*

Таким образом, в леджерах может быть два разных типа транзакций. Их называют «торговыми парами»:

Определение 2.3 (Торговая пара) *отправленная транзакция или транзакции запроса контракта, именуется в целом как «транзакция запроса»; принимаемая транзакция или транзакция ответа контракта, именуется в целом как «транзакцией ответа». Операция запроса и соответствующая транзакция ответа называются транзакционными парами. Учетная запись для инициирования запроса на транзакцию t записывается как A(t); соответствующая транзакция ответа записывается как: t, учетная запись, соответствующая транзакции, записывается как A (ť).*

Исходя из вышеприведенного определения, мы можем заключить возможную связь между любыми двумя транзакциями в Vite:

Определение 2.4 (Связь транзакций) *Для двух транзакций t_1 и t_2 могут существовать следующие связи:*

Ортогональность: *если $A(t_1) \neq A(t_2)$, две транзакции ортогональны, записывается как $t_1 \perp t_2$;*

Параллельность: *если $A(t_1) = A(t_2)$, две транзакции параллельны, записывается как $t1 \parallel t2$;*

Причинность: *если $t_2 = t_1$, то две транзакции являются причинными, записывается как $t_1 \triangleright t_2$ или $t_2 \triangleleft t_1$.*

2.3. Определение Леджера

Чтобы определить леджер, необходимо определить частичное упорядоченное множество (poset). Во-первых, давайте определим частичную упорядоченную связь между транзакциями в Vite:

Определение 2.5 (Частичный порядок транзакций) мы используем дуалистические связи < для представления частичной упорядоченной связи двух транзакций:

Ответная транзакция должна соответствовать подобной транзакции запроса: t1 <t2 ⇔ t1 ▷ t2;

Все транзакции в учетной записи должны быть строго и глобально упорядочены: ∀t1 // t2, должно быть: t1<t2 или t2<t1.

В связи с частично упорядоченной связью, установленной в транзакции, множество T соответствует характеристикам:

• Невозвратность: ∀t ∈ T, нет t <t;

• Транзитивность: ∀t1, t2, t3 ∈ T, если t1 <t2, t2 <t3, то t1 <t3;

• Асимметричность: ∀t1, t2 ∈ T, если t1 <t2, то не существует t2 <t1

Таким образом, мы можем определить учетную запись Vite в строгом частичном порядке:

Определение 2.6 (Vite леджер) *Vite леджер - это строго частично упорядоченное множество, состоящее из множества T транзакций, и частично упорядоченного множества <.*

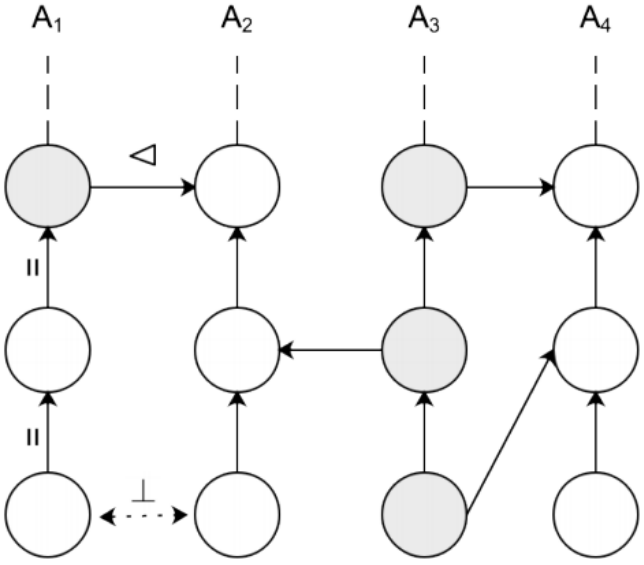

Рисунок 5: Связь между леджером и транзакцией в Vite

Строгое частично упорядоченное множество может соответствовать структуре леджера DAG. Как показано на рисунке выше, круги обозначают транзакции, а стрелки обозначают зависимости между транзакциями. a → b указывает, что a зависит от b.

Vite леджер, определение которого дано выше, структурно похож на блок-решетку. Транзакции делятся на транзакции запроса и ответа, каждая из которых соответствует отдельному блоку, каждая учетная запись Ai соответствует цепочке, транзакционная пара и транзакция ответа, ссылаются на хэш соответствующей транзакции запроса.

цепь, новый форк будет выбран в качестве консенсусного результата, и оригинальный форк будет брошен, а транзакция в оригинальном форке откатится. В такой системе, откат транзакций - очень серьезное событие, которое приведет к двойной трате средств. Представьте, что бизнес получает оплату за предоставление товаров или услуг, и после этого платеж снимается еще раз, продавец может столкнуться с убытками. Следовательно, когда пользователь получает транзакцию платежа, ему нужно подождать для того, чтобы система «подтвердила» транзакцию, чтобы гарантировать, что вероятность отката назад достаточно низкая.

Определение 3.1 (Подтверждение транзакции) *когда вероятность отката транзакции меньше, чем заданный порог ε, транзакция называется подтвержденной. $P_r(t) < ε$ ⇔ t значит, что транзакция подтверждена.*

Подтверждение транзакций - очень запутанная концепция, потому что независимо от того, признана ли транзакция, по факту она зависит от неявного уровня достоверности 1 - ε. Торговец бриллиантами и продавец кофе понесут разные потери, когда они получат двойную трату средств. В результате, сначала необходимо установить меньшее значение є транзакции. В Bitcoin важны также количества подтверждений. В Bitcoin номер подтверждения указывает глубину транзакции в цепочке блоков. Чем больше количество подтверждений, тем меньше вероятность отката транзакции [2]. Таким образом, продавцы могут косвенно устанавливать доверительный уровень подтверждения, задавая номер ожидания количества подтверждений.

Вероятность отката транзакции со временем уменьшается из-за того, что хэш связан со структурой учетной записи. Как упоминалось выше, когда леджер имеет лучшие характеристики от вмешательства, откат транзакции требует восстановления всех последующих блоков обмена в блоке. Поскольку новые транзакции постоянно добавляются в леджеры, появляется все больше и больше успешных нод в транзакции, поэтому вероятность подделки будет уменьшаться.

В структуре блок-решетки, когда транзакция группируется по учетной записи, транзакция будет привязана только к концу цепочки учетных записей собственной учетной записи, а транзакция, сгенерированная большинством других учетных записей, автоматически не станет нодой-преемником транзакции. Поэтому необходимо разработать согласованный алгоритм, чтобы избежать скрытых опасностей двойных трат.

Nano принял на вооружение алгоритм, основанный на голосовании [10], транзакция подписывается набором представительских нод, выбранных группой пользователей. Каждый представительская нода имеет вес. Когда сигнатура транзакции имеет достаточный вес, считается, что транзакция подтверждена. В этом алгоритме есть следующие проблемы:

Во-первых, если требуется более высокая степень достоверности подтверждения, необходимо повысить порог веса голосования. Если в онлайн недостаточно представительских нод, скорость пересечения не может быть гарантирована, и, возможно, пользователь никогда не соберёт количество тикетов, необходимых для подтверждения обмена;

3 Снэпшот цепь

3.1 Подтверждение транзакции

Когда учетная запись разветвляется, результат консенсуса может колебаться между двумя разветвленными леджерами. Например, на основе блочной структуры, если нода получает более длинную разветвленную (форкнутую)

Во-вторых, вероятность отката транзакций со временем не уменьшается. Это потому, что в любое время, стоимость свержения прошедшего голосования одинакова.

Наконец, результаты прошедшего голосования не сохраняются в леджере и хранятся только в локальном хранилище нод. Когда нода получает свою учетную запись из других нод, нет возможности достоверно оценить вероятность отката прошлых транзакций.

По сути, механизм голосования является частично централизационным решением. Мы можем рассматривать результаты голосования как снимок (snapshot) статуса леджеров. Этот снимок будет распространен в локальном хранилище каждой ноды в сети. Чтобы иметь ту же самую способность защиты от несанкционированного доступа в леджер, мы также можем организовать эти снимки в цепные структуры, которые является одним из главных особенностей дизайна Vite – Snapshot chain [21].

3.2 Определение снэпшот цепи

Снэпшот цепь (Snapshot chain, дословно - цепочка моментальных снимков) - самая важная структура хранения в Vite. Его основная функция заключается в поддержании консенсуса леджера Vite. Сначала мы дадим определение снэпшот цепи:

Определение 3.2 (Снэпшот блок и снэпшот цепь) *В снэпшот блоке хранятся снэпшот состояния леджера Vite, включая баланс учетных записей, хэш-дерево Меркла состояния контрактов и хэш последнего блока в каждой учетной записи цепи. Снэпшот цепь представляет собой цепочку, состоящую из снэпшот блоков, следующий блок ссылается на хэш предыдущего блока.*

Состояние учетной записи пользователя содержит баланс и хэш последнего блока цепочки учетной записи; в дополнение к вышеуказанным двум полям, состояние учетной записи контракта содержит хэш-дерево Меркла. Структура состояния учетной записи выглядит следующим образом:

```
struct AccountState {
  // account balance
  map<uint32, uint256> balances;
  // Merkle root of the contract state
  optional uint256 storageRoot;
  // hash of the last transaction
  // of the account chain
  uint256 lastTransaction;
}
```

Структура снэпшот блока определяется следующим образом:

```
struct SnapshotBlock {
  // hash of the previous block
  uint256 prevHash;
  // snapshot information
  map<address, AccountState> snapshot;
  // signature
  uint256 signature;
}
```

Чтобы одновременно поддерживать несколько токенов, структура записи информации баланса в состоянии учетной записи Vite является не uint256, а отображается через идентификатор токена (token ID) в балансе.

