LevelDB 기반 Ethereum 클라이언트 디스크 증폭

LevelDB-based Ethereum client disk amplification

1st Gunwo Do
dept. Computer Engineering
Pusan National Univercity
Busan, Korea
doogunwo@pusan.ac.kr

2nd SJ Yun
dept. Computer Engineering
Pusan National Univercity
Busan, Korea
yunekorea@pusan.ac.kr

***Abstract*— 2015년에 처음 출시된 이후 이더리움 플랫폼은 2024년 현재까지 활발하게 성장하고 있다. Geth와 같은 이더리움 클라이언트는 LevelDB를 백엔드 데이터베이스로 사용하는데, 이는 계층적 SSTable 구조를 관리하는 동안 압축 및 쓰기 중단과 관련된 성능 문제가 발생한다. 이러한 문제는 블록체인 네트워크가 성장함에 따라 증가하는 저장 공간으로 인해 더욱 악화되고 있다. 이 논문은 LevelDB 아키텍처가 이더리움 클라이언트에 미치는 성능 영향을 연구한다. 이 연구는 '프라이빗 네트워크 노드'와 '메인넷 동기화 노드'를 분석하여 이더리움 클라이언트와 디스크 증폭 사이의 관계를 분석하였다.**

Keywords—Blockchain, Disk amplification, LevelDB

I. INTRODUCTION

블록체인은 디지털 자산의 원장을 공유하여 일관성과 신뢰성을 제공하는 자료구조이다. 블록 간의 얽힘으로 제공하는 무결성을 통해 물류, 금융과 같은 추적성이 필요한 산업에 응용되고 있다. 이러한 환경에서 이더리움은 스마트 컨트랙트를 통해 프로그래밍 가능한 일종의 계약(Contract)를 제시하여 블록체인 네트워크상에서 동작가능한 프로그램인 '스마트 컨트랙트'를 도입했다. 2015년 등장한 이더리움은 2024년 현재까지 가장 활발한 블록체인 프로젝트중의 하나이다. 하지만 네트워크가 활발해지는 것은 성장의 원동력이 될 수 있지만, 동시에 '양날의 검'으로 작용한다. 이더리움은 아키텍처의 문제로 디스크 입출력 증폭(Disk I/O amplification) 문제를 가지고 있다.[1] 이더리움은 네트워크의 안정성을 위해 블록 생성 주기를 짧게 하였음에도[2] 9년간 약 2000만 개의 블록과 5.2억개의 트랜잭션이 생성되었다. 이와 같이 거대한 백엔드 스토리지는 이더리움의 읽기 및 쓰기 성능 저하시킨다.[3] 이더리움의 KV 저장소는 읽기 작업이 동작하면 Key, Value가 올바른지 검증하기 위해서 Mekle Proof를 반환하고 읽을 수 있는 Key들은 Merkle 트리의 일부분인 reaf로 포함된다.[3] 각 쓰기 작업은 각 쓰기 작업은 루트에서부터 잎(read)까지 경로에 존재하는 모든 노드의 해시를 변경한다. 이러한 과정은 이더리움 클라이언트에서 많은 비용이 소모되는 작업에 해당된다. 따라서 이더리움 클라이언트는 디스크 읽기/쓰기 작업을 과도하게 수행하도록 설계되었다. 이에 따라 LevelDB 테이블 비율에 따른 Go-ethereum의 디스크 증폭을 본 논문에서 다룬다.

II. BACKGROUND

A. Go-ethereum

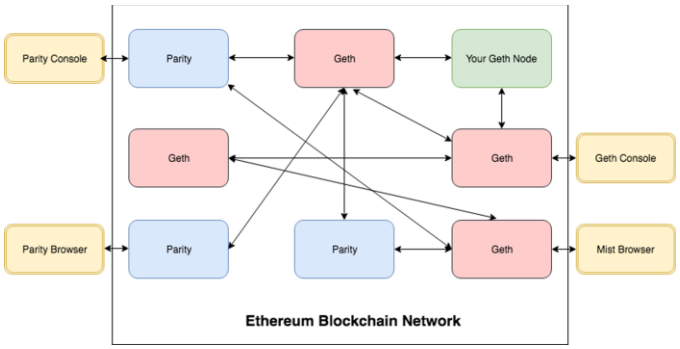


Fig 1. Ethereum network diagram [4]

