공학석사학위논문

이더리움 전역 상태 트라이 분석 도구

Ethereum World State Trie Analysis Tool

2022년 8월

서울대학교 대학원

전기 정보 공학부

김 재 훈

공학석사학위논문

이더리움 전역 상태 트라이 분석 도구

Ethereum World State Trie Analysis Tool

2022년 8월

서울대학교 대학원

전기 정보 공학부

김 재 훈

이더리움 전역 상태 트라이 분석 도구

Ethereum World State Trie Analysis Tool

지도교수 문 수 묵

이 논문을 공학석사 학위논문으로 제출함

2022년 8월

서울대학교 대학원

전기 정보 공학부

김 재 훈

김재훈의 공학석사 학위 논문을 인준함

2022년 8월

위 원 장: _____

부위원장: _____

위　　원: _____

초록

이더리움[1]은 전 세계에서 시가총액이 두번째로 큰 암호화폐이자 가장 활발히 사용되는 블록체인으로, 암호화폐와 블록체인에 대한 관심이 높아짐에 따라 폭발적으로 거래량이 증가했다. 이더리움은 모든 사용자의 상태를 저장하는 상태 기반(State-based)블록체인으로, 거래가 증가함에 따라 계좌 수와 트랜잭션 데이터가 폭증했다. 증가한 데이터는 네트워크 동기화(Synchronization) 시간을 증가시키고, 트랜잭션 수행 성능을 감소시키며 필요한 저장 용량을 증가시켰다. 이렇게 증가한 데이터의 95%는 상태 트라이(State trie)와 스토리지 트라이(Storage trie)를 포함하는 전역 상태 트라이(World state trie)이다. 이러한 문제를 해결하기 위하여 상태 트라이의 경량화 및 최적화를 위한 많은 기법[8, 9]들이 제안되고 있으며, 이들의 분석을 위한 상태 트라이 분석 도구 개발이 필요해졌다.

본 논문에서는 이더리움의 상태 데이터를 저장하는 전역 상태 트라이를 분석하는 도구를 개발하고 이를 이용하여 특성을 파악했다. 이더리움 전역 상태 트라이의 크기, 구성 노드들의 비율과 깊이, 스토리지 트라이의 통계 등의 데이터를 얻을 수 있으며, 이더리움 블록체인 탐색기인 Etherscan[2]보다 정밀한 storage 관련 분석 데이터를 제공한다. 본 분석 도구를 활용하여 효율적인 상태 트라이 최적화 설계에 기여하며, 상태 트라이 분석을 통한 공격을 방지한다.

주요어: 이더리움, 블록체인, 상태 트라이, 머클 패트리시아 트라이

학 번: 2020-23168

목차

표 목차

그림 목차

제 1 장 서론

암호화폐에 대한 관심이 증가함에 따라 비트코인, 이더리움과 같은 블록체인의 사용량이 폭발적으로 증가했다. 이더리움은 블록체인의 데이터를 모든 사용자의 계정 상태로 저장하는 상태 기반(State-based) 블록체인의 대표적인 예시이다. 이더리움은 상태 데이터를 머클 패트리시아 트라이(Merkle Patricica Trie)라는 특수한 키-값(Key-Value) 맵핑 자료구조에 저장하고, 이더리움의 모든 사용자들은 상태정보를 담고있는 이 머클 패트리시아 트라이로 데이터의 무결성을 검사하게 된다.

블록이 진행하여 사용자가 증가할수록 저장해야하는 계정의 숫자와 트랜잭션 데이터의 크기가 증가하는데, 이미 2022년 2월 이더리움의 아카이브 싱크(Archive sync) 노드의 크기는 10TB를 초과[10]했다. 폭증한 데이터는 새로운 블록체인 사용자가 네트워크에 참여하기 위해 동기화(Synchronization)하는 시간을 증가시키고, 트랜잭션의 처리를 위해 소요되는 시간을 증가시키는 등의 문제를 발생시킨다. 표 1은 이더리움 데이터의 구성 비율을 분석한 결과이다. 데이터의 96%이상을 상태 트라이와 스토리지 트라이를 포함하는 전역 상태 트라이의 노드들이 차지하는 것을 확인할 수 있다.

최근 이러한 한계점을 해결하기 위해 이더리움 경량 클라이언트에 대한 연구들이 제안되고 있다. 기존의 연구들은 개선된 형태의 PoW(Proof-of-Work)의 제시로 빠른 합의(Consensus)를 제시[3]하거나 TEE(Trusted Execution Environment)를 이용해 효율적인 트랜잭션의 수행을 제안[4]한다. 하지만 상태 트라이의 크기가 폭발적으로 증가하는 문제점에 대해서는 해결하지 못하였으며, 이 계속 대두되고 있으며, 스토리지 관점에서의 블록체인 분석은 아직 이루어지지 않고있다. 본 논문에서는 이더리움 블록체인을 위한 효율적인 데이터베이스 설계를 위하여 이더리움의 전역 상태 트라이의 분석 도구를 제안한다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어 2장에서는 이더리움의 전역

표 1.1: 이더리움 데이터베이스 분석결과

Category	Ratio(%)
Headers	0.001
Bodies	0.069
Receipt lists	0.065
Transaction index	0.561
Contract Codes	0.031
Trie nodes	96.108
Ancient store	3.166
Etc.	0.458
Total	100

상태 트라이의 분석에 앞서 알아야 할 이더리움의 기본 개념을 설명한다. 3장에서는 전역 상태 트라이 분석 도구의 방법론을 제시하며, 이후 4장에서는 이러한 방법론을 이용한 분석 결과를 제시한다. 5장에서는 본 논문의 분석 도구의 사용 사례를 제시하고, 6장에서는 관련 연구를, 7장에서는 결론과 향후 연구 방향에 대해 소개할 것이다.

