

메가이더(MegaETH)

해당 번역본은 이용자가 디지털자산 관련 정보를 편리하게 확인할 수 있도록 단순 참고용으로 번역된 것이며, 투자 권유를 목적으로 하지 않습니다.

그 내용이 정확하지 않을 수 있으므로 정확한 정보 습득을 위해서는 원문을 참고하시거나 원문 작성 주체 측에 문의하시기 바랍니다.

두나무는 해당 번역본 내용의 진실성 및 정확성을 보장하지 않고, 제공된 정보에 의한 투자 결과에 대하여 어떠한 책임도 지지 않으며, 투자 손실은 투자자에게 귀속됩니다.

MegaETH: 최초의 실시간 블록체인 공개

MegaLabs

MegaETH는 EVM 호환 블록체인으로, Web2 수준의 실시간 성능을 암호화폐 세계에 최초로 도입합니다. 우리의 목표는 성능을 하드웨어 한계까지 끌어올려, 블록체인과 기존 클라우드 컴퓨팅 서버 간의 격차를 해소하는 것입니다. MegaETH는 높은 트랜잭션 처리량, 풍부한 연산 능력, 그리고 무엇보다도 과부하 상태에서도 밀리초 단위의 응답 속도라는 독보적인 특징을 제공합니다. MegaETH를 통해 개발자들은 가장 높은 요구 조건의 애플리케이션도 제약 없이 구축하고 결합할 수 있습니다.

왜 또 다른 블록체인이 필요한가?

블록체인 프레임워크의 발전은 새로운 체인을 만드는 장벽을 상당히 낮췄습니다. 그 결과 최근에는 수많은 신규 체인이 등장하고 있습니다. 예를 들어, [L2Beat](#)는 레이어 2 생태계 내에서 120개 이상의 프로젝트를 문서화하고 있습니다. 그러나 단순히 더 많은 체인을 만든다고 해서 블록체인 확장성 문제를 해결할 수는 없습니다. 각각의 체인은 여전히 그 위에 올라가는 dApp에 심각한 제약을 가하기 때문입니다. 예를 들어, 아래 표는 오늘날 주요 EVM 체인의 초당 가스 처리량과 블록 생성 시간을 보여줍니다.

Select EVM Chains	Gas Per Second	Target Gas Per Block (Supply)	Block Time
opBNB	100.0 mg/s	100M	1.0s
BSC	46.6 mg/s	140M	3.0s
Polygon	7.5 mg/s	15M	2.0s
Avalanche C-Chain	7.5 mg/s	15M	2.0s
Arbitrum One	7.0 mg/s	1.75M	0.25s
Base	5.0 mg/s	15M	2.0s
Optimism Mainnet	2.5 mg/s	5M	2.0s
Conduit*	2.5 mg/s	5M	2.0s
Ethereum L1	1.25 mg/s	15M	12.0s
*Representative of most Conduit chains.			

이 표는 기존 EVM 체인들이 여러 측면에서 심각한 제약을 받고 있음을 명확히 보여줍니다. 첫째, 모든 체인이 낮은 트랜잭션 처리량을 보입니다. 예를 들어, opBNB는 동급 체인 중 예외적으로 높은 100 MGas/s의 가스 처리 속도를 자랑하지만, 이는 여전히 최신 Web2 서버의 처리 능력과 비교하면 미미한 수준입니다. 참고로, 100 MGas/s는 초당 약 650건의 Uniswap 스왑이나 3,700건의 ERC-20 전송에 해당합니다. 반면, 최신 데이터베이스 서버는 이미 TPC-C 벤치마크에서 초당 100만 트랜잭션을 초과하고 있습니다. 둘째, 정교한 애플리케이션을 온체인으로 구현하기 어렵습니다. 연산 능력이 부족하기 때문입니다. 예를 들어, n 번째 피보나치 수를 계산하는 단순한 EVM 컨트랙트는 $\$n=10^8\$$ 일 때 약 55억 가스를 소비하며, 이는 opBNB 체인 전체를 100 MGas/s 기준으로 55초 동안 점유하게 됩니다. 반면, 동일한 작업을 C 언어로 작성한 프로그램은 단일 CPU 코어에서 30밀리초 만에 완료할 수 있어, 이미 1833배 빠릅니다! 여기에 멀티코어 처리를 활용하여 100배 이상의 연산 능력을 추가로 확보한다면, 실현 가능한 가능성은 더욱 확장됩니다. 마지막으로, 높은 갱신 주기나 빠른 피드백 루프가 요구되는 애플리케이션은 긴 블록 시간으로 인해 구현이 불가능합니다. Arbitrum One을 제외한 모든 체인은 상태를 1초 간격 이상으로 업데이트합니다. 그러나 완전 온체인 dApp 중에서도 자율 세계(autonomous world)와 같이 실시간 전투나 물리 시뮬레이션을 필요로 하는 애플리케이션은 100밀리초 미만의 블록 간격(높은 틱레이트)이 요구됩니다. 또한, 온체인 고빈도 거래는 주문이 10밀리초 이내에 제출되거나 취소될 수 없다면 불가능합니다. 다행히도, 이러한 제약은 EVM 체인에게 극복 불가능한 것이 아닙니다. 우리의 기술적 진보를 통해 이제는 이러한 잠재력을 실현할 수 있는 실시간 블록체인을 구축할 수 있는 시점입니다. 보다 형식적으로 말하면, 실시간 블록체인이란 트랜잭션이 도착하자마자 처리하고 그 결과를 실시간으로 게시할 수 있는 블록체인을 말합니다. 또한, 실시간 경험을 사용자 수요 피크 시점에도 유지하기 위해서는 높은 트랜잭션 처리량과 충분한 연산 능력을 갖추어야 합니다.

노드 특수화: 성능 중심 설계를 위한 패러다임 전환

블록체인 확장성은 수년간 활발히 연구되어 온 분야입니다. 그렇다면 어떻게 현존 기술을 수 배 이상 뛰어넘는 성능 향상을 갑자기 달성할 수 있을까요? 그 해답은 놀랍도록 간단합니다. 보안성과 검열 저항성을 Ethereum 및 EigenDA 같은 기반 레이어에 위임함으로써, 공격적인 성능 최적화를 실현할 수 있는 광범위한 설계 공간을 탐색할 수 있게 됩니다.

이 개념을 더 잘 이해하기 위해, 먼저 오늘날 블록체인이 어떻게 작동하는지를 살펴보겠습니다. 모든 블록체인은 합의(consensus)와 실행(execution)이라는 두 가지 근본적인 구성 요소로 이루어져 있습니다. 합의는 사용자 트랜잭션의 순서를 결정하며, 실행은 이 순서에 따라 트랜잭션을 처리하여 블록체인 상태를 갱신합니다.

대부분의 L1 블록체인에서는 각 노드가 특수화 없이 동일한 작업을 수행합니다. 모든 노드는 분산 프로토콜에 참여하여 합의를 이루고, 이후 각 트랜잭션을 로컬에서 실행합니다. 이 구성은 성능과 탈중앙화 사이의 근본적인 트레이드오프를 야기합니다. 모든 L1은 일반 사용자가 노드를 운영할 수 있는 수준에서 하드웨어 요구사항을 어디까지 높일 수 있는지 결정해야 하며, 이로 인해 블록체인의 핵심 속성인 보안성과 검열 저항성이 훼손되지 않아야 합니다.