그림 1(Fig 1)과 같은 다양한 종류의 이더리움 클라이언트로 이더리움은 메인넷을 구성하고 있다. 다양한 클라이언트는 네트워크의 안정성을 제공하고 네트워크 분산화를 강화한다. 그럼에도 Geth, Parity같은 이더리움 클라이언트들의 취약점과 버그가 발견되어 해킹이 여전히 시도되고 있다. 클라이언트의 취약점에 못지않게 블록체인 자체의 확장성 또한 이더리움 네트워크 발전을 저지하고 있다. 블록체인 구조는 확장성(scalability)가 낮다는 명확한 단점을 가지고 있다.[4] 이더리움의 동기화 방법 중 하나인 '아카이브(Archive) 모드'는 2019년 기준 '약 3TB 이상의 저장공간'이 필요했으며[4], 2024년 현재 '약 13TB 이상의 읽기/쓰기 속도'가 보장되는 고성능 저장장치가 필요하다. 또한 저장공간이 늘어남에 따라 네트워크 동기화 시간도 증가하고 있다. 낮은 확장성 문제는 블록체인의 네트워크를 중앙화로 이끈다.[4]

B. LevelDB

이더리움 데이터베이스는 'Key-value store' 구조인 LevelDB를 백엔드로 구성하고 있다. Go-Ethereum(Geth)

Identify applicable funding agency here. If none, delete this text box.

클라이언트의 경우에는 go언어로 작성된 LevelDB 패키지를 래핑(wrapping)하여서 사용한다. 이더리움의 데이터베이스 구조는 최신 90,000번째 블록까지의 대한 'Key-value store'인 데이터와 이전 블록에 대한 'Ancient Store' 데이터 두 종류로 나뉜다.[5] 데이터에는 블록 헤더, 바디, 트랜잭션 영수증에 대한 정보가 저장되고 LevelDB에는 트랜잭션, 계정(Account), 블록에 대한 정보를 관리하는 '트라이 노드(Trie node)'를 두고 있다.[6]

많은 클라이언트 중에서 Go-Ethereum(Geth)는 이더리움 생태계에서 가장 널리 사용되는 소프트웨어다.[8] Geth를 기반으로 다양한 확장성 연구가 진행되고 있으며, 연구 목적에 따라 다양한 실험 및 분석을 진행할 수 있다. 이더리움 홈페이지[8]에서 참고한 자료에 따르면 이더리움 클라이언트 분포도는 Geth가 42.1%를 차지하고 있다.[7]

C. Go-ethereum Backend

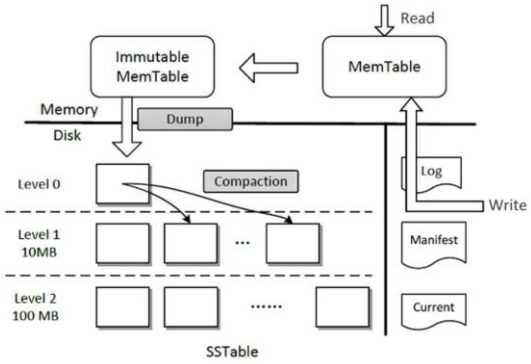


Fig 2. Architecture of LevelDB [8]

그림 2(Fig 2)는 LevelDB의 아키텍처이다. Memtable과 Immutable Memtable로 메모리 구성 요소는 분리된다. 보조 저장소는 여러 레로 구성되며 여러 레벨인 L0(Level1), L1(Level2), L2(Level2) 나뉜다. 각 레벨에는 KV항목들이 SSTable 파일로 구성된다. 추가되는 KV항목들은 Memtable(메모리 위치)에 먼저 삽입된다. Memtable이 가득 차면 모든 KV항목은 읽기 전용 Immutable Memtable로 변환된다.[8] 컴팩션 연산은 L0이 가득 차면 L0 부터 L1 까지 데이터를 압축(컴팩션)하고 데이터를 메모리로 읽고 KV 항목을 정렬한 뒤, 분리된 새로운 Key 범위를 가진 새로운 SSTable 파일을 생성하여 L1에 기록하는 연산을 가진다. 기본적으로 각 레벨에 대한 컴팩션 작업은 큰 쓰기 증폭을 발생한다.[8] 따라서 Geth와 같은 애플리케이션에서 발생하는 쓰기 증폭은 LevelDB의 기본 아키텍처인 LSM-트리를 개선해야 한다.

