Accelerating Verifiable Queries over Blockchain Database System Using Processing-in-memory

YIFAN HUA, Peking University, Beijing, China
SHENGAN ZHENG, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
WEIHAN KONG, Shanghai Jiao Tong University, Shanghai, China
DONGLIANG XUE, computer, Shanghai Jiao Tong University, Shanghai, China
KE XI, Xinjiang Electronics Research Institute Co., Ltd., Urumqi, China
YUHENG WEN, Shanghai Jiao Tong University, Shanghai, China
LINPENG HUANG, Shanghai Jiao Tong University, Shanghai, China
HONG MEI, Peking University, Beijing, China

Blockchain database systems, such as Ethereum and vChain, suffer from limited memory bandwidth and high memory access latency when retrieving user-requested data. Emerging processing-in-memory (PIM) technologies are promising to accelerate users' queries, by enabling low-latency memory access and aggregated memory bandwidth scaling with the number of PIM modules. In this paper, we present Panther, the first PIM-based blockchain database system supporting efficient verifiable queries. Blocks are distributed to PIM modules for high parallelism with low inter-PIM communication cost, managed by a regression-based model. For load balance across PIM modules, data are adaptively promoted and demoted between the host and PIM sides. In multiple datasets, Panther achieves up to 23.6× speedup for verifiable queries and reduces metadata storage by orders of magnitude compared to state-of-the-art designs on real PIM hardware.

CCS Concepts: • **Hardware** → *Emerging architectures*; • **Computer systems organization** → *Multicore architectures*; *Other architectures*; • **Software and its engineering** → *Distributed memory*.

Additional Key Words and Phrases: Processing in memory, Blockchain, Merkle hash tree, Pointer chasing, Verifiable query

1 Introduction

With the widespread adoption of blockchain for data-intensive applications such as finance, supply chain, and health care [3, 25, 30, 32, 37, 41–43, 47], there is a growing demand from users to query data stored in blockchain database systems. Two types of queries [29, 44] are commonly performed. The first is a single-query to get a single data object from a block. The second is a multi-query to retrieve multiple data objects from blocks within a time range. The blockchain system returns user-requested data along with their proofs by traversing Merkle Hash Tree (MHT) indexes [19, 31, 33]. The proofs are used to verify whether these data have been tampered with. Pointer

New Paper, Not an Extension of a Conference Paper.

Authors' Contact Information: Yifan Hua, Peking University, Beijing, China, huayifan@pku.edu.cn; Shengan Zheng, Shanghai Jiao Tong University, Shanghai, China, shengan@sjtu.edu.cn; Weihan Kong, Shanghai Jiao Tong University, Shanghai, China, weihankong@sjtu.edu.cn; Dongliang Xue, Shanghai Jiao Tong University, Shanghai, China, xuedongliang010@sjtu.edu.cn; Ke Xi, Xinjiang Electronics Research Institute Co., Ltd., Urumqi, China, 601784808@qq.com; Yuheng Wen, Shanghai Jiao Tong University, Shanghai, China, wenyuheng@sjtu.edu.cn; Linpeng Huang, Shanghai Jiao Tong University, Shanghai, China, lphuang@sjtu.edu.cn; Hong Mei, Peking University, Beijing, China, meih@pku.edu.cn. Hong Mei, Linpeng Huang and Shengan Zheng are corresponding authors.

ACM 1544-3973/2025/9-ART
https://doi.org/10.1145/3768318

chasing [11, 20, 38, 45, 49] incurred by traversing MHT indexes exhibits irregular and unpredictable memory access patterns, leading to a poor cache hit rate and excessive memory accesses. The increasing performance gap between processor speeds and memory access speeds (i.e., the memory wall [1, 2, 34, 35]) renders pointer chasing the dominant performance bottleneck when traversing MHT indexes. Therefore, current blockchain database systems, such as Ethereum [42] and vChain [44], suffer from limited memory bandwidth and high memory access latency when retrieving user-requested data and proofs.

By embedding processors in memory, processing-in-memory (PIM) technologies [8, 10, 16, 18, 24, 26, 36] are promising to mitigate these performance bottlenecks, by enabling low-latency memory access and aggregated memory bandwidth scaling with the number of PIM modules. The PIM system consists of two parts: the host side and the PIM side. The host side involves powerful CPU cores and the PIM side includes a set of PIM modules with wimpy cores. The host side dispatches batches of data processing tasks to PIM modules and collects results from the PIM side. The PIM modules process these tasks within the memory modules that integrate computational resources, reducing data movement to the host CPU for processing. Therefore, PIM has potential to accelerate verifiable queries over blockchain database systems by exploiting the high parallelism between PIM modules and accelerating pointer chasing.

Prior works [5, 6, 12, 22, 27] use PIM to accelerate queries over traditional key-value database systems. They partition the key space into multiple disjoint key ranges, and distribute data and their ordered indexes (e.g., skiplist and B^+-tree) within these key ranges to PIM modules. Hash tables are employed on the host side to manage distributed data stored in PIM modules. For processing users' requests, previous works first locate the PIM module containing the requested data through hash tables on the host side, and then traverse the ordered indexes in PIM modules to retrieve the requested data. For load balance across PIM modules under skewed workloads, hot data are promoted from the PIM side to the host side during runtime.

Unfortunately, directly applying techniques from existing PIM-based key-value database systems to blockchain database systems faces three challenges. (1) The hash-based management for distributed data in PIM modules overlooks the temporal characteristics of blockchain data distribution and results in significant metadata cost. Prior works employ hash tables for managing distributed data, assuming that the data distribution does not follow a consistent pattern during runtime. The hash tables consume significant storage space. However, in blockchain, data blocks exhibit a temporal distribution pattern, as the time interval between block generation is determined by the underlying consensus protocol. For example, the Ethereum system [42] generates a block about every twelve seconds. This gives us an opportunity to accelerate verifiable queries and lower metadata cost by building a regression function between timestamps and block heights (i.e., block IDs) for managing distributed data blocks in PIM modules. (2) The key-range-based partitioning method for traditional indexes (e.g., skiplist and B^+-tree) is unsuitable for MHT indexes in blockchain database systems, since traditional indexes are not used for data verification while MHT indexes are. Prior works distribute data within a key range to the same PIM module and build an ordered index for data within key ranges in each PIM module. However, in blockchain, data are grouped into blocks, and data within a key range exist in multiple blocks. Each block constructs an MHT index for data indexing and verification. Hash values in an MHT are calculated by data in the block and used for data verification. If MHT indexes are built by data within key ranges across blocks rather than data within each block, the blockchain fails to verify data since the hash values in MHTs are modified. As a result, the index partitioning scheme should be redesigned in blockchain database systems with consideration for data verification. (3) Prior works only promote hot data to the host side without demoting cold data back to PIM modules during runtime, resulting in a large amount of data and significant load on the host side. This leaves PIM modules underutilized and lowers parallelism between the host and PIM sides. Thus, the method for achieving load balance across PIM modules should adapt to data hotness changes.

This paper presents Panther, the first PIM-based blockchain database system that supports efficient verifiable single-queries and multi-queries. Panther incorporates a regression-based model rather than hash tables to manage

distributed data blocks in PIM modules. The model extracts the mapping relationship between timestamps and block heights through a piecewise linear function to locate data blocks in PIM modules, accelerating verifiable queries and lowering metadata cost. In addition, Panther employs a subtree-based MHT index partitioning mechanism to distribute data and indexes in blockchain systems across PIM modules. Data and index nodes used for data indexing and verification in each MHT subtree are stored in the same PIM module to lower inter-PIM communication cost. MHT subtrees are evenly distributed to PIM modules for high parallelism and load balance across PIM modules, as each PIM module stores a similar amount of data. In this way, the MHT index partitioning mechanism achieves high parallelism across PIM modules and incurs low inter-PIM communication cost. To overcome the limitation of existing data promotion strategy for load balance across PIM modules, Panther adaptively promotes hot data to the host side and demotes cold data to the PIM side with data hotness changes during runtime. Concisely, this paper makes the following contributions:

• We introduce Panther, a PIM-based blockchain database system supporting efficient verifiable single-queries and multi-queries. To the best of our knowledge, Panther is the first design to accelerate queries and data verification over blockchain database systems using PIM.

• We propose a regression-based model for managing distributed data in PIM modules, accelerating verifiable queries and reducing metadata cost by leveraging the temporal characteristics of blockchain.

• We design a subtree-based MHT index partitioning mechanism to distribute data and indexes across PIM modules, enabling high parallelism with low inter-PIM communication cost.

• We present an adaptive data promotion and demotion mechanism to reduce load imbalance across PIM modules with data hotness changes during runtime.

• We evaluate Panther on real PIM hardware. In multiple datasets, Panther achieves up to 23.6× speedup for single-query, 19.6× speedup for multi-query, 16.1× speedup for blockchain construction, and reduces metadata storage by 1 to 2 orders of magnitude compared to state-of-the-art blockchain database systems.

2 Background and motivation

This section provides background on blockchain database system, pointer chasing, PIM system architecture, and prior PIM-based key-value database systems. We analyze the limitations of previous works and summarize our motivations to design a PIM-based blockchain database system.

