RISE: The Gigagas Layer 2

Samuel Battenally *Hai Nguyen* *Thanh Nguyen*

RISE Labs

Abstract

We present RISE, an innovative Layer 2 (L2) platform designed to address the pressing performance limitations within the Ethereum rollup ecosystem. Despite notable advancements, current Ethereum L2 solutions lag in transaction throughput, significantly underperforming competitors like Solana. RISE leverages a parallel Ethereum Virtual Machine (EVM), a continuous execution pipeline, and a novel state access architecture built on Rust-based Reth SDK to enhance throughput and performance substantially. The core aim of RISE is to achieve the target of The Surge [4], 100,000 transactions per second (TPS) and over 1 Gigagas per second, with the potential for further scalability. This paper details the challenges of existing L2 technologies, the architectural innovations of RISE, and future directions for optimising blockchain scalability and efficiency. RISE promises to meet and exceed the most performant Layer 1 (L1) solution, establishing a new benchmark in blockchain technology.

1 Introduction

1.1 Technology Landscape

When Vitalik Buterin first set the Ethereum blockchain into motion in July 2015 [1], he was not only advancing the foundational technology introduced by Bitcoin [2] but also pioneering a profoundly innovative concept: smart contracts. Smart contracts have since unlocked applications and global coordination, which were only dreamed of after their inception. It's clear from the adoption that the world is realising the potential of this incredible technology. However, the adoption highlighted scaling challenges that ultimately led to the Ethereum Endgame proposed by Vitalik Buterin [3], which has evolved into the Ethereum Roadmap [4]. A core roadmap component is *The Surge*, which aims to reach 100,000 transactions per second and beyond on rollups. Arbitrum and Optimism have both played a significant role in the progress towards this goal by demonstrating EVM rollups as an undeniable solution to scaling. Still, their throughput is falling short [5][6]. The technology is primed for a second generation of rollups, hyper-focused on performance, to close this gap. Thanks to recent innovations, The Surge is becoming a reality. Vitalik argues that we are reaching the exponential component of the capabilities S-curve [7]; RISE aims to unlock this rapid change and the next wave of applications not yet dreamed of.

1.2 Introduction

Ethereum's rollup ecosystem has made commendable progress in enhancing blockchain scalability, securely supporting various applications and capturing substantial market value. However, even with these successes, the combined throughput of the Layer 2 (L2) ecosystem typically processes about 100 transactions per second (TPS)[8], which is far from meeting the performance of competitors like Solana, which consistently achieves over 1000 TPS, ten times that of all L2s combined [9]. Solana achieved this by prioritising scalability over decentralisation, with a lower validator count and high emissions to fund expensive infrastructure. Solana adoption indicates the market's support for a pragmatic approach to scaling. The success of Solana and the disparity in performance between Solana and L2s also sends a clear message that there is demand for high-performance, low-cost EVM L2s, but said demand still needs to be met by the technology.