D. Write Stall

메모리 버퍼가 가득 차면 테이블 파일 형태로 스토리지에 플러시를 수행하여 다음에 들어올 KV 데이터를 축적할 공간을 확보한다. 이 때, 중복된 KV 를 정리하여 그림 2(Fig 2)의 SSTable 과 같은 계층적인 구조를 구성하고 유지한다. 하지만 컴팩션은 스토리지 대역폭을 차지하는 작업이기 때문에, 자주 진행하면 플러시(Flush)가 미뤄지고 메모리 버퍼에 더 이상 KV 쓰기 요청을 수행하지 못하는 쓰기 스톨(Write Stall) 현상이 발생할 수 있다.[9] 이러한 쓰기 스톨을 줄이기 위해 Level 0(최상위 계층)의 테이블 파일 개수 임계값을 늘림으로서 컴팩션 작업을 미루는 방법을 사용할 수 있다. 그러나 이러한 방법은 읽기 성능에 문제가 발생한다.[9] Geth 는 LevelDB 를 백엔드 데이터베이스로 기반을 둔 애플리케이션이다. 하지만 LevelDB 의 읽기 작업은 Level 0 의 테이블 파일을 전부 순회하면서 KV 탐색이 수행되는 반면, Level 0 계층의 테이블 파일들은 정렬이 되지 않기 때문에 Level 0 에 해당되는 파일이 많아질 수록 비효율적인 스토리지 접근 빈도가 높아지며 읽기 성능 저하가 발생하게 된다.[9] 이러한 문제는 자연스럽게 Geth 의 읽기 성능 저하로 이어진다.

E. LevelDB Get

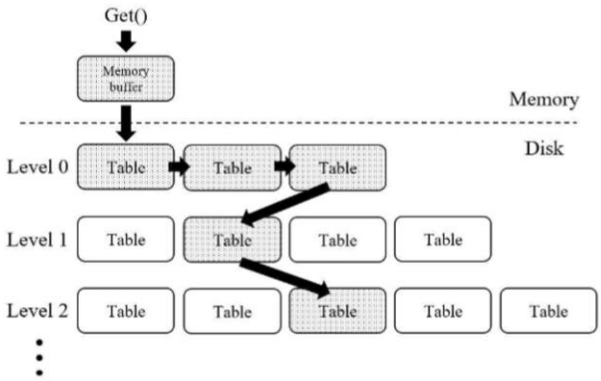


Fig 3. Key display order for Get moves in LSM-tree [9]

그림 3(Fig 3)과 같이 LSM-tree 기반 KV 스토어의 읽기 연산은 Get, Seek 연산을 수행한다. 데이터를 찾기 위해 메모리 버퍼를 우선적으로 탐색하고, SSTable 을 구성하는 각각의 레벨을 순서대로 탐색한다. 이 때, 레벨 0 에서는 모든 테이 파일의 인덱스 블록을 탐색하는 구조이다.[9] LSM-트리 기반 KV 스토어의 읽기를 최적화하는 기법에는 RocksDB 와 같이 인덱스 블록[9]을 통해 메모리에 전부 캐싱하는 방법이 있지만, 거대한 블록체인 네트워크 노드에 대한 정보를 모두 메모리에 담을 수는 없다.

F. Disk amplification

이더리움은 저장소 속도 확보를 위해 SSD 를 제외한 저장장치를 저장소로 지원하지 않는다. SSD 와 같은 플래시 메모리 디스크는 실제로 기록된 데이터 양보다 더 많은 데이터가 저장치에 기록되는 현상인 쓰기 증폭이 존재한다. 디스크 쓰기 증폭은 시스템의 수명, 성능 및 안정성에 영향을 미칠 수 있다. 이더리움 같은 블록체인 시스템에서도 충분히 쓰기 증폭이 발생할 수 있다고 고려할 수 있다. 이더리움 메인넷 노드 동기화 과정은 많은 디스크 쓰기(Disk Write)가 발생하는 동작이며 이는 데이터베이스 구조와 상태 데이터 저장 방식에서 기인된다.

