OPT-MorphDAG: An efficient implementation of DAG-Based Blockchain

Anish Gupta
Student- 3rd year
B.Tech - Computer Science and Engineering
SoE, Shiv Nadar University
ag801@snu.edu.in

Dr. Sweta Kumari
Assistant Professor
Dept. of Computer Science and Engineering
Shiv Nadar University
sweta.kumari@snu.edu.in

Suryansh Rohil
Student- 3rd year
B.Tech - Computer Science and Engineering
SoE, Shiv Nadar University
sr738@snu.edu.in

Dr. Archit Somani
Assistant Professor
Dept. of Computer Science and Engineering
Shiv Nadar University
archit.somani@snu.edu.in

***Abstract*—Blockchain technology is revolutionizing the way we store data, ensure security, and manage decentralized transactions across a wide range of industries including finance, supply chain management, healthcare, and education. However, conventional blockchains like Bitcoin, Ethereum, and Hyperledger rely on fundamental structures (linked lists/Merkle trees) that struggle with throughput and dynamic workload. Directed Acyclic Graph(DAG) based blockchains mark a shift from these designs while enabling significantly higher throughput through concurrent storage and execution. MorphDAG is a recent workload-aware DAG-based blockchain designed to improve throughput and storage efficiency. In this work, we propose optimised MorphDAG (OPT-MorphDAG) in which we replace orignal's MorphDAG's coarse operation-count threshold with a per-account read/write frequency analysis, which leads to better transaction classification enhancing throughput. Integrated into the MorphDAG prototype and tested with a randomly generated set of transactions based on synthethic workload of smallbank benchmark, this refinement yields a reduction in block-processing latency and an increase in throughput, with no data inconsistencies over a variable number of transactions. Numerous tests show that OPT-MorphDAG can increase end-to-end throughput by up to 2.5× and 1.4× over the a serial execution baseline and original MorphDAG, respectively.**

***Index Terms*—Blockchain, Directed Acyclic Graph (DAG), MorphDAG, Throughput, Threshold, Concurrency, Parallelism.**

I. Introduction

Blockchain technology underpins a wide range of decentralized applications (dApps), from permissionless cryptocurrencies to permissioned ledgers for supply chain, healthcare, etc. Its core innovations are immutability, transparency, and decentralized consensus derive from chaining cryptographically linked blocks, as in Bitcoin's linked-list/merkle tree based structure [7], or Merkle tree based designs such as Hyperledger Fabric [3]. However, these traditional architectures suffer from inherent limitations:

- **Throughput bottlenecks**: Sequential block creation and strict ordering limit transactions per second (TPS).
- **Reduced parallelism:** Since each block must be processed in a fixed order, even transactions that don't interfere with each other are often serialized, missing opportunities to run them in parallel.
- **Scalability issues:** As more transactions are added, the chain grows linearly, increasing storage needs and slowing down verification and synchronization across nodes.

Blockchains based on Directed Acyclic Graphs (DAGs) solve these issues by enabling the simultaneous appending of multiple blocks, which increases parallelism and throughput. IOTA's Tangle [16], Nano's [48] block-lattice, and hybrid strategies like LightDAG [26] and MorphDAG [50] are notable DAG systems. In particular, MorphDAG combines an *elastic storage concurrency* model dynamically tuning the number of parallel block producers via Proof of Stake (PoS) based Verifiable Random Function(VRF) sortition with a dual-mode transaction processor as made by MorphDAG developers that handles "hot" and "cold" data differently to mitigate conflicts under skewed workloads.

MorphDAG classifies transactions as hot or cold wether they touch an hot account which is identified based on a simple threshold of total read/write operations per account. This method helped MorphDAG improve its performance, showing up to a **2.4×** increase in throughput compared to earlier DAG-based systems like OHIE [46] and AdaptChain [40]. While this approach improves over uniform execution, transactions with even more reads and less writes can end up on the "hot" path simply because their total operation count is high even if they touch only less conflict causing accounts and might never actually conflict. We observe that a simple per-account operation count fails to distinguish between transactions touching busy (high-contention) accounts and those accessing rarely used (low-contention) accounts. This leads to unnecessary serialization and underutilized concurrency.

To address this problem in MorphDAG's original hot/cold

transaction missclassification, we propose a **per-account read-/write frequency-based refinement**. Our approach computes `writeFreq` and `readFreq` for each account over the current block's transaction set and designates high-frequency accounts (hot) and low-frequency accounts (cold) using a tunable percentile threshold. Transactions that only touch cold accounts are executed parallely in concurrent groups, while those involving any hot accounts incur lightweight dependency checks.

The Contributions of the Paper are as follows:

- We refine MorphDAG's transaction classification without modifying its DAG structure or dual-mode execution mechanism, by introducing a per-account read/write frequency analysis to more accurately distinguish "hot" and "cold" transactions, making it lightweight and widely applicable.
- We enable higher parallelism by allowing transactions that touch only low-contention accounts to execute fully in parallel while correctly isolating high-contention transactions.
- We evaluate the OPT-MorphDAG using the Smallbank benchmark and show improved scalability and throughput under skewed workloads, with up to **2.5 times improvement in throughput** over the baseline and significant improvement over original MorphDAG too.

The rest of this paper is organized as follows. Section II reviews the relevant literature. Section III details our proposed per-account write frequency–based classification methodology. Section IV describes the implementation of this refinement within the MorphDAG prototype. Section V evaluates the performance and reports our experimental results. Section VI shows our conclusion and future work, followed by a complete list of references.

II. Literature Review

Directed Acyclic Graphs (DAGs) have emerged as a promising alternative to traditional blockchain architectures, offering significant improvements in scalability, latency, and energy efficiency. Unlike traditional linear structures, DAG-based systems allow for parallel transaction processing, which lowers confirmation times and transaction costs in contrast to traditional blockchains' linear chain or outdated structures. Because of this, they are especially well-suited for use cases where high throughput and little delays are essential, like the financial industry, Internet of Things (IoT) industry etc.

One such structure traditional blockchain systems often rely on is a Merkle tree to ensure data integrity. These structures use a binary tree of hashes to verify transactions, and although they offer cryptographic guarantees, their inherently sequential validation flow imposes limitations. In comparison, DAG-based blockchains still prove to be better than Merkle tree in most scenarios due to:

- **Higher scalability** – DAG enable parallel transaction processing, whereas Merkle trees follow a sequential validation approach.
- **Greater throughput**– DAG can efficiently process high transaction volumes, while Merkle trees may create bottlenecks in large networks.
- **Faster confirmations** – Transactions in a DAG validate each other, reducing delays compared to Merkle tree-based pow(proof of work) systems.
- **No reliance on mining** – Some DAG-based networks such as Tangle [16] eliminate the need for mining, making them more efficient.

Many studies have analyzed DAG-based blockchains like Tangle [16], Nano [48], and LightDAG [26], comparing their consensus strategies and transaction validation mechanisms with traditional ones. These comparisons often highlight that, by removing mining and enabling concurrent transaction validation, DAGs achieve better scalability and responsiveness. For example, hybrid designs like JointGraph [12] combine DAG structures with enhanced consensus mechanisms to isolate faulty nodes and increase efficiency, while the Tangle [16]architecture integrates blockchain and DAG to support scalable and flexible IoT transactions. A tabular comparison of some popular DAG based blockchains, MorphDAG and our proposed OPT-MorphDAG is shown in Table I.

To improve concurrency in such parallel execution environments, many blockchain systems incorporate Optimistic Concurrency Control (OCC) [17] . OCC assumes that conflicts are rare and allows speculative execution without locks, detecting conflicts at the commit stage. Common algorithms like Basic Timestamp Ordering (BTO) and Multi-Version Timestamp Ordering (MVTO) support this by using versioned data validation. However, under skewed workloads, OCC may lead to high abort rates, negatively impacting throughput despite its reduced synchronization cost [17] .

Conventional systems like *OHIE* [46] and *AdaptChain* [40] rely on a fixed level of storage concurrency, which leads to queuing delays under high load and wasted resources under low load. In contrast, a notable DAG-based system, **MorphDAG** [50], dynamically adjusts its concurrency level based on the size of the pending transaction pool—scaling up during peak loads to reduce delays and scaling down during lighter loads to avoid redundant block creation. To further optimize execution, MorphDAG calculates the total number of read and write operations on each account across a block and labels an account as "hot" if its operation count exceeds a static threshold. Based on this classification, transactions are tagged as hot (accessing only hot accounts), cold (accessing only cold accounts), or hot-cold (accessing both), allowing for more informed scheduling and improved parallelism.