일반 사용자가 노드를 실행할 수 있어야 블록체인의 탈중앙성이 보장됩니다.
– [Vitalik Buterin](#)

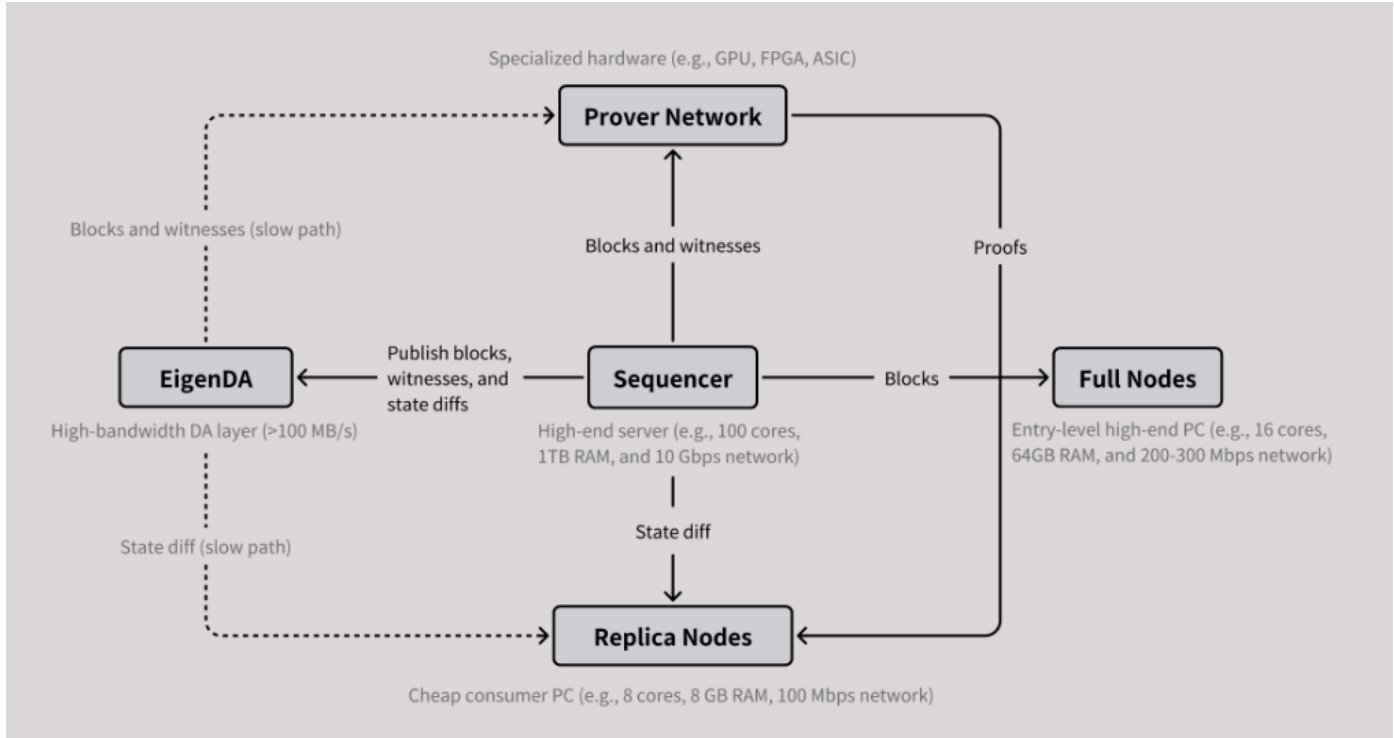
그러나 풀 노드를 위한 허용 가능한 하드웨어 요구사항이 무엇인지에 대한 명확한 답은 없습니다. 각 L1은 성능과 탈중앙화 사이의 스펙트럼에서 서로 다른 위치를 취합니다. 예를 들어, 다음 표는 여러 L1 체인의 권장 하드웨어 사양을 보여줍니다.

	CPU	메모리	네트워크	스토리지
Ethereum	2코어	4–8 GB	25 Mbps	SSD
Solana	12코어	256 GB	1–10 Gbps	SSD
Aptos	32코어	64 GB	1 Gbps	SSD

이에 반해, L2는 블록체인 설계에 있어 패러다임의 전환을 이끌며, 노드 간 일률적인 하드웨어 요구사항을 제거합니다. 이는 L2 블록체인이 본질적으로 이질적인 구조이기 때문입니다. 각기 다른 노드가 특정 작업을 보다 효율적으로 수행하도록 특화되어 있기 때문입니다. 예를 들어, 일반적인 L2는 트랜잭션 순서를 결정하는 특수한 시퀀서(sequencer) 노드를 사용합니다. 또한, ZK-Rollup의 프루버(prover) 노드는 종종 GPU 및 FPGA와 같은 특수 가속기를 활용하여 증명 생성 비용을 줄입니다. MegaETH는 노드 특수화를 한층 더 발전시켜, 트랜잭션 실행을 완전히 생략하는 새로운 유형의 노드인 레플리카 노드(replica node)를 도입합니다.

이로써 MegaETH에는 시퀀서, 프루버, 풀 노드, 레플리카 노드라는 네 가지 주요 역할이 존재합니다. 시퀀서 노드는 사용자 트랜잭션의 순서를 정하고 이를 실행합니다. 그러나 MegaETH는 항상 하나의 활성 시퀀서만을 운영함으로써, 정상 상태 실행 시 합의의 오버헤드를 제거합니다. 레플리카 노드는 시퀀서로부터 P2P 네트워크를 통해 상태 변화 정보(state diff)를 받아, 이를 직접 적용하여 로컬 상태를 갱신합니다. 이들은 트랜잭션을 다시 실행하지 않으며, 대신 프루버가 제공하는 증명을 통해 간접적으로 블록을 검증합니다. 풀 노드는 기존 방식대로 작동합니다. 이들은 모든 트랜잭션을 재실행하여 블록을 검증합니다. 이는 브리지 운영자나 마켓 메이커 같은 파워 유저가 빠른 파이널리티를 확보하기 위해 필수적이며, 시퀀서의 속도를 따라가기 위한 높은 하드웨어 요구사항을 수반합니다. 마지막으로, 프루버는 stateless validation scheme을 활용하여 비동기적이고 순서에 관계없이 블록을 검증합니다.

아래 그림은 MegaETH의 기본 아키텍처와 주요 구성 요소 간의 상호작용을 나타냅니다. 참고로 EigenDA는 EigenLayer 위에 구축된 외부 구성 요소입니다.



노드 특수화의 주요 장점은 각 노드 유형별로 하드웨어 요구사항을 독립적으로 설정할 수 있다는 점입니다. 예를 들어, 시퀀서 노드는 실행의 핵심을 담당하므로 고성능 서버에서 실행하는 것이 바람직합니다. 반면, 레플리카 노드는 증명 검증만 수행하므로, 비교적 낮은 사양의 하드웨어로도 충분합니다. 또한, 풀 노드는 여전히 실행을 수행하지만, 시퀀서가 생성한 보조 정보를 활용하여 보다 효율적으로 트랜잭션을 재실행할 수 있습니다. 이와 같은 구조의 함의는 큼니다. Vitalik의 Endgame 게시글에서 언급되었듯, 노드 특수화는 블록 생성이 중앙집중화되더라도 블록 검증은 신뢰 불필요하고 고도로 탈중앙화된 상태를 유지할 수 있게 합니다.