III. THE ETHEREUM STORAGE BOTTLENECK [3]

이전 연구에 따르면 이더리움 저장소에는 IO 증폭문제가 있다. 이는 모든 이더리움 읽기 작업에 다수의 LevelDB 읽기가 필요하다. 모든 이더리움 쓰기 작업은 여러 번의 LevelDB 쓰기를 요구하고 이러한 저장소 병목 현상은 DoS 공격에 악용될 수 있다. 실험구성을 위해 메인넷의 160만 개의 블록을 가져와 노드를 구성하고 go-ethereum을 사용하여 블록체인 데이터를 동기화하는 노드으로 실험환경을 구성했다. 하드웨어 장비는 16GB의 RAM과 raid0을 사용한 2TB Intel 750시리즈 SSD를 저장소로 설정했다.[3] 160만 개의 블록에서 블록, 트랜잭션, 계정주소를 추출하고 Geth가 제공하는 내부 매트릭을 보완하여 다음과 같은 결과를 얻었다.

TABLE 1. Ethereum block latency metrics

Metrics	*# of execution*	*Value(LevelDB gets)*
getBlock	1.6M	8M (5x)
getTx	5.2M	10.4M (2X)
getBalance	0.22M	1.4M (7X)
depth of trie		7

표 1(TABLE 1)에 따르면 결론적으로 2018년 기준 go-ethereum 클라이언트에서는 I/O 증폭이 발생하고 있었음을 확인할 수 있다. 3개의 매트릭(getBlock, getTx, getBalance)에서는 각각 5배, 2배, 7배의 매트릭 실행횟수보다 많은 LevelDB get이 발생했으며, 현재 2024년 기준 약 2000만개의 이더리움 네트워크에서는 더 큰 증폭이 발생할 것이라고 예상할 수 있다.

IV. EXPERIMENT

2018년부터 2024년까지 이더리움 네트워크에는 많은 변화가 발생했다. 이더리움은 메인넷 배포 이후부터 지속적으로 DoS 공격을 받았고, 이를 방어하기 위해 지속적인 하드포크를 수행했다. 또한 이를 위해 클라이언트도 지속적인 개선이 이루어졌다. PoW 합의 알고리즘으로부터 PoS 합의 엔진으로 전환하고 Geth 클라이언트는 버전 문제가 중요해졌다. 낮은 버전(PoS를 지원하지 않는) Geth 클라이언트는 메인넷 접속을 지원하지 않는다. 따라서 실험에서는 프라이빗 네트워크 노드 구성은 Geth1.10.26버전으로 수행되었고 메인 네트워크 노드 구성은 Geth1.14.5 버전을 사용하였다. 이더리움 클라이언트의 저장공간, 동기화 시간은 지속적으로 증가하고 있으며, 블록체인에 대한 관심도가 높아질수록 활발해지는 트랜잭션 발생은 네트워크 확장에 상당한 부담을 준다. 이러한 문제로 메인넷을 모방한 프라이빗 네트워크 노드와 메인넷을 동기화한 노드를 통해 실험환경을 구성하였고, 메인넷을 모방한 프라이빗 네트워크 노드에서 발생하는 I/O 증폭 문제와 실제 노드에서 발생하는 I/O증폭 문제를 비교하여 어떠한 차이가 존재하는지 확인하고, 실제 메인넷에서 블록 수에 따른 LevelDB SSTable Level0의 파일 개수가 읽기 성능을 저하시키는지(Write Stall) 확인하도록 한다.