Первый снэпшот блок в снэпшот цепи называется «genesis snapshot» (снэпшот зарождения), который сохраняет снэпшоты генезис-блока в учетную запись.

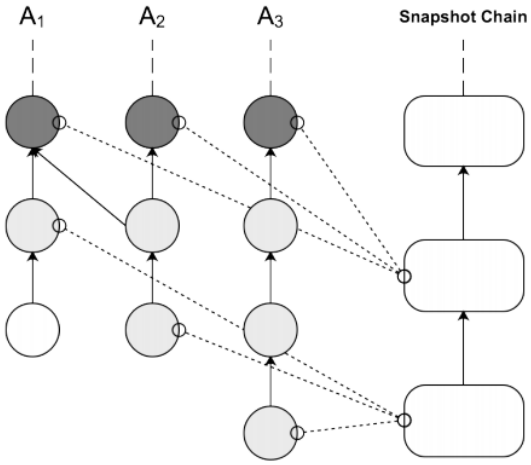

Рисунок 6: Снэпшот цепь

Поскольку каждый снэпшот блок в снэпшот цепи соответствует единственному форку леджера Vite, можно определить консенсусный результат леджера Vite с помощью снэпшот блоков, когда снэпшот блок не разделяется.

3.3 Снэпшот цепь и подтверждение транзакций

После введения снэпшот цепи устраняются естественные недостатки безопасности блочно-решетчатой структуры. Если злоумышленник хочет сгенерировать транзакцию с двойной тратой, в дополнение к перестройке хеш-ссылки в леджере Vite, ее также необходимо перестроить в снэпшот цепи для всех блоков после первого снэпшота транзакции и потребовать создать более длинную снэпшот цепочку. Таким образом, стоимость атаки будет значительно увеличена.

В Vite механизм подтверждения транзакций похож на механизм подтверждения транзакций в Bitcoin, который определяется следующим образом:

Определение 3.3 (подтверждение транзакции в Vite) *в Vite, если транзакция является снэпшотом от снэпшот цепи, транзакция подтверждается. Глубина снэпшот блока в первом снэпшоте, называется номером подтверждения транзакции.*

В соответствии с этим определением, количество подтвержденных транзакций увеличится на 1, когда снэпшот цепь будет расти, а вероятность атаки двойной траты будет уменьшаться с увеличением снэпшот цепи. Таким образом, пользователи могут настроить требуемое количество подтверждений в соответствии с конкретным сценарием.

Сама снэпшот цепь полагается на консенсусный алгоритм. Если снэпшот цепь раздвоена, **самый длинный** форк будет выбран в качестве валидного форка. Когда снэпшот цепь переключается на новый форк, исходная снэпшот информация будет откатана назад, что означает, что первоначальный консенсус в леджере был свергнут и заменен новым консенсусом. Таким образом, снэпшот цепь является краеугольным камнем всей системы безопасности, и к ней нужно относиться серьезно.

3.4 Сжатое хранение

Поскольку все состояния учетной записи должны быть сохранены в каждом снэпшот блоке в снэпшот цепи, пространство для хранения должно быть очень большим, поэтому необходимо сжатие снэпшот цепи.

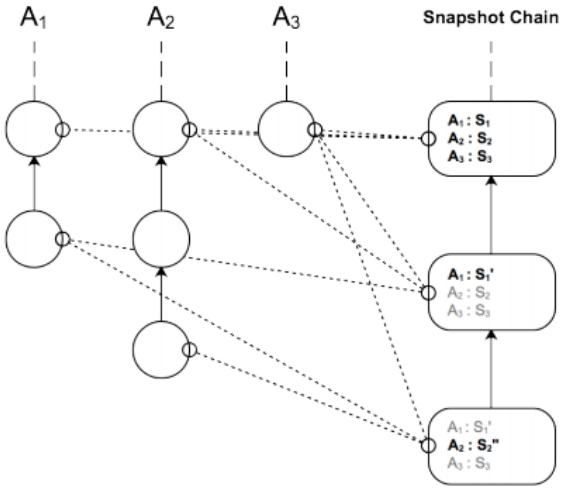

Рисунок 7: Снэпшот перед сжатием

Основной подход сжатия пространства для хранения снэпшот цепи заключается в использовании инкрементного (возрастающего) хранилища: снэпшот блок сохраняет только данные, которые были изменены по сравнению с предыдущим снэпшот блоком. Если не было транзакции для одной учетной записи между двумя снэпшотами, последний снэпшот блок не сохранит данные учетной записи.

Чтобы восстановить информацию снэпшота, вы можете перемещать снэпшот блок от начала до конца и открывать данные каждого снэпшот блока с текущими данными.

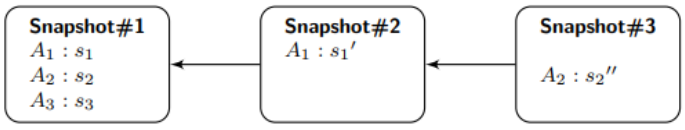

Рисунок 8: Снэпшот после сжатия

Только окончательный статус каждого снэпшота учетной записи сохраняется при снэпшоте, промежуточное состояние не учитывается, поэтому сохраняется только одна копия данных в снимке, независимо от количества транзакций, генерируемых учетной записью между двумя снэпшотами.

Следовательно, снэпшот блок занимает максимум S ∗ A байт, где S = size of (si) - количество байт, занятых для каждого состояния учетной записи, а A - общее количество учетных записей системы. Если среднее отношение активных счетов к общим счетам равно a, то степень сжатия равна 1-a.

4 Консенсус

4.1 Цель разработки

При разработке консенсусного протокола нам необходимо учесть следующие факторы:

- **Производительность.** Основная цель Vite - скорость. Чтобы обеспечить высокую пропускную способность и низкую задержку в системе, нам нужно принять алгоритм консенсуса с более высокой скоростью сходимости.
- **Масштабируемость.** Vite - это общедоступная платформа, открытая для всех децентрализованных приложений, поэтому масштабируемость также важно учитывать.
- **Безопасность.** Принцип строения Vite не преследует абсолютной безопасности, однако в конечном итоге необходимо обеспечить достаточную базовую линию безопасности и эффективно защищаться от всех видов атак.

По сравнению с некоторыми существующими консенсус-алгоритмами безопасность PoW лучше, и можно достичь консенсуса, если вычислительная мощность злонамеренных узлов ниже 50%. Однако скорость пересечения PoW медленная и не может соответствуют требованиям к производительности; PoS (и его варианты) алгоритм устраняет шаги для решения математических задач, увеличивает скорость пересечения и стоимость одной атаки, а также снижает потребление энергии. Но масштабируемость PoS по-прежнему плоха, а проблема «отсутствия ставки» [22] трудно решить; алгоритмы BFT имеют лучшую производительность и безопасность, но их масштабируемость является проблемой, обычно такой консенсус более подходит для частной цепи или консорциумной цепи; серия алгоритмов DPoS [11] эффективно снижает вероятность ложного форка, ограничивая разрешения генерации блоков. Производительность и масштабируемость здесь хороши. Как следствие, DPoS имеет небольшую жертву для безопасности, т.к. количество вредоносных узлов не должно превышать 1/3 [23].

Как правило, алгоритм DPoS имеет очевидные преимущества в производительности и масштабируемости. Поэтому мы выбираем DPoS как основу консенсусного протокола Vite и расширяем его. Благодаря иерархическому делегированному консенсусному протоколу и асинхронной модели общая производительность платформы может быть дополнительно улучшена.

4.2 Иерархический консенсус

Консенсусным протоколом Vite является HDPoS (Hierarchical Delegated Proof of Stake - иерархический делегированный PoS). Основная идея состоит в том, чтобы разложить консенсусную функцию Φ (функциональное разложение):

$$\Phi(l_1, l_2, \ldots, l_n) = \Psi(\Lambda_1(l_1, l_2, \ldots, l_n), \Lambda_2(l_1, l_2, \ldots, l_n), \ldots \Lambda_m(l_1, l_2, \ldots, l_n)) \quad (5)$$

Λ_i: $2^L \rightarrow L$, называется локальной консенсусной функцией, возвращаемый результат называется локальным консенсусом; Ψ: $2^L \rightarrow L$, известная как глобальная функция консенсуса, выбирает уникальный

результат от группы кандидатов в местном консенсусе, как окончательный результат консенсуса.

После этого разделения консенсус всей системы становится двумя независимыми процессами:

Локальный консенсус генерирует блоки, соответствующие запросу транзакций и ответным операциям в учетной записи пользователя или учетной записи контракта, и записывает их в леджер.

Глобальный консенсус создает снэпшоты данных в леджере и генерирует снэпшот блоки. Если леджер форкнут, значит выбирает один из форков.

4.3 Право на создание блоков и консенсус группа

Кто имеет право генерировать блоки транзакций в леджере и снэпшот блоки в снэпшот цепи? Какой консенсусный алгоритм принят для достижения консенсуса? Поскольку структура леджер Vite организована в несколько цепей учетных записей в соответствии с разными учетными записями, мы можем легко определить как право производства блоков в леджере в соответствии с размером учетной записи, так и право на производство снэпшот блоков, принадлежащих к одной группе пользователей. Таким образом, мы можем направить определенное количество учетных записей цепей или снэпшот цепей в консенсусную группу, а в консенсусной группе мы можем использовать единый способ создания блока и достичь консенсуса.