To schedule execution safely and efficiently, MorphDAG handles hot and hot-cold transactions using dependency-aware scheduling with fine-grained conflict checks, while cold transactions are grouped into concurrent sets based on their account access integrity. This approach significantly reduces the risk of conflicts and enhances throughput by allowing low-contention transactions to execute in parallel.

However, despite its effectiveness, MorphDAG's reliance on total operation counts for account classification limits

TABLE I: Comparison of DAG-based Transaction Processing Technologies

Comparing Factor	DAG Technologies			
	MorphDAG [50]	***Tangle [16]***	***JointGraph [12]***	***OPT-MorphDAG (Proposed)***
Scalability	Higher (Dynamic DAG structure)	High, but orphaned transactions	Moderate	Highest (Selective grouping, low contention)
Transaction Speed	Higher(with less parallelism	Fast, but slows down in some scenarios	High (limited by bottlenecks)	Highest (parallelism with contention-aware grouping)
Storage Efficiency	Elastic storage reduces redundant data	Prone to unbounded DAG growth	Snapshots reduce memory load	Efficient (avoids redundant groups, minimizes overhead)
Latency	Lower (adaptive consensus process)	Low, but varies based on network activity	Low (due to direct voting mechanism)	Lowest (bypasses serialization via hot/cold isolation)
Security Model	Stronger (adaptive conflict resolution)	Secure, but susceptible to network attacks	Strong (malicious nodes removed)	Stronger (reduced contention reduces attack vectors)
Ideal Use Case	Large-scale decentralized applications	IoT and microtransactions	Consortium blockchains	General-purpose + high-contention blockchains

its adaptability to distinguish between read-heavy and write-heavy workloads, which might lead to missclassification of accounts which might reduce the number of transactions eligible for parallel execution and results in unnecessary conflict detection. , MorphDAG [50] orders transactions by access type (Read-only, Incremental Write, Non-Incremental Write) and further applies dependency checks to hotspot data. To find and separate conflicting transactions, JointGraph [12] and AdaptChain [40] use account-based grouping and mapping techniques.

Three access patterns used in MorphDAG are commonly used to classify transactions:

- **Read-only (R):** Transactions that don't change the data and just read it. Parallel execution of these is safe by nature.
- **Incremental Write (IW):** Transactions that update state incrementally (e.g., balance deductions). They require ordering guarantees to preserve correctness but allow more parallelism.
- **Non-Incremental Write (NIW):** Transactions that perform read-modify-write (RMW) operations or overwrite entire state values. These have the highest contention and are most prone to conflicts.

To maximize throughput without compromising correctness, MorphdAG employs **safe scheduling** techniques. Transactions are grouped into concurrent execution sets, ensuring that no conflicting transactions are placed in the same set. MorphDAG [50] uses a dual-mode execution engine: low-contention transactions (cold) are executed in a fast path without runtime conflict detection, while high-contention (hot) transactions undergo stricter dependency checks.

To decide whether a transaction is hot or cold, MorphDAG counts the total number of read and write operations being performed on the account. If this number of operations is higher than a certain threshold (T_{op}), the account is marked as **hot**. Otherwise, it is treated as **cold**. Then transactions are marked as hot if they touch at least one hot account.

- **Hot transactions** go through more careful processing. They are grouped by access type—Read-Only (R), Non-Incremental Write (NIW), and Incremental Write (IW)—and are checked for conflicts at the account level. Incremental writes only apply changes (not full values), which helps reduce conflicts.
- **Cold transactions** are assumed to have very low chances of conflict. So, they are made to run parallely in concurrent groups which are created on basis of access integrity based grouping [50], which makes execution much faster.

However, this approach is still limited. Since it only looks at the number of operations, it can't tell the difference between transactions that touch atually potentially conflicting busy (hot) accounts and those that touch rarely used (cold) ones. As a result, some low-risk transactions may be treated as hot and processed more slowly than necessary.

III. Proposed Methodology

This section presents our methodology for OPT-MorphDAG, explains the limitations in MorphDAG, and state the advantages of our proposed approach.

A. Overview

To address the limitations of the original MorphDAG classification mechanism, which uses a static per-account operation count threshold to distinguish between hot and cold account, we introduce a more precise classification strategy based on per-account write frequency analysis. Our goal is to better separate conflict-prone "hot" transactions from low-contention "cold" ones and thereby enhance overall throughput and parallelism without compromising correctness.

B. Limitations of Operation Count-Based Classification

In MorphDAG [50], each account is labeled as "hot" if its total number of read/write operations being performed on that account exceeds a calculated threshold as shown in Listing 1; otherwise, it is labeled "cold". Then transactions touching at least one hot account are marked as hot and if they touch both hot and cold, they are marked as hot-cold and if they touch only cold accounts, they are marked as cold. However, this method is coarse:

- Transactions with **many operations on a less popular accounts** may be mislabeled as hot, potentially causing conflicts in the hot path which would create more groups and decrease speed as transactions with multiple read operations and even one write on a hot account would be classified as hot and would have to go through intense conflict detection even scheduling the read-read conflicts in different concurrent groups reducing the parallelism that could have been achieved from parallelizing those read-read conflicts .
- Conversely, transactions with **less operations on high-contention accounts** are unnecessarily serialized as cold, As transactions with less operations but most of them

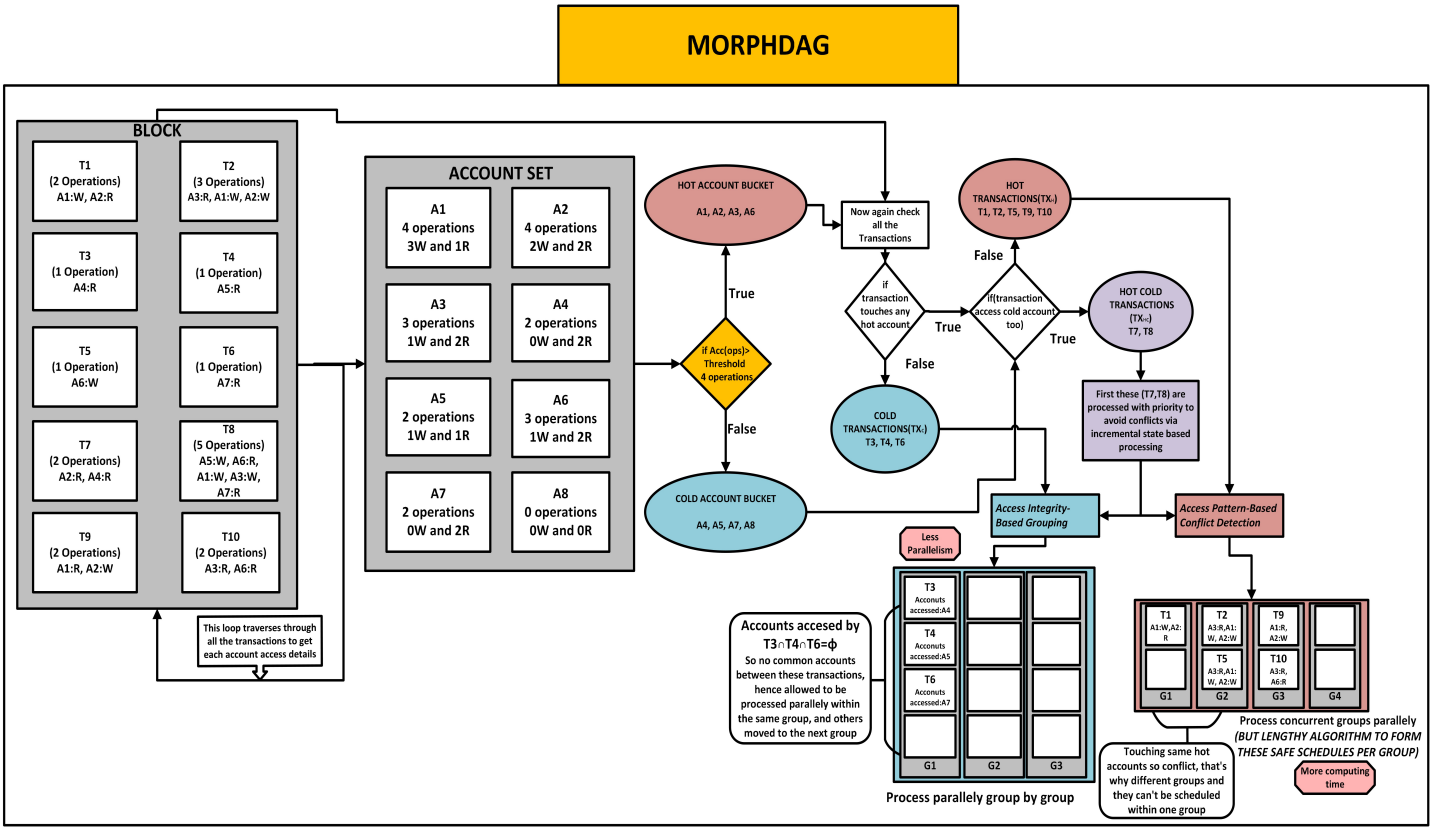

Fig. 1: The original MorphDAG pipeline, showing how transactions flow through the system.

being writes could reduce the parallelism in access integrity based grouping [50] due to very sensitive conflict checking, that could end up creating more groups in the cold path reducing parallelism.