A. Considerations

2024 년 기준 메인 네트워크를 구성하는 노드에는 2 가지 유형(아카이브, 스냅샷)이 있다. 스냅샷 모드는 동기화 속도, 저장공간 절약의 효과적인 장점이 존재하지만 실험에는 적합하지 않다. 저장 공간 절약을 위해 최근에 만들어진 블록의 트랜잭션이 아니면 삭제하는 스냅샷은 모든 트랜잭션을 조회할 수 없다. 따라서 프라이빗 네트워크를 구성하여 Geth 클라이언트이 디스크 증폭과 관련된 매트릭을 수집하였다.

TABLE 2. Blockchain network configuration

Network	*Block*	*Tx*	*Account*	*Level0 table*	*Storage*
Private	400,00	2,457,488	2,457,487	3	7.8GB
Mainnet	4,800,000	120,000,000	10,269,080	300	1.5TB

표 2(TABLE 2)는 프라이빗 네트워크를 구성하는 블록 수, 트랜잭션 수, 계정 수, Level0 테이블 수, 차지하는 스토리지 공간을 확인할 수 있다. 또한 동일하게 메인넷 노드에 대한 데이터도 확인할 수 있다.

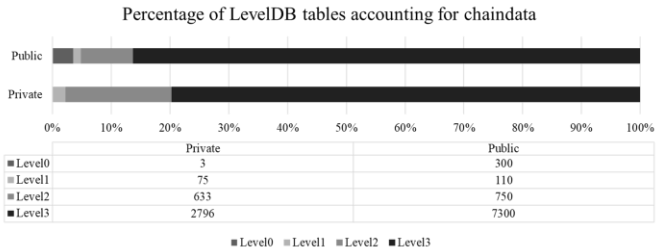
TABLE 3. Measurement results

Node	*Metrics*	*Execute*	*Read*			*Work time*
			Disk	*Chaindata*	*bytes*	
Private	getBlock	0.4	5.2	92.3	1	20
	getTx	2.4	20	150	5.4	27
	getBalance	2.4	0.64	42	0.4	3
Mainnet	getBlock	4.8	665	224,000	34	1200
	getTx	10.6	99	61,000	107	200
	getBalance	1.2	23.7	0.0013	4	50

# of execute (Million)
Disk read (Million)
Chaindata read (Million)
Read bytes (GB)
Work time (minute)

표 3(TABLE 3)의 결과는 위와 같아. 표 1(TABLE 1)과 비교해서 볼 점은 표 1(TABLE 1)에서는 getBalance에 상당한 읽기 오버헤드가 발생한다고 측정되었으나 표 1(TABLE 1)의 결과를 확인해보면 오히려 다른 매트릭 측정에 비해 적은 오버헤드를 보인다. 또한 표 3(TABLE 3)의 Private와 Mainnet의 두 노드를 비교하면 결과의 디스크 증폭 비율이 비슷한 양상을 보인다. 표 3(TABLE 3)의 매트릭 측정결과 프라이빗 네트워크에서 getBlock, getTx, getBalance 연산의 실행횟수, 디스크 읽기, 워크로드 작업 시간을 측정했다. 메인넷 노드에서도 동일한 매트릭을 측정했으며 결과는 getBlock, getTx 매트릭은 프라이빗 네트워크와 메인넷 노드 모두 상당한 Disk 읽기와 Chaindata 읽기가 발생했다. getBalance 매트릭은 프라이빗 네트워크와 메인넷 모두 상대적으로 적은 오버헤드 결과를 보인다. 이

결과는 블록체인의 네트워크가 규모와 상관없이 Geth 클라이언트에서 디스크 증폭 문제가 존재함을 시사한다.



	Private	Public
Level0	3	300
Level1	75	110
Level2	633	750
Level3	2796	7300

Fig 4. Percentage of LevelDB tables accounting for chaindata

최종적으로 그림 4(Fig 4)의 LevelDB table 의 비율과 표 3(TABLE 3)의 결과를 비교하면 블록체인 네트워크와 상관없이 디스크 증폭은 발생하지만 Level0 Table 이 차지하는 비율이 클수록 더 높은 디스크 증폭이 발생하는 것을 확인할 수 있다. 결론적으로는 네트워크가 성장함에 따라 컴팩션을 강력하게 하여 Level0 table 의 수를 조절하지 않으면 더욱 강한 디스크 읽기 증폭이 발생할 것이다.

