LevelDB-based Ethereum client read amplification

Gunwo Do
School of Computer Engineering
Pusan National Univercity
Busan, Korea
doogunwo@pusan.ac.kr

SJ Yun
School of Computer Engineering
Pusan National Univercity
Busan, Korea
yunekorea@pusan.ac.kr

Sungyong Ahn
School of Computer Engineering
Pusan National Univercity
Busan, Korea
sungyong.ahn@pusan.ac.kr
*Corresponding Author

***Abstract*— This paper addresses about Go-ethereum, an Ethereum client, uses LevelDB as its backend database. The LSM-tree architecture of LevelDB ensures write performance but at the cost of read performance. This trade-off between write and read performance worsens as the blockchain network grows. This paper investigates the performance impact of the LevelDB architecture on the Ethereum client. The results show that an increased number of Level0 tables negatively affects read performance, causing read amplification where the Ethereum client must read small pieces of data multiple times.**

Keywords—leveldb, ethereum, disk, read-amplification, blockchain

I. INTRODUCTION

Blockchain is a data structure that provides consistency and reliability by sharing a ledger of digital assets. Ethereum, which emerged in 2015, introduced smart contracts, enabling the execution of programmable contracts on blockchain networks. As of 2024, Ethereum remains one of the most active blockchain projects [1]. However, Ethereum network's growth simultaneously presents challenges. Go-Ethereum, one of the Ethereum clients, faces disk I/O amplification issues due to its backend architecture [2]. The expansive backend storage, comprising approximately 20 million blocks and 520 million transactions, degrades Ethereum's read performance. This paper addresses the disk amplification problem in LevelDB-based Ethereum clients, focusing on the relationship between LevelDB table ratios and disk amplification.

II. BACKGROUND

Geth uses LevelDB, which has a 'Key-value store' structure [3], as its backend storage. In the case of the Geth client, it wraps the LevelDB package written in Go language for its use.

A. LevelDB

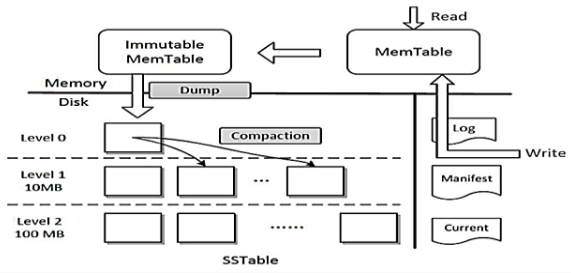

Fig. 1. Architecture of LevelDB [4]

Fig 1. illustrates the LevelDB architecture. The memory storage consists of a Memtable and an Immutable Memtable, while the disk storage is organized into multiple levels through compaction. At each level, KV (Key-Value) entries are compacted into sorted files (*.sst), except for Level 0. To ensure optimal write performance [4], files in Level 0 are unsorted and allow duplicates.

B. LevelDB get

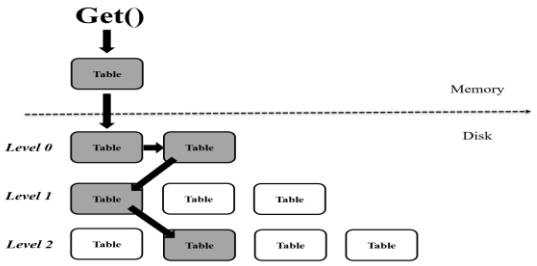

Fig. 2. LevelDB Get executetion operation process [5]

When compaction occurs in LevelDB SSTable, it eliminates duplicate KV and organizes them [4], forming a layered structure, as shown in Fig 2. However, since compaction consumes storage bandwidth, frequent compactions can delay flushing, causing write stalls where write requests to the memory buffer cannot be executed. To reduce write stalls, the threshold for Level0 Tables can be increased, thereby delaying compaction operations and preventing write stalls. However, this method degrades read performance [5]. LevelDB read operations traverse all Level0 Tables to search for KV. If the value is not found, it continues the search in subsequent layers. Importantly, Level0 Tables are not sorted and allow KV duplicates to facilitate write operations for updating the most recent data. Consequently, as the number of Level0 Tables increases, the frequency of inefficient storage access rises, leading to degraded read performance [5]. This problem naturally results in decreased read performance for Geth.

