Vite : Une plate-forme d'application décentralisée et asynchrone avec haute performance

Résumé

Vite est une plate-forme d'application décentralisée et généralisée qui répond au mieux aux exigences des applications industrielles pour une haute capacité de chargement et de déchargement, une faible latence et une évolutivité tout en prenant en compte la sécurité. Vite utilise la structure du grand livre DAG et les transactions dans les ledgers sont regroupées par comptes. La structure Snapshot Chain de Vite peut compenser le manque de sécurité du ledger DAG. L'algorithme de consensus HDPoS, par lequel l'écriture et la confirmation des transactions sont asynchrones, fournit des performances et une évolutivité élevées. La VM Vite est compatible avec EVM, et le langage de contrat intelligent étendu de Solidity, fournissant une capacité de description plus puissante. En outre, une amélioration importante dans la conception de Vite est l'adoption d'une architecture événementielle asynchrone, qui transmet des informations via des messages entre les contrats intelligents, ce qui améliore considérablement le débit et l'évolutivité du système. En plus des jetons natifs intégrés, Vite aide également les utilisateurs à émettre leurs propres ressources numériques, et fournit également un transfert et un échange de valeur croisés basés sur le protocole Loopring. [1]. Vite réalise l'allocation des ressources par quotas, et les petits utilisateurs n'ont pas besoin de payer les frais de transaction. Vite prend également en charge la planification des contrats, le service de nommer, la mise à jour des contrats, l'élagage des blocs et d'autres fonctionnalités.

1 Introduction

1.1 Définition

Vite est une plate-forme d'App universelle qui peut prendre en charge un ensemble de contrats intelligents, dont chacun est une machine d'états avec un état indépendant et une logique opérationnelle différente, qui peut communiquer par l'envoi de messages.

En général, le système est une machine d'état transactionnelle. L'état du système $\boldsymbol{s} \in \boldsymbol{S}$, également connu comme l'état mondial, est composé de l'état de chaque compte indépendant. Un événement qui provoque des changements dans l'état du compte s'appelle des transactions. La définition plus formalisée est la suivante :

Définition 1.1 (Machine d'état transactionnelle) *Une machine d'état transactionnelle est une 4-tuple :* $(\boldsymbol{T}, \boldsymbol{S}, \boldsymbol{g}, \boldsymbol{\delta})$*,où* $\boldsymbol{T}$ *est un ensemble de transactions,*$\boldsymbol{S}$ *est un ensemble d'états,*$\boldsymbol{g} \in \boldsymbol{S}$ *est l'état initial, aussi connu sous le nom* `bloc de genèse`*,* $\boldsymbol{\delta} : \boldsymbol{S} \times \boldsymbol{T} \to \boldsymbol{S}$ *est une fonction de transition d'état.*

La sémantique de cette machine d'état transactionnelle est un système de transition discret, qui est défini comme suit :

Définition 1.2 (Sémantique de machine d'état transactionnelle) *La sémantique d'une machine d'état transactionnelle* $(\boldsymbol{T}, \boldsymbol{S}, \boldsymbol{s_0}, \boldsymbol{\delta})$ *est un système de transition discret :* $(\boldsymbol{S}, \boldsymbol{s_0}, \to)$*.*$\to \in \boldsymbol{S} \times \boldsymbol{S}$ *est une relation de transition.*

En même temps, la plate-forme d'application décentralisée est un système distribué avec une cohérence finale. Grâce à un algorithme de consensus, l'état final peut être atteint entre les nuds. Dans les scénarios réalistes, ce qui est stocké dans l'état des contrats intelligents est un ensemble de données complètes dans une application décentralisée, avec un volume important et ne peut pas être transmis entre les nuds. Par conséquent, les nuds doivent transférer un ensemble de transactions pour obtenir la cohérence de l'état final. Nous organisons un tel groupe de transactions dans une structure de données spécifique, souvent appelée ńlivreż.

Définition 1.3 (Ledger) *Livre est composé d'un ensemble de transactions, avec un type de données abstrait construit récursivement. Il est défini comme suit :*

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

Parmi eux,$T_t \in 2^T$*,représentant un ensemble de transactions,* $\Gamma \in 2^T \to L$*,représente une fonction de construction d'un livre par un ensemble de transactions,*L *est un ensemble de livres,*$+ : L \times L \to L$*,représentant l'opération de fusion de deux sous-livres en un.*

Il faut noter que dans de tels systèmes, les livres de comptes sont généralement utilisés pour représenter un groupe transactions, plutôt qu'un état. Dans Bitcoin cite nakamoto2008bitcoin et Ethereum cite wood2014ethereum, le livre de comptes est une structure de chaîne de blocs, dans laquelle les transactions sont globalement mis en ordre.

Pour modifier une transaction dans le livre de comptes, nous devons reconstruire un sous-livre dans le livre de comptes, augmentant ainsi le coût de la falsification de la transaction.

Selon le même groupe de transactions, les livres différents et valides peuvent être construits, mais ils représentent des transactions avec ordres différents et ils peuvent causer le système entrer dans un état différent. Quand cela arrive, il est généralement appelé "fourchette".

Définition 1.4 (Fourchette) *Supposons* $T_t, T_t' \in 2^T$,$T_t \subseteq T_t'$. *si* $l = \Gamma_1(T_t)$,$l' = \Gamma_2(T_t')$,*et condition* $l \preceq l'$ *n'est pas satisfait,nous pouvons nommer* l *et* l' *sont livres fourchette.* $\preceq$ *représente relation préfixe.*

Selon la sémantique de la machine d'état transactionnelle, nous pouvons facilement prouver qu'à partir d'un état initial, si le livre n'est pas à bifurcation, chaque nud finira par entrer dans le même état. Donc, si un livre fourchu est reçu, va-t-il certainement entrer dans un état différent ? Cela dépend de la logique inhérente à la transaction dans le livre et de la manière dont les livres organisent les ordres partiels entre les transactions. En réalité, il y a souvent des transactions qui satisfont aux lois commutatives, mais à cause du problème de la conception des comptes, elles provoquent fréquemment des fourchettes. Lorsque le système démarre à partir d'un état initial, reçoit deux livres à bifurcation et finit dans le même état, nous appelons ces deux livres un faux livre fourchu.

Définition 1.5 (Fause Fourchette) *Etat Initial* $s_0 \in S$,*livre* $l_1, l_2 \in L$,$s_0 \xrightarrow{l_1} s_1, s_0 \xrightarrow{l_2} s_2$. *si* $l_1 \neq l_2$,*et* $s_1 = s_2$,*on appelle les deux livres* l_1, l_2 *comme livres de fause fourchette.*

Un livre bien conçu devrait minimiser la probabilité de fausse fourchette

Lorsque la fourchette se produit, chaque nud doit en choisir un dans plusieurs livres fourchu. Afin d'assurer la cohérence de l'état, les nuds doivent utiliser le même algorithme pour compléter la sélection. Cet algorithme est appelé l'algorithme de consensus.

Définition 1.6 (Algorithme de Consensus) *L'algorithme de consensus est une fonction qui reçoit un ensemble de livres et renvoie qu'un seul livre :*

$$\Phi : 2^L \to L$$

L'algorithme de consensus est la partie importante du design du système. Un bon algorithme de consensus doit posséder une vitesse de convergence élevée pour réduire l'influence du consensus dans différentes fourchette, et avoir une grande capacité à se prémunir contre les attaques malveillantes.

1.2 Progrès en cours

L'Ethereum [2]est parmi les premiers dans la réalisation d'un tel système. Dans la conception d'Ethereum, la définition de l'état mondial est : $S = \Sigma^A$, un mapping à partir du compte $a \in A$ et l'état de ce compte $\sigma_a \in \Sigma$.Par conséquent, tout état dans la machine d'état de l'Ethereum est global, ce qui signifie qu'un nud peut atteindre le statut de n'importe quel compte à tout moment.

La fonction de transition d'état *delta* d'Ethereum est définie par un ensemble de codes de programme. Chaque groupe de code est appelé un contrat intelligent. L'Ethereum définit une machine virtuelle Turing-complet, appelée EVM, dont l'ensemble d'instructions est appelé code EVM. Les utilisateurs peuvent développer des contrats intelligents grâce à un langage de programmation Solidity similaire à JavaScript, et les compiler en code EVM, et les déployer sur Ethereum cite dannen2017introducing. Une fois que le contrat intelligent est correctement déployé, il est équivalent à la définition du compte de contrat a qui reçoit la fonction de transition d'état $delta_a$. EVM est largement utilisé dans de telles plateformes, mais il y a aussi quelques problèmes. Par exemple, il manque la fonction de bibliothèque pour le support, ainsi que les problèmes de sécurité.

La structure du livre de l'Ethereum est une chaîne de blocs cite nakamoto2008bitcoin la chaîne de blocs est constituée des blocs, chaque bloc contient une liste de transactions, et le dernier bloc fait référence au Hash (fonction de hachage) du bloc précédent pour former une structure en chaîne.

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

Le plus grand avantage de cette structure est d'empêcher efficacement les transactions d'être falsifiées, parce qu'elle maintient l'ordre complet de toutes les transactions, l'échange de deux ordres de transaction génèrera un nouveau livre, qui a une probabilité plus élevée de fork. En fait, selon cette définition, l'espace d'état d'une machine d'état transactionnelle est considéré comme un arbre : l'état initial est le nud racine, l'ordre de transaction différent représente différents chemins et le nud feuille est l'état final. En réalité, l'état d'un grand nombre de nuds feuilles est le même, ce qui conduit à un grand nombre de fausses fourchettes.

L'algorithme de consensus *Phi* est appelé PoW, qui a d'abord été proposé dans le protocole Bitcoin cite nakamoto2008bitcoin. L'algorithme PoW repose sur un problème mathématique facilement vérifiable mais difficile à résoudre. Par exemple, basé sur une fonction de hachage $h : N\ rightarrow N$, trouvant le résultat de x, pour répondre à l'exigence $h(T + x)\ geqd$, d est un nombre donné, appelé la difficulté, T est une représentation binaire de la liste de commerce contenue dans le bloc. Chaque bloc de la chaîne de blocs contient une solution à ces problèmes. Additionnez la difficulté de tous les blocs, ce qui est la difficulté totale d'un registre de chaîne de blocs :

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

Par conséquent, lorsque vous choisissez le bon compte de fork, choisissez le fork avec la plus grande difficulté :

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

L'algorithme de consensus de PoW a une meilleure sécurité et a bien fonctionné dans Bitcoin et Ethereum. Cependant, il y a deux problèmes principaux dans cet algorithme.Le premier est de résoudre un problème mathématique qui exige une grande quantité de ressources informatiques, ce qu'il en résulte un extra d'énergie. La seconde est que la vitesse de convergence de l'algorithme est long, affectant ainsi le chargement et le déchargement global du système.Maintenant, le TPS de l'Ethereum est que d'environ 15, ce qui est totalement incapable de répondre aux besoins des applications décentralisées.

1.3 Direction de l'amélioration

Après la naissance de l'Ethereum, la communauté Ethereum et d'autres projets similaires ont commencé à améliorer le système à partir de différentes directions. À partir du modèle abstrait du système, les directions suivantes peuvent être améliorées :

- Améliorer l'état du système $\boldsymbol{S}$
- Améliorer la fonction de transition d'état $\boldsymbol{\delta}$
- Améliorer la structure du livre $\boldsymbol{\Gamma}$
- Améliorer l'algorithme de consensus $\boldsymbol{\Phi}$

1.3.1 Améliorer l'état du système

L'idée principale de l'amélioration de l'état du système est de localiser l'état global du monde, chaque nud ne s'occupe plus de toutes les transactions et transferts d'état et ne maintient qu'un sous-ensemble de la machine d'état entière. De cette manière, les potentiels de l'ensemble S et de l'ensemble T sont largement réduits, améliorant ainsi l'élasticité du système. De tels systèmes incluent : Cosmos cite cosmos, Aelf cite aelf, PChain et ainsi de suite.

En effet, ce schéma basé sur la chaîne latérale sacrifie la totalité de l'état du système en échange de l'évolutivité. Cela rend la décentralisation de chaque dApp en cours de fonctionnement affaiblie - l'historique des transactions d'un contrat intelligent n'est plus sauvegardé par tous les nuds du réseau entier, mais seulement par une partie du nud. De plus, l'interaction entre les contrats deviendra le goulot d'un tel système. Par exemple, dans Cosmos, les interactions dans différentes zones nécessitent un Hub de chaîne commune pour compléter cite cosmos.

