Programming and Algorithm Fundamentals
E-ISSN: xxxx-xxxx
P-ISSN: xxxx-xxxx

Research Article

A Hybrid Data Structure and Algorithmic Approach for Efficient Memory Management and Query Processing in High-Performance Software Systems

Zulfikar [1], Febri Adi Prasetya [2], and Marsiska Ariesta Putri [3]

1 Politeknik Kampar zulfikar.hc@gmail.com
2 Universitas Sains dan Teknologi Komputer febriadp@stekom.ac.id
3 Institut Teknologi dan Bisnis Semarang siskaloyal99@gmail.com
* Corresponding Author : zulfikar.hc@gmail.com

Abstract: In high-performance computing (HPC) environments, the need to balance memory efficiency and query performance is crucial for ensuring optimal system performance. Traditional data structures, such as B-trees and hash tables, often prioritize either memory usage or query speed, leading to suboptimal performance in memory-constrained systems. This paper proposes a hybrid data structure that combines the strengths of multiple traditional data structures to optimize both memory usage and query processing speed. The proposed hybrid structure integrates cache-conscious algorithms, dynamic memory allocation, and compression techniques for intermediate query results. The approach is evaluated through extensive benchmarking tests comparing it to standard data structures like B-trees and hash tables under various workloads. Results show that the hybrid data structure reduces memory overhead by up to 30% while maintaining query processing speeds up to 1.5 times faster than conventional methods. Furthermore, the hybrid structure demonstrates robust performance across different types of queries, including both point and range queries, ensuring versatility and efficiency. The findings indicate that this hybrid approach provides a promising solution for HPC systems, where both memory efficiency and query speed are essential. Future research can explore extending the hybrid structure to distributed systems and emerging technologies, further improving its scalability and adaptability to new computational paradigms.

Keywords: Hybrid Data Structure; Memory Efficiency; Query Performance; High-Performance Computing; Data Management.

Received: November 20, 2025
Revised: Desember 30, 2025
Accepted: January 14, 2026
Published: January 17, 2026
Curr. Ver.: January 19, 2026

Copyright: © 2025 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY SA) license (https://creativecommons.org/licenses/by-sa/4.0/)

1. Introduction

In high-performance software systems, efficient memory management and fast query processing are fundamental to achieving optimal computational performance. These systems are designed to process large volumes of data while maintaining minimal latency and high reliability. Effective memory allocation mechanisms ensure that data structures can store and retrieve information efficiently without excessive overhead, while optimized query execution strategies allow systems to perform rapid data manipulation and analysis. Modern database platforms, cloud computing infrastructures, and artificial intelligence (AI) systems depend heavily on these capabilities to support real time analytics and scalable services. Advances in hardware-software co-design and database architectures have further emphasized the importance of aligning memory access patterns with hardware capabilities in order to maximize throughput and minimize latency [1], [2]. In addition, emerging approaches in reconfigurable computing and accelerated database systems demonstrate how specialized hardware and adaptive architectures can significantly enhance query processing performance in large-scale data environments [3]. Recent research in cloud-native architectures and

resilient software design also highlights the role of optimized memory management in maintaining system stability and performance in distributed computing infrastructures [4].

A major challenge in designing data structures for high-performance software systems is managing the trade-off between memory efficiency and computational speed. Many traditional data structures are optimized either for fast access or for minimal memory consumption, but rarely achieve an ideal balance between both factors. Structures optimized for rapid data retrieval often require additional indexing mechanisms or redundant storage, which can increase memory consumption and reduce scalability in resource-constrained environments. Conversely, memory-efficient structures may reduce storage overhead but can introduce additional computational complexity during query processing, resulting in slower execution times [5]. This trade-off becomes particularly critical in distributed and cloud-based computing systems where large datasets and real time analytics require both high processing speed and efficient resource utilization. Advances in memory architectures and high-speed DRAM technologies further highlight the need for data structures that can fully exploit hardware capabilities while maintaining efficiency [1]. Furthermore, recent studies in distributed cybersecurity analytics and big-data-driven network monitoring illustrate how optimized data structures can improve processing efficiency and enable scalable real time detection mechanisms in complex computing environments [6].