Определение 4.1 (Группа консенсуса) *Группа консенсуса – это набор (L, U, Ф, P), описывающий консенсусный механизм части учетных записей или снэпшот цепи, L ∈ A|{As}, представляет собой одно или несколько цепочек учетных записей или снэпшот цепи консенсусной группы в леджере; U представляет собой пользователя с правом создания блоков на цепи, указанной в L; Ф определяет консенсусный алгоритм консенсус группы; и P задает параметры консенсусного алгоритма.*

В соответствии с этим определением, пользователи могут гибко настраивать консенсусные группы и выбирать различные консенсусные параметры для своих потребностей. Затем мы рассмотрим различные консенсусные группы.

4.3.1 Консенсусная группа снэпшота

Консенсусная группа снэпшот цепей называется консенсусной группой снэпшотов, которая является самой важной консенсусной группой в Vite. Консенсусный алгоритм Ф консенсусной группы снэпшотов принимает алгоритм DPoS и соответствует Ψ в иерархической модели. Количество агентов и интервал генерации блока определяются параметром P.

Например, мы можем указать консенсусные группы снэпшота с 25 прокси-нодами для создания снэпшот блоков с интервалами в 1 секунду. Это гарантирует, что транзакция будет подтверждена достаточно быстро. Для достижение 10 кратного подтверждения транзакции необходимо дождаться 10 секунд максимум.

4.3.2 Частная консенсусная группа

Частная консенсусная группа применима только к производству блоков транзакций в леджерах и принадлежит к цепи счетов частной консенсусной группы. Блоки могут производится только владельцем приватного ключа учетной записи. По умолчанию все учетные записи пользователей принадлежат частным консенсусным группам.

Наибольшее преимущество частной консенсусной группы заключается в уменьшении вероятности форка. Потому что только один пользователь имеет право производить блоки, единственная возможность форка заключается в том, что пользователь инициирует атаки двойной траты лично или из-за программной ошибки.

Недостатком частной консенсусной группы является то, что пользовательские ноды должны быть в сети, прежде чем они смогут валидировать транзакции. Это не очень подходит для аккаунта контракта. После сбоя ноды владельца, ни одна другая нода не сможет заменить транзакцию ответа, которая создает контракты, что эквивалентно сокращению доступности сервиса децентрализованного приложения.

4.3.3 Группа консенсуса делегатов

В группе консенсуса делегатов вместо учетной записи пользователя набор назначенных прокси-узлов используется для валидации транзакции через алгоритм DPoS. Учетные записи пользователей и аккаунты контрактов могут быть добавлены к консенсусной группе. Пользователи могут настроить набор отдельных агентов нод и создать новую консенсусную группу. В Vite существует также консенсусная группа по умолчанию, чтобы помочь всем валидировать транзакции других счетов, которые не создали индивидуально своей консенсусной группы делегатов, которая также известна как **публичная консенсусная группа**.

Консенсусная группа делегатов подходит для большинства аккаунтов контрактов, поскольку большая часть транзакций в аккаунте контракта - это транзакции ответа контракта, в которых требуются более высокая доступность и более низкие задержки, чем получаемые транзакции в учетной записи пользователя.

4.4 Приоритет консенсуса

В протоколе Vite приоритет глобального консенсуса выше, чем у местного консенсуса. Когда местный консенсус разветвляется, результат глобального консенсусного выбора будет преобладать. Другими словами, как только глобальный консенсус выберет форк местного консенсуса в качестве конечного результата, даже если будет более длинный форк определенной аккаунтной цепи в будущих аккаунтах, это не приведет к откату результатов глобального консенсуса.

Эта проблема требует большего внимания при реализации кросс-чейн протокола. Поскольку целевая цепь может откатываться назад, соответствующая аккаунтная цепь контракта, связывающего цепочку, также должна откатываться соответственно. В настоящий момент, если местный консенсус в цепи связующего контракта был принят глобальным консенсусом, невозможно завершить откат, что может привести к несогласованности данных между связующим контрактом и целевой цепочкой.

Способ избежать этой проблемы состоит в том, чтобы установить параметр *delay* (задержка) в параметре консенсусной группы P, который определяет консенсусную группу снэпшота, снэпшот только местного консенсуса будет выполнен после P количества блоков. Это значительно уменьшит вероятность несогласованности связующих контрактов, но её полностью не избежать. В кодовой логике связующих контрактов также необходимо иметь дело с откатом целевой цепи отдельно.

4.5 Асинхронная модель

Для дальнейшего повышения пропускной способности системы нам необходимо поддерживать более совершенную асинхронную модель на консенсусном механизме.

Жизненный цикл транзакции включает инициацию транзакции, запись транзакции и подтверждение транзакции. Чтобы улучшить производительность системы, нам необходимо сконструировать эти три шага в асинхронном режиме. Это связано с тем, что в разное время количество транзакций, инициированных пользователями различно, скорость записи транзакций и подтверждение транзакции, обрабатываемых системой, фиксируются относительно этого. Асинхронный режим помогает сгладить пики и впадины, тем самым улучшая общую пропускную способность системы.

Асинхронная модель Bitcoin и Ethereum проста: все транзакции, инициированные пользователями, помещаются в неподтвержденный пул. Когда майнер помещает их в блок, транзакция записывается и подтверждается одновременно. Когда цепочка блоков продолжает расти, транзакция в конечном итоге достигает предварительно заданного уровня подтверждения.

В этой асинхронной модели есть две проблемы:

- Транзакции не сохраняются в леджере в неподтвержденном состоянии. Непризнанные транзакции нестабильны, и консенсус не вовлечен в это, он не может предотвратить повторную отправку транзакций.
- Нет асинхронного механизма для написания и подтверждения транзакций. Транзакции записываются только при подтверждении, а скорость записи ограничена скоростью подтверждения.

Протокол Vite устанавливает более совершенную асинхронную модель: сначала транзакция разделяется на транзакционную пару на основе модели «запрос-ответ», будь то передача или вызов контракта, и транзакция успешно запускается, когда транзакция запроса записывается в леджер. Кроме того, запись и подтверждение транзакции также асинхронны. Транзакции могут быть записаны в DAG учетную запись Vite и не будут заблокированы процессом подтверждения. Подтверждение транзакции выполняется через снэпшот цепь, а проведение снэпшота также асинхронно.

Это типичная модель производителя - потребителя. В жизненном цикле транзакции, независимо от того, как изменяется скорость производства в восходящем потоке, нисходящий поток может иметь дело с транзакциями с постоянной скоростью, чтобы полностью использовать ресурсы платформы и повысить пропускную способность системы.

5 Виртуальная машина

5.1 Совместимость с EVM

В настоящее время в области Ethereum есть много разработчиков, в Ethereum применяются многие смарт-контракты, написанные на языке Solidity и развернутые на EVM. Поэтому мы решили обеспечить совместимость с EVM на виртуальной машине Vite, а исходная семантика в большинстве наборов команд EVM сохранены в Vite. Так как структура учетной записи Vite и определение транзакции отличаются от Ethereum, семантику некоторых инструкций EVM необходимо переопределить, например, набор инструкций для получения информации о блоке. Подробные семантические различия упоминаются в приложении A.

Среди них наибольшее различие заключается в семантике сообщений. Мы поговорим подробно об этом далее.

5.2 Событие

В протоколе Ethereum транзакция или сообщение могут влиять на состояние нескольких учетных записей. Например, транзакция вызова контракта может привести к изменению статуса нескольких учетных записей контрактов для изменения в одно и то же время через сообщение. Эти изменения происходят либо в одно и то же время, либо вообще не происходят. Таким образом, транзакция в Ethereum на самом деле является своего рода жесткой транзакцией, которая удовлетворяет характеристикам ACID (Atomicity, Consistency, Isolation, Durability - атомность, согласованность, изоляция, долговечность) [24], что также является важной причиной недостатка расширяемости в Ethereum.

Основываясь на соображениях масштабируемости и производительности, Vite принял окончательную схему согласованности, удовлетворяющую семантике BASE (Basically Available, Soft state, Eventual consistency – базовая доступность, гибкое состояние, возможная согласованность) [25]. В частности, мы разрабатываем Vite как Event-Driven Architecture – Управляемую событиями Архитектуру (EDA) [26]. Каждый смарт-контракт считается независимой службой, а сообщения могут передаваться между контрактами, но никакое условие не разделяется.

Поэтому в EVM Vite нам нужно отменить семантику синхронных вызовов функций через контракты и разрешать обмен сообщениями между контрактами. EVM-командами, приведящими в действие, в основном являются **CALL** и **STATICCALL.** В Vite EVM, эти две инструкции не могут немедленно выполняться и не могут возвращать результат вызова. Они только генерируют транзакцию запроса для записи в леджер. Поэтому в Vite семантика функции вызова не будет включена в эту инструкцию, а скорее будет отправляться сообщением в учетную запись.

5.3 Язык смарт-контрактов

Ethereum предоставляет полный язык программирования Тьюринга Solidity для разработки смарт-контрактов. Чтобы поддерживать асинхронную семантику, мы расширили Solidity и определили набор синтаксиса в коммуникации через сообщения. Расширенная версия Solidity называется Solidity++.

Большая часть синтаксиса Solidity поддерживается Solidity ++, но не включает функции вызова вне контракта. Разработчик может определять сообщения через ключевое слово *message* и определять процессор сообщений (MessageHandler) через ключевое слово *on* для реализации функции межконтрактной связи.