2.1 Blockchain database system

Blockchain [3, 25, 30, 32, 37, 41–43, 46] has become a promising distributed ledger technology for multiple parties to engage and share a decentralized tamper-proof database, recording the transactions between these parties in timestamped and chronologically linked data blocks. Figure 1 shows the overview of a Merkle-based blockchain database system consisting of three actors: miner node, full node, and light node. The miner node is responsible for packaging data into blocks, appending new blocks to the blockchain, and synchronizing new blocks to other nodes in the blockchain system. The light node has limited storage and only stores block headers containing metadata such as timestamps and root hash values of blocks. It requests data stored in the full node (①). Two types of queries are commonly performed [29, 44]. The first is a single-query to get a single data object from a block. The second is a multi-query to retrieve multiple data objects from blocks within a time range. Each data object in the blockchain is modeled as a tuple in the form of $O_i = \langle T_i, V_i, W_i \rangle$ [4, 44], where T_i is the timestamp of the block containing the object, V_i is the attribute set of the object, and W_i is the keyword set of the object. The full node stores both block headers and bodies and is responsible for processing query requests from the light node (②). The full node first locates the requested block height (i.e., block ID) using the block's timestamp, and then retrieves the requested data object within the block using the data object's keyword. Current blockchain systems employ traditional indexes (e.g., skiplist in vChain [44]) for inter-block indexing and Merkle Hash Tree

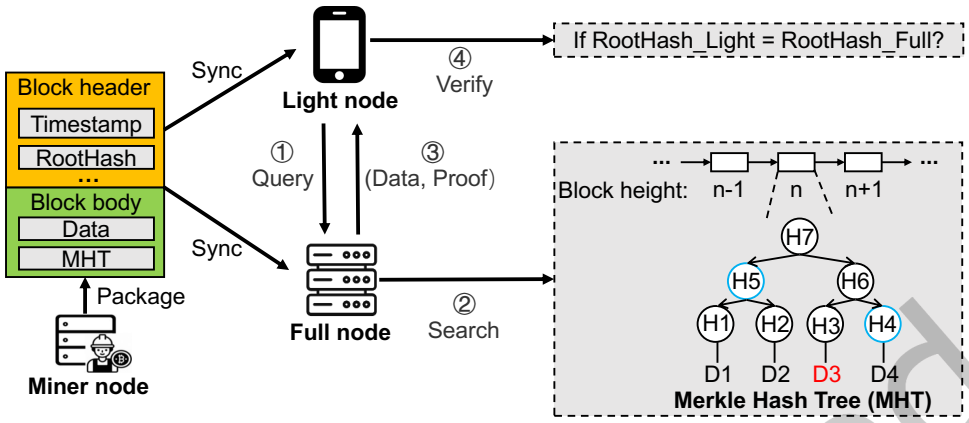


Fig. 1. Blockchain database system.

(MHT) (e.g., Merkle patricia trie in Ethereum [42]) for intra-block data indexing and verification. In an MHT, apart from the index function of tree, the leaf nodes contain the hash values of the data (e.g., $H3 = hash(D3)$). The internal nodes include the hash values of their child nodes (e.g., $H5 = hash(H1||H2)$). If the data is tampered with, the tampering will be forwarded up to the upper-level hash values and finally to the root hash (i.e., $H7$). MHT enables the proof of existence for requested data. For example, to prove $D3$, the sibling hashes along the search path (i.e., $H4$ and $H5$) are returned as the proof. The full node returns the requested data and proof to the light node (③). The light node can verify the requested data $D3$ by reconstructing the root hash using the proof (i.e., $RootHash\_Full = hash(H5||hash(hash(D3)||H4))$) and comparing it with the $RootHash\_Light$ stored in the block header (④). If the two values are equal, the returned data is trusted.

Data blocks in blockchain systems exhibit a temporal distribution pattern, as the time interval between block generation is determined by the underlying consensus protocol. Blockchain systems adopt an append-only ledger structure in which transactions are grouped into sequentially generated blocks. These blocks are strictly ordered in time and form a linear chain structure with well-defined temporal semantics. Block production in blockchain systems is governed by consensus protocols that regulate the inter-block interval in a predictable manner—such as every 10 minutes in Bitcoin (Proof of Work), every 12 seconds in Ethereum (Proof of Stake), every 1 second in BNB Smart Chain (Proof of Staked Authority), etc.

2.2 Pointer chasing and PIM system

Pointer chasing [11, 20, 38, 45, 48] is a fundamental operation in many data structures [13, 15, 23, 27, 28, 40] such as tree, linked list, skiplist, and graph. Since nodes linked by pointers may be stored in memory far away from each other, pointer chasing has a poor cache hit rate and requires a significant number of memory accesses to look for the next hop. The increasing performance gap between processor speeds and memory access speeds (i.e., the memory wall [1, 2, 34, 35]) renders pointer chasing the dominant performance bottleneck for Merkle-based blockchain database systems when traversing MHT indexes.

The emergence of Process-in-memory (PIM) technologies [8, 10, 16, 18, 24, 26, 36] introduce a new paradigm that can enhance the efficiency of pointer chasing by allowing it to be executed within the memory devices. As shown in Figure 2, the PIM system consists of two parts: the host side and the PIM side. The host side involves powerful CPU cores, caches, and memory. The PIM side includes multiple PIM modules. Each PIM module is composed of a data processor unit (DPU) with weaker computing capability than CPU, and capacity-limited

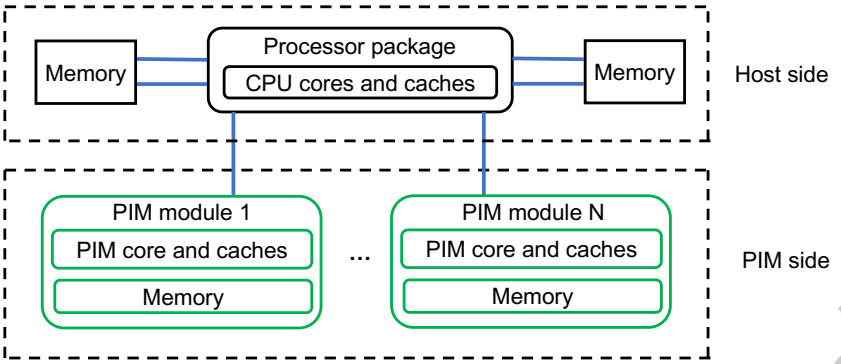


Fig. 2. PIM system architecture.

caches and memory. PIM cores are closer to memory and access memory faster than CPU cores. The host side dispatches data processing tasks to PIM modules and gathers the results after the PIM modules finish the computation. The PIM modules process these tasks within the memory modules that integrate computational resources, reducing data movement to the host CPU for processing. The two components of the PIM system, the host side and the PIM side, prefer different kinds of workloads [22, 23, 39]. The distributed PIM side prefers uniformly random workloads and suffers from the load imbalance under skewed workloads. In contrast, the host side prefers skewed workloads since the spatial and temporal locality characteristics in these workloads lead to better CPU cache efficiency.

Currently, UPMEM [8] (2.1 GHz CPUs on the host side and 350 MHz DPUs on the PIM side) is the only commercially-available PIM product. Compared with a conventional machine equipped with DDR4-2666, UPMEM has higher data processing parallelism and lower memory access latency since a large number of DPUs are embedded in memory. However, the existing UPMEM system has high overheads for host side calling PIM modules, host-PIM communication, and inter-PIM communication. The memory bandwidth of a conventional 8GB DDR4-2666 DIMM is 21 GB/s. The latency of a CPU core accessing the conventional DDR4-2666 memory for a 64-bit memory word is roughly 90 ns. In the existing UPMEM system, the memory bandwidth is 630 MB/s in a PIM module at a DPU frequency of 350 MHz. The aggregate intra-PIM bandwidth is linear with the number of PIM modules. An 8GB UPMEM DIMM has 128 DPUs and its aggregate intra-PIM bandwidth is up to 79 GB/s. The latency of a PIM core accessing the PIM's memory for a 64-bit memory word in existing UPMEM is 14 ns. Therefore, the DPUs in UPMEM access memory faster than CPUs since they are closer to memory. In addition, if the system achieves load balance among all PIM modules, UPMEM can provide higher aggregate intra-PIM memory bandwidth than conventional DDR4-2666. Despite the DPUs in UPMEM accessing memory faster than CPUs, the latency for the host side calling DPUs in UPMEM is 0.5 milliseconds, which is much higher than the memory access latency. Invoking a DPU includes transferring input data to the DPU, initiating computation (dpu_launch()), and synchronizing for results. Besides, the interaction between user-space applications and the DPU system involves runtime libraries (e.g., libdpu) and system-level calls, which introduce additional software stack overhead, including context switching and queue management. The existing UPMEM system only offers 25 GB/s host-PIM communication bandwidth, which is smaller than the aggregate intra-PIM bandwidth. The existing UPMEM system does not support direct communication between PIM modules due to the scarce on-chip routing resources. The communication between PIM modules relies on host-PIM communication to exchange data, which involves loading data from one PIM module to the host's cache and then storing it to the destination PIM module. Considering the poor host-PIM communication ability, inter-PIM communication can easily become

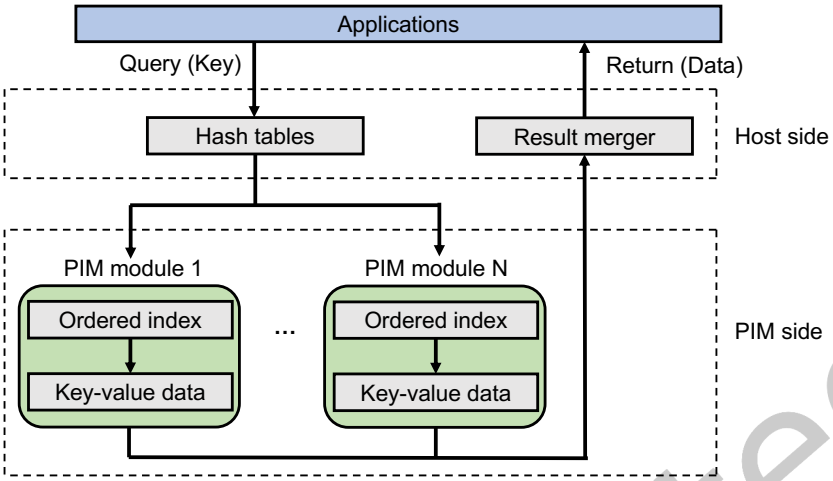


Fig. 3. PIM-based key-value database system.

the performance bottleneck. The existing UPMEM system only offers 25 GB/s total host-PIM and inter-PIM communication bandwidth.