1.3.2 Améliorer la fonction de transition d'état

Conformément à l'amélioration de la gestion EVM, certains projets fournissent des langages de programmation de contrats intelligents plus abondants. Par exemple, un langage de contrat intelligent Rholang est défini dans RChain basé sur *pi* calculus ; le contrat intelligent dans NEO s'appelle NeoContract, qui peut être développé dans les langages de programmation populaires tels que Java, C # etc ; EOS est programmé avec C / C ++.

1.3.3 Améliorer la structure du livre

La direction d'amélioration de la structure du livre est la construction de la classe équivalente. Le livre linéaire avec un ordre global des transactions multiples est amélioré en un livre non linéaire qui n'enregistre que les relations d'ordre partiel. Cette structure de livre non linéaire est un DAG (Directed Acyclic Graph). A présent, Byteball cite byteball, IOTA cite iota, Nano cite nano et d'autres projets ont réalisé la fonction de chiffrement de l'argent basé sur la structure de compte du DAG. Certains projets tentent d'utiliser le DAG pour mettre en uvre des contrats intelligents, mais jusqu'à présent, des améliorations dans cette direction sont encore à l'étude et à la recherche.

1.3.4 Améliorer l'algorithme de consensus

L'amélioration de l'algorithme de consensus est principalement d'améliorer la capacité de chargement et de déchargement du système, et la direction principale est de supprimer la génération de false fork. Ensuite, nous allons discuter des facteurs dans le false fork.

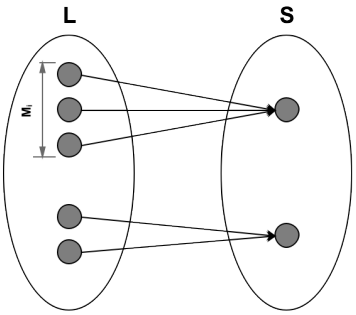

FIGURE 1 : False Fork

Comme indiqué,L est une collection de tous les comptes forked possibles pour un ensemble de transactions, et S est une collection d'états qui peuvent être atteints avec différents ordres. Selon la définition, **??**, mapping $f : L \to S$ est un surjectif, et selon la définition **??**, ce mapping n'est pas injectif. Ici, nous calculons la probabilité de la fausse fork :

Supposons que C les utilisateurs ont le droit de produire des livres, $M = |L|$,$N = |S|$,$M_i = |L_i|$, where $L_i = \{l|f(l) = s_i, s_i \in S\}$. La probabilité de fausse fork est la suivante :

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

À partir de cette formule, nous pouvons noter que, pour réduire la probabilité de fausse fork, il y a deux manières :

- Établir des relations d'équivalence sur le L d'un ensemble de livre, diviser les classes d'équivalence en eux, et construire moins forked livres.

— Restreindre les utilisateurs qui ont le droit de produire des livres, réduisant ainsi C

Le premier moyen est la direction importante dans le design du Vite. Nous allons en discuter en détail plus tard. Les deuxièmes moyens ont été acceptés par de nombreux algorithmes. Dans l'algorithme PoW, tout utilisateur a le droit de produire un bloc ; et l'algorithme PoS limite la puissance de la production du bloc à ceux qui ont des droits du système ; l'algorithme DPoS [3] limite utilisateur sur le droit à produire le block et être restreint davantage dans un groupe de noeuds d'agent.

A présent, Grâce à un algorithme de consensus amélioré, certains projets influents émergent. Par exemple, Cardano utilise un algorithme de PoS appelé Ouroboros, et la littérature [4] donne une preuve stricte des caractères liés de l'algorithme ; BFT-DPOS algorithme utilisé par EOS[5], est une variante de l'algorithme DPoS et améliore la capacité de chargement et déchargement du système en produisant rapidement des blocs ; Algorithme de consensus de Qtum [6] est aussi un algorithme PoS ; L'algorithme de Casper adopté par RChain [7] est aussi l'un des algorithmes de PoS..

Il y a aussi d'autres projets qui posent leurs propres propositions pour améliorer l'algorithme de consensus. NEO[8] utilise un algorithme de BFT, appelé dBFT, et Cosmos[9] utilise un algorithme appelé Tendermint [10].

2 Livre

2.1 Aperçu

Le rôle des livres est de déterminer l'ordre des transactions, et l'ordre des transactions affectera les deux aspects suivants :

— **Cohérence du statut :** Puisque l'état du système n'est pas un TDRC (Types de données répliquées sans conflit) [**?**], toutes les transactions ne sont pas échangeables et la séquence d'exécution de la transaction différente peut conduire le système à entrer dans un état différent.

— **Efficacité de Hash :** Dans le livre, la transaction sera empaquetée dans des blocs qui contiennent un hash référencé l'un par rapport à l'autre. L'ordre des transactions affecte la connectivité du hash cité dans les livres. Plus le scope de cet impact soit important, plus le coût de la falsification des transactions est élevé. En effet, toute modification apportée à une transaction doit être reconstruite par hash, qui se réfère directement ou indirectement au bloc de la transaction.

Le design du livre a également deux objectifs principaux :

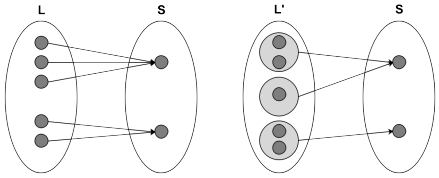

FIGURE 2 : Ledger merge

— **Réduire le taux du false fork :** Comme déja discuté dans la section précédente, la réduction du taux de fausse fork peut être obtenue en établissant une classe équivalente et en combinant un groupe de comptes qui amène le système dans le même état dans un seul compte. Comme indiqué ci-dessus, selon la formule de taux de fausse fork, le taux de fausse fork de livre sur la gauche $P_{ff} = \left(\frac{3}{5}\right)^C + \left(\frac{2}{5}\right)^C - \frac{1}{5^{C-1}}$;après la merge de l'espace du livre, le taux de fausse fork du graphique de droite est $P_{ff}{}' = \left(\frac{2}{3}\right)^C + \left(\frac{1}{3}\right)^C - \frac{1}{3^{C-1}}$.On sait que quand $C > 1$,$P_{ff}{}' < P_{ff}$. C'est-à-dire que nous devrions minimiser la relation d'ordre partielle entre les transactions et permettre l'échange séquentiel de plus de transactions.

— **Tamper preuve :** lorsqu'une transaction t est modifiée dans le livre l,dans les deux sous-livres du livre $l = l_1 + l_2$, le sous livre l_1 n'est pas affecté, et les références du hash dans le sous livre l_2 doivent être reconstruites pour former un nouveau livre valide $l' = l_1 + l_2{}'$. Sous-livre affecté $l_2 = \Gamma(T_2), T_2 = \{x | x \in T, x > t\}$.comme cela, pour augmenter le coût de la falsification des transactions, il est nécessaire de maintenir autant que possible la relation d'ordre partiel entre les transactions afin d'élargir le scope de la falsification $|T_2|$.

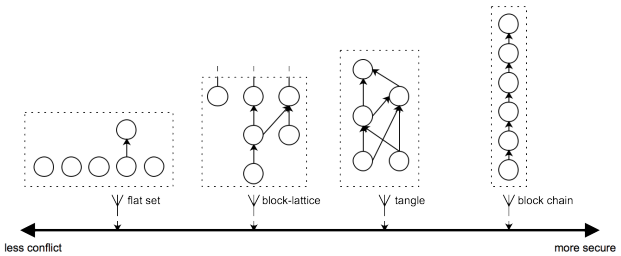

FIGURE 3 : Comparaison de la structure du livre

Evidemment, les deux objectifs ci-dessus sont contradictoires et les arbitrages nécessaires doivent être faits lors de la conception de la structure du compte. Puisque la gestion du compte est un ordre partiel entre les transactions, il s'agit essentiellement d'un ensemble ordonné partiel (poset) [11], si représenté par diagramme de Hasse (diagramme de Hasse)[12], c'est un DAG dans la topologie.

L'image ci-dessus compare plusieurs structures de livres communes, et les livres près de la gauche sont maintenus

avec moins d'ordre partiel. Le diagramme de Hasse semble plat et il a un taux de fausse fork inférieur ; les livres près du côté droit maintiennent plus de relations d'ordre partielles, et le diagramme de Hasse est plus effilé et plus inviolable.

Dans l'image, le côté le plus à gauche est une structure basée sur un ensemble commun dans un système de centralisation sans aucune fonctionnalité d'inviolabilité ; le côté le plus à droite est un livre blockchain typique avec les meilleures caractéristiques inviolables ; entre les deux, il y a deux livres DAG, le compte de bloc-réseau [13] utilisé par Nano à gauche ; et le côté droit, le livre d'enchevêtrement [14] est utilisé par l'IOTA. En termes de caractéristiques, le blocklattice maintient moins de relations d'ordre partielles et il est plus adapté à la structure comptable des plates-formes d'applications décentralisées avec hautes performances. En raison de ses mauvaises caractéristiques d'altération, il peut exposer les risques de sécurité, jusqu'à présent, aucun autre projet n'adopte cette structure de livre, sauf Nano.

Afin de poursuivre la haute performance, Vite adopte la structure du livre DAG. En même temps, en introduisant une Snapshot Chain de structure de chaîne supplémentaire et en améliorant l'algorithme de consensus, les failles de la sécurité en réseau de blocs sont correctement résolues, et les deux améliorations seront discutées en détail plus tard.

2.2 Pré-contrainte

D'abord, examinons la condition préalable à l'utilisation de cette structure de livre pour le modèle de machine d'état. Cette structure est essentiellement une combinaison de la machine d'état entière en tant qu'ensemble de machines d'état indépendantes, chaque compte correspondant à une machine d'état indépendante, et chaque transaction n'affecte que l'état d'un seul compte. Dans le livre, toutes les transactions sont regroupées dans des comptes et organisées en une chaîne de transactions dans le même compte. Par conséquent, nous avons les restrictions suivantes sur l'état S et la transaction T dans Vite :

Définition 2.1 (Contrainte de liberté d'un seul degré) *l'état du système $s \in S$,est le vecteur $s = (s_1, s_2, ..., s_n)$ formé par l'état s_i de chaque compte. Pour $\forall t_i \in T$,après effectuer la transaction t_i, l'état du système est transféré comme suit : $(s_1{}', ..., s_i{}', ..., s_n{}') = \sigma(t_i, (s_1, ..., s_i, ..., s_n))$,il nécessite de répondre à la condition :$s_j{}' = s_j, j \neq i$.Cette contrainte est appelée contrainte de liberté d'un seul degré pour une transaction.*

Intuitivement, une transaction avec la liberté d'un seul degré changera seulement l'état d'un compte sans affecter le statut d'autres comptes dans le système. Dans l'espace multidimensionnel où se trouve le vecteur d'espace d'état, une transaction est exécutée et l'état du système ne se meut que au long de la direction parallèle à un axe de coordonnées. Veuillez noter que cette définition est plus stricte que la définition de transaction dans Bitcoin, Ethereum et d'autres modèles. Une transaction dans Bitcoin changera l'état des deux comptes de l'expéditeur et du destinataire ; l'Ethereum peut changer l'état de plus de deux comptes via un appel de message.

Sous cette contrainte, la relation entre les transactions peut être simplifiée. Toute transaction est orthogonale ou parallèle. Cela fournit des conditions pour regrouper les transactions selon les comptes. Voici un exemple à expliquer :

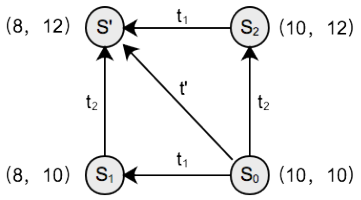

FIGURE 4 : Single degree of freedom trading and intermediate state

Comme montré dans la figure ci-dessus, supposons qu'Alice et Bob aient respectivement 10 USD. L'état initial du système est $s_0 = (10, 10)$. Quand Alice veut transférer 2 USD à Bob, dans le modèle de Bitcoin et Ethereum, une transactiont',peut conduire le système directement à l'état final :$s_0 \xrightarrow{t'} s'$.