제 2 장 배경지식

2.1 이더리움 기본 개념

이더리움은 비트코인 다음으로 시가총액이 가장 큰 암호화폐이면서, 스마트 계약을 지원하는 가장 큰 블록체인 이다.

이더리움은 비트코인과 같은 블록체인과 다르게 스마트 계약(Smart contract)이라는 시스템을 가지고 있다. 스마트 계약은 계약의 당사자들이 사전에 협의한 조건을 미리 프로그래밍하여 전자 계약서 문서에 넣어고, 조건을 충족시키면 자동으로 계약 내용을 수행한다. 이 스마트 계약은 튜링 완전(Turing complete)언어인 솔리디티(Solidity)로 작성되기 때문에 다양한 종류의 계약을 구현할 수 있으며, 이를 이용한 분산 애플리케이션인 디앱(DApp)도 꾸준히 배포 및 이용되고 있다.

상태 기반(State-based) 블록체인인 이더리움은 모든 사용자 계정의 잔고, 논스(Nonce), 스마트 계약 코드해시 값, 스토리지 트라이 루트해시값을 머클 패트리시아 트라이라는 자료구조에 저장한다. 이더리움의 계정은 외부 소유 계정(EOA, Externally Owned Account)과 컨트랙트 계정(CA, Contract Account) 두 가지 종류가 존재한다. 외부 소유 계정은 일반 사용자가 트랜잭션을 발동시키기 위해 사용하는 계정으로 이더리움 잔고와 논스 값을 가지고 있고, 스마트 계약 코드해시는 빈 코드를 가리킨다. 컨트랙트 계정도 이더리움 잔고와 논스 값을 가지고 있으며 스마트 계약 코드를 가리키는 코드 해시값과 스마트 계약에서 사용할 스토리지 공간을 가리키는 스토리지 트라이 루트해시값을 가지고 있다.

2.1.1 머클 패트리시아 트라이

이더리움의 상태 트라이와 스토리지 트라이는 머클 패트리시아 트리(Merkle Patricia Tree)라는 키-값(Key-Value) 매핑 자료구조로 구현되어 있다. 머클 패트리시아

트라이는 정보의 효율적인 수정, 검색, 삭제를 위한 패트리시아 트리(Patricia Tree)와 신속하게 데이터의 올바름 검증이 가능한 머클 트리(Merkle Tree)의 장점을 합친 자료구조이다. 머클 트리의 루트노드는 트리에 포함된 모든 리프 노드의 정보를 하나의 해시값으로 압축한 머클 루트를 가지고 있으며, 이를 이용해 서로 다른 머클 트리가 동일한 내용을 담고있는지 빠르게 판단할 수 있다. 트리의 가장 가까운 노드끼리 묶어 해시화하고 더이상 쌍을 지을 수 없을때까지 이 과정을 반복하여 구해진 머클 루트이기 때문에, 역으로 어떤 리프노드가 머클 트리에 포함되어있는지 검증하는데 필요한 머클 증명(Merkle proof)가 있다면 빠르게 증명할 수 있다. 패트리시아 트리는 키-값 쌍의 값을 담고있는 리프노드까지의 경로를 키로하는 래딕스 트리(Radix Tree)의 한 종류이다. 공통된 prefix를 가지는 노드들은 같은 경로를 공유하기 때문에 적은 저장공간으로 구현할 수 있다. 따라서 머클 패트리시아 트라이는 리프 노드에 저장할 데이터의 키를 리프노드까지의 경로로 하며, 작은 데이터인 머클 루트를 이용해 쉽게 전체 데이터의 변화여부를 파악할 수 있는 자료구조이다.

상태 트라이에는 계정 주소의 해시값과 해당 계정의 상태가 키-값 쌍으로 저장된다. 외부 소유 계정은 남아있는 이더리움 잔고와 해당 계정이 몇 번의 트랜잭션을 발생시켰는지를 센 논스 값을 가지고 있다. 컨트랙트 계정은 추가적으로 코드 해시값과 스토리지 트라이 루트해시값을 가지고 있다. 스토리지 트라이에 저장되는 데이터는 해당 컨트랙트가 실행되기 위해 필요한 변수들과 스마트 계약의 수행 결과 등을 저장한다.

머클 패트리시아 트라이를 구성하는 노드는 리프 노드(Leaf Node), 브랜치 노드(Branch Node), 익스텐션 노드(Extension Node)로 총 3가지가 있으며 각 노드는 노드의 해시값을 키로 하여 데이터베이스에 저장된다. 리프 노드는 트리의 가장 마지막에 위치하며 계정의 상태정보를 저장하고 있다. 해당 계정 주소의 해시값을 경로로 하여 상태 트라이를 내려가면 찾을 수 있다. 브랜치 노드는 40자리의 16진수로 이루어진 이더리움 계좌 주소 해시값을 찾아가는 경로(Path)의 분기점을 만들어주는 노드이다. 최대 16개의 자식 노드와 연결될 수 있으며, 다른 브랜치 노드, 익스텐션

노드, 리프 노드와 연결될 수 있다. 익스텐션 노드는 리프노드까지의 경로에서 불필요한 브랜치 노드의 사용을 피하고 저장 공간을 절약하기 위한 노드이다. 1개의 자식 노드와 연결된 브랜치 노드를 공통된 주소 prefix를 가지는 1개의 익스텐션 노드로 치환한다. 자식 노드가 1개뿐인 브랜치 노드는 모두 익스텐션 노드로 치환될 수 있으며, 연속으로 연결된 익스텐션 노드는 합쳐져 다양한 길이의 공통 주소 prefix를 가지는 익스텐션 노드로 치환할 수 있다.