2.3 PIM-based key-value database systems

Prior works [5, 6, 21, 22, 27] use PIM to accelerate queries over traditional key-value database systems, as shown in Figure 3. They partition the key space into multiple disjoint key ranges, and distribute data and ordered indexes (e.g., skiplist and B^+-tree) within these key ranges to PIM modules. Hash tables are employed on the host side to manage distributed data stored in PIM modules. To benefit from the parallelism across PIM modules and amortize the communication cost between PIM modules (inter-PIM) and between the host side and the PIM side (host-PIM), requests are dispatched to PIM modules in batches. For processing users' requests, prior works first locate the PIM module containing the requested data through hash tables on the host side, and then traverse the ordered indexes in PIM modules to retrieve the requested data. After retrieving requested data from PIM modules, these data are transmitted to the host side, merged by the result merger, and returned to applications. Prior works can be divided into two categories based on the methods for partitioning the key space and constructing ordered indexes: range partition [5, 6, 27] and node distribution [21, 22].

Range partition. Range partition designs [5, 6, 27] statically partition the entire key space into disjoint coarse-grained ranges with the same length, maintained by each PIM module. Each PIM module builds an independent ordered index based on their key-value pairs. The search path from top to bottom for each index operation is integral in one PIM module, without communication overhead between PIM modules. Since the system performance is determined by the busiest PIM module, for workloads that request for uniformly random keys, each PIM module handles a similar number of requests and range partition designs achieve high parallelism. However, for skewed workloads that concentrate on keys in a small subset of the partitions, some PIM modules are overwhelmed while the rest PIM modules are idle, resulting in load imbalance across PIM modules.

Node distribution. Node distribution designs [21, 22] aim to mitigate the load imbalance issue in range partition. Since the coarse granularity approach to partition the key space in range partition designs suffers from load imbalance, node distribution designs partition the key space into multiple disjoint fine-grained ranges across PIM modules. Since fine-grained key ranges are distributed across PIM modules, the search path from top to bottom

for each index operation is sliced into many segments in different PIM modules. At the end of each segment, index operations are redispatched to different PIM modules to the next segment, requiring inter-PIM communication. Frequently requested data are promoted from the PIM side to the host side for better load balance. However, promoting a large amount of data to the host side increases the load on the host side and leaves PIM modules underutilized.

In general, range partition designs [5, 6, 27] suffer from load imbalance across PIM modules under skewed workloads. Batches of requests concentrate on hot data in a small number of PIM modules, resulting in large gaps in the time taken by different PIM modules to process requests. Node distribution designs [21, 22] focus on addressing the load imbalance issue by adopting fine-grained methods for partitioning the key space and promoting hot data from the PIM modules to the host side. However, fine-grained key range partitioning results in high inter-PIM communication cost. Additionally, a large amount of data promoted to the host side increases the load on the host side and leaves PIM modules underutilized.

2.4 Existing partitioning and load balancing schemes in non-PIM distributed systems

Prior (non-PIM) designs adopt three schemes to distribute blockchain data to distributed devices: static sharding [9], dynamic sharding [14], and full replication [30].

Static sharding is one of the most common partitioning approaches in blockchain systems. Static sharding uses a fixed hash function to distribute data across distributed devices, with each shard being responsible for a subset of the overall data. Once the shards are allocated, the system does not dynamically adjust them based on changing workloads. Static sharding provides an easy-to-implement and scalable solution for large-scale distributed blockchains, but does not incorporate any load balancing schemes. It allows for parallel processing across different shards, leading to improved throughput and reduced bottlenecks in transaction processing. However, static sharding suffers from skewed access patterns with load imbalances across shards and lacks runtime adaptability. As some shards may experience higher transaction volumes than others, this can lead to uneven resource utilization and potential bottlenecks, thus degrading system performance. Additionally, static sharding lacks the ability to adapt to changing network conditions or data access patterns.

Compared with static sharding, dynamic sharding introduces flexibility in the partitioning process. In dynamic sharding, the system can adjust the number and size of the shards based on memory access patterns by migrating hot data across distributed devices. This dynamic reconfiguration helps in balancing the load more efficiently across the system, allowing it to respond to fluctuating demands in real-time. However, the complexity of managing dynamic shards increases the system's overhead. Maintaining data consistency and ensuring balanced shard distribution during dynamic adjustments introduce additional challenges, such as the need for complex coordination mechanisms, increased computational overhead, and substantial data migration costs across distributed devices.

Full replication is an approach in which every node in the blockchain network stores a complete copy of the entire blockchain's data. Every node keeps a full record of all blocks and transactions. Although this approach guarantees high data availability and redundancy, it results in significant storage requirements for each node, especially as the blockchain grows over time. Full replication ensures that every node has access to the complete state of the blockchain, enabling high reliability and fault tolerance. It also simplifies the process of verifying transactions, as each node independently validates the entire blockchain. However, the main limitation of full replication is its lack of scalability. Storing a full copy of the blockchain can become computationally expensive and inefficient, particularly as the size of the blockchain increases. This strategy is less suited for large-scale blockchains due to the heavy storage demands and the limited ability to handle high transaction volumes.

2.5 Motivation

PIM has potential to accelerate verifiable queries over blockchain database systems. However, directly applying techniques from existing PIM-based key-value database systems and non-PIM distributed systems to blockchain database systems faces several challenges. We summarize our motivations as follows.

First, the hash-based management for distributed data stored in PIM modules, designed for PIM-based key-value database systems, does not leverage the temporal characteristics of blockchain data distribution and results in significant metadata overhead. State-of-the-art works employ hash tables for managing distributed data in PIM modules, assuming that the data distribution does not follow a consistent pattern during runtime. The hash tables record the positions of data stored in PIM modules, facilitating data migration between PIM modules. The hash tables consume significant storage space. For example, the size of hash tables in [12] is approximately 10% of the total size of all key-value data. However, in blockchain systems, data blocks exhibit a temporal distribution pattern, as the time interval between block generation is determined by the underlying consensus protocol. For example, the Ethereum system [42] generates a block about every twelve seconds. This gives us an opportunity to build a regression function between timestamps and block heights for locating the block containing the requested data, supporting single-queries and multi-queries and lowering metadata overhead. *Thus, the management technique for distributed data stored in PIM modules should be redesigned to leverage the temporal characteristics of blockchain.*

Second, the key-range-based partitioning method for traditional indexes (e.g., skiplist and B^+-tree) is not suitable for Merkle-based indexes in blockchain systems, since traditional indexes are not used for data verification while MHT indexes are. Prior works partition the key space into multiple disjoint key ranges, and distribute data within a key range to the same PIM module for enhanced node locality. An ordered index is built for data within key ranges in each PIM module. In blockchain, however, data are grouped into blocks. Data within a key range exist in multiple blocks. Each block constructs an MHT [19, 31, 33] for data indexing and verification. Hash values in an MHT are calculated by data in the block and used for data verification. If data within a key range are distributed to the same PIM module and MHT indexes are built by data within key ranges across blocks rather than data within each block, the blockchain fails to verify data since the hash values in MHTs are modified. Furthermore, the MHT nodes returned as proof should be considered when enhancing node locality in each PIM module. *Therefore, the index partitioning scheme in blockchain database systems should be redesigned with consideration for data verification.*

Third, in prior works, the technique for achieving load balance across PIM modules results in excessive data and significant load on the host side. Since the load imbalance issue is attributed to hot data in PIM modules, state-of-the-art designs promote hot data from the PIM side to the host side and process requests for these hot data on the host side. Although promoting hot data lowers the number of requests processed in busy PIM modules and mitigates the load imbalance risk, previous designs do not consider the hotness changes of data. They only promote data that becomes hot to the host side and do not demote data that becomes cold to PIM modules during runtime, resulting in a large amount of data on the host side and leaving PIM modules underutilized. *As a result, the method for achieving load balance across PIM modules should adapt to data hotness changes during runtime.*

Fourth, existing blockchain partitioning and load balancing schemes in non-PIM distributed systems all have limitations and cannot be directly transferred to PIM systems due to differences in memory access models, communication bandwidth between distributed devices, computing capabilities of processing units, and storage capacity among distributed devices. Static sharding suffers from skewed access patterns with load imbalances across shards and lacks runtime adaptability. Even more critically, compared to traditional distributed architectures, the adverse effects of load imbalance are significantly amplified in PIM-based systems. This is primarily due to the fact that the computational capacity of a single PIM core is substantially weaker than that of a CPU core. As a result, PIM modules with heavier loads process requests much slower than those with lighter loads, and the overall system performance is dominated by the slowest module, consistent with the bottleneck effect. Furthermore, the

storage capacity of an individual PIM module is typically much smaller than that of a conventional distributed node. Consequently, load imbalance may not only degrade processing throughput but can also lead to data overflow in heavily loaded PIM modules, threatening the stability and correctness of the system. Dynamic sharding dynamically migrates data across distributed devices under different access patterns, thus achieving better load balance than static sharding. However, the load balancing scheme is not well-suited to PIM-based systems due to fundamental differences in the data access and inter-device communication models. Specifically, in PIM architectures, all requests must be dispatched from the host to individual PIM modules, and inter-PIM communication must be mediated by the host. For example, transferring data from one PIM module to another requires a two-phase process—data must first be promoted to the host side and then demoted to the target PIM module. This indirection introduces substantial communication overhead due to the limited host-PIM bandwidth, which is typically constrained to roughly 12.2 MB/s for each PIM module on average, significantly lower than the several or tens of GB/s inter-device bandwidth typically observed in distributed network environments. This limitation stems from the scarcity of on-chip communication resources in PIM systems. Moreover, the computational capability of PIM cores is significantly weaker than that of conventional CPU cores, making it challenging to support complex coordination protocols that are often required by dynamic sharding mechanisms such as rebalancing decisions. As a result, the overhead of dynamic coordination outweighs its potential benefits in the context of PIM, highlighting the need for lightweight, static, and topology-aware partitioning and load balancing strategies tailored to the architectural constraints of PIM systems. Full replication requires a large amount of storage space as the size of the blockchain increases. This method is highly unsuitable for PIM systems, as the storage capacity of a single PIM module is only 64 MB, which is insufficient to accommodate all the data in a blockchain.