These misclassifications leads to two issues:

1) Increased conflict risk in the cold path (which performs no runtime validation) that reduces parallelism.
2) Underutilized concurrency due to false positives in the hot path.

C. Per-Account Frequency-Based Classification

We address this by utilising per-account write frequency statistics to suggest a dynamic, percentile-based classification of ”hot accounts.” The way the classification operates is as follows:

1) We create two frequency maps for a given set of transactions by iterating through all operations:
   - `readFreq`: quantity of reads for each account `writeFreq`: quantity of writes for each account
2) All account addresses are sorted in descending order by `writeFreq`. A cutoff is caculated based on a configurable ratio parameter k, such that the top $k\%$ of most-written-to accounts are labeled as **hot**.
3) Each transaction is then classified based on whether it touches a hot account:
   - **Hot Transaction**: touches at least one hot account
   - **Cold Transaction**: touches only cold accounts
4) The DAG construction and transaction execution phases remain unchanged, but transactions are now routed more accurately:
   - Cold transactions are assigned to the parallel `coldQueues and groups` and processed with less dynamic and less rigorous conflict checks through *access-integrity based grouping* [50].
   - Hot transactions undergo dependency analysis and conflict resolution via *access-pattern based grouping and incremental state based processing* [50].
5) Working of these grouping and processing methods in MorphDAG [50] are explained as follows:
   - **Incremental State Based Processing**: Prioritising transaction ordering and detecting conflicts based on access patterns prior to transaction execution, MorphDAG targets the extreme data contention on hotspot accounts. It also commits incremental states to significantly lessen read-write conflicts on hotspot accounts.
   - **Access pattern based grouping**: To enable safe parallel execution of hot transactions, this method forms concurrent groups based on fine-grained access pattern classification. Transactions are grouped by checking their compatibility using predefined mapping rules for read (R), incremental write (IW),

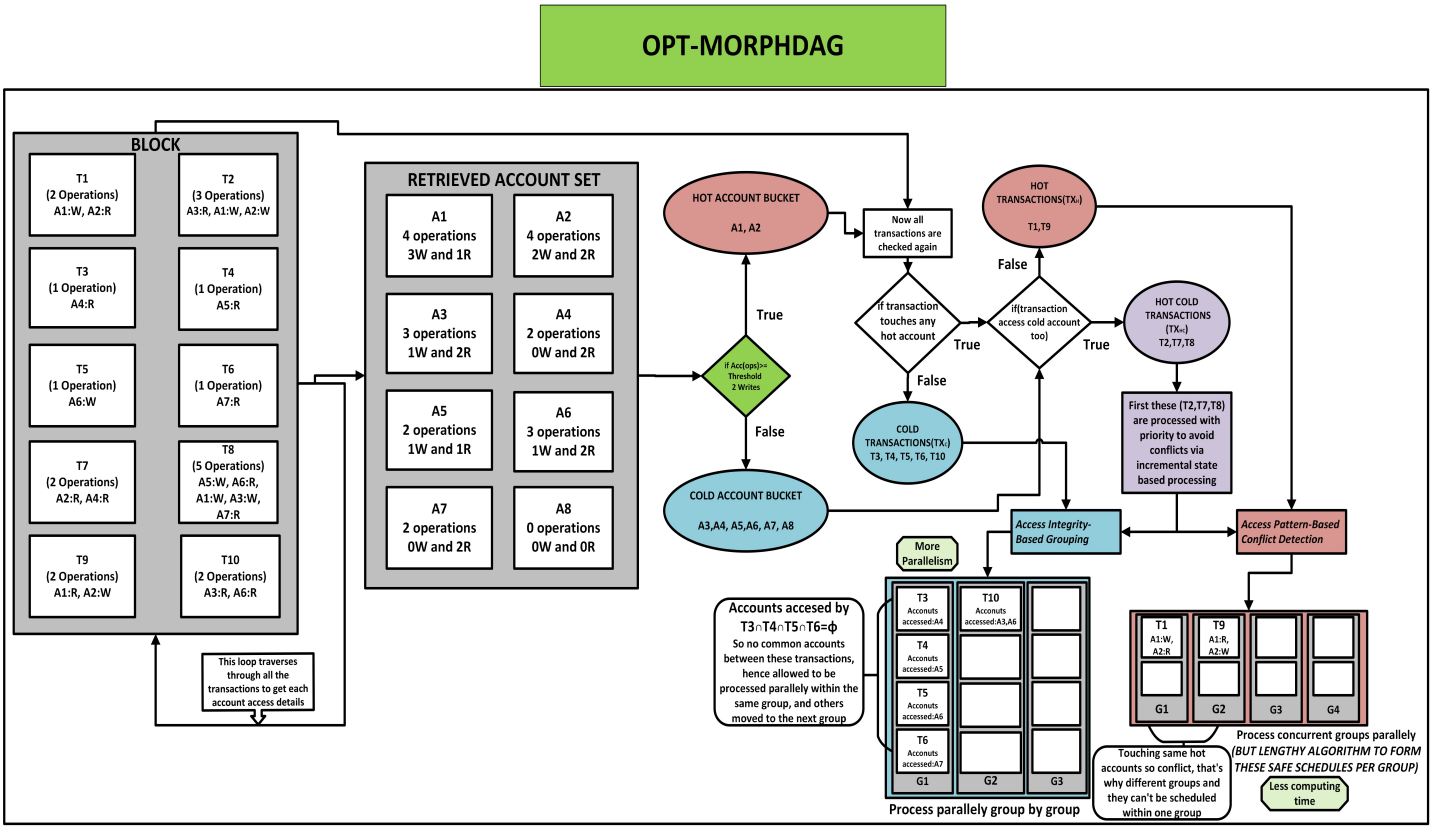

Fig. 2: The Updated MorphDAG pipeline, showing how transactions flow through the system.

Non incremental write (NIW) accesses. This minimizes conflicts by ensuring only access-compatible transactions are executed together.

- **Access integrity based grouping**: In this, cold transactions are grouped based on whether all their accesses are currently free from conflicts in their respective transaction sets. A transaction is added to a group only if none of its accessed accounts are already involved in transactions assigned to that group, ensuring safe parallel execution.

D. Advantages of OPT-MorphDAG

This approach provides multiple benefits over the original operation count-based threshold method:

- **Improved Precision:** Account Classification is now sensitive to type of operation rather than just operation count.
- **Higher Parallelism:** More non-conflicting transactions can safely enter the fast-path cold-execution mode.
- **Conflict Avoidance:** Transactions on high-risk accounts (write-heavy) are reliably routed to the hot path correctly, increasing parallelism.
- **Adaptability:** Our proposed optimization works better on most of the data skew levels as tested and shown in section V.

IV. Implementation: Refining Hot/Cold Account Classification

This section shows how we replace MorphDAG's simple per-account operation count with a per-account write-frequency analysis to more accurately flag hot and cold accounts, cutting unnecessary serialization and unlocking greater parallelism.

A key contribution of this work is the refinement of the mechanism used to classify "hot" and "cold" accounts within MorphDAG. The original implementation relied on a coarse-grained, operation-count metric based only on the total number of operations, which could lead to performance bottlenecks. Our change introduces a more precise, block-wide analysis of account access patterns. Here is an analysis of the old method and our proposed one:-

A. MorphDAG: Per-Account Operation Count

In the original MorphDAG pipeline, a transaction was classified as "hot" based on the hotness of the accounts it accessed. First, the system computed the **total number** of both read and write operations performed on each account across all transactions in the block. Accounts exceeding a predefined threshold were labeled as "hot." Transactions were then categorized accordingly: those accessing only hot accounts were marked hot, those accessing only cold accounts were marked cold, and mixed-access transactions were marked hot-cold.