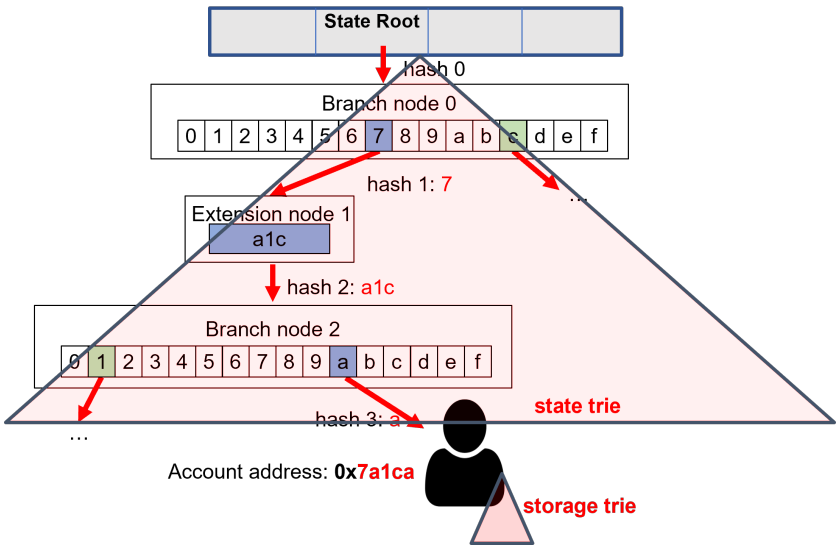

그림 2.1: 머클 패트리시아 트라이

그림 2.1은 머클 패트리시아 트라이로 구현된 이더리움 상태 트라이에 해시값 0x7a1ca를 주소로 가지는 계정이 저장되는 예시이다. 머클 패트리시아 트라이의 루트노드에서 첫번째 브랜치 노드로 진입한다. 이 브랜치 노드에서 계정 주소의 첫번째 글자인 7에 해당하는 자식 노드로 접근한다. 데이터베이스에서 확인한 자식 노드는 a1c라는 공통 주소 prefix를 가지고 있으며 다음 브랜치 노드로 연결되어 있다. 다음 브랜치 노드에서는 남은 주소 a에 해당하는 자식 노드로 접근한다. 이 자식노드에서 0x7a1ca라는 주소의 계정 정보를 확인하게된다. 이 계정이 EOA라

서 트랜잭션을 보내거나, CA라서 컨트랙트를 실행하기 위해서는 매 트랜잭션마다 머클패트리시아 트라이에서 계정을 찾아야하며 이때마다 데이터베이스에 여러번 접근하게된다.

머클 패트리시아 트라이의 모든 구성 노드는 각각 노드 해시와 데이터를 키-값으로 맵핑(mapping)하여 levelDB라는 데이터베이스에 저장된다. 노드의 데이터에는 익스텐션 노드와 브랜치 노드가 가지고 있는 계정의 주소 해시값에 대한 정보와 그 노드가 가지고 있는 자식 노드의 데이터베이스 위치가 포함되어있다. 즉, 상태 트라이에서 어떤 계정의 상태를 알기 위해서는 계정의 정보를 담고있는 리프 노드까지의 경로에 위치한 모든 노드의 정보를 데이터베이스에서 접근해야한다. 어떤 노드의 정보를 얻기 위하여 데이터베이스에 접근하는 횟수를 깊이(depth)라고 표현하며, 노드의 깊이가 깊어질수록 상태 트라이의 root node로부터 멀리 떨어져있으며 root node의 깊이는 1이다. 예를 들어,그림 2.1에서 이 노드의 정보를 얻기 위해서 데이터베이스에 4번 접근해야하기 때문에 깊이가 4가 된다.

2.2 트랜잭션 수행에 의한 상태 변화

이더리움에는 참여자가 능동적으로 발생시킬 수 있는 3가지 종류의 트랜잭션과 컨트랙트 내용에 의해 자동으로 발생하는 1가지 종류의 수동 트랜잭션이 존재한다. 능동 트랜잭션에는 외부 소유 계정끼리의 이더리움 잔고를 전송하는 전달(Transfer) 트랜잭션, 새로운 스마트 계약을 생성하는 계약 배포(Contract Deploy) 트랜잭션, 그리고 배포된 스마트 계약을 실행하는 계약 호출(Contract Call) 트랜잭션이다. 계약 배포와 계약 호출 트랜잭션에서 컨트랙트의 내용에 따라 수동 트랜잭션인 계약 내부 트랜잭션(Contract internal transaction)이 발생한다. 모든 트랜잭션의 내용은 영수증(Transaction receipt)이라는 형태로 그 결과가 저장된다. 영수증에는 블록의 해시값, 블록 넘버, 보내는 사람과 받는 사람의 주소, 해당 트랜잭션의 해시값 등이 저장되어 있다.