3 Design

This section introduces the overview of Panther and the three key designs incorporated in Panther. Panther is a PIM-based blockchain database system supporting efficient single-queries and multi-queries. To exploit the temporal characteristics of blockchain, Panther incorporates a lightweight regression-based model for distributed data management. To preserve the data verification function of blockchain and enhance MHT node locality in each PIM module, Panther presents a subtree-based MHT index partitioning method. For load balance across PIM modules, Panther designs an adaptive data promotion and demotion mechanism.

3.1 Performance modeling of PIM-based blockchain systems

PIM has the potential to accelerate verifiable queries over blockchain database systems. We derive a general formula for average access latency as a function of key system parameters when processing a batch of query requests. The notations used in this section are summarized in Table 1. The total latency for processing a batch of requests consists of the inter-block indexing latency and the maximum intra-block indexing latency on both the host and PIM sides, as shown in Formula 1.

$$T = T_{inter} + max\{T_{host}, T_{pim}\} \tag{1}$$

The total inter-block indexing latency is calculated by the sum of inter-block indexing latency for each request, as shown in Formula 2.

$$T_{inter} = \sum_{i=1}^{N_{batch}} T_{inter\_i} \tag{2}$$

Since MHT nodes are connected by hash pointers, traversing MHTs on the host side involves both memory accesses for pointer chasing and hash computations performed by the CPU. For each query request, Panther

Table 1. Notations in this section

Notation	Definition
T_{inter}	The total latency for inter-block indexing
T_{host}	The total latency for traversing the promoted MHTs on the host side
T_{PIM}	The total latency for traversing the MHTs on the PIM side
N_{batch}	The number of requests in a batch
T_{cpu_mem}	The latency of a single memory access by the CPU
T_{pim_mem}	The latency of a single memory access by the PIM
T_{cpu_hash}	The latency for the CPU to compute a single hash function
T_{pim_hash}	The latency for the PIM to compute a single hash function
S_{host}	The average MHT size on the host side
S_{pim}	The average MHT size on the PIM side
N_{host}	The number of requests processed on the host side
N_{pim_max}	The number of requests processed by the busiest PIM module
T_{pim_comm}	The latency for communication between PIM modules

performs $\log S_{host}$ pointer chasing operations. Therefore, the total latency for traversing the promoted MHTs on the host side is calculated by Formula 3.

$$T_{host} = (T_{cpu\_mem} + T_{cpu\_hash}) \times \log S_{host} \times N_{host} \tag{3}$$

On the PIM side, traversing MHTs involves both memory accesses for pointer chasing and hash computations performed by PIM cores. For each query request, Panther performs $\log S_{pim}$ pointer chasing operations. The total latency of MHT traversal on the PIM side is determined by the busiest PIM module, i.e., the one processing the largest number of query requests. Moreover, the communication latency between PIM modules, incurred during cross-module MHT traversal, should also be taken into account. Therefore, T_{pim} is calculated by Formula 4.

$$T_{pim} = (T_{pim\_mem} + T_{pim\_hash}) \times \log S_{pim} \times N_{pim\_max} + T_{pim\_comm} \tag{4}$$

Panther incorporates several techniques to enhance the performance of the blockchain database system. Section 3.3 presents the regression-based model for distributed data management, aiming to lower the inter-block indexing latency and metadata cost. Section 3.4 introduces the subtree-based MHT index partition mechanism to minimize the communication cost between PIM modules. Section 3.5 shows the adaptive data promotion and demotion mechanism that balances the load across PIM modules, thereby reducing the number of requests handled by the busiest PIM module.

3.2 Overview of Panther

Figure 4 presents the overview of Panther. To achieve high parallelism across PIM modules and enhance MHT node locality within each PIM module, blocks are evenly distributed to PIM modules. On the PIM side, data and MHT of each block are stored in PIM modules. For workloads that request data with uniformly random keys, each PIM module handles a similar number of requests and the blockchain system exhibits high parallelism. However, for skewed workloads that concentrate on some hot data, the blockchain system may suffer from load imbalance across PIM modules. To address the load imbalance issue, Panther adaptively promotes MHT branches containing hot data to the host side and demotes MHT branches containing cold data to the PIM side with data hotness changes during runtime. This mechanism is detailed in Section 3.5 and is represented by orange dotted

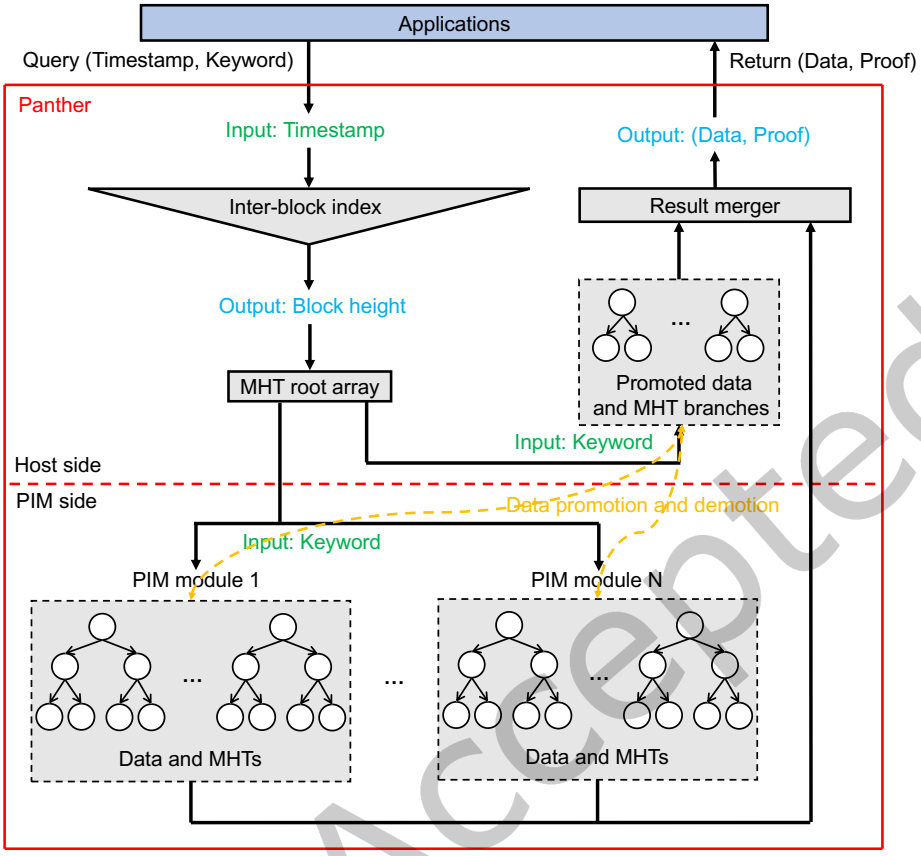
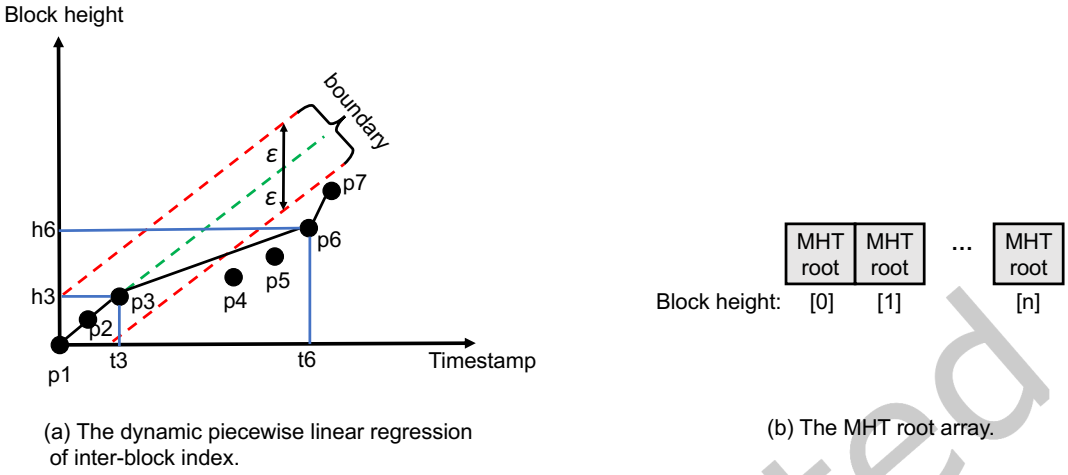


Fig. 4. Overview of Panther. Black arrows represent the data flow of query processing, while orange bidirectional dotted arrows indicate the paths of data promotion and demotion.

bidirectional arrows in Figure 4. The MHT branches include the MHT nodes on the search path for data indexing and the proof nodes for data verification. On the host side, Panther employs a regression-based inter-block index to leverage the temporal characteristics of blockchain, automatically mapping the relationship between timestamps and block heights for managing distributed data blocks. Compared with hash tables in prior works, the regression-based inter-block index requires less storage space for locating the block containing the requested data. Since block heights in blockchain are successive integers and the MHT root of each block is frequently used for data indexing and verification, MHT roots are stored in an array on the host side to optimize CPU cache locality.

The data flow of query processing is represented by black arrows in Figure 4. For processing a batch of single-queries and multi-queries from applications for data objects, Panther first uses timestamps of the blocks containing the requested data objects as input to retrieve the block heights through the regression-based inter-block index. Then, Panther uses the block heights as subscripts in the MHT root array to retrieve the MHT roots of these blocks. After that, Panther uses keywords of the requested data objects to traverse MHTs on the PIM side and the host side in parallel. Finally, the requested data and their proofs are merged on the host side and returned to applications.



