VITE: 고성능 비동기식 분산 애플리케이션 플랫폼

VITE는 높은 처리율과 낮은 지연성 그리고 확장성을 위한 산업 애플리케이션의 요건을 만족시키는, 그리고 동시에 보안 문제를 참작한 종합적이고 분산된 애플리케이션 플랫폼이다. VITE는 DAG 원장 구조를 사용하며 원장에 있는 트랜잭션은 계정 별로 분류된다. VITE에서 스냅숏 체인 구조는 DAG 원장의 보안 부족 문제를 보완한다. HDPoS 합의 알고리즘은 높은 성능과 확장성을 제공하며, 저 알고리즘을 통한 트랜잭션의 기록 및 확인은 비동기식이다. VITE VM은 EVM과 솔리디티로부터 확장한 스마트계약 언어와 바꾸어 사용할 수 있으며(호환성) 더욱 강력한 기술 능력을 제공한다. 이에 더해 VITE 설계에서 중요한 개선은 스마트계약 사이의 메시지를 통하여서 정보를 전송하는 비-동시적인 사건 기반(event-driven) 아키텍처를 채택한 일이며 개발자는 이를 통해 시스템의 처리율과 확장성을 매우 개선시켰다. 내장 고유 토큰(native token)과 더불어 VITE는 또한 사용자가 자신의 디지털 자산을 발행하는 일을 지원한다. 그리고 루프링 프로토콜에 기반한 크로스 체인 가치 전송과 교환 서비스를 제공한다[1]. VITE는 할당량으로 자원 할당을를 실현한다. 그리고 구매빈도가 낮은 사용자는 트랜잭션 수수료를 지불하지 않아도 된다. VITE는 또한 계약 일정 관리, 이름 서비스, 계약 갱신, 블록 제거, 그리고 다른 특징적인 기능을 지원한다.

1 서론

1.1 정의

VITE는 스마트 계약 집합을 지원할 수 있는 보편적인 분산 애플리케이션 플랫폼이다. 이들 각각은 독립적인 상태와 다른 작동 논리를 가진 상태 기계이며, 메시지 전달을 통하여서 의사 소통하는 일이 가능하다.

일반적으로 시스템은 트랜잭션을 사용하는 상태 기계다. s∈S인 시스템의 상태는, 세계 상태로 또한 알려졌으며 각 독립적인 계정의 상태로 구성된다. 계정 지위에서 변화를 야기하는 사건을 트랜잭션이라고 부른다. 더욱 공식적인 정의는 다음과 같다.

정의 1.1 (트랜잭션 상태 기계) *트랜잭션 상태 기계는 4개 튜플 즉* ***(T, S, g, δ)****로 이루어지며* ***T****가 트랜잭션의 집합이고* ***S****가 상태의 집합일 때* $g \in S$*는 초기 상태이며 최초 블록(genesis block)이라고 알려져 있다.* $\delta : S \times T \to S$*는 상태 변화 함수다.*

이 트랜잭션 상태 기계의 의미는 개별적인 변화 시스템이며 다음과 같이 정의한다.

정의 1.2 (트랜잭션 활용 상태 기계의 의미) *트랜잭션 상태 기계* ***(T, S, g, δ)****의 어의는 개별적인 변화 시스템이다.* $(S, s_0, \to) \to \in S \times S$*는 변화 관계이다.*

동시에 분산애플리케이션 플랫폼은 최종 일관성을 가지는 분포된 시스템이다. 어떤 합의 알고리즘을 통해서 노드 사이에서 최종 상태에 도달할 수 있다. 현실적인 시나리오들에선 스마트 완성된 데이터 집합으로, 데이터 집합은 분산된 애플리케이션의 형태를 띠며 큰 부피를 차지하며 노

드 사이에서 전송될 수 없다. 따라서 노드는 일관된 최종 상태를 달성하기 위해 일련의 트랜잭션들을 이전시킬 필요가 있다. 우리는 이러한 트랜잭션 그룹을 하나의 구체적인 데이터 구조로 조직할 것인데, 이러한 조직은 대개 원장이라고 불린다.

정의 1.3 (원장) *원장은 반복적으로 구성된 한 추상적인 데이터 형태를 가진 일련의 트랜잭션들로 구성된다. 이것은 다음과 같이 정의된다.*

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

이 수식에서 $T_t \in S^T$는 일련의 트랜잭션들을 나타내며 $\Gamma \in 2^T \rightarrow L$은 일련의 트랜잭션들을 통해 한 장부를 구성하는 함수를 나타내며, L은 일련의 원장들이며 $+: L \times L \rightarrow L$은 두 하위 원장을 하나로 통합하는 연산을 나타낸다.

이러한 시스템에서 원장을 대개 상태라기보다는 오히려 트랜잭션 그룹을 나타내기 위해 사용된다는 사실을 유념해야 한다. 비트코인과 이더리움에서 원장은 블록 체인 구조이며, 이 구조 속에서 트랜잭션들은 세계적으로 질서가 정해진다. 원장에서 트랜잭션을 변경하기 위해서 우리는 계정 장부에서 하위 원장을 재구성할 필요가 있는데, 그렇게 함으로써 트랜잭션을 할 때마다 점점 줄어드는 비용을 늘리게 된다.

같은 트랜잭션 그룹에 따라 각기 다른 유효 장부들을 만들 수 있지만, 그러나 그들은 트랜잭션들의 다른 순서를 나타내며 시스템이 한 다른 상태로 진입하는 일을 초래할 수 있다. 이런 일이 발생할 때, 우리는 이를 대개 "분기"라고 부른다.

정의 1.4 (분기) *T_t, $T_t' \in 2^T$, $T_t \subseteq T_t'$라고 가정해 보자. 만약 $l = \Gamma_1(T_t)$, $l' = \Gamma_2(T_t')$이고 $l \preceq l'$에 부합하지 않으면 우리는 l과 l'가 분기된 원장이라고 명명할 수 있다.$\preceq$는 접두 관계를 나타낸다.*

트랜잭션 상태 기계의 의미에 따라 우리는 만약 원장이 분기되지 않는다면 초기 상태로부터 각 노드가 궁극적으로 같은 상태에 진입하리라는 사실을 쉽게 증명할 수 있다. 만약 분기된 원장을 받는다면, 이 원장은 틀림없이 다른 상태에 진입하게 될까? 그것은 원장의 트랜잭션의 내재 논리 회로와 원장이 두 트랜잭션 사이에서 부분 순서들을 조직하는 방식에 달려 있다. 실제 세계에 종종 교환 법칙을 만족시키는 일부 트랜잭션들이 있지만, 그러나 계정 설계의 문제 때문에 그들은 자주 분기를 야기한다. 시스템이 초기 상태에서 시작하고, 두 개의 분기된 원장을 받고 같은 상태에서 끝이 난다면 우리는 이 두 원장을 거짓 분기된 원장이라고 일컫는다.

정의 1. 5(거짓 분기) *초기 상태 $S_0 \in S$, 원장 $l_1, l_2 \in L$, $S_0 \xrightarrow{l_1} S_1$, $S_0 \xrightarrow{l_2} S_2$. 만약 $l_1 \neq l_2$이고 $S_1 = S_2$이면 우리는 이 두 원장 l_1, l_2을 거짓 분기된 원장이라고 부른다.*

잘 설계된 원장은 거짓 분기 발생 확률을 최소로 줄여야 한다.

분기 현상이 발생했을 때 각 노드는 다수의 분기된 원장에서 하나를 선택할 필요가 있다. 상태의 일관성을 확고히 하기 위해서 노드를 써서 선택을 완료하기 위해서 같은 알고리즘을 사용할 필요가 있다. 이 알고리즘을 합의 알고리즘이라고 부른다.

정의 1.6 (합의 알고리즘) 합의 알고리즘은 원장 집합을 받아들이고 유일한 원장으로 되돌아가는

함수이다.

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

합의 알고리즘은 시스템 설계에 중요한 일부이다. 좋은 합의 알고리즘은 높은 수렴속도를 가져야 하며 이는 다양한 분기에서 합의 중 알고리즘이 동요하는 일을 줄이기 위함이다. 그리고 해로운 공격으로부터 방어할 고도의 능력을 가지기 위함이다.

1.2 최근 연구

이더리움은 이러한 시스템의 실현을 이끌었다. 이더리움의 설계에서 세계 상태의 정의는 $S = \Sigma^{A}$ 이고 $a \in A$ 계정과 $\sigma_a \in \Sigma$ 계정의 상태에서 도출한 함수이다. 그러므로 이더리움의 상태 기계에서 어떤 상태는 세계적이고 이 말은 노드가 어떤 때에 어떤 계정의 상태를 획득하는 일이 가능하다는 뜻이다.

이더리움의 상태 변화 함수인 δ를 프로그램 코드 집합으로 정의한다. 코드의 각 그룹을 스마트 계약이라고 부른다. 이더리움은 튜링 기계를 완전 가상 기계로 정의하며 이더리움 가상머신(EVM)이라고 부른다. EVM의 지침 집합을 EVM 코드라고 부른다. 사용자는 자바스크립트와 비슷한 프로그래밍 언어인 솔리디티를 써서 스마트 계약을 발전시킬 수 있다. 그리고 그들을 EVM 코드로 컴파일하며 이더리움에서 그들을 전개한다. 스마트 계약을 성공적으로 전개했을 때, 그것은 계약 계정 a를 정의하는 일과 마찬가지이며 상태 변화 함수 δ_a를 얻는다. EVM은 이러한 플랫폼에서 폭넓게 사용하지만 또한 어떤 문제가 존재한다. 예를 들어서, 라이브러리 함수 지원의 부족과 보안 문제가 존재한다.

이더리움의 원장 구조는 블록 체인이다. 블록이 블록체인을 구성하며 각 블록에 트랜잭션 목록을 포함하고 맨 뒤의 블록이 체인 구조를 형성하기 위한 이전의 블록의 해시를 의미한다.

$$\Gamma(\{t_1,\ t_2,\ \ldots \mid t_1,\ t_2,\ \ldots \in T\}) = (\ldots, (t_1,\ t_2,\ \ldots)) \qquad (1)$$

이 구조의 가장 큰 이점은 간섭으로부터 트랜잭션을 효과적으로 방지할 수 있다는 점이다. 그러나 저 함수는 모든 트랜잭션의 완전한 순서를 유지하기 때문에 두 트랜잭션 순서의 교환으로 새로운 원장을 생성할 것이다. 이 원장은 더 높은 분기 가능성을 가진다. 실제로, 이 정의에 따라서 트랜잭션 상태 기계의 상태 공간을 우리는 나무라고 간주한다. 초기 상태는 뿌리 노드이고, 다른 트랜잭션 순서가 다른 경로를 나타낸다. 그리고 잎 노드는 최종 상태다. 실제로 큰 수의 잎 노드의 상태는 같다. 이 사실로 다수의 거짓 분기가 발생한다.

합의 알고리즘 Φ는 PoW라고 부르며, 비트코인 프로토콜에서 처음으로 제언하였다. PoW 알고리즘은 쉽게 증명할 수 있지만 풀기가 어려운 수학적 문제에 의존한다. 예를 들어서, 해시 함수인 $h : N \rightarrow N$에 근거할 경우, x의 결과를 찾는다면 $h(T + x) \geq d$의 요건을 충족시키는 d는 주어진 수이며, 난이도(difficulty)의 약자이고, T는 블록에 포함된 트랜잭션 목록을 이진법적으로 보여준다. 블록 체인에서 각 블록에 이러한 문제의 해결책을 포함한다. 문제를 풀기 위해서 블록 체인 원장의 전체 난이도인 모든 블록의 난이도를 더해야 한다.

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

그러므로 분기로부터 올바른 계정을 선택할 때 가장 높은 난이도를 가진 분기를 선택한다.

$$\Phi(l_1, l_2, \ldots l_n) = l_m \text{이고 이 경우 m= 독립변수 } \max_{i \in 1..n}\left(D(l_1)\right) \text{이다.} \quad (3)$$

PoW 합의 알고리즘은 더 나은 보안성을 가지며 비트코인과 이더리움에서 잘 작동하였다. 그러나 이 알고리즘에 두 가지 주요한 문제가 존재한다. 첫 번째 문제는 대량의 컴퓨팅 자원을 요구하는 수학 문제를 풀어야 한다는 점이고 결과적으로 에너지의 낭비를 낳는다는 점이다. 두 번째 문제는 알고리즘의 느린 수렴 속도로 시스템의 전반적인 처리량에 영향을 미친다. 현재 이더리움의 TPS는 오직 15이고, 분산된 애플리케이션의 수요를 맞추기가 전적으로 불가능하다.

1.3 개선 방향

이더리움이 탄생한 이후에, 이더리움 사회와 다른 비슷한 프로젝트로 개발자들이 다양한 방향에서 시스템을 개선시키기 시작했다. 시스템의 추상적인 모델로부터 다음의 목표를 개선시킬 수 있다.

- 시스템 상태 S의 개선
- 상태 변화 함수 δ의 개선
- 원장 Γ의 시스템 개선
- 합의 알고리즘 Φ의 개선

1.3.1 시스템 상태 개선

시스템 상태 개선의 주요한 생각은 세계의 세계적(global) 상태를 지역화 하는 일이다. 각 노드는 더 이상 모든 트랜잭션과와 상태 이전(transfer)에 관여하지 않으며 오직 전체 상태 기계의 하위집합을 유지할 뿐이다. 이 방식에서 집합 S와 집합 T의 잠재력은 대단히 줄어들고 그러므로 시스템의 확장성을 개선시킨다. 이러한 시스템에 코스모스, 알레프, P체인 등을 포함한다.

본질적으로 설계에 근거한 이 사이드 체인은 시스템 상태의 총합을 희생하고 그 대신 확장성을 획득한다. 이 때문에 저 위에서 작동하는 각각의 분산 애플리케이션의 분산성을 약화시킨다. 그러므로 스마트 계약의 트랜잭션 기록을 더 이상 전체 네트워크에서 모든 노드를 사용해서 저장하지 않으며 그러나 오직 노드의 일부를 사용해서 저장한다. 게다가, 크로스 계약 상호작용이 이러한 시스템의 진행을 방해할 것이다. 예를 들어서 코스모스에서는 다른 구역에서 발생하는 상호작용에 완성을 위한 공통적인 체인 허브를 요구한다.