C. Read amplification of previous research [2]

Previous research indicates that Geth's Ethereum storage suffers from I/O amplification issues. Write operations in Geth require multiple LevelDB get operations. To conduct experimental measurements, a node is set up by synchronizing the Geth client with the ‘mainnet’ from block 1 to 1.6 million. The underlying hardware configuration includes 16GB of RAM and a 2TB Intel 750 series SSD configured in raid0. From blocks 1 to 1.6 million, there are 0.22 million account addresses and 5.2 million transactions within blocks. And when measuring the influence of storage layer, to eliminate the overhead of RPC calls, directly interacting with the geth client (with the golang layer) [2].

TABLE I. READ AMPLIFICATION IN PIROR STUDIES [2]

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

According to To TABLE I, it can be confirmed that I/O amplification was occurring in the go-ethereum client as of 2018. For the three metrics (getBlock, getTx, getBalance), LevelDB get operations occurred 5 times, 2 times, and 7 times more frequently than their respective metric execution counts. It can be anticipated that even greater amplification is likely occurring in the current Ethereum network [2].

III. EXPERIMENT

The experimental setup includes two distinct node types within the main network environment, an archive node and a snapshot node. These configurations allow for a comparative analysis of read amplification metrics across different node architectures. The archive node maintains a comprehensive historical record of the blockchain, while the snapshot node preserves a condensed version of the state.

TABLE II. CONFIGURATION EXPERIMENT WORKLOAD

Node type	**Workload(getBlock)**		
	Measuring block range	***getBlock Execute time***	***Workload Full time***
archive	1 - 400,000	459 s	464 s
archive	1 - 1,600,000	3105 s	3126 s
snapshot	1 - 400,000	475 s	481 s
snapshot	1 - 1,600,000	3045 s	3067 s

TABLE III. RESULT OF EXPERIMENT

Node type	**Read Amplification Metric**			
	Chaindata	***Disk***	***Read bytes***	***Level0 table***
archive	333 M	9.3 M	2.5 GB	300
archive	2489 M	30.6 M	7.9 GB	
snapshot	290 M	6 M	0.77 GB	393
snapshot	2172 M	26.8 M	2.2 GB	

Table II shows the workload range for getBlock and the execution time of getBlock within the total workload duration. Although it could be expected that the archive node would take more time due to the larger amount of data it needs to store compared to the snapshot node, this expectation was not met. Table II suggests that there is no significant difference in read performance across the tested scenarios. Table II demonstrates no substantial difference in read performance across the evaluated scenarios. However, upon examination of Table III, it becomes evident that the archive node exhibits a significantly higher number of read operations compared to the other configurations. When comparing the overall read ratio, it is natural that archvie nodes with larger chain data volumes have a higher number of Read bytes, but when comparing Read bytes, snapshot nodes with larger Level0 tables had a higher amplification rate. The results show that the read amplification ratio is higher because the snapshot node has a larger number of Level0 tables. This is not a comparison of the read performance of snapshot nodes and archive nodes, and the results show that the number of level0 tables affects go-ethereum read performance.

IV. CONCLUSION

In summary, LevelDB backend database of the Ethereum client, causes read amplification issues in go-ethereum due to architectural issues. Because LevelDB Get must traverse the entire Level0 table, if the number increases, read performance may be disadvantageous. If so, such a problem would also occur in go-ethereum using LevelDB, so we created two mainnet nodes and varied the number of blocks by varying the number of Level0 tables. The experimental results show that Level0 tables are causing a major problem in deteriorating read performance, and the more Level0 tables there are, the more small-sized data reads occur. Based on these results, we emphasize that for the scalability and stability of go-ethereum, it is necessary to devise a new back-end system and configure an architecture suitable for the characteristics of blockchain. In follow-up research, we aim to improve performance by proposing an optimized architecture suitable for the characteristics of blockchain.

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,

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

[3] M. Kim and S. Moon, "Analysis of the Characteristics of Storage Engines and Their Adoption Effects in Blockchain Databases," in Proc. of the Korea Information Science Society Annual Conference, 2023, pp. 31-33.

[4] 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.

[5] Jeongho Lee, Yongju Song, Young Ik Eom.(2022).Read/Write Performance Analysis depending on Trigger Condition for LSM-tree Compaction. Korean Institute of Information Scientists and Engineers,101-102.