Universal Library of Innovative Research and Studies Research Article

ISSN: 3065-0003 | Volume 3, Issue 1

Open Access | PP: 13-20

DOI: https://doi.org/10.70315/uloap.ulirs.2026.0301003

State Management in Distributed Systems Using the Example of Multi-Version Concurrency Control (MVCC)

Brandon Vrooman

Software Engineer, Innobit Inc, Toronto, Canada.

Abstract

This article analyzes state management in distributed systems based on Multi-Version Concurrency Control (MVCC). The relevance of this work is driven by the growth of high-load blockchain platforms and agentic AI systems, for which scalability and deterministic state consistency are critical requirements. The novelty of the research lies in the comprehensive examination of MVCC not as a local mechanism for transaction isolation, but as a central state management layer integrated with architectures for replication, consensus, inter-shard interaction, and disaggregated memory. The paper describes modern approaches to version garbage collection, lock-free data structures, hybrid blockchain-DBMS systems, cloned control protocols, and cross-shard transactions; their limitations and areas of applicability are studied. Particular attention is paid to identifying architectural patterns and recommendations for high-performance L2 blockchains and agentic AI platforms. To achieve the set objective, methods of comparative and systemic analysis of scientific publications are employed. The conclusion describes findings regarding the role of MVCC as a state management core and formulates directions for further research. The article will be useful to engineers and researchers involved in the design of large-scale distributed, blockchain, and AI systems.

Keywords: *Distributed Systems, Multi-Version Concurrency Control, Blockchain, Cloned Concurrency Control, Hybrid Blockchain-DBMS, Agentic AI Platforms, Disaggregated Memory, State Management.*

INTRODUCTION

Modern distributed systems—ranging from blockchain platforms and financial infrastructures to agentic AI networks—face the necessity of simultaneously ensuring high throughput, strict consistency, and deterministic reproduction of computations. The growth in load, the increasing complexity of protocols, and the emergence of architectures with disaggregated memory intensify the requirements for state management layers: they must support massive parallelism, fault tolerance, and behavior auditability without an exponential increase in overhead costs. Against this background, Multi-Version Concurrency Control (MVCC) is outgrowing its role as a local transaction isolation mechanism and is becoming a key tool for organizing consistent snapshots and controlled state evolution in a distributed environment.

The objective is to systematize modern approaches to multi-version concurrency control in distributed systems and demonstrate how MVCC functions as a state management layer in high-performance blockchain platforms and agentic AI networks. The tasks are:

To compare theoretically oriented and engineering MVCC schemes, covering version garbage collection, lock-free data structures, memory-optimized solutions, and architectures with disaggregated memory.

To analyze the interaction of MVCC with protocols for consensus, replication, and inter-shard exchange in blockchain systems and transactional clusters.

To identify robust architectural patterns of state management for L2 blockchains, permissioned consortiums, and agentic AI platforms, and to formulate recommendations for designing the state layer.

The novelty of the work lies in considering MVCC through the prism of the entire distributed system architecture, rather than a separate DBMS or node: multi-versioning is interpreted as a common language for state reconciliation between replicas, shards, chains, and agents. Unlike existing studies focused either on local concurrency control algorithms or on specific aspects of blockchain storage, this article offers a holistic picture linking multi-version GC, formally verifiable transactional cores, hybrid blockchain-DBMS systems,

Citation: Brandon Vrooman, "State Management in Distributed Systems Using the Example of Multi-Version Concurrency Control (MVCC)", Universal Library of Innovative Research and Studies, 2026; 3(1): 13-20. DOI: https://doi.org/10.70315/uloap.ulirs.2026.0301003.

cloned control protocols, and MVCC-oriented cross-sharding schemes into a single state management model.