트랜잭션의 수행과 상관없이 모든 블록의 채굴자(Miner)에게 채굴 보상(Mining reward)를 위한 각 트랜잭션의 수행으로 인해 발생하는 이더리움 상태 트라이의 변화는 다음과 같다. 전달 트랜잭션에서 받는 사람의 잔고는 보내는 이더리움만큼 증가한다. 보내는 사람의 잔고는 보내는 이더리움과 트랜잭션을 생성하기 위한 가스비가 추가로 감소한다. 계약 배포 트랜잭션에서 보내는 사람의 잔고는 트랜잭션을 생성하기 위한 가스비와 배포될 컨트랙트 코드가 이더리움 가상 머신(EVM, Ethereum Virtual Machine)에서 실행되기 위한 연산량을 가스비로 환산한 만큼 감소한다. 배포된 컨트랙트는 새로운 컨트랙트 계정으로 만들어져 상태 트라이에 추가되며, 컨트랙트의 실행에 필요한 변수등의 메모리는 스토리지 트라이에 저장된다. 계약 호출 트랜잭션에서도 보내는 사람의 잔고가 트랜잭션을 생성하기 위한 가스비만큼 감소한다. 계약 코드의 함수를 수행하기 위해 책정된 가스비도 추가로 소모되며, 이 가스비는 코드가 복잡할수록 더 많이 소모된다. 일부 컨트랙트 코드는 계약 내부 트랜잭션을 발생시키는데, 특정 주소로 이더리움을 전송하는 트랜잭션, 새로운 계약 배포 트랜잭션, 또 다른 컨트랙트를 호출하는 트랜잭션 등 다양한 내부 트랜잭션이 존재할 수 있다.

제 3 장 전역 상태 트라이 분석

3.1 전역 상태 트라이 분석

이더리움의 상태 트라이의 분석은 Go 언어로 작성된 이더리움 클라이언트인 Geth(Go-ethereum) 클라이언트를 이용하였다. Geth는 거대한 상태 트라이 노드 데이터를 효율적으로 저장하기 위하여, 해당 블록에서 변경된 노드만 새로 저장하고 변경되지 않은 노드들은 기존의 데이터를 사용한다. 따라서 특정 블록에서의 가장 최신 상태의 트라이만을 구하기 위해서는 직접 순회해야한다. 알고리즘 1은 상태 트라이의 순회를 위한 알고리즘인 Inspect-StateTrie 함수를 나타낸 것으로, 상태 트라이의 루트 노드(Root node)에서 시작하여 모든 상태 트라이의 노드들을 순회한다. 순회하면서 만난 모든 노드의 정보를 데이터베이스에 저장하고, 자식 노드를 가지고 있는지 확인하여 모든 자식 노드에 대해 재귀적으로 실행하게된다.

Algorithm 1 State trie size inspection

```
Function Inspect-StateTrie(n, result):
    update-Result(result)
    if n is Extension Node then
        Inspect-StateTrie(child of n, result)
    else if n is Branch Node then
        for node in children of n do
            Inspect-StateTrie(node, result)
    else if n is Leaf Node then
        if n is CA then
            Inspect-StorageTrie
```

모든 블록에서 해당 블록의 상태 트라이를 저장하는 것은 공통된 중간 구성 노드들이 많기 비효율적이다. 이더리움은 새 블록의 상태 트라이를 저장할 때 새로

변화한 노드들만을 저장하고 변하지 않은 노드는 이전 블록의 노드를 사용한다. 따라서 특정 블록에서의 전체 상태 트라이 데이터를 구하기 위해서는 직접 모든 노드를 traverse해야한다는 문제점이 존재한다. 일반적인 1 스레드 순회에서 Ryzen 9 5950X CPU를 이용한 naïve 순회는 10일 이상의 시간이 소요되는 문제가 있다. 이를 해결하기 위해 기존의 트라이 분석 결과를 바탕으로 최적화된 병렬처리를 활용해 순회했다. 기존의 트라이 분석 결과에서 얕은 깊이의 트라이 노드들은 모두 자식 노드를 최대로 가지는 브랜치 노드라는 점에서 착안하여, 깊은 트라이 노드들에서 최적화를 수행하여 각 스레드의 대기시간을 최소화하고 disk I/O를 효율적으로 사용할 수 있도록 병렬처리를 구현했다. 특히 Geth클라이언트를 작성한 Golang은 더 가벼운 병렬처리를 위하여 경량 논리적 스레드인 goroutine을 활용하여 효과적으로 병렬처리 동작을 수행할 수 있도록 하였다. 트라이 순회를 위한 trieinspector(500 lines)와 데이터베이스에 저장되는 트라이 노드 추적을 위한 코드(150 lines), 그리고 stand-alone으로 사용하기 위한 geth 명령어를 추가하여 구현한 결과 1200만 블록에서도 3일 이내로 순회를 완료할 수 있도록 성능을 향상시켰다.

본 분석도구를 활용하여 특정 블록에서의 전역 상태 트라이의 특성을 분석한 결과를 얻을 수 있다. 상태 트라이 전체의 크기, 각 구성 노드별 개수, 비율과 크기를 통해 기본적인 특성을 파악한다. 모든 리프노드를 확인하여 EOA와 CA의 비율을 각각 구할 수 있으며, 리프 노드의 깊이 통계와 CA들에 대한 통계도 구할 수 있다. 스토리지 트라이에 대해서도 상태 트라이와 마찬가지로 기본적인 구성 노드들에 대한 통계량을 구할 수 있으며, 동일한 코드에 대해서 여러 스토리지 트라이가 존재할 때, 그 트라이들의 특성에 대한 비교도 가능하다.