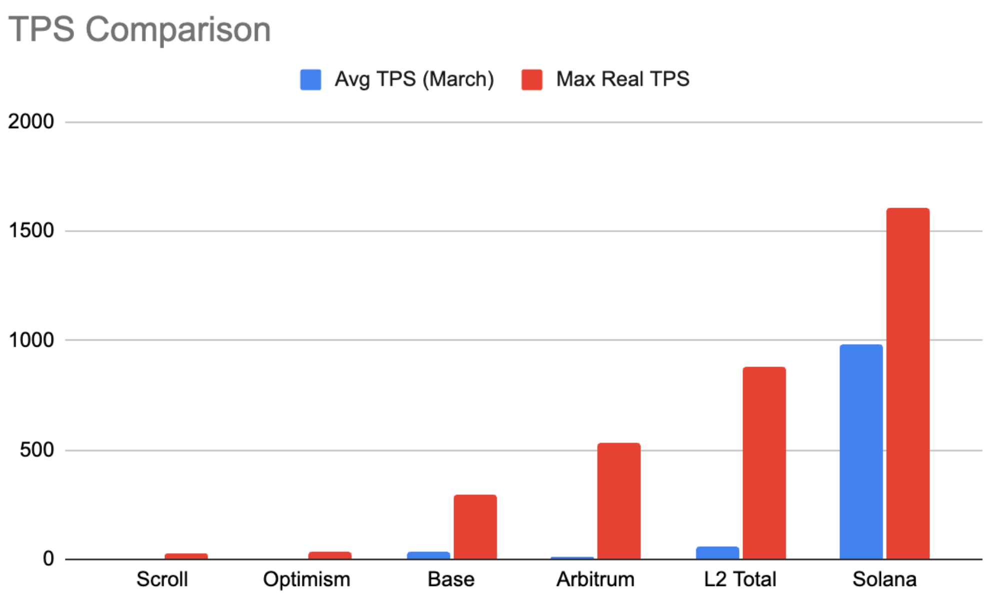
We introduce RISE, a next-generation Ethereum Virtual Machine (EVM) Layer 2 stack. With key components designed from scratch, RISE maximises transaction throughput to achieve the ambitious target of 100,000 transactions per second (TPS) outlined in the Ethereum Roadmap. Data Availability (DA) has historically been a significant roll-up bottleneck, but recent advances, such as Celestia, EigenDA and EIP-4844, have significantly mitigated this issue. The challenge now shifts to execution performance. In tackling execution performance, RISE combines the best technology current research offers and novel innovations, including Parallel EVM Execution, Continuous Block Execution, and an innovative state access design. These are all integrated with the robust, rust-based Reth SDK.

The RISE stack is for a rollup-centric future; to enable flexible deployments, both Sequencing and DA are agnostic in design, meaning RISE app chain deployments can choose a custom Sequencing and DA stack. The core of the RISE stack is high-performance execution. Not only will the RISE mainnet support a throughput beyond the most performant Layer 1's (L1), but so will any number of RISE app chains.

This paper introduces the RISE stack and its performance innovations. However, it does not cover RISE's ambitions to decentralise through based sequencing and participate in a credibly neutral and composable modular ecosystem; this will be covered in future work by the RISE Labs team.

1.3 The state of rollups

The Ethereum approach to scaling has ostensibly been a success. L2s have held over $40 billion in value combined and have averaged over 100 transactions per second (TPS) since February 2024 [2]. With over 40 rollups now live, we are witnessing the early days of Ethereum horizontal scaling. This growth hasn't been frictionless; the lack of a shared state causes applications, users, and liquidity fragmentation. Given the growth of horizontal scaling in L2s, it's fair to conclude vertical scaling has experienced limitations and anti-network effects slowing adoption and activity in individual rollups.



Performance comparison, Layer 2s and Solana.

This is even clearer when we compare L2 adoption and performance to the most performant competitor, Solana. Solana averaged nearly 1000 TPS [2] during March 2024, ten times that of all Ethereum L2s combined. Arbitrum and Base reached noteworthy TPS highs of 532 TPS and 293 TPS. However, both networks experienced outages during peak congestion, whereas Solana steadily supports 2000+ TPS alone. Solana has made significant gains on Ethereum despite the EVM tailwind, further strengthening the conclusion that there is an unmet demand for performance L2s.

1.4 Virtual Machine Selection

The EVM tailwind is powerful; EVM sports more live products, investment, active talent building, and users than any other Smart Contract VM [10]. Building the technology to support 100,000 TPS is only meaningful with adoption from both developers and users. There are design decisions in the EVM that make scaling extremely difficult; however, the EVM tailwind is far too complicated to contend with, so RISE focuses solely on an EVM-compatible L2 stack.

1.5 Zero Knowledge vs Optimistic

zkEVM L2's have made significant progress recently, adoption is at all-time highs and proofing speeds are trending north. RISE aims to usher in The Surge [6] and reach the 100,000 TPS target. This will be possible in the future with a zkEVM, though it will require hardware solutions such as zkASIC to achieve it, which will take to become available [x]. With an optimistic roll-up, however, we're confident high throughput can be achieved without custom hardware acceleration. Due to this, the RISE team will initially focus on an optimistic stack and transition to zkEVM once suitable hardware becomes available.