다음 표는 MegaETH 내 각 노드 유형의 예상 하드웨어 요구사항을 정리한 것입니다:

노드 유형	CPU	메모리	네트워크	스토리지	예시 VM (시간당 비용)
시퀀서	100코어	1-4 TB	10 Gbps	SSD	AWS r6a.48xlarge (\$10)
프루버 (OP)	1코어	0.5 GB	느림	없음	AWS t4g.nano (\$0.004)
레플리카 노드	4-8코어	16 GB	100 Mbps	SSD	AWS lm4gn.xlarge (\$0.4)
풀 노드	16코어	64 GB	200 Mbps	SSD	AWS lm4gn.4xlarge (\$1.6)

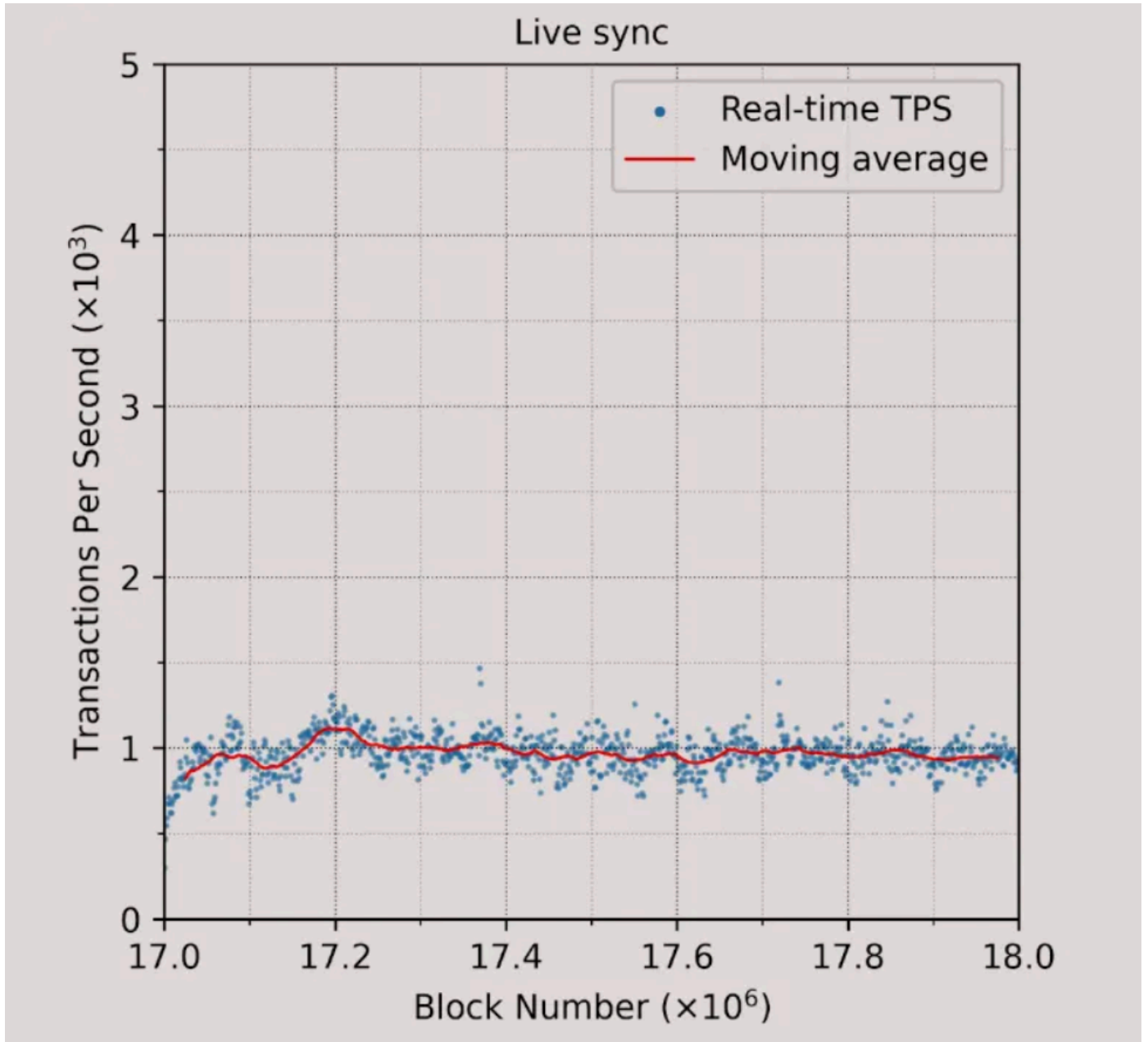
ZK 프루버 노드는 사용되는 증명 스택에 따라 하드웨어 요구사항이 크게 달라지므로 본 표에서는 생략합니다. 다양한 VM 인스턴스의 시간당 비용은 [instance-pricing.com](#)에서 인용되었습니다. 특히 주목할 점은, 노드 특수화를 통해 평균 Solana 검증자보다 20배 비싸고 5-10배 성능이 높은 시퀀서를 운용할 수 있으면서도, 풀 노드의 비용은 Ethereum L1 노드 수준으로 유지할 수 있다는 것입니다.

실시간 블록체인 엔지니어링

노드 특수화 개념은 우아하면서도 강력합니다. 자연스럽게 이런 질문이 떠오릅니다: MegaETH의 비밀은 단지 강력한 중앙 집중형 시퀀서에 있는 것인가요? 답은 "아니오"입니다.

MegaETH의 가능성을 이해하기 위해 중앙화된 서버에 비유하는 것은 유용한 사고 모델일 수 있지만, 그 이면에 존재하는 연구 및 엔지니어링의 복잡성을 심각하게 과소평가하는 것입니다. 실시간 블록체인을 만드는 것은 기존 Ethereum 실행 클라이언트를 가져다가 시퀀서의 하드웨어를 강화하는 것 이상의 것을 요구합니다.

예를 들어, 우리의 [성능 실험](#)은 512GB RAM을 탑재한 강력한 서버에서도, Reth는 최근 Ethereum 블록을 대상으로 한 라이브 싱크 환경에서 초당 약 1000 트랜잭션, 즉 약 100 MGas/s 이상의 성능을 내기 어렵다는 사실을 보여줍니다. 이 실험에서는 Reth의 성능 병목이 각 블록마다 Merkle Patricia Trie (MPT)를 업데이트하는 오버헤드에 있으며, 이는 트랜잭션 실행보다 거의 10배 더 계산적으로 부담이 큰 작업입니다. 자세한 내용은 EthDenver'24에서 발표한 [Ethereum 실행 클라이언트 성능 이해](#)를 참고하십시오.



요약하자면, 노드 특수화는 성능 향상의 중요한 기회를 열어주지만, 극도로 최적화된 블록체인을 설계하고 구현하는 일은 여전히 풀리지 않은 과제입니다.

우리의 설계 철학

복잡한 컴퓨터 시스템과 마찬가지로, 블록체인에는 여러 상호작용 구성 요소에 걸쳐 다양한 병목 지점이 존재할 수 있습니다. 병목 하나만 고립적으로 해결하는 방식은 거의 항상 종단 간 성능 향상으로 이어지지 않습니다. 이는 (1) 해당 병목이 아직 가장 치명적인 제한 요소가 아니거나, (2) 병목이 다른 구성 요소로 단순히 이동하기 때문입니다. 우리는 수많은 프로젝트들이 특정 구성 요소 최적화에만 집중하는 모습을 반복적으로 보아 왔습니다. 이들은 분리된 마이크로 벤치마크에서 인상적인 속도 향상을 보여줄 수 있지만, 이러한 결과는 종종 최종 사용자에게 실질적인 성능 향상으로 이어지지 못합니다.

MegaETH에서는 연구 개발 초기부터 보다 총체적이고 원칙 중심의 접근 방식을 적용하기로 결정했습니다. 우리의 설계 철학은 다음과 같이 요약할 수 있습니다.