Fig 1 illustrates the original MorphDAG pipeline where the classification of accounts (A1,A2,A3...A8) and transactions (T1,T2,T3...T10) is based on the total number of operations performed on each account across all transactions in a block. Initially, each transaction is scanned to extract read and write accesses per account. The system then matches the number of accesses per account (reads + writes) and determines a threshold (e.g., 4). Accounts whose operation count meets or exceeds this threshold are placed in the "hot account" bucket (e.g., A1, A2, A3, A6), while the rest are considered cold. Once accounts are labeled, transactions are revisited and categorized based on the types of accounts they access: transactions touching only hot accounts are marked as hot (e.g., T1, T2, T5, T9, T10), those accessing only cold accounts are cold (e.g., T3, T4, T6), and those touching both are hot-cold (e.g., T7, T8). Hot and hot-cold transactions are subject to dependency-aware scheduling using safe group formation via access-pattern conflict detection. Cold transactions are grouped based on disjoint account access and processed in parallel with minimal overhead [50].

```
accountAccessCount := make(map[string]int) //
    Account -> total R/W accesses
//Count access frequency per account
for _, tx := range block.Transactions {
    for _, rwsets := range tx.Payload.RWSets {
        for _, acct := range rwsets {
            accountAccessCount[acct]++
        }}}
//Compute hot threshold using top-k% logic
    accessCounts
accessCounts := []int{}
for _, count := range accountAccessCount {
    accessCounts = append(accessCounts, count)
}
sort.Sort(sort.Reverse(sort.IntSlice(accessCounts)
))// Descending order

k := 0.05 // e.g., top 5% hot accounts
thresholdIndex := int(float64(len(accessCounts))
    * k)
Hot_threshold := accessCounts[thresholdIndex]

// Mark hot accounts
hotAccounts := make(map[string]bool)
for acct, count := range accountAccessCount {
    if count >= Hot_threshold {
        hotAccounts[acct] = true
    }
}
//Classify transactions
for _, tx := range block.Transactions {
    hotTouched := false
    coldTouched := false
    for _, rwsets := range tx.Payload.RWSets {
        for _, acct := range rwsets {
            if hotAccounts[acct] {
                hotTouched = true
            } else {
                coldTouched = true
            }}}
    if hotTouched && coldTouched {
        // Mark as hot-cold transaction
    } else if hotTouched {
        // Mark as hot transaction
    } else {
        // Mark as cold transaction
    }}
```

Listing 1: MorphDAG Method: Total R/W accesses on account based Classification in Golang

Listing 1 outlines MorphDAG's dynamic account-based classification strategy for identifying hot and cold transactions. The process begins by scanning all transactions in a block to match the total number of read/write operations per account (lines 1–7). These access counts are then sorted in descending order to compute a dynamic threshold based on the top-$k\%$ most-accessed accounts (lines 9–17), which are labeled as hot (lines 20–25). Next, each transaction is classified by examining the accounts it accesses: if all accounts are hot, it is marked as hot; if all are cold, it is cold; and if it accesses both, it is labeled hot-cold (lines 27–43). This classification forces MorphDAG to isolate even less-contention operations with less chance of conflict and schedule cold transactions in parallel with a possible risk of conflict.

This is because, this method uses a flat operation count across all access types, it may misclassify some accounts as hot even if their contention potential is low which results in **reduced parallelism and increased computation** during scheduling, especially for transactions that don't require such strict ordering.

B. OPT-MorphDAG: Per-Account Frequency Analysis

To address these limitations of MorphDAG's original hot/-cold transaction classification, we propose a refined approach that analyzes the access patterns of all transactions in the current block. Instead of relying on a coarse per-account operation count, OPT-MorphDAG calculates the **write frequency** for each account and classifies them as hot or cold using a configurable percentile threshold.

Our implementation directly modifies the `AnalyzeHotAccounts()` function in the MorphDAG codebase. The rest of the DAG construction and dual-mode execution engine is left untouched.

Fig 2 illustrates the full pipeline of our proposed OPT-MorphDAG system. Given a block of transactions, we first extract all account-level read and write operations to compute per-account write frequencies. Based on a configurable threshold, accounts are classified into **hot** or **cold** buckets. Transactions are then revisited and labeled as **hot**, **cold**, or **hot-cold** based on the accounts they access. This classification enables differentiated scheduling: cold transactions are grouped using *access-integrity-based parallelism* [50] without conflict checks, while hot and hot-cold transactions are processed using safe *access-pattern-based conflict detection* [50]. Notably, hot-cold transactions are prioritized for early execution to minimize their blocking impact. This hybrid scheduling mechanism maximizes parallelism on low-contention paths while maintaining correctness on high-contention paths, improving overall throughput and reducing conflict overhead.

Algorithm 1 Hot/Cold Account Detection in OPT-MorphDAG (Per-Account Write Frequency)

1: **Input:** Transactions T, Ratio threshold r
2: **Output:** Sets of hot and cold accounts $HotSet$, $ColdSet$
3: Initialize empty maps: `writeFreq`, `readFreq`
4: Initialize empty sets: `HotSet`, `ColdSet`
5: **for** each transaction tx in T **do** ▷ Collect access frequencies
6: **for** each account acc accessed in tx **do**
7: **if** operation is a `read` **then**
8: `readFreq[acc]++`
9: **else if** operation is a `write` **then**
10: `writeFreq[acc]++`
11: **end if**
12: **end for**
13: **end for**
14: Extract all `writeFreq` values into a list F
15: Sort F in descending order
16: Calculate number of hot accounts: $k \leftarrow \lfloor |F| \times r \rfloor$
17: **if** $k = 0$ and $|F| > 0$ **then**
18: $k \leftarrow 1$ ▷ Ensure at least one hot account
19: **end if**
20: Set `hotThreshold` = $F[k-1]$ ▷ Determine frequency cut-off
21: **for** each account acc in `writeFreq` **do**
22: **if** `writeFreq[acc]` $\geq$ `hotThreshold` **then**
23: Add acc to `HotSet`
24: **else**
25: Add acc to `ColdSet`
26: **end if**
27: **end for**
28: **return** `HotSet`, `Coldset`

Algorithm 1 describes the pseudocode of the hot/cold account detection policy. In this, we first traverse all transactions and record the number of read and write accesses per account (lines 6–13). Once the frequencies are collected, we sort the accounts by their write frequency in descending order and determine a cut-off threshold that marks the top $k\%$ of most frequently written-to accounts as hot (lines 14–20). Accounts that meet or exceed this threshold are added to the hot set $HotSet$ (lines 21–23) and accounts which are less than this threshold are added to the $coldset$(lines 24–25).

This account-level classification is a fundamental shift from MorphDAG's operations-based one and allows the system to more intelligently schedule transactions. Transactions touching only cold accounts bypass expensive conflict detection and execute fully in parallel, while hot transactions are subject to dependency checks, ensuring correctness.

This block-level, account-centric strategy allows the system to **better exploit parallelism and improves throughput** without compromising consistency. This targeted approach directly contributes to the observed improvement in throughput.

V. Performance Evaluation

This section assesses how well the proposed OPT-MorphDAG performs, demonstrating improved scalability and reduced latency across growing transaction volumes and different R/W mixes, while verifying that its per-account frequency refinement maintains accurate state outcomes.

A. Experimental Setup

Testbed: To verify core functionality and performance in a non-distributed environment, all experiments were carried out on a local development laptop running Arch Linux (Kernel 6.15.8). The system in use is a Lenovo IdeaPad Gaming 3, which has 16 GB DDR5 RAM and an AMD Ryzen™ 5 6600H processor (6 cores, 12 threads, base clock 3.3 GHz, boost up to 4.5 GHz). With an integrated AMD Radeon 660M graphics card and an NVIDIA GeForce RTX 3050 (4 GB GDDR6), it has a dual-GPU configuration. For precise performance measurement, the Arch Linux environment offered low-level control and minimal overhead.

Workload Used: We make use of the Smallbank benchmark, which is a typical workload for blockchain systems. It is intended to assess concurrency and state conflicts by simulating basic financial operations across 10,000 accounts. We create synthetic workloads with different read-write ratios: 20% read and 80% write (write-heavy), 50% read and 50% write (balanced), and 80% read and 20% write (read-heavy) for a more realistic test enviorment . These enable us to test performance in various access patterns and contention scenarios. To assess our proposed approach, we contrast it with three configurations: the original, unaltered **MorphDAG** as the baseline; **OPT-MorphDAG**, our optimised version with dual-mode processing; and **MorphDAG-Serial**, which employs conventional serial execution. We examine the performance benefits of hot/cold classification and parallel execution with the aid of this comparison. For a fair assessment, every setup is tested on the same machine and we take an average of five runs for each test. Our optimisation has been coded in Golang itself, our source code for OPT-MorphDAG will be available at: https://github.com/s-uryansh/OPT-MorphDAG/edit/main/README.md

B. Scalability Analysis: Transaction Volume vs. Time

This test measures the time taken to process a block as the number of transactions within it increases. Fig 3 illustrates the results, comparing the processing time of the original MorphDAG against our proposed OPT-MorphDAG, which shows that our implementation processes the block more quickly, resulting in lower latency for any given number of transactions.