1.6 Structure of the paper

In this paper, we begin in section 2 by identifying the performance challenges faced when building a high-performance rollup stack; in section 3, we provide a detailed breakdown of the RISE technology stack, and we finish in section 4 with future work and closing remarks.

2 Performance Challenges

The primary performance challenge faced by L2s to date has been data availability (DA). However, this is no longer the case thanks to innovations in external DAs such as Celestia and eigenDA and enshrined solutions like EIP-4844 [8]. We now see many other bottlenecks, such as Networking, Execution, Block Size limitations, Fraud-Proof Restrictions, Merkelisation, and Storage IO. Seemingly overnight, L2 performance has become an exciting and challenging design space with various design directions.

2.1 EVM Execution

Most implementations of EVM execution are single-threaded. Due to this, today, the upper limit for performance is dictated by the clock rate of a single CPU. Adopting a parallel execution model is also non-trivial, given the transactions in a block are inherently sequential.

2.2 Ethereum State Access

Ethereum requires authenticated data structures encoded as modified Merkle-Patricia Tries (MPTs) to store and authenticate its key-value data.

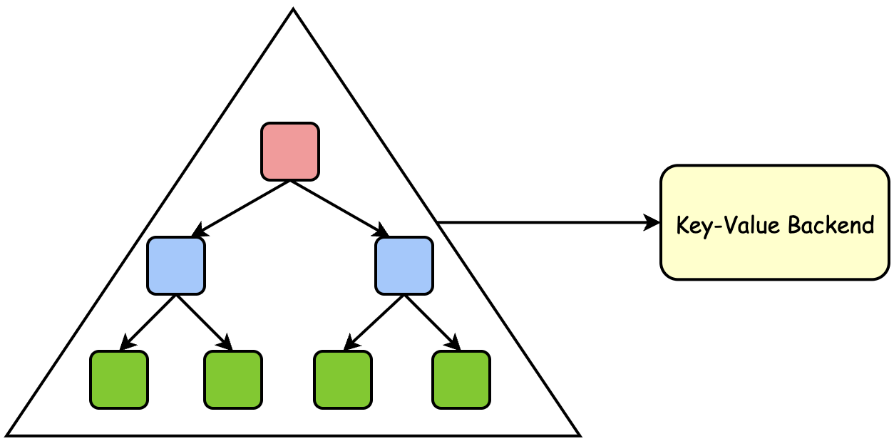


Figure: Key-value data in Ethereum is encoded in an MPT, which is eventually stored in a key-value backend database. The backend database is usually implemented as a B-Tree or LSM-Tree data structure.

Unfortunately, storage access in Ethereum is slow due to several reasons.

- Key-value data isn't stored in the database; it is encoded in an MPT that is stored in the database. Therefore, a read/write of a key in the application layer would be amplified to multiple reads/writes to the database backend. This read/write amplification results in low storage speed, translating into low execution throughput.

- A write operation would require multiple hash re-computations of all inner nodes along the MPT path.

- Lastly but most importantly, the storage I/O is synchronous. Execution threads must wait for costly read/write operations to complete before processing the next transaction.

The impact of this is significant. State bloat leads to an anti-network effect. As the state grows, so does the average trie depth, further slowing storage access. Additionally, the MPT root must be re-computed every block, resulting in a significant portion of the execution time going towards Merkleisation.

2.3 Storage IO

State storage isn't explicit in EVM design; node clients can innovate on storage solutions. LevelDB, PebbledDB, and MDBX are all key-value stores used in Ethereum and Layer 2 clients, although they are not natively authenticated key-value stores (AKVS). A native AKVS solution would have its advantages, though no suitable FOSS solutions are available today.

2.4 Block Size

An obvious limitation of Layer 2s is the block gas limit. This configuration can easily be increased if the system can support higher throughput. However, the increased block limit may impact the fraud-proof system. If challengers are to submit an entire block for fraud-proofs, they risk running into Ethereum block size limitations.

2.5 Layer 2 Block Productions

All L2 clients began as Ethereum clients and often hold onto L1 concepts. The block production pipeline for an L2 is inherently different to the L1. This offers opportunities to optimise and parallelise the block production pipeline in the L2 sequencer and execution client.