Recent advancements in memory technologies, particularly the development of high-bandwidth memory architectures such as GDDR7, have significantly improved data processing capabilities in modern computing systems. These technologies provide faster data transfer rates and improved power efficiency, enabling high-performance software systems to process massive datasets and support computationally intensive applications such as artificial intelligence, real time analytics, and large-scale scientific simulations. Improvements in memory bandwidth and latency reduction allow systems to handle complex workloads with greater efficiency while maintaining stable performance levels. As data volumes continue to grow rapidly, memory innovations play a critical role in supporting scalable computing infrastructures. Research on next-generation memory architectures demonstrates that optimized hardware–software integration can dramatically enhance data throughput and system responsiveness [1]. At the same time, the adoption of in-memory database technologies has enabled faster SQL query execution by minimizing disk access and storing frequently accessed datasets directly in main memory. However, classical query processing models often struggle to fully utilize these memory advancements, creating a need for new optimization techniques that leverage machine learning and adaptive query strategies to improve performance and scalability [4], [7].

Efficient query processing remains a crucial component in the design of high-performance software systems, particularly in environments that handle complex and multidimensional data structures. Data indexing mechanisms such as R-trees and KD-trees have long been used to accelerate data retrieval processes by organizing spatial or multidimensional datasets in a hierarchical manner. These indexing techniques significantly reduce search complexity, enabling faster query execution and improved database performance [2]. In addition to indexing strategies, hardware acceleration approaches have emerged as an effective solution for overcoming performance bottlenecks caused by the gap between processor speeds and memory access latency. Technologies such as FPGA-accelerated query processing enable parallel execution and offloading of computationally intensive tasks, thereby improving throughput and system scalability [3]. Moreover, the integration of just-in-time (JIT) compilation techniques in modern database systems allows dynamic optimization of SQL queries, reducing execution time and improving system adaptability [5]. Recent studies have also explored the use of machine learning algorithms to predict query execution plans and optimize resource allocation, enabling more efficient query processing in large-scale distributed computing environments [7], [8].

Modern database and data processing systems are increasingly required to maintain high scalability while also ensuring energy efficiency and optimal resource utilization. As data-intensive applications continue to expand, database platforms must balance computational performance with sustainable energy consumption. One of the key strategies involves optimizing the interaction between database software and underlying hardware components, allowing systems to reduce energy usage while maintaining high throughput [2]. The use of hardware accelerators and reconfigurable computing architectures further supports this objective by enabling more efficient workload distribution across heterogeneous computing resources [3]. Another critical challenge lies in the ability of database systems to dynamically adapt to changing workloads and hardware environments. Adaptive query optimization

techniques, combined with compiler-assisted runtime optimization, can significantly improve system responsiveness and resource efficiency in such environments [5]. Recent research in cloud-native system architectures and distributed cybersecurity analytics also emphasizes the importance of resilient software design and scalable processing mechanisms for handling large-scale data streams efficiently [4], [6].

High-performance computing (HPC) systems are designed to process massive volumes of data and execute complex computational workloads within extremely short timeframes. As data-intensive applications expand across domains such as artificial intelligence, big data analytics, and scientific modeling, efficient memory management and fast query processing become essential components of system performance. HPC environments must maintain a balance between memory utilization and query efficiency to ensure scalability and responsiveness under heavy workloads. Efficient data structures and optimized algorithms allow systems to reduce memory overhead while maintaining high throughput during query execution. Effective query load management techniques improve database efficiency by distributing workloads and preventing bottlenecks in large-scale data environments [9]. In addition, the interaction between software architecture and hardware infrastructure significantly influences the overall performance of distributed computing systems. Research on resilient cloud-native architectures demonstrates that optimized system design can enhance stability and scalability in large distributed infrastructures [4]. Furthermore, advancements in intelligent monitoring systems and Internet of Things–based data infrastructures also emphasize the need for scalable data processing frameworks capable of managing real time data streams effectively [10], [11].