1.3.2 상태 변화 함수 개선

EVM 개선에 근거하였을 때 어떤 프로젝트는 더욱 충분한 스마트 계약 프로그래밍 언어를 제공한다. 예를 들어 스마트 계약 언어인 롤랑은 π계산에 근거한 RChain에서 정의한다. 그러므로 NEO의 스마트 계약을 NeoContract라고 부르며, 저 계약으로 자바, C# 등과 같은 대중적인 프로그래밍 언어를 사용하여서 프로그램을 개발하는 일이 가능하다. EOS는 C/C++를 써서 프로그래밍한다.

1.3.3 원장 구조 개선

원장 구조의 개선 목표는 동등한 계층의 구축에 있다. 여러 트랜잭션의 세계적 질서를 포함하는 선형 원장은 비선형 원장으로 개선되었고 저 원장은 오직 부분적인 순서 관계만을 기록한다. 이 비선형 원장 구조는 DAG(방향성이 있는 비-순환 그래프)이다. 현재 바이트볼과 IOTA, 나노, 그리고 다른 프로젝트로 개발자들이 DAG 계정 구조에 근거한 암호화 화폐 기능을 실현하였다. 어떤

프로젝트에서 개발자들은 스마트 계약을 이행하기 위해서 DAG를 사용하려는 시도를 하며, 그러나 여태까지 이 방향에서 개선과 관련된 연구를 진행하는 중이다.

1.3.4 합의 알고리즘의 개선

합의 알고리즘의 개선은 대체로 시스템의 처리량을 개선시킨다. 그리고 주요한 방향은 거짓 분기의 생성을 제압하는 일이다. 다음에서 우리는 거짓 분기에 어떤 요인이 포함되는지 논의할 것이다.

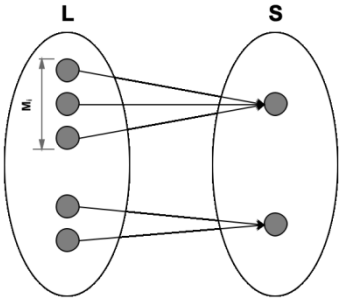

그림 1: 거짓 분기

보여지는 바와 같이 L은 일련의 트랜잭션들을 위한 모든 가능한 분기 계정의 모음이다. 그리고 S는 다른 순서에 닿는 일이 가능한 상태의 모음이다. 정의 1.4에 따르면 함수 f : L → S는 전사 함수이다. 그리고 정의 1.5에 따르면 이 함수는 단사 함수가 아니다. 여기에서 우리는 거짓 분기 확률을 계산한다.

C 사용자가 $L_i=\{l \mid f(l) = s_i,\ s_i \in S\}$일 때$M= \mid L \mid$, $N= \mid S \mid$, $M_i= \mid L_i \mid$ 인 원장을 제공할 권리를 가졌다고 가정해 보자. 거짓 분기 확률은 다음과 같다.

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

이 공식에서 우리는 거짓 분기 확률을 줄이기 위한 두 가지 방법이 존재함을 본다.

- 원장 집합의 L과 동등한 관계를 수립하고 같은 계층으로 그들을 나누고 더 적은 수의 분기된 원장을 만든다
- 원장을 생산할 권리는 가진 사용자를 제한하고 그럼으로써 C를 줄인다.

VITE 설계에서 첫 번째 방식은 중요한 목적을 가진다. 두 번째 방식은 다수의 알고리즘에서 채택하였다. PoW 알고리즘에서 어떤 사용자는 블록을 만들 권리는 가진다. 그리고 PoS 알고리즘은 시스템 권리를 가진 사람들에 이르기까지 블록 생산이 가지는 힘을 제한한다. DPoS 알고리즘은 에이전트 노드의 그룹 내에서 블록을 생산할 권리를 가진 사용자를 한층 더 제한한다.

오늘날 개선된 합의 알고리즘을 거치면서 어떤 영향력을 가진 프로젝트가 나타났다. 예를 들어 카르다노는 우로보로스라고 불리는 PoS 알고리즘을 사용한다. 그리고 문헌을 볼 경우 알고리즘의 관련된 특성에 대한 엄격한 증거가 제시된다. EOS가 사용하는 BFT-DPOS 알고리즘은 DPoS 알고리즘의 변형이며 빠르게 블록을 생산함으로써 시스템 처리량(throughout)을 개선시킨다. 퀀텀의 합의 알고리즘은 또한 PoS 알고리즘이다. RChain이 채택한 캐스퍼 알고리즘은 마찬가지로 PoS 알고리즘 가운데 하나이다.

합의 알고리즘을 개선시키기 위해 그들 자신의 제언을 내놓은 다른 프로젝트가 또한 존재한다. 네오는 dBFT라고 불리는 BFT 알고리즘을 사용하며 코스모스는 텐더민트라고 불리는 알고리즘을 사용한다.

2 원장

2.1 개요

원장은 트랜잭션의 순서를 결정하는 역할을 한다. 그리고 트랜잭션의 순서가 다음 두 측면에 영향을 미칠 것이다.

- **상태의 일관성**: 시스템의 상태가 CRDT(충돌이 없는 복제된 데이터 유형)가 아니기 때문에 모든 트랜잭션을 교환할 수는 없다. 그리고 다른 트랜잭션 수행 배열이 다른 상태에 입장하는 시스템을 이끌 것이다.
- **해시의 효율성**: 원장에서 트랜잭션을 일괄 처리하여 블록으로 만들 것이고, 블록에 해시를 포함하며 해시는 서로를 참조한다. 트랜잭션의 순서는 원장에서 인용한 해시의 접속 가능성에 영향을 미친다. 이 영향의 범위가 더 커질수록 트랜잭션에 간섭하는데 드는 비용이 더 커진다. 이것이 트랜잭션으로의로의 어떤 변화를 반드시 해시를 써서 다시 구축해야 하는가에 대한 이유이다. 해시는 직접적으로 혹은 간접적으로 트랜잭션의 블록을 가리킨다.

원장의 설계에 두 가지 주요한 목적을 포함한다.

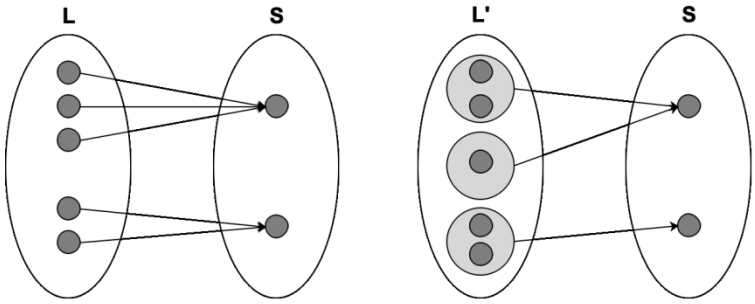

그림 2: 원장 통합

- **거짓 분기율 감소**: 이전 부분에서 언급한 바와 같이 거짓 분기율은 동등한 계층을 수립하고 시스템을 같은 상태로 이끌어 단일한 계정을 가지도록 이끄는 계정의 그룹을 결합함으로써 획득하는 일이 가능하다. 위에서 보는 바와 같이 거짓 분기율의 공식에 따르면 왼쪽에서 원장의 거짓분기비율은 $P_{ff}' = (\frac{3}{5})^C + (\frac{2}{5})^C - \frac{1}{5^{C-1}}$이다. 원장 공간을 통합한 이후에 오른쪽 그래프의 거짓 분기율은 C>1이고 $P_{ff}' < P_{ff}$일 때 $P_{ff}' = (\frac{2}{3})^C + (\frac{1}{3})^C - \frac{1}{3^{C-1}}$이라고 알려져 있다. 말하자면 우리는 트랜잭션 사이의 부분적인 순서 관계를 최소화해야 하고 더 많은 트랜잭션 교환을 허용해야 한다.
- **조작 불가능성**: 트랜잭션 t가 원장 l에서 변경될 때, $l = l_1 + l_2$인 장부의 두 하위 원장 규칙에서 하위 원장 l_1은 영향을 받지 않으며 새로운 유효 원장 $l' = l_1 + l_2'$을 형성하기 위해 하위 원장 l_2에서 해시 참조 목록을 다시 구축할 필요가 있다. 하위 원장 $l_2 = \Gamma(T_2)$가 영

향을 받을 때 $T_2 = \{x \mid x \in T, x > t\}$가 된다. 그러므로 트랜잭션과와 함께 간섭 비용을 증가시키기 위해서 간섭의 범위 $|T_2|$를 확장시키기 위해서 가능하면 많은 트랜잭션 사이의 부분적인 순서 관계를 유지하는 일이 필요하다.

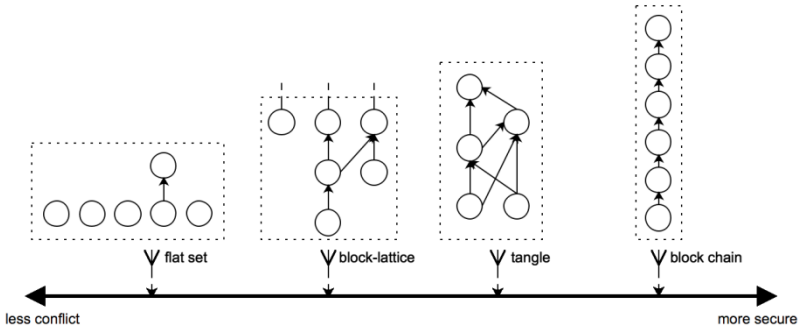

그림 3: 원장 구조 비교

분명히, 위의 두 객체는 서로 대조적이고 필요한 균형은 반드시 계정 구조를 설계할 때 만들어야 한다. 계정 유지가 트랜잭션 사이의 부분 순서 관계를 이루기 때문에 이것은 본질적으로 부분 순서 집합(poset)이다. 만약 하세 다이어그램으로 나타낸다면, 이것은 위상 수학의 범위에서 방향성이 있는 비-순환 그래프이다.

위의 그림을 통해서 우리는 공통적인 다수의 원장 구조를 비교한다. 그리고 왼쪽에 가까운 원장은 부분 순서 관계를 덜 이룬다. 하세 다이어그램은 평면으로 나타나고 더 낮은 거짓 분기율을 가진다. 오른쪽에 가까운 원장은 부분 순서 관계를 더 많이 이루는 경향을 가진다. 그리고 하세 다이어그램은 더욱 훌륭하며 간섭에 더욱 저항적이다.

그림에서 가장 왼쪽이 어떤 조작 불가능성이라는 특질을 지니지 않은 집중화된 시스템에 근거한 공통적인 집합이다. 가장 오른쪽은 전형적인 블록 체인 원장으로 가장 우수한 조작 불가능성을 지닌다. 둘 사이에 DAG 원장이 존재하며, 왼쪽의 나노가 사용하는 블록 격자 계정이다. 그리고 서로 얽힌 모양을 한 오른쪽은 IOTA가 사용한다. 특성의 면에서 보았을 때 블록 격자는 부분 순서 관계를 더 적게 이루는 경향을 유지한다. 그리고 고성능 분산 애플리케이션 플랫폼의 계정 구조에 더욱 적합하다. 형편없는 간섭 특성 때문에 보안 위험에 노출되는 일이 가능했고 여태껏 나노 를 제외하고 어떤 다른 프로젝트도 이 원장을 채택하지 않았다.

고성능을 추구하기 위해서 VITE는 DAG 원장 시스템을 채택하였다. 동시에 추가적인 체인 구조은 스냅숏 체인을 도입하고 합의 알고리즘을 개선시킴으로써 블록 격자 안보의 단점을 성공적으로 보완하였다. 그리고 두 가지 개선 사항을 이후에 세부적으로 논의한다.

2.2 사전 제약

우선 상태 기계 모델을 위한 이 원장 구조를 사용하여서 전제조건을 보도록 하자. 이 시스템은 독립적인 상태 기계로서 전체 상태 기계의 본질적인 조합으로 구성된다. 이 경우 각 계정은 독립적인 상태 기계와 일치하고 각 트랜잭션은 오직 계정의 상태에만 영향을 미친다. 원장에서 모든 트랜잭션을 계정으로 묶고 같은 계정에서 트랜잭션 체인으로 구조화한다. 그러므로 우리는 VITE에서 상태 S와 트랜잭션 T에 대한 다음의 제한 사항을 본다.

5쪽

정의 2.1 (단일 자유 제약도) *시스템 상태 s ∈ S는 각 계정의 상태 si별로 형성된 벡터 s=(s1, s2,..., sn)이다. ∀ti ∈ T 경우, 트랜잭션 ti를 수행한 후, 시스템 상태는 다음과 같이 전송된다(transfers): (s1',...,si',...,sn')= σ(ti,(s1,...,si,...,sn)) 은 sj' = sj, j ≠ i에 부합할 필요가 있다. 이 제약은 트랜잭션의 단일 자유 제약도(single degree of freedom constraint)라고 부른다.*

직관적으로 트랜잭션의 단일 자유 제약도는 이 시스템에 있는 다른 계정들의 지위에 영향을 미치지 않은 채 한 계정 상태만 바꿀 것이다. 상태 공간 벡터가 위치한 다차원 공간에서, 한 트랜잭션이 실행되고, 이어 시스템 상태는 한 좌표 축을 따라 평행 방향으로만 이전하게 된다. 이 정의는 비트 코인, 이더리움이나 기타 가상 화폐 모델들에서 정한 트랜잭션 정의보다 더 엄격하다는 점에 유의할 것. 비트 코인에서 하나의 트랜잭션은 송신자와 수신자의 계정, 즉 두 계정의 상태를 변경시킬 것이며, 이더리움 경우엔, 한 메시지 콜(message call)을 통해 두 개 이상의 계정들의 상태를 변경시킬 것이다.