2.6 Interoperability and Decentralisation

Interoperability through shared sequencing will be a significant unlock for L2s, and decentralising sequencing is also an important step. Each introduces restrictions to the execution and sequencer that must be considered when building a future-proof system.

3 RISE Stack Design

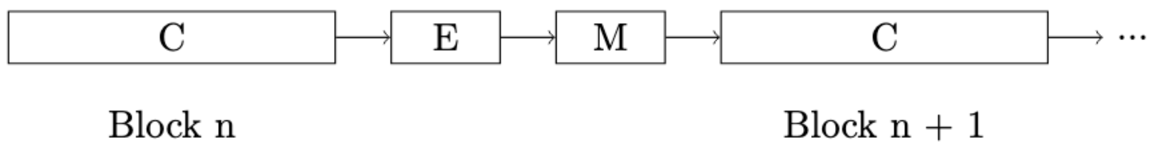
The RISE Stack builds on the OP Stack with Reth as the execution client. This allows us to start with a robust foundation and focus on performance.

3.1 Continuous Block Pipeline

L2 block production consists of the following sequential steps [13]:

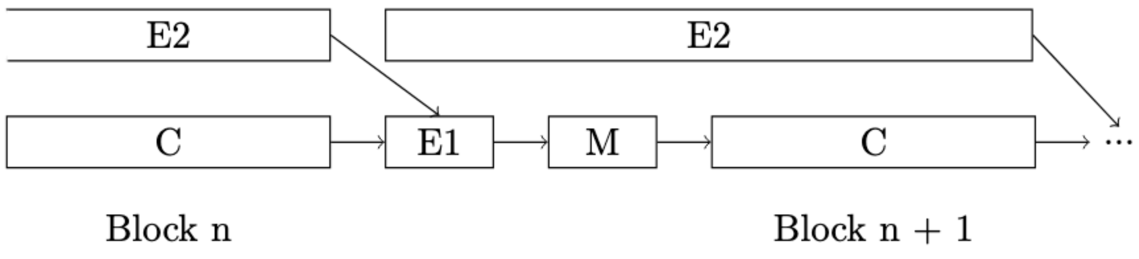
1. C: Consensus deriving L1 transactions and new block attributes
2. E: Execute derived L1 transactions (1) and L2 transactions from mempool (2)
3. M: Merkelize the chain state and execution results to complete the block header

Typically, these steps are sequential; the figure below illustrates a typical block pipeline for an L2 with a one-second block time and large state. The block gas limit will be limited by how much gas can be processed in the time allocated to execution; however, in this system, execution is allowed only a minority of the block-building pipeline. In practice, C consumes 40-80% of the small block time, and as the state grows, M consumes up to 60% of the remaining execution time (12-36% of the total block time). In a worst-case scenario, E will only have 8% of the block time to execute, which is very inefficient, and block gas limits need to consider the worst-case scenario closely to avoid block time drift and reorgs.



Sequential Block Pipeline

RISE introduces the Continuous Block Pipeline (CBP), a parallelised block pipeline with concurrent stages and a Continuous Execution (CE) thread. The CE thread monitors the Mempool for L2 transactions and executes in multiple block segments, no longer waiting for consensus to request a new block. Recall the worst-case scenario of 8% execution; in comparison, CBP executes transactions close to 100% of the available time. Merkelization also occurs concurrently with execution.



Continuous Block Pipeline (CBP)