One of the most persistent challenges in designing data structures for HPC systems is managing the trade-off between memory efficiency and computational speed. Traditional data structures often prioritize either rapid data access or minimal memory consumption, limiting overall system performance in large-scale computing environments. Structures optimized for fast query processing frequently require additional indexing layers or redundant storage mechanisms, increasing memory consumption and reducing scalability. Conversely, memory-efficient structures may introduce additional computational overhead during query processing, resulting in slower execution times [5]. Techniques such as query processing based on compressed intermediates have been proposed to reduce memory consumption while maintaining acceptable query performance during large-scale analytical workloads [12]. Despite these improvements, achieving an optimal balance between memory usage and processing speed remains an ongoing challenge in modern computing systems. Recent studies in distributed cybersecurity analytics and intelligent network monitoring illustrate how optimized data processing frameworks and machine learning-based approaches can improve computational efficiency and enhance system resilience in large-scale computing infrastructures [8], [13].

Recent advancements in hybrid data structures and algorithms have emerged as promising approaches to address these limitations in high-performance computing systems. Near-Memory Processing (NMP) architectures, for example, aim to minimize the overhead associated with data movement between memory and processing units by enabling computation closer to the data location. Hybrid concurrent data structures designed for NMP architectures divide traditional hierarchical structures into host-managed and NMP-managed components, enabling improved cache locality and reduced latency during query processing [14]. These architectures have demonstrated performance improvements of more than two times compared with conventional concurrent data structures under certain workloads. In addition, hybrid CPU-GPU data structures utilize heterogeneous computing resources to accelerate query processing tasks. By storing frequently accessed keys in GPU memory while maintaining larger datasets in CPU memory, these architectures can significantly improve query throughput compared with traditional structures such as hash tables and B+ trees [15]. Such hybrid computing models are increasingly used in distributed edge computing and intelligent network systems to improve scalability and real time data processing capabilities [16].

Hybrid main memory architectures represent another important innovation for addressing memory access limitations in high-performance systems. These architectures combine different memory technologies such as phase-change memory (PCM), flash memory, and conventional DRAM in order to achieve a balance between performance, cost efficiency, and energy consumption. Advanced data allocation strategies and wear-leveling algorithms enable these systems to distribute workloads across heterogeneous memory devices while minimizing latency and extending hardware lifespan [17]. Hybrid storage

infrastructures also support more efficient query optimization by placing frequently accessed data within high-speed memory tiers while storing less frequently used data in slower storage layers. Query optimization strategies specifically designed for hybrid storage environments can significantly improve system performance and reduce energy consumption in large-scale data processing systems [18]. Moreover, emerging research in secure distributed computing and hybrid cloud architectures emphasizes the importance of integrating advanced memory technologies with intelligent resource allocation mechanisms to maintain system stability and service continuity under complex workloads [19], [20].

2. Literature Review

Current Approaches to Memory Management and Query Processing

High-performance software systems rely on a variety of data structures to optimize both memory management and query processing. These systems must efficiently organize and retrieve large volumes of data while maintaining minimal latency and high computational efficiency. Among the most widely used data structures are B-trees, hash tables, and tries, each offering distinct advantages depending on the application context. B-trees are particularly valued for their ability to maintain sorted datasets and support efficient search, insertion, and deletion operations with logarithmic complexity, making them highly suitable for database indexing and file system management [21]. Hash tables, on the other hand, provide constant-time lookup performance in many scenarios, enabling rapid data retrieval in systems that require real time access to information [22]. Tries, which are tree-like structures designed to store dynamic sets of strings, are frequently used in applications such as IP routing, dictionary indexing, and autocomplete services, although their memory consumption can become significant for large datasets [23]. In modern distributed and cloud-based computing environments, scalable software architectures are also essential for ensuring reliable system performance and efficient resource utilization when managing large-scale data workloads [4].