제 4 장 분석 결과

4.1 전역 상태 트라이 크기

4.1.1 상태 트라이

표 4.1은 이더리움의 1050만 블록에서부터 1250만 블록까지 50만 블록마다 상태 트라이의 크기와 각 노드들의 갯수를 측정한 결과이다. 시간이 지날수록 상태 트라이(State Trie)의 크기가 증가하는 것을 확인할 수 있다. 1250만 블록에서 전체 아카이브 싱크의 스토리지 크기는 7.2TB이고, 그 중에서 트라이 노드가 차지하는 크기는 6.9TB이다. 하지만 측정 결과 확인한 1250만 블록에서의 전체 상태 트라이의 크기는 29.9GB로 전체 스토리지에서 0.4%를 차지함을 확인할 수 있다.

		State Trie	Branch Nodes		Extension Nodes		Leaf Nodes		
Date	#Block(M)	Size(MB)	#	Size(MB)	#	Size(MB)	#EOA	#CA	Size(MB)
20-08-02	10.5	21,556	30,790,538	4,580	94,829,014	10,099	78,223,719	15,781,550	6,876
20-10-07	11	23,522	33,235,655	4,983	103,595,341	11,030	85,111,974	17,559,791	7,509
20-12-23	11.5	25,734	36,087,286	5,442	113,495,264	12,074	93,010,798	19,436,719	8,217
21-03-09	12	27,701	38,683,523	5,850	122,223,205	13,002	103,148,830	17,908,152	8,848
21-12-21	12.5	30,669	42,791,537	6,479	135,464,864	14,396	115,684,761	18,425,025	9,793

표 4.1: 상태 트라이 분석 결과

각 블록에서의 상태 트라이의 구성 노드의 비율을 비교하면 그림 4.1과 같다. 블록이 진행되어 상태 트라이의 크기가 증가하더라도 구성 노드들의 비율은 브랜치 노드가 13.7%, 익스텐션 노드가 43.3%, 그리고 리프노드가 42.9%로 거의 비슷하게 유지되는 것을 확인할 수 있다.

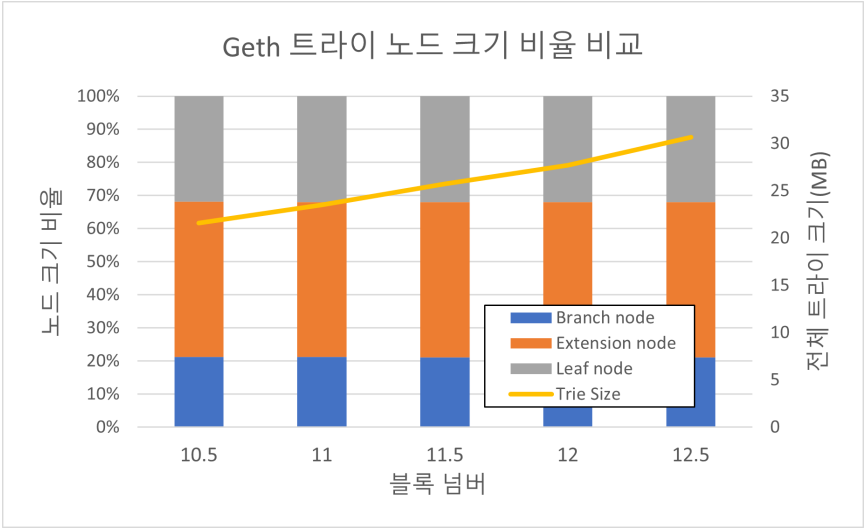

그림 4.1: Geth 트라이 노드 크기 비율 비교

4.1.2 스토리지 트라이

이더리움 1100만번째 블록에서의 스토리지 트라이에 대해 측정했다. 전체 컨트랙트 계정 1750만개 중에서 스토리지 트라이를 가지는 계정은 70.3%인 1235만개 였고, 이 계정들의 스토리지 트라이 크기 총합은 38,415MB, 평균 스토리지 트라이의 크기는 3,109B이다. 전체 스토리지 트라이의 구성 노드 비율은 브랜치 노드가 14.5%, 익스텐션 노드가 42.9%, 그리고 리프 노드가 42.5%로 상태 트라이의 구성 비율과 약간의 차이를 보였다. 스토리지 트라이의 크기가 큰 순서대로 500개의 컨트랙트를 뽑아봤을 때, 이 컨트랙트들의 스토리지 트라이가 전체 스토리지 트라이 저장공간의 60.8%를 차지하고 있었다. 가장 많은 저장용량을 사용하는 스토리지 트라이[1]의 크기는 1,967MB이며, 이 스토리지 트라이의 평균 리프 노드 깊이는 8.72였다.

[1] 0xbb91d90516ce901742bfd5d73e4bf99b510776ef941e1f86ea8a1fd7d0d2e2c5

4.2 리프 노드 깊이

이더리움의 1250만번째 블록에서의 리프 노드의 깊이를 측정한 결과는 그림 4.2와 같다. 리프 노드의 평균 깊이는 9.42이고 주로 9와 10의 깊이를 가지고 있음을 알 수 있으며, 최소 깊이는 7이고 최대 깊이는 14이다. 다시 말해, 똑같이 계정의 정보를 얻기 위한 동작이더라도, 리프 노드가 존재하는 상태 트라이의 위치에 따라 데이터베이스에 접근해야하는 횟수가 최대 2배 차이가 날 수도 있다. 하지만 이러한 디스크 I/O적 관점에 대해 아직 Geth는 최적화가 되어있지 않으며, 이를 이용한 더스트 공격을 5.2에서 제안한다.