MATERIALS AND METHODS

Modern scientific works reflecting various aspects of MVCC application in distributed systems, databases, and blockchain architectures were used as research materials. N. Ben-David, G. E. Blelloch, P. Fatourou, et al. proposed schemes for multi-version garbage collection with simultaneous time and space bounds and integration with wait-free data structures [1]. Y.-S. Chang, R. Jung, U. Sharma, J. Tassarotti, M. F. Kaashoek, and N. Zeldovich developed and formally verified the vMVCC transactional library, demonstrating the possibility of strictly proving the correctness of a high-performance MVCC core [2]. M. Freitag, A. Kemper, and T. Neumann investigated memory-optimized MVCC for disk-based DBMSs, showing how separating "hot" and "cold" versions between memory and disk affects latency and storage costs [3]. Z. Ge, D. Loghin, B. C. Ooi, P. Ruan, and T. Wang analyzed hybrid blockchain-DBMS systems where the transactional model and MVCC are implemented in the database, while the blockchain serves as an immutable log and consensus layer [4]. J. Helt, A. Sharma, D. J. Abadi, W. Lloyd, and J. M. Faleiro proposed the C5 protocol for cloned concurrency control in primary–backup configurations, ensuring limited replica lag while maintaining parallelism [5]. J. Kalajdjieski, M. Raikwar, N. Arsov, G. Velinov, and D. Gligoroski presented a review of databases suitable for use in a blockchain context, systematizing data and transaction models, including multi-version approaches [6]. W. Lin, Q. Qu, L. Ning, J. Fan, and Q. Jiang proposed an MVCC approach to parallelizing the interoperability of consortium blockchains, in which cross-chain transactions are described through object version reconciliation [7]. G. Sheffi, P. Ramalhete, and E. Petrank developed EEMARQ—a lock-free structure supporting efficient range queries and memory management in a multi-version environment [8]. L. Yang, X. Dong, Z. Wan, D. Lu, Y. Zhang, and Y. Shen, in their work on HiCoCS, showed how MVCC conflicts in Hyperledger Fabric limit the throughput of cross-sharding transactions and proposed schemes for logical separation of state hot spots [9]. Finally, M. Zhang, Y. Hua, and Z. Yang described Motor, a system implementing multi-versioning for distributed transactions in architectures with disaggregated memory, which allowed for evaluating the impact of network latency on version metadata costs [10].

Comparative analysis and synthesis of scientific sources, methods of systemic and structural-functional analysis of distributed system architectures, as well as elements of conceptual and architectural modelling, were applied to write the article. These methods allowed for formulating generalizing state management patterns and recommendations for their practical application.

RESULTS

The results of the analytical research show that multi-version concurrency control (MVCC) in modern distributed systems is evolving towards an increasingly tight connection with storage, replication, and consensus architectures. A comparison of recent works on MVCC, hybrid blockchain-DBMSs, and high-performance cross-shard transaction mechanisms allows for the formulation of a holistic state management model suitable for high-load blockchain platforms and distributed AI systems processing millions of requests per day [1–10].

Common to the examined solutions is a shift in focus from the local "server–database" level to the global "cluster–blockchain–agent platform" level. MVCC ceases to be exclusively a transaction isolation mechanism and transforms into a universal state management layer, upon which consensus protocols, sharding, cross-shard transactions, and agent interactions are implemented. In this layer, the following become key: 1) version storage cost, 2) determinism and behavioral auditability, 3) reconciliation of version snapshots between replicas and shards, 4) the impact of version sampling on throughput and latency.

A basic limiting factor is the accumulation of old versions. The garbage collection method for multi-version structures proposed by Ben-David et al. demonstrates that it is possible to achieve simultaneous time and space bounds: the average time to reclaim one version remains O(1), and the number of supported "extra" versions does not exceed a constant factor of the minimum necessary [1]. The technique relies on two components—a wait-free structure for tracking ranges of obsolete versions and a new lock-free doubly linked list structure that allows for efficient removal of arbitrary nodes from version lists. Analytically, this sets a lower bound for MVCC overhead at the memory level and provides a basis for high-load systems where numerous long-lived reads (e.g., analytics over live blockchain state or global state scans by AI agents) should not lead to an explosive growth in the number of versions.

The verifiability of the MVCC mechanism becomes critical as protocols grow in complexity. The work of Chang et al. on vMVCC shows that a high-performance transactional library with MVCC can be formally verified, including correctness of visibility, absence of serializability anomalies, and invariants regarding the locking protocol and log structure [2]. For distributed systems, this means that the state management layer can act as a formally proven core upon which replication and consensus protocols are layered. This approach is particularly important for financial and infrastructural blockchain systems, where an error in MVCC implementation at a bottleneck of the execution layer can lead to irretrievable loss of funds or non-deterministic forks.

An important direction in MVCC evolution has been the transfer of responsibility for multi-versioning from local disk storage to new hardware and architectural models. Freitag et al. proposed memory-optimized MVCC for disk-based DBMSs: by applying multi-versioning in memory and

offloading long-lived versions and "cold" data to disk, the authors manage to approach the latencies of in-memory systems while retaining the cost-effectiveness of disk storage [3]. This shifts the optimization focus from the classic "I/O bottleneck" to a fine balance between the volume of versions in memory and the frequency of garbage collection, which, for distributed systems with a large number of replicas, transforms into the task of a coordinated GC policy between cluster nodes.