Dans la définition de Vite, la transaction t' a également modifié le statut de deux comptes d'Alice et de Bob, ce qui n'était pas conforme au principe de de liberté d'un seul degré. Par conséquent, la transaction doit être scindée en deux transactions :

1) Une transaction t_1 représentant le transfert de 2 USD par Alice

2) Une transaction t_2 représente la réception de 2 USD par Bob

De cette manière, de l'état initial à l'état final s' il pourrait y avoir deux chemins différents $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} s'$ et $s_0 \xrightarrow{t_2} s_2 \xrightarrow{t_1} s'$. Ces deux chemins sont respectivement passés à travers l'état intermédiaire s_1 et s_2, et ces deux états intermédiaires sont le mapping de l'état final s' dans la dimension de ces deux comptes.En d'autres termes, si vous voudriez savoir uniquement de l'état de l'un des comptes, vous devez uniquement exécuter toutes les transactions qui correspondent au compte et ne pas avoir à effectuer les transactions des autres comptes.

Ensuite, nous allons définir comment diviser les transactions dans Ethereum en transactions à un seul degré de liberté requises par Vite :

Définition 2.2 (Décomposition de transaction) *Division d'une transaction avec le degré de liberté supérieur à 1 en un ensemble de transactions avec liberté en un seul degré, dénommé Décomposition des transactions. Une*

transaction de transfert peut être divisée en une transaction d'envoi et une transaction de réception ; une transaction d'appel de contrat peut être divisée en une transaction de demande de contrat et une transaction de réponse de contrat ; un appel de message dans chaque contrat peut être divisé en une transaction de demande de contrat et une transaction de réponse contractuelle.

Donc, il y aurait deux types différents de transactions dans les livres. Ils sont appelés "paires de négoce" :

Définition 2.3 (paires de négoce) *une transaction d'envoi ou une transaction de demande de contrat, collectivement appelée "transaction de demande" ; une transaction de réception ou une transaction de réponse de contrat, collectivement appelée "transaction de réponse". Une transaction de requête et une transaction de réponse correspondante sont appelées paires de transactions. Le compte pour initier la demande de transaction t est enregistré comme $A(t)$; la transaction de réponse correspondante est enregistrée comme : $\widetilde{t}$, le compte correspondant à la transaction est enregistré comme $A(\widetilde{t})$.*

Basé sur la définition ci-dessus, nous pouvons conclure la relation possible entre deux transactions de Vite :

Définition 2.4 (Relation de transaction) *Il peut exister pour les relations suivantes pour deux transactions t_1 and t_2 :*

Orthogonalité *:Si $A(t_1) \neq A(t_2)$,les deux transactions sont orthogonales, enregistrées comme : $t_1 \perp t_2$;*

Parallèle *:Si $A(t_1) = A(t_2)$,les deux transactions sont parallèles, enregistrées comme $t_1 \parallel t_2$;*

Causalité *:Si $t_2 = \widetilde{t_1}$,alors les deux transactions sont raisonnables, enregistrées comme $t_1 \rhd t_2$, or $t_2 \lhd t_1$.*

2.3 Définition du Livre

Pour définir un livre, c'est définir un poset. Tout d'abord, définissons la relation d'ordre partielle entre les transactions dans Vite :

Définition 2.5 (Ordre partiel des transactions) *Nous utilisons une relation dualiste $<$ pour représenter la relation d'ordre partiel de deux transactions : Une transaction de réponse doit suivre une transaction de demande correspondante :$t_1 < t_2 \Leftrightarrow t_1 \rhd t_2$;*

Toutes les transactions sur un compte doivent être strictement et globalement ordonnées :$\forall t_1 \parallel t_2$, la condition doit être satisfait :$t_1 < t_2$,ou $t_2 < t_1$.

Dû à la relation de commande partielle établie sur l'ensemble de transaction T répondent aux caractéristiques :

- Irréflexive : $\forall t \in T$,il n'y a pas $t < t$;
- Transitif :$\forall t_1, t_2, t_3 \in T$, Si $t_1 < t_2, t_2 < t_3$, puis $t_1 < t_3$;
- Asymétrique :$\forall t_1, t_2 \in T$, if $t_1 < t_2$, alors ça n'existe pas $t_2 < t_1$

De cette façon, nous pouvons définir le compte Vite dans un ensemble d'ordre partiel strict :

Définition 2.6 (Livre de Vite) *Vite Ledger est un poset restreint composé de l'ensemble de T de la transaction donnée, et du poset partiel $<$*

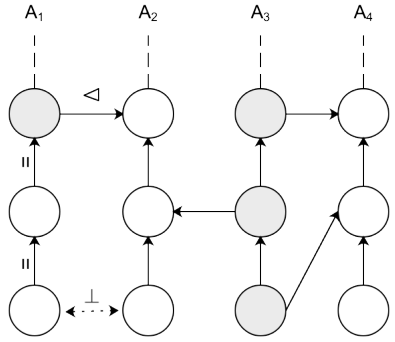

FIGURE 5 : La relation entre le livre et la transaction à Vite

Un poset strict peut correspondre à une structure DAG. Comme montré dans la figure ci-dessus, les cercles représentent les transactions et les flèches indiquent les dépendances entre les transactions. $a \rightarrow b$ indique que a dépend à b.

Le livre du Vite défini ci-dessus est structurellement similaire à un treillis-bloc. Les transactions sont divisées en transactions de requête et de réponse, chacune correspondant à un bloc distinct, chaque compte Ai correspond à une chaîne, une paire de transactions et une transaction de réponse référençant le hash de sa transaction de requête correspondante.

3 Chaîne instantanée

3.1 Confirmation de transaction

Lorsque le compte est forked, le résultat du consensus peut osciller entre deux livres forked. Par exemple, sur la base d'une structure blockchain, si un nud reçoit une chaîne forked plus longue, la nouvelle fork sera sélectionnée comme résultat consensuel et la fork d'origine sera abandonnée et la transaction sur la fork d'origine sera annulée. Dans un tel système, l'annulation d'une transaction est un événement très grave, ce qui entraînera une double dépense. Imaginez juste qu'une entreprise reçoit un paiement, fournit des biens ou des services, et après que le paiement est retiré, le commerçant peut rencontrer des pertes. Par conséquent, lorsqu'un utilisateur reçoit une transaction de paiement, il doit attendre que le système "confirme" la transaction pour s'assurer que la probabilité de reculer est suffisamment faible.

Définition 3.1 (Confirmation de Transaction) *lorsque la probabilité d'annulation d'une transaction est inférieure à un seuil donné ϵ, la transaction est considérée comme confirmée. $P_r(t) < \epsilon \Leftrightarrow t$ is **confirmed**.*

La confirmation des transactions est un concept très déroutant, parce que la reconnaissance d'une transaction dépend en fait du niveau de confiance implicite de $1 - \epsilon$. Un commerçant vendant des diamants et un vendeur de café ont subi des pertes différentes lorsqu'ils ont été attaqués par une double dépense. En conséquence, le premier doit définir ϵ plus petit sur la transaction. C'est aussi l'essence du nombre de confirmations dans Bitcoin. Dans Bitcoin, le numéro de confirmation indique la profondeur d'une transaction dans la chaîne de blocs. Plus le nombre de confirmations est élevé, plus la probabilité d'annulation de la transaction est faible cite nakamoto2008bitcoin. Par conséquent, les commerçants peuvent définir indirectement le niveau de confiance de la confirmation en définissant le nombre de numéros de confirmation en attente.

La probabilité d'annulation d'une transaction diminue avec le temps dû à la relation de référence de hash dans la structure du compte. Comme mentionné ci-dessus, lorsque la conception du livre a de meilleures caractéristiques d'altération, l'annulation d'une transaction doit reconstruire tous les blocs suivants de l'échange dans le bloc. Comme de nouvelles transactions sont constamment ajoutées aux livres, il y a de plus en plus de nuds successifs dans une transaction, donc la probabilité d'être trafiqué diminuera.

Dans la structure block-treillis, comme la transaction est groupée par compte, une transaction ne sera attachée qu'à la fin de la chaîne de compte de son propre compte, et la transaction générée par la plupart des autres comptes ne deviendra pas automatiquement un nud successeur de la transaction. Par conséquent, il est nécessaire de concevoir un algorithme de consensus raisonnablement pour éviter les dangers cachés de double dépense.

Nano adopte un algorithme de consensus basé sur le vote, [13], la transaction est signée par un ensemble de nuds représentatifs sélectionnés par un groupe d'utilisateurs. Chaque noeud représentatif a un poids. Lorsque la signature d'une transaction a assez de poids, on croit que la transaction est confirmée. Il y a des problèmes suivants dans cet algorithme :

Premièrement, si la confirmation avec le degré de confiance plus élevé est nécessaire, le seuil du poids de vote doit être augmenté. S'il n'y a pas assez de nuds représentatifs en ligne, la vitesse d'intersection ne peut pas être garantie, et il est possible qu'un utilisateur ne recueille jamais le nombre de tickets nécessaire pour confirmer un échange ;

Deuxièmement, la probabilité que les transactions soient annulées ne diminue pas au fils du temps. En effet, à tout moment, le coût du renversement d'un vote historique est le même.

Enfin, les résultats de vote historiques ne sont pas conservés dans le livre et sont stockés uniquement dans le stockage local des noeuds. Lorsqu'un nud obtient son compte auprès d'autres nuds, il n'existe aucun moyen de quantifier de façon fiable la probabilité d'annulation d'une transaction historique.

En effet, le mécanisme de vote est une solution de centralisation partielle. Nous pouvons considérer les résultats du vote comme un instantané de l'état des livres. Cet instantané sera distribué dans le stockage local de chaque noeud du réseau. Afin d'avoir la même capacité inviolable avec la chaîne de blocs, nous pouvons également organiser ces instantanés en structures de chaîne, qui est l'un des noyaux de la conception de Vite - la chaîne de clichés [15].

3.2 Définition de chaîne instantanée

La chaîne instantanée est la structure de stockage la plus importante de Vite. Sa fonction principale est de maintenir le consensus de Vite livre. D'abord, nous donnons la définition de la chaîne instantanés :

Définition 3.2 (Bloc d'instantanés et chaîne d'instantanés) *un bloc d'instantané qui stocke un état instantané d'un Vite livre, y compris le solde du compte, la racine Merkle de l'état du contrat et le hash du dernier bloc dans chaque chaîne de comptes. La chaîne instantanés est une structure en chaîne composée de blocs instantanés et le bloc d'instantanés suivant fait référence au hash du bloc instantanés précédent.*

L'état d'un compte d'utilisateur contient le solde et le hash du dernier bloc de la chaîne de compte ; en plus des deux champs ci-dessus, l'état d'un compte de contrat contient le hash racine de Merkle, la structure de l'état d'un compte est la suivante :

```
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;
}
```

La structure du bloc instantané est définie comme le suivant :

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

Afin de prendre en charge plusieurs jetons en même temps, la structure d'enregistrement des informations de solde dans l'état de compte de Vite n'est pas une `uint256`, mais un mapping a partir de token ID à la balance.

Le premier block instantané dans la chaîne d'instantanés est appelé ńinstantané de genèseż, qui enregistre des instances du bloc de genèse dans le compte.

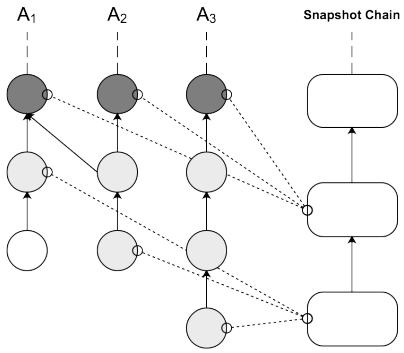

FIGURE 6 : snapshot chain

Étant donné que chaque bloc instantané dans la chaîne instantanée correspond à la seule fork du livre Vite, il est possible de déterminer le résultat consensuel du livre Vite par le bloc instantanés lorsque le bloc instantanés ne passe pas dans le bloc d'instantané.

3.3 Chaîne instantanée et confirmation de transaction

Après l'introduction de la chaîne instantanée, les défauts de sécurité naturels de la structure en treillis ont été corrigés. Si un attaquant veut générer une transaction de double dépense, en plus de reconstruire la référence de hash dans le livre Vite, il doit également être reconstruit dans la chaîne instantanée pour tous les blocs après le premier bloc instantané de la transaction, et doit produire une chaîne instantanée plus longue. De cette façon, le coût de l'attaque sera considérablement augmenté.

Dans Vite, le mécanisme de confirmation des transactions est similaire à Bitcoin, qui est défini comme suit :

Définition 3.3 (Confirmation de Transaction dans Vite) *Dans Vite, si une transaction est une instance par la chaîne instantanée, la transaction est confirmée, la profondeur du bloc instantanée dans le premier instance, est appelée le numéro de confirmation de la transaction.*

Selon cette définition, le nombre de transactions confirmées augmentera de 1 lorsque la chaîne instantanée augmentera, et la probabilité de l'attaque de double dépense diminuera avec l'augmentation de la chaîne d'instantanée. De cette manière, les utilisateurs peuvent personnaliser le numéro de confirmation requis en attendant différents numéros de confirmation selon le scénario spécifique.