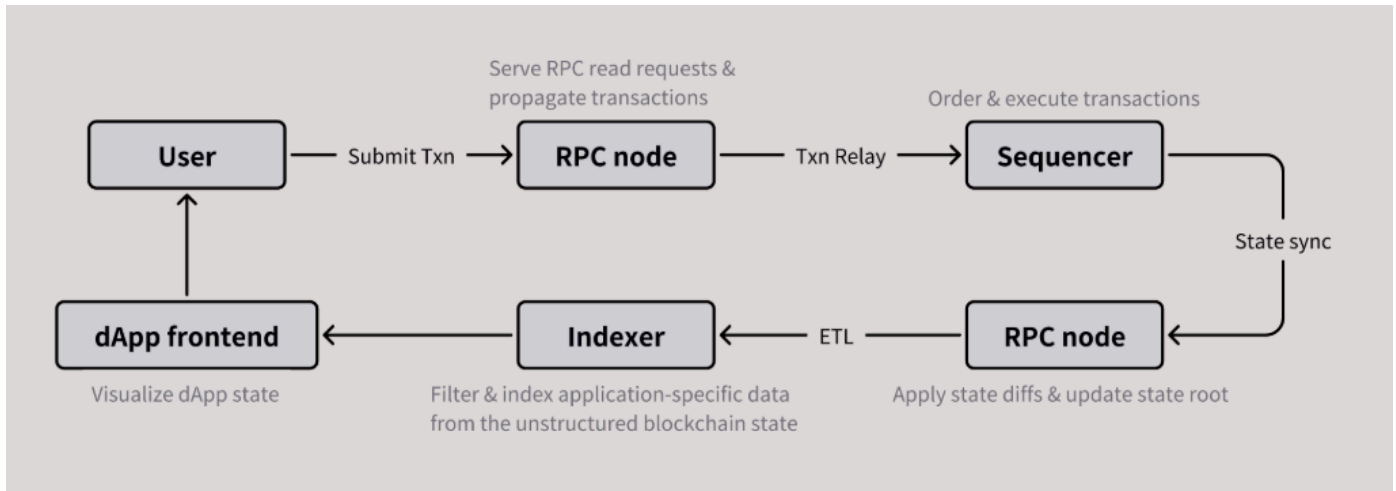
첫째, 우리는 항상 "측정하고, 그 다음 구축한다"는 원칙을 따릅니다. 즉, 기존 시스템의 진짜 문제를 파악하기 위해 심층적인 성능 측정을 먼저 수행합니다. 그 통찰을 바탕으로, 모든 문제를 동시에 해결할 수 있는 새로운 기술을 설계합니다.

둘째, 우리는 시스템을 하드웨어 한계에 근접하도록 설계하려고 노력합니다. 기존 시스템에 대해 점진적인 개선을 가하기보다는, 이론적 상한에 근접하는 클린 슬레이트 설계를 선호합니다. 우리의 목표는 암호 인프라의 성능을 더 이상 향상시킬 여지를 거의 남기지 않도록 하여, 업계가 채택을 방해하는 다른 과제에 자원을 집중할 수 있도록 하는 것입니다.

아래에서는 고성능 실시간 블록체인을 설계하고 구현하는 데 따른 주요 기술적 과제를 설명합니다.

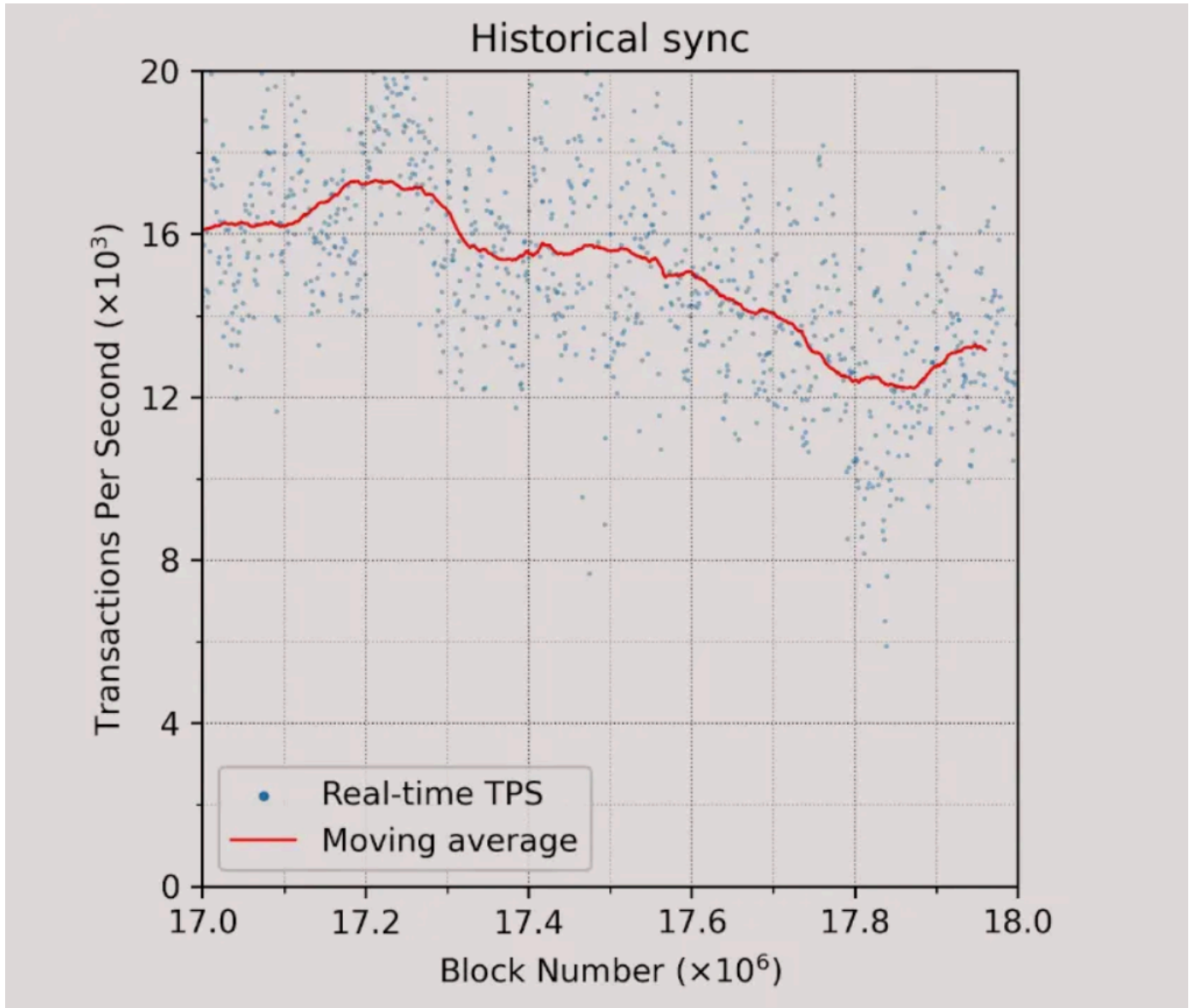
트랜잭션 실행

아래 그림은 사용자 트랜잭션이 시스템을 통해 이동하는 여정을 보여주며, 이후 설명할 기술적 과제들의 참조도표로 사용됩니다.