An even more radical step is taken by Motor (Zhang et al.), extending multi-versioning to the scenario of distributed transactions over disaggregated memory [10]. In such an architecture, compute nodes access "shared" memory pools via the network, and MVCC must ensure consistent snapshots and version chains under high access latency. The analytical breakdown of Motor shows that traditional assumptions about the cheapness of local memory cease to work: any additional version metadata (timestamps, pointers to predecessors) directly converts into additional network requests. This leads to the development of a principle for minimizing specific network load per version, which is transferred in the proposed architecture of this article to the level of designing the state storage scheme of a distributed system: versions must be grouped and aggregated so that a single network round-trip covers the maximum possible range of data and metadata.

The main idea of EEMARQ is to store multiple version nodes, serving range queries over a logically frozen snapshot, while memory is reclaimed via a lock-free reclamation scheme coordinated with the version lifecycle. For practice, this means that the MVCC subsystem can be moved out of the classic DBMS into a library of concurrent data structures underlying the execution layer of blockchain virtual machines or the internal storage of AI agents: contract and agent states can be stored in such structures, obtaining linearizable scans without global locks.

At the level of distributed systems, reconciling MVCC between primary and backup replicas proves to be an important task. Helt et al. in the C5 paper show that existing protocols for cloned concurrency control in primary–backup schemes, where secondary nodes are obliged to reproduce the commit order of the primary, inevitably limit the degree of parallelism and can lead to unlimited replication lag [5]. The proposed C5 protocol ensures limited lag by forcing the backup to execute writes with the same granularity as the main system and to use row-granularity serialization. Analytically, this emphasizes that any state management architectures in a distributed system using MVCC and asynchronous replication must be designed so that the primary replica's form of parallelism is reproducible by the backups. Otherwise, the system faces a "choked" replication effect, where secondary nodes constantly lag despite having greater total computational power.

Integrating MVCC with blockchain architectures requires accounting for additional factors—determinism, cryptographic auditability, and consensus costs. The review by Kalajdjieski et al. systematizes the family of databases applicable in a blockchain context, including pure blockchain DBMSs, "blockchain + classic DBMS" hybrids, and solutions with state logging [6]. This review emphasizes that for many scenarios, a hybrid approach is more promising, where the DBMS layer provides rich data and transaction models (including MVCC), and the blockchain is responsible for the immutable log and consensus. In other words, multi-versioning naturally "belongs" in the database layer, not the block level.

Ge et al. concretize this idea in Hybrid Blockchain Database Systems: they analyze the architecture of hybrid systems where blockchain and DBMS work jointly, showing that under typical loads, the bottleneck is not local query processing but the consensus protocol [4]. Furthermore, part of the solutions they utilize relies on multi-version storage, allowing the separation of read latency for a consistent snapshot from the transaction confirmation time in the chain. For the architecture we are designing, this leads to an important conclusion: for high-load blockchain execution layers, a "thick" MVCC layer around a relatively "thin" consensus core is more advantageous than the reverse. Consensus is responsible for the linear order of commits, but specific state snapshots for reading and even part of the preliminary transaction validation should be served within the MVCC subsystem, minimizing the number of calls to consensus.

A separate direction is the application of MVCC to ensure interoperability and parallel processing in multi-shard blockchain systems. Lin et al. propose an MVCC approach to parallelizing the interaction of consortium blockchains: the key idea is that data participating in cross-chain operations is wrapped in multi-version objects, and the protocol ensures the serializability of cross-chain transactions by reconciling versions rather than through coarse global locks [7]. Analytically, this manifests as replacing the distributed two-phase commit (2PC) with a "version reconciliation" protocol between chains: instead of locking resources on all chains, the system checks the compatibility of object versions participating in the operation and allows conflict only at the level of version metadata.