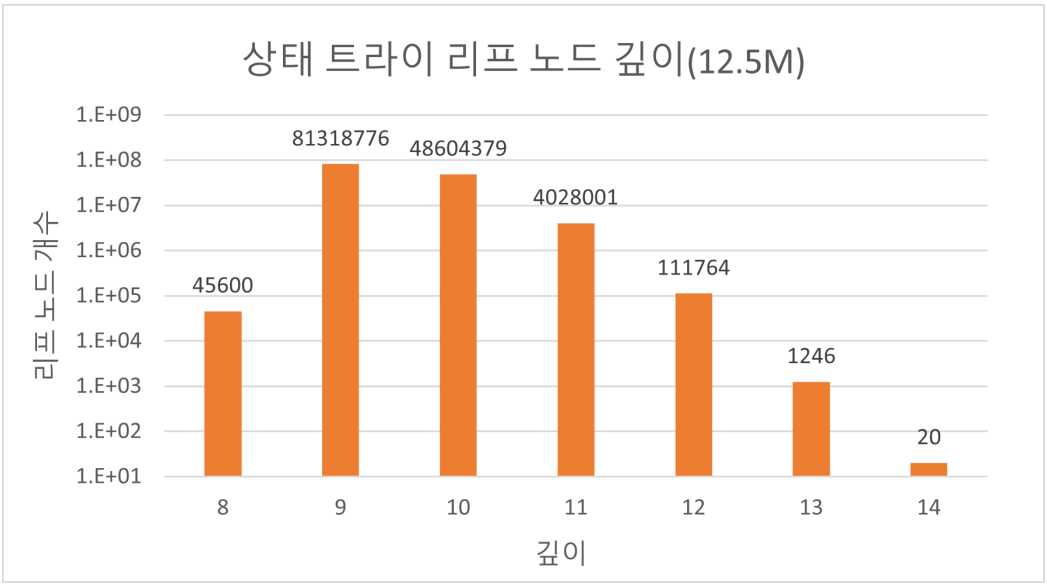

그림 4.2: 1100만 블록에서의 리프 노드 깊이 측정 결과

같은 방법으로 블록 넘버를 변화시켜가며 측정한 결과는 그림 4.3과 같다. 블록이 진행될수록 평균 리프 노드 깊이가 균일하게 증가하는 것을 확인할 수 있는데, 현재 이더리움이 사용하고 있는 머클 패트리시아 트라이가 계정 주소를 만들 때 해시값을 사용하기 때문에 균일하게 트라이의 크기가 커진다고 이해할 수 있다.

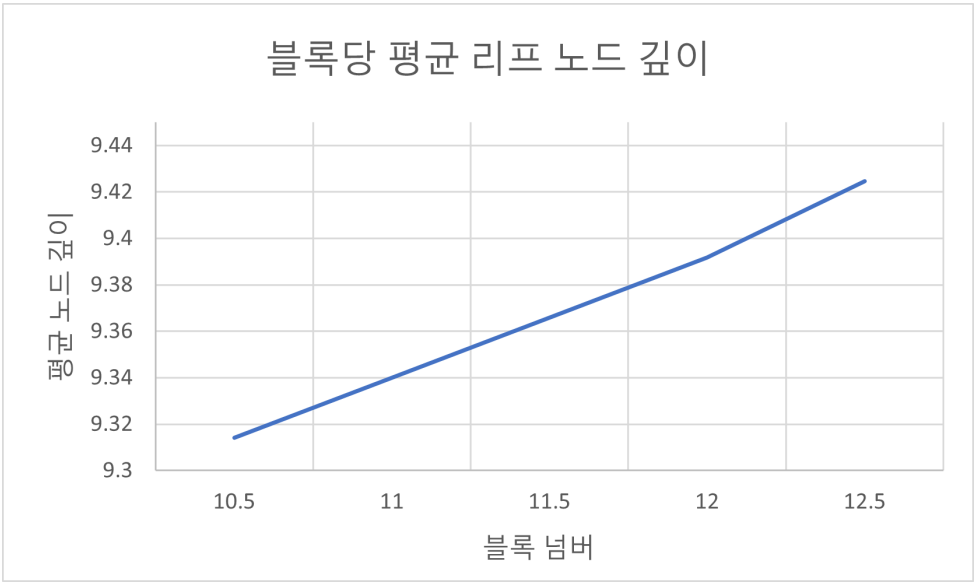

그림 4.3: 블록당 평균 리프 노드 깊이 측정

제 5 장 사용 사례

5.1 이더리움 상태 트라이 경량화 기법 성능 평가

이더리움의 상태 트라이는 블록이 진행할수록 급격하게 증가하여, 현재 아카이브 싱크를 유지하기 위해서는 10TB이상의 저장공간을 요구하고 있다. 따라서 이를 해결하기 위한 선행연구들이 진행되고 있으며, 이러한 최적화 성능 기법들의 평가를 본 논문의 트라이 측정 도구를 이용하여 수행할 수 있을 것으로 기대한다. 이더리움 이외에도 머클 패트리시아 트라이를 이용하여 상태 트라이를 구현한 블록체인에서는 모두 활용할 수 있으며 본 측정 도구를 이용하여 상태 트라이 경량화 기법의 성능을 정량적으로 평가 및 비교분석할 수 있다.

Ethane[11]은 이더리움의 계정 주소가 임의의 해시값으로 정해지기 때문에 비효율적으로 트라이의 중간 노드들이 생긴다는 점에서 착안하여, 계정 주소를 0부터 1씩 증가시켜가면서 할당하는 최적화 방법을 제시했다. 본 측정도구를 이용하여 상태 트라이 저장용량을 비교한 결과는 그림5.1과 같다. Ethane과 Geth에서 동일하게 매 블록마다 임의의 주소로 이더리움을 보내는 트랜잭션을 200개씩 발생시켜 그 변화를 측정했다. 다양한 크기의 상태 트라이에서 모두 Ethane이 Geth보다 균일하게 약 20%의 크기가 감소한 모습을 확인할 수 있다.