La chaîne instantanée elle-même repose sur un algorithme de consensus. Si la chaîne instantanée est forkée, le fork textbf longest est choisi comme fork valide. Lorsque la chaîne instantanée est passée à une nouvelle branche, les informations instantanées d'origine seront annulées, ce qui signifie que le consensus original sur le livre a été renversé et remplacé par le nouveau consensus. Par conséquent, la chaîne instantanée est la première pierre de la sécurité du système entier, et doit être traitée sérieusement.

3.4 Compressed storage

Car tous les états de compte doivent être enregistrés dans chaque bloc instantané dans la chaîne instantanée, l'espace de stockage doit être très grand, la compression vers les chaînes instantanées est nécessaire.

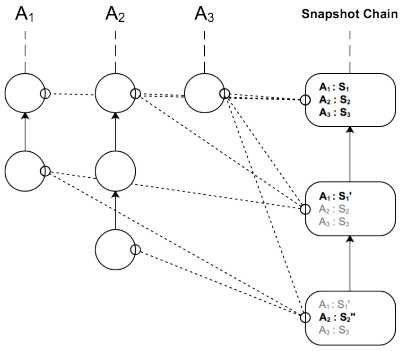

FIGURE 7 : Instantané avant compression

L'approche de base de la compression de l'espace de stockage de la chaîne instantanée consiste à utiliser le stockage incrémentiel : un bloc instantané stocke uniquement les données modifiées par rapport au bloc instantané précédent. S'il n'y a pas de transaction pour un compte entre les deux instantanés, le dernier bloc instantané n'enregistrera pas les données du compte.

Pour récupérer des informations instantanés, vous pouvez parcourir le bloc instantané du début à la fin et couvrir les données de chaque bloc instantanés en fonction des données en cours.

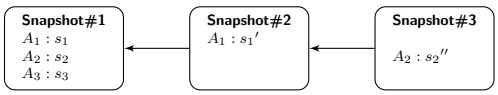

FIGURE 8 : Instantané après compression

Que le status final de chaque instantané d'un compte est sauvegardé lors qu'il est capturé, l'état intermédiaire ne sera pas pris en compte, de sorte qu'une seule copie des données instantanés sera sauvegardée, quel que soit le nombre de transactions générées par un compte entre deux instantanés. Par conséquent, un bloc instantané prend jusqu'à $S * A$ octets au maximum. Parmi eux, $S = \text{sizeof}(s_i)$, est le nombre d'octets occupés pour chaque état de compte, et A est le nombre total de comptes sur le système. Si le taux moyen des comptes actifs par rapport au total est a, le taux de compression est de $1 - a$.

4 Consensus

4.1 Goal of Design

Lors concevoir un protocole de consensus, nous devons prendre en compte les facteurs suivants :

- **Performance**.L'objectif principal de Vite est rapide. Pour garantir un chargement et déchargement élevé et des performances avec faible retard du système, nous devons adopter un algorithme de consensus avec une vitesse de convergence plus élevée.
- **Scalability**.Vite est une plate-forme publique et ouverte à toutes les applications décentralisées, donc l'évolutivité est également un facteur important.
- **Security**.Le principe de conception de Vite ne poursuit pas la sécurité extrême, cependant, il doit toujours assurer une ligne de base de sécurité suffisante et se prémunir efficacement contre toutes sortes d'attaques.

Comparé à certains algorithmes de consensus existants, la sécurité de PoW est meilleure, et un consensus peut être atteint si la puissance de calcul des nuds malveillants est inférieure à 50%. Cependant, la vitesse d'intersection de PoW est lente et ne peut pas répondre aux exigences de performance ; Le PoS et ses variantes d'algorithmes éliminent les étapes permettant de résoudre les problèmes mathématiques, d'améliorer la vitesse d'intersection et le coût d'attaque unique, et de réduire la consommation d'énergie. Mais l'évolutivité du PoS est encore faible, et le problème "Nothing at Stake" [16] est difficile à résoudre ; Les algorithmes BFT ont de meilleures performances en termes de sécurité et de performances, mais leur évolutivité est un problème, généralement plus adapté à une chaîne privée ou à une chaîne de consortium ; l'algorithme DPoS [3] réduit efficacement la probabilité de fausse fork en limitant les permissions de génération de blocs. Les performances et l'évolutivité sont bonnes. En conséquence, DPoS a un léger sacrifice de sécurité, et le nombre de nuds malveillants ne doit pas dépasser 1/3 [17].

Généralement, l'algorithme DPoS présente des avantages évidents en termes de performances et d'évolutivité. Par conséquent, nous choisissons DPoS comme la base du protocole de consensus Vite et nous l'étendons correctement sur la base de celui-ci. Grâce au protocole de consensus hiérarchique délégué et au modèle asynchrone, la performance globale de la plateforme peut être encore améliorée.

4.2 Hierarchical Consensus

The consensus protocol of Vite is HDPoS (Preuve de participation déléguée et hiérarchique). L'idée de base est de décomposer la fonction de consensus Φ (la décomposition fonctionnelle) :

$$\begin{aligned}\Phi(l_1, l_2, \ldots, l_n) = \Psi(&\Lambda_1(l_1, l_2, \ldots, l_n),\\ &\Lambda_2(l_1, l_2, \ldots, l_n), \ldots\\ &\Lambda_m(l_1, l_2, \ldots, l_n))\end{aligned} \tag{5}$$

$\Lambda_i : 2^L \to L$,est appelé comme fonction de consensus local, le résultat renvoyé est appelé le consensus local ; $\Psi : 2^L \to L$, connu sous le nom de la fonction de consensus globale, il sélectionne un résultat unique d'un groupe de candidat dans un consensus comme le résultat final consensuel.

Après cette séparation, le consensus de l'ensemble du système est devenu deux processus indépendants :

Consensus local générer les blocs correspondant aux transactions de requête et de transaction de réponse dans le compte d'utilisateur ou le compte de contrat, et écrire dans les ledgers.

Global consensus instantanés les données dans le livre et génère des blocs instantanés. Si le registre est forked, choisissez-en un.

4.3 Right of Block Generation and Consensus Group

Ensuite, qui a le droit de générer le bloc de transaction dans le bloc-notes et le bloc instantané dans la chaîneinstantanée ? Quel algorithme de consensus est adopté pour parvenir à un consensus ? Comme la structure de livre de Vite est organisée en plusieurs chaînes de compte selon différents comptes, nous pouvons définir facilement le droit de production des blocs dans le livre en fonction de la dimension du compte, et le droit de production du bloc instantané appartient à à un seul groupe d'utilisateurs. De cette manière, nous pouvons mettre un certain nombre de chaînes de comptes ou de chaînes instantanée dans un groupe de consensus, et dans le groupe de consensus, nous pouvons utiliser une manière unifiée pour produire le bloc et atteindre un consensus.

Définition 4.1 (Consensus Group) *Consensus group is a tuple (L, U, Φ, P), describing the consensus mechanism of a portion of the account or snapshot chain.,$L \in A|\{A_s\}$, represents one or a number of account chains, or snapshot chains of the consensus group in the ledger ; U represents the user with the block production right on the chain specified by the L ; Φ specifies the consensus algorithm of the consensus group ; and P specifies the parameters of the consensus algorithm.*

D'apès cette définition, les utilisateurs peuvent définir des groupes de consensus de manière flexible et sélectionner différents paramètres de consensus sur leurs besoins. Ensuite, nous allons élaborer sur les différents groupes de consensus.

4.3.1 Groupe de consensus instantané

Le groupe de consensus des chaînes instantanées est appelé groupe de consensus instantané, qui est le groupe de consensus le plus important de Vite. L'algorithme de consensus Φ du groupe consensus instantané adopte l'algorithme DPoS et correspond à Ψ dans le modèle hiérarchique. Le

nombre d'agents et l'intervalle de génération du bloc sont spécifiés par le paramètre P.

For example, we can specify snapshot consensus groups with 25 proxy nodes to produce snapshot blocks at intervals of 1 second. This ensures that the transaction is confirmed to be fast enough. Achieving 10 times transaction confirmation need to wait 10 seconds in maximum.

4.3.2 Private Consensus Group

Par exemple, nous pouvons spécifier des groupes consensus instantanés avec 25 noeuds proxy pour produire des blocs instantanés à intervalles de 1 seconde. Cela garantit que la transaction est confirmée pour être assez rapide. Obtenir 10 fois la confirmation de la transaction doit attendre 10 secondes au maximum.

Le plus grand avantage du groupe de consensus privé est de réduire la probabilité de fork. Car un seul utilisateur a le droit de produire des blocs, la seule possibilité de fork est que l'utilisateur initie une double attaque personnelle ou une erreur de programme.

L'inconvénient du groupe de consensus privé est que les nuds utilisateur doivent être en ligne avant de pouvoir emballer la transaction. Ce n'est pas très convenable pour le compte du contrat. Une fois que le nud du propriétaire a échoué, aucun autre nud ne peut remplacer la transaction de réponse qu'il produit, ce qui équivaut à réduire la disponibilité du service dApp.

4.3.3 Delegate Consensus Group

Dans le groupe de consensus délégué, au lieu du compte d'utilisateur, un ensemble de noeuds proxy désignés est utilisé pour conditionner la transaction via l'algorithme DPoS. A la fois les comptes d'utilisateur et les comptes contractuels peuvent être ajoutés au groupe de consensus. Les utilisateurs peuvent définir un ensemble de nuds d'agent distincts et établir un nouveau groupe de consensus. Il existe également un groupe de consensus par défaut dans Vite pour aider à regrouper les transactions pour tous les autres comptes qui n'ont pas établi individuellement leur groupe de consensus délégués, également connu sous le nom de **groupe de consensus public**.

Le groupe de consensus délégué convient à la plupart des comptes de contrats, car la plupart des transactions dans le contrat du compte sont des transactions de réponse de contrat, pour lesquelles une disponibilité plus élevée et des délais plus courts sont nécessaires au lieu des transactions recevables dans le compte utilisateur.

4.4 La priorité du consensus

Dans le protocole de Vite, la priorité du consensus global est supérieure à celle du consensus local. Lorsque le consensus local est forked, le résultat de la sélection du consensus mondial prévaudra. En d'autres termes, une fois que le consensus global a sélectionné un fork du consensus local comme le résultat final, même un fork plus longue d'une certaine chaîne de compte dans les comptes futurs se produit, elle ne provoquera pas le retour des résultats du consensus global.

Nous devons faire plus attention à ce problème lors de la mise en uvre du protocole cross-chain. Étant donné qu'une chaîne cible peut revenir en arrière, la chaîne de comptes correspondante du contrat de relais qui mappe la chaîne doit également être annulée en conséquence. A ce moment, si le consensus local de la chaîne de relais a été adopté par le consensus global, il est impossible de terminer le rollback, ce qui peut rendre incohérentes les données entre le contrat de relais et la chaîne cible.

La manière d'éviter ce problème est de définir un paramètre *delay* dans le paramètre de groupe de consensus P, qui spécifie le groupe de consensus instantané pour prendre un instantané seulement le consensus local est terminé après les blocs *delay*. Cela réduira considérablement la probabilité d'incohérence des contrats de relais, mais elle ne peut pas être complètement évitée. Dans la logique de code des contrats de relais, il est également nécessaire de traiter séparément la restauration de la chaîne cible.

4.5 Modèle asynchrone

Afin d'améliorer davantage le chargement et déchargement du système, nous devons supporter un modèle asynchrone plus parfait sur le mécanisme du consensus.

Le cycle de vie d'une transaction comprend l'initiation de la transaction, la rédaction de la transaction et la confirmation de la transaction. Afin d'améliorer les performances du système, nous devons concevoir ces trois étapes en mode asynchrone. En effet, à moments différents, le nombre de transactions initiées par les utilisateurs est différent, la rapidité de l'écriture des transactions et la confirmation des transactions traitées par le système sont fixées de manière relative. Le mode asynchrone aide à aplatir les pics et les creux, donc améliore le chargement et déchargement global du système.

Le modèle asynchrone du Bitcoin et de l'Ethereum est simple : la transaction initiée par tous les utilisateurs est placée dans un pool non confirmé. Lorsque le mineur l'empaquette dans un bloc, la transaction est écrite et confirmée en même temps. Lorsque la chaîne de blocs continue à croître, la transaction atteint finalement le niveau de confiance de confirmation prédéfini.