Modern permissioned blockchains, such as Hyperledger Fabric, already use MVCC as a basic mechanism for parallel transaction processing; however, under high competition of inter-shard transactions, it is MVCC conflicts that become the key limitation. The work of Yang et al. (HiCoCS) shows that with a large number of Cross-Shard Transactions (CSTx) passing through the same intermediary accounts, Fabric faces an avalanche-like growth in rollbacks due to concurrent reading and writing of the same key versions [9]. The text of the HiCoCS article emphasizes that Fabric implements data

consistency using MVCC: each key–value pair is associated with a version number, and any change generates a new version; if the expected version does not match the actual one, the transaction is rolled back. The authors demonstrate that during inter-shard transfer via intermediaries, this leads to constant conflicts over the state of intermediary accounts, since many CSTx read and update the same key simultaneously. Based on their analysis, a figure illustrating a typical MVCC conflict scenario and the cross-shard scheme in Hyperledger Fabric is presented (see Fig. 1).

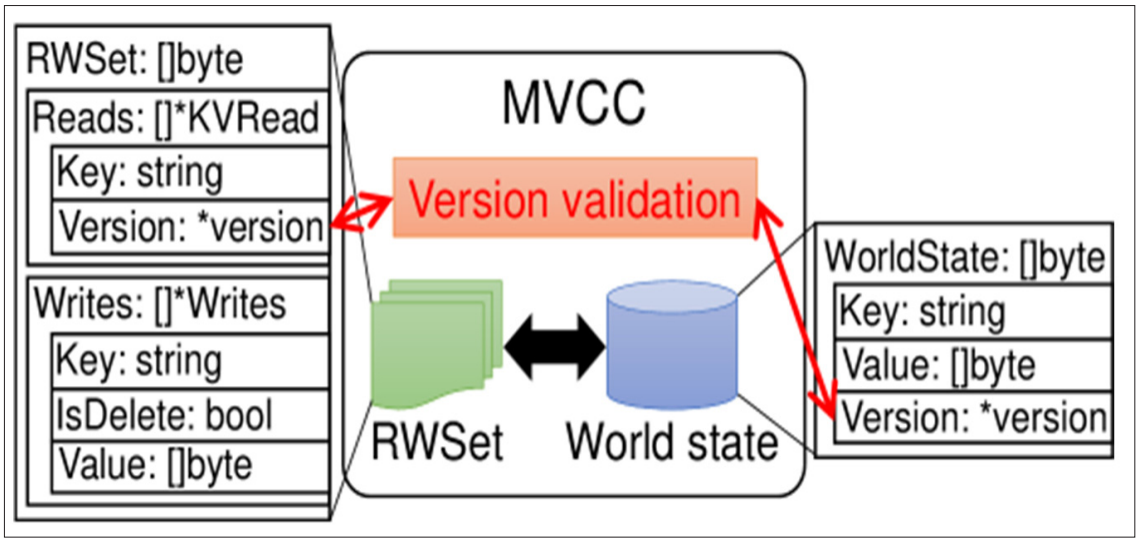

Figure 1. MVCC conflict and example of cross-shard interaction in Hyperledger Fabric [9].

Figure 1 schematically shows how a series of cross-shard transactions passing through the same intermediary account causes Fabric to constantly detect desynchronization between the expected and actual version number of the key I.balance. Under conditions of high contention, the majority of transactions are forced to rollback and re-execute, which radically reduces the system's effective throughput. HiCoCS proposes using composite keys and virtual sub-brokers to "split" the conflicting account into multiple logical keys and thereby separate update streams, which in experimental scenarios yields a 3.5–20.2 times increase in CSTx throughput and a significant reduction in latency [9].

From the perspective of distributed system state management, an important result can be derived from this: MVCC at the key level is insufficient for scalable cross-shard protocols; it is necessary to design the key space and versioning scheme in such a way that state hot spots (e.g., an intermediary's balance) are logically scattered across multiple keys or structured into batches allowing for batch updates and homomorphic aggregation, as HiCoCS does.

The consolidated analysis shows that for modern distributed systems with a high degree of parallelism, several robust patterns can be identified:

- "Multi-version snapshot over consensus" pattern: the consensus protocol provides a linear order of commits, while MVCC provides cheap reads of a logically consistent snapshot without involving the consensus layer, as in hybrid blockchain-DBMSs [4, 6];
- "Version lists with GC and lock-free access" pattern: version garbage collection is synchronized with lock-free data structures, allowing long reads and range queries to be served without locks [1, 3, 8];
- "Cloned concurrency control" pattern: backup replicas are forced to reproduce the version order of the primary replica, and protocols like C5 ensure limited lag while maintaining a high degree of parallelism [5];
- "Versions as a unit of cross-chain interoperability" pattern: cross-chain and cross-shard transactions operate on versioned objects rather than raw records, reducing the need for global locks and 2PC [7, 9, 10].