We observe that:

- At a load of **400 transactions**, the proposed OPT-MorphDAG processes the block in approximately **490ms** compared to MorphDAG's **573ms**.

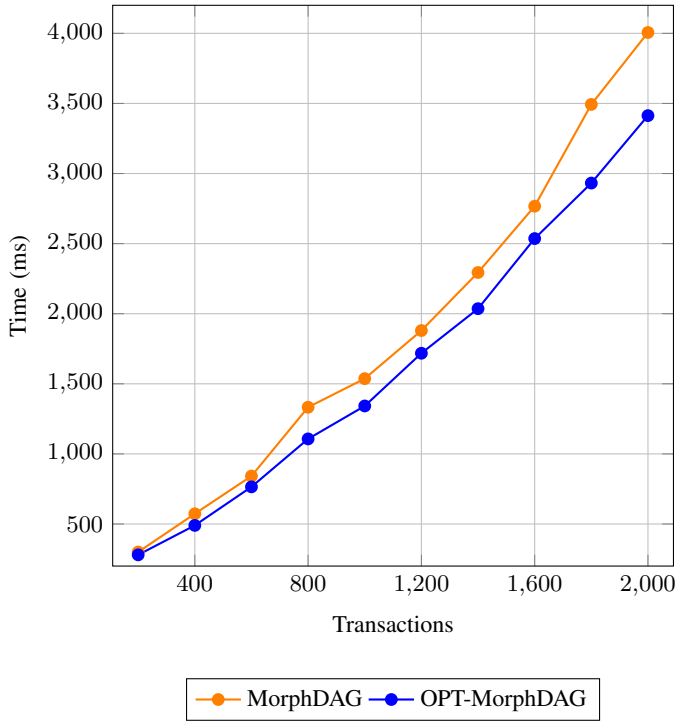

Fig. 3: Transaction executed overtime graph, it clearly shows that, more the number of transactions, more better our OPT-MorphDAG performs

- As the load increases to **1600 transactions**, the performance gap widens. The proposed OPT-MorphDAG takes **2,536ms**, while MorphDAG takes **2,767ms**.

This shows that OPT-MorphDAG exhibits improved scalability in addition to reduced latency. The performance gap between the two approaches indicates that OPT-MorphDAG gets even better with increasing network load, enabling much higher throughput.

C. Speedup over Serial

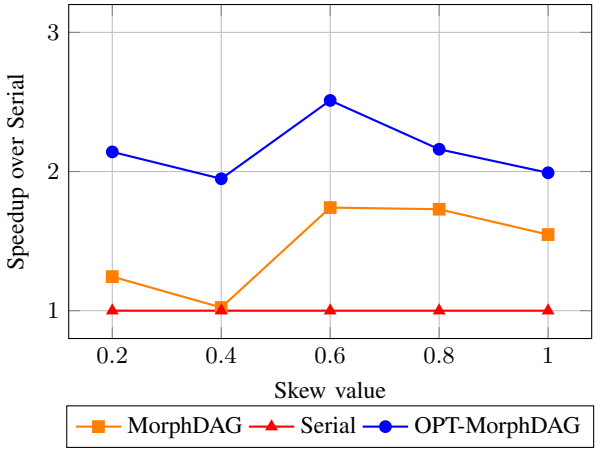

Fig. 4: Speedup of Original MorphDAG and OPT-MorphDAG over Serial under read-heavy workload with varying skew values.

In, fig 4, under a **read-heavy workload**, we plot speedup at skew levels 0.2–1.0 considering execution of MorphDAG-Serial as a baseline. Here skewness refers to how unevenly transactions access data in the blockchain.
Low skew(0.2) means data is accessed more uniformly — most accounts are used equally.
High skew(1.0) means a few accounts (hotspots) are accessed very frequently, causing conflicts.

At moderate skew (0.6), the original MorphDAG reaches its maximum speedup of approximately 1× to 1.75×. OPT-MorphDAG consistently performs better than it, reaching a speedup of up to 2.5× at skew 0.6 and never dropping below 1.9×. This indicates that parallelism is better exploited across all contention patterns because we identify these cold accounts more precisely in OPT-MorphDAG, particularly at mid-range skews, through our account-based hot/cold classification.

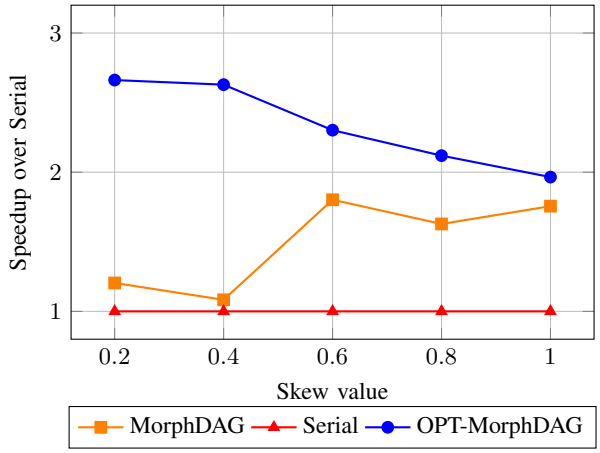

Fig. 5: Speedup of Original MorphDAG and OPT-MorphDAG over Serial under balanced (read/write) workload with varying skew values.

In, fig 5, we see that MorphDAG achieves 1.08–1.80× speedup over MorphDAG-serial under a **balanced read/write mix**, while OPT-MorphDAG hits (approx)1.96–2.66×. We observe that OPT-MorphDAG gains are greatest between skews 0.2 and 0.4, where parallel cold-only transactions are common. OPT-MorphDAG exhibits robustness to a variety of R/W patterns, **maintaining 2× improvement even at extreme skew (1.0)**.

We observe a convergence in this figure as we increase the skewness because, at higher skew values (e.g., 0.8 and 1.0), a small subset of hot accounts dominates most transactions, reducing the number of cold-only transactions that can be executed in parallel. As both Original and OPT-MorphDAG are constrained by the **same high-contention accounts**, their performance starts to align. Consequently, the advantage of OPT-MorphDAG diminishes as skew value increases, leading to approximately same results.

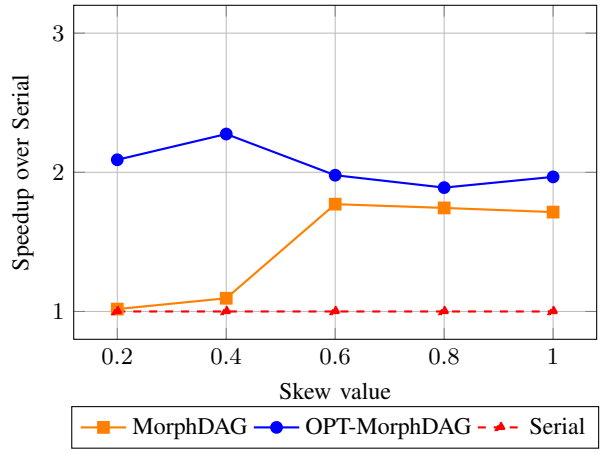

Fig. 6: Speedup of Original MorphDAG and OPT-MorphDAG over Serial under write-heavy workload (20R:80W) with varying skew values.

Fig. 6 illustrates the speedup achieved by MorphDAG and OPT-MorphDAG over the serial baseline when subjected to a write-heavy workload (20% reads, 80% writes) under varying levels of data access skew. Across all skew values, OPT-MorphDAG outperforms both MorphDAG and Serial execution due to its refined classification of hot accounts using per-account write frequency. This more precise identification reduces unnecessary serialization and allows conflict-free (cold) transactions to proceed in parallel.

We observe that both methods converge in performance at higher skew values. This is because as skew increases (e.g., skew = 0.8 or 1.0), most transactions target a small subset of high-conflict accounts. As a result, fewer transactions qualify as cold, reducing the scope for parallelism. In such cases, even the OPT-MorphDAG's fine-grained classification offers diminishing returns, leading to speedup lines that begin to align. Thus, the benefits of OPT-MorphDAG remain significant at low-to-moderate skew but gradually reduce under extreme workload contention .