이 제약 하에, 두 트랜잭션의 관계는 단순해질 수 있다. 두 트랜잭션 중 어떤 것이든 하나는 축과 직교하거나 평행을 이룬다. 이것은 계정들에 따라 트랜잭션들을 그룹으로 묶기 위한 조건들을 제공한다. 이를 설명하는 예가 여기 제시되어 있다:

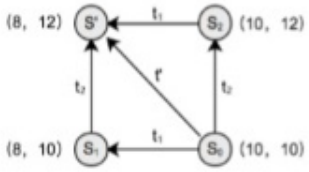

그림 4: 트랜잭션의 단일 자유도와 중간 상태

위 그림에 나온 것처럼 앨리스와 밥이 각각 10달러를 갖고 있다고 가정하자. 이 시스템의 초기 상태는 s0 = (10,10).이다. 앨리스가 2달러를 밥에게 보내고 싶을 때, 비트 코인과 이더리움 모델에선, 한 트랜잭션 t'는 최종 상태로 바로 가는 시스템 즉 s0 t' → s'를 만들 수 있다.

VITE의 정의에선, 트랜잭션 t'는 앨리스뿐만 아니라 밥의 계정까지, 즉 두 계정 모두를 변경시켰는데, 이것은 트랜잭션의 단일 자유도 원칙을 따르지 않은 것이다. 따라서 이 트랜잭션은 다음과 같이 두 개의 트랜잭션로 나누어져야 한다:

1) 앨리스가 2달러를 보내는 것을 나타내는 트랜잭션 t1
2) 밥이 2달러를 받는 것을 나타내는 트랜잭션 t2

이런 식으로, 초기 상태에서 최종 상태 s'에 이르는 사이, 두 개의 다른 경로들 즉 s0 t1 → s1 t2 → s'와 s0 t2 → s2 t1 → s'가 생길 수 있게 된다. 이 두 개의 경로들은 각각 중간 상태 s1과 s2를 통과하게 되고, 이 두 개의 중간 상태들이 두 개의 계정 도메인에서 최종 상태 s'의 맵핑

(mapping)이 된다. 다시 말해서, 이 두 계정들 중 한 개의 상태에 대해서만 신경을 쓰면, 그 계정에 상응하는 트랜잭션들만 수행하면 되고 나머지 계정들의 트랜잭션들을 실행할 필요는 없다.

이어, 이더리움에서 이루어진 트랜잭션들을 VITE에서 요구하는 트랜잭션의 단일 자유도 원칙에 따라 나누는 방법에 대해 정의할 것이다:

정의 2.2 (트랜잭션 해체) *1보다 더 큰 자유도 값을 가진 트랜잭션을 일련의 트랜잭션 단일 자유도들로 나누는 것을 트랜잭션 해체라고 부른다. 이전 트랜잭션(transfer transaction)는 송신 트랜잭션(sending transaction)과 수신 트랜잭션(receiving transaction)로 나뉘어질 수 있으며; 계약 콜 트랜잭션은(contract call transaction)은 계약 요청 트랜잭션(contract request transaction)와 계약 응답 트랜잭션(contract response transaction)로 나뉘어 질 수 있으며; 각 계약 내에서 메시지 콜은 계약 요청 트랜잭션과와 계약 응답 트랜잭션로 나뉘어질 수 있다.*

따라서 원장에는 두 개의 다른 유형의 트랜잭션들이 있게 될 것이다. 이들은 "트레이딩 페어(trading pairs)"라고 불린다:

정의 2.3 (트레이딩 페어) 송신 트랜잭션 즉 계약 요청 트랜잭션은 합쳐서 "요청 트랜잭션(request transaction)"로 불리며; 수신 트랜잭션 즉 계약 응답 트랜잭션은 합쳐서 "응답 트랜잭션(response transaction)"로 불린다. 요청 트랜잭션과와 그에 상응하는 응답 트랜잭션은 트랜잭션 쌍(transaction pairs)라고 불린다. 트랜잭션 t 요청 시작용 계정은 A(t)로 기록되며; 이에 상응하는 응답 트랜잭션은: t로 기록되는데, 트랜잭션에 상응하는 계정은 A(~t)로 불린다.

위의 정의에 기반해, VITE에서 두 개의 트랜잭션들 사이에 발생할 수 있는 관계라고 결론 내릴 수 있다:

정의 2.4 (트랜잭션 관례) *트랜잭션 t1과 t2 경우 다음 관계들이 존재할 수 있다:*
직교: A(t1)≠A(t2)인 경우, 이 두 개의 트랜잭션은 직교하며, 1 ⊥ t2로 기록되며;
평행: A(t1)=A(t2)인 경우, 이 두 개의 트랜잭션은 평행을 이루며, t1 ∥ t2로 기록되며;
인과 관계: t2=t1인 경우, 이 두 개의 트랜잭션은 인과 관계로, t1 ▷t2, or t2 ◁t1로 기록된다.

2.3 원장 정의

원장을 정의하는 것은 한 부분 순서 집합(poset)을 정의하는 것이다. 우선 VITE에서 두 트랜잭션의 부분 순서 관계를 정의해 보자.

정의 2.5 (트랜잭션 부분 순서) *우리는 두 개의 트랜잭션의 부분 순서를 나타내기 위해 이원 관계(dualistic relationship) <를 이용한다:*

응답 트랜잭션은 이에 상응하는 요청 트랜잭션에 뒤이어 나와야 한다: t1 < t2 ⇔ t1 ▷t2;

한 계정에 있는 모든 트랜잭션은 엄격하게 세계적으로 순서가 정해져야 한다: ∀t1 ∥ t2 때, t1 < t2, 또는 t2 < t1가 있어야 한다.

트랜잭션 세트 T에 수립된 부분 순서 관계 때문에 다음 특징들에 부합해야 한다:

- 비-반사적(irreflexive): ∀t ∈ T, no t < t가 있어야 하며;
- 과도기적(transitive): ∀t1, t2, t3 ∈ T, t1 < t2, t2 < t3이면, t1 < t3이 있어야 하며;
- 비-동기적(asymmetric): ∀t1,t2 ∈ T, t1 < t2이면, t2 < t1은 존재하지 않는다.

이런 식으로, 우리는 엄격한 부분 순서 세트에서 VITE 계정을 정의할 수 있다:

정의 2.6 (VITE 원장) *VITE 원장은 주어진 트랜잭션의 T 세트에 의해 구성된 엄격한 부분 순서 집합으로서 이 부분 순서 집합은 < 다.*

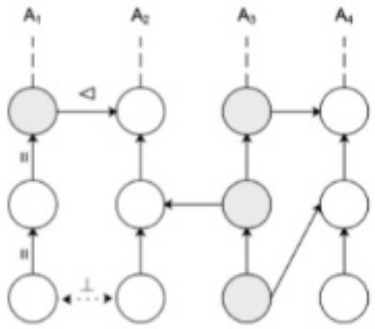

그림 5: VITE에서 원장과 트랜잭션의 관계

엄격한 부분 순서 집합은 DAG 구조에 상응할 수 있다. 위에 나온 그림에서 보듯이, 원들은 트랜잭션들을 나타내며 화살표들은 두 트랜잭션의 의존도를 표시한다. a → b는 a가 b에 의존한다는 것을 시사한다.

위에 정의된 VITE 원장은 블록-격자와 비슷하다. 트랜잭션들은 요청 트랜잭션과와 응답 트랜잭션로 나뉘어지고, 각 트랜잭션은 각 블록에 상응하며, 각 계정 Ai는 한 체인, 한 트랜잭션 쌍(translation pair), 그리고 응답 트랜잭션에 상응하는 요청 트랜잭션의 해시를 참조하는(referencing) 응답 트랜잭션에 상응한다.

3 스냅숏 체인

3.1 트랜잭션 확정(transaction confirmation)

계정이 분기되면, 합의 결과는 두 분기된 원장들 사이에서 왔다 갔다 할 지 모른다. 예를 들면, 한 블록 체인을 기반로으로, 한 노드가 길이 더 긴 분기된 체인을 받을 경우, 이 새로운 분기는 합의 결과로 선택될 수 있으며, 원래 분기는 포기될 것이며 원래 분기에서 이루어지는 트랜잭션은 철회될 것이다(roll back). 이러한 시스템에서 트랜잭션 철회(transaction rollback)는 아주 심각한 사건으로, 이중 소비로 이어질 것이기 때문이다. 한 사업체가 대금을 받고, 재화나 서비스를 제공하는데, 그 대금을 인출한 후 상인은 손해를 입을 수 있다는 점을 생각해보라. 따라서 한 사용자가 트랜잭션 대금을 받으면, 트랜잭션 철회(rolling back) 확률이 충분히 낮다는 것을 보장할 수 있도록 시스템이 트랜잭션을 "확정"할 때까지 기다릴 필요가 있다.

정의 3.1 (트랜잭션 승인) *한 트랜잭션의 철회(being rolled back) 확률이 제시된 역치 ϵ보다 낮을*

때, 그 트랜잭션은 확정되었다고 부른다. ***Pr(t) < ϵ ⇔ t가 확정됨.***

트랜잭션 확정은 아주 헷갈리는 개념인데, 사실 한 트랜잭션이 인정되는지 여부는 1−ϵ라는 암묵적 확신에 좌우되기 때문이다. 한 다이아몬드 판매상과 한 커피 판매상이 이중 소비 공격을 받았을 때 이들은 규모가 다른 손해로 고통받았다. 그 결과, 전자는 트랜잭션 때 더 작은 트랜잭션 역치 ϵ를 설정할 필요가 있다. 이것은 비트 코인에서 확정 수(number of confirmations)의 본질이기도 하다. 비트 코인에서 확정 수(confirmation number)는 블록 체인으로 하는 트랜잭션의 깊이를 암시한다. 확정 수가 클수록, 트랜잭션이 철회될 확률은 더 낮아진다. 따라서 상인들은 확정 수의 대기 수(waiting number)를 설정해서 확정의 확신 수준(confidence level)을 간접적으로 설정할 수 있다.

트랜잭션 철회 확률은 계정 구조에 있는 해시 참조(hash reference) 관계 때문에 시간이 지나면서 줄어든다. 위에 언급한 것처럼 원장 설계가 더 나은 부당 변경(tampering) 특징들을 가질 경우, 트랜잭션 철회는 그 블록에서 그 이후 이루어지는 모든 후속 교환 블록들을 재구성할 필요가 있다. 새로운 트랜잭션들은 일정하게 원장에 추가되기 때문에, 한 트랜잭션에는 점점 더 많은 후속 노드들(successive nodes)이 있게 되고, 그러면 부당 변경될(being tampered) 확률은 줄어들게 될 것이다.

블록-격자 구조에서, 트랜잭션은 계정 별로 각 그룹으로 분류되기 때문에, 트랜잭션은 자체 계정의 계정 체인의 끝에만 부착될 것이며, 그러면 나머지 대부분 계정들에 의해 발생된 트랜잭션은 자동으로 이 트랜잭션의 후속 노드가 되지 않을 것이다. 따라서 이중 소비라는 숨겨진 위험을 피하기 위해서 합리적으로 합의 알고리즘을 설계하는 것이 반드시 필요하다.

나노는 투표(voting)에 기반한 합의 알고리즘을 채택하며, 트랜잭션은 한 사용자 그룹이 선택한 일련의 대표 노드들에 의해 서명된다. 각 대표 노드는 하나의 무게(weight)를 가진다. 한 트랜잭션의 서명이 충분한 무게를 가질 때, 그 트랜잭션은 확정되었다고 믿는다. 이 알고리즘에서 발생되는 문제들로는 다음과 같은 것들이 있다:

첫째, 확정에 대해 더 높은 수준의 확신이 필요한 경우, 투표 무게의 역치도 올라갈 필요가 있다. 온라인에 대표 노드들이 충분하지 않다면, 교차 속도(intersecting speed)를 보장할 수 없으며, 따라서 사용자는 한 교환을 확정하는데 반드시 필요한 수의 티켓들을 결코 모으지 못할 가능성이 있으며;

둘째, 트랜잭션이 철회될 확률은 시간이 지나면서 줄어들지 않게 된다. 이는 언제나 역사적 투표를 뒤집는 비용은 동일하기 때문에 생기게 된다.

마지막으로 역사적 투표 결과는 지속되지 못해 원장에 기입되지 못하고, 노드들의 지역 스토리지(local storage)에만 저장될 뿐이다. 한 노드가 자신의 노드를 다른 노드들로부터 얻을 경우, 한 역사적 트랜잭션의 철회 확률을 확실하게 정량화 할 방법이 전혀 없다. 본질적으로, 투표 메커니즘은 부분적 중앙 집중화 해결책이다. 우리는 투표 결과를 원장 지위를 보여주는 한 스냅숏으로 간주할 수 있다. 이 스냅숏은 네트워크에 있는 각 노드의 지역 스토리지(local storage)에서 배포될 것이다. 블록 체인과 동일한 부당 변경 방지(tamper proof) 능력을 갖기 위해, 우리는 도한 이 스냅숏들을 체인 구조들로 조직할 수 있는데, 이것이 VITE 설계의 핵심 특징들 중 하나인 스냅숏 체인이다.

3.2 스냅숏 체인 정의

스냅숏 체인은 VITE에서 가장 중요한 스토리지(storage) 구조다. 이것의 주된 기능은 VITE 원장들의 합의를 유지하는 것이다. 우선 우리는 스냅숏 체인 정의를 제시할 것이다:

정의 3.2 (스냅숏 블록과 스냅숏 체인). *스냅숏 블록은 계정의 잔액, 계약 상태의 머클(Merkle) 뿌리 및 각 계정 체인의 마지막 블록 해시를 포함해 한 VITE 원장의 상태 스냅숏을 저장한다. 스냅숏 체인은 스냅숏 블록들로 구성된 체인 구조이며, 다음 스냅숏 블록이란 이전 스냅숏 블록의 해시를 지칭한다.*

사용자 계정 상태에는 잔액과 이에 상응하는 계정 체인의 마지막 블록의 해시가 포함되어 있으며; 위의 두 개의 필드에 더해, 한 계약 계정 상태에는 이 계정의 머클 뿌리 해시가 포함되어 있다. 한 계정 상태의 구조는 다음과 같다:

```
구조 계정 상태{
//계정 잔액
맵<uint32, uint256> 잔액;
//계약 상태의 머클 뿌리 최적 uint256 스토리지 루트;
//마지막 계약의 해시
//계정 체인 uint256 마지막 트랜잭션 lastTransaction;
}
```

스냅 블록 구조는 다음과 같이 정의된다:

```
스냅숏 블록 구조 {
//이전 블록의 해시 uint256 prevHash;
//스냅숏 정보 맵<주소, 계정상태> 스냅숏;
//서명 uint256 서명;
}
```

동시에 여러 토큰들을 지원하기 위해, VITE 계정 상태에서 잔액 정보 기록 구조는 uint256이 아니라, 토큰 ID부터 잔액까지의 맵핑(mapping)이다.