The state management layer in these systems is built around MVCC stores with change logging and cryptographic data structures. Read state snapshots are formed via multi-version lists over the commit log, while consensus and replication protocols are responsible only for the linear order of writes. This separation of load allows read latency to be kept at the level of in-memory operations, even with significant costs of the consensus protocol and replication between nodes and data centers.

The greatest effect is yielded by a combination of three techniques:

1. use of composite keys and logical partitioning of balances, gas counters, and state aggregates, which reduces the frequency of MVCC conflicts in hot spots;
2. offloading long-lived versions to storage tiers with lower cost while maintaining the "hot" part in memory;
3. strict synchronization of the versioning scheme between primary and backup nodes, which keeps replica lag within operational requirements under a large number of parallel transactions.

These solutions correlate directly with the previously identified patterns of "multi-version snapshot over consensus", "version lists with efficient GC and lock-free access", and "versions as a unit of cross-chain interoperability".

For systems of the class of high-performance L2 blockchains and agentic AI platforms processing hundreds of thousands to millions of operations daily, the analysis results mean that a reasonable state management architecture must include:

- a multi-version core oriented towards in-memory or disaggregated-memory scenarios, with strict guarantees regarding version garbage collector overhead and support for locally lock-free data structures [1, 3, 8];
- a formally verified transaction layer (similar to vMVCC), separated from the specific consensus protocol and capable of serving as a common basis for both blockchain nodes and AI agent backends [2];
- mechanisms for cloned control and version reconciliation between replicas and shards, ensuring limited lag and reproducibility of primary replica parallelism [5, 10];
- a versioned model of cross-shard and cross-chain interaction, where the unit of reconciliation is not individual records but versions of logical objects and aggregates, with the key space designed taking into account the potential for "disaggregating" hot spots via composite keys and virtual intermediaries [7, 9].

Thus, MVCC ceases to be a "local trick" for transaction optimization and becomes a central mechanism for state management in distributed systems. Modern approaches—from time-and-space-bounded multi-version GC to highly concurrent cross-shard schemes and hybrid blockchain-DBMSs—set clear design principles that can serve as a foundation for designing high-performance blockchain execution layers and scalable platforms of interacting AI agents.

DISCUSSION

Multi-version concurrency control in distributed systems should not be viewed as an isolated technique for organizing transactions, but as a central state management layer around which architectural decisions regarding replication, consensus, scaling, and inter-shard interaction are built. Comparing the results of the works analyzed in the previous section allows for the formulation of several key conclusions for the practice of designing high-load blockchain systems and agentic AI platforms using distributed state [1–10].

It becomes obvious that the classic view of MVCC as an expensive but convenient transaction isolation mechanism no longer corresponds to the realities of modern systems. Constructions proposed by Ben-David et al. demonstrate that version garbage collection can be organized in such a way as to guarantee simultaneous time and space boundedness without sacrificing wait-free/lock-free properties of data structures [1]. This means that the use of MVCC does not necessarily lead to uncontrollable growth in overhead costs if the version architecture and reclamation algorithms are designed taking into account the relevant theoretical constraints. For the practice of scalable distributed systems, this removes one of the traditional arguments against the widespread application of MVCC in high-load scenarios.

The importance of formal verification of the MVCC core is growing. The work on vMVCC demonstrates that the implementation of a multi-version transactional library can be strictly checked for the correctness of visibility and serializability guarantees [2]. For financial and infrastructural systems, including L2 blockchains and decentralized AI agent platforms, this provides an important advantage: the state management core ceases to be a "black box" and becomes an object of formal proofs. Combined with fault-tolerant consensus protocols and cryptographic auditability of logs, such a verifiable MVCC subsystem can be viewed as a trusted computing base from which the properties of the entire platform are inherited.

Another aspect is related to exactly how MVCC is embedded into the data storage and access architecture. Memory-optimized MVCC for disk systems shows that a reasonable separation of "hot" and "cold" versions between memory and disk allows achieving latencies close to in-memory systems while retaining the economy of disk storage [3]. In distributed systems with a large number of replicas, this leads to the necessity of coordinating version storage policy between nodes: if some replicas aggressively offload versions to disk while others do not, snapshot semantics and response times to identical requests can differ significantly. In hybrid blockchain-DBMSs described by Ge et al., similar decisions directly affect how effectively the database layer can hide consensus layer latencies [4].