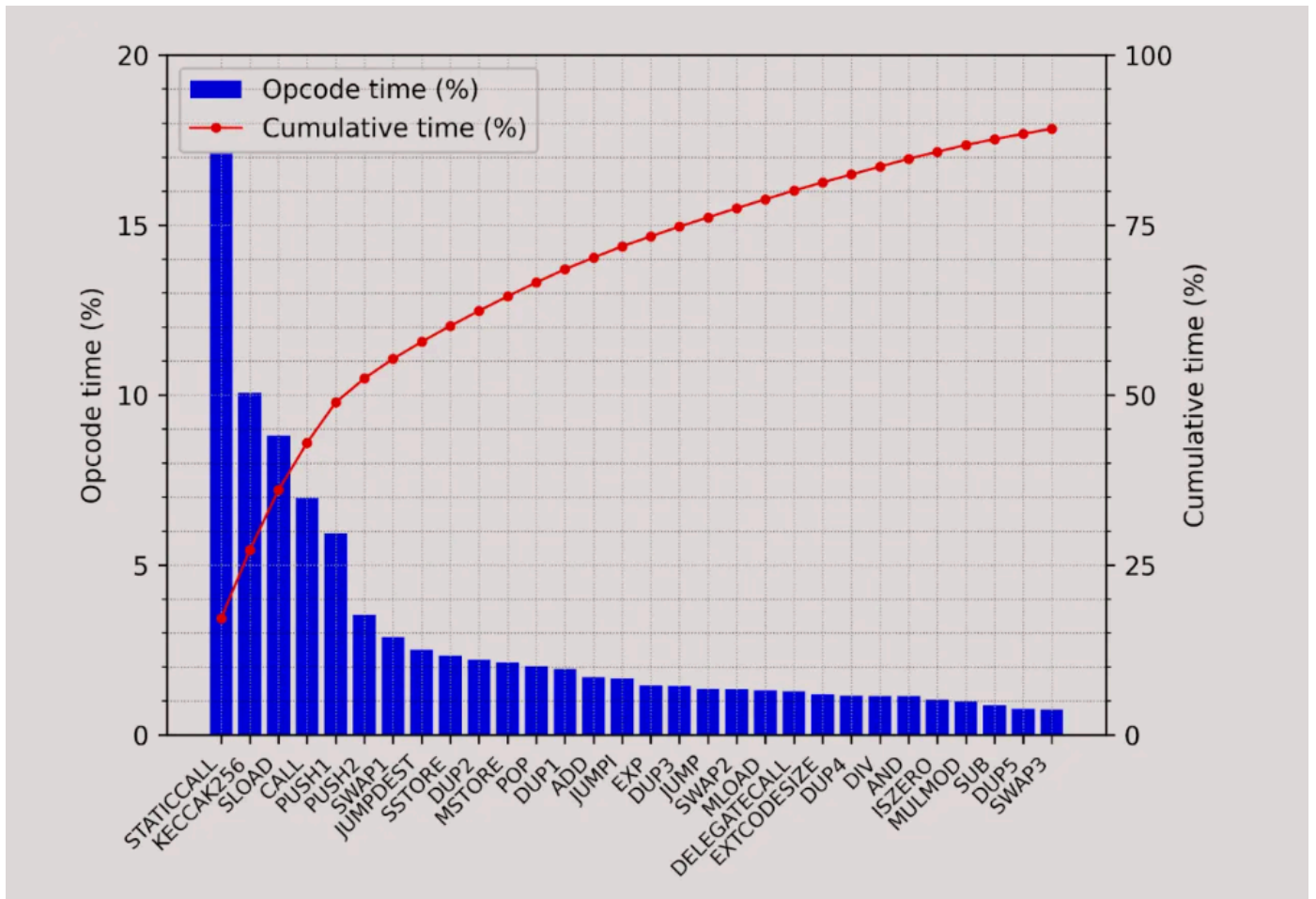


많은 사람들이 가장 익숙한 구성 요소인 시퀀서부터 시작하겠습니다. 시퀀서는 트랜잭션의 순서를 정하고 이를 실행하는 역할을 합니다. EVM은 상대적으로 낮은 실행 성능의 주범으로 자주 지목되며, 이로 인해 Ethereum L2에서는 EVM을 대체하려는 다양한 노력이 진행되어 왔습니다.

하지만 이러한 고정관념은 사실과 다릅니다. 우리의 성능 측정 결과에 따르면, revm은 과거 블록을 동기화하는 환경에서 초당 약 14,000 TPS의 성능을 이미 달성할 수 있습니다. 머신 구성은 앞서 언급한 라이브 싱크 실험과 동일합니다.

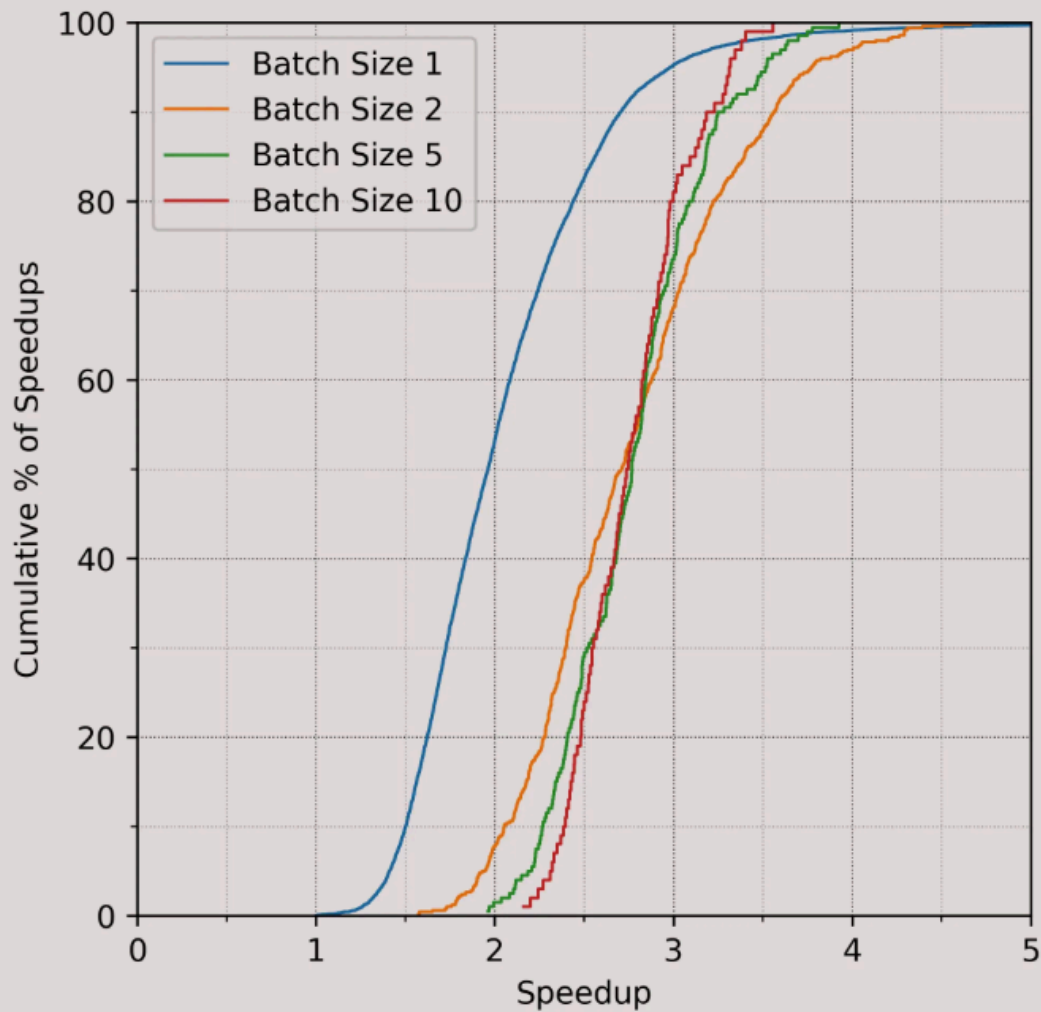


14,000 TPS는 대부분의 L2에는 충분한 수준이지만, MegaETH처럼 실시간 블록체인에는 부족합니다. 기존 EVM 구현체는 세 가지 주요 비효율성을 가집니다: 1) 높은 상태 접근 지연, 2) 병렬 실행 부재, 3) 인터프리터 오버헤드. 노드 특수화를 통해 MegaETH의 시퀀서 노드는 이더리움 전체 상태(현재 약 100GB)를 RAM에 적재할 수 있을 만큼 충분한 메모리를 갖추고 있으며, 이를 통해 SSD 읽기 지연 없이 상태 접근 속도를 대폭 향상시킬 수 있습니다. 예를 들어, 위의 과거 동기화 실험에서는 sload 연산이 런타임의 단 8.8%만을 차지했습니다. 따라서 아래에서는 나머지 두 가지 과제에 집중하겠습니다.