(a) The dynamic piecewise linear regression of inter-block index.

(b) The MHT root array.

Fig. 5. The regression-based model for managing distributed data on the host and PIM sides.

3.3 Regression-based model for distributed data management

To better leverage the temporal characteristics of blockchain and lower the metadata overhead, Panther incorporates a lightweight regression-based model to manage distributed data on the host and PIM sides. The main idea is to dynamically extract the mapping relationship between timestamps and block heights through a piecewise linear function as shown in Figure 5(a), and store the MHT root of each block in an array as shown in Figure 5(b).

In Figure 5(a), the regression-based inter-block index consists of multiple linear regression functions. The error bound ε of each segment ensures that all data in the inter-block index can be retrieved, including the occurrence of outliers that deviate from the normal distribution. The upper and lower boundaries are defined as the regression function plus and minus the error bound, respectively. For example, Points 1, 2, and 3 are within the two boundaries (the red dotted lines) of the first linear regression function (the green dotted line). As point 4 lies outside the two boundaries, a new linear regression function is generated. Panther maintains the parameters of each segment, including the starting point, the slope, and the intercept. Since the timestamp values are increasing integers, Panther uses the binary search method to quickly locate the requested segment s. Using the timestamp t and the parameters of the segment, the predicted block height $pred_h(t)$ is calculated by

$$pred_h(t) = t \times s.slope + s.intercept \tag{5}$$

Recall that the actual block height $actual_h(t)$ is retrieved within in a range (i.e., error bound ε) from the $pred_h(t)$.

$$actual_h(t) \in [pred_h(t) - \varepsilon, pred_h(t) + \varepsilon] \tag{6}$$

The regression-based model uses a simple linear function to map timestamps to block heights. This model is fully characterized by just two parameters, slope and intercept, which provide an $O(1)$-space representation of the mapping throughout the block range. From an information-theoretic perspective, this linear model can be viewed as a lossy compressed representation of the full timestamp-to-height mapping. Compared to an explicit lookup table that stores each (timestamp, block height) pair at $O(N)$ space cost, our approach achieves significant compression while preserving sufficient accuracy for practical applications. This efficiency is enabled by the temporal regularity inherent in blockchain systems, which is a consequence of consensus protocols

(e.g., Proof of Work in Bitcoin or Proof of Stake in Ethereum) enforcing a roughly fixed block interval. This results in a strong linear correlation between timestamps and block heights. Our model captures the dominant low-entropy component—the global linear trend—while discarding the high-entropy residuals, which arise from stochastic block discovery delays, network variability, and transient difficulty adjustments. These residuals are typically zero-mean, weakly correlated, and high in entropy, making them costly to compress effectively. Our model intentionally discards them, embodying a classic rate-distortion trade-off in source coding. The "rate" is minimized (only two float parameters), while the "distortion" reflects the approximation error from ignoring residuals. Crucially, this distortion is acceptable in our target scenarios, such as transaction confirmation time estimation, macro-level on-chain trend analysis, or approximate cross-chain alignment—use cases where fine-grained precision is unnecessary, and the global temporal structure is sufficient.

The novelty of our approach lies in its explicit exploitation of the structural regularity inherent to blockchain systems—specifically, the approximate linearity between timestamps and block heights—to construct an extremely compact representation with $O(1)$ space complexity. While general-purpose lossless compression algorithms such as LZ77 or Huffman coding can be applied to (timestamp, block height) pairs, these methods are domain-agnostic. As a result, they are incapable of identifying and isolating the dominant linear trend present in the data. Although they may yield reasonable compression ratios, their space complexity remains $O(N)$, albeit with a reduced constant factor. Moreover, such approaches require storing and fully decompressing the entire dataset to perform mapping operations, which limits their practicality for real-time query scenarios. In contrast, our minimal representation not only reduces storage cost but also enables constant-time ($O(1)$) evaluation of the timestamp-to-height mapping via a simple linear function. This computational efficiency is a direct consequence of our model-based compression strategy, which is fundamentally unachievable with conventional compression techniques that depend on full-data reconstruction. The effectiveness of this minimal representation is grounded in information theory: the low entropy of the linear trend relative to the high entropy of the residuals. Source coding theory indicates that the minimal achievable rate for a given average distortion is defined by the rate-distortion function. Our fixed-rate (2 parameters), model-based approach provides a practical and highly efficient operating point on this curve for the specific source (blockchain timestamp/height data) and the acceptable distortion level pertinent to our target applications. The minimal representation of the mapping is unequivocally the pair of regression coefficients (slope, intercept). This constitutes a highly efficient, domain-aware, model-driven lossy compression scheme that exploits the fundamental linear structure inherent in blockchain timestamp data – a direct consequence of the consensus mechanism's target block rate. Framing our approach within source coding theory strengthens it by highlighting the exploitation of low-entropy structural regularity (the linear trend) and the inherent rate-distortion trade-off involved in discarding high-entropy residuals. Achieving $O(1)$ storage and $O(1)$ query complexity through this domain-specific insight, distinct from generic $O(N)$ compression methods, constitutes a core aspect of our method's novelty and efficiency.

In Figure 5(b), since data blocks are generated in sequence and block heights are successive integers, the MHT root with pointers to its child nodes in each block is sequentially stored in the MHT root array to optimize cache locality. After obtaining the required block heights from the inter-block index, Panther uses the block heights as subscripts in the MHT root array to retrieve the MHT roots of these blocks, and traverses the MHTs and the promoted MHT branches from the roots for data indexing and verification.

The regression-based model supports single-queries for data at a specific time and multi-queries for data within a time range. Compared with hash-table-based designs in prior works, the regression-based model requires less storage space to store parameters of the inter-block index and has better cache efficiency for managing distributed data blocks stored on the host and PIM sides. Note that the proposed regression-based model is not specific to Ethereum and can be readily applied to other blockchain systems such as Bitcoin, BNB Smart Chain, etc.

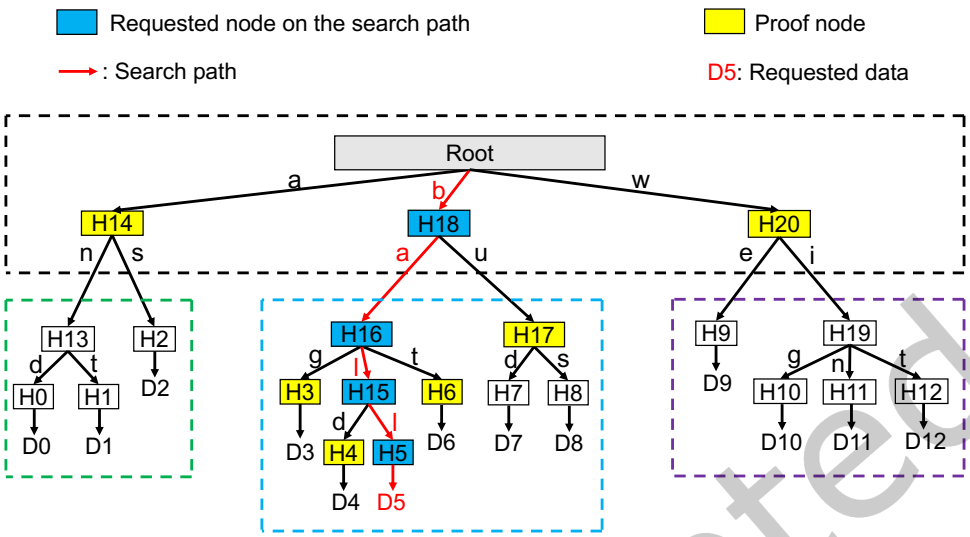


Fig. 6. Subtree-based MHT index partition. The black dashed box represents the host side and the dashed boxes with other colors represent different PIM modules.

3.4 Subtree-based MHT index partition

The index partitioning method for MHT in each block aims to distribute data and MHT indexes across PIM modules to achieve high parallelism with low inter-PIM communication cost. We observe that when retrieving a requested data object and its proof, all MHT nodes on the search path or used as proof reside within a single MHT subtree. Besides, nodes within other subtrees that do not contain the requested data are neither required on the search path nor as proof. Based on these observations, Panther stores all nodes in an MHT subtree for data indexing and verification within the same PIM module. This enhances node locality in each PIM module and reduces inter-PIM communication cost, as both the requested data object and proof nodes are retrieved from a single PIM module. MHT subtrees are evenly distributed across PIM modules to achieve high parallelism.

Figure 6 illustrates the subtree-based MHT index partitioning scheme and the data indexing and verification process. MHT combines two functions: data indexing and data verification. For data indexing, Panther uses the keyword traversing the MHT from root to leaf to retrieve the requested data (e.g., using the keyword "ball" to retrieve the requested data D5), indicated by the red arrows. For data verification, all sibling nodes along the search path are returned as proof. Thus, all ancestor nodes (blue nodes) and their sibling nodes (yellow nodes) are required when retrieving the requested data. To retrieve any data in the MHT, the MHT root node and its child nodes within the black dashed box are required either on the search path or as proof. These frequently required nodes are stored on the host side to leverage the large CPU caches for better cache locality. In an MHT, there are multiple subtrees whose subtree root is the child node of the MHT root. Nodes within these subtrees that do not contain the requested data are neither required on the search path nor as proof (e.g., nodes in the green and purple dashed boxes are not required when searching for D5). These subtrees are stored in different PIM modules for high parallelism across PIM modules. Since different subtrees may contain different amounts of data, for load balance across PIM modules, Panther records the size of data stored in each PIM module and adopts a greedy strategy to store the subtrees from largest to smallest in the PIM modules with the least amount of data. In this way, each PIM module stores a similar amount of data. Additionally, for each requested data object, all

required nodes on the search path or as proof are retrieved from a single PIM module, reducing the inter-PIM communication cost.