Il y a deux problèmes dans ce modèle asynchrone :

- Les transactions ne sont pas insistant aux livres dans un état non confirmé. Les transactions non reconnues sont instables et ne font l'objet d'aucun consensus. Elles ne peuvent empêcher répéter l'envoi de transactions.
- Il n'y a pas de mécanisme asynchrone pour l'écriture et la confirmation des transactions. Les transactions ne sont écrites que lorsqu'elles sont confirmées, et la vitesse d'écriture est limitée par la vitesse de

confirmation.

Le protocole de Vite établit un modèle asynchrone amélioré : d'abord, la transaction est divisée en une paire de transactions basée sur un modèle "requête-réponse",qu'il s'agisse d'un transfert ou d'un contrat, et la transaction est lancée avec succès est écrit dans le livre. De plus, l'écriture et la confirmation d'une transaction sont également asynchrones. Les transactions peuvent d'abord être écrites dans le compte DAG de Vite et ne seront pas bloquées par le processus de confirmation. La confirmation de la transaction est effectuée via la chaîne d'instantané, et l'action d'instantané est également asynchrone.

C'est un modèle producteur-consommateur typique. Dans le cycle de vie de la transaction, quel que soit le changement de taux de production en amont, l'aval peut traiter la transaction à un rythme constant, de manière à utiliser pleinement les ressources de la plate-forme et améliorer le débit du système.

5 Machine Virtuelle

5.1 Compatibilité EVM

A présent, il existe de nombreux développeurs dans le domaine Ethereum, et de nombreux contrats intelligents sont appliqués sur la base de Solidity et EVM. Par conséquent, nous avons décidé de fournir une compatibilité EVM sur la machine virtuelle Vite, et la sémantique d'origine dans la plupart des jeux d'instructions EVM est conservée dans Vite. Parce que la structure de compte et la définition de transaction de Vite sont différentes d'Ethereum, la sémantique de certaines instructions EVM doit être redéfinie, par exemple, un ensemble d'instructions pour obtenir des informations de bloc. Les différences sémantiques détaillées peuvent être référées à l'appendice A..

Parmi eux, la plus grande différence est la sémantique des appels de messages. Ensuite, nous allons discuter en détail.

5.2 Event Driven

Dans le protocole d'Ethereum, une transaction ou un message peut affecter l'état de plusieurs comptes. Par exemple, une transaction d'appel de contrat peut entraîner le changement simultané du statut de plusieurs comptes de contrat via des appels de message. Ces changements se produisent en même temps ou pas du tout. Par conséquent, la transaction dans l'Ethereum est en fait une sorte de transaction rigide qui satisfait les caractéristiques de l'ACID (Atomicity, Consistency, Isolation, Durability [18], qui est aussi une raison importante pour le manque d'expansibilité dans l'Ethereum.

Basé sur des considérations d'évolutivité et de performance, Vite a adopté un schéma de cohérence final satisfaisant la sémantique BASE (Basically Available, Soft state, Cohérence éventuelle) [19]. Plus précisément, nous concevons Vite comme une architecture événementielle (EDA) [20]. Chaque contrat intelligent est considéré comme un service indépendant et les messages peuvent être communiqués entre les contrats, mais aucun état n'est partagé.

Par conséquent, dans EVM of Vite, nous devons annuler la sémantique des appels de fonction synchrones entre les contrats, et permettre uniquement la communication de messages entre les contrats. Les EVM instructions affectées sont principalement **CALL** et **STATICCALL**.Dans Vite EVM, ces deux instructions ne peuvent pas être exécutées immédiatement, ni renvoyer le résultat de l'appel. Ils génèrent uniquement une transaction de demande à écrire dans le livre. Par conséquent, dans Vite, la sémantique des appels de fonctions ne sera pas incluse dans cette instruction, mais enverra plutôt des messages à un compte.

5.3 Smart Contract Language

Ethereum fournit un langage de programmation Turing complet et Solidité pour développer des contrats intelligents. Pour supporter la sémantique asynchrone, nous avons amplifié la Solidité et défini un ensemble de syntaxe pour la communication de messages. La Solidité amplifié s'appelle Solidité++.

La plupart des syntaxes de Solidité sont supportées par Solidité ++, mais n'incluent pas les appels de fonctions en dehors du contrat. Le développeur peut définir des messages via le mot-clé *message* et définir le processeur de message (MessageHandler) à l'aide du mot-clé *on* pour implémenter la fonction de communication inter-contrat.

Par exemple, le contrat A doit appeler la méthode add () dans le contrat B pour mettre à jour son état en fonction de la valeur de retour. Dans Solidité, il peut être implémenté par l'appel de fonction. Le code est comme suit :

```
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;
        }
    }
}
```

Dans Solidité++, le code de fonction d'appel`uint sum = B(addr).add(a, b);` n'est plus valide ; au lieu de cela, le contrat A et le contrat B communiquent de manière asynchrone en envoyant des messages les uns aux autres. Le code est comme suit :

```
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;
        }
    }
}
```

Dans la première ligne, le code `pragma solidity++ ^0.1.0;` indique que le code source est écrit dans Solidité ++ mais ne sera pas compilé directement avec le compilateur Solidité pour éviter que le code EVM compilé ne soit pas conforme au sémantique. Vite fournira un compilateur spécialisé pour compiler Solidity ++. Ce compilateur est partiellement compatible avec l'avant : s'il n'y a pas de code de Solidité en conflit avec la sémantique de Vite, il peut être compilé directement, sinon l'erreur sera signalée. Par exemple, la syntaxe des appels de fonctions locales, les transferts vers d'autres comptes resteront compatibles ; l'obtention de la valeur de retour de l'appel de fonction "cross contract", ainsi que de l'unité monétaire `ether`, ne sera pas compilée.

Dans le contrat A, lorsque la fonction `invoker` est appelée, le message `Add` sera envoyé au contrat B, qui est asynchrone et le résultat ne sera pas renvoyé immédiatement.Par conséquent, il est nécessaire de définir un processeur de message dans A en utilisant le mot-clé `on` pour recevoir le résultat retourné et mettre à jour l'état.

Dans le contrat B, le message `Add` est surveillé. Après le traitement, un message `Sum` est envoyé à l'expéditeur du message `Add` pour retourner le résultat.

Les messages dans Solidité++ seront compilés dans des instructions **CALL** et une transaction de requête sera ajoutée au livre. Dans Vite, les registres servent de middleware de message pour la communication asynchrone entre les contrats. Il assure un stockage fiable des messages et empêche la duplication. Plusieurs messages envoyés à un contrat par le même contrat peuvent garantir FIFO (First In First Out) ; les messages envoyés par différents contrats au même contrat ne garantissent pas le FIFO.

Il est à noter que les événements dans Solidité (Event) et les messages dans Solidité++ ne sont pas le même concept. Les événements sont envoyés indirectement à l'avant via le journal EVM.

5.4 Librairie Standard

Les développeurs qui développent des contrats intelligents sur Ethereum sont souvent ennuyés par l'absence de bibliothèques standard dans Solidité. Par exemple, la vérification de boucle dans le protocole Loopring doit être effectuée en dehors de la chaîne, une des raisons importantes est que la fonction de calcul à virgule-flottante n'est pas fournie dans Solidité, en particulier la racine n [1][1] pour les nombres flottants.

Dans EVM, un contrat pré-déployé peut être appelé par la commande **DELEGATECALL** pour réaliser la fonction de la bibliothèque. Ethereum fournit également plusieurs contrats précompilés, qui sont principalement quelques opérations de hash. Mais ces fonctions sont trop simples à répondre aux besoins d'applications complexes.

Par conséquent, nous fournirons une série de bibliothèques standard dans Solidité++, telles que le traitement de chaînes, les opérations à virgule flottante, les opérations mathématiques de base, les conteneurs, le tri, etc.

Basé sur des considérations de performances, ces bibliothèques standard seront implémentées dans une extension locale (extension native) et la plupart des opérations sont intégrées dans le code local de Vite, et la fonction est appelée uniquement via l'instruction **DELEGATECALL** dans le Code EVM.

La bibliothèque standard peut être amplifié au besoin, cependant comme le modèle de machine d'état de l'ensemble du système est déterministe, il ne peut pas fournir de fonctions comme des nombres aléatoires. C'est similaire

que Ethereum, nous pouvons simuler des nombres pseudo-aléatoires à travers le hash de chaînes d'instantanés.s.

5.5 Gaz

Il y a deux fonctions principales pour le gaz dans l'Ethereum, la première est de quantifier les ressources de calcul et les ressources de stockage consommées par l'exécution de code EVM, et la seconde est de s'assurer que le code EVM a une halte. Selon la théorie de la calculabilité, le problème d'arrêt sur les machines de Turing est un problème incompatible cite haltingproblems. Cela signifie qu'il est impossible de déterminer si un contrat intelligent peut être arrêté après une exécution limitée en analysant le code EVM.

Par conséquent, le calcul du gaz dans EVM est également conservé dans Vite. Cependant, il n'y a pas de concept de prix du gaz dans Vite. Les utilisateurs n'achètent pas le gaz pour un échange en payant les frais, mais à travers un modèle basé sur le quota pour obtenir des ressources informatiques. Le calcul des quotas sera discuté en détail plus loin dans le chapitre "Modèle Economique".

6 Modèle Economique

6.1 Native Token

Afin de quantifier les ressources de calcul et de stockage de la plate-forme et d'encourager les nuds à s'exécuter, Vite a construit un "Native Token" ViteToken. L'unité de base du token est *vite*, la plus petite unité est *attov*,,1 *vite* = 10^{18} *attov*.

La chaîne d'instantanés est la clé de la sécurité et des performances de la plate-forme Vite. Afin d'inciter un nud à participer à la vérification de la transaction, le protocole Vite établit la prime de FORGING pour la production du bloc d'instantané.

Au contraire, lorsque les utilisateurs émettent de nouveaux tokens, déploient des contrats, enregistrent des noms de domaine VNS [1] et obtiennent des quotas de ressources, ils ont besoin de consommer ou mettre en gage ViteToken.

Sous l'action combinée de ces deux facteurs, elle est utile à l'optimisation de l'allocation des ressources du système.

6.2 Allocation de Ressource

Étant donné que Vite est une plate-forme dApp commune, les capacités des contrats intelligents qui y sont déployés varient, et chaque contrat intelligent différent a des exigences différentes en termes de capacité de chargement et déchargement et de délai. Même pour le même contrat intelligent, les exigences de performance à différentes étapes sont différentes.

Dans la conception de l'Ethereum, chaque transaction doit être attribuée avec un prix du gaz lors du lancement, afin de concurrencer avec d'autres transactions pour écrire des comptes. Il s'agit d'un modèle d'appel d'offres type qui permet de contrôler efficacement l'équilibre entre l'offre et la demande en principe. Cependant, l'utilisateur est difficile à quantifier la situation actuelle de l'offre et de la demande, et ne peut pas prédire le prix d'autres concurrents, donc la défaillance du marché se produit facilement. De plus, les ressources en concurrence pour chaque offre sont dirigées contre une transaction, et il n'y a pas d'accord sur l'allocation rationnelle des ressources selon la dimension du compte.

6.2.1 Quota Calculation

Nous avons adopté un protocole d'allocation de ressources basé sur les quotas dans Vite, qui permet aux utilisateurs d'obtenir des quotas de ressources plus élevés de trois façons :

- Un PoW est calculé lorsque la transaction est initiée ;
- Hypothèque une certaine quantité de *vite* dans le compte ;
- Pour détruire une petite quantité de *vite* en une seule fois.

Les quotas spécifiques peuvent être calculés à l'aide de la formule suivante :

$$Q = Q_m \cdot \left(\frac{2}{1 + exp\left(-\rho \times \xi^{\top}\right)} - 1 \right) \tag{6}$$

Parmi eux, Q_m est une constante, représentant la limite supérieure d'un seul quota de compte, qui est liée au chargement et déchargement total du système et au nombre total de comptes.$\xi = (\xi_d, \xi_s, \xi_f)$ est un vecteur qui représente le coût d'un utilisateur pour l'obtention d'une ressource : ξ_d est la difficulté de calcul que l'utilisateur calcule lors de la génération d'une transaction, ξ_s est le solde *vite* de l'hypothèque dans le compte, et ξ_f est le coût ponctuel que l'utilisateur est prêt à payer pour l'augmentation du quota. Il est à noter que,ξ_f est différent des frais de gestion. Ces *vite* seront détruits directement au lieu d'être payés aux mineurs.