The review of hybrid blockchain-DBs and the generalizing classification of databases for blockchain emphasize that the most promising are architectures in which a rich transactional model and MVCC are implemented at the DBMS level, while the blockchain acts as an immutable log and consensus layer [4, 6]. This opposes traditional attempts to embed multi-versioning directly into the block level or smart contracts. Hybrid solutions discussed in the literature show that such a separation of responsibility allows obtaining both high throughput of local read/write operations and a strict order of commits set by the blockchain protocol [4, 6]. For systems striving for tens of thousands of transactions per second per core and sub-second confirmation latency, such a "thick" MVCC layer around a thin consensus core appears to be the most rational choice.

Fundamental limitations manifest when considering replication and concurrency control in primary–backup configurations. The analysis of C5 shows that a naive cloned implementation of concurrency control, where backup replicas "blindly" reproduce the primary's commit sequence, is either forced to significantly limit parallelism or leads to

unlimited lag relative to the primary replica [5]. The proposed protocol, conversely, demonstrates that it is possible to maintain a limited lag while preserving a sufficiently high degree of parallelism if serialization granularity and version structure are consistent between the primary and backup systems [5]. For high-load blockchain execution layers, this means that state replication cannot be viewed as a secondary solution atop a local MVCC implementation: replication protocols and multi-version layer design must be engineered jointly.

Significant importance is also acquired by how MVCC is used in cross-chain and cross-shard interaction. The MVCC approach to parallelizing the interoperability of consortium blockchains described by Lin et al. showed that cross-chain operations can be described in terms of object version reconciliation rather than global locks and two-phase commits [7]. Instead of holding locks on all participating chains, the system can check the compatibility of versions of logical objects (balances, contract states, metadata) and reject only those transactions that violate version consistency. This brings cross-chain interaction closer to classic systems with multi-version snapshots, where conflicts manifest at the version level rather than at the level of coarse resource locks.

In the context of permissioned blockchains such as Hyperledger Fabric, the analysis of cross-shard transaction competition shows that MVCC can turn into both a powerful accelerator and a system bottleneck. The HiCoCS example demonstrates that when using intermediary accounts through which numerous cross-shard transactions pass, Fabric faces an avalanche of MVCC conflicts: many transactions read the same key versions and attempt to update them, leading to mass rollbacks [9]. Projects like HiCoCS propose designing the key space in such a way that state "hot spots" are logically separated (via composite keys and virtual sub-brokers), which significantly reduces conflict frequency and increases throughput [9].

For a visual comparison of design principles deriving from the reviewed works, it is expedient to highlight a set of criteria by which MVCC architectures are evaluated in the context of distributed systems and to correlate different classes of solutions with them. A generalized picture is presented in Table 1, which consolidates key emphases characteristic of theoretically oriented works on version garbage collection and lock-free structures [1, 8], systemic-practical optimizations of memory and storage [3, 10], as well as hybrid blockchain-DBs and cloned control protocols.

Table 1. Key Design Emphases of MVCC Architectures in Distributed Systems [1–6, 8, 10]

Class of Solutions	Main Focus	Typical Trade-offs	Practical Implications for Distributed Systems
Theoretical GC and Lock-free Structures	Guarantees on time/ memory, and progress	Complexity of implementation and debugging	Possibility of safe version scaling
Memory-optimized MVCC for Disk DBMS	Balance between memory/disk	Complex version placement policy	Different latency profiles on replicas
Disaggregated-memory MVCC	Minimization of network load	Increased role of metadata format	Necessity of co-design with network architecture
Hybrid Blockchain-DBs	Separation of DBMS and blockchain roles	Complexity of transaction and block reconciliation	Reduction of load on the consensus layer
Cloned Control (Primary–Backup)	Limited replica lag	Constraints on the form of parallelism	Requirement for joint design of MVCC and replication
Permissioned and Consortium Blockchains with MVCC	Parallelism and interoperability	Conflicts at hot state spots	Necessity of designing the key space

As seen in Table 1, different directions of MVCC development accentuate various aspects of the same problem—managing a large number of versions under conditions of distribution and high competition. Works on version garbage collection and lock-free structures [1, 8] maximize progress and theoretical guarantees but are complex to implement; memory-optimized and disaggregated-memory approaches [3, 10] concentrate on the latency profile and storage cost; hybrid blockchain-DBs and cloned control [4–6] emphasize the necessity of joint design of MVCC with replication and consensus. Collectively, this reinforces the thesis that a distributed system designer must think of MVCC not as a library, but as the core of the data architecture.