3.2 Parallel EVM

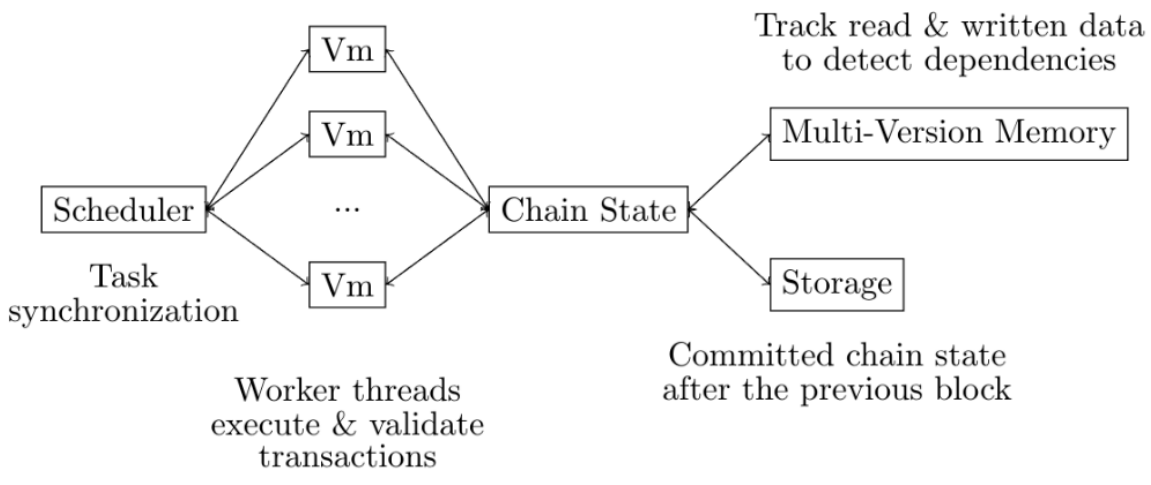
Parallelising execution to accelerate blockchain throughput has gained popularity thanks to Aptos, Sui, and Solana. However, it is still unfruitful in the EVM land. Early adaptions of Block-STM[13] by Polygon and Sei have shown limited speedup due to the lack of EVM-specific optimisations and implementation limitations. Both Polygon and Sei use Go, a garbage-collected language unsuitable for optimisations at the micro-second scale. Parallel execution in Go is mostly slower than sequential execution in Rust and C++ for Ethereum mainnet blocks today.

RISE pevm sets to address this problem by designing an EVM-specialized parallel executor and implementing it in Rust to minimise runtime overheads. The result is the fastest EVM block executor, with a peak speedup of 22x and a raw execution throughput of 55 Gigagas/s on 32 AWS Graviton3 CPUs. RISE pevm also enables new parallel dApp designs for EVM like Sharded AMM [17].

Design

Blockchain execution must be deterministic so that network participants agree on blocks and state transitions. Therefore, parallel execution must arrive at the same outcome as sequential execution. Having race conditions that affect execution results would break consensus.

RISE pevm builds on Block-STM's optimistic execution. We also use a collaborative scheduler and a multi-version data structure to detect state conflicts and re-execute transactions accordingly.


Track read & written data
to detect dependencies
Vm
Vm
...
Vm
Scheduler
Task synchronization
Chain State
Multi-Version Memory
Storage
Committed chain state after the previous block
Worker threads execute & validate transactions

We made several contributions fine-tuned for EVM. For instance, all EVM transactions in the same block read and write to the beneficiary account for gas payment, making all transactions interdependent by default. RISE pevm addresses this by lazy-updating the beneficiary balance. We mock the balance on gas payment reads to avoid registering a state dependency and only evaluate it at the end of the block or when there is an explicit read. We apply the same technique for common scenarios like raw ETH and ERC-20 transfers. Lazy updates help us parallelise transfers from and to the same address, with only a minor post-processing latency for evaluating the lazy values.

While others embed their parallel executor directly into their nodes, we built a dedicated repository to serve as a playground for further pevm R&D. The codebase is open-source at https://github.com/risechain/pevm.

3.2.1 Mempool Preprocessing

Unlike previous rollups that ordered transactions by first-come-first-served or gas auctions, RISE innovates a new Mempool structure that balances latency and throughput. The goal is to pre-order transactions to minimise shared states and maximise parallel execution. This has a relatively similar effect as the local fee market on Solana, where congested contracts & states are more expensive regarding gas & latency. Since the number of threads to execute transactions is much smaller than our intended TPS, we can still arrange dedicated threads to execute high-traffic contract interactions sequentially and others in parallel in other threads.