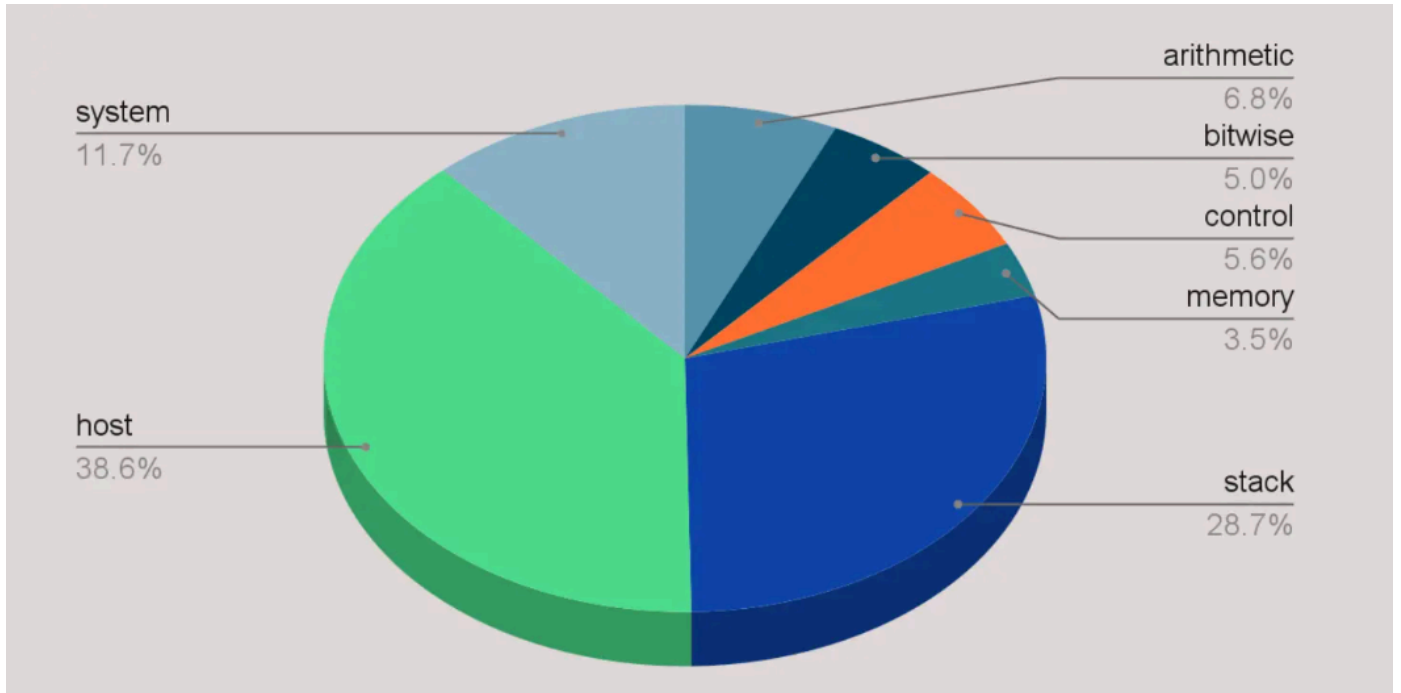
최근 병렬 EVM은 화두가 되고 있으며, 많은 팀들이 원래 MoveVM을 위해 설계된 Block-STM 알고리즘을 EVM에 포팅하는 데 집중하고 있습니다. 병렬 EVM이 원칙적으로는 해결된 문제라고 말할 수 있지만, 실제 생산 환경에서 달성할 수 있는 속도 향상에는 본질적인 한계가 있습니다. 그 이유는 워크로드에서 활용 가능한 병렬성이 제한적이기 때문입니다.

불행히도, 우리의 측정에 따르면 최근 이더리움 블록에서의 중간값 병렬성은 2 미만입니다. 심지어 여러 블록을 인위적으로 병합해 큰 배치로 처리해도 병렬성 중간값은 2.75에 불과합니다. 원시 실험 데이터를 들여다 보면, 오늘날 이더리움 워크로드에는 긴 의존성 체인이 만연해 있다는 사실을 확인할 수 있습니다. 충돌을 해결하고 병렬성을 높이기 위한 추가 기술 없이, 병렬 EVM의 이점은 제한적일 수밖에 없습니다.



앞서 피보나치 예시에서 설명한 것처럼, `revm` 같은 비교적 빠른 EVM 인터프리터조차도 네이티브 실행에 비해 1–2자릿수 느립니다. 이러한 성능 격차를 해소하기 위해, 최근에는 단일 스레드 EVM 실행을 가속화하려는 AOT/JIT 컴파일 방식에 대한 관심이 다시 높아지고 있으며, [revmc](#), [evm-mlir](#), [LL-EVM](#) 등 여러 팀이 경쟁적으로 접근하고 있습니다.

이 프로젝트들은 일부 연산 집약적 컨트랙트에 대해 유망한 결과를 보여주었지만, 실제 프로덕션 환경에서의 속도 향상은 제한적입니다. 그 이유 중 하나는 현재 대부분의 컨트랙트가 계산 집약적이지 않기 때문입니다. 예를 들어, 우리가 과거 동기화 중 각 opcode에 소비된 시간을 분석한 결과, `revm` 실행 시간의 약 50%가 `keccak256`, `sload`, `sstore` 같은 “host” 및 “system” opcode에 사용되며, 이들은 이미 Rust로 구현되어 있습니다. 결과적으로, 이 opcode들은 컴파일 최적화의 영향을 받지 않으므로, 실제 환경에서 최대 속도 향상은 2배에 불과합니다.



지금까지는 고성능 EVM 체인에 공통적인 과제들을 다루었습니다. 그러나 실시간 블록체인의 요구사항은 여기에 최소 두 가지 과제를 추가로 제시합니다. 첫째, 10밀리초마다 블록을 안정적으로 생성할 수 있어야 합니다. 둘째, 병렬 실행 엔진이 트랜잭션 우선순위를 지원해야 하며, 이는 혼잡한 시기에도 중요한 트랜잭션이 지연 없이 처리될 수 있게 합니다. 따라서 Block-STM은 일반적인 병렬 실행에는 좋은 해법이지만, 우리의 저지연 환경에는 적합하지 않습니다.

상태 동기화

상태 동기화(state sync)는 풀 노드를 시퀀서와 동일한 최신 상태로 맞추는 과정입니다. 이는 고성능 블록체인 설계에서 가장 도전적인 측면 중 하나이지만, 종종 간과되곤 합니다.

상태 동기화가 왜 어려운지 이해하기 위해, ERC-20 전송과 같은 단순한 트랜잭션을 고려해봅시다. 초당 100,000건의 ERC-20 전송을 동기화하는 데 필요한 대역폭을 손으로 계산해 보겠습니다. 각 ERC-20 전송은 세 개의 값을 수정합니다: 송신자의 잔액 (주소 20바이트 + 값 32바이트) 그리고 ERC-20 계약의 두 개 저장 슬롯 (각 64바이트, 주소 20바이트). 따라서 상태 변경(state diff)을 단순히 인코딩하면 약 200바이트가 소요됩니다. 초당 100,000건의 전송은 152.6 Mbps의 대역폭을 소모하게 되며, 이는 이미 우리의 대역폭 예산을 초과합니다. 비교하자면, 이는 177바이트의 원시 트랜잭션 데이터를 직접 전송하는 것보다도 더 비용이 많이 듭니다.