Moving from classification to target scenarios, three types of systems appear most important: high-performance public or semi-public blockchains (including L2 solutions), permissioned consortia with cross-shard interaction, and large-scale agentic AI platforms. In each of these scenarios, requirements for MVCC differ noticeably, although the basic principles remain unified.

In high-performance blockchain systems, the key factor is the ability of the execution layer to support massive parallelism without imposing excessive limitations on the consensus level. Hybrid blockchain-DBs and blockchain database classifications show that in such systems, it is expedient to transfer the implementation of complex isolation logic

and multi-versioning to the database layer [4, 6], leaving consensus the role of a linear commit log. Ideas of memory-bounded GC [1], memory-optimized storage [3], and cloned control protocols that guarantee backup nodes "keep up" with primary ones [5] are particularly valuable here.

For permissioned consortia, where the main bottleneck is cross-shard transaction conflicts, the design of the key space and versioning scheme proves critical. Works on MVCC interoperability parallelization [7] and scaling cross-sharding transactions [9] demonstrate that even with identical MVCC mechanisms (key versioning, optimistic version checking), changes in the logical data model—introduction of composite keys, virtual sub-brokers, splitting state into multiple logical segments—are capable of radically improving throughput and reducing the conflict rate.

Agentic AI platforms, although significantly different in domain, face similar problems: a multitude of independent agents simultaneously read and update shared knowledge stores, task queues, and interaction logs. For such systems, it is expedient to use ideas from lock-free structures and version GC [1, 8] to support long analytical queries and historical agent state scans without blocking short transactional updates. At the same time, a hybrid approach with a separation of responsibility between a transactional DB and an external event log (e.g., blockchain or log-oriented systems) can ensure both recoverability and auditability of interaction history [4, 6].

From the perspective of architectural solutions, it is useful to correlate the key requirements of these three scenarios with recommended MVCC usage patterns deriving from the reviewed works. A generalization of such alignment is presented in Table 2.

Table 2. Target Scenario Requirements and Corresponding MVCC Patterns (based on [1–10])

System Type	Key State Management Requirements	Recommended MVCC Patterns and Techniques
High-Performance Public/ L2 Blockchains	Maximum parallelism, determinism, verifiability	Hybrid Blockchain-DB with "thick" MVCC layer; formally verified transaction core; memory-constrained GC; consistent cloned control
Permissioned Consortiums with Cross-Sharding	High share of cross-shard transactions, absence of key bottlenecks	MVCC cross-chain protocols; key space design (composite keys, sub-proxies); minimization of hot spot conflicts
Agent AI Platforms	Massive agent parallelism, long analytical queries	Lock-free structures with MVCC; efficient version GC; separate storage of operational and historical data; possible integration with event log/blockchain
Disaggregated Memory Systems	High memory access latency, network scaling	Compact version metadata; aggregated traversal of version chains; joint design of data format and network architecture
Replicated Transactional Clusters	Limited replica lag, reproducibility of parallelism	Cloned control protocols; reconciliation of serialization granularity and version structure between primary and backups

Table 2 shows that the same basic MVCC ideas—multi-version snapshots, separation of reading from confirmed commit, version as a reconciliation element—manifest differently in distinct classes of systems. For blockchain systems, emphasis shifts to determinism and auditability: formal verification and integration with the consensus protocol are important [2, 4, 6]. For permissioned consortiums, data modeling and minimization of cross-shard transaction conflicts come to the fore [7, 9]. For agentic AI platforms, lock-free properties and the ability to serve long analytical queries without degrading short transactions are paramount [1, 3, 8]. In systems with disaggregated memory, the requirement to minimize the number of network requests per version unit is added [10].

MVCC not only fits into the specialization of an engineer working with large-scale distributed systems, blockchain platforms, and agentic AI architectures but effectively becomes the central mechanism ensuring their scalability, consistency, and auditability. The results of the analytical research form a conceptual framework upon which high-performance blockchain execution layers and scalable platforms of interacting AI agents can be designed and developed, and also set the agenda for future experimental and applied works in this field.

CONCLUSION

The conducted research confirmed that MVCC in modern distributed systems should be treated as a key state management layer, not an auxiliary transaction isolation mechanism. Comparative analysis of theoretical and systems-oriented approaches showed that multi-version garbage collection can be organized with simultaneous time and space constraints, integrated with lock-free data structures, and adapted to various storage profiles, including in-memory, disk, and disaggregated memory architectures. This allows scaling the number of versions without an exponential increase in overhead costs and provides a basis for safe parallelism.