Designed based on the required nodes for data indexing and verification functions of MHT, the subtree-based MHT index partitioning mechanism preserves the data verification function without modifying hash values in each MHT, and achieves high parallelism across PIM modules with low inter-PIM communication cost.

3.5 Adaptive data promotion and demotion

Load imbalance across PIM modules is a common issue for PIM-based systems under skewed workloads, where requests concentrate on hot data objects stored in a small number of PIM modules. To address the load imbalance issue, Panther dynamically promotes hot data to the host side and demotes cold data to the PIM side with data hotness changes during runtime. Data objects along with their MHT branches, including the MHT nodes on the search path for indexing and the proof nodes for verifying the data objects, are adaptively promoted and demoted between the host and PIM sides. This approach significantly reduces the communication cost between the two sides for processing queries, as the requested data and proof nodes are retrieved from one side. To avoid promoting excessive data to the host side, which would leave PIM modules underutilized, Panther bases its data promotion and demotion strategy on the request processing time on the host side (T_{host}) and the PIM side (T_{pim}).

Case 1: If $T_{host} < T_{pim}$, T_{host} is completely overlapped by T_{pim} since the two sides process requests in parallel. Panther promotes recently requested data objects to the host side but does not demote any data object to the PIM side, as the load on the host side is not heavy. Given that the performance bottleneck on the PIM side is determined by the busiest PIM module (i.e., the PIM module that processes the highest number of requests), data promotion is greedily applied to PIM modules in sequence from busy to idle.

Case 2: If $T_{host} \geq T_{pim}$, the load on the host side is heavy. To prevent the load on the host side from continually increasing, Panther demotes multiple cold data objects to the PIM side when promoting a hot data object to the host side. Data promotion and demotion decisions are made based on whether they reduce the load imbalance across PIM modules. The performance bottleneck on the PIM side is determined by the busiest PIM module. Since the block generation interval is determined by the underlying consensus protocol and the number of data objects per block remains relatively stable (i.e., the temporal characteristics of data distribution inherent to blockchain database systems), different requests have a similar resource requirement. Therefore, the demotion mechanism is designed based on the number of requests processed on the host and PIM sides. The maximum load across PIM modules before data promotion and demotion is represented by the number of requests processed by the busiest PIM module.

$$Load_{max} = Num(P_{busiest}) \tag{7}$$

where $Load_{max}$ is the maximum load across PIM modules, and $Num(P_{busiest})$ is the number of requests processed by the busiest PIM module before data promotion and demotion. Demoting multiple least-frequently-requested data objects, with a similar number of requests as the promoted data object, from the host side to their PIM modules increases the loads of these PIM modules. Assume that the loads of n PIM modules increase.

$$Load_i = Num(P_i) + Num(D_i) \tag{8}$$

where $i = 1, 2, \ldots, n$. $Load_i$ is the load of each PIM module, $Num(P_i)$ is the number of requests processed by each PIM module before data demotion, and $Num(D_i)$ is the number of requests for the data objects demoted to each PIM module. The maximum load $Load'_{max}$ across these PIM modules is represented by the number of requests processed by the busiest PIM module after data promotion and demotion.

$$Load'_{max} = max\{Load_1, Load_2, ..., Load_n\} \tag{9}$$

If $Load'_{max} < Load_{max}$, Panther promotes the hot data object and demotes the cold data objects since the load imbalance across PIM modules is reduced. Otherwise, Panther does not perform data promotion and demotion.

The data promotion and demotion adapt to data hotness changes and to the degree of parallelism between the host and PIM sides, reducing the load imbalance across PIM modules and achieving high parallelism between the host side and the PIM side.

4 Evaluation

In this section, we evaluate the performance of Panther against state-of-the-art traditional and PIM-based blockchain database systems in multiple datasets. Since the traditional design is unable to run on the PIM hardware, as in previous works [5, 6, 12, 22, 39], it runs on traditional hardware for comparison with five PIM-based designs, thereby demonstrating the ability of PIM to alleviate the memory bottleneck. Panther is then compared with the other four PIM-based designs on the PIM hardware to highlight the effectiveness of our novel contributions.

4.1 Experimental setup

Datasets. Three real-world datasets are used in the experiments. (1) Ethereum (ETH) [42]: The ETH dataset is extracted from the Ethereum system from September 2 to September 11, 2024. It contains 72,000 blocks with 10 million transaction records. Each record is in the form of $\langle timestamp, amount, addresses \rangle$, where *amount* is the transfer amount and *addresses* are the addresses of senders and receivers. The block size is around 100 kilobytes and the MHT size is a few kilobytes. (2) Bitcoin [30]: The Bitcoin dataset is extracted from the Bitcoin system from June 12 to September 11, 2024. It contains 13,393 blocks with 44 million transaction records. Each record is in the form of $\langle timestamp, amount, addresses \rangle$, where *amount* is the transfer amount and *addresses* are the addresses of senders and receivers. The block size is a few megabytes and the MHT size is around 100 kilobytes. (3) Foursquare (4SQ) [50]: The 4SQ dataset includes 1 million user check-in records with timestamps. Records within every 30 seconds are packed in a block. Each record is in the form of $\langle timestamp, [longitude, latitude], check\text{-}in\ place's\ keywords \rangle$.

Compared systems. We use the blockchain database system in Ethereum [42] as a baseline system without using PIM. To demonstrate the effectiveness of our proposed techniques, we implement four PIM-based blockchain database systems employing techniques in previous works to replace one of our techniques in Panther. (1) HT-Chain: HT-Chain employs the hash tables in [12] to replace the regression-based model for managing distributed data on the host and PIM sides. Hash tables are built on the host side to locate the PIM module containing the requested data. (2) SDB-Chain: SDB-Chain employs the block-based index partitioning technique in [23] to replace the subtree-based MHT index partition. Data blocks are randomly distributed to PIM modules. (3) SDN-Chain: SDN-Chain employs the node-based index partitioning technique in [21] to replace the subtree-based MHT index partition. Index nodes of each MHT are randomly distributed to PIM modules. (4) PP-Chain: PP-Chain employs the data promoting technique proposed in [22] to replace the adaptive data promotion and demotion. Hot data objects are promoted from the PIM side to the host side for load balance across PIM modules, without data demotion from the host side to the PIM side. (5) SS-Chain: SS-Chain employs the static sharding scheme [9] widely used in distributed architectures. Static sharding uses a fixed hash function to distribute data across distributed devices, with each shard being responsible for a subset of the overall data. (6) DS-Chain: DS-Chain employs the dynamic sharding scheme [14] widely used in distributed architectures. Dynamic sharding adjusts the number and size of the shards by migrating hot data across distributed devices. For fair comparison, all PIM-based and baseline designs adopt the same blockchain structure, operate on identical datasets, use identical metrics to quantitatively assess their performance, and use similar computing and storage resources.

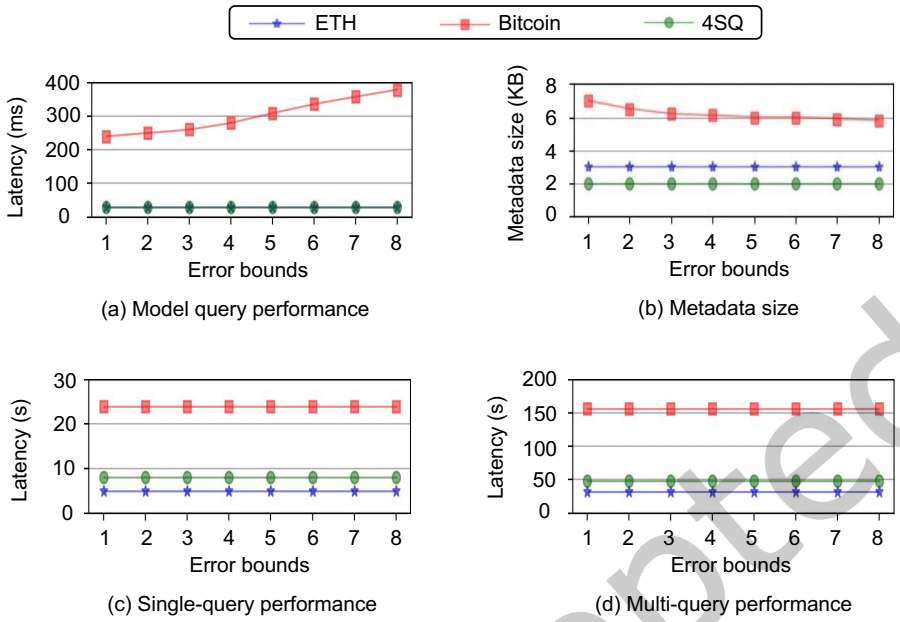


Fig. 7. The impact of error bounds in Panther.

Testbed. Our testbed machine is a dual-socket server with two Intel(R) Xeon(R) Silver 4216 CPUs [7], each equipped with 16 cores at 2.10 GHz and 22 MB cache. Each socket has four memory channels: four DDR4-2666 DIMMs [17] are installed on two channels and four UPMEM DIMMs [8] are on the other two channels. Each UPMEM DIMM has 128 PIM modules. Since the baseline design is unable to run on PIM hardware, for fair comparisons, PIM-based designs and the baseline design utilize similar computing and memory resources. Panther and other PIM-based designs utilize one dedicated CPU core, one UPMEM DIMM, and one DDR4-2666 DIMM. The baseline design utilizes 32 CPU cores and two DDR4-2666 DIMMs.

Workload setup. In all experiments, we first construct the blockchain using data in each dataset, then evaluate the blockchain system performance by running 10 million query operations. Each batch contains 100 thousand queries. The query keywords are selected from the most frequently requested 10 thousand keywords. Each single-query retrieves one data object from a block, and each multi-query retrieves data objects from 2 to 128 blocks within a time range.

