Frequency-Aware Skewed Merkle Trees for Blockchain Optimization: Performance Benefits and Fundamental Limitations

A Comprehensive Study

Research Analysis Study
University of Neuchâtel
Blockchain Systems Research
Switzerland

Abstract

This study presents a comprehensive analysis of frequency-aware skewed Merkle tree structures as a potential optimization for blockchain systems. We investigate the application of skewed tree architectures that exploit transaction frequency patterns to improve performance in blockchain networks. Through systematic analysis of 21 research papers spanning dynamic integrity trees, traditional Merkle tree optimizations, sparse data structures, and blockchain implementations, we identify significant performance benefits (up to 30% improvement) alongside critical fundamental limitations. Our key finding demonstrates that while frequency-based skewed structures provide substantial performance gains, they violate core blockchain requirements including state reversibility, order independence (commutativity), and deterministic consensus properties. This study establishes the theoretical boundaries for frequency-based optimizations in distributed consensus systems.

Keywords: Blockchain, Merkle Trees, Performance Optimization, Distributed Systems, Consensus Protocols, Frequency Analysis

1 Introduction

Blockchain technology has revolutionized distributed computing by providing decentralized consensus mechanisms without requiring trusted intermediaries [1]. At the core of most blockchain implementations lie Merkle trees [2], which provide efficient cryptographic verification of large data structures while enabling lightweight client operations through compact proofs.

However, current Merkle tree implementations in blockchain systems face significant performance challenges that become critical bottlenecks as networks scale. As documented in our research logs, the primary bottleneck stems from the database access pattern where "hash values are stored as keys in the database," resulting in O(log n × log n) time complexity instead of the theoretical O(log n) for tree traversal operations. This inefficiency becomes particularly problematic as blockchain networks scale to handle millions of transactions - Ethereum, for example, processes over 1 million transactions daily, each requiring multiple tree operations for state verification and updates.

An intriguing observation in blockchain systems is the highly skewed nature of transaction patterns. Similar to other distributed systems, blockchain networks exhibit access patterns resembling a 1/x function, where a small percentage of accounts generate the majority of transactions [3]. This natural frequency distribution suggests potential for optimization through frequency-aware data structures.

This study investigates the feasibility of applying frequency-aware skewed Merkle tree structures to optimize blockchain performance. We systematically analyze existing research on dynamic skewed integrity trees [3], frequency-aware structures for embedded systems [4], memory-efficient adaptive algorithms [5], and traditional Merkle tree optimizations [7].

Our research reveals a fundamental trade-off between performance optimization and correctness requirements in distributed consensus systems. While skewed structures demonstrate significant performance improvements, they introduce critical limitations that render them unsuitable for blockchain applications requiring state reversibility, order independence, and deterministic consensus properties.

2 Related Work

2.1 Dynamic Skewed Integrity Trees

Vig et al. [3] proposed dynamic skewed integrity trees for memory authentication in embedded systems. Their approach dynamically restructures tree nodes based on runtime access patterns, positioning frequently accessed memory blocks closer to the root. The system achieved an average performance improvement of 30% compared to balanced trees through adaptive migration and rebalancing operations similar to adaptive Huffman coding.

Their experimental setup used Altera NIOS II processor with external DRAM, testing on CHStone and SNU Real-Time benchmarks. The results showed a 35% reduction in tree levels accessed and maintained near-optimal performance even in worst-case scenarios. The key innovation involves grouping data elements with identical access frequencies into single nodes, using node weights $w_i = NE_i \times f_i$ where NE_i is the number of elements and f_i is the access frequency.

The rebalancing criterion triggers when $(w_i > w_{sibling}+1) \wedge (w_i > w_{uncle})$, ensuring frequently accessed nodes migrate toward the root. This approach dramatically reduces the number of tree nodes from O(n) symbols to O(frequency_classes), with set migration operations moving symbols between frequency classes as access patterns evolve.

2.2 Frequency-Aware Skewed Merkle Trees

Zou and Lin [4] developed FAST (Frequency-Aware Skewed Merkle Tree) for embedded systems. Their approach profiles application memory access patterns during a simulation phase, then generates application-specific optimal skewed Merkle trees using Huffman encoding principles. They tested on five real-world benchmarks including sorting algorithms, binary search, and neural networks using gem5 simulator with Xilinx KC705 parameters.