스냅숏 체인에 있는 첫 번째 스냅숏 블록은 "최초 스냅숏(genesis snapshot)"이라고 불리는데, 이것은 이 계정에 있는 최초 블록(genesis block)의 스냅샷을 저장한다.

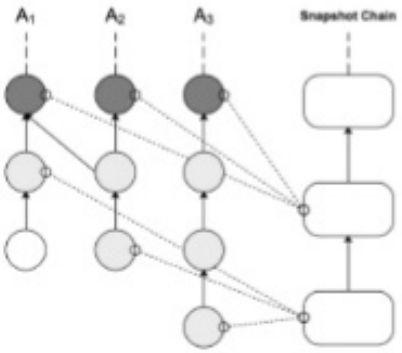

그림 6: 스냅숏 체인

스냅숏 체인에 있는 각 스냅숏 블록은 VITE 원장의 유일한 분기에 상응하기 때문에, 스냅숏 블록이 스냅숏 블록에서 분기하지 않을 때 상응하는 스냅숏 블록에 의해 VITE 원장의 합의 결과를 판단하는 것이 가능하다.

3.3 스냅숏 체인과 트랜잭션 확정

스냅숏 체인을 도입한 후, 블록-격자 구조가 원래 갖고 있는 보안상 약점들이 해결되었다. 한 공격자가 이중 소비 트랜잭션을 발생시키길 원할 경우, VITE 원장에 있는 해시 참조(hash reference)를 재건하는 것뿐만 아니라 트랜잭션의 첫 번째 스냅숏 블록 이후 모든 블록을 위해 스냅숏 체인을 재건할 필요가 있으며, 또한 더 긴(longer) 스냅숏 체인을 생성할 필요가 있다. 이런 식으로 하면 공격의 비용은 크게 증가할 것이다.

VITE에서, 트랜잭션 확정 메커니즘은 비트 코인과 유사한데, 이것은 다음과 같이 정의된다:

정의 3.3 (VITE에서 트랜잭션 확정) *VITE에서 한 트랜잭션이 스냅숏 체인에 의해 스냅숏 되면 트랜잭션은 확정되고 첫 번째 스냅숏에 있는 스냅숏 블록은 트랜잭션의 확정 수(confirmation number)라고 불리게 된다.*

이 정의에 의거하면, 스냅숏 체인이 성장할 때 확정된 트랜잭션 수는 1까지 증가할 것이며, 스냅숏 체인이 증가함에 따라 이중 소비 공격 확률은 떨어지게 될 것이다. 이런 식으로 하면, 사용자들은 특정 시나리오에 다라 다양한 확정 수들을 기다리는 것으로, 트랜잭션에 요구되는 확정 수를 맞춤 설정할 수 있다.

스냅숏 체인 자체는 합의 알고리즘에 의존한다. 한 스냅숏 체인이 분기되면, 가장 긴 분기가 유효 분기(valid fork)로 선택된다. 스냅숏 체인이 한 새로운 분기로 바뀌게 되면, 원래의 스냅숏에 있는 정보는 철회될 것인데(roll back), 이 말은 원장에서 원래의 합의는 번복되고 새로운 합의로 대체된다는 것을 의미한다. 따라서 스냅숏 체인은 전체 시스템의 보안의 초석이 되어 진지하게 처리될 필요가 있다.

3.4 압축 스토리지(compressed storage)

모든 계정 상태들은 스냅샷 체인에 있는 모든 스냅샷 블록에서 저장될 필요가 있고, 저장 공간은 아주 커야 하기 때문에, 스냅샷 체인을 압축하는 것이 반드시 필요하다.

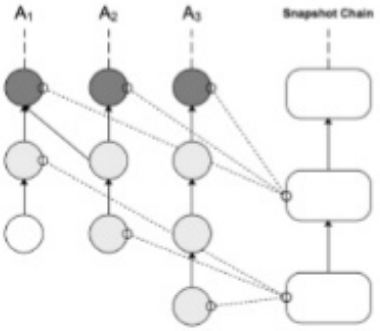

그림 7: 압축 전 스냅숏

스냅숏 체인 스토리지 공간 압축에 대한 기본 접근은 점점 늘어나는 스토리지(incremental storage)를 이용하는 것인데, 한 스냅숏 블록은 이전 스냅숏 블록과 비교해 보면 변경된 데이터만 저장하기 때문이다. 두 스냅숏 사이에서 한 개의 계정을 위한 트랜잭션이 없으면, 후자 스냅숏 블록은 그 계정의 데이터를 저장하지 않을 것이다.

스냅숏 정보를 복원하기 위해, 시작부터 끝까지 스냅숏 블록을 횡단할 수(traverse) 있으며, 또한 현재 데이터로 모든 스냅숏 블록의 데이터를 덮을 수 있다.

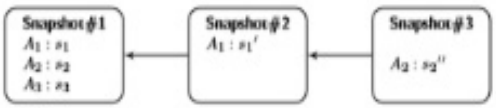

그림 8: 압축 후 스냅숏

스냅숏을 할 때 한 계정의 각 스냅숏의 최종 상태만 저장되며, 중간 상태는 고려되지 않기 때문에, 따라서 두 개의 스냅숏들 사이에 한 계정에 의해 얼마나 많은 트랜잭션들이 생성되든 그렇지 않든 이와 관계없이 스냅숏에 있는 데이터의 한 사본이 저장될 것이다. 따라서 스냅숏 블록은 최대 S*A까지 차지하게 될 것이다. 이들 중에서, S=(si) 크기는 각 계정 상태가 차지하는 바이트 수이며, A는 전체 시스템 계정들의 수다. 총 계정 대비 활성 계수(active accounts)의 평균 비율이 a라면, 압축률은 1−a다.

4 합의

4.1 설계 목표

한 합의 프로토콜을 설계할 때, 우리는 다음 요인들을 충분히 고려할 필요가 있다:

• 성능. VITE의 주요 목표는 빠른 속도다. 높은 처리량(throughout)과 낮은 지연 성능을 보장하기 위해, 우리는 더 높은 수렴 속도(convergence speed)를 가진 합의 알고리즘을 채택할 필요가 있다.
• 확장성(scalability). VITE는 모든 분산 애플리케이션에 개방적인 공공 플랫폼(public platform)이기 때문에 따라서 확장성 역시 중요한 고려 사항이다.
• 보안성. VITE 설계 원칙은 절대적 안전을 추구하는 것은 아니나, 그렇지만, 충분한 안전 기준선을 보장하고 또한 모든 종류의 공격들을 효과적으로 막는 것이 필요하다.

일부 기존 합의 알고리즘과 비교해, PoW의 보안은 더 좋으며, 악의적 노드들의 처리 능력이 50% 미만인 경우, 합의에 도달할 수 있다. 그러나 PoW의 교차 속도는 느리고 성능 요구 사항들에 부합할 수 없으며; PoS와 이것의 변종 알고리즘은 수학적 문제들을 해결하는 단계들을 없애, 교차 속도와 단일 공격 비용을 개선시키고, 에너지 소비를 줄인다. 그러나 PoS의 확장성은 여전히 부실하며, 또한 "Nothing at Stake" 문제는 해결하기 어려우며; BFT 알고리즘이 보안과 성능에서 더 나은 성과를 보이지만, 그러나 BFT의 확장성이 문제로, 이것은 대개 프라이빗 체인이나 컨소시엄 체인에 더 적합하며; DPoS 시리즈 알고리즘은 블록 생성 허가를 제한함으로써 거짓 분기 확률을 효과적으로 줄이지만 성능과 확장성은 좋은 편이다. 결과적으로 DPoS는 보안에서 약간의 희생이 있으며, 따라서 악의적 노드 수는 1/3 이상이 되어서는 안 된다.

일반적으로 DPoS 알고리즘은 성능과 확장성에서 명백한 이점을 갖고 있다. 따라서 우리는 VITE 합의 프로토콜의 토대로 DPoS를 선택하고 이 프로토콜을 토대로 DPoS를 적절하게 확장한다. 서열 지정 합의 프로토콜(Hierarchical Delegated consensus protocol)과 비동기 모델을 통해, 플랫폼의 전반적 성능은 더욱 개선될 수 있다.

4.2 서열적 합의(hierarchical consensus)

VITE의 합의 프로토콜은 HDPoS(서열 지정 지분 증명, Hierarchical Delegated Proof of Stake)다. 기본 아이디어는 합의 기능 Φ을 해체하는 것(기능 해체)이다:

$$\Phi(l1,l2,...,ln) = \Psi(\Lambda1(l1,l2,...,ln), \Lambda2(l1,l2,...,ln),... \Lambda m(l1,l2,...,ln)) \quad (5)$$

$\Lambda i : 2L \rightarrow L$은 지역적 합의 기능으로 불리며, 반환된 결과는 지역적 합의(local consensus)라고 불리며; $\Psi : 2L \rightarrow$ 전체 합의 기능(global consensus function)으로 알려진 L은 최종 합의 결과로서 지역적 합의(local consensus)에서 한 후보 그룹으로부터 나온 한 독특한 결과를 선택한다.

9쪽

이러한 분리 후, 전체 시스템의 합의는 두 개의 독립 과정들이 되었다:
지역적 합의(local consensus)는 사용자 계정 또는 계약 계정에서 요청 트랜잭션과와 응답 트랜잭션에 상응하는 블록들을 생성하고 원장에 이들을 기록한다. 전체 합의(global consensus)는 원장에서 데이터를 스냅숏 찍고, 스냅숏 블록들을 생성한다. 원장이 분기되면 이들 스냅숏들 하나를 선택한다.

4.3 블록 생성 권리와 합의 그룹

그럼 원장에서 트랜잭션 블록을 생성하고 스냅숏 체인에서 스냅숏 블록을 생성하는 권리는 누가 갖고 있는가? 합의에 도달하기 위해 어떤 합의 알고리즘이 채택되는가? VITE의 원장 구조는 다양한 계정에 따라 다중 계정 체인들로 조직되기 때문에, 우리는 계정 차원에 따라 원장에서 블록을 생산하는 권리와 한 단일 사용자 그룹에 속한 스냅숏 블록을 생산하는 권리를 편리하게 정의할 수 있다. 이런 식으로 하면 우리는 많은 계정 체인들이나 스냅숏 체인들을 한 합의 그룹에 포함시킬 수 있으며, 이 합의 그룹에서 우리는 블록을 생산하기 위해 한 통일된 방식을 사용하고 합의에 도달할 수 있다.

정의 4.1 (합의 그룹) *합의 그룹은 계정 또는 스냅숏 체인의 한 부부분에서 이루어지는 합의 메커니즘을 기술하는, 하나의 튜플(L,U,Φ,P)이며, L ∈ A|{As}은 원장에 있는 합의 그룹의 하나 또는 다수의 계정 체인들이나 스냅숏 체인들을 나타내며; U는 L에 의해 규정된 체인에서 블록 생산 권리를 가진 사용자를 나타내며; Φ은 합의 그룹의 합의 알고리즘을 규정하며; P는 합의 알고리즘의 매개변수들을 규정한다.*

이 정의에 의거하면, 사용자들은 융통성 있게 합의 그룹을 설정하고 그들의 필요에 따라 다양한 합의 매개 변수들을 선택할 수 있다. 이어 다양한 합의 그룹들에 대한 자세한 설명이 있을 것이다.

4.3.1 스냅숏의 합의 그룹

스냅숏 체인의 합의 그룹은 스냅숏 합의 그룹이라 불리는데, 이 그룹은 VITE에서 가장 중요한 합의 그룹이다. 스냅숏 체인의 합의 알고리즘 Φ은 DPoS 알고리즘과 서열 모델(hierarchical model)에서는 이에 상응하는 Ψ를 채택한다. 블록 생성의 에이전트와 간격(interval) 수는 매개변수 P에 의해 규정된다.

예컨대, 우리는 1초 간격으로 스냅숏 블록들을 생산하기 위해 25개의 프록시 노드를 가진 스냅숏 합의 그룹들을 규정할 수 있다. 이것은 트랜잭션이 충분히 빠르게 확정된다는 것을 보장한다. 10배 빠른 속도로 트랜잭션 확정에 도달하기 위해선 최대 10초까지 기다릴 필요가 있다.

4.3.2 프라이빗 합의 그룹(Private Consensus Group)

프라이빗 합의 그룹은 원장에서 이루어지는 트랜잭션 블록 생산에만 적용할 수 있으며 또한 프라이빗 합의 그룹의 계정 체인에 속한다. 이 블록들은 계정의 프라이빗 키 소유자에 의해서만 생산될 수 있다. 디폴트에 의해 모든 사용자 계정은 프라이빗 합의 그룹에 속하게 된다.

프라이빗 합의 그룹의 최대 장점은 분기 확률을 줄인다는 것이다. 블록을 생산할 수 있는 권리는 오직 한 사용자만 갖기 때문에, 분기 발생은 사용자가 개인적으로 이중 소비 공격이나 프로그램 오류를 시작할 때만 가능하다.

프라이빗 합의 그룹의 단점은 사용자 노드들이 트랜잭션을 하나로 통합하기(pack) 전에 그것들은 온라인 상에 있어야 한다는 점이다. 이러한 특징은 계약 계정에 아주 적합한 것이 아니다. 소유주의 노드가 실패하면, 어떠한 다른 노드도 계약들을 생산하는 응답 트랜잭션을 대체할 수

없는데, 이는 dApp 서비스 이용 가능성을 줄이는 것과 맞먹는 큰 단점이다.