Например, контракт A должен вызвать метод add () в контракте B для обновления его состояния на основе возвращаемого значения. В Solidity он может быть реализован вызовом функции. Код выглядит следующим образом:

```
pragma solidity ^0.4.0;

contract B {
    function add(uint a, uint b) returns
    (uint ret) {
        return a + b;
    }
}

contract A {
    uint total;

    function invoker(address addr, uint a,
    uint b) {
        // message call to A.add()
        uint sum = B(addr).add(a, b);
        // use the return value
        if (sum > 10) {
            total += sum;
        }
    }
}
```

В Solidity++ вызов функции codeuint sum=B(addr).add(a, b); более недействителен; вместо этого контракт A и контракт B коммуницируют асинхронно, через отправку сообщений друг другу. Код выглядит следующим образом:

```
pragma solidity++ ^0.1.0;

contract B {
    message Add(uint a, uint b);
    message Sum(uint sum);

    Add.on {
        // read message
        uint a = msg.data.a;
        uint b = msg.data.b;
        address sender = msg.sender;
        // do things
        uint sum = a + b;
        // send message to return result
        send(sender, Sum(sum));
    }
}

contract A {
    uint total;

    function invoker(address addr, uint a,
    uint b) {
        // message call to B
        send(addr, Add(a, b))
        // you can do anything after sending
        // a message other than using the
        // return value
    }
    Sum.on {
        // get return data from message
        uint sum = msg.data.sum;
        // use the return data
        if (sum > 10) {
            total += sum;
        }
    }
}
```

В первой строке кода pragma solidity ++ 0.1.0; указывает, что исходный код написан в Solidity++, но не будет скомпилирован непосредственно с помощью компилятора Solidity, чтобы избежать того, что скомпилированный код EVM не соответствовал ожидаемой семантике. Vite предоставит специализированный компилятор для компиляции Solidity++. Этот компилятор частично совместимый: если нет кода Solidity, который конфликтует с семантикой Vite, можно скомпилировать напрямую, иначе будет сообщена ошибка. Например, синтаксис местных функций вызова, перевода на другие учетные записи останутся совместимыми; получение возвращаемого значения кросс-контрактной функции вызова, а также денежной единицы **ether** не будет компилироваться.

В контракте A, когда вызывается функция invoker, сообщение «Add» будет отправлено в контракт B, который является асинхронным, и результат не будет возвращен немедленно. Поэтому, необходимо определить процессор сообщений в A, используя ключевое слово on для получения возвратного результата, и обновить состояние.

В контракте B отслеживается сообщение Add. После обработки, сообщение Sum отправляется отправителю сообщения Add, чтобы вернуть результат.

Сообщения в Solidity++ будут скомпилированы в CALL инструкции и транзакция запроса будут добавлены в леджер. В Vite, леджеры служат промежуточным программным обеспечением для асинхронной коммуникации между контрактами. Это обеспечивает надежное хранение сообщений и предотвращение дублирования. Несколько сообщений, отправленных к контракту через тот же контракт, могут гарантировать FIFO (First In First Out – Первый пришёл, первый вышел); сообщения, отправленные через разные контракты на тот же контракт не гарантируют FIFO.

Следует отметить, что события в Solidity (Event) и сообщения в Solidity ++ - это не одно и то же понятие. События отправляются не напрямую через журнал EVM.

5.4 Стандартная библиотека

Разработчики, которые разрабатывают смарт-контракты на Ethereum, часто страдают из-за отсутствия стандартных библиотек в Solidity. Например, проверка цикла в протоколе Loopring должна выполняться вне цепочки, одна из важных причин заключается в том, что функция вычисления с плавающей запятой не предоставляется в Solidity, особенно n квадратный корень [1] [1] для чисел с плавающей запятой.

В EVM предварительно установленный контракт может быть вызван командой **DELEGATECALL** для реализации библиотечной функции. Ethereum также предоставляет несколько предварительно скомпилированных контрактов, в основном это несколько операций с хэшем. Но эти функции слишком просты для удовлетворения сложных потребностей приложения.

Поэтому мы предоставим серию стандартных библиотек в Solidity ++, таких как обработка строк, операции с числами с плавающей запятой, основные математические операции, контейнеры, сортировки и т. д.

Основываясь на соображениях производительности, эти стандартные библиотеки будут реализованы через локальное расширение (Native Extension), и большая часть операций встроена в локальный код Vite, и эта функция вызывается только через инструкцию **DELEGATECALL** в EVM-коде.

Стандартная библиотека может быть расширена по мере необходимости, но поскольку модель машины всей системы детерминирована, она не может предоставлять такие функции, как случайные числа. Подобно Ethereum, мы можем моделировать псевдослучайные числа через хэш снэпшот цепи.

5.5. Газ

Существует две основные функции для Gas в Ethereum, первая из которых заключается в количественной оценке вычислительных ресурсов и ресурсов хранения, потребляемых при выполнении кода EVM, а вторая заключается в том, чтобы остановить EVM-код. Согласно теории вычислимости, проблема остановки на машинах Тьюринга (Halting Problem on Turing Machines) является неоспоримой проблемой [27]. Это означает, что невозможно определить, можно ли остановить смарт-контракт после ограниченного выполнения путем анализа кода EVM.

Поэтому расчет газа в EVM также сохраняется в Vite. Однако в Vite отсутствует концепция цены на газ. Пользователи не покупают газ для обмена, с целью платы fee (оплаты), а через модель на основе квот (долей) для получения вычислительных ресурсов. Расчет квот будет подробно рассмотрен позднее в главе «Экономическая модель».

6 Экономическая модель

6.1 Нативный токен

Чтобы количественно вычислить ресурсы платформы, хранилища ресурсов и стимулировать запуск нод, Vite построил собственный токен ViteToken. Базовая единица токена - *vite*, наименьшая единица - *attov* , 1 vite = 10^{18} attov.

Снэпшот цепь - ключ к безопасности и производительности платформы Vite. Чтобы подтолкнуть ноду к участию в проверке транзакции, протокол Vite устанавливает вознаграждение для производства снэпшот блоков. Стимулирование форджинга (forging – кование) будет увеличивать ликвидность ViteToken'а и добавлять преимущества холдерам токена *vite*. Поэтому мы будем ограничивать инфляцию до 3% в год.

Напротив, когда пользователи выпускают новые токены, разворачивают смарт-контракты, регистрируют доменные имена VNS [1] и получают квоты ресурсов, им необходимо потреблять или закладывать ViteToken для снижения ликвидности.

При совместном действии этих двух факторов ликвидность *vite* может поддерживаться на здоровом уровне, и в то же время она способствует оптимизации распределения ресурсов системы.

6.2 Распределение ресурсов

Поскольку Vite является общей платформой для децентрализованных приложений, возможности смарт-контрактов, развернутых на ней, различаются, и каждый смарт-контракт имеет разные требования к пропускной способности и задержке. Даже для одного и того же смарт-контракта требования производительности различны на разных этапах.

В строении Ethereum каждая транзакция требует, чтобы была назначена цена на газ при запуске, чтобы конкурировать с другими транзакциями для записи в аккаунт. Это типичная модель торгов, которая может эффективно контролировать баланс между предложением и спросом. Однако пользователю трудно количественно оценить текущую ситуацию предложения и спроса и не может предсказать цену других конкурентов, поэтому легко происходит сбой рынка. Более того, ресурсы, конкурирующие за каждую заявку, направлены против одной транзакции, и нет соглашения о рациональном распределение ресурсов по размеру учетной записи.

6.2.1 Расчет квот

Мы приняли в Vite протокол распределения ресурсов на основе квот, что позволяет пользователям получать более высокие квоты ресурсов тремя способами:

- PoW вычисляется при инициировании транзакции;
- Заложенное количество *vite* на счете;
- уничтожать небольшое количество *vite* за один раз.

Конкретные квоты могут быть рассчитаны с помощью следующей формулы:

$$Q = Q_m * \left(\frac{2}{1 + \exp(-\rho * \xi^T} - 1 \right) \qquad (6)$$

Среди них Q_m - постоянная, представляющая верхний лимит одной учетной записи, которая связана с общей пропускной способностью системы и общим количеством учетных записей. $\xi = (\xi_d, \xi_s, \xi_f)$ - вектор, представляющий ценность пользователя для получения ресурса: ξ_d – трудность PoW, которую пользователь вычисляет при создании транзакции,

[1] Ссылка на главу 7.2 Сервис имен

ξ_s - это количество токенов *vite,* заложенных в аккаунте, и ξ_f - это единовременная стоимость, которую пользователь готов заплатить за увеличение квоты. Следует отметить, что ξ_f отличается от платы за обработку. Эти vite будут напрямую уничтожены, вместо того, чтобы выплачивать их майнерам.

В формуле вектор $\rho = (\rho_d, \rho_s, \rho_f)$ представляет собой вес трех способов получения квоты, то есть квота, полученная уничтожением 1 *vite*, эквивалентна заложенным ρ_s/ρ_f *vite*.

Из этой формулы видно, что если пользователь не закладывает *vite* и не платит единовременную стоимость, необходимо вычислить PoW, иначе не будет никаких квот для инициирования транзакции, данная мера может эффективно предотвращать спам-атаку (dust attack), а также защищать системные ресурсы от перенагрузки. В то же время эта формула является Логистической функцией. Относительно легко для пользователей получить более низкие квоты, тем самым уменьшая порог пользователей, которые пользуются системой редко; а пользователям, которые часто пользуются системой, необходимо инвестировать много ресурсов, чтобы получить более высокие квоты. Дополнительная плата, которую они платят, увеличивает прибыль для всех пользователей.

6.2.2 Количественная оценка ресурсов

Поскольку снэпшот цепь эквивалентна глобальным часам, мы может использовать ее для точной количественной оценки использования ресурсов учетной записью. В каждой транзакции Хэш снэпшот блока квотируется, высота снэпшот блока принимается как отметка времени транзакции. Поэтому, согласно разнице между двумя временными метками транзакций, мы может судить, является ли интервал между двумя транзакциями достаточно длинным.