더 복잡한 트랜잭션은 더 큰 상태 변경을 유발합니다. 예를 들어, Uniswap 스왑은 세 개의 계약에서 8개의 저장 슬롯을 수정하며, 송신자의 잔액도 변경됩니다. 즉, $64B \times 8 + 20B \times 3 + 52B = 624B$ 입니다. 초당 100,000건의 스왑은 이제 476.1 Mbps의 대역폭을 소모합니다!

또한, 풀 노드가 100 Mbps의 네트워크 연결을 가진다고 해서 실제로 100% 네트워크 사용률로 동기화할 수 있는 것은 아닙니다. 그 이유는 여러 가지가 있습니다. 첫째, 인터넷 제공업체는 실제 지속 가능한 대역폭보다 과장된 수치를 광고하는 경우가 많습니다. 둘째, 동일한 노드 내의 다른 애플리케이션들과 대역폭을 공유해야 합니다. 셋째, 새로운 풀 노드가 부트스트랩되어 네트워크에 합류할 수 있도록 충분한 여유 대역폭이

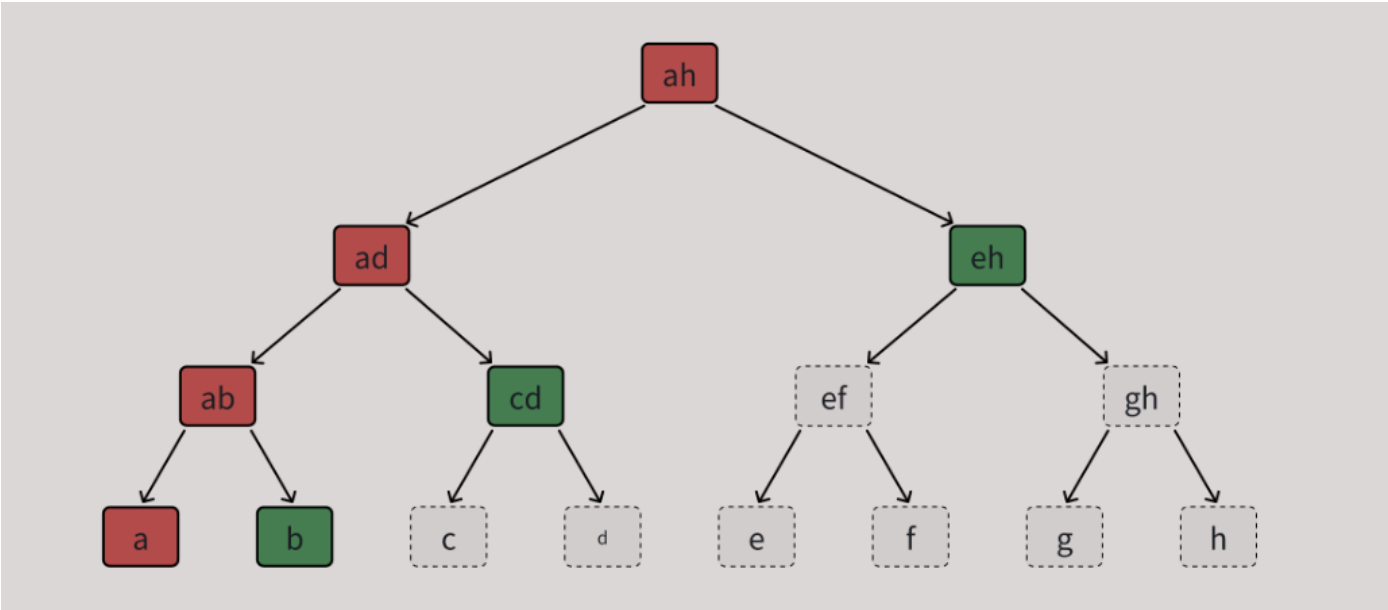
남아 있어야 합니다. 풀 노드가 항상 100% 네트워크 사용률로 작동한다면, 신규 노드는 절대로 최신 블록까지 따라잡을 수 없습니다. 마지막으로, 피어 투 피어 네트워크 프로토콜은 필연적으로 자체적인 오버헤드를 발생시킵니다.

그렇다면, 상태 동기화에 어느 정도의 대역폭을 편안히 할당할 수 있을까요? 평균 지속 가능 대역폭이 75 Mbps라고 가정하고, 그 중 2/3을 다른 애플리케이션과 신규 노드 부트스트랩 용도로 남겨둔다면, 상태 동기화에는 25 Mbps만 사용할 수 있습니다. 이 경우, 초당 100,000건의 Uniswap 스왑을 동기화하려면 상태 변경 데이터에 대해 약 19배의 압축률이 필요합니다!

상태 루트 갱신

Ethereum을 포함한 대부분의 블록체인은 각 블록 이후 상태를 커밋하기 위해 MPT와 같은 트리 형태의 인증 데이터 구조(authenticated data structure)를 사용합니다. 상태 루트(state root)로 알려진 이 커밋은 라이트 클라이언트에게 저장 증명을 제공하는 데 필수적입니다. 따라서 노드 특수화가 존재하더라도, 풀 노드는 시퀀서 노드와 마찬가지로 상태 루트를 유지해야 합니다.

앞서 논의한 바와 같이, Reth에서 상태 루트를 갱신하는 작업은 트랜잭션 실행보다 약 10배나 더 비쌉니다. 이 과정은 극도로 IO 집약적이기 때문입니다. 예를 들어, 아래 그림은 이진 MPT에서 리프 노드를 갱신할 때 발생하는 IO 패턴을 보여줍니다.



MPT에서 각 리프 노드는 블록체인 상태의 키-값 쌍을 저장하고, 각 내부 노드는 해당 서브트리를 커밋하는 중간 해시를 저장하며 자식 노드에 대한 포인터를 포함합니다. 리프를 수정한 후 상태 루트를 갱신하려면, 루트부터 해당 리프까지의 경로에 있는 모든 내부 노드를 갱신해야 하며, 이를 위해서는 자식 노드들을 읽어서 해시를 재계산해야 합니다. 예를 들어, 위 그림에서 리프 a를 갱신하려면 총 3개의 노드를 읽고 4개의 노드를 갱신해야 합니다. 내부 노드는 갱신 전에 반드시 읽어야 하며, 그렇지 않으면 자식 노드를 어떻게 가져와야 할지 결정할 수 없습니다.

일반적으로, n 개의 리프를 가진 k 진 MPT에서 하나의 키-값 쌍을 갱신하려면 $O(k \log_k n)$ 번의 읽기와 $O(\log_k n)$ 번의 쓰기가 필요합니다. 160억 개의 키(약 1TB 블록체인 상태)를 가진 이진 MPT에서는 약 68회의 읽기 작업과 34회의 쓰기 작업이 필요합니다. 다행히, 일반적으로 MPT의 첫 24단계는 캐시할 수 있으므로 마지막 20번의 읽기와 10번의 쓰기만 랜덤 디스크 IO를 유발할 수 있습니다.