4.2 Impact of error bounds

We evaluate the impact of various error bounds in our regression-based model for distributed data management. Figure 7(a) illustrates the average latency for each batch of queries to retrieve the requested MHT roots by querying the regression-based model. The blockchain using the ETH and 4SQ datasets consumes less time to retrieve MHT roots of the requested blocks than the blockchain using the Bitcoin dataset. This is because the relationship between timestamps and block heights in ETH and 4SQ is more regular than that in Bitcoin, contributing to less time to determine the block height using the timestamp as input. The model query latency for ETH and 4SQ remains constant as the error bound increases, since the timestamp and block height follow a linear regression. The model query latency for Bitcoin increases as the error bound grows, since more block traversals are needed within a larger error bound. Figure 7(b) shows the metadata size of the regression-based

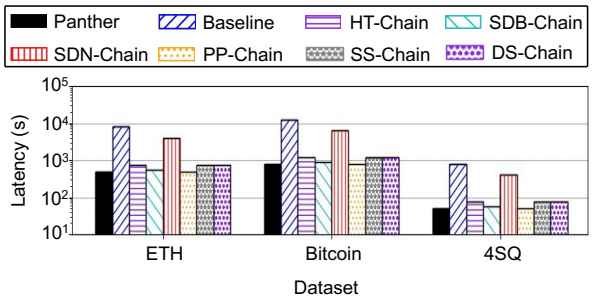


Fig. 8. Blockchain construction performance.

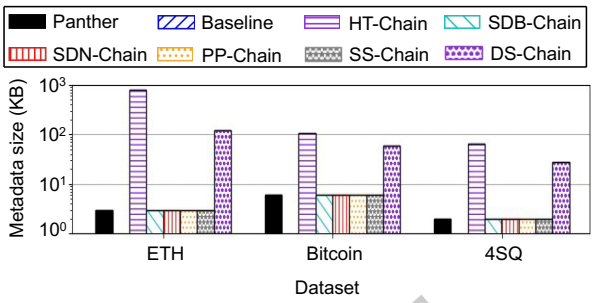


Fig. 9. Metadata storage space.

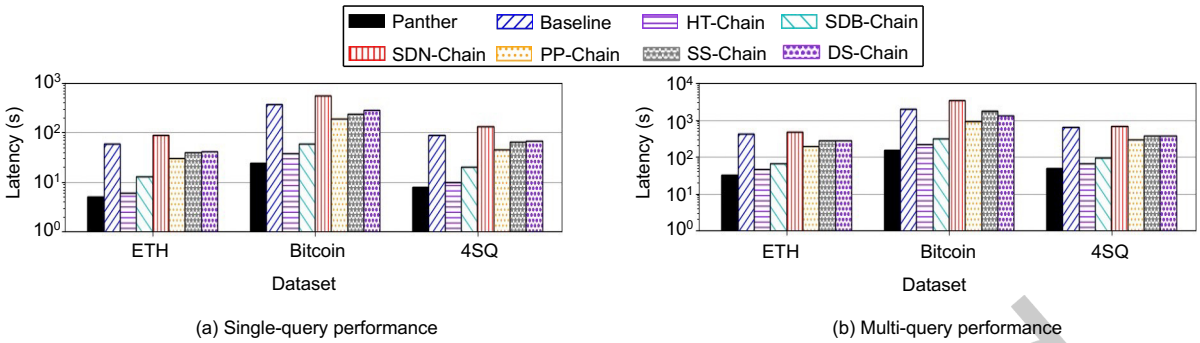
model. The blockchain using the 4SQ dataset has the smallest metadata size, as the 4SQ dataset has the fewest blocks. The blockchain using the Bitcoin dataset has the largest metadata size, as the inter-block index uses the greatest number of linear regression functions to map the relationship between timestamps and block heights. The single-query and multi-query performance for each batch of requests are shown in Figure 7(c) and Figure 7(d), respectively. Since the model query time is far smaller than the MHT traversal time, the latency for processing each batch of single-query and multi-query requests remains almost constant as the error bound increases. For the remaining experiments in this paper, we present our results with an error bound value of 1.

4.3 Blockchain construction performance

Figure 8 illustrates the blockchain construction latency for the three datasets. Baseline consumes 2.3x to 16.1x more time than PIM-based designs to construct blockchains, as it does not leverage the high parallelism and fast memory access of PIM modules. SDN-Chain consumes the most time among all PIM-based designs. This is because data and nodes of MHT in each block are distributed to multiple PIM modules, contributing to significant communication cost between PIM modules. The latency for HT-Chain to construct blockchain is 36.2% higher than SDB-Chain and 51.7% higher than PP-Chain and Panther on average, due to the additional time to build hash tables on the host side for managing data in PIM modules. SS-Chain and DS-Chain exhibit comparable execution times, as the load balancing scheme has negligible impact on the blockchain construction phase. Compared with Panther and PP-Chain, SDB-Chain consumes 11.8% more time on average for blockchain construction. The data block distribution strategy in SDB-Chain does not leverage the parallelism between the host and PIM sides, leading to high MHT construction cost on the PIM side. Panther and PP-Chain consume the same time, as the adaptive data promotion and demotion are triggered during the workload runtime rather than the blockchain construction process.

4.4 Metadata storage space

The metadata sizes of state-of-the-art blockchain database systems are shown in Figure 9. Baseline requires no metadata to manage distributed data since it does not utilize PIM devices. HT-Chain requires the largest metadata storage space due to the large hash tables used to record the mapping relationship between timestamps and block heights, as well as the PIM modules that store the requested blocks. DS-Chain introduces additional metadata overhead to track data migration trajectory. Compared with HT-Chain and DS-Chain, the metadata size of Panther is reduced by 1 to 2 orders of magnitude, since the regression-based model leverages the temporal characteristics of blockchain for distributed data management. SDB-Chain, SDN-Chain, PP-Chain, and SS-Chain have the same metadata size as Panther, as they employ the same regression-based model introduced in this paper to manage distributed data on the host and PIM sides.



(a) Single-query performance

(b) Multi-query performance

Fig. 10. Query performance.

4.5 Query performance

Figure 10(a) and Figure 10(b) show the average latency for processing each batch of verifiable single-queries and multi-queries, respectively. The query latency includes the full node's query processing latency and the light node's data verification latency. Panther consumes the least time and achieves up to 23.6x and 19.6x speedups over state-of-the-art designs for processing single-queries and multi-queries, respectively. The speedup stems from three key optimizations. First, Panther significantly reduces the inter-PIM communication overhead to retrieve requested data. This is achieved via our subtree-based MHT index partitioning mechanism (introduced in Section 3.4), which ensures that all nodes of an MHT subtree used for indexing and verification are stored within the same PIM module. Second, Panther lowers the inter-block indexing cost. This is because our proposed regression-based model (introduced in Section 3.3) provides better cache locality compared to traditional hash table-based approaches. Third, Panther fully leverages parallelism across PIM modules and between the host and PIM sides, enabled by our adaptive data promotion and demotion mechanism (introduced in Section 3.5), which mitigates load imbalance across PIM modules and between the host and PIM sides. SDN-Chain consumes the most time for processing verifiable queries due to the significant inter-PIM communication cost to retrieve requested data and proofs. SS-Chain suffers from significant load imbalance across PIM modules, resulting in inferior performance compared to both PP-Chain and DS-Chain. While DS-Chain adopts a load balancing mechanism to enhance parallelism across PIM modules, its performance still lags behind PP-Chain. This is primarily because data migration in DS-Chain is mediated by the host, introducing considerable communication overhead due to the limited host-PIM bandwidth. HT-Chain consumes 40.3% and 58.3% more time than Panther for processing single-queries and multi-queries respectively, since the regression-based model in Panther has better cache efficiency than hash tables in HT-Chain. SDB-Chain consumes 2.1x and 2.6x more time than Panther for processing single-queries and multi-queries, respectively. This is because SDB-Chain stores all contents of a block in one PIM module, reducing the parallelism across PIM modules. The latency for PP-Chain to process single queries and multi-queries is 6.2x and 7.9x higher than that for Panther, respectively. PP-Chain promotes excessive data to the host side, which leaves PIM modules underutilized.

5 Conclusion

This paper presents Panther, the first PIM-based blockchain database system that supports efficient verifiable single-queries and multi-queries. Panther incorporates three key designs: (1) a regression-based model that leverages the temporal characteristics of blockchain for distributed data management; (2) a subtree-based MHT index partitioning scheme that achieves high parallelism across PIM modules with low inter-PIM communication

cost; and (3) an adaptive data promotion and demotion mechanism that mitigates load imbalance across PIM modules. In multiple datasets, Panther achieves up to 23.6× speedup for single-query, 19.6× speedup for multi-query, 16.1× speedup for blockchain construction, and reduces metadata storage by orders of magnitude compared to state-of-the-art designs.

Acknowledgments

This work is supported by National Natural Science Foundation of China (NSFC) (Grant No. 62227809), National Key Research and Development Program of China (Grant No. 2023YFB4502902), National Natural Science Foundation of China (NSFC) (Grant No. 62332012, 62302290), the Fundamental Research Funds for the Central Universities, and Key Research and Development Program of Xinjiang Uygur Autonomous Region (Grant No. 2023B01027, 2023B01027-1), Xinjiang Uygur Autonomous Region Science and Technology Innovation Team Project (2024TSYCTD0007).