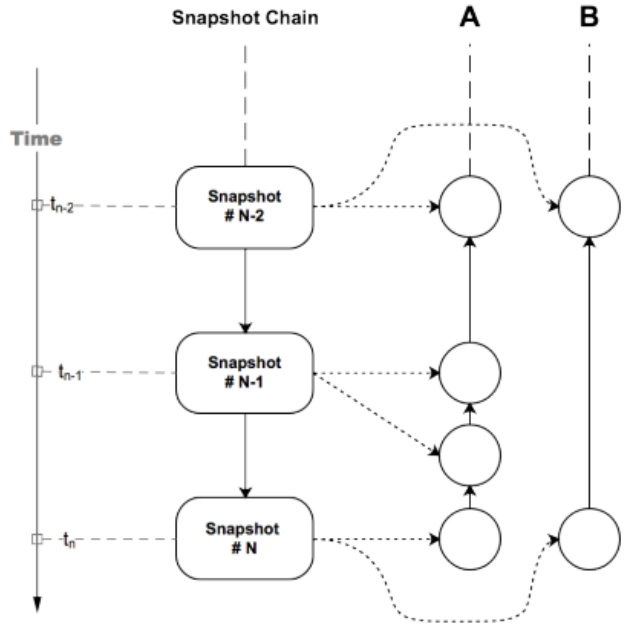

Рисунок 9: использование снэпшот цепочки в качестве глобальных часов

Как показано выше, учетная запись A сгенерировала 4 транзакции за 2 временных интервала, тогда как учетная запись B сгенерировала всего 2 транзакции. Следовательно, среднее TPS (количество транзакций в секунду) учетной записи A в этот период в 2 раза больше, чем у Б. Если это всего лишь транзакция передачи, достаточно среднего значения TPS для учетной записи.

Для смарт-контрактов каждый обмен данными имеет различное потребление ресурсов, поэтому для каждой транзакции необходимо суммировать газ, чтобы рассчитать среднее потребление ресурсов в течение определенного периода времени. Среднее потребление ресурсов последних k транзакций в цепочке учетной записи с высотой «n» вычисляется как:

$$Cost_k(T_n) = \frac{k * \sum_{i=n-k+1}^{n} gas_i}{timestamp_n - timestamp_{n-k+1} + 1} \quad (7)$$

Где, для транзакции T_n $timestamp_n$ - это метка времени транзакции, то есть высота снэпшот блока, к которому она относится; gas_n - это топливо, потребленное для транзакции.

При верификации транзакции нода определит, удовлетворяет ли квота условию: *Cost (T) ≤ Q*, и если оно не выполняется, транзакция будет отклонена. В этом случае пользователям необходимо переделать транзакцию, увеличив квоту через разовую плату, или подождать некоторое время.

6.2.3 Аренда квоты

Если пользователь обладает большим количеством токенов *vite*, но не нуждается в использовании такого количества ресурсных квот (долей ресурсов), он может сдавать в аренду свои квоты другим пользователям.

Система Vite поддерживает специальный тип транзакции для передачи права на использование квоты ресурса учетной записи. В этой транзакции может быть указано количество *vite*, которое может быть заложено, адрес получателя и продолжительность аренды. После подтверждения транзакции квота ресурсов, соответствующая сумме токена, будет включена в учетную запись правоприемника. Как только истечет срок аренды, квота будет рассчитана на счет переводчика. Вторая величина - количество времени аренды. Система будет преобразована в разницу высот снэпшот блока, поэтому могут быть некоторые отклонения.

Доход от сдачи аренды может быть получен пользователем. Система Vite предоставляет только транзакцию передачи квоты, а ценообразование и выплата за аренду могут быть достигнуты посредством стороннего смарт-контракта.

6.3. Выпуск активов

В дополнение к нативному токену ViteToken, Vite также поддерживает пользователей для выпуска своих токенов. Выпуск токенов может быть выполнен с помощью специальной транзакции - Mint Transaction. Целевой адрес Mint transaction - 0. В поле *data* транзакции параметры токена указаны следующим образом:

```
Mint: {
    name: "MyToken",
    totalSupply: 99999999900000000000000000,
    decimals: 18,
    owner: "0xa3c1f4...fa",
    symbol: "MYT"
```

Как только запрос будет принят сетью, токены *vite* включенные в транзакцию Mint transaction, будут вычтены из счета инициатора в качестве комиссии за транзакцию выпуска монет. Система записывает информацию о новом токене и назначает token_id для него. Все балансы вновь созданных токенов будут добавлены к адресу владельца (*owner*), то есть, учетная запись владельца является учетной записью генезиса токена.

6.4 Кросс-чейн протокол

В целях поддержки трансфера цифровых активов между цепочками и устранения «изолированных активов» («value island»), Vite спроектировал Vite Cross-chain Transfer Protocol (VCTP, Протокол трансфера между цепочками).

Для каждого актива, которому требуется кросс-чейн передача на целевую цепочку, созданный и циркулирующий токен внутри Vite соответствует необходимому токену извне, выступает в роли ваучера целевого токена, который называется ToT (Token of Token). Например, если вы хотите перенести *ether* в учетную запись Ethereum в Vite, вы можете выпустить ToT с идентификатором *ETH* в Vite, начальное количество TOT должно быть равным к общему количеству *ether*.

Для каждой целевой цепочки в Vite существует Шлюзовой Контракт (Gateway Contract) для поддержания связи отображения между транзакциями в Vite и транзакциями в целевой цепочке. В консенсусной группе, в которой находится контракт, нода, ответственная за генерирование блоков называется VCTP транслятором (VCTP Relay). Транслятор VCTP необходим для того, чтобы быть нодой в Vite и быть полной нодой целевой цепочки в одно и то же время, обрабатывать транзакции с обеих сторон. На целевой цепочке, нам также необходимо развернуть Шлюзовой Контракт Vite.

Прежде чем Транслятор VCTP начнет работать, соответствующий ToT в Vite должен быть переведен на шлюзовый контракт. После этого циркуляция TOT может контролироваться только шлюзом, и никто не сможет вмешаться в обеспечение соотношения 1:1 между ToT и целевым активом. В то же время активы в целевой цепочке контролируются шлюзовым контрактом Vite, и ни один пользователь не может использовать их, таким образом, ToT имеет полный обеспеченный резерв.

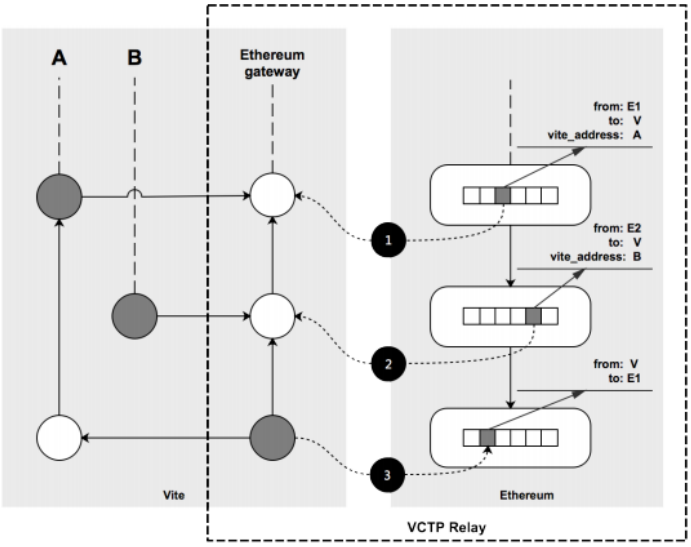

Рисунок 10: Кросс-чейн протокол

Вышеприведенное изображение является примером кросс-чейн трансфера активов между Vite и Ethereum. Когда пользователь Ethereum E1 хочет перенести токен из Ethereum в Vite, он должен отправить транзакцию на шлюзовой контракт Vite с адресом V, адрес пользователя A в Vite помещается в специальный параметр. Баланс трансфера будет заблокирован в учетной записи шлюзового контракта и станет частью резерва ToT. После обработки транзакции, нода Транслятора VCTP сгенерирует соответствующую учетную запись отправленной транзакции Vite, отправляя то же количество ToT на счет пользователя A в Vite. На рисунке 10 ①, ② соответственно отражает, что E1 и E2 перевели на средства в учетные записи A и B в Vite. Следует отметить, что если пользователь не укажет адрес в Vite при передаче, контракт отклонит транзакцию.

Обратный порядок показан в ③: когда пользователь A запускает трансфер с Vite на учетную запись Ethereum, транзакция будет отправлена на шлюзовой контракт Vite, который передает определенное количество ToT и указывает адрес приема E1 Ethereum в транзакции. Нода Транслятора VCTP будет генерировать соответствующий ответный блок на шлюзовом контракте Ethereum и создавать транзакцию Ethereum к шлюзовому контракту Vite в Ethereum. В Ethereum шлюзовой контракт Vite проверит, инициирована ли эта транзакция через доверенный транслятор VCTP, а затем такое же количество *ether* переносится из шлюзового контракта Vite на целевую учетную запись E1.

Все кросс-чейн ноды-трансляторы будут отслеживать целевую сеть, и они могут проверить, является ли каждая кросс-чейн транзакция правильной и достигает ли консенсуса в рамках консенсусной группы. Но консенсусная группа снэпшота не будет контролировать транзакции целевой цепочки, а также не будет проверять, является ли отображение между двумя цепями правильным. Если целевая сеть откатывается назад или в ней проводится хард форк, отображенные транзакции в системе Vite нельзя отменить;

аналогично, если кросс-чейн транзакция в Vite откатана назад, соответствующая транзакция целевой сети не может быть откатана в одно и то же время. Поэтому, когда создается кросс-чейн транзакции, необходимо иметь дело с откатом транзакций в логике контракта. В то же время, как описано в части 4.4, нам нужно установить параметр задержки (*delay*) для группы консенсусной группы кросс-чейн Транслятора.