Dans la formule, le vecteur $\rho = (\rho_d, \rho_s, \rho_f)$ représente le poids des trois manière d'obtenir le quota, c'est-à-dire le quota obtenu par le la destruction de 1 *vite* est équivalente à ρ_s/ρ_f *vite* hypothéqué.

On peut voir à partir de cette formule que si l'utilisateur ne met pas en gage le coût ponctuel ni le paie, il est nécessaire de calculer un PoW, sinon il n'y aura pas de quotas pour lancer une transaction, ce qui peut effectivement empêcher l'attaques de poussière et protège les ressources du système contre les abus. En même temps, cette formule est une fonction logistique. Il est relativement facile pour les utilisateurs d'obtenir des quotas plus bas, réduisant ainsi le seuil des utilisateurs de basse fréquence ; et les utilisateurs à haute fréquence doivent investir beaucoup de ressources pour obtenir des quotas plus élevés. Les coûts

1. se référer au service de nommage *7.2*

supplémentaires qu'ils paient vont augmenter les avantages de tous les utilisateurs.

6.2.2 Resource Quantification

Parce que la chaîne d'instantané est équivalente à une horloge globale, nous pouvons l'utiliser pour quantifier l'utilisation des ressources d'un compte avec précision. Dans chaque transaction, le hash d'un bloc d'instantané est cité, la hauteur du bloc d'instantané est prise comme horodatage de la transaction. Par conséquent, en fonction de la différence entre les deux horodatages de transaction, nous pouvons juger si l'intervalle entre les deux transactions est suffisamment long.

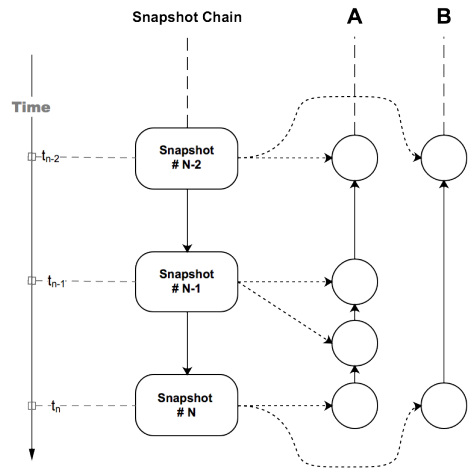

FIGURE 9 : chaîne d'instantané comme une horloge globale

Comme indiqué ci-dessus, le compte A a généré 4 transactions à 2 intervalles de temps, tandis que le compte B n'a généré que 2 transactions. Par conséquent, le TPS moyen de A dans cette période est 2 fois celui de B. Si c'est juste une transaction de transfert, le TPS moyen du compte quantifié est suffisant. Pour les contrats intelligents, chaque échange a une consommation de ressources différente, il est donc nécessaire d'accumuler du gaz pour chaque transaction afin de calculer la consommation moyenne de ressources pendant une période donnée. La consommation moyenne de ressources des transactions k récentes dans une chaîne de compte avec une hauteur de "n" est la suivante :

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

Parmi eux, pour une transaction T_n, $timestamp_n$ est l'horodatage de la transaction, c'est-à-dire la hauteur du bloc d'instantané auquel elle fait référence ; gas_n est le carburant consommé pour la transaction.

Lors de la vérification d'une transaction, le nud déterminera si le quota satisfait à la condition : $Cost(T) \leq Q$, et s'il n'est pas satisfait, la transaction sera rejetée. Dans ce cas, les utilisateurs doivent reconditionner une transaction, augmenter les quotas en payant des frais uniques ou attendre une période de temps pour indiquer un instantané supérieur dans la transaction.

6.2.3 Bail de quota

Si un utilisateur possède des actifs *vite* abondants, mais n'a pas besoin d'utiliser autant de quotas de ressources, il peut choisir de louer son quota à d'autres utilisateurs.

Le système de Vite prend en charge un type de transaction spécial pour transférer le droit d'utiliser un quota de ressources de compte. Dans cette transaction, le nombre de *vite* pouvant être mise en gage, l'adresse d'un cessionnaire et la durée d'un bail peuvent être spécifiés. Une fois la transaction confirmée, le quota de ressources correspondant au montant du token sera inclus dans le compte du cessionnaire. Une fois le délai de location dépassé, le quota sera calculé dans le compte du cédant. L'unité de temps de location est la deuxième. Le système sera converti en différence de hauteur du bloc d'instantanés, de sorte qu'il peut y avoir une déviation.

Le revenu de location peut être obtenu par l'utilisateur. Le système Vite fournit uniquement une transaction de transfert de quota, et le prix et le paiement du crédit-bail peuvent être obtenus grâce à un contrat intelligent tiers.

6.3 Assurance de biens

En plus du token natif ViteToken, Vite permet également aux utilisateurs d'émettre leurs token. L'issue des tokens peut être effectuée via une transaction spéciale, Mint Transaction. L'adresse cible de la transaction mint est 0. Dans le champ *data* de la transaction, les paramètres du token sont spécifiés comme suit :

```
Mint: {
    name: "MyToken",
    totalSupply: 999999999000000000000000000,
    decimals: 18,
    owner: "0xa3c1f4...fa",
    symbol: "MYT"
}
```

Une fois la demande acceptée par le réseau, le *vite* inclus dans la transaction mint sera déduit du compte de l'initiateur comme frais de transaction de la Monnaie. Le système enregistre les informations du nouveau token et lui attribue un *token_id*. Tous les soldes des token nouvellement générés seront ajoutées à l'adresse *owner*, c'est-à-dire que le compte propriétaire est le compte de genèse du token.

6.4 Cross Chain Protocol

Afin de supporter le transfert de valeur croisée des ressources numériques et d'éliminer ńl'îlot de valeurż, Vite a conçu un protocole VCTP (Vite Cross-Chain Transfer Protocol).

Pour chaque actif nécessitant une transmission croisée sur la chaîne cible, un token qui lui correspond est nécessaire dans le Vite comme le bon du token cible circulant dans le Vite, appelé ToT (Token of Token). Par exemple, si vous voulez transférer le *ether* dans le compte Ethereum vers Vite, vous pouvez émettre un ToT avec un identifiant de *ETH* dans Vite, la quantité initiale de TOT doit être égale à la quantité totale de *ether*.

Pour chaque chaîne cible, il existe un contrat de passerelle sur Vite pour maintenir la relation de mapping entre les transactions Vite et les transactions de chaîne cible. Dans le groupe de consensus où se trouve le contrat, le nud responsable de la génération de blocs est appelé relais VCTP. Le relais VCTP doit être le nud Vite et le nud complet de la chaîne cible en même temps, et suivre les transactions des deux côtés. Sur la chaîne cible, nous devons également déployer un contrat de passerelle Vite.

Avant que le relais VCTP commence à fonctionner, le ToT correspondant dans Vite doit être transféré au contrat de passerelle. Après cela, l'offre de ToT ne peut être contrôlée que par le contrat de passerelle, et personne ne peut être ajouté pour assurer le taux d'échange 1 : 1 entre le ToT et l'actif cible. En même temps, les actifs de la chaîne cible sont contrôlés par le contrat de passerelle Vite, et aucun utilisateur ne peut l'utiliser, de manière à garantir que ToT dispose d'une réserve d'acceptation complète.

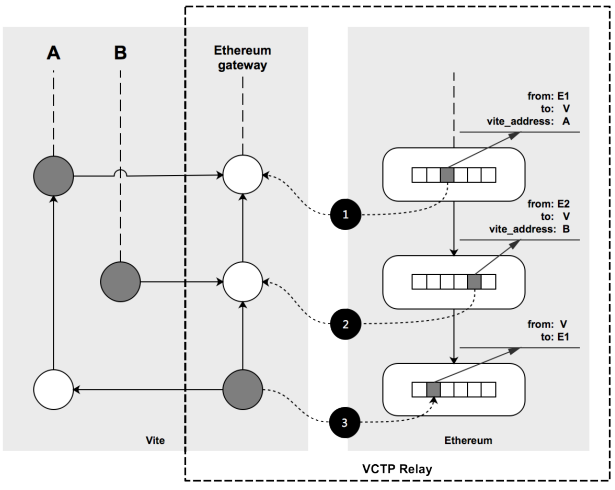

Figure 10 : Cross Chain Protocol

L'image ci-dessus est un exemple de transmission de la valeur de la chaîne croisée entre le Vite et l'Ethereum. Lorsque l'utilisateur Ethereum $E1$ veut transférer le token de l'Ethereum vers le Vite, il peut envoyer une transaction à l'adresse de contrat de passerelle Vite V, tandis que l'adresse de l'utilisateur A sur le Vite est placée dans le paramètre. Le solde du transfert sera verrouillé dans le compte de contrat passerelle et deviendra partie de la réserve ToT. Après avoir écouté la transaction, le nud relais VCTP génère une transaction d'envoi de compte correspondante de Vite, envoyant la même quantité de ToT au compte A de l'utilisateur dans le Vite. Dans l'image, ① et ② indiquent respectivement que $E1$ et $E2$ sont transférés sur le compte Vite A et B. Il est à noter que si l'utilisateur ne spécifie pas l'adresse Vite lors du transfert, le contrat rejettera la transaction.

Le flux inverse est indiqué dans ③, Lorsque l'utilisateur A lance le transfert du compte Vite vers le compte Ethereum, une transaction sera envoyée au contrat de passerelle Vite, transférée à une certaine quantité de ToT, et spécifie l'adresse de réception $E1$ de l'Ethereum dans la transaction. Le nud de relais VCTP va générer le bloc de réponse correspondant sur le contrat "Ethereum Gateway", et empaqueter une transaction de l'Ethereum vers le contrat de passerelle Vite sur l'Ethereum. Dans l'Ethereum, le contrat de passerelle Vite vérifie si cette transaction est initiée par un relais VCTP approuvé, puis la même quantité de *ether* est transférée du contrat de passerelle Vite vers le compte cible $E1$.

Tous les nuds relais de la chaîne croisée surveillent le réseau cible et peuvent vérifier si chaque transaction interchaîne est correcte et parvenir à un consensus au sein du groupe de consensus. Mais le groupe de consensus d'instantanée ne surveillera pas la transaction de la chaîne cible et ne vérifiera pas non plus si le mapping entre les deux chaînes est correct. Si le réseau cible est annulé ou durci, les transactions mappées dans le système Vite ne peuvent pas être annulées ; De même, si les transactions croisées dans le Vite sont annulées, la transaction correspondante du réseau cible ne peut pas être annulée en même temps. Par conséquent, lorsque vous effectuez des transactions entre chaînes, il est nécessaire de traiter l'annulation de la transaction dans la logique du contrat. En même temps, comme décrit dans la partie 4.4 nous devons définir un paramètre *delay* pour le groupe de consensus relais de la chaîne croisée.

6.5 Loopring Protocol

Loopring protocol cite loopring est un protocole open source pour construire un réseau de trading d'actifs décentralisé. Comparé à d'autres solutions DEX, le protocole Loopring est basé sur le mapping de boucle multipartite, qui fournit une technologie de double autorisation double pour empêcher les transactions préemptives et est entièrement ouverte.

Nous construisons le protocole Loopring dans Vite, ce qui favorise la circulation des actifs numériques dans Vite, de sorte que tout le système de valeurs puisse circuler. Dans ce système de valeurs, les utilisateurs peuvent émettre leurs propres ressources numériques, transférer des ressources

en dehors de la chaîne via VCTP et utiliser le protocole Loopring pour réaliser l'échange de ressources. L'ensemble du processus peut être complété dans le système Vite et est complètement décentralisé.

Dans Vite, le protocole de Loopring Smart contract (LPSC) fait partie du système Vite. L'autorisation de transfert d'actifs et la protection atomique multi-parties sont toutes supportés dans Vite. Le relais de Loopring est toujours ouvert à etre intégré pleinement à son propre écosystème.

Les utilisateurs peuvent utiliser *vite* pour payer les transactions d'échange d'actifs, de sorte que le token gagné par les mineurs de Looping qui effectuent le mapping de boucles dans la plate-forme Vite est toujours *vite*.

7 D'autres conceptions

7.1 Planification

Dans l'Ethereum, les contrats intelligents sont pilotés par des transactions, en plus l'exécution des contrats ne peut être déclenchée que par les utilisateurs initiant une transaction. Dans certaines applications, une fonction d'ordonnancement est nécessaire pour déclencher l'exécution d'un contrat via une horloge.