D. Speedup performance under varying k and skewness

In Fig 7, we sweep the percentile of accounts classified as "hot" ($k = 1\%$ to 5%) for three skew levels (0.2, 0.6, 1.0).

- **Skew 0.2:** Original MorphDAG attains its maximum speedup of $\approx 1.97\times$ at $k = 4\%$ before dropping off, whereas OPT-MorphDAG starts at $\approx 2.50\times$ at $k = 1\%$ and remains above $1.90\times$ across all thresholds.
- **Skew 0.6:** OPT-MorphDAG consistently outperforms the original by $0.4 - 1.3\times$, peaking near $k = 5\%$, while Original MorphDAG lags throughout.
- **Skew 1.0:** At every top hot-account percentile, OPT-MorphDAG exceeds Original MorphDAG , demonstrating more robust gains.

Hence, OPT-MorphDAG's per-account frequency classification yields higher and more stable speedups over the original operation-count method. By accurately isolating true "hot" accounts, it avoids unnecessary serialization and maximizes

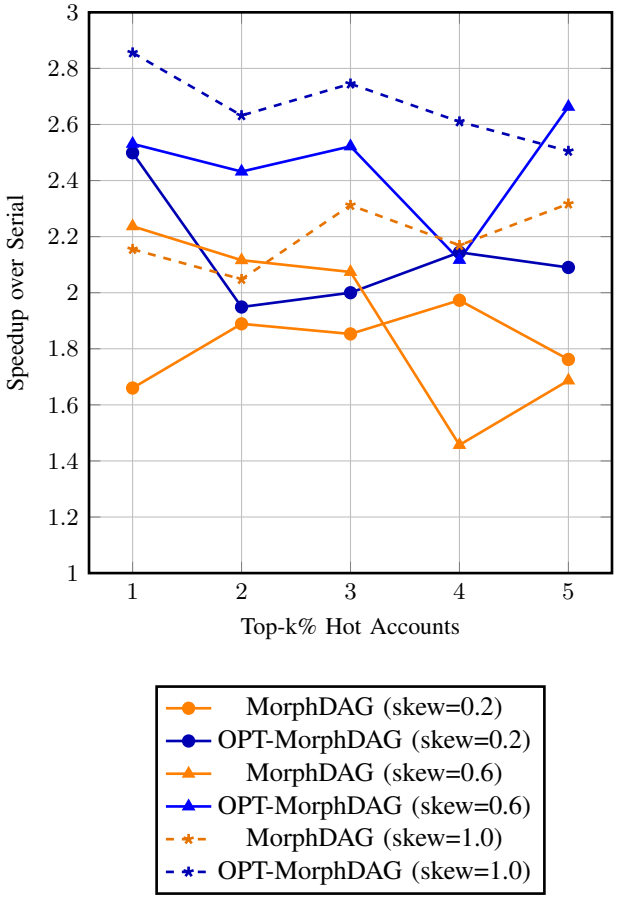

Fig. 7: Speedup of MorphDAG and OPT-MorphDAG over Serial under Read-Heavy Workload (80R:20W)

parallel execution regardless of the chosen percentile by **preventing read-only transactions** MorphDAG both reduces sensitivity to the exact k value and delivers uniformly superior performance compared to the original MorphDAG under a **read-heavy workload**.

In Fig8, we evaluate speedup across varying hot-account thresholds (k = 1% to %) for a balanced workload (50R:50W) under three skew settings (0.2, 0.6, 1.0):

- **Skew 0.2:** Original MorphDAG performs best at k = 1%with a 2.15× speedup, but drops steadily to 1.64× by k = 4%, recovering slightly at k = 5%. OPT-MorphDAG starts strong at 2.53× but **sharply declines** to just 1.20× by k = 5%, showing high sensitivity to mislabeling cold accounts as hot.
  This happens because OPT-MorphDAG relies on per-account write frequency to classify accounts as "hot." At low skew (0.2), access patterns are more uniformly distributed, so increasing k leads to many moderately accessed but still low contention accounts being misclassified as hot. As a result, more transactions are flagged as hot or hot-cold, triggering unnecessary conflict detection or serialization. This **reduces** the number of cold transactions that can run in parallel, thereby diminishing the speed.

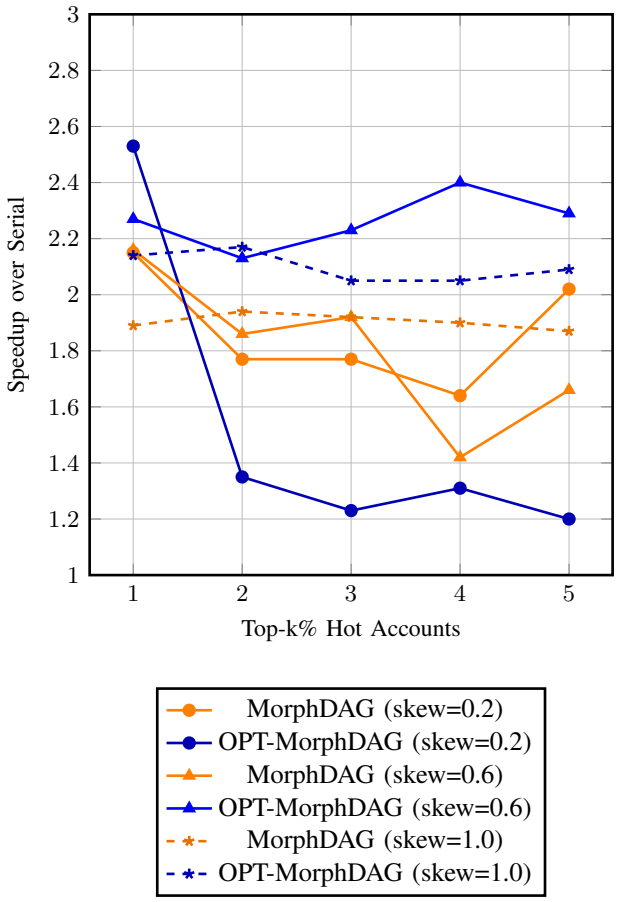

Fig. 8: Speedup of MorphDAG and OPT-MorphDAG over Serial under Balanced Workload (50R:50W)

- **Skew 0.6:** OPT-MorphDAG steadily improves from 2.27× to 2.40× as k increases to 4%, maintaining consistently better speedup than Original MorphDAG, which fluctuates between 1.42× and 2.16× with no clear trend.
- **Skew 1.0:** Both methods plateau with minimal variation. OPT-MorphDAG maintains stable performance around 2.1×, while Original MorphDAG hovers around 1.9×.

We observe that under balanced workloads, OPT-MorphDAG consistently outperforms the original by more accurately identifying contention-heavy accounts. At higher skew (0.6 and 1.0), where access is concentrated on few accounts, its write-frequency–based policy enables greater concurrency for cold transactions, leading to strong performance gains. At low skew (0.2), however, increasing k too much, misclassified cold accounts as hot, reducing parallelism. OPT-MorphDAG achieves higher and more stable speedups, especially under moderate to high skew, by better isolating conflicts and maximizing parallelism.

E. Latency and Throughput

In Fig 9, OPT-MorphDAG consistently achieves the lowest block latency across all skew levels, outperforming both Serial and original MorphDAG by 40–53%, even under high contention, due to better conflict isolation and parallel execution.

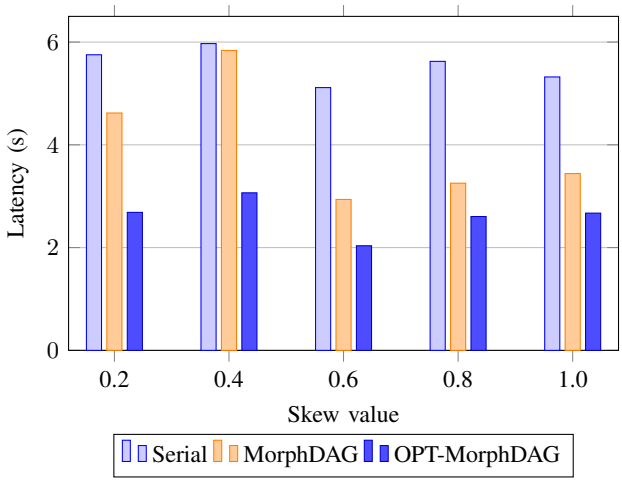

Fig. 9: Block-processing latency vs. skew (500 txs/block) shown as grouped bar chart.