The investigation of MVCC integration with protocols for consensus, replication, and inter-shard interaction demonstrated that the best results are achieved in architectures where complex multi-version and transaction logic is implemented at the level of a database or specialized transactional core, while the blockchain and other log subsystems are responsible for the linear order of commits and cryptographic auditability. Cloned control protocols, hybrid blockchain-DBMSs, and MVCC-oriented cross-sharding schemes show that the coordinated design of the multi-version layer, data model, and replication algorithms allows simultaneously limiting replica lag, reducing conflict frequency, and increasing the system's effective throughput.

The formulated architectural patterns of state management—"multi-version snapshot over consensus", "version lists with efficient GC and lock-free access", "cloned control with limited lag", and "versions as a unit of cross-chain interoperability"—set a practical framework for designing high-performance L2 blockchains, permissioned consortiums, and agentic AI platforms. It is shown that the correct choice of key space, versioning scheme, and metadata format is as important as the choice of a specific consensus algorithm. A limitation of the work is the analytical nature of the conclusions without independent experimental measurements on industrial infrastructure. Nevertheless, the comprehensive comparison of theoretical and applied studies allowed for forming a holistic model of using MVCC as a state management core in distributed systems and outlining directions for further work: experimental validation of the proposed patterns on real loads, deepened formalization of interfaces between verifiable MVCC cores and high-level languages of smart contracts and agents, as well as the adaptation of multi-version approaches to new hardware and network architectures.

REFERENCES

1. Ben-David, N., Blelloch, G. E., Fatourou, P., Ruppert, E., Sun, Y., & Wei, Y. (2021). Space and time bounded multiversion garbage collection. arXiv. https://arxiv.org/abs/2108.02775
2. Chang, Y.-S., Jung, R., Sharma, U., Tassarotti, J., Kaashoek, M. F., & Zeldovich, N. (2023, July 10–12). Verifying vMVCC, a high-performance transaction library using multi-version concurrency control. Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23). USENIX Association. https://www.usenix.org/conference/osdi23/presentation/chan
3. Freitag, M., Kemper, A., & Neumann, T. (2022). Memory-optimized multi-version concurrency control for disk-based database systems. Proceedings of the VLDB Endowment, 15(11), 2797-2810. https://doi.org/10.14778/3551793.3551832
4. Ge, Z., Loghin, D., Ooi, B. C., Ruan, P., & Wang, T. (2022). Hybrid blockchain database systems: Design and performance. Proceedings of the VLDB Endowment, 15(5), 1092-1104. https://doi.org/10.14778/3510397.3510406
5. Helt, J., Sharma, A., Abadi, D. J., Lloyd, W., & Faleiro, J. M. (2022). C5: Cloned concurrency control that always keeps up. Proceedings of the VLDB Endowment, 16(1). 1-14. https://doi.org/10.14778/3561261.3561262
6. Kalajdjieski, J., Raikwar, M., Arsov, N., Velinov, G., & Gligoroski, D. (2023). Databases fit for blockchain technology: A complete overview. Blockchain: Research and Applications, 4(1), 100116. https://doi.org/10.1016/j.bcra.2022.100116
7. Lin, W., Qu, Q., Ning, L., Fan, J., & Jiang, Q. (2021). A MVCC approach to parallelizing interoperability of consortium blockchain. In K.-M. Chao, M. A. Jabbar, & H. Zhang (Eds.), Parallel and distributed computing, applications and technologies: 22nd International Conference, PDCAT 2021, Guangzhou, China, December 17–19, 2021, Proceedings, 273–285. Springer. https://doi.org/10.1007/978-3-030-96772-7_25
8. Sheffi, G., Ramalhete, P., & Petrank, E. (2022). EEMARQ: Efficient lock-free range queries with memory reclamation. arXiv preprint arXiv:2210.17086. https://doi.org/10.48550/arXiv.2210.17086
9. Yang, L., Dong, X., Wan, Z., Lu, D., Zhang, Y., & Shen, Y. (2025). HiCoCS: High concurrency cross-sharding on permissioned blockchains. arXiv preprint arXiv:2501.04265. https://doi.org/10.48550/arXiv.2501.04265
10. Zhang, M., Hua, Y., & Yang, Z. (2024, July 10–12). Motor: Enabling multi-versioning for distributed transactions on disaggregated memory. Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24). USENIX Association. https://www.usenix.org/conference/osdi24/presentation/zhang-ming