5.2 더스트 공격

더스트 공격(Dust Attack)[5]은 블록체인에서 임의의 주소로 아주 작은 양의 잔고를 보내는 공격이다. 이더리움에서 아직 존재하지않는 주소로 잔고를 보내면 새로운 계정이 생성된다는 것을 악용한 공격자는 전체 계정의 숫자를 폭발적으로 증가시켜 이더리움 상태 데이터의 저장용량을 증가시킬 수 있다. Evm의 opcode중 계정을 생

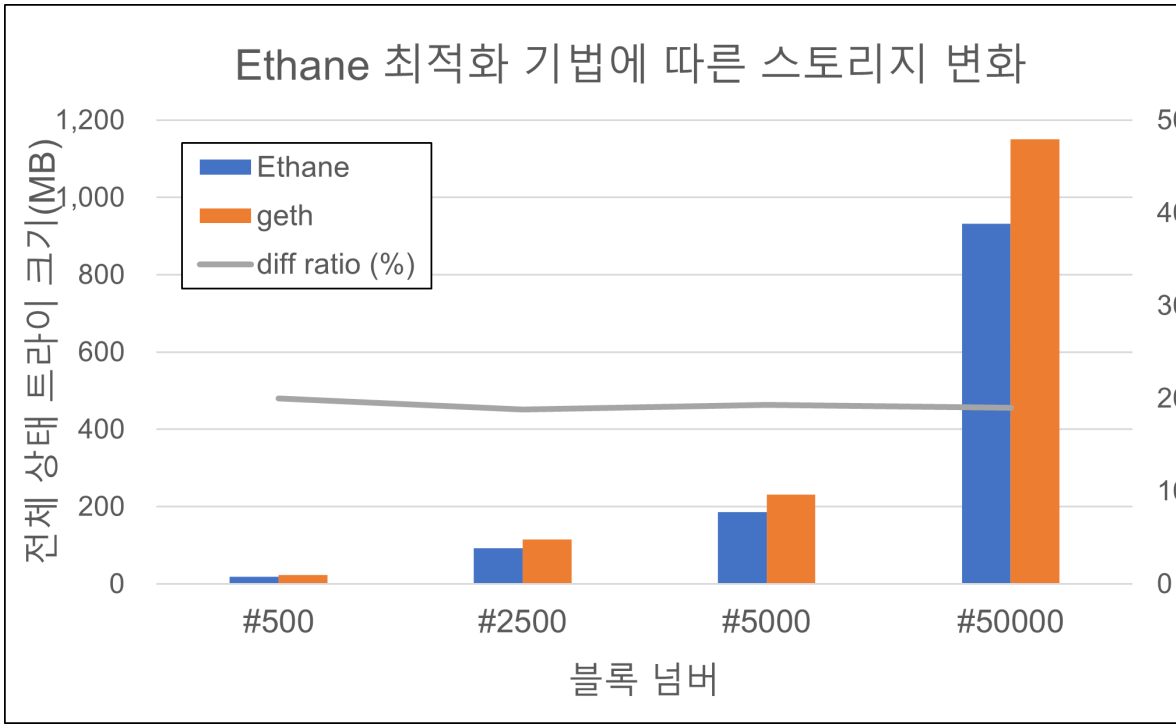

그림 5.1: Ethane 최적화 기법에 따른 스토리지 변화

성하는 CREATE 명령어가 스토리지의 변경량과 상관없이 항상 고정된 32000 gas fee를 요구한다. 공격자는 본 논문의 상태 트라이 분석 도구를 이용하여 더 효율적으로 전체 스토리지를 증가시키는 더스트 공격이 가능해진다.

상태 트라이의 리프노드들은 모두 트라이에서 다른 깊이를 가진 상태로 위치한다. 즉, 똑같이 어떤 계정의 상태에 접근하려고 하더라도 데이터베이스에 접근 횟수가 다르다. 공격자는 상태 트라이 분석 도구를 이용하여 실시간으로 가장 스토리지 변화를 많이 만들어내는, 또는 가장 스토리지를 많이 증가시키는 계정을 생성시킬 수 있다. 이러한 더스트 공격은 상태 트라이 분석 결과를 바탕으로 opcode CREATE의 gas fee 요구량을 스토리지 변화에 비례하도록 조정하여 막을 수 있을 것으로 기대된다.

제 6 장 관련 연구

이전까지의 이더리움 데이터 분석 연구는 트랜잭션 데이터의 데이터베이스화와 그 데이터베이스의 응용을 목적으로 진행되었다.

6.1 이더리움 데이터 분석 프레임워크

이전의 이더리움 데이터 분석 프레임워크는 이더리움의 복잡한 내부 메커니즘을 이해하고 계약 내부 트랜잭션 데이터를 기록하는 것에 그쳤다.[2, 6] 특히 이더리움의 데이터 분석 프레임워크로 가장 많이 사용되고있는 Etherscan은 트랜잭션으로 발생한 계정의 상태변화에 초점을 맞추고 있을 뿐, 상태 트라이 자체에 대한 분석은 이루어지지 않는다. 따라서 상태 트라이를 구현하는 머클 패트리시아 트라이의 경량화 및 최적화 연구에서 본 측정 도구를 활용하여 기여할 수 있을 것으로 기대한다.