References

[1] McKee. S. A. 2004. Reflections on the memory wall. In *Proceedings of the 1st conference on Computing frontiers*. 162.
[2] Wulf. W. A and McKee. S. A. 1995. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH computer architecture news* 23, 1 (1995), 20–24.
[3] S. Aggarwal and N. Kumar. 2021. Hyperledger. *Advances in computers* (2021), 323–343.
[4] J. Chang, B. Li, J. Lin, et al. 2023. Anole: A Lightweight and Verifiable Learned-Based Index for Time Range Query on Blockchain Systems. In *International Conference on Database Systems for Advanced Applications*. 519–534.
[5] J. Choe, A. Crotty, T. Moreshet, M. Herlihy, and R. I. Bahar. 2022. HybriDS: Cache-Conscious Concurrent Data Structures for Near-Memory Processing Architectures. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*. 321–332.
[6] J. Choe, A. Huang, T. Moreshet, M. Herlihy, and R. I. Bahar. 2019. Concurrent Data Structures with Near-Data-Processing: an Architecture-Aware Implementation. In *Proceedings of the 31st ACM Symposium on Parallelism in Algorithms and Architectures*. 297–308.
[7] Intel Corporation. 2023. Intel Xeon Silver 4216 Processor. https://ark.intel.com/content/www/us/en/ark/products/192444/intel-xeon-silver-4216-processor-16-5m-cache-2-10-ghz.html
[8] UPMEM Corporation. 2021. UPMEM DIMM. https://www.upmem.com/technology
[9] Kokoris-Kogias E, Jovanovic P, Gasser L, et al. 2018. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *IEEE symposium on security and privacy*. 583–598.
[10] A. Fatima, S. Liu, K. Seemakhupt, R. Ausavarungnirun, and S. Khan. 2023. vPIM: Efficient Virtual Address Translation for Scalable Processing-in-Memory Architectures. In *Proceedings of the 60th ACM/IEEE Design Automation Conference*. 1–6.
[11] Weisz. G, Melber. J, Wang. Y, et al. 2016. A study of pointer-chasing performance on shared-memory processor-FPGA systems. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 264–273.
[12] Y. Hua, S. Zheng, W. Kong, C. Zhou, K. Huang, R. Ma, et al. 2024. RADAR: A Skew-resistant and Hotness-aware Ordered Index Design for Processing-in-memory Systems. *IEEE Transactions on Parallel and Distributed Systems* 35, 9 (2024), 1598–1614.
[13] Hu. J, Wei. Z, Chen. J, et al. 2023. RWORT: A Read and Write Optimized Radix Tree for Persistent Memory. In *Proceedings of the IEEE 41st International Conference on Computer Design*. 194–197.
[14] Wang J and Wang H. 2019. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th USENIX symposium on networked systems design and implementation*. 95–112.
[15] Zhang. J, Wu. S, Tan. Z, et al. 2019. S3: A scalable in-memory skip-list index for key-value store. In *Proceedings of the VLDB Endowment*. 2183–2194.
[16] J. Jang, H. Kim, and H. Lee. 2023. Characterizing Memory Access Patterns of Various Convolutional Neural Networks for Utilizing Processing-in-Memory. In *International Conference on Electronics, Information, and Communication*. 1–3.
[17] JEDEC. 2017. DDR4 SDRAM Specification. https://www.jedec.org/standards-documents/docs/jesd79-4b
[18] T. Jeong and E. Y. Chung. 2024. PipePIM: Maximizing Computing Unit Utilization in ML-Oriented Digital PIM by Pipelining and Dual Buffering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 12 (2024), 4585–4598.
[19] S. Jing, X. Zheng, and Z. Chen. 2021. Review and Investigation of Merkle Tree's Technical Principles and Related Application Fields. In *International Conference on Artificial Intelligence, Big Data and Algorithms*. 86–90.
[20] Yang. K, Ma. X, Thirumuruganathan. S, et al. 2021. Random walks on huge graphs at cache efficiency. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 311–326.
[21] H. Kang, P. B. Gibbons, G. E. Blelloch, L. Dhulipala, Y. Gu, and C. McGuffey. 2021. The Processing-in-Memory Model. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*. 295–306.

[22] H. Kang, P. B. Gibbons, G. E. Blelloch, L. Dhulipala, Y. Gu, C. McGuffey, et al. 2022. PIM-tree: A Skew-resistant Index for Processing-in-Memory. In *Proceedings of the VLDB Endowment 16, 4*. 946–958.
[23] H. Kang, Y. Zhao, P. B. Gibbons, G. E. Blelloch, L. Dhulipala, Y. Gu, C. McGuffey, et al. 2023. PIM-trie: A Skew-resistant Trie for Processing-in-Memory. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*. 1–14.
[24] J. H. Kim, S. Kang, S. Lee, H. Kim, W. Song, Y. Ro, et al. 2021. Aquabolt-XL: Samsung HBM2-PIM with in-memory processing for ML accelerators and beyond. In *IEEE Hot Chips 33 Symposium*. 1–26.
[25] J. Leng, M. Zhou, J. L. Zhao, Y. Huang, and Y. Bian. 2022. Blockchain Security: A Survey of Techniques and Research Directions. *IEEE Transactions on Services Computing* 15, 4 (2022), 2490–2510.
[26] C. Li, Z. Zhou, S. Zheng, J. Zhang, Y. Liang, and G. Sun. 2024. SpecPIM: Accelerating Speculative Inference on PIM-Enabled System via Architecture-Dataflow Co-Exploration. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 950–965.
[27] Z. Liu, I. Calciu, M. Herlihy, and O. Mutlu. 2017. Concurrent Data Structures for Near-Memory Computing. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*. 235–245.
[28] Vilim. M, Rucker. A, and Olukotun. K. Aurochs. 2021. An architecture for dataflow threads. In *Proceedings of the ACM/IEEE 48th Annual International Symposium on Computer Architecture*. 402–415.
[29] V. Mardiansyah, A. Muis, and R. F. Sari. 2023. Multi-State Merkle Patricia Trie (MSMPT): High-Performance Data Structures for Multi-Query Processing Based on Lightweight Blockchain. *IEEE Access* 11 (2023), 117282–117296.
[30] S. Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf
[31] M. Niaz and G. Saake. 2015. Merkle Hash Tree based Techniques for Data Integrity of Outsourced Data. *GvD* (2015).
[32] J. K. Oladele, A. A. Ojugo, C. C. Odiakaose, F. U. Emordi, R. A. Abere, et al. 2024. BEHeDaS: A Blockchain Electronic Health Data System for Secure Medical Records Exchange. *Journal of Computing Theories and Applications* 1, 3 (2024), 231–242.
[33] T. Osterland, G. Lemme, and T. Rose. 2021. Discrepancy Detection in Merkle Tree-Based Hash Aggregation. In *IEEE International Conference on Blockchain and Cryptocurrency*. 1–9.
[34] Jain. P, Jain. A, Nrusimha. A, et al. 2020. Checkmate: Breaking the memory wall with optimal tensor rematerialization. In *Proceedings of Machine Learning and Systems*. 497–511.
[35] Yang. S. P, Kim. M, Nam. S, et al. 2023. Overcoming the memory wall with CXL-EnabledSSDs. In *2023 USENIX Annual Technical Conference*. 601–617.
[36] J. Park, S. Lee, and J. Lee. 2023. NTT-PIM: Row-Centric Architecture and Mapping for Efficient Number-Theoretic Transform on PIM. In *Proceedings of the 60th ACM/IEEE Design Automation Conference*. 1–6.
[37] Y. Peng, M. Du, F. Li, R. Cheng, and D. Song. 2020. FalconDB: Blockchain-based Collaborative Database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 637–652.
[38] Zhang. Q, Song. H, Zhou. K, et al. 2024. A prefetching indexing scheme for in-memory database systems. *Future Generation Computer Systems* 156 (2024), 179–190.
[39] Ma. R, Zheng. S, Wang. G, et al. 2024. Accelerating Regular Path Queries over Graph Database with Processing-in-Memory. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.
[40] Assadi. S, Kol. G, Saxena. R. R, et al. 2020. Multi-pass graph streaming lower bounds for cycle counting, max-cut, matching size, and other problems. In *Proceedings of the IEEE 61st Annual Symposium on Foundations of Computer Science*. 354–364.
[41] L. Tseng, X. Yao, S. Otoum, M. Aloqaily, and Y. Jararweh. 2020. Blockchain-based database in an IoT environment: challenges, opportunities, and analysis. *Cluster Computing* 23 (2020), 2151–2165.
[42] G. Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. https://ethereum.github.io/yellowpaper/paper.pdf
[43] H. Wu, Z. Peng, S. Guo, Y. Yang, and B. Xiao. 2022. VQL: Efficient and Verifiable Cloud Query Services for Blockchain Systems. *IEEE Transactions on Parallel and Distributed Systems* 33, 6 (2022), 1393–1406.
[44] C. Xu, C. Zhang, and J. Xu. 2019. VChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases. In *Proceedings of the 2019 International Conference on Management of Data*. 141–158.
[45] Huang. Y, Zhu. H, and Li. Y. 2018. Performance Improvements by Deploying L2 Prefetchers with Helper Thread for Pointer-Chasing Applications. *International Journal of Performability Engineering* 14, 10 (2018), 2312.
[46] Hua. Y, Huang. K, Zheng. S, et al. 2021. PMSort: An adaptive sorting engine for persistent memory. *Journal of Systems Architecture* 120 (2021), 102279.
[47] Hua. Y, Huang. K, Zheng. S, et al. 2021. Redesigning the Sorting Engine for Persistent Memory. In *Database Systems for Advanced Applications: 26th International Conference*. 393–412.
[48] Hua. Y, Zheng. S, Yin. J, et al. 2023. Bumblebee: A MemCache design for die-stacked and off-chip heterogeneous memory systems. In *2023 60th ACM/IEEE Design Automation Conference*. 1–6.
[49] Hua. Y, Zheng. S, Kong. W, et al. 2025. Phoenix: A Dynamically Reconfigurable Hybrid Memory System Combining Caching and Migration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 44, 3 (2025), 1126 – 1140.

[50] D. Yang, D. Zhang, and B. Qu. 2016. Participatory Cultural Mapping Based on Collective Behavior Data in Location-Based Social Networks. *ACM Trans. Intell. Syst. Technol* 7, 3 (2016), 1–23.

Received 27 February 2025; revised 22 July 2025; accepted 10 September 2025