V. Conclusion

본 연구는 이더리움 클라이언트인 Geth 의 백엔드 데이터베이스로 사용되는 LevelDB 의 아키텍처가 디스크 증폭에 미치는 영향을 분석하였다. 프라이빗 네트워크와 메인넷 노드를 구성하여 실험한 결과, 이더리움 클라이언트는 네트워크 규모와 상관없이 디스크 증폭 문제가 발생하는 것을 확인하였다. 특히, LevelDB 의 Level 0 테이블 파일 수가 많을수록 더 높은 디스크 읽기 증폭이 발생하는 것으로 나타났다. 블록체인 네트워크의 지속적인 성장으로 인해 이더리움 클라이언트의 백엔드 스토리지 크기가 급격히 증가하고 있다. 이는 LevelDB 의 계층적 구조에서 발생하는 컴팩션 및 쓰기 중단 문제를 악화시키며, 결과적으로 이더리움 클라이언트의 성능 저하를 초래한다. 실험 결과에 따르면, 프라이빗 네트워크와 메인넷 노드 모두에서 getBlock 과 getTx 연산 시 상당한 디스크 읽기와 Chaindata 읽기가 발생하였으며, 이는 LevelDB 의 Level 0 테이블 파일 수와 밀접한 관련이 있는 것으로 나타났다. 따라서 이더리움 클라이언트의 성능을 향상시키기 위해서는 LevelDB 의 컴팩션 알고리즘을 최적화하여 Level 0 테이블 파일 수를 적절히 조절하는 것이 필요하다. 이를 통해 디스크 읽기 증폭을 완화하고, 이더리움 클라이언트의 전반적인 성능을 개선할 수 있을 것으로 기대된다. 본 연구의 결과는 이더리움 클라이언트의 성능 향상을 위한 중요한 단서를 제공한다. 향후 연구에서는 LevelDB 의 컴팩션 알고리즘 최적화 방안, 효율적인 데이터 관리 기법 등을 심층적으로 분석하고, 이를 바탕으로 이더리움 클라이언트의 성능을 한층 더 개선할 수 있는 방안을 모색할 필요가 있다. 나아가 블록체인 네트워크의 확장성 문제를 해결하기 위한 다양한 접근 방식에 대한 연구도 병행되어야 할 것이다.

References

[1] Xia, D., Yao, P., Liang, J., & Chen, W. (2024). RemoteBlock: A Scalable Blockchain Storage Framework for Ethereum. In *2024 IEEE 4th International Conference on Power, Electronics and Computer Applications (ICPECA)* (pp. 878). IEEE. DOI: 10.1109/ICPECA60615.2024.10470941.K. Elissa, “Title of paper if known,” unpublished.

[2] Choi, Jemin Andrew, et al. "LMPTs: Eliminating Storage Bottlenecks for Processing Blockchain Transactions." 2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). IEEE, 2022. DOI: 10.1109/ICBC54727.2022.9805484J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] Raju, P., Ponnapalli, S., Oved, G., Keener, Z., Kaminsky, E., Chidambaram, V., & Abraham, I. (Year). "mLSM: Making Authenticated Storage Faster in Ethereum.".

[4] https://ethernodes.org/.

[5] Koo, Y., & Moon, S. (2019). "Analysis of Ethereum Storage Towards Decentralization." In *Proceedings of the 2019 Korea Software Congress* (pp. Pages). Department of Electrical and Computer Engineering, Seoul National University.

[6] Kim, J., & Moon, S. (Year). "Analysis of Merkle Patricia Tries for Lightening Ethereum." Department of Electrical and Computer Engineering, Seoul National University.

[7] https://ethernodes.org/.

[8] Seo, Y.-C., & Park, S.-H. (2024). PlexDB: Efficient Compaction Algorithm for Deduplication of LSM-tree based Key-Value Store. Department of Computer Engineering, Chungbuk National University, Cheongju, Korea.

[9] Lee, J., Song, Y., & Eom, Y. I. (2024). Read/Write Performance Analysis depending on Trigger Condition for LSM-tree Compaction. Dept. of Electrical and Computer Engineering, Sungkyunkwan University, Korea.