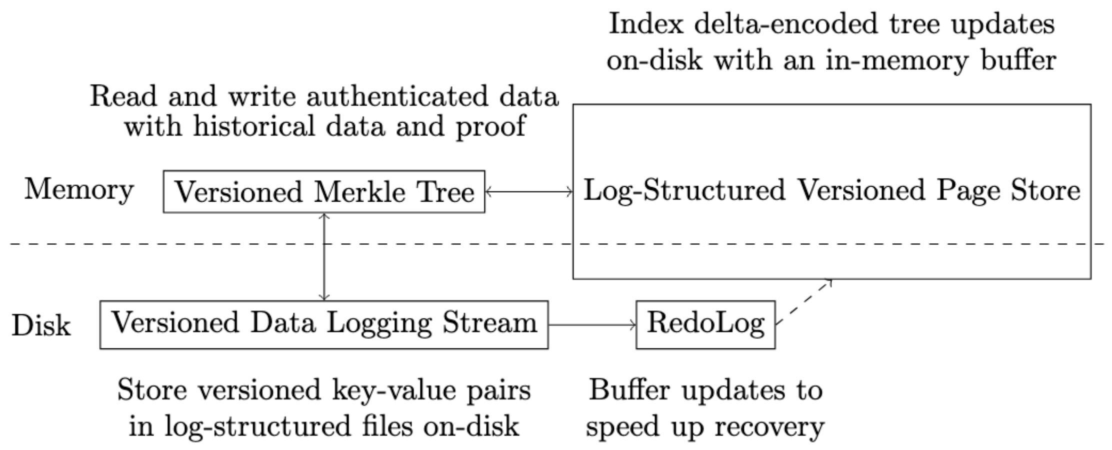
3.3 Versioned Merkle Tree & RiseDB

Recall Ethereum State Access significantly impacts throughput; our initial solution is to store the entire MPT in memory to reduce read/write overhead. Unfortunately, the chain state can grow enormous as we process hundreds of thousands of transactions per second. As the state grows, storing the entire MPT in memory is not feasible for full nodes.

Instead, we replace Ethereum's MPT with a much more storage-efficient Versioned Merkle Tree. Versioned Merkle Trees use a version-based node key so all keys are naturally sorted on insertion. This minimises the compaction cost of the underlying key-value storage. Versioned trees also enable robust and efficient historical queries and proofs.

A popular Versioned Merkle Tree is the Jellyfish Merkle Tree[14] designed for Aptos. While JMT minimises the key-value storage compaction with the versioned node keys, it still uses a general-purpose RocksDB with high write amplification. Compaction is also still triggered during pruning.

To address this, we take the LETUS[15] approach of replacing the two-layered storage architecture with a streamlined one. Tree updates are delta-encoded and the key-value pairs are stored in log-structured files (not trees) for minimal read & write amplification.


Index delta-encoded tree updates on-disk with an in-memory buffer
Read and write authenticated data with historical data and proof
Memory
Versioned Merkle Tree
Log-Structured Versioned Page Store
Disk
Versioned Data Logging Stream
RedoLog
Store versioned key-value pairs in log-structured files on-disk
Buffer updates to speed up recovery

Finally, we experiment with ideas from NOMT[16] where we lower the radix of the Merkle tree to reduce merkelization complexity. Regarding the increased read complexity, we use an optimal on-disk representation to parallelly fetch all sub-trees for a key in one SSD roundtrip. NOMT can go down to binary as each node is only 32 bytes. This doesn't apply to Versioned Merkle Trees as they must store the extra version. We must also store the offset and value size to index the value in the log-structured files. The final design could be an 8-ary tree to balance between read and write amplification.

3.4 Additional Improvements

3.4.1 Networking

The sequencer mempool will be connected to thousands of nodes, and each connection has overheads. The RISE network employs the QUIC protocol, an alternative to traditional TCP that offers faster connection setups and reduced latency, among other features.

3.4.2 JIT Compilation

Just-in-time allows for native VM execution rather than execution via an interpreter. RISE pevm is EVM executor-agnostic and can switch between a JIT like https://github.com/paradigmxyz/revmc and an interpreter one like https://github.com/bluealloy/revm depending on the block and deployment environment.

4 Conclusion

The development of RISE represents a significant stride forward in the evolution of blockchain scalability and performance. This whitepaper has explored the inherent challenges of existing Layer 2 technologies within the Ethereum ecosystem and introduced RISE as a transformative solution designed to surpass these limitations. By implementing a parallel EVM, refined execution processes, and innovative database designs, RISE targets an ambitious throughput of 100,000 TPS, setting a new blockchain technology standard and entering the Gigagas Era.