4.3.3 지정 합의 그룹(Delegate Consensus Group)

지정 합의 그룹에선, DPoS 알고리즘을 통해 트랜잭션을 일괄 처리하는데(package) 사용자 계정 대신 일련의 지정된 프록시 노드들이 이용된다. 사용자 계정과 계약 계정 모두 이 합의 그룹에 추가될 수 있다. 사용자들은 일련의 개별 에이전트 노드들을 설정하고 한 새로운 합의 그룹을 세울 수 있다. 또한 개별적으로 그들의 지정 합의 그룹을 설립하지 않은 모든 나머지 계정들이 트랜잭션을 일괄 처리하는 것을 도울 수 있도록 VITE에는 디폴트 합의 그룹이 있는데, 이것은 **공공 합의 그룹(public consensus group)**으로 알려져 있기도 하다.

지정 합의 그룹은 대부분 계약 계정들에 적합한데, 계약 계정에서 이루어지는 트랜잭션 대부분이 계약 응답 트랜잭션들로, 이런 종류의 트랜잭션에선 사용자 계정에서 수신 가능 트랜잭션들보다 더 높은 이용 가능성과 더 낮은 지연이 필요하다.

4.4 합의의 우선 순위

VITE 프로토콜에서, 세계적 합의(global consensus)의 우선 순위는 지역적 합의(local consensus)보다 더 높다. 지역적 합의가 분기될 경우, 세계적 합의 선택 결과들이 우선적으로 채택될 것이다. 다시 말해서 세계적 합의가 최종 결과로 지역적 합의의 분기를 선택하면, 향후 계정들에서 한 특정 계정 체인의 더 긴 분기가 발생할 지라도, 그것은 세계적 합의(global consensus) 결과들의 철회(roll back)를 초래하지 못할 것이다.

이 문제 때문에 크로스 체인 프로토콜을 실행할 때 더 많은 주의를 기울여야 한다. 한 목표 대상 체인이 철회(roll back)할 수 있기 때문에, 이 체인을 매핑하는 릴레이 계약에 상응하는 계정 체인 역시 그에 따라 철회(roll back)할 필요가 있기 때문이다. 이 시점에서, 세계적 합의에 의해 릴레이 체인의 지역적 합의가 채택되었다면, 철회(rollback)을 완료하는 것은 불가능하게 되는데, 이것은 릴레이 계약과 목표 대상 체인 사이의 데이터를 일관성 없게 만들 수 있다.

10쪽

이 문제를 피할 수 있는 방법은 합의 그룹 매개 변수인 P에서 한 매개 변수 지연을 설정하는 것으로, 이것은 스냅숏 합의 그룹에 지연 블록들 이후 지역적 합의가 완료된 후에만 한 스냅숏을 찍도록 규정한다. 이렇게 하면 릴레이 계약의 비-일관성 가능성이 크게 줄겠지만, 이 방법으로 세계적 합의 결과들의 철회(rollback) 문제를 완전히 피할 순 없다. 릴레이 계약의 코드 논리에선, 또한 목표 대상 체인의 철회(rollback) 문제를 각각 다루는 것이 반드시 필요하다.

4.5 비-동기적 모델

시스템 처리량(throughput)을 더욱 개선시키기 위해, 합의 메커니즘에서 더욱 완벽한 비-동기 모델을 지원할 필요가 있다.

한 트랜잭션의 수명 주기(life cycle)에는 트랜잭션 시작(transaction initiation), 트랜잭션 기록(transaction writing) 및 트랜잭션 확정(transaction confirmation)이 포함된다. 시스템의 성능을 개선하기 위해 우리는 이 3가지 단계들을 비-동기 모드로 설계할 필요가 있다. 이렇게 하는 이유는 때마다 사용자들이 시작한 트랜잭션량이 다르며, 시스템에 의해 처리된 트랜잭션 기록 및 트랜잭

션 확정 속도가 비교적 고정되기 때문이다. 비-동기 모드는 정점(peaks)과 통과점(throughs)를 평평하게(flatten) 만드는데 도움을 주며, 그에 따라 시스템 전반의 처리량(throughput)을 개선시킨다.

비트 코인과 이더리움의 비-동기적 모델은 간단하다: 모든 사용자에 의해 시작된 트랜잭션은 비-확정 풀(pool) 안에 위치한다. 채굴꾼이가 이 비-확정 풀을 하나의 블록으로 일괄 처리(package)할 때, 트랜잭션 기록과 확정이 동시에 이루어진다. 블록 체인이 계속 성장할 경우, 트랜잭션은 결국 사전에 정한 확정 확신 수준에 도달하게 된다.

이 비-동기 모델에는 다음 두 가지 문제가 있다:

• 거레들은 한 비-확정 상태로 지속되며 원장까지 진입하지 못한다. 인정받지 못한 트랜잭션들은 불안정한 상태이며, 또한 관여된 합의가 없기 때문에, 트랜잭션들을 반복적으로 송신하는 것을 막을 수 없다.

• 트랜잭션들을 기록하고 확정하는데 이용되는 비-동기 메커니즘이 없다. 트랜잭션들은 확정된 경우에만 기록되고, 또한 기록 속도는 확정 속도에 의해 제한된다.

VITE 프로토콜은 더욱 개선된 비-동기 모델을 만들었다: 첫째, 트랜잭션이 한 이전(transfer) 트랜잭션이든 아니면 계약 호출(contract call)이든 관계없이 한 "요청-응답" 모델에 기반한 트랜잭션 쌍(transaction pair)으로 나뉘어지며, 한 요청 트랜잭션이 원장에 기록될 때 이 트랜잭션은 성공적으로 시작된다. 게다가, 한 트랜잭션의 기록 및 확정 역시 비-동기적이다. 트랜잭션은 처음에 VITE의 DAG 계정에 기록될 수 있으며 확정 과정에 의해 차단되지 않을 것이다. 트랜잭션 확정은 스냅숏 체인을 통과하며 이루어지게 되며, 따라서 스냅숏 작용(action) 역시 비-동기적이다.

이것이 전형적인 생산자-소비자 모델이다. 트랜잭션의 수명 주기에서, 생산 속도가 상류(upstream)에서 어떻게 변하든 관계없이, 플랫폼을 자원을 철저히 활용하고 시스템의 처리량을 개선하기 위해 하류에서는 일정한 속도로 트랜잭션을 다룰 수 있다.

5 가상 기계

5.1 EVM 호환성

현재 이더리움 필드에는 많은 개발자들이 있으며, 또한 많은 스마트 계약들이 솔리디티와 EVM을 기반로으로 적용된다. 따라서 우리는 VITE 가상 미계에서 EVM 호환성(compatibility)을 제공하기로 결정했으며, 따라서 대부분 EVM 명령 세트에 있는 원래 의미(original semantics)는 VITE에서 그대로 유지된다. VITE의 계정 구조와 트랜잭션 정의는 이더리움의 것과 다르기 때문에, 일부 EVM 명령들의 의미는 재-정의될 필요가 있는데, 예를 들면 블록 정보를 얻기 위한 일련의 명령들의 의미는 재-정의될 필요가 있다. 자세한 의미상의 차이에 대한 정보는 부록 A를 참고할 것.

이들 중 가장 큰 차이는 메시지 콜의 의미다. 이어서 이에 대해 자세히 논의할 것이다.

5.2 사건 기반(event driven)

이더리움의 프로토콜에서, 트랜잭션 또는 메시지는 여러 계정들의 지위에 영향을 미칠 수 있다. 예컨대 계약 호출 트랜잭션(invocation transaction)은 메시지 콜들을 통해 동시에 여러 계약 계정들의 지위 변경을 초래할 수 있다. 이러한 지위 변경은 동시에 일어나거나 전혀 일어나지 않는다. 따라서 이더리움에서 트랜잭션은 실질적으로 ACID (원자성, 일관성, 분리, 지속성)의 특징들을 만족시키는 일종의 엄격한 트랜젝션으로, 이것은 또한 이더리움에서 확충성(expansibility) 결여의 중요한 이유이기도 하다.

확장성(scalability)과 성능을 고려해 VITE는 BASE(기본적으로 이용 가능, 소프트 상태, 최종적 일관성) 의미를 만족시키는 최종 일관성 도식(final consistency scheme)을 채택했다. 구체적으로 말해 우리는 사건-기반 아키텍처(EDA)로 VITE를 설계했다. 각 스마트 계약은 독립적인 서비스로 간주되고, 또한 계약들 사이에서 메시지를 주고받을 수 있지만 어떤 상태도 공유하지는 않는다. 따라서 VITE의 EVM에서 우리는 계약들을 아우리는 동기적 기능 호출들(synchronous function calls)의 의미를 철회하고, 또한 두 계약 간에 메시지 소통만 허용할 필요가 있다. 이로부터 영향 받는 EVM 명령들은 주로 CALL과 STATICCALL이다. VITE의 EVM에서 이 두 개의 명령들은 즉시 실행될 수 없으며 또한 이들은 호출의 결과를 반환할 수도 없다. 이 두 명령들은 원장에 기록할 수 있는 요청 트랜잭션만 생성한다.

따라서 VITE에서, 기능 호출들의 의미는 이 명령에 포함되는 것이 아니라, 대신 한 계정으로 메시지들을 보낼 것이다.

5.3 스마트 계약 언어

이더리움은 스마트 계약을 개발하도록 튜링 컴플릿 프로그래밍 언어인 솔리디티(Solidity)를 제공한다. 비-동기적 의미를 지원하기 위해, 우리는 솔리디티(Solidity)를 확장해, 메시지 소통을 위한 일련의 구문들(syntax)을 정의했다. 이 확장된 솔리디티는 Solidity++라고 불린다.

11쪽

솔리디티의 구문들(syntax)의 대부분은 Solidity++로부터 지원받지만, 여기에는 계약 외부 기능 호출들은 포함하지 않는다. 계약 개발자는 키워드 메시지들을 통해 메시지를 정의하고 또한 크로스 계약 소통 기능을 실행하기 위해 키워드를 통해 메시지 처리기(MessageHandler)를 정의할 수 있다.

예를 들면 계약 A는 반환 가치(return value)에 기반해 계약 A의 상태를 갱신하기 위해 계약 B에서 추가된 ()방법을 호출할 필요가 있다. 이 코드는 다음과 같다:

```
프로그램 솔리디티 ~0.4.0;
계약 B {
Add 기능(uint a, uint b) 반환 가치
(uint ret) {
반환 가치 a + b;
}
}
계약 A {
uint 총합;
기능 호출자(주소 addr, uint a, uint b) {
//A.add( )로 메시지 호출
uint sum = B(addr).add(a, b);
//(합계 > 10)이면 반환 가치 이용 {총합 +=합계;
    }
    }
```

}

Solidity++에서, 기능 호출 코드 uint 합계=B(addr).add(a, b)이며; 더 이상 유효하지 않으며; 이것 대신 계약 A와 B는 서로에게 메시지를 보내 비-동기적으로 소통한다. 이 코드는 다음과 같다:

```
프로그램 솔리디티++~0.10;
계약 B {Add 메시지(uint a, uint b); 메시지 합계(uint 합계);
Add.on {
//메시지 uint a = msg.data.a 읽기;
uint b = msg.data.b;
주소 송신인= msg.sender;
//할 일 uint 합계= a + b;
//메시지를 반환 결과 가치로 보내기 (송신인, 합계(sum)) 보내기;
}
계약 A {
uint 총 가치;
기능 호출자 (주소 addr, uint a, uint b) {
//B에게 메시지 호출 (addr, Add(a, b)) 보내기
//메시지 호출 보낸 후 무엇이든 할 수 있음
//반환 가치를을 이용하는 대신
//다른 메시지
}
합계.on {
//메시지로부터 반환 가치 데이터 얻기
uint 합계 = msg.data.sum;
//반환 가치 데이터 이용하기
(합계 > 10)이면 {
총 가치 += 합계;
}
}
}
```

위 공식들의 첫 줄에 나온 코드 프로그램 솔리디티++ ~0.1.0은 소스 코드가 솔리디티++에 기록되었다는 것을 시사하지만 그러나 컴파일된(명령어로 번역된) EVM 코드는 기대된 의미를 따르지 않는 것을 피하기 위해 솔리디티 컴파일러를 이용해 직접 컴파일 되지(명령어로 번역되지) 않을 것이다. VITE는 솔리디티++의 컴파일 작업을 위해 특수 컴파일러를 제공할 것이다. 이 컴파일러는 부분적으로 상위 호환할 수 있으며; VITE 의미와 상충하는 솔리디티 코드가 전혀 없는 경우엔 이 VITE 의미는 직접 컴파일 될 수 있으며, 그렇지 않은 경우엔 에러가 보고될 것이다. 예를 들면, 지역 기능 호출들, 다른 계정들로 가는 이전 호출들(transfers)의 의미는 호환 가능한 것으로

남아 있을 것이며; 통화단위 뿐만 아니라 크로스 계약 기능 호출의 반환 가치 역시 컴파일 되지 않을 것이다.

계약 A에서, 호출자 기능(invoker function)을 호출할 경우, Add 메시지가 계약 B로 보내질 것인데, 이 메시지는 비-동기적인 것으로 따라서 결과는 즉시 반환되지 않을 것이다. 따라서 반환된 결과를 받고 상태를 갱신하기 위해선 키워드를 이용해 계약 A에서 메시지 처리기를 정의하는 것이 반드시 필요하다.

계약 B에서, 메시지 Add는 모니터링 된다. Sum 메시지는 결과를 반환하기 위해 메시지 Add의 송신자에게 보낸다.