FAST achieved up to 3× performance improvement over balanced Merkle trees by configuring the branching factor k to provide a tunable trade-off between computational complexity and bandwidth efficiency. Their results showed that k=32 provides 5× bandwidth reduction with 8.4ms construction time per leaf, while k=1024 achieves 10× bandwidth reduction at 110.1ms per leaf. The system distinguishes between Parallel Verification (PV) operations with weight=1 and Chained Update (CU) operations with weight=5, reflecting the higher cost of sequential hash computations in write operations versus parallelizable verification.

2.3 Memory-Efficient Adaptive Huffman Coding

Pigeon and Bengio [5] introduced memory-efficient adaptive Huffman coding algorithms for large symbol alphabets. Their Algorithm M uses set-based nodes containing symbols with identical frequencies, reducing memory requirements from O(n) individual symbols to O(frequency_classes). Testing on the Calgary Corpus, they achieved dramatic memory savings: for 8-bit symbols, traditional Huffman requires 511 nodes while Algorithm M averaged only 192 nodes (37.6%

reduction). For 16-bit symbols, the improvement was even more striking - from 131,071 nodes down to just 360.6 nodes on average.

The algorithm maintains entropy bounds of [H(S), H(S)+2) and demonstrates particular effectiveness for large alphabets (millions vs. hundreds of symbols). Despite the memory efficiency, compression performance remained competitive: Algorithm M achieved 5.07 bits/symbol compared to 4.75 for Vitter's Algorithm on 8-bit data. Key operations include set migration when symbols change frequency classes and rebalancing using shift-up procedures adapted from AVL tree algorithms.

2.4 Traditional Merkle Tree Optimizations

Buchmann et al. [7] proposed improved algorithms for Merkle tree traversal in signature schemes. Their approach balances the number of leaves computed in each authentication path computation rather than balancing the total number of nodes, since leaf computation requires significantly more hash function evaluations than inner nodes.

Other optimization approaches include fractal Merkle tree traversal [8], which splits trees into smaller subtrees with stacked series for multiple authentication paths, and caching strategies that store frequently accessed nodes in memory [17].

2.5 Alternative Data Structures

Sparse Merkle Trees: Dahlberg et al. [10] develop efficient sparse Merkle trees supporting 2^N possible keys with constant-size proofs for membership and non-membership verification. With strategic caching and memory management, they report sub-4 ms proof generation.
Verkle Trees: Kuszmaul [11] introduces Verkle trees as bandwidth-efficient alternatives to Merkle trees. By replacing hash functions with vector commitments, Verkle trees achieve proof sizes of $\mathcal{O}(\log_k n)$ (versus $\mathcal{O}(\log n)$), providing up to 10× bandwidth reduction with manageable computational overhead.
B^ε **Trees**: Bender et al. [12] present write-optimized B^ε-trees that buffer operations to amortize I/O costs. Through batched message flushing, these structures achieve orders-of-magnitude faster insert performance than traditional B-trees.

3 Methodology

Our research methodology involved systematic analysis of 21 research papers across four key areas:

1. **Core Skewed Tree Research**: Dynamic integrity trees [3], frequency-aware structures [4], and adaptive Huffman coding [5, 6]

2. **Traditional Merkle Optimizations**: Traversal algorithms [7], fractal approaches [8], and optimal trade-offs [9]

3. **Alternative Data Structures**: Sparse Merkle trees [10], Verkle trees [11], B^ε trees [12], and hash grids [13]

4. **Blockchain Context**: Bitcoin [1], Ethereum [14, 15], and implementation analysis [16]

We also implemented and analyzed a Merkle Patricia Trie prototype to understand practical performance characteristics and identify implementation challenges in frequency tracking and tree restructuring operations.

4 Fundamental Limitations of Frequency-Based Structures

Our analysis reveals three critical limitations that render frequency-aware skewed structures unsuitable for blockchain applications:

4.1 State Reversibility Violation

Blockchain systems require the ability to "undo" transactions to reach previous states for fork resolution, smart contract failures, and network consensus. In balanced Merkle trees, each node position is deterministic based on data content, enabling efficient rollback operations.

However, in skewed trees, node positions depend on access history and frequency patterns. Consider an example where accounts A, B, C have frequencies (5, 3, 1) creating a specific tree structure. After processing transactions that change frequencies to (5, 4, 2), the tree restructures. To rollback, the system must not only restore account balances but also reconstruct the exact frequency distribution that existed before, requiring storage proportional to O(N $\times$ history_depth) where N is the number of accounts.

This creates a fundamental scaling problem: as blockchain history grows, the storage required for frequency state history grows linearly, making long-term operation prohibitively expensive. Unlike content-based trees where positions are always derivable from current state, frequency-based positions require maintaining complete access pattern history.