The ongoing development of RISE aims to meet current demands and anticipate and innovate ahead of future challenges in the blockchain space. We invite the community to join us in this exciting journey as we work towards building a more scalable, efficient, and inclusive blockchain ecosystem.

Acknowledgements

We extend our gratitude to those who have paved the way for advancements in Ethereum scalability and roll-up technology. We acknowledge the foundational work and significant contributions of the teams at Arbitrum and Optimism. Their pioneering efforts in optimistic rollups proved rollups to be a viable scaling option for Ethereum.

We also thank the Reth team for building the execution client and SDK on which RISE is being constructed. The modularity and performance Reth offers are crucial to RISE, as the team is forward-thinking leaders. We look forward to paying it forward by making meaningful contributions to the Reth project.

References

[1] Buterin, V. (2014). Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. Retrieved from https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf

[2] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved from https://bitcoin.org/bitcoin.pdf

[3] Buterin, V. (2021, December 6). Endgame. Retrieved from https://vitalik.eth.limo/general/2021/12/06/endgame.html

[4] Understanding the Ethereum Development Roadmap, Retrieved April 10, 2024, from https://www.blocknative.com/blog/ethereum-roadmap-guide

[5] Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S. M., & Felten, E. W. (2018). Arbitrum: Scalable, Private Smart Contracts. Princeton University. Retrieved from https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-kalodner.pdf

[6] TPS, Max TPS, TTF, Block Time & Governance Model. (2024). Retrieved April 14, 2024, from https://chainspect.app/dashboard?range=30d

[7] Buterin, V. [@VitalikButerin]. (2024, March 29). What does this mean more broadly? I argue that now that the merge and EIP-4844 are done, we are decidedly on the right side of the S-curve. Further L1 changes will be quite meaningful and significant, but relatively milder. [X Post]. Retrieved from https://x.com/VitalikButerin/status/1773356271300706507

[8] Value Locked. (2024). Retrieved April 10, 2024, from https://l2beat.com/scaling/summary

[9] Buterin, V., & Beiko, T. (2024). EIP-4844: Shard Blob Transactions. Ethereum Improvement Proposals. Retrieved April 2, 2024, from https://eips.ethereum.org/EIPS/eip-4844

[10] Electric Capital. (2023). 2023 Crypto Developer Report.

[11] Choi, J., Beillahi, S., Singh, S., Michalopoulos, P., Li, P., Veneris, A., & Long, F. (2022). LMPT: A Novel Authenticated Data Structure to Eliminate Storage Bottlenecks for High Performance Blockchains. IEEE.

[12] Ethereum Optimism. (2024). Optimism: Optimistic Ethereum. Retrieved from https://github.com/ethereum-optimism/optimism

[13] Gelashvili, R., Spiegelman, A., Xiang, Z., Danezis, G., Li, Z., Malkhi, D., Xia, Y., & Zhou, R. (2022). Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing. arXiv preprint arXiv:2203.06871. https://doi.org/10.48550/arXiv.2203.06871

[14] Zhenhuan Gao, Yuxuan Hu, Qinfan Wu (2021). Jellyfish Merkle Tree https://developers.diem.com/papers/jellyfish-merkle-tree/2021-01-14.pdf

[15] Shikun Tian, Zhonghao Lu, Haizhen Zhuo, Xiaojing Tang, Peiyi Hong, Shenglong Chen, Dayi Yang, Ying Yan, Zhiyong Jiang, Hui Zhang, and Guofei Jiang (2024). LETUS: A Log-Structured Efficient Trusted Universal BlockChain Storage https://doi.org/10.1145/3626246.3653390

[16] Preston Evans (2024). Nearly Optimal State Merklization https://sovereign.mirror.xyz/jfx_cJ_15saejG9ZuQWjnGnG-NfahbazQH98i1J3NN8

[17] Hongyin Chen, Amit Vaisman, Ittay Eyal (2024). SAMM: Sharded Automated Market Maker https://arxiv.org/abs/2406.05568