Solidity++에서 메시지는 CALL 명령들로 컴파일 될 것이며 한 요청 트랜잭션이 원장에 추가될 것이다. VITE에서 원장들은 두 계약 간의 비-동기적 소통을 위한 메시지 미들웨어 역할을 하게 된다. 이것은 신뢰할 수 있는 메시지 스토리지를 보장하고 복제를 방지한다. 동일한 계약에 의해 한 계약으로 송신된 여러 메시지들은 FIFO(선입 선출법)을 보장할 수 있으며; 서로 다른 계약들에서 동일 계약으로 보낸 메시지들은 FIFO를 보장하지 않는다.

솔리디티에 있는 사건들(사건)과 솔리디티++에 있는 메시지는 동일한 개념이 아니라는 점에 유의해야 한다. 사건들은 EVM 로그를 통해 프론트로 간접적으로 송신된다.

12쪽

5.4 표준 라이브러리

이더리움에서 스마트 계약을 개발하는 개발자들은 솔리디티에서 표준 라이브러리가 결여된 것으로 인해 크게 고생을 하는 경우가 종종 있다. 예를 들면 루프링 프로토콜에서 루프 검증(loop verification)은 체인 밖에서 수행되어야 하는데 그렇게 해야 하는 중요한 이유들 중 하나는 Solidity에는 부동 소수점(floating-point) 연산 기능 특히 부동 소수점 수들 경우 n 제곱근에 대한 연산 기능은 전혀 제공되어 있지 않기 때문이다.

EVM에서, 사전 배치된(pre-deployed) 계약은 라이브러리 기능을 실현하기 위해 DELEGATECALL 명령어에 의해 호출될 수 있다. 이더리움은 또한 여러 개의(several) 사전 컴파일드 계약(Precompiled Contract)을 제공하는데, 이 계약이 소수의 해시 연산들(Hash operations)이다. 그러나 이러한 기능들은 너무 간단해소 복잡한 애플리케이션의 욕구들을 충족시킬 순 없다.

따라서 우리는 솔리디티++에서 문자열 처리(string processing), 부동 소수점 연산(floating point operations), 기본 수학 연산(basic mathematical operations), 컨테이너(containers), 정렬(sorting) 등 같은 일련의 표준 라이브러리들을 제공할 것이다.

성능 고려 사항들에 기반한, 이 표준 라이브러리들은 지역 확장(Native Extension) 방식으로 실행될 것이며, 이 연산들의 대부분은 VITE의 지역 코드에 내장될 것이며, 이 기능은 EVM 코드에서 DELEGATECALL 명령을 통해서만 호출된다.

이 표준 라이브러리는 필요할 때 확장할 수 있지만, 전체 시스템의 상태 기계 모델이 결정 모델이기 때문에, 이것은 난수(random numbers) 같은 기능은 제공할 수 없다. 이더리움과 마찬가지로 우리는 스냅숏 체인들의 해시를 통해 준 의사 난수(pseudo random numbers)로 시뮬레이션 할 수 있다.

5.5 가스

이더리움에는 가스와 관련해 두 가지 주요 기능이 있는데, 첫 번째는 EVM 코드 실행에 의해 소비되는 연산 자원(computing resources)와 저장 자원을 정량화 하는 기능이며, 두 번째는 EVM 코드가 중지된 것을 보장하는 기능이다. 계산 가능성 이론(computability theory)에 의하면, 튜링(Turing)기계에서 중지 문제(Halting Problem)은 계산할 수 없는 문제다. 이 말은, EVM 코드 분석에 의해 제한된 집행 후 한 스마트 계약은 정지될 수 있는지 여부를 판단하는 것이 불가능하다는 뜻이다.

따라서 EVM에서 가스 계산 문제는 VITE에서도 문제로 여전히 남게 된다. 그러나 VITE에는 가스 가격(Gas Price)이란 개념이 아예 없다. 사용자들은 수수료를 내는 것으로 교환을 위한 가스를 구입할 수 없고, 대신 연산 자원을 얻기 위해서 할당량에 기반한 모델을 통해 가스를 얻을 수 있다. 할당량 계산은 뒤이어 나오는 "경제적 모델" 장에서 자세히 논의할 것이다.

6 경제적 모델

6.1 고유 토큰(Native Token)

플랫폼 연산 자원과 저장 자원을 정량화 하고 노드들의 작동을 장려하기 위해, VITE는 고유 토큰인 VITE 토큰을 개발했다. 토큰의 기본 단위는 VITE(vite)이고, 가장 작은 단위는 아토브(attove)로서, 1 VITE(vite)=1018 아토브(attove)가 된다.

스냅숏 체인이 VITE 플랫폼의 보안과 성능의 핵심이다. 노드로 하여금 트랜잭션 검증에 참여하도록 자극하기 위해 VITE 프로토콜은 스냅숏 블록 생산을 위한 단조 보상(forging reward)를 설정한다. 단조 인센티브(forging incentive)는 VITE 토큰의 유동성을 증가시키고 VITE(vite) 보유자들의 혜택들을 희석시킬 것이다. 따라서 우리는 상승률(inflation)을 연간 3%로 제한할 것이다. 이와 반대로 사용자들이 새로운 토큰을 발행할 때, 계약들을 배치할 때(deploy), VNS 도메인 이름을 등록하고 자원 할당량을 얻을 때, 사용자들은 유동성을 줄이기 위해 VITE 토큰을 소비하거나 담보로 내놓을 필요가 있다. 이 두 요인들의 복합 작용(combined action)으로 VITE(vite)의 유동성은 한 건강한 수준으로 유지될 수 있으며 또한 동시에 시스템 자원의 분배를 최적하는데 기여한다.

6.2 자원 할당(resource allocation)

VITE(Vite)는 흔한 dApp 플랫폼이기 때문에 이러한 플랫폼에 배치된 스마트 계약들의 능력은 다양하며, 각기 다른 스마트 계약은 처리 효율과 지연에 대해 다른 요구 사항들을 가진다. 동일한 스마트 계약이라 할지라도 서로 다른 단계에서 성능 요구 사항들은 달라진다.

이더리움 설계에서 각 트랜잭션은 계정들을 기록할 수 있는 다른 트랜잭션들과 경쟁할 수 있기 위해서 각 트랜잭션은 시작할 때 한 가스 가격을 배정받을 필요가 있다. 이것은 전형적인 입찰 모델(bidding model)로서, 이 모델은 원칙적으로 공급과 수요의 균형을 효과적을 관리할 수 있다. 그러나 사용자는 현재 공급과 수요 상황을 정량화 하기 어려우며 또한 다른 경쟁자들의 가격을 예측할 수 없기 때문에, 따라서 시장 실패가 쉽게 일어난다. 더욱이 한 트랜잭션을 대상으로 각 입찰의 자원 경쟁이 이루어지고 있으며 또한 계정 차원에 따라 자원들의 합리적 할당에 대한 어떠한 합의도 없다.

6.2.1 할당량 계산

우리는 VITE(Vite)에서 할당량에 기반한 자원 할당 프로토콜을 채택했는데, 이 프로토콜은 사용자

들이 다음 3가지 방식으로 더 높은 자원 할당량을 얻는 것을 가능하게 해 준다:
• PoW는 트랜잭션이 시작되었을 때 계산되며;
• 계정에서 특정 VITE(vite) 금액을 담보로 내놓으며;
• 한 번에 작은 VITE 금액 파괴하기. 구체적 할당량은 다음 공식으로 계산할 수 있다:

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

이들 중 한 단일 계정 할당량의 상위 한계를 나타내는 Qm은 한 상수로서, 시스템의 총 처리량과 총 계정 수와 관련되어 있다. ξ = (ξd,ξs,ξf)는 한 자원을 획득하는데 드는 사용자 비용을 나타내는 벡터이고, ξd는 한 트랜젝션을 생성할 때 사용자가 계산하는 PoW 난이도고, ξs는 계정에서 담보로 내놓은 VITE(vite) 단위로 표시된 잔액이며, 또한 ξf는 사용자가 이 할당량 가치 증가를 위해 지불할 의사가 있는 한 회(one-time) 비용을 나타낸다.

13쪽
ξf 는 취급 수수료(handling fee)와 다르다는 데 유의해야 한다. 이 VITE 금액들은 채굴꾼들에게 지불되는 대신 직접 파괴될 것이다.

할당량을 나타내는데, 즉 1VITE를 파괴해 획득한 할당량은 담보로 내놓은 ps/pf VITE(vite)와 동등한 가치를 가진다. 사용자가 VITE 단위의 금액을 담보로 내놓지 않고 또한 트랜잭션을 하지 않을 경우, PoW를 반드시 계산할 필요가 있는데, 그렇게 하지 않으면 한 트랜잭션을 시작할 수 있는 할당량이 전혀 없을 것이며, PoW를 계산하면 더스트 공격(dust attack)와 시스템 자원 남용을 효과적으로 방지할 수 있기 때문이다. 동시에 이 공식은 논리적 기능이다. 사용자들이 더 낮은 할당량을 얻는 것은 상대적으로 쉽기 때문에, 따라서 방문 빈도가 낮은 사용자들의 역치를 감소시키고; 방문 빈도가 높은 사용자들은 더 높은 할당량을 획득하기 위해 많은 자원을 투자할 필요가 있다. 사용자들이 지불하는 추가 비용은 모든 사용자들의 혜택들을 증가시킬 것이다.

6.2.2 자원 정량화

스냅숏 체인은 글로벌 시계(global clock)에 상응하기 때문에, 우리는 정확하게 한 계정의 자원 이용을 정량화하는데 이 시계를 사용할 수 있다. 각 트랜잭션에서 한 스냅숏 블록의 해시가 인용되고, 스냅숏 블록의 높이는 트랜잭션의 타임 스탬프로 여겨진다. 따라서, 이 두 트랜잭션 타임 스탬프 사이의 차이에 따라 우리는 이 두 트랜잭션 사이 간격이 충분히 긴 지(long) 아닌지 판단할 수 있다.

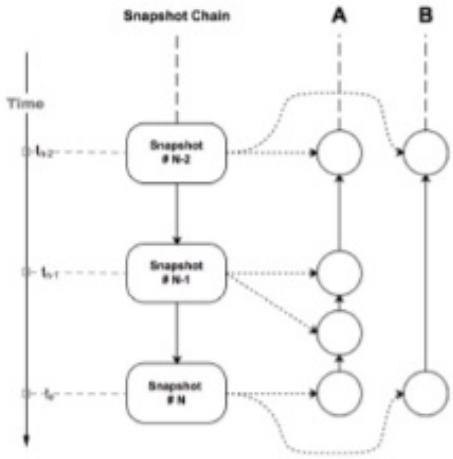

그림 9: 글로벌 시계로서 스냅숏 체인

위에 나온 것처럼, 계정 A는 2개의 시간 간격 내 4개의 트랜잭션들을 생성했는데 반면 계정 B는 2개 트랜잭션만 생성했다. 따라서 이 기간에서 계정 A의 평균 TPS는 계정 B의 것의 2배다. 스마트 계약들에 있어 각 교환은 자원들의 다른 소비를 갖고 있기 때문에, 한 시간 기간동안 평균 자원 소비를 계산하기 위해 각 트랜잭션 경우 가스를 축적하는 것이 반드시 필요하다. "n" 높이를 가진 한 계정 체인에서 최근 k 트랜잭션의 평균 자원 소비는 다음 공식으로 산출한다:
비용k(Tn) =k·∑n i=n−k+1 가스$_i$ 시간 스탬프$_n$−시간 스탬프$_n$−k+1 + 1 (7)

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

이들 중, 트랜잭션 Tn 경우, 타임스탬프$_n$은 트랜잭션의 타임 스탬프로, 즉 스냅숏 블록의 높이를 지칭하고; 가스$_n$은 트랜잭션에 소비된 연료를 지칭한다. 한 트랜잭션을 검증할 때, 이에 상응하는 노드는 할당량이 다음 조건 즉 Cost(T) ≤ Q 조건을 만족시키는지 여부를 판단할 것이며, 이 조건에 만족하지 못하면, 트랜잭션은 거부될 것이다. 이 경우, 사용자들은 한 트랜잭션을 재-일괄처리(repackage)하거나 한 회 수수료를 지불해 할당량을 늘리거나 아니면 트랜잭션에서 더 높은 스냅숏을 인용하기 위해 한 시간 기간 동안 기다릴 필요가 있다.

6.2.3 할당량 임대

사용자가 풍부한 VITE(vite) 자산을 갖고 있지만 아주 많은 자원 할당량을 활용할 필요가 없다면, 이 사용자는 다른 사용자들에게 자신의 할당량을 빌려주는 것을 선택할 수 있다.

VITE 시스템은 한 계정 자원 할당량을 이용할 권리를 전송할 수 있는 한 특별한 유형의 트랜잭션을 지지한다. 이 트랜잭션에서 담보로 내놓을 수 있는 VITE 수, 전송받는 자(transferee)의 주소, 한 임대의 지속 기간이 규정될 수 있다. 일단 거래가 확정되면, 토큰 금액에 상응하는 자원 할당량은 할당 받은 자의 계정에 포함될 것이다. 임대 시간이 지나면, 할당량은 전송하는 자

(transferor)의 계정으로 포함되는 것으로 계산될 것이다. 임대 시간의 단위는 초(second)다. 이 시스템은 스냅숏 블록의 높이 차이로 전환할 수 있으며 따라서 일부 편차가 있을 수 있다.

임대 소득(leasing income)은 사용자가 얻을 수 있다. VITE 시스템은 할당량 전송 트랜잭션만 제공할 뿐이며 임대의 가격 책정과 대금 지불은 제 3자의 스마트 계약을 통해 달성될 수 있다.

6.3 자산 발행

고유 토큰인 VITE 토큰뿐만 아니라, VITE는 또한 사용자들이 자신들의 토큰을 발행하는 것을 지원한다. 토큰 발행은 한 특별한 트랜잭션 즉 민트 트랜잭션을 통해 이루어질 수 있다. 민트 트랜잭션의 목표 대상 주소는 0이다. 이 트랜잭션의 필드 데이터에서 토큰의 매개 변수들은 다음과 같이 규정된다:

민트: {
이름: "MyToken",
총 공급량: 99999999900000000000000000,
소수점 자리: 18,

소유주: "0xa3c1f4...fa", 상징 기호: "MYT"
}

네트워크가 토큰 발행 요청을 받아들이면, 민트 트랜잭션에 포함된 VITE(vite)는 민트 트랜잭션 수수료로서 개시자(initiator)의 계정으로부터 공제될 것이다. 이 시스템은 새로운 토큰의 정보를 기록하고 한 토큰 _id를 이 토큰에 할당한다. 새로 생성된 토큰들의 잔액은 소유주의 주소로 추가될 것이며, 다시 말해서 소유주의 계정은 토큰의 최초 계정(genesis account)이 된다.