이 과제를 더 잘 이해하기 위해 초당 100,000건의 ERC-20 전송 예시를 다시 살펴봅시다. 각 전송이 세 개의 값을 수정하므로 상태 트리는 초당 300,000개의 키를 갱신해야 합니다. 단순한 이진 MPT를 사용할 경우, 이는 약 600만 번의 비캐시 데이터베이스 읽기로 이어집니다. 각 읽기 작업이 단일 디스크 I/O로 처리된다고 가정하더라도, 600만 IOPS는 현재 어떤 소비자용 SSD도 감당할 수 없습니다 — 그리고 이 계산에는 쓰기 작업조차 포함되어 있지 않습니다.

디스크 I/O를 줄이기 위한 일반적인 최적화 전략은 여러 트라이 노드를 서브트리 단위로 묶어 하나의 4KB 디스크 페이지에 저장하는 것입니다. 예를 들어, NOMT는 깊이 6의 루트 없는 이진 서브트리를 하나의 디스크 페이지에 담습니다. 이상적으로는, 이렇게 하면 키 하나를 갱신할 때 필요한 비캐시 데이터베이스 읽기 수가 20개에서 2개로 줄어들어, 약 600,000 IOPS로 감소할 수 있습니다. 그러나 실제 구현에서는 소프트웨어 오버헤드로 인해 이론적인 계산과 실제 성능 사이에 큰 차이가 발생할 수 있다는 점을 유념해야 합니다. Thrum의 벤치마크에 따르면, 1억 3천4백만 개의 키를 가진 NOMT는 초당 최대 50,000개의 리프 업데이트를 처리할 수 있습니다. 이는 기존 상태 트라이 구현체보다 상당히 개선된 수치이지만, 여전히 우리가 원하는 처리량의 6분의 1 수준이며, 그마저도 상태 크기가 위의 예시보다 128배 작을 때 기준입니다.

블록 가스 한도

지금까지 우리는 블록체인 노드가 수행하는 다양한 작업을 가속화하는 데 따른 도전 과제들을 살펴보았습니다. 하지만 여기에는 하나의 중요한 제약이 존재합니다: 어떤 특정 작업을 대부분의 경우 10배 빠르게 만드는 훌륭한 기술을 개발하더라도, 그것이 반드시 블록체인 전체를 더 빠르게 만든다는 것을 의미하지는 않습니다.

그 이유는 블록체인의 최대 속도는 블록 가스 한도(block gas limit)라는 인위적인 상한선에 의해 제한되기 때문이며, 이 한도는 합의 규칙에 포함되어 있습니다. 블록 가스 한도는 하나의 블록 내에서 소비될 수 있는 최대 가스 양을 정의합니다. 블록 시간과 함께 이 한도는 시스템 내 모든 노드가 최소 하드웨어 요구사항만 갖추고 있다면 네트워크 전체와 신뢰성 있게 보조를 맞출 수 있도록 하는 제어 메커니즘 역할을 합니다.

블록 가스 한도는 블록체인의 보안성과 신뢰성을 보장하는 데 필수적인 요소입니다. 일반적인 경험칙은, 블록 가스 한도는 해당 한도 내의 어떤 블록이라도 해당 블록 시간 안에 안정적으로 처리할 수 있어야 한다는 것입니다. 이 기준을 충족하지 못하면, 악의적 공격자들로부터 네트워크가 취약해질 수 있습니다. 다시 말해, 블록 가스 한도는 최악의 경우를 감안해 보수적으로 설정되어야 합니다.

예를 들어, 병렬 EVM의 실제 속도 향상은 워크로드에 크게 의존한다는 점을 떠올려보십시오. 과거의 워크로드에서 평균적으로 2배의 속도 향상을 얻을 수 있다고 해도, 상당수의 블록은 긴 의존성 체인을 포함하고 있어 거의 속도 향상이 발생하지 않습니다. 따라서, Solana의 로컬 수수료 시장과 같은 다차원적인 가스 가격 책정 메커니즘이 없이 충돌이 많은 상태 접근에 대해 가격을 더 정교하게 부여하지 않는 한, 병렬 EVM의 평균 또는 최대 속도 향상만을 근거로 블록 가스 한도를 높이는 것은 현실적이지 않습니다.

마찬가지로, JIT 컴파일된 계약의 속도 향상도 계약의 성격에 크게 의존합니다. 연산 집약적인 계약은 1-2 자릿수 속도 향상을 얻을 수 있지만, 대부분의 오늘날 계약은 그다지 큰 개선을 경험하지 않습니다. 게다가 JIT 컴파일은 현재 가스 모델에 반영되지 않은 오버헤드를 동반합니다. 예를 들어, 컴파일 자체가 CPU 사이클을 소비하며, 컴파일된 네이티브 코드를 유지하는 데 메모리 또는 디스크 공간이 필요합니다. 따라서 가스 모델이 (1) 컴파일 오버헤드를 반영하고, (2) 컴파일로 이득을 보지 못하는 opcode에 대해 적절히 가스 재조정을 수행하기 전까지는, 연산 집약적 애플리케이션을 가능하게 하기 위해 블록 가스 한도를 공격적으로 상향 조정할 수 없습니다.

지원 인프라

마지막으로, 사용자들은 시퀀서 노드와 직접 상호작용하지 않으며, 대부분은 풀 노드를 집에서 운영하지도 않습니다. 대신, 사용자들은 제3자 RPC 노드에 트랜잭션을 제출하고, etherscan.io 같은 블록체인 익스플로러나 dApp의 웹 프론트엔드를 통해 트랜잭션 결과를 확인합니다.

따라서 블록체인의 실제 사용자 경험은 RPC 노드나 인덱서와 같은 지원 인프라에 크게 좌우됩니다. 아무리 실시간 블록체인이 빠르더라도, RPC 노드가 트래픽 피크 시점에 대량의 읽기 요청을 효율적으로 처리하지 못하거나, 트랜잭션을 시퀀서 노드에 신속히 전달하지 못하거나, 인덱서가 체인의 상태를 빠르게 반영하지 못한다면, 그 성능은 무의미해집니다.

원칙 기반의 블록체인 확장

이제 블록체인은 많은 구성 요소가 움직이는 복잡한 시스템이며, 블록체인 성능을 향상시키기 위해서는 병렬 EVM이나 JIT 컴파일과 같은 고립된 기술만으로는 부족하다는 점이 명확해졌기를 바랍니다. Solana와 Aptos 같은 고성능 L1 블록체인들은 시스템 전반에 걸쳐 인상적인 엔지니어링 노력을 기울여 왔습니다.

이것이 바로 MegaETH가 처음부터 총체적이고 원칙 중심의 R&D 접근 방식에 전념하는 이유입니다. 초기부터 심층적인 성능 분석을 수행함으로써, 우리는 항상 실제 사용자에게 의미 있는 이득을 가져다줄 문제를 해결하는 데 집중할 수 있습니다. 지금까지 살펴본 모든 과제들을 지나오며, 우리는 여러분께 보여드릴 수 있는 멋진 성과들이 있다는 것에 매우 흥분하고 있습니다. 구체적인 솔루션은 본 게시물의 범위를 벗어나지만, 앞으로 더 자세히 다룰 수 있기를 기대합니다.

실시간 성능을 수십억 사용자에게

실시간 블록체인은 Web2 서버와 블록체인의 경계를 허물게 될 것입니다. MegaETH를 통해 최종 사용자들은 전례 없는 실시간 성능을 경험하게 되며, 개발자들은 가장 대담한 아이디어를 제약 없이 탐구할 수 있게 될 것입니다. 실험의 10년이 지나, 우리는 마침내 Web2 규모의 애플리케이션을 온체인에서 실현할 수 있게 되었습니다.

지금까지 저희를 지지해주신 모든 분들께 깊이 감사드리며, 앞으로 더 많은 것을 나눌 수 있게 되어 매우 기쁩니다. 다음에 또 만나요.