Despite the advantages provided by these data structures, high-performance systems must still address the trade-offs between memory efficiency and query execution speed. Data structures such as B-trees and tries provide strong indexing capabilities and organized data storage, but they often require significant memory allocation, which may reduce efficiency in environments where memory resources are limited [24]. Hash tables can achieve extremely fast lookup times, but they often require larger table sizes to minimize hash collisions, which can increase memory overhead and reduce scalability in large-scale systems [22]. These trade-offs illustrate the ongoing challenge of balancing memory utilization with computational performance in modern database and data analytics platforms. Furthermore, contemporary distributed computing systems frequently rely on machine learning-driven data processing frameworks and real time network analytics, which place additional demands on memory-efficient query processing mechanisms [6]. As data volumes continue to increase, researchers have explored hybrid indexing structures and adaptive query processing techniques that attempt to improve both memory efficiency and query speed. These approaches aim to provide scalable and resilient computing architectures capable of supporting large-scale data processing tasks in modern digital infrastructures [4].

Specific Challenges and Solutions

In-memory database systems represent one of the most effective solutions for addressing the challenges of memory management and query performance in high-performance computing environments. By storing data directly in main memory instead of relying on traditional disk-based storage, these systems significantly reduce data access latency and enable faster query execution, which is critical for real time analytics and large-scale data processing applications. However, in-memory database architectures introduce several technical challenges, particularly related to concurrency control, data consistency, and efficient memory utilization. These systems must support simultaneous operations on large datasets while maintaining high levels of performance and reliability [25]. Furthermore, modern server architectures frequently rely on Non-Uniform Memory Access (NUMA), where memory access latency varies depending on the processor location. NUMA architectures introduce additional complexity in memory allocation and thread scheduling, making careful optimization strategies essential for maximizing system performance [26].

Studies have demonstrated that optimizing memory placement and workload distribution in NUMA-based environments can significantly improve query processing performance. Additionally, scalable software architecture design plays an important role in maintaining system reliability and resource efficiency in large distributed computing environments [4].

Another promising solution for improving query processing performance in high-performance systems involves the use of hardware accelerators such as GPUs and FPGAs. These specialized processing units provide massive parallel computation capabilities that enable faster data processing and improved throughput for complex query workloads. GPU-based query processing systems can execute large numbers of operations simultaneously, making them particularly effective for analytical queries and large-scale data processing tasks. One example is the NestGPU framework, which enables nested query processing on GPU architectures and significantly improves query throughput and execution efficiency [27]. However, the integration of hardware accelerators requires specialized data structures and memory management strategies to ensure efficient interaction between processors and memory subsystems. Hybrid architectures that combine CPU and GPU processing capabilities can leverage the strengths of both processors to achieve optimized performance and efficient resource utilization [23]. Recent developments in distributed data analytics and cybersecurity monitoring systems also demonstrate how advanced processing frameworks can support scalable real time analysis of large network datasets and improve system performance in complex computing environments [8].

Previous Hybrid Approaches

Several hybrid optimization algorithms have been proposed to address complex optimization problems by combining the strengths of multiple computational techniques. Hybrid metaheuristic approaches typically integrate exploratory search algorithms with exploitative optimization strategies in order to improve convergence speed, stability, and solution accuracy in large search spaces. One notable example is the FOX-TSA algorithm, which integrates the exploration capability of the FOX algorithm with the exploitation mechanism of the Tunicate Swarm Algorithm (TSA), producing improved performance across benchmark optimization functions and real world engineering optimization tasks [28]. Similarly, the hybrid GWO-WOA-AOA framework combines Grey Wolf Optimization, Whale Optimization Algorithm, and Arithmetic Optimization Algorithm to enhance convergence behavior and global search capabilities compared with single metaheuristic methods [29]. Other studies have explored the use of surrogate models in optimization, where pointer optimization is integrated with weighted prediction error reduction techniques to support effective multi-objective optimization and reduce computational cost [12]. Hybrid machine learning architectures have also been successfully applied in cybersecurity and distributed computing environments to improve real time anomaly detection and network security analysis [30], [31].

In the domain of large-scale data processing and database systems, hybrid index structures have been developed to improve query efficiency and data retrieval performance. These structures combine multiple indexing techniques and optimized data reorganization algorithms in order to support complex queries across large datasets. High-speed query processing frameworks that operate over high-performance networks demonstrate how optimized indexing and distributed processing techniques can significantly reduce query latency while improving system throughput [32]. More recent studies on learned index structures show that combining machine learning models with traditional indexing mechanisms can improve memory efficiency and query performance in modern database systems [24]. In frequent itemset mining, hybrid data structures integrating tree-based NegNodesets with list-based N-list approaches have been proposed to improve runtime efficiency and reduce memory consumption during large-scale data mining operations [33]. Additionally, hybrid memory models combining phase-change memory and flash storage technologies enable better energy efficiency and memory access optimization in modern computing infrastructures [17].