6.5 Протокол Loopring

Протокол Loopring [1] является открытым протоколом для построения децентрализованной сети торговли активами. По сравнению с другими DEX решениями, протокол Loopring основан на многократном совпадении циклов, которое обеспечивает технологию двойной авторизации для предотвращения превентивных транзакций и является полностью открытым.

Мы создаем протокол Loopring в Vite, который способствует распространению цифровых активов в Vite, так что вся система активов может циркулировать. В этой системе активов, пользователи могут выпускать свои собственные цифровые активы, пересылать активы за пределы сети через VCTP и использовать протокол Loopring для обмена активами. Весь данный процесс может быть завершен внутри системы Vite, и он полностью децентрализован.

В Vite, Loopring Protocol Smart contract (LPSC, смарт-контракт протокола Loopring) является частью системы Vite. Авторизация для трансфера активов и многоступенчатая атомная защита поддерживаются в Vite. Транслятор Loopring также открыт для полной интеграции с собственной экосистемой.

Пользователи могут использовать *vite* для оплаты транзакций обмена активами. Токен *vite* также является токеном, зарабатываемым майнерами Loopring, которые выполняют цикл согласования на платформе Vite.

7 Другие усовершенствования

7.1 Планирование

В Ethereum, смарт-контракты управляются транзакциями, а выполнение контрактов может быть инициировано только через пользовательские инициирующие транзакции. В некоторых приложениях функция планирования по времени необходима для запуска выполнения контракта через часы.

В Ethereum эта функция достигается за счет сторонних контрактов [2], при этом производительность и безопасность не гарантируются. В Vite, мы добавляем функцию планирования времени при создании контракта. Пользователи могут создавать свою логику планирования в запланированном по времени контракте. Публичная консенсусная группа будет использовать снэпшот цепь в качестве часов и отправлять транзакцию запроса к целевому контракту в соответствии с определенной пользователем логикой планирования.

В Solidity++ есть специальное сообщение **Timer**. Пользователи могут настроить свою собственную логику планирования в код контракта через **Timer.on**.

7.2 Сервис имен

В Ethereum контракт будет генерировать адрес, для определения контракта при его развертывании. Существует две проблемы при идентификации контрактов с адресами:

• Адрес - это идентификатор с 20 байтами без значения. Это неудобно для пользователей и неудобно использовать.

• Контракты и адреса являются взаимно однозначными. Они не могут поддерживать перенаправление контрактов.

Чтобы решить эти две проблемы, разработчик Ethereum предоставил сторонний контракт **ENS**[3]. Однако в реальном сценарии использование сервиса имен будет очень частым, и использование сторонних контрактов не может гарантировать глобальную уникальность имени, поэтому мы создадим службу VNS (ViteName Service) в Vite.

Пользователи могут регистрировать набор имен, которые легко запомнить и разрешить их через фактический адрес через VNS. Имена организованы в виде доменных имен, таких как *vite.myname.mycontract*. Доменное имя верхнего уровня будет сохранено системой для определенных целей. Например, vite.xx представляет адрес Vite, а eth.xx представляет адрес Ethereum. Доменное имя второго уровня доступно для всех пользователей. Когда пользователь владеет доменным именем второго уровня, субдомен может быть произвольно расширен. Владелец доменного имени может в любой момент изменить адрес, назначенный доменным именем, поэтому эту функцию можно использовать для обновления контракта.

Длина имени домена не ограничена. В VNS хэш имени домена сохраняется. Целевой адрес может быть адресом не Vite менее 256 бит, который может использоваться для кросс-чейн взаимодействия.

Следует отметить, что VNS отличается от смарт-контрактной Спецификация пакета EIP190[4] в Ethereum. VNS - это служба разрешения имен, имя устанавливается во время выполнения, а правила разрешения могут быть динамически изменены; и EIP190 - это пакет управления спецификациями, место имени является статическим, и оно устанавливается во время компиляции.

7.3 Обновление контракта

Смарт-контракт Ethereum неизменяем. После развертывания его нельзя изменить. Даже если в контракте есть ошибка, он не может быть обновлен. Это очень неудобно для разработчиков и делает непрерывное повторение dApp очень трудным. Поэтому Vite должен предоставить схему поддержки обновления смарт-контракта.

В Vite процесс обновления контракта включает:

A. Развертывает новую версию контракта, чтобы наследовать статус первоначального контракта.

[2] Ethereum Alarm Clock - это сторонний контракт, используемый для планирования выполнения других контрактов, см. Http://www.ethereum-alarm-clock.com/

[3] Ethereum Name Service является сторонним контрактом, используемым для разрешения имени, см. Https://ens.domains/

[4] EIP190 Ethereum Smart Contract Packaging Specification, см. Https://github.com/ethereum/EIPs/issues/190

B. Указывает название контракта на новый адрес в VNS.

C. Удаляет старый контракт с помощью инструкции **SELFDESTRUCT**

Эти три этапа необходимо выполнить одновременно, а протокол Vite обеспечивает атомарность операции. Разработчикам необходимо убедиться, что старые данные контракта правильно обработаны в контракте в новую версию.

Следует отметить, что новый контракт не наследует адрес старого контракта. Если он указан по адресу, транзакция по-прежнему будет отправлена в старый контракт. Это связано с тем, что разные версии контрактов являются по существу двумя совершенно разными контрактами, независимо от того, могут ли они динамически изменяться или нет, в зависимости от семантики контрактов.

В системе Vite смарт-контракты фактически делятся на два типа: первый - это фон dApp, и его бизнес-логика; а второй - своего рода контракт, который отображает реальный мир. Первый тип эквивалентен фоновому сервису приложения, который необходимо обновлять с помощью апгрейдов; второй тип эквивалентен контракту, и как только он вступает в силу, никакие изменения не могут быть сделаны, в противном случае это является нарушением контракта. Для такого контракта, который не может быть изменен, он может быть выделен ключевым словом *static* в Solidity ++, например:

```
pragma solidity++ ^0.1.0;

static contract Pledge {
    // the contract that will never change
}
```

7.4 Обрезка блоков

В леджере любая транзакция является неизменной, и пользователи могут только добавлять новые транзакции в леджер, без изменения или удаления прошлых транзакций. Поэтому при работе системы леджеры будут становиться все больше и больше. Если новая нода, которая присоединяется к сети, хочет восстановить последний статус, ей придется начинать с начального блока и сохранить все прошлые транзакции. После работы системы в течение некоторого периода времени, пространство, занимаемое учетной записью, и время, затраченное на сохранение прошлых транзакций, станут неприемлемыми. Для системы с высокой пропускной способностью темп роста Vite будет намного выше, чем Bitcoin и Ethereum, поэтому необходимо предусмотреть метод обрезки блоков в леджерах.

Обрезка блоков относится к удалению прошлых транзакций, которые нельзя использовать в леджерах, и не влияет на работу транзакционной машины. Итак, какие транзакции можно безопасно удалить? Это зависит от того, в каком сценарии будет использоваться транзакция, в том числе:

- **Восстановление**. Основная роль транзакции - восстановить статус. Поскольку в Vite снэпшот цепь сохраняет снэпшот информацию статуса учетной записи, ноды могут восстановить состояние из снэпшот блока. Все транзакции перед *lastTransaction* в снэпшот блоке могут быть адаптированы при восстановлении состояния.

- **Проверка транзакций.** Чтобы проверить новую транзакцию, необходимо проверить предыдущую транзакцию обмена в цепочке учетной записи, и если это транзакция ответа, также необходимо проверить транзакцию соответствующего запроса. Поэтому в адаптированных учетных записях леджеров, по крайней мере одна последняя транзакция должна сохраняться в каждой цепочке учетной записи. Кроме того, все транзакции с открытым запросом не могут быть адаптированы, поскольку их хеши могут ссылаться на последующие транзакции ответа.

- **Вычисление квоты.** Если транзакция соответствует квоте, вычисленной путем оценки скользящего среднего из последних 10 источников транзакций, поэтому по крайней мере последние 9 транзакций должны быть сохранены в каждой цепочке аккаунта.

- **Запрос справки из истории.** Если нода должна запросить историю транзакций, то транзакция, участвующая в запросе, не будет адаптирована.

В соответствии с различными сценариями использования, каждая нода может выбрать несколько комбинаций из вышеупомянутой стратегии обрезки. Важно отметить, что обрезка включает транзакции в леджерах, в то время как снэпшот цепи должны быть сохранены. Кроме того, то, что записано в снепшот цепи, является хешем состояния контракта. Когда учетная запись обрезана, соответствующее состояние снэпшота должно быть сохранено.

Чтобы обеспечить целостность данных в Vite, нам необходимо сохранить некоторые «Полные ноды» в сети, чтобы сохранить все данные транзакций. Снэпшот ноды консенсусной группы являются полными нодами, и, кроме того, важные пользователи, такие как биржи, также могут стать полными нодами.

8 Управление

Для децентрализованной платформы приложений, эффективная система управления имеет важное значение для поддержания здоровой экосистемы. Эффективность и справедливость следует учитывать при разработке систем управления.

Система управления Vite делится на две части: on-chain и off-chain. On-chain - это механизм голосования на основе протокола, а off-chain - итерация самого протокола.