4.2 Order Independence (Commutativity) Violation

A fundamental blockchain requirement is that transaction processing order must not affect the final tree structure. For transactions T1 and T2, applying T1→T2 must produce identical tree structure as T2→T1, assuming equivalent final account states.

Frequency-based trees violate this property because:

Consider a concrete example with initial state having accounts A, B, C with frequencies (0, 0, 0). Two transactions occur:

- T1: Transfer A→B (accesses A, B)
- T2: Transfer C→A (accesses C, A)

Order 1 (T1 → T2):
After T1: frequencies = (A:1, B:1, C:0)
Tree places A, B near root based on equal frequency
After T2: frequencies = (A:2, B:1, C:1)
Final tree prioritizes A at top level

Order 2 (T2 → T1):
After T2: frequencies = (A:1, B:0, C:1)
Tree places A, C near root (tie-breaking affects structure)
After T1: frequencies = (A:2, B:1, C:1)
Final tree has different structure despite same final frequencies

This demonstrates how identical final account states can produce different tree structures based on processing order, violating the fundamental blockchain requirement that final state should be independent of transaction ordering.

This order dependency creates consensus failure scenarios where different nodes processing identical transactions in different orders reach different tree structures, breaking blockchain's deterministic consensus requirements.

4.3 Consensus Safety Violations

The combination of frequency dependency and order sensitivity creates multiple attack vectors that fundamentally compromise blockchain security:

- **Frequency Manipulation**: Attackers can create dummy accounts and generate artificial transactions to inflate target account frequencies. For example, an attacker could make 1000 micro-transactions between two accounts to artificially boost their frequency counts, forcing the tree to restructure and placing the attacker's accounts closer to the root for faster access.
- **Information Leakage**: Tree structure becomes a side-channel that reveals access patterns. High-frequency accounts near the root can be identified as likely exchange wallets or popular contracts, while sudden structural changes indicate shifts in user behavior or business operations.
- **DoS via Rebalancing**: Each frequency change can trigger O(log k) node relocations and hash recomputations. An attacker can deliberately target accounts near rebalancing thresholds, forcing expensive tree restructuring operations that consume significant computational resources across the network.
- **Persistent Consensus Splits**: Unlike temporary network partitions, frequency-based consensus failures cannot be resolved through longest-chain rules since both chains may be equally valid but structurally incompatible. This creates permanent network splits that cannot be automatically reconciled.

5 Performance Analysis

Despite fundamental correctness issues, frequency-aware structures demonstrate significant performance benefits in appropriate contexts:

5.1 Empirical Performance Gains

Our analysis of existing implementations shows substantial performance benefits across different approaches:

- **Dynamic Skewed Trees**: 30% average runtime improvement with 35% reduction in tree levels accessed [3]. Performance gains varied by application: high-skew access patterns achieved up to 45% improvement while uniform patterns still saw 10-15% gains.
- **FAST Architecture**: Up to 3× improvement with hardware optimization, achieving 40% reduction in tree traversals [4]. The bandwidth reduction scales with branching factor: k=32 provides 5× bandwidth reduction while k=1024 achieves 10× reduction.
- **Memory Efficiency**: Algorithm M reduced node count from 511 to 192 nodes on average (37.6% of traditional) for 8-bit alphabets, and from 131K to 360 nodes for 16-bit alphabets [5].
- **Access Pattern Exploitation**: Performance improvements are most pronounced with highly skewed access patterns following Pareto distributions, where 20% of accounts generate 80% of accesses.

5.2 Theoretical Complexity Analysis

For a blockchain with N accounts following a skewed access pattern:

- **Traditional Merkle Trees**: O(log N) per operation, O(log N × log N) with database overhead
- **Frequency-Aware Trees**: O(log k) per operation for k frequency classes where k ¡¡ N
- **Migration Overhead**: O(log k) per frequency update with rebalancing costs
- **Rollback Complexity**: O(N × history_depth) for maintaining frequency state history

The performance benefits become more pronounced as the skewness of the access pattern increases. For instance, in systems where 80% of accesses target just 20% of accounts (following Pareto distribution), frequency-aware trees can achieve 2-3x performance improvements. However, the rollback complexity of O(N × history_depth) makes this approach impractical for blockchain systems requiring frequent state transitions - a blockchain with 1 million accounts and 1000 blocks of history would require maintaining 1 billion frequency state entries for complete reversibility.