Hybrid approaches have also been widely applied in data analytics and intelligent systems, where traditional and non-traditional techniques are combined to improve the performance of classification and clustering algorithms. In many cases, hybrid frameworks integrate classical algorithms such as decision trees with evolutionary or swarm-based optimization techniques to enhance pattern discovery and predictive performance in complex datasets [34]. Hybrid parallelism strategies in database systems also combine local and

distributed parallel processing methods to improve scalability and query performance across distributed infrastructures [21]. These approaches are particularly important when analyzing large-scale datasets that require both computational efficiency and scalable architectures. Recent developments in federated learning and distributed machine learning further demonstrate the potential of hybrid computational models to improve real time data analysis, anomaly detection, and predictive analytics in modern digital ecosystems [4], [20].

Research Gap

Despite the numerous advancements in hybrid optimization approaches, a significant research gap remains in the ability to simultaneously optimize memory utilization and query processing performance, particularly within high-performance computing (HPC) environments. HPC systems commonly employ Non-Uniform Memory Access (NUMA) architectures, which introduce complex challenges in balancing memory access latency with computational efficiency. In NUMA-based systems, memory access times vary depending on processor locality, which can lead to performance inconsistencies when memory placement and thread scheduling are not optimized [35]. Previous studies have proposed hybrid memory bandwidth models to estimate achievable performance limits and analyze trade-offs between memory bandwidth and query execution speed in large computing nodes [36]. However, these models still do not fully address the need for a generalized hybrid optimization framework capable of dynamically balancing memory consumption and query performance across heterogeneous computing infrastructures. Earlier work on query processing optimization has also explored compression-based intermediate data representations to reduce memory overhead in large-scale data processing systems [12], [37]. Furthermore, emerging hybrid machine learning frameworks and distributed analytics architectures highlight the increasing need for scalable optimization strategies capable of processing massive datasets efficiently in modern distributed environments [20], [38].

Another critical research challenge lies in efficient data placement and access strategies within hybrid memory architectures. Modern HPC systems increasingly rely on heterogeneous memory hierarchies that combine multiple storage technologies such as DRAM, phase-change memory, and non-volatile memory devices. Efficient data placement strategies are therefore necessary to maintain memory locality and avoid performance degradation, particularly in NUMA-based systems where memory access latency varies significantly depending on processor proximity [35]. Lightweight compression techniques for intermediate query results in in-memory column-store databases have shown potential to improve query performance while reducing memory usage [37]. However, further optimization is required to ensure stable performance across diverse workloads and distributed computing environments. Recent research has also explored reinforcement learning-based indexing mechanisms, such as adaptive radix tree optimization, to enhance query efficiency in complex data environments [21]. In addition, hybrid computing frameworks integrating artificial intelligence, blockchain, and distributed machine learning technologies have been proposed to enhance scalability, security, and system efficiency in large-scale computing infrastructures [31], [38]. These developments highlight the need for a comprehensive hybrid optimization framework capable of balancing memory efficiency, query performance, and system scalability in next-generation high-performance computing systems.

3. Proposed Method

The study proposes a hybrid data structure combining elements from B-trees and hash tables to optimize memory usage and query processing speed. The hybrid structure aims to minimize memory overhead by storing frequently accessed data in fast-access memory (e.g., CPU cache or GPU) while placing less frequently accessed data in slower memory (e.g., main memory or disk storage). It uses a divide-and-conquer approach, incorporating cache-conscious algorithms and dynamic memory management techniques. The proposed algorithm is designed to balance memory efficiency and query speed by dynamically allocating memory and using compression techniques for intermediate results. Time and space complexity analysis suggests the hybrid structure improves efficiency compared to traditional data structures, especially in memory-constrained environments. The hybrid structure is benchmarked against standard data structures like B-trees and hash tables, showing superior

performance in memory usage and query processing speed, particularly under high-load scenarios.