Dans Ethereum, cette fonction est atteint par des contrats de tiers. [1], les performances et la sécurité ne sont pas garanties. Dans Vite, nous ajoutons la fonction d'ordonnancement temporel au contrat intégré. Les utilisateurs peuvent enregistrer leur logique d'ordonnancement dans le contrat d'ordonnancement temporisé. Le groupe de consensus public utilisera la chaîne d'instantanée comme une horloge et enverra la transaction de requête au contrat cible conformément à la logique de planification définie par l'utilisateur.

Il y a un message spécialisé `Timer` dans Solidity ++. Les utilisateurs peuvent définir leur propre logique de planification dans le code du contrat via `Timer.on`.

7.2 Service de nommer

Dans Ethereum, le contrat va générer une adresse pour identifier un contrat lors de son déploiement. Il y a deux problèmes dans l'identification des contrats avec des adresses :

— Une adresse est un identifiant avec 20 octets sans signification. Il n'est pas aisément utilisable aux utilisateurs et inconvénient à utiliser.

— Les contrats et les adresses ont une un-à-un relation. Ils ne peuvent pas supporter la redirection de contrat.

Afin de résoudre ces deux problèmes, le développeur d'Ethereum a fourni un contrat tiers ; ENS [2]. Cependant, dans le scénario actuel, l'utilisation de services de nommer sera très fréquente, et l'utilisation de contrats tiers ne peut garantir l'unicité globale de la dénomination, nous allons donc construire un service de noms VNS (ViteName Service) dans Vite.

Les utilisateurs peuvent enregistrer un ensemble de noms faciles à retenir et les résoudre à l'adresse réelle via VNS. Les noms sont organisés sous la forme de noms de domaine, tels que*vite.myname.mycontract*. Le nom de domaine de premier niveau sera conservé par le système à des fins spécifiques. Par exemple, *vite.xx* représente l'adresse Vite et *eth.xx* représente une adresse Ethereum. Le nom de domaine de deuxième niveau est open à tous les utilisateurs. Une fois que l'utilisateur possède le nom de domaine de deuxième niveau, le sous-domaine peut être amplifié arbitrairement. Le propriétaire du nom de domaine peut à tout moment modifier l'adresse indiquée par le nom de domaine. Cette fonction peut donc être utilisée pour la mise à niveau du contrat.

La longueur du nom de domaine n'est pas restreinte. Dans VNS, le hash du nom de domaine est réellement stocké. L'adresse cible peut être une adresse non Vite de moins de 256 bits, qui peut être utilisée pour l'interaction entre chaînes.

Nous devons noter que VNS est différent de la spécification du package smart contract EIP190 [3] dans Ethereum. VNS est un service de résolution de noms, le nom est établi à l'exécution, et les règles de résolution peuvent être modifiées dynamiquement ; et EIP190 est une spécification de gestion de package ; l'espace de noms est statique et il est établi au moment de la compilation.

7.3 Mise à jour

Le contrat intelligent d'Ethereum est immuable. Une fois déployé, il ne peut pas être modifié. Même s'il y a un bug dans le contrat, il ne peut pas être mis à jour. Ceci n'est vraiment pas aisément utilisable aux développeurs et rend l'itération continue de dApp très difficile. Par conséquent, Vite doit fournir un schéma pour prendre en charge la mise à jour des contrats intelligents.

Dans Vite, le processus de mise à jour du contrat comprend :

A. Déploie une nouvelle version du contrat pour hériter du statut du contrat d'origine.

B. Indique le nom du contrat à la nouvelle adresse dans VNS.

C. Supprime l'ancien contrat via l'instruction **SELF-DESTRUCT**.

Ces trois étapes doivent être complétées en même temps, et le protocole Vite assure l'atomicité de l'opération. Les développeurs doivent s'assurer que les anciennes données de contrat sont correctement traitées dans le contrat de

1. *Ethereum Alarm Clock* est un contrat tiers utilisé pour planifier l'exécution d'autres contrats, se référer à `http://www.ethereum-alarm-clock.com/`

2. *Ethereum Name Service* est un contrat tiers utilisé pour la résolution de nommer, reportez-vous à `https://ens.domains/`

3. *EIP190* Ethereum Smart Contract Packaging Spécification, reportez-vous à `https://github.com/ethereum/EIPs/issues/190`

nouvelle version.

Nous devons noter que le nouveau contrat n'héritera pas de l'adresse de l'ancien contrat. Si cité par l'adresse, la transaction sera toujours envoyée à l'ancien contrat. En effet, différentes versions de contrats sont essentiellement deux contrats totalement différents, qu'ils soient ou non modifiables dynamiquement, en fonction de la sémantique des contrats.