По механизму голосования оно разделен на два типа: глобальное голосование и местное голосование. Глобальное голосование основано на количестве *vite*, которым обладает пользователь, это необходимо, чтобы подсчитать права как вес голоса. Глобальное голосование в основном используется для избрания ноды прокси-сервера консенсусной группы снэпшота. Местное голосование нацелено на контракт. Когда контракт развертывается, в качестве основы для голосования назначается токен. Он может использоваться для выбора узлов-агентов консенсусной группы, в которой расположен контракт.

В дополнение к проверке транзакции, прокси-нода консенсусной группы снепшота имеет право выбрать, следует ли выполнять обновление без совместимости с системой Vite.. Делегированная консенсусная группа прокси-ноды имеет право решать, разрешать ли контракту быть обновленным, чтобы избежать потенциальных рисков, связанных с расширением контрактов. Нода-агент используется для обновления полномочий принятия решений от имени пользователей в целях повышения эффективности принятия решений и предотвращения провала от принятия решений из-за недостаточного участия в голосовании. Сами эти прокси-ноды также ограничены консенсусным протоколом. Только если большинство[5] нод-агентов примет решение, только тогда произойдет обновление. Если эти агенты не выполняют свои полномочия в соответствии с ожиданиями пользователя, пользователи могут также отменять их доверенную квалификацию путем голосования.

Управление off-chain осуществляется сообществом. Любой участник сообщества Vite может предложить план улучшения для протокола Vite или связанных с ним систем, который называется VEP (Vite Enhancement Proposal – Предложение по Усовершенствованию Vite). VEP может широко обсуждаться в сообществе, а решение о внедрении решения принимается участниками экосистемы Vite. Будет ли протокол обновляться для реализации VEP, в конечном итоге будет решаться нодой-агентом. Конечно, когда различия велики, вы также можете начать раунд голосования on-chain для сбора мнений пользователей широкого круга, а прокси-нода будет решать, следует ли обновлять по результатам голосования.

Хотя некоторые участники Vite могут не иметь достаточно токенов *vite*, чтобы голосовать за свое мнение. Но они могут свободно представлять VEP и полностью выражать свои взгляды. Пользователи, имеющие право голоса, должны в полной мере учитывать здоровье всей экосистемы для своих собственных прав Vite и, следовательно, серьезно относиться к мнениям всех участников экосистемы.

9 Задачи в будущем

Проверка транзакций в снэпшот цепочках является основным узким местом производительности системы. Поскольку Vite использует асинхронный дизайн и структуру учетной записи DAG, проверка транзакции может выполняться параллельно. Однако из-за зависимости между транзакциями разных аккаунтов степень параллелизма будет сильно ограничена. Как улучшить параллельность проверки транзакций или принять распределенную стратегию проверки - это станет важным направлением для будущей оптимизации.

Некоторые недостатки существуют и в текущем консенсусном алгоритме HDPoS. Это также направление оптимизации для улучшения консенсусного алгоритма или совместимость с большим количеством консенсусных алгоритмов в делегированной консенсусной группе.

Кроме того, оптимизация виртуальной машины также очень важна для снижения задержки системы и повышения пропускной способности системы. Из-за простой конструкции EVM и упрощения набора инструкций может потребоваться для разработки более мощной виртуальной машины в будущем и определения языка программирования смарт-контрактов с большей способностью описывать и с меньшим количеством уязвимостей безопасности.

Наконец, помимо основного направления развития Vite, создание вспомогательных объектов, поддерживающих экологическое развитие, также является важной темой. В дополнение к поддержке SDK для разработчиков dApp, в архитектуре экосистемы dApp много работы. Например, вы можете создать движок dApplet на основе HTML5 в приложении мобильного кошелька Vite, что позволит разработчикам разрабатывать и публиковать dApp по низкой цене.

10 Резюме

По сравнению с другими аналогичными проектами, Vite обладает следующими характеристиками:

- **Высокая пропускная способность**. Vite использует структуру DAG леджера, ортогональная транзакция может быть записана параллельно цепи; кроме того, несколько консенсусных групп не зависят друг от друга в консенсусном алгоритме HDPoS и могут работать параллельно; самое главное, что межконтрактная связь Vite основана на асинхронной модели сообщения. Все это полезно для повышения пропускной способности системы.

- **Низкая задержка.** Vite использует консенсусный алгоритм HDPoS для завершения производственного блока совместно с прокси-нодами. Нет необходимости вычислять PoW, а интервал блока можно сократить до 1 секунды, что выгодно уменьшить задержку подтверждения транзакции.

- **Масштабируемость.** Чтобы удовлетворить требованиям к масштабируемости, Vite сделала единый лимит свободы для транзакций, группируя транзакции в регистре в соответствии с размером учетной записи, позволяя заполнять блок разных учетных записей различными узлами и удаляя ACID по контрактным вызовам. Семантика, измененная на семантику BASE на основе сообщений. Таким образом, узлу больше не нужно сохранять все мировое состояние, а данные хранятся на всей распределенной сети (шардинг).

- **Удобство использования.** Улучшение удобства использования Vite включает в себя: предоставление поддержки библиотеки стандартов в Solidity++, предназначенной для обработки синтаксиса сообщений, планирования контрактов по времени, сервиса имен VNS, поддержки обновления контрактов и т.д.

- **Циркуляция активов.** Vite поддерживает выпуск цифровых активов, кросс-чейн трансфер активов, обмен токенами на основе протокола Loopring и т.д., таким образом формируя завершенную систему активов. С точки зрения пользователя, Vite является полнофункциональным децентрализованным обменом.

[5] согласно протоколу DPoS, действительным большинством является 2/3 от общего числа узлов-агентов.

- **Экономика.** Поскольку Vite использует модель распределения ресурсов на основе квот, простые пользователи (имеющие низкий вес голоса), которые не торгуют часто, не должны платить высокие комиссионные или газовые сборы. Пользователи могут выбирать различные способы изменения расчета. Дополнительная квота также может быть передана другим пользователям через соглашение о сдаче в аренду квот для повышения эффективности использования системных ресурсов.

11 Благодарность

С уважением, мы хотели бы поблагодарить наших консультантов за их руководство и помощь в этой статье. Особенно Мы хотели бы оценить вклад команды Loopring и сообщества Loopring в этот проект.

References

[1] Daniel Wang, Jay Zhou, Alex Wang, and Matthew Finestone. Loopring: A decentralized token exchange protocol. URL https://github.com/Loopring/whitepaper/blob/master/en_whitepaper.pdf.

[2] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[3] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 151, 2014.

[4] Vitalik Buterin. Ethereum: a next generation smart contract and decentralized application platform (2013). URL http://ethereum.org/ethereum.html, 2017.

[5] Chris Dannen. Introducing Ethereum and Solidity. Springer, 2017.

[6] Jae Kwon and Ethan Buchman. Cosmos a network of distributed ledgers. URL https://cosmos.network/whitepaper.

[7] Anonymous. aelf - a multi-chain parallel computing blockchain framework. URL https://grid.hoopox.com/aelf_whitepaper_en.pdf, 2018.

[8] Anton Churyumov. Byteball: A decentralized system for storage and transfer of value. URLhttps://byteball.org/Byteball.pdf.

[9] Serguei Popov. The tangle. URL https://iota.org/IOTA_Whitepaper.pdf.

[10] Colin LeMahieu. Raiblocks: A feeless distributed cryptocurrency network. URL https://raiblocks.net/media/RaiBlocks_Whitepaper__English.pdf.

[11] Anonymous. Delegated proof-of-stake consensus, a robust and flexible consensus protocol. URL https://bitshares.org/technology/delegated-proof-of-stake-consensus/

[12] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semisynchronous proof-of-stake blockchain. URL https://eprint.iacr.org/2017/573.pdf, 2017.

[13] Anonymous. Eos.io technical white paper v2. URL https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md.

[14] Dai Patrick, Neil Mahi, Jordan Earls, and Alex Norta. Smart-contract value-transfer protocols on a distributed mobile application platform. URL https://qtum.org/uploads/files/cf6d69348ca50dd985b60425ccf282f3.pdf, 2017.

[15] Ed Eykholt, Lucius Meredith, and Joseph Denman. Rchain platform architecture. URL http://rchainarchitecture.readthedocs.io/en/latest/.

[16] Anonymous. Neo white paper a distributed network for the smart economy. URL http://docs.neo.org/enus/index.html.

[17] Anonymous. Byzantine consensus algorithm. URL https://github.com/tendermint/tendermint/wiki/ByzantineConsensus-Algorithm.

[18] Shapiro Marc, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. URL https://hal.inria.fr/inria-00609399v1, 2011.

[19] Deshpande and Jayant V. On continuity of a partial order. Proc. Amer. Math. Soc. 19 (1968), 383-386, 1968.

[20] Weisstein and Eric W. Hasse diagram. URL http://mathworld.wolfram.com/HasseDiagram.html.

[21] Chunming Liu. Snapshot chain: An improvement on block-lattice. URL https://medium.com/@chunming.vite/snapshot-chain-an-improvement-on-block-lattice-561aaabd1a2b.

[22] Anonymous. Problems. URL https://github.com/ethereum/wiki/wiki/Problems.

[23] Dantheman. Dpos consensus algorithm - the missing white paper. URL https://steemit.com/dpos/@dantheman/dposconsensus-algorithm-this-missing-white-paper.

[24] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. ACM Comput. Surv., 15(4):287–317, December 1983.

[25] Dan Pritchett. Base: An acid alternative. Queue, 6(3):48–55, May 2008.

[26] Jeff Hanson. Event-driven services in soa. URL https://www.javaworld.com/article/2072262/soa/event-drivenservices-in-soa.html.