6.4 크로스 체인 프로토콜

디지털 자산의 크로스 가치 전송을 지지하고 "가치 섬(value island)"를 삭제하기 위해서, VITE는 VITE 크로스체인 프로토콜(VCTP)을 설계했다.

목표 대상 체인에서 크로스 체인 전송(transmission)이 필요한 모든 자산 경우, 이에 상응하는 토큰은 VITE에서 필요한데, VITE 안에서 회전하는 목표 대상 토큰의 이용권(voucher)은 ToT(Token of Token)이라고 불린다. 예를 들면, 여러분이 이더리움 계정에서 이더를 VITE로 전송시키고 싶다면, 여러분은 VITE에서 ETH의 식별자(identifier)와 함께 한 ToT를 발행할 수 있으며, ToT의 초기 정량(initial quantity)은 이더의 총 정량과 같아야 한다.

각 목표 대상 체인 경우, VITE 트랜잭션들과 목표 대상 체인 트랜잭션들의 매핑 관계를 유지할 수 있는 VITE에 한 게이트웨이 계약이 있다. 이 계약이 위치한 합의 글부에서, 블록들의 생성을 책임지는 노드는 VCTP 릴레이라고 불린다. VCTP 릴레이는 VITE의 노드가 될 필요가 있으며 동시에 목표 대상 체인의 풀 노드(full node)가 될 필요가 있으며, 양쪽 측면에선 트랜잭션들에 귀기울인다. 목표 대상 체인에서도 또한 VITE 게이트 계약을 배치할 필요가 있다.

VCTP 릴레이가 작업일 시작하기 전 VITE에서 이에 상응하는 ToT는 이 게이트웨이 계약으로 전송되어야 한다. 이것이 끝난 후, ToT 공급은 이 게이트 계약에 의해서만 통제될 것이며, 이 ToT

와 목표 대상 자산의 1:1 교환 비율을 보장하기 위해 어떤 사람도 추가할 필요가 없다. 동시에 목표 대상 체인에 있는 자산들은 이 VITE 게이트웨이 계약에 의해 통제되며, 또한 모든 사람이 ToT가 완전 수용 준비금 을 갖고 있다는 것을 보장하기 위해 이 VITE 게이트웨이 계약을 사용할 수 없다.

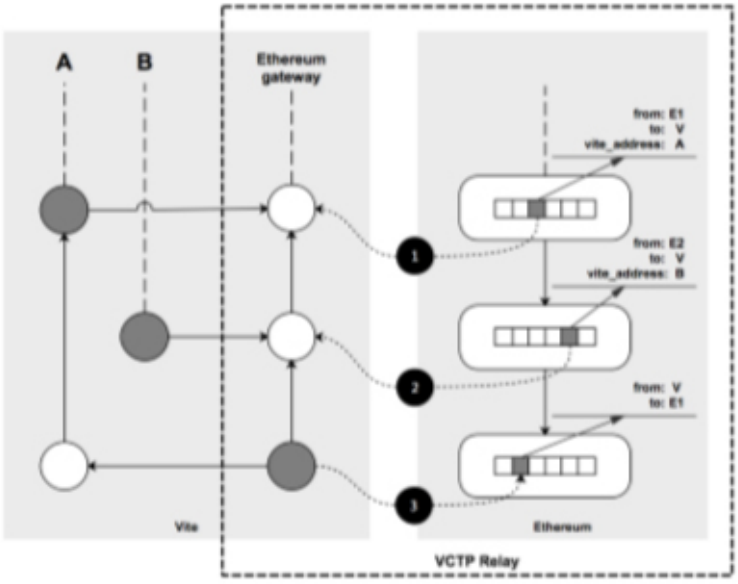

그림 10: 크로스 체인 프로토콜

위의 그림은 VITE와 이더리움 간의 크로스 체인 가치 전송을 보여주는 한 예다. 이더리움 사용자 E1이 이더리움에서 VITE로 토큰을 전송하고 싶을 때, VITE 게이트웨이 계약 주소 V로 한 트랜잭션을 보낼 수 있으며, 그 사이 VITE에서 이 사용자의 주소 A는 매개 변수에 놓여 있게 된다. 이 전송 잔액은 게이트웨이 계약 계정 안에 동결되고 ToT 준비금(ToT reserve)의 일부가 된다. 트랜잭션에 귀 기울인 후, 이 트랜잭션에 귀 기울인 후, VCTP 릴레이 노드는 VITE의 트랜잭션을 보내는 상응 계정을 생성해, VITE에 있는 이 사용자의 계정 A로 ToT로 동일 금액을 보낸다. 그림에서 ①과 ②는 각각 VITE 계정 A와 B로 보내는 E1과 E2의 전송을 나타낸다. 사용자가 전송할 때 VITE 주소를 명시하지 않을 경우, 이 계약은 이 트랜잭션을 거부할 수 있다는 것에 유념해야 한다.

이와 반대 방향 송금은 ③에 나와 있다. 사용자 A가 VITE 계정에서 이더리움 계정으로 전송을 시작할 때, 한 트랜잭션이 VITE 게이트 계약으로 전송될 것이며, VITE 게이트 계약은 이 트랜잭션을 특정 수량의 ToT로 전송하고, 이더리움 트랜잭션의 수신 주소 E1을 규정할 것이다. The VCTP 릴레이 노드는 이더리움 게이트 계약에서 이에 상응하는 응답 블록을 생성하고, 이어 이더리움에서 VITE 게이트웨이 계약으로 전송하는 이더리움 트랜잭션을 일괄 처리할 것이다. 이더리움에서, VITE 게이트웨이 계약은 이 트랜잭션이 한 신뢰하는 VCTP 릴레이에서 시작되는지 여부를 검증한 후, 이어 이더(ether)로 동일 금액을 VITE 게이트 계약으로부터 목표 대상 계정 E1으로 전송할 것이다.

모든 크로스 체인 릴레이 노드들이 목표 대상 네트워크를 모니터링하고, 또한 이 노드들은 각 크로스 체인 트랜잭션이 올바르고 합의 그룹 내에서 합의에 도달했는지 여부를 검증할 것이다. 그러나 스냅숏 합의 그룹은 목표 대상 체인의 트랜잭션을 모니터링 하지도, 이 두 체인 사이 맵핑이 제대로 되어 있는지도 검증하지 않을 것이다. 목표 대상 네트워크가 철회될 경우(roll back) 또는 심하게 분기된 경우, VITE 시스템에서 매핑된 트랜잭션들은 철회될 수 없으며; 마찬가지로 VITE에 있는 크로스 체인 트랜잭션들이 철회되면, 이에 상응하는 목표 대상 네트워크의 트랜잭션은 동시에 철회될 수 없다.

15쪽
따라서 크로스 체인 트랜잭션을 할 땐, 계약 논리에서 트랜잭션 철회를 반드시 다룰 필요가 있다. 동시에 4.4 파트에서 기술된 것처럼 우리는 크로스 체인 릴레이 합의 그룹을 위한 한 지연 매개변수를 설정할 필요가 있다.

6.5 루프링 프로토콜

루프링 프로토콜은 분산된 자산 거래 네트워크(trading network)를 구축할 수 있는 개방 프로토콜이다. 다른 DEX 솔루션들과 비교해 보면 루프링 프로토콜은 멀티파티 루프 매칭(multiparty loop matching)에 기반하고 있는데, 이것은 선매 트랜잭션(preemptive transactions)을 예방할 수 있는 한 이중 승인된 기술(dual authorized technology)를 제공하고 완전히 개방되어 있다.

우리는 VITE 내부에 루프링 프로토콜을 구축했는데, 이것은 VITE에 있는 디지털 자산의 순환을 촉진시키는데 도움이 되어, 따라서 전체 가치 시스템이 순환할 수 있다. 이 가치 시스템에서 사용자들은 자신들의 디지털 자산을 발행하고, VCTP를 통해 체인 밖으로 자산을 전송할 숭 | 씨고 또한 자산 교환을 달성하기 위해 루프링 프로토콜을 이용할 수 있다. 이 전 과정은 VITE 시스템 내에서 완료되고 완전히 분산된다.

VITE에서 루프링 프로토콜 스마트 계약(LPSC)는 VITE 시스템의 한 부분이다. VITE에선 자산 전송 승인과 멀티 파티 원자성(multi-party atomic) 보호, 모두 지원받고 있다. 루프링 릴레이는 아직까지 개방되어 있어 자신의 생태계와 완전히 결합할 수 있다.

사용자들은 자산 교환 트랜잭션들 관련 비용을 VITE(vite)로 지불할 수 있어서, 따라서 VITE 플랫폼에서 루프 매칭을 수행하는 루프링 채굴꾼들이 벌어들인 토큰은 여전히 VITE 단위(vite)의 가상 화폐다.

7 기타 설계

7.1 일정 관리

이더리움에서, 스마트 계약은 트랜잭션에 의해 이루어지며 또한 계약 실행은 한 트랜잭션을 시작하는 사용자들에 의해서만 촉발될 수 있다. 일부 애플리케이션에선, 시간 일정 관리(timing scheduling) 기능은 한 시계를 통해 계약 실행을 촉발하는데 필요하다.

이더리움에서 이 기능은 제3자 계약을 통해 달성되지만 성능과 보안은 보장되지 않는다. VITE에서 우리는 내장된 계약에 이 시간 일정 관리 기능을 추가했다. 사용자들은 그들의 일정 관리 논리를 시간이 정해진 일정 관리 계약(timed scheduling contract)에 등록할 수 있다. 공공 합의 그룹(public consensus group)은 스냅숏 체인을 시계로 사용해, 사용자 규정 일정 관리 논리에 따라

목표 대상 계약으로 요청 트랜잭션을 보낼 것이다.

솔리디티++에는 한 전문적인 타이머 메시지가 있다. 사용자들은 Timer.on을 통해 계약 코드로 자신들의 일정 관리 논리를 설정할 수 있다.

7.2 이름 서비스

이더리움에서 한 계약을 배치할 때 그것을 확인할 수 있는 주소를 생성할 것이다. 주소들이 있는 계약들을 확인하는 것과 관련해 다음 두 가지 문제가 있다:

• 주소는 의미 없이 20바이트(bytes)를 가진 식별자다. 이것은 사용자들에게 불친절하고 사용하기 불편하다.

• 계약과 주소는 1대1로 배정된다. 이들은 계약 재-방향(redirection)을 지원할 수 없다.

이 두 가지 문제를 해결하기 위해 이더리움 개발자는 제3자 계약 ENS[2]를 제공했다. 그러나 실제 시나리오에서 명명 서비스는 아주 자주 이용될 것이며, 또한 제3자 계약 이용이 명명의 세계적 고유성(global uniqueness)을 보장할 수 없기 때문에, 따라서 우리는 이름 서비스 VNS(ViteName Service)를 VITE 내부에 구축했다.

사용자들은 기억하기 쉬운 일련의 이름들을 등록하고 그것들을 VNS를 통해 실제 주소로 변환할 수(resolve) 있다. 이름들은 vite.myname.mycontract 같이 도메인 이름 형태로 구성된다. 최상위(top-level) 도메인 이름은 특정 목적들을 위해 시스템이 보관할 것이다. 예컨대 vite.xx는 VITE(Vite) 주소를 나타내며 eth.xx는 이더리움 주소를 나타낸다. 차상위(second-level) 도메인 이름은 모든 사용자들에게 개방된다. 사용자가 차상위 도메인 이름을 소유하게 되면, 서브 도메인은 사용자가 임의로 확장할 수 있다. 도메인 이름 소유자는 언제든지 도메인에 의해 지시되는 주소를 수정할 수 있기 때문에, 따라서 이 기능은 계약 업그레이드하는데 이용될 수 있다.

도메인 이름 길이는 제한이 없다. VNS에서, 도메인 이름의 해시가 실제로 저장된다. 목표 대상 주소는 256비트(bit) 미만의, VITE가 아닌(non-Vite) 주소가 될 수 있는데, 이러한 주소는 크로스 체인 상호 작용에 이용할 수 있다. VNS는 이더리움에 있는 스마트 계약 패키지 사양 EIP1903과 다르다는 점에 유의해야 한다. VNS는 이름 변환 서비스(name resolution service)로, 이름은 실행 시간(runtime) 때 정해지고, 또한 변환 규칙(resolution rules)은 역동적으로 바뀔 수 있으며; EIP190은 한 패키지 관리 사양으로, 명칭 공간은 정적이며, 이것은 컴파일 시간 때 정해진다.

7.3 계약 갱신

이더리움의 스마트 계약은 변경할 수 없다. 이것은 수정될 수 없다. 이 계약에 버그가 있다 할지라도, 이 계약은 갱신될 수 없다. 이것은 개발자들에게 매우 불친절하고 또한 dApp의 지속적인 반복 계산/처리를 매우 어렵게 만든다. 따라서 VITE는 스마트 계약 갱신을 지원하는 도식을 제공할 필요가 있다.

VITE에는 계약 갱신 과정을 포함한다:

A. 원래 계약 지위를 승계하기 위해 이 계약의 신규 버전을 배치한다.

B. 계약 이름을 VNS에 있는 신규 주소로 지정한다.

C. SELFDESTRUCT 명령을 내려 구 계약을 제거한다.

이 3가지 단계는 동시에 완료될 필요가 있으며 VITE 프로토콜은 이 연산의 원자성(atomicity)를 보장한다. 개발자들은 구 계약 데이터가 새로운 버전 계약에서 올바르게 처리된다는 것을 보장할 필요가 있다.