6 Conclusions

This study demonstrates that while frequency-aware skewed Merkle tree structures provide substantial performance improvements (30-50% in various implementations), they are fundamentally incompatible with blockchain system requirements. The core limitations include:

1. **State Reversibility Violation**: Cannot efficiently support transaction rollback operations essential for blockchain consensus and error recovery
2. **Order Independence Violation**: Transaction processing order affects tree structure, breaking deterministic consensus requirements
3. **Consensus Safety Risks**: Multiple attack vectors and consensus failure scenarios emerge from frequency-dependent behavior

These findings establish important theoretical boundaries for optimization approaches in distributed consensus systems. Performance improvements that compromise safety or determinism properties cannot be applied to blockchain systems regardless of their efficiency benefits.

This study contributes to the broader understanding of trade-offs between performance optimization and correctness guarantees in distributed systems. The findings demonstrate that frequency-based optimizations, while achieving impressive performance gains (30-300% in various contexts), are fundamentally incompatible with the deterministic requirements of blockchain consensus protocols.

Key Implications: The research establishes that not all performance improvements are suitable for blockchain systems, even when they provide substantial benefits in other contexts. The deterministic nature of blockchain consensus creates constraints that may not be obvious from single-node performance analysis.

Future Directions: While frequency-aware optimizations are unsuitable for consensus-critical blockchain components, they may still be applicable in layer-2 systems, application-specific blockchains with relaxed consistency requirements, or non-consensus components like local caching layers where the trade-offs between performance and determinism can be evaluated differently.

Acknowledgment

This study was conducted as part of blockchain systems optimization research at the University of Neuchâtel. We acknowledge the comprehensive literature analysis spanning multiple domains of tree data structures, cryptographic protocols, and distributed systems theory.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Cryptography Mailing list at https://metzdowd.com, March 2009.

[2] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology — CRYPTO '87*, C. Pomerance, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 369–378.

[3] S. Vig, G. Jiang, and S.-K. Lam, "Dynamic skewed tree for fast memory integrity verification," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 642–647.

[4] Y. Zou and M. Lin, "FAST: A frequency-aware skewed merkle tree for FPGA-secured embedded systems," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 268–273.

[5] S. Pigeon and Y. Bengio, "A memory-efficient adaptive huffman coding algorithm for very large sets of symbols," in *Proceedings DCC '98 Data Compression Conference*. IEEE, 1998, pp. 568.

[6] S. Pigeon and Y. Bengio, "A memory-efficient adaptive huffman coding algorithm for very large sets of symbols revisited," Université de Montréal, Rapport technique #1095, 1998.

[7] J. Buchmann, E. Dahmen, and M. Schneider, "Merkle tree traversal revisited," in *Post-Quantum Cryptography*. Springer, 2008, pp. 63–78.

[8] M. J. Jacobson Jr., A. Menezes, and A. Stein, "Solving elliptic curve discrete logarithm problems using Weil descent," *Journal of the Ramanujan Mathematical Society*, vol. 16, no. 4, pp. 231–260, 2001.

[9] M. Szydlo, "Merkle tree traversal in log space and time," in *Advances in Cryptology–EUROCRYPT 2004*. Springer, 2004, pp. 541–554.

[10] R. Dahlberg, T. Pulls, and R. Peeters, "Efficient sparse merkle trees: Caching strategies and secure (non-)membership proofs," in *Nordic Conference on Secure IT Systems*. Springer, 2016, pp. 199–215.

[11] J. Kuszmaul, "Verkle trees," Bachelor's thesis, MIT, 2018.

[12] M. A. Bender, M. Farach-Colton, W. Jannen, R. Johnson, B. C. Kuszmaul, D. E. Porter, J. Yuan, and Y. Zhan, "An introduction to B$^{\varepsilon}$-trees and write-optimization," *;login:*, vol. 40, no. 5, pp. 22–28, 2015.

[13] J.-F. Pâris and T. Schwarz, "Merkle hash grids instead of merkle trees," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, pp. 253–258.

[14] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum project yellow paper, 2014.

[15] V. Buterin, “Merkling in ethereum,” Ethereum Foundation Blog, Nov. 2015. [Online]. Available: https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/

[16] H. Sáez de Ocáriz Borde, “An overview of trees in blockchain technology: Merkle trees and merkle patricia tries,” Preprint, Feb. 2022.

[17] B. Gassend, G. E. Suh, D. Clarke, M. Van Dijk, and S. Devadas, “Caches and hash trees for efficient memory integrity verification,” in *The Ninth International Symposium on High-Performance Computer Architecture*. IEEE, 2003, pp. 295–306.