OPT-MorphDAG delivers consistently lower and less variable latency by isolating truly contentious accounts. This allows conflict-free transactions to proceed in parallel, unlike Original MorphDAG, which suffers from higher latency due to misclassification and unnecessary serialization. In comparison to both Serial and original MorphDAG, OPT-MorphDAG lowers latency by **precisely identifying hot accounts**, avoiding needless serialisation, and permitting greater parallelism.

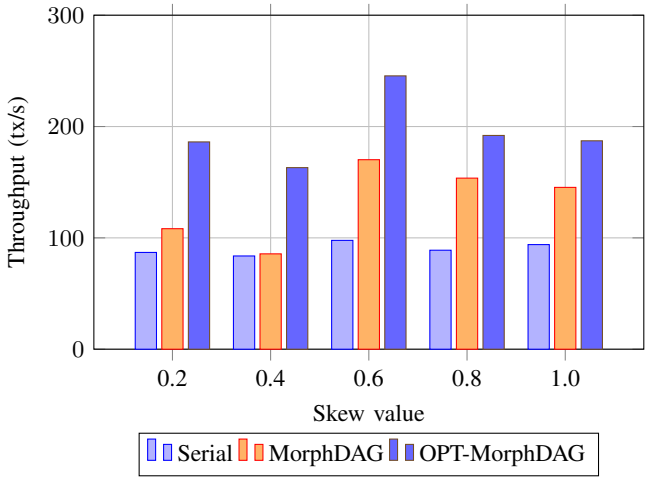

Fig. 10: Throughput vs. skew (500 txs/block) shown as grouped bar chart.

In Fig 10, OPT-MorphDAG consistently achieves the highest throughput across all skew levels, peaking at (approx) 246 tx/s—nearly 2.5× higher than Serial and Original MorphDAG—demonstrating superior scalability even under high contention.

By leveraging per-account access patterns, OPT-MorphDAG preserves parallelism and consistently achieves higher throughput across all contention levels. It avoids the bottlenecks of naive classification used in Original MorphDAG, especially under balanced or skewed loads. In contrast to

Serial and Original MorphDAG, which serialise needlessly, OPT-MorphDAG achieves higher throughput by isolating true conflicts, enabling more transactions to run in parallel and efficiently utilise available concurrency.

F. Verification of State Integrity

A critical aspect of our evaluation was to confirm that the performance enhancements did not compromise the transactional integrity or correctness of the ledger. The goal is to achieve faster processing without altering the final state of the blockchain.

To validate this, we inspected the output of the `stateTransfer()` function, which is responsible for applying the final state changes to the database as part of the Smallbank benchmark execution. We logged the resulting account balances and operational values after a series of transactions for both the original MorphDAG and our proposed OPT-MorphDAG.

Our tests confirmed that the final-state values are identical in both implementations. For example, for a given set of operations, both versions resulted in the exact same final account balances (e.g., `Balance For Read Operations: 100000`) and transaction values (e.g., `Value used in RW operations: 5`). This was achieved even while OPT-MorphDAG recorded lower processing times.

This result provides conclusive evidence that our refinement correctly preserves data consistency and produces deterministic outcomes. The increased parallelism achieved through per-account frequency analysis does not introduce race conditions or data loss. Maintaining integrity of the ledger.

In a nutshell, the per-account frequency analysis is fundamentally more accurate than a simple operation count. It correctly identifies truly ”cold” transactions even if they have many read operations, allowing for greater parallelism. This results in not only a faster system overall but one that is significantly more robust and predictable under changing application behavior.

In a nutshell, we observe that due to its dual-mode processing, ability to handle dynamic workload and per-account hot/cold classification, OPT-MorphDAG consistently improves throughput and lowers latency compared to both Original MorphDAG and a purely serial executor, according to our evaluation using the Smallbank benchmark under various read-/write mixes. It reduces block-processing times by 10–15%, lowers latency by 40–53%, and maintains nearly 2.5× higher throughput across all skew levels, while achieving up to 2.5× speedup over serial execution and 1.4×-1.5× speedup over MorphDAG at moderate contention skew. Stable parallelism is shown by sensitivity analyses across a range of hot-account thresholds, and stateTransfer verifications verify that these enhancements maintain precise final account balances and overall data integrity.

VI. Conclusion and Future work

We conclude that OPT-MorphDAG achieves much higher parallelism and throughput by classifying accounts according to read/write frequencies, maintaining the simplicity of MorphDAG's original design while achieving up to 2.5× over serial baseline and 1.4 to 1.5× over MorphDAG and 40-53% latency reductions without sacrificing state integrity. This opens the door for scalable multi-node deployments, adaptive thresholds, and a variety of real-world benchmarks.

Although this work shows notable performance gains, there are still a number of directions in which future research can go to improve the MorphDAG framework. The following crucial directions are as follows:

- **Dynamic Threshold Adjustment:** To go beyond a static percentile, create an adaptive mechanism that can dynamically modify the ratio threshold for hot account classification based on observed conflict rates and real-time network load.
- **Broader Workload Evaluation:** To evaluate the improved model's performance under more complex and varied transaction patterns, compare it to a broader range of real-world application workloads, such as those from Decentralised Finance (DeFi) or Non-Fungible token(NFT) marketplaces, and try other workloads like Bitcoin workload as well for better analysis.
- **Large-Scale Distributed Testing:** To examine the system's resilience, scalability, and performance under a more realistic heavy workload, run experiments on a distributed, multi-node network.

References

[1] Ihsan H. Abdulqadder and Shijie Zhou. Sliceblock: Context-aware authentication handover and secure network slicing using dag-blockchain in edge-assisted sdn/nfv-6g environment. *IEEE Internet of Things Journal*, 9(18):18079–18097, 2022.

[2] Subhi Alrubei, Jonathan Rigelsford, Callum Willis, and Edward Ball. Ethereum blockchain for securing the internet of things: Practical implementation and performance evaluation. In *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–5, 2019.

[3] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 1–15. ACM, April 2018.

[4] Parwat Singh Anjana, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. An efficient framework for optimistic concurrent execution of smart contracts. *Proceedings of the 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 83–92, 2019.

[5] Federico Matteo Benčić and Ivana Podnar Žarko. Distributed ledger technology: Blockchain compared to directed acyclic graph. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1569–1570, 2018.

[6] Ved Prakash Chaudhary, Chirag Juyal, Sandeep Kulkarni, Sweta Kumari, and Sathya Peri. Achieving starvation-freedom in multi-version transactional memory systems, 2019.

[7] Anubhav Chauhan. Bitcoin: A peer-to-peer electronic cash system. *Journal of Pre-College Engineering Education Research (J-PEER)*, 09 2024.

[8] Yourong Chen, Yang Zhang, Yubo Zhuang, Kelei Miao, Seyedamin Pouriyeh, and Meng Han. Efficient and secure blockchain consensus algorithm for heterogeneous industrial internet of things nodes based on double-dag. *IEEE Transactions on Industrial Informatics*, 20(4):6300–6312, 2024.

[9] Xiaohai Dai, Guanxiong Wang, Jiang Xiao, Zhengxuan Guo, Rui Hao, Xia Xie, and Hai Jin. LightDAG: A low-latency DAG-based BFT consensus through lightweight broadcast. Cryptology ePrint Archive, Paper 2024/160, 2024.

[10] Xiaohai Dai, Yifan Zhou, Jiang Xiao, Feng Cheng, Xia Xie, Hai Jin, and Bo Li. Geckodag: Towards a lightweight dag-based blockchain via reducing data redundancy. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 451–462, 2023.

[11] Zhongxu Dong, Huanyu Wu, Zongyao Li, De Mi, Olaoluwa Popoola, and Lei Zhang. Trustworthy vanet: Hierarchical dag-based blockchain solution with proof of reputation consensus algorithm. In *2023 IEEE International Conference on Blockchain (Blockchain)*, pages 127–132, 2023.

[12] Xiang Fu, Wang Huaimin, Shi Peichang, Ouyang Xue, and Xunhui Zhang. Jointgraph: A dag-based efficient consensus algorithm for consortium blockchains. *Software: Practice and Experience*, 51:1987–1999, 09 2019.

[13] Ning Gao, Xingyuan Wang, and Xiukun Wang. Multi-layer progressive face alignment by integrating global match and local refinement. *Applied Sciences*, 9(5), 2019.

[14] Fengyang Guo, Xun Xiao, Artur Hecker, and Schahram Dustdar. Characterizing iota tangle with empirical data. In *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pages 1–6, 2020.