이 새로운 계약은 구 계약의 주소를 승계하지 않을 것이란 점에 유의해야 한다. 이 주소에 의해 인용된 경우 트랜잭션은 여전히 구 계약으로 보내질 것이다. 그렇게 되는 이유는 다양한 버전의 계약들은 근본적으로 계약의 의미에 따라 완전히 다른 두 개의 계약으로 계약의 의미에 따라 역동적으로 수정될 수 있거나 수정될 수 없는 계약이기 때문이다.

VITE 시스템에서 스마트 계약은 실제로 두 개의 유형으로 나뉘어지는데, 첫 번째 유형은 dApp의 배경이고, 이것의 비즈니스 로직이 기술되며; 두 번째 유형은 실제 세상을 매핑하는 한 종류의 계약이다. 이전 계약은 한 애플리케이션의 배경과 같다 할 수 있는데, 이 서비스는 업그레이드를 통해 지속적으로 통합될 필요가 있으며, 후자는 하나의 계약과 같다 할 수 있으며, 일단 계약이 발효하면 어떤 변경할 수 없으며, 그렇지 않고 변경하면 그것은 계약 위반이 된다. 변경이 허용되지 않는 이런 유형의 계약 경우, 다음과 같이 솔리디티++에선 고정 키워드(keyword static)로 장식할 수 있다:

```
프로그램 솔리디티++ ^0.1.0;
고정 계약 약속 {
//결코 바뀌지 않을 계약
}
```

7.4 블록 프루닝(전지 작업)

원장에서 한 트랜잭션은 변경할 수 없으며, 사용자들은 역사적 트랜잭션들을 변경하거나 삭제하지 않고 원장에 새로운 트랜잭션만 추가할 수 있다. 따라서 시스템 작동과 함께 원장은 점점 더 커지게 될 것이다한 새로운 노드가 참여하기 원하면 네트워크는 최초 블록(genesis block)으로부터 시작해 모든 역사적 트랜잭션들을 다시 하며, 최신 지위를 복원하기 원한다. 일정 시간동안 시스템을 가동한 후, 계정 장부가 차지하는 공간과 트랜잭션들을 다시 하는데 소비된 시간은 수용할 수 없는 것이 된다. VITE의 높은 출력량 시스템 덕분에, 성장 속도는 비트 코인과 이더리움보다 훨씬 더 높아질 것이며, 따라서 원장에서 블록 클립핑(clipping, 자르기) 기법을 제공하는 것이 반드시 필요하다.

블록 클립핑(block clipping)은 원장에서 사용할 수 없는 역사적 트랜잭션의 삭제를 지칭하며, 이것은 트랜잭션 상태 기계 작동에 영향을 미치지 않는다. 그렇다면 어떤 트랜잭션들이 안전하게 삭제될 수 있을까? 그것은 다음을 포함해 이 트랜잭션이 사용될 시나리오에 달려 있다:

• 트랜잭션의 주요 역할은 지위를 회복하는 것이다. VITE에서 스냅숏 체인은 계정 지위의 스냅숏 정보를 저장하기 때문에 노드들은 한 스냅숏 블록에서 상태를 회복시킬 수 있다. 스냅숏 블록에 있는 마지막 트랜잭션에 앞서 모든 트랜잭션들은 상태 회복에 맞춰 맞춤 제작될 수 있다.

• 트랜잭션 검증. 한 새로운 트랜잭션을 검증하려면, 이 계정 체인에 있는 교환의 이전 트랜잭션을 검증할 필요가 있으며, 한 응답 트랜잭션이 있으면, 그에 상응하는 요청 트랜잭션을 검증할 필요도 있다. 따라서 맞춤 제작된 계정 원장(accounting ledger)에서, 최소 1개 마지막 트랜잭션은

각 계정 체인에서 보관해야 한다. 게다가 모든 개방 요청 트랜잭션들은 맞춤 제작할 수 없는데 이들의 해시들을 후속 응답 트랜잭션들이 참조할 수 있기 때문이다.

• 할당량 계산하기. 한 트랜잭션이 할당량을 충족시키는지 여부는 마지막 트랜잭션 자원들의 슬라이딩 평균을 판단해 계산할 수 있기 때문에, 따라서 각 계정 체인마다 최소 마지막 트랜잭션 9개를 저장해 둘 필요가 있다.

• 역사에 대한 질의. 노드들이 트랜잭션 역사에 대해 질의할 필요가 있다면, 이 질의에 관여된 트랜잭션은 맞춤 제작할 필요가 없다.

다양한 사용 시나리오들에 따라 위의 클립핑(자르기) 전략들의 다양한 조합들을 선택할 수 있다. 클립핑은 원장에서 트랜잭션에 관여하지만, 반면 스넵숏 체인은 그대로 있을 필요가 있다는 점에 유의하는 것이 중요하다. 게다가 스냅숏 체인에 기록된 것은 계약 상태의 해시다. 계정이 클립 되었을 때(잘랐을 때), 이에 상응하는 스냅숏 상태는 그대로 있을 필요가 있다.

VITE 데이터의 무결성을 보장하기 위해, 우리는 모든 트랜잭션 데이터를 저장할 수 있는 네트워크 안에 일부 "풀 노드(full nodes)"를 보관할 필요가 있다. 스냅숏 합의 그룹 노드들이 풀 노드(full nodes)이며, 또한 게다가 교환 같은 중요한 사용자들 역시 풀 노드(full nodes)가 될 수 있다.

8 거버넌스(governance)

분산 애플리케이션 플랫폼 경우, 효율적인 거버넌스 시스템(governance system)은 건강한 생태계를 유지하는데 필수적이다. 거버넌스 시스템을 설계할 대 효율성과 공정성을 고려해야 한다.

VITE의 거버넌스 시스템은 두 파트 즉 온-체인과 오프-체인으로 구분된다. 온-체인(on-chain_은 프로토콜에 기반한 투표 메커니즘이며 오프-체인은 프로토콜 자체의 반복이다.

투표 메커니즘에서, 투표는 두 가지 유형 즉 세계 투표(global voting)과 지역 투표(local voting)으로 나뉜다. 세계 투표(global voting)은 투표 무게로서 권리를 계산하기 위해 사용자가 보유한 VITE 수에 기반한다. 세계 투표(global voting)은 스냅숏 합의 그룹 프록시 노드를 선출할 때 주로 사용한다. 지역 투표(local voting)은 계약을 목적으로 한다. 계약이 배치되면, 투표를 위한 토대로서 한 개의 토큰이 지정된다. 이 토큰은 계약이 위치한 합의 그룹의 에이전트 노드를 선출하는데 이용될 수 있다.

17쪽

트랜잭션 검증뿐만 아니라 스냅숏 합의 그룹의 에이전트 노드는 VITE 시스템의 비호환성(incompatibility)를 업그레이드 여부를 선택할 수 있는 권리를 가진다. 지정된 합의 그룹 프록시 노드는 계약들의 단계적 확대(escalation)으로부터 제기되는 잠재적 위험들을 피하기 위해 계약이 업그레이드되는 것을 허용할지 말지 결정할 권리를 갖고 있다. 에이전트 노드는 의사 결정의 효율성을 높이고 또한 투표 참여 저조로 인한 의사 결정 실패를 피하기 위해 사용자들을 대표해 의사 결정력을 업그레이드하는데 이용된다. 이 프록시 노드들 자체 역시 합의 프로토콜에 의해 제약 받는다. 대부분 에이전트 노드들이 통과했을 때만 업그레이드 효력을 발휘할 것이다. 이러한 에이전트들은 사용자의 기대에 따라 자신들의 의사 결정력을 충분히 발휘하지 못할 경우, 사용자들은 또한 투표를 통해 이들의 프록시 자격을 취소할 수 있다.

오프-체인의 거버넌스는 커뮤니티를 통해 실현된다. VITE의 어떤 커뮤니티 참가자도 VITE 프로토콜 자체나 관련 시스템의 개선안을 제안할 수 있으며, 이러한 제안은 VEP(VITE 향상 제안)으로 불린다. VEP는 커뮤니티에서 광범위하게 논의될 수 있으며 솔루션을 실행할지 여부는 VITE의 생태계 참가자들에 의해 결정된다. 한 VELP의 실행을 위해 프로토콜을 업그레이드 여부는 최종적으로는 에이전트 노드에 의해 결정된다. 물론 업그레이드 찬반 차이가 클 경우, 여러분은 또한 광범위한 사용자 의견들을 수집하기 위해 체인에서 투표를 시작할 수 있으며, 그 투표 결과에 따라 업그레이드할 지 여부를 프록시 노드가 결정할 것이다.

일부 VITE 참가자들은 찬반에 대한 자신의 의견을 표시하기 위해 투표하기에 필요한 VITE 토큰이 부족할 수 있다. 그러나 그들은 자유롭게 VEP를 제출하고 자신들의 의견을 충분히 표현할 수 있다. 투표 권리를 가진 사용자들은 그들 자신의 VITE 권리를 위해 전체 생태계의 건강을 전적으로 고려해야 하고, 그에 따라 진지하게 모든 생태적 참가자들의 의견을 받아들여야 한다.

9 향후 과제

스냅숏 체인에서 트랜잭션 검증은 시스템의 주요한 성능 장애물이다. VITE는 비-동기 설계와 DAG 계정 구조를 채택하기 때문에 트랜잭션 검증을 병행해서 집행할 수 있다. 그러나 서로 다른 두 계정의 트랜잭션들 사이에 의존하기 때문에 병행성(parallelism) 수준은 크게 제약을 받게 된다. 트랜잭션 검증의 병행성을 개선하거나 분산 검증 전략을 채택하는 방법이 향후 최적화를 위한 중요한 방향이 될 것이다.

또한 현재 HDPoS 합의 알고리즘에도 일부 단점은 존재한다. 이 합의 알고리즘을 개선하거나 또는 지정된 합의 그룹에서 더 많은 합의 알고리즘들과 호환할 수 있는 것 역시 최적화 방향이다. 게다가 가상 기계의 최적화 역시 시스템 지연을 줄이고 시스템 처리량을 개선하는데 아주 중요하다. EVM의 단순 설계와 명령 세트의 단순화 때문에, 향후에 더욱 강력한 가상 기계를 설계하고 더 많은 서술 능력과 더 작은 보안 취약점들을 가진 스마트 계약 프로그래밍 언어를 정의하는 것이 반드시 필요할 수 있다.

마지막으로 VITE의 핵심 합의뿐만 아니라 생태적 발전을 지원하는 보조 시설들의 건설 역시 중요한 화두다. dApp 개발자들을 위한 SDK 지원뿐만 아니라, dApp 전면 생태계(foreground ecosystem) 건설에는 해야 할 일이 많다. 예를 들어 여러분은 VITE의 모바일 지갑 애플리케이션에서 HTML5에 기반한 dApp 엔진을 구축할 수 있으며, 이러한 엔진이 구축되면 개발자들은 낮은 비용으로 dApp을 개발하고 발표할 수 있다. 다른 유사한 프로젝트들

10 요약

다른 유사한 프로젝트들과 비교하면, VITE는 다음과 같은 특징들을 포함하고 있다:

• 높은 처리량. VITE는 DAG 원장 구조를 이용하고 있어, 직교 트랜잭션은 계정 장부에 병행해서 기록될 수 있으며; 게다가, 여러 합의 그룹들은 HDPoS 합의 알고리즘에서 서로에 의존하지 않고 병행으로 작업할 수 있으며; 가장 중요한 점은 VITE의 계약 간 소통(inter contract communication)은 메시지의 비-동기 모델에 기반하고 있다는 것이다. 이 모든 특징들은 이 시스템의 처리량을 개선하는데 도움이 된다.

• 낮은 지연성. VITE는 PoW를 계산할 필요 없이 프록시 노드를 통해 회전 생산 블록(rotation production block)을 완수하기 위해 협업할 수 있는 HDPoS 합의 알고리즘을 이용하기 때문에, 블

록 간격(block interval)은 1초 이내로 줄어들 수 있는데, 이는 트랜잭션 확정 지연을 줄이는데 도움이 된다.

• 확장성. 확장성 요건들을 충족하기 위해, VITE는 트랜잭션에 단일 자유도 한계(single degree of freedom limit)를 만들었으며, 계정 차원에 따라 계정에서 트랜잭션을 그룹 별로 분류해, 서로 다른 노드들에서 서로 다른 계정들의 블록 생산이 완료되고 또한 메시지에 기반해 BASE 의미로 보내지는 크로스 계약 호출의 ACID 의미를 제거하는 것을 가능하게 한다. 이런 식으로 하면 노드들은 더 이상 세계의 모든 상태를 저장할 필요가 없으며, 데이터는 샤딩 모드(sharding mode)로 전체 분산 네트워크에 저장된다.

• 가용성(usability). VITE의 가용성 개선 특징들 중에는 솔리디티++에서 처리 메시지 구문, 계약의 시간 일정 관리, VNS 명명 서비스, 계약 업그레이드 지원 등을 전적으로 처리하는 표준 라이브러리 지원 제공이 포함되어 있다.

• 가치 계산. VITE는 디지털 자산 발행, 크로스 체인 가치 전송, 루프링 프로토콜에 기반한 토큰 교환 등을 지원해, 완전한 가치 시스템을 형성한다. 사용자 관점에서 보면 VITE는 완전한 기능 분산 교환소다.

18쪽

• 경제. VITE는 할당량에 기반한 자원 할당 모델을 채택하기 때문에, 자주 거래하지 않는 경량(lightweight) 사용자들은 높은 수수료나 가스 요금을 지불할 필요가 없다. 사용자들은 계산을 바꿀 다양한 방법을 선택할 수 있다. 또한 시스템 자원 활용의 효율성을 개선하기 위해 할당량 임대 합의를 통해 추가 할당량을 다른 사용자에게 전송할 수도 있다.

11 감사의 말

진심으로 조언해 주고 이 논문을 작성하는데 도움을 준 우리의 컨설턴트들에게 감사를 전하고 싶다. 특히 이 프로젝트에 기여해 준 루프링 팀과 루프링 커뮤니티에 정말 감사하다는 인사를 전하고 싶다.