6.2 스마트 계약 실행환경

이더리움의 스마트 계약은 Geth 클라이언트 내부의 가상 머신 위에서 동작을 수행한다. 많은 스마트 계약이 가지고 있는 보안 취약점을 찾기 위해 fuzzing을 이용하는 연구가 많은데, 이를 위해 standalone으로 동작하는 스마트 계약 실행환경을 구현하고 데이터 분석을 수행하는 프레임워크 연구들이 진행되었다.[7] 하지만 해당 연구는 스마트 계약의 빠른 수행 및 트랜잭션 수행 전후의 상태를 비교하는 것에 목적을 두고있으며, 스토리지 관점에서의 분석은 수행하지 않는다.

6.3 이더리움 경량화

이더리움 아카이브 싱크의 거대한 상태 트라이는 새로운 사용자의 노드 참여를 어렵게 만드는 진입장벽이다. 아카이브 싱크의 거대한 상태 트라이 중에서 실제로 최근에 트랜재션에 참여한 계정의 비율이 매우 낮다는 점에서 착안하여 active한 계정만으로 이루어진 작은 상태 트라이를 만드는 연구인 Ethanos가 수행되었다.[9]

제 7 장 결론 및 향후 연구

이더리움의 폭발적인 사용자 증가는 상태 데이터 저장 용량의 증가로 이어졌다. 이러한 저장 용량의 폭발적인 증가는 블록체인의 탈중앙화에 악영향을 주는 큰 문제로 제기되었다. 본 논문에서는 이더리움의 블록체인을 위한 효율적인 데이터베이스 설계를 위하여 상태 트라이 데이터 분석도구를 제안한다. 본 논문의 분석도구를 이용하여 이더리움의 상태 트라이를 구현하고있는 머클 패트리시아 트라이의 특성에 대해 분석할 수 있었다. 나아가 상태 트라이 경량화 및 최적화 기법에 대한 후속연구들의 성능을 평가하고 비교하는데 활용할 수 있을 것으로 기대된다. 그리고 상태 트라이에 대한 분석 내용을 바탕으로 효과적인 더스트 공격을 제안한다.

상태 트라이 최적화 기법의 성능을 평가하기 위한 지표로 매 블록마다 변화한 트라이 노드의 정보를 사용할 수 있다. 하지만 모든 블록에서 전체 상태 트라이 순회를 하는것은 매우 오랜 시간이 소요된다. 따라서 변화한 트라이 노드의 정보를 효율적으로 얻기 위해 디스크에 flush 되는 상태 트라이 노드의 정보를 확인할 필요가 있다. 이와 같은 정보를 합쳐 상태 트라이 뿐만 아니라 전반적인 스토리지 관련 정보를 제공할 수 있는 종합 분석 프레임워크를 개발을 향후 연구방향으로 하고자한다.

참고 문헌

[1] G. Wood, ”Ethereum: A secure decentralised generalised transaction ledger,” Ethereum Project Yellow Paper, 2014

[2] Etherscan.io. https://etherscan.io/

[3] B. Bünz, L. Kiffer, L. Luu and M. Zamani, ”FlyClient: Super-Light Clients for Cryptocurrencies,” 2020 IEEE Symposium on Security and Privacy (SP), 2020, pp. 928-946, doi: 10.1109/SP40000.2020.00049.

[4] Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame, and Srdjan Capkun. 2019. BITE: bitcoin lightweight client privacy using trusted execution. In Proceedings of the 28th USENIX Conference on Security Symposium (SEC'19). USENIX Association, USA, 783–800.

[5] Dusting Attack Announcement of Samourai Wallet https://twitter.com/SamouraiWallet/status/1055345822076936192

[6] T. Chen et al., ”DataEther: Data Exploration Framework For Ethereum,” 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), 2019, pp. 1369-1380, doi: 10.1109/ICDCS.2019.00137.

[7] KIM, Yeonsoo, et al. An Off-The-Chain Execution Environment for Scalable Testing and Profiling of Smart Contracts. In:2021 USENIX Annual Technical Conference (USENIX ATC 21). 2021. p. 565-579.

[8] Ask about Geth: Snapshot acceleration, https://blog.ethereum.org/2020/07/17/ask-about-geth-snapshot-acceleration/

[9] Jae-Yun Kim, Junmo Lee, Yeonjae Koo, Sanghyeon Park, and Soo-Mook Moon. 2021. Ethanos: efficient bootstrapping for full nodes on account-based blockchain. In Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys '21). Association for Computing Machinery, New York, NY, USA, 99–113. https://doi.org/10.1145/3447786.3456231

[10] Etherscan.io. Ethereum Full Node Sync (Archive) Chart, https://etherscan.io/chartsync/chainarchive

[11] Ethane, https://github.com/eth4ne/go-ethereum

ABSTRACT

Ethereum is the second largest cryptocurrency and the most popular smart contract blockchain. The volume of trading ethereum has increased for years as the interests of cryptocurrency and blockchain grow. Since ethereum is state-based blockchain which stores every state of users, the amount of accounts and transaction data was exploded. Exploded data delayed network synchronization, decreased performance of transaction execution, and increased database storage. More than 95% of storage is world state trie which contains state trie and storage trie. To overcome these limitations, some lightening and optimzing techniques for state trie are proposed, and it is necessary to develop a state trie analysis tool.

In this paper, we developed a tool for anayzing ethereum world state trie and inspected properties. This tool provides more elaborate storage related analysis result than Etherscan; The size of world state trie of certain block, the ratio and depth of configured nodes, and statistics of the storage tries. It is expected that we will contribute to effiecient state trie optimization technique develeopments and prevent effective dust attack based on state trie analysis.

주요어: Ethereum, Blockchain, Merkle Patricia Trie, State Trie

학번: 2020-23168