[15] Hechuan Guo, Minghui Xu, Jiahao Zhang, Chunchi Liu, Dongxiao Yu, Schahram Dustdar, and Xiuzhen Cheng. Filedag: A multi-version decentralized storage network built on dag-based blockchain. *IEEE Transactions on Computers*, 72(11):3191–3202, 2023.

[16] Houssein Hellani, Layth Sliman, Abed Ellatif Samhat, and Ernesto Exposito. Tangle the blockchain:towards connecting blockchain and dag. In *2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 63–68, 2021.

[17] Ohnmar Hlaing, Hla Hla Htwe, and Wai Wai Myint. Read-write-validate approach for optimistic concurrency control about michael kors (mk) sale transaction. In *2024 IEEE Conference on Computer Applications (ICCA)*, pages 1–5, 2024.

[18] Xiaoge Huang, Wenjing Li, Caiwei Yu, Chengchao Liang, and Qianbin Chen. Layer-wise personalized federated learning for dag blockchain-enabled internet of vehicles. In *2024 16th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1145–1150, 2024.

[19] Misbah Khan, Shabnam Kasra Kermanshahi, and Jiankun Hu. Ghostforge: A scalable consensus mechanism for dag-based blockchains. *IEEE Open Journal of the Computer Society*, 5:736–747, 2024.

[20] I.D. Kotilevets, I.A. Ivanova, I.O. Romanov, S.G. Magomedov, V.V. Nikonov, and S.A. Pavelev. Implementation of directed acyclic graph in blockchain network to improve security and speed of transactions. *IFAC-PapersOnLine*, 51(30):693–696, 2018. 18th IFAC Conference on Technology, Culture and International Stability TECIS 2018.

[21] Lang Li, Dongyan Huang, and Chengyao Zhang. An efficient dag blockchain architecture for iot. *IEEE Internet of Things Journal*, 10(2):1286–1296, 2023.

[22] Guoqiong Liao, Hao Ding, Chuanling Zhong, and Yinxiang Lei. Rt-dag: Dag-based blockchain supporting real-time transactions. *IEEE Internet of Things Journal*, 11(20):32759–32772, 2024.

[23] Haojun Liu, Xinbo Luo, Hongrui Liu, and Xubo Xia. Merkle tree: A fundamental component of blockchains. In *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*, pages 556–561, 2021.

[24] Sunil Dhondu Mone. Timestamp-ordering protocol for concurrent transactions - a performance study. In *National Conference on Emerging Trends in Computer Technology (NCETCT-2014), International Journal of Computer Applications*, volume NCETCT, pages 24–26, 2014.

[25] SeJoon Park and Jihie Kim. Dag-gcn: Directed acyclic causal graph discovery from real world data using graph convolutional networks. In *2023 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 318–319, 2023.

[26] Seongjoon Park, Seounghwan Oh, and Hwangnam Kim. Performance analysis of dag-based cryptocurrency. *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2019.

[27] Huma Pervez, Muhammad Muneeb, Muhammad Usama Irfan, and Irfan Ul Haq. A comparative analysis of dag-based blockchain architectures. In *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)*, pages 27–34, 2018.

[28] Naina Qi, Yong Yuan, and Fei-Yue Wang. Dag-block: A novel architecture for scaling blockchain-enabled cryptocurrencies. *IEEE Transactions on Computational Social Systems*, 11(1):378–388, 2024.

[29] Xidi Qu, Shengling Wang, Kun Li, Jianhui Huang, and Xiuzhen Cheng. Tidyblock: A novel consensus mechanism for dag-based blockchain in iot. *IEEE Transactions on Mobile Computing*, 24(2):722–735, 2025.

[30] Praveen Reddy, Viet Pham, Prabadevi B, N. Deepa, Kapal Dev, Thippa Gadekallu, Rukhsana Ruby, and Madhusanka Liyanage. Industry 5.0: A survey on enabling technologies and potential applications. *Journal of Industrial Information Integration*, 26, 07 2021.

[31] Bumho Son, Jaewook Lee, and Huisu Jang. A scalable iot protocol via an efficient dag-based distributed ledger consensus. *Sustainability*, 12(4), 2020.

[32] Anping Song, Yi Dai, Ruyi Ji, and Ziheng Song. Extend pbft protocol with l-dag. In *2021 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 963–970, 2021.

[33] Xingshuo Song, Shiyong Li, Yanxia Chang, Chi Zhang, and Quanlin Li. Performance analysis for dag-based blockchain systems based on the markov process. *Journal of Systems Science and Systems Engineering*, 34(1):29–54, 2025.

[34] Akylbek Tokhmetov, Vyacheslav Lee, and Liliya Tanchenko. Development of dag blockchain model. *Scientific Journal of Astana IT University*, 01 2024.

[35] Qin Wang, Jiangshan Yu, Shiping Chen, and Yang Xiang. Sok: Diving into dag-based blockchain systems. 12 2020.

[36] Tianyu Wang, Qian Wang, Zhaoyan Shen, Zhiping Jia, and Zili Shao. Understanding intrinsic characteristics and system implications of dag-based blockchain. In *2020 IEEE International Conference on Embedded Software and Systems (ICESS)*, pages 1–6, 2020.

[37] Tianyu Wang, Qian Wang, Zhaoyan Shen, Zhiping Jia, and Zili Shao. Understanding characteristics and system implications of dag-based blockchain in iot environments. *IEEE Internet of Things Journal*, 9(16):14478–14489, 2022.

[38] Hiroki Watanabe, Tatsuro Ishida, Shigenori Ohashi, Shigeru Fujimura, Atsushi Nakadaira, Kota Hidaka, and Jay Kishigami. Enhancing blockchain traceability with dag-based tokens. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 220–227, 2019.

[39] Jin Xie, Ke Zhang 0008, Yunlong Lu, and Yan Zhang 0002. Resource-efficient dag blockchain with sharding for 6g networks. *IEEE Network*, 36(1):189–196, 2022.

[40] Jie Xu, Qingyuan Xie, Sen Peng, Cong Wang, and Xiaohua Jia. Adaptchain: Adaptive scaling blockchain with transaction deduplication. *IEEE Transactions on Parallel and Distributed Systems*, 34(6):1909–1922, 2023.

[41] Jie Xu, Qingyuan Xie, Sen Peng, Cong Wang, and Xiaohua Jia. Adaptchain: Adaptive scaling blockchain with transaction deduplication. *IEEE Transactions on Parallel and Distributed Systems*, 34(6):1909–1922, 2023.

[42] Jie Xu, Qingyuan Xie, Sen Peng, Cong Wang, and Xiaohua Jia. Adaptchain: Adaptive scaling blockchain with transaction deduplication. *IEEE Transactions on Parallel and Distributed Systems*, 34(6):1909–1922, 2023.

[43] Shu Yang, Ziteng Chen, Laizhong Cui, Mingwei Xu, Zhongxing Ming, and Ke Xu. Codag: An efficient and compacted dag-based blockchain protocol. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 314–318, 2019.

[44] Weiwei Yang, Long Shi, Hui Liang, and Wei Zhang. Trusted mobile edge computing: Dag blockchain-aided trust management and resource allocation. *IEEE Trans. Wirel. Commun.*, 23(5):5006–5018, May 2024.

[45] Wenhui Yang, Xiaohai Dai, Jiang Xiao, and Hai Jin. Ldv: A lightweight dag-based blockchain for vehicular social networks. *IEEE Transactions on Vehicular Technology*, 69(6):5749–5759, 2020.

[46] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. Ohie: Blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 90–105, 2020.

[47] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. Ohie: Blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 90–105, 2020.

[48] Chuan Zhang, Mingyang Zhao, Jinwen Liang, Qing Fan, Liehuang Zhu, and Song Guo. Nano: Cryptographic enforcement of readability and editability governance in blockchain databases. *IEEE Transactions on Dependable and Secure Computing*, 21(4):3439–3452, 2024.

[49] Hanwen Zhang, Supeng Leng, Fan Wu, and Haoye Chai. A dag blockchain-enhanced user-autonomy spectrum sharing framework for 6g-enabled iot. *IEEE Internet of Things Journal*, 9(11):8012–8023, 2022.

[50] Shijie Zhang, Jiang Xiao, Enping Wu, Feng Cheng, Bo Li, Wei Wang, and Hai Jin. Morphdag: A workload-aware elastic dag-based blockchain. *IEEE Transactions on Knowledge and Data Engineering*, 36(10):5249–5264, 2024.

[51] Xiaodong Zhang, Ru Li, and Hui Zhao. A parallel consensus mechanism using pbft based on dag-lattice structure in the internet of vehicles. *IEEE Internet of Things Journal*, 10(6):5418–5433, 2023.