[27] Michael Sipser. Introduction to the Theory of Computation. PWS Publishing, second edition, 2006

Приложения

Приложение A – Набор команд EVM

A.0.1 0s: Стоп и набор алгебраических команд

Номер	Слова	POP	PUSH	Семантика в EVM	Семантика в Vite
0x00	STOP	0	0	Остановить выполнение	Та же семантика
0x01	ADD	2	1	Добавьте два операнда.	Та же семантика
0x02	MUL	2	1	Умножение двух операндов.	Та же семантика
0x03	SUB	2	1	Вычитание двух операндов.	Та же семантика
0x04	DIV	2	1	Разделение двух операндов Если делитель равен 0, затем возвращает 0	Та же семантика
0x05	SDIV	2	1	Разделен с символом	Та же семантика
0x06	MOD	2	1	Операция Модуль.	Та же семантика
0x07	SMOD	2	1	Модуль с символом.	Та же семантика
0x08	ADDMOD	3	1	Добавление первых двух операндов и модуль с 3-м	Та же семантика
0x09	MULMOD	3	1	Умножение первых двух операндов и модуль с 3-м	Та же семантика
0x0a	EXP	2	1	Квадрат двух операндов.	Та же семантика
0x0b	SIGNEXTEND	2	1	Расширение символа.	Та же семантика

A.0.2 10s: Сравнение и набор битных команд

Номер	Слова	POP	PUSH	Семантика в EVM	Семантика в Vite
0x10	LT	2	1	Меньше чем.	Та же семантика
0x11	GT	2	1	Больше чем.	Та же семантика
0x12	SLT	2	1	Меньше чем с символом.	Та же семантика
0x13	SGT	2	1	Больше чем с символом.	Та же семантика
0x14	EQ	2	1	Равно	Та же семантика
0x15	ISZERO	1	1	Если это 0.	Та же семантика
0x16	AND	2	1	И	Та же семантика
0x17	OR	2	1	ИЛИ	Та же семантика
0x18	XOR	2	1	исключающее ИЛИ	Та же семантика
0x19	NOT	1	1	Ни	Та же семантика
0x1a	BYTE	2	1	Взять один байт из вторых операндов.	Та же семантика

A.0.3 20s: Набор команд SHA3

Номер	Слова	POP	PUSH	Семантика в EVM	Семантика в Vite
0x20	SHA3	2	1	Вычислить Keccak-256 хэш	Та же семантика

A.0.4 30s: Набор команд для получения информации из среды

Номер	Слова	POP	PUSH	Семантика в EVM	Семантика в Vite
0x30	ADDRESS	0	1	Получить адрес текущего счета	Та же семантика
0x31	BALANCE	1	1	Получить баланс счета	Та же семантика. возвращается vite баланс счета
0x32	ORIGIN	0	1	Получить адрес отправителя исходной транзакции	Разная семантика, Vite не поддерживает причинно-следственную связь между внутренней транзакцией и транзакцией пользователя.
0x33	CALLER	0	1	Получить адрес вызывающего.	Та же семантика
0x34	CALLVALUE	0	1	Получать переведенную сумму в вызванной транзакции	Та же семантика
0x35	CALLDATALOAD	1	1	Получить параметр в этом вызове	Та же семантика
0x36	CALLDATASIZE	0	1	Получите размер параметров данных в этом вызове.	Та же семантика
0x37	CALLDATACOPY	3	0	Скопировать данные параметров в память.	Та же семантика
0x38	CODESIZE	0	1	Получить размер текущего кода в текущей среде.	Та же семантика
0x39	CODECOPY	3	0	Скопировать текущий код в текущую среду в память.	Та же семантика
0x3a	GASPRICE	0	1	Получить цену на газ в текущей среде	Разная семантика Всегда возвращает 0
0x3b	EXTCODESIZE	1	1	Получить размер кода учетной записи	Та же семантика
0x3c	EXTCODECOPY	4	0	Скопировать код учетной записи в память	Та же семантика
0x3d	RETURNDATASIZE	0	1	Получить размер данных, полученных от предыдущего вызова.	Та же семантика
0x3e	RETURNDATACOPY	3	0	Скопировать обратно полученные данные в память.	Та же семантика

A.0.5 40s: Команды для получения информации о блоке

Номер	Слова	POP	PUSH	Семантика в EVM	Семантика в Vite
0x40	BLOCKHASH	1	1	Получить хэш блока	Разная семантика возвращает хэш соответствующего снэпшот блока
0x41	COINBASE	0	1	Получить адрес майнера-получателя выгоды в текущем блоке	Разная семантика Возвращает 0 всегда
0x42	TIMESTAMP	0	1	Возвращает временную метку текущего блока.	Разная семантика Возвращает 0 всегда
0x43	NUMBER	0	1	Возвращает число или текущий блок	Разная семантика Возвращает количество блоков транзакций ответа в цепочке аккаунта
0x44	DIFFICULTY	0	1	Возвращает трудность блока	Разная семантика Возвращает 0 всегда
0x45	GASLIMIT	0	1	Возвращает лимита газа блока	Разная семантика Возвращает 0 всегда

A.0.6 50s: Набор команд для управления потоком стека хранилища памяти

Номер	Слова	POP	PUSH	Семантика в EVM	Семантика в Vite
0x50	POP	1	0	Вытащить данные из верхней части стека	Та же семантика
0x51	MLOAD	1	1	загрузить слово из памяти.	Та же семантика
0x52	MSTORE	2	0	Сохранить слово в памяти	Та же семантика
0x53	MSTORE8	2	0	Сохранение байта в память	Та же семантика
0x54	SLOAD	1	1	Загрузить слово из хранилища	Та же семантика
0x55	SSTORE	2	0	Сохранить слово в хранилище	Та же семантика
0x56	JUMP	1	0	Инструкции по переходу.	Та же семантика
0x57	JUMPI	2	0	Инструкции перехода с условием	Та же семантика
0x58	PC	0	1	Получить значение счетчика программ.	Та же семантика
0x59	MSIZE	0	1	Получить размер памяти.	Та же семантика
0x5a	GAS	0	1	Получение доступного газа.	Та же семантика Возвращает 0 всегда
0x5b	JUMPDEST	0	0	Отметить пункт назначения прыжка.	Та же семантика

A.0.7 60s и 70s: Команды по управлению стеком

Номер	Слова	POP	PUSH	Семантика в EVM	Семантика в Vite
0x60	PUSH1	0	1	Вставить один байт объекта в верхнюю часть стека	Та же семантика
0x61	PUSH2	0	1	Вставить два байта объекта в верхнюю часть стека	Та же семантика
…	…	…	…	…	…
0x7f	PUSH32	0	1	Вставить 32 байта объекта (целое слово) в верхнюю часть стека	Та же семантика

A.0.8 80s: Команды дублирования

Номер	Слова	POP	PUSH	Семантика в EVM	Семантика в Vite
0x80	DUP1	1	2	Дублировать 1-й объект и вставьте его в верхнюю часть стека.	Та же семантика
0x81	DUP2	2	3	Дублировать 2-й объект и вставьте его в верхнюю часть стека.	Та же семантика
…	…	…	…	…	…
0x8f	DUP16	16	17	Дублировать 16-й объект и вставьте его в верхнюю часть стека.	Та же семантика

A.0.9 90s: Команды замена

Номер	Слова	POP	PUSH	Семантика в EVM	Семантика в Vite
0x90	SWAP1	2	2	Заменить 1-й и 2-й объекты в стеке	Та же семантика
0x91	SWAP2	3	3	Заменить 1-й и 3-й объекты в стеке.	Та же семантика
…	…	…	…	…	…
0x9f	SWAP16	17	17	Заменить 1-й и 17-й объекты в стеке	Та же семантика

A.0.10 a0s: Команды логирования

Номер	Слова	POP	PUSH	Семантика в EVM	Семантика в Vite
0xa0	LOG0	2	0	Расширить запись журнала, без схемы.	Та же семантика
0xa1	LOG1	3	0	Расширить запись журнала, 1 схема.	Та же семантика
…	…	…	…	…	…
0xa4	LOG4	6	0	Расширить запись журнала, 4 схемы.	Та же семантика

A.0.11 f0s: Команды эксплуатации системы

Номер	Слова	POP	PUSH	Семантика в EVM	Семантика в Vite
0xf0	CREATE	3	1	Создать новый контракт.	Та же семантика
0xf1	CALL	7	1	Вызвать другой контракт.	Другая семантика указать отправку сообщения в учетную запись Возвращенная доля равна 0 всегда.
0xf2	CALLCODE	7	1	Вызов кода другого контракта Изменение статуса учетной записи	Та же семантика
0xf3	RETURN	2	0	Остановить выполнение и вернуть значение	Та же семантика
0xf4	DELEGATECALL	6	1	Вызов кода другого контракта, изменение контракта, изменение статуса текущего счета, сохранение оригинальной информации о транзакции.	Та же семантика
0xfa	STATICCALL	6	1	Вызов другого контракта, не позволяет изменять статус.	Другая семантика представляет собой отправку сообщения в контракт, не меняет статус целевого контракта. Возвращает 0 всегда необходимый результат Отправка другого сообщения через целевой контракт и возврат.
0xfd	REVERT	2	0	Остановить выполнение, восстановить статус и вернуть значение	Та же семантика нет семантики возврата оставшегося газа.
0xfe	INVALID			неверные инструкции.	Та же семантика
0xff	SELFDESTRUCT	1	0	Прекратить выполнение, установить контракт как ожидающий удаления вернуть весь баланс.	Та же семантика