Dans les systèmes Vite, les contrats intelligents sont en fait divisés en deux types, le premier est en fait une circonstance d'un dApp, et sa logique métier est décrite ; et la seconde est une sorte de contrat qui cartographie le monde réel. Le précédent est équivalent au service de circonstance d'une application, qui doit être continuellement réitéré lors d'une mise à niveau ; ce dernier est équivalent à un contrat, et une fois qu'il entre en vigueur, aucune modification ne peut être faite, sinon c'est une rupture de contrat. Pour un tel contrat qui n'est pas autorisé à être modifié, il peut être décoré avec le mot-clé *static* dans Solidity ++, par exemple :

```
pragma solidity++ ^0.1.0;

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

7.4 Bloc élagage

Dans un livre, toute transaction est immuable et les utilisateurs peuvent uniquement ajouter de nouvelles transactions au livre sans modifier ou supprimer les transactions historiques. Par conséquent, avec l'opération du système, les livres deviendront de plus en plus gros. Si un nouveau noeud qui rejoint le réseau veut restaurer le dernier état, en commençant par le bloc de genèse et en refaisant toutes les transactions historiques. Après avoir exécuté le système pendant un certain temps, l'espace occupé par le livre de comptes et le temps nécessaire pour refaire les transactions deviendront inacceptables. Pour le système à haut chargement et déchargement de Vite, le taux de croissance sera beaucoup plus élevé que Bitcoin et Ethereum, il est donc nécessaire de fournir une technique pour écrêter les blocs dans les livres.

BL'écrêtage de bloc fait référence à la suppression des transactions historiques qui ne peuvent pas être utilisées dans les livres, et n'affecte pas l'opération de machine d'état transactionnelle. Alors, quelles transactions peuvent être supprimées en toute sécurité ? Cela dépend du scénario dans lequel la transaction sera utilisée, y compris :

— **Récupération**.Le rôle principal d'une transaction est de récupérer le statut. Parce que dans Vite, la chaîne d'instantané stocke les informations d'instantané sur l'état du compte, les noeuds peuvent récupérer l'état à partir d'un bloc d'instantané. Toutes les transactions avant *lastTransaction* dans le bloc de capture instantanée peut être adapté à la récupération d'état.

— **Vérification des transactions**.Pour vérifier une nouvelle transaction, elle doit vérifier la transaction précédente de l'échange dans la chaîne de compte, et s'il s'agit d'une transaction de réponse, elle doit également vérifier la transaction de demande correspondante. Par conséquent, dans les livres comptables personnalisés, au moins une dernière transaction doit être conservée dans chaque chaîne de compte. En outre, toutes les transactions de demande ouvertes ne peuvent pas être adaptées car leurs hachages peuvent être référencés par des transactions de réponse ultérieures.

— **Calculer les quotas.**Si une transaction atteint le quota, elle est calculée en évaluant la moyenne glissante des 10 dernières ressources de transaction, de sorte qu'au moins les 9 dernières transactions doivent être sauvegardées sur chaque chaîne de compte.

— **Renseignez-vous sur l'histoire**. Si le noeud a besoin d'interroger l'historique des transactions, la transaction impliquée dans la requête ne sera pas personnalisée.

Selon différents scénarios d'usage, chaque nud peut choisir plusieurs combinaisons a partir de la stratégie de découpage ci-dessus. Il est important de noter que l'écrêtage implique des transactions dans les livres, tandis que les chaînes d'instantanés doivent rester intactes. En outre, ce qui est enregistré dans la chaîne d'instantanés est le hash de l'état du contrat. Lorsque le compte est écrêté, l'état correspondant de l'instantané doit rester intact.

Afin de garantir l'intégrité des données Vite, nous devons conserver certains ńnuds completsż dans le réseau pour enregistrer toutes les données de transaction. Les nuds de groupe consensus Snapshot sont des nuds complets, et en outre, des utilisateurs importants tels que des échanges peuvent également devenir des nuds complets.

8 Gouvernance

Pour une plate-forme d'application décentralisée, un système de gouvernance efficace est essentiel pour maintenir un écosystème. L'efficacité et l'équité doivent être prises en compte lors de la conception des systèmes de gouvernance.

Le système de gouvernance de Vite est divisé en deux parties : en-chaine et dehors-chaîne. en-chaine est un mécanisme de vote basé sur le protocole, et dehors-chaîne est l'itération du protocole lui-même.

Sur le mécanisme de vote, il est divisé en deux types : le vote global et le vote local. Le vote global est basé sur le *vite* détenu par l'utilisateur pour calculer les droits comme poids de vote. Le vote global est principalement utilisé pour l'élection du nud proxy du groupe de consensus d'instantanée. Le vote local vise un contrat. Lorsque le contrat est déployé, un token est désigné comme base pour le vote. Il peut être utilisé pour choisir les nuds d'agent du groupe de consensus dans lequel se trouve le contrat.

Outre la vérification des transactions, le noeud agent du groupe de consensus d'instantanée a le droit de choisir de mettre à niveau l'Incompatibilité du système Vite. Le noeud proxy du groupe de consensus délégué a le droit de décider s'il faut autoriser la mise à niveau du contrat afin d'éviter les risques potentiels découlant de l'escalade des contrats. Le nud agent est utilisé pour améliorer le pouvoir de décision au nom des utilisateurs afin d'améliorer l'efficacité de la prise de décision et d'éviter l'échec de la prise de décision en raison d'une participation insuffisante au vote. Ces nuds proxy eux-mêmes sont également limités par un protocole de consensus. Uniquement si la plupart des [1] Les nuds d'agent sont passés, la mise à niveau prendra effet. Si ces agents ne remplissent pas leur pouvoir de décision en fonction des attentes de l'utilisateur, les utilisateurs peuvent également annuler leur qualification de proxy par voter.

La gouvernance hors-chaîne est réalisée par la communauté. Tout participant de la communauté Vite peut proposer un plan d'amélioration pour le protocole Vite lui-même ou des systèmes connexes, appelé VEP (Vite Enhancement Proposal). VEP peut être largement discuté dans la communauté et si la mise en uvre de la solution est décidée par les participants écologiques Vite. Si le protocole sera mis à niveau pour la mise en uvre d'un VEP sera finalement décidé par le nud de l'agent. Bien sûr, lorsque les différences sont importantes, vous pouvez également commencer un tour de vote sur la chaîne pour recueillir un large éventail d'opinions d'utilisateurs, et le noeud proxy décidera s'il faut mettre à niveau en fonction du résultat du vote.

Bien que certains participants de Vite n'aient pas assez de token *vite* pour voter pour leurs opinions. Mais ils peuvent soumettre librement le VEP et exprimer pleinement leurs avis. Les utilisateurs qui ont le droit de vote doivent prendre pleine responsabilité de la santé de l'ensemble de l'écologie pour leurs propres droits de Vite, et donc ils doivent prendre au sérieux les opinions de tous les participants écologiques.

9 Tâches à terme

La vérification des transactions sur les chaînes d'instantané est un goulot d'étranglement majeur sur des performances du système. Parce que Vite adopte une structure asynchrone et une structure de compte DAG, la validation des transactions peut être exécutée en parallèle. Cependant, dû à la dépendance entre les transactions de différents comptes, le degré de parallélisme sera fortement restreint. Comment améliorer le parallélisme de la vérification des transactions ou adopter une stratégie de vérification distribuée sera une sujet importante pour l'optimisation future.

Certaines lacunes existent également dans l'algorithme de consensus HDPoS actuel. C'est aussi une direction de sujet d'optimisation pour améliorer l'algorithme de consensus, ou pour être compatible avec plus d'algorithmes de consensus dans le groupe de consensus délégué.

En outre, l'optimisation de la machine virtuelle est également très importante pour réduire les délais du système et améliorer le chargement et déchargement du système. En raison de la conception simple d'EVM et de la simplification de l'ensemble d'instructions, il peut être nécessaire de concevoir une machine virtuelle plus performante et de définir un langage de programmation de contrat intelligent avec plus de capacité à décrire et moins de failles de sécurité.

Enfin, outre l'accord de noyau de Vite, la construction d'installations auxiliaires qui supporte le développement écologique est également un sujet important. En plus de la prise en charge SDK pour les développeurs dApp, il y a beaucoup de travail à faire dans la construction d'un écosystème de premier plan dApp. Par exemple, vous pouvez créer un moteur dApplet basé sur HTML5 dans l'application de portefeuille mobile de Vite, ce qui permet aux développeurs de développer et de publier dApp avec un coût moins cher.

10 Sommaire

Par rapport à d'autres projets similaires, les caractéristiques de Vite comprennent :

- **Haut chargement et déchargement**.Vite utilise la structure du livre DAG, la transaction orthogonale peut être écrite en parallèle au livre ; en outre, plusieurs groupes de consensus ne dépendent pas l'un de l'autre dans l'algorithme de consensus HDPoS et peuvent travailler en parallèle ; le plus important est que la communication interne de contrat de Vite est basée sur le modèle asynchrone du message. Tout cela est utile pour améliorer le chargement et déchargement du système.
- **Faible retard**.Vite utilise l'algorithme de consensus HDPoS pour collaborer pour terminer le bloc de production de rotation via le noeud proxy, sans avoir besoin de calculer le PoW, l'intervalle de bloc peut être réduit à 1 seconde, ce qui est bénéfique pour réduire le délai de confirmation de transaction.
- **Évolutivité**.Afin de répondre aux exigences d'évolutivité, Vite limite la transaction à un seul degré de liberté, en regroupant les transactions dans le compte en fonction de la dimension du compte, ce qui permet de compléter la production de différents comptes par différents nuds et de supprimer La sémantique ACID des appels de contrat croisés à la sémantique BASE basée sur le message. De cette façon, les nuds n'ont plus besoin de sauvegarder tout l'état du monde, et les données sont sauvegardées dans le réseau distribué entier en mode sharding.
- **Utilisabilité**.Les améliorations de l'utilisabilité de Vite comprennent le support de bibliothèque standard dans Solidité++, dédié au traitement de la syntaxe des messages, le calendrier de contrat, les services de nommage VNS, la prise en charge de la

1. according to DPoS protocol, the valid majority is 2/3 of total agent nodes.

mise à niveau des contrats, etc.

— **La circulation des valeurs**.Vite prend en charge l'émission d'actifs numériques, le transfert de valeur de la chaîne croisée, l'échange de token basé sur le protocole Loopring, etc., formant ainsi un système de valeur complet. Du point de vue de l'utilisateur, Vite est un échange décentralisé entièrement fonctionnel.

— **Économie**.Parce que Vite adopte un modèle d'allocation des ressources basé sur les quotas, les utilisateurs légers qui ne négocient pas fréquemment n'ont pas à payer des frais élevés ou des frais de gaz. Les utilisateurs peuvent choisir différentes façons de changer le calcul. Un quota supplémentaire peut également être transféré à d'autres utilisateurs via un contrat de location de quota pour améliorer l'efficacité de l'utilisation des ressources du système.

11 Remerciement

Sincèrement, nous tenons à remercier nos consultants pour leurs conseils et leur aide à cet article. Nous aimerions particulièrement apprécier la contribution de l'équipe Loopring et de la communauté Loopring à ce projet.

Remerciement à ZHANG Xuan pour le travail de traduire ce White Paper en version français.

Références

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

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

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

[5] Anonymous. Eos.io technical white paper v2. *URL https ://github.com/EOSIO/Documentation/blob/master/TechnicalWhiteP*

[6] 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.

[7] Ed Eykholt, Lucius Meredith, and Joseph Denman. Rchain platform architecture. *URL http ://rchain-architecture.readthedocs.io/en/latest/*.

[8] Anonymous. Neo white paper a distributed network for the smart economy. *URL http ://docs.neo.org/en-us/index.html*.

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

[10] Anonymous. Byzantine consensus algorithm. *URL https ://github.com/tendermint/tendermint/wiki/Byzantine-Consensus-Algorithm*.

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

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

[13] Colin LeMahieu. Raiblocks : A feeless distributed cryptocurrency network. *URL https ://raiblocks.net/media/RaiBlocks_Whitepaper___English.pdf*.

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

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

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

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

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

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

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

Appendices

Annexe A EVM Instruction set

A.0.1 0s : Stop and algebraic operation instruction set

No.	Words	POP	PUSH	Semantics in EVM	Semantics in Vite
0x00	STOP	0	0	Stop to Excute.	Sanme semantics
0x01	ADD	2	1	Add two operands.	Same semantics
0x02	MUL	2	1	Multiplying two operands.	Same semantics
0x03	SUB	2	1	Subtracting two operands.	Same semantics
0x04	DIV	2	1	Divide two operands If the divisor is 0 then returns 0	Same semantics
0x05	SDIV	2	1	Divided with symbol.	Same semantics
0x06	MOD	2	1	Modulus Operation.	Same semantics
0x07	SMOD	2	1	Modulus with symbol.	Same semantics
0x08	ADDMOD	3	1	Add the first two operands and module with 3rd	Same semantics
0x09	MULMOD	3	1	Mmultiply the first two operands and module with 3rd	Same semantics
0x0a	EXP	2	1	The square of two operands.	Same semantics
0x0b	SIGNEXTEND	2	1	Symbol extension.	Same semantics

A.0.2 10s : Comparison and bit operation instruction set

No.	Words	POP	PUSH	Semantics in EVM	Semantics in Vite
0x10	LT	2	1	less than.	Same semantics
0x11	GT	2	1	greater than.	Same semantics
0x12	SLT	2	1	less than with symbol.	Same semantics
0x13	SGT	2	1	greater than with symbol.	Same semantics
0x14	EQ	2	1	equal to.	Same semantics
0x15	ISZERO	1	1	if it is 0.	Same semantics
0x16	AND	2	1	And by bit.	Same semantics
0x17	OR	2	1	Or by bit.	Same semantics
0x18	XOR	2	1	Xor by bit.	Same semantics
0x19	NOT	1	1	Nor by bit.	Same semantics
0x1a	BYTE	2	1	Take one of byte from the second operands.	Same semantics

A.0.3 20s : SHA3 instruction set

No.	Words	PoP	PUSH	Semantics in EVM	Semantics in Vite
0x20	SHA3	2	1	Calculate Keccak-256 hash.	Same semantics

A.0.4 30s : Environmental information instruction set

No.	Words	POP	PUSH	Semantics in EVM	Semantics in Vite
0x30	ADDRESS	0	1	Obtain address . of current account	Same semantics
0x31	BALANCE	1	1	Obtain the balance of an account.	Same semantics. returned is the *vite* balance of account
0x32	ORIGIN	0	1	Obtain the sender addresss of original transaction	Different samantics return 0 forever Vite doesn't maintain the causal relationship between internal transaction and user transaction.
0x33	CALLER	0	1	Obtain the address of direct caller.	Same semantics.
0x34	CALLVALUE	0	1	Obtain the transferred amount in called transaction.	Same semantics
0x35	CALLDATALOAD	1	1	Obtain the parameter in this calling	Same semantics
0x36	CALLDATASIZE	0	1	Obtain size of parameter data in this calling.	Same semantics
0x37	CALLDATACOPY	3	0	Copy called parameter data into memory.	Same semantics
0x38	CODESIZE	0	1	Obtain the size of the running code in current environment.	Same semantics
0x39	CODECOPY	3	0	Copy the running code in current environment into memory.	Same semantics
0x3a	GASPRICE	0	1	Obtain the gas . price in current enviroment	Different samantics ,return 0 forever.
0x3b	EXTCODESIZE	1	1	Obtain the code size of an account.	Same semantics
0x3c	EXTCODECOPY	4	0	Copy the code of. an account into memory	Same semantics
0x3d	RETURNDATASIZE	0	1	Obtain data size of returned from previous calling.	Same semantics
0x3e	RETURNDATACOPY	3	0	Copy the returned data calling previously into memory into memory.	Same semantics

A.0.5 40s : Block info instructions set

No.	Words	POP	PUSH	Semantics in EVM	Semantics in Vite
0x40	BLOCKHASH	1	1	Obtain hash of a block.	Different semantic. return Hash of corresponing snapshot block.
0x41	COINBASE	0	1	Obtain the address. of miner beneficiary in current block	Different semantic. return 0 forever.
0x42	TIMESTAMP	0	1	Return timestamp of current block.	Different semantic. return 0 forever.
0x43	NUMBER	0	1	Return the number or current block.	Different semantic. Return the number of responding transaction block in account chain
0x44	DIFFICULTY	0	1	Return the difficulty of the block.	Different semantic. return 0 forever.
0x45	GASLIMIT	0	1	Return the gas. limitation of the block	Different semantic. return 0 forever.

A.0.6 50s : Stach, Memory, Storege, Control stream operation instruction set

No.	Words	POP	PUSH	Semantics in EVM	Semantics in Vite
0x50	POP	1	0	Pop one data from top of stack.	Same semantics
0x51	MLOAD	1	1	load a word from memory.	Same semantics
0x52	MSTORE	2	0	Save a word to memory	Same semantics
0x53	MSTORE8	2	0	Save a byte to memory.	Same semantics
0x54	SLOAD	1	1	Load a word from storage.	Same semantics
0x55	SSTORE	2	0	Save a word into storage.	Same semantics
0x56	JUMP	1	0	Jump instructions.	Same semantics
0x57	JUMPI	2	0	Jump instructions with condition.	Same semantics
0x58	PC	0	1	Obtain program counter's value.	Same semantics
0x59	MSIZE	0	1	Obtain size of memory.	Same semantics
0x5a	GAS	0	1	Obtain available gas .	Different semantic. return 0 forever.
0x5b	JUMPDEST	0	0	Mark a destination of jumping .	Same semantics

A.0.7 60s and 70s : Stack operation instructions

No.	Words	POP	PUSH	Semantics in EVM	Semantics in Vite
0x60	PUSH1	0	1	Push one byte object. into top of stack	Same semantics
0x61	PUSH2	0	1	Push two bytes object into top of stack.	Same semantics
⋮	⋮	⋮	⋮	⋮	
0x7f	PUSH32	0	1	Push 32 bytes object (whole word) into top of stack	Same semantics

A.0.8 80s : Duplication operation instructions

No.	Words	POP	PUSH	Semantics in EVM	Semantics in Vite
0x80	DUP1	1	2	Duplicate 1st object and push it into top of stack.	Same semantics
0x81	DUP2	2	3	Duplicate 2nd object . and push it into top of stack.	Same semantics
⋮	⋮	⋮	⋮	⋮	
0x8f	DUP16	16	17	Duplicate 16th object and push it into top of stack.	Same semantics

A.0.9 90s : Swap operation instructions

No.	Words	POP	PUSH	Semantics in EVM	Semantics in Vite
0x90	SWAP1	2	2	Swap 1st and 2nd object in stack.	Same semantics
0x91	SWAP2	3	3	Swap 1st and 3rd object in stack.	Same semantics
⋮	⋮	⋮	⋮	⋮	
0x9f	SWAP16	17	17	Swap 1st and 17th object in stack.	Same semantics

A.0.10 a0s : Log operation instructions

No.	Words	POP	PUSH	Semantics in EVM	Semantics in Vite
0xa0	LOG0	2	0	Extend log record, no scheme.	Same semantics
0xa1	LOG1	3	0	Extend log record,. 1 scheme	Same semantics
⋮	⋮	⋮	⋮	⋮	
0xa4	LOG4	6	0	Extend log record,. 4 schemes	Same semantics

A.0.11 f0s : System operation instructions

No.	Words	POP	PUSH	Semantics in EVM	Semantics in Vite
0xf0	CREATE	3	1	Create a new contract.	Same semantics
0xf1	CALL	7	1	Call another contract.	Different semantic. indicate sending a message to an account The returned vale is 0 forever.
0xf2	CALLCODE	7	1	Call the code of another contract Change the status of account.	Same semantics
0xf3	RETURN	2	0	Stop execution and return value.	Same semantics
0xf4	DELEGATECALL	6	1	Call the code of another contract, change contract, change current account status keep original transaction info.	Same semantics
0xfa	STATICCALL	6	1	Call another contract, not allow to change status.	Different semantic. represents to sending message to a contract, don't change status of target contract. return 0 forever.needed result Sending another message through target contract and return.
0xfd	REVERT	2	0	Stop execution and . recover status and return value	Same semantics no semantics of returning left gas.
0xfe	INVALID	∅	∅	invalid instructions.	Same semantics
0xff	SELFDESTRUCT	1	0	Stop execution, set the contract as waiting for deleting return all balance.	Same semantics