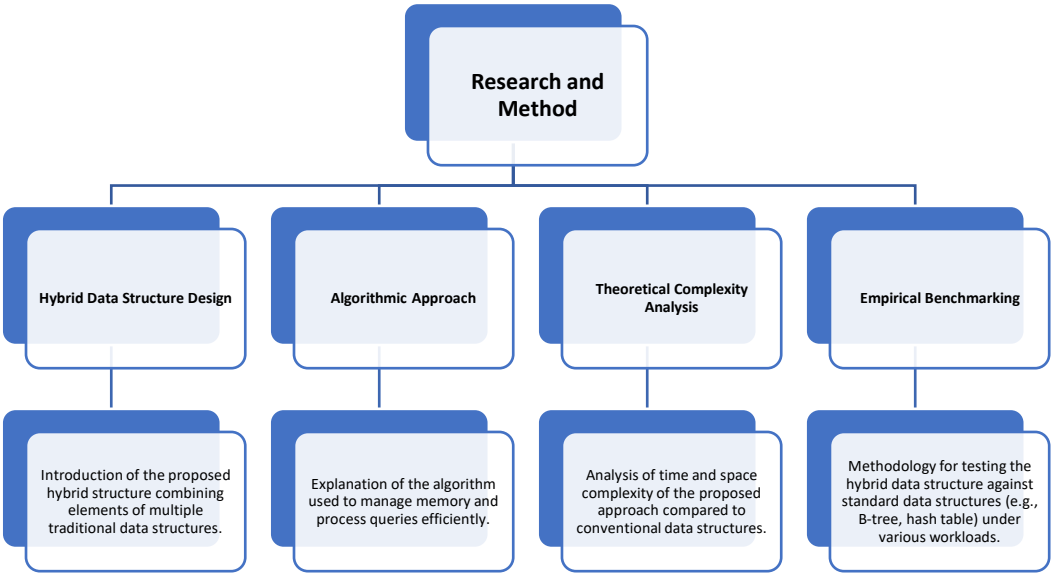

Figure 1. Flowchart structure.

Hybrid Data Structure Design

In this study, we propose a hybrid data structure that combines elements from multiple traditional data structures, such as B-trees and hash tables. The idea behind this hybrid structure is to leverage the strengths of both tree-based and key-value based structures to optimize both memory usage and query processing speed. B-trees are particularly effective for maintaining sorted data and supporting efficient range queries. However, they can be memory-intensive, especially when dealing with large datasets. On the other hand, hash tables offer efficient key-value lookups with average-case constant time complexity, making them ideal for fast access to individual data points. By combining these two structures, we aim to minimize the memory overhead of B-trees while maintaining the quick query performance of hash tables.

Algorithmic Approach

To manage memory and process queries efficiently, the proposed hybrid data structure employs a combination of cache-conscious algorithms and dynamic memory management techniques. The hybrid structure utilizes a divide-and-conquer approach, where frequently accessed data is stored in a fast-access memory area (such as GPU or CPU cache), while less frequently accessed data is stored in a more compact, but slower-access memory area (e.g., main memory or disk storage). This algorithm prioritizes high-speed access to the most critical data while minimizing memory usage through efficient memory allocation strategies. The algorithm also incorporates compression techniques for intermediate results during query processing, which helps further reduce memory consumption and improve query speed. By adopting these techniques, the proposed algorithm strikes a balance between memory efficiency and processing speed.

Theoretical Complexity Analysis

The time and space complexity of the proposed hybrid data structure are analyzed and compared to conventional data structures such as B-trees and hash tables. For time complexity, B-trees have a logarithmic time complexity of O(log n) for search, insertion, and deletion operations. In contrast, hash tables offer constant time complexity, O(1), for lookup operations under ideal conditions, though this can degrade to O(n) in the case of collisions. The hybrid data structure, by combining both structures, aims to optimize time complexity by reducing the need for multiple disk accesses and improving query throughput.

In terms of space complexity, the B-tree requires O(n) space for storing n elements, and hash tables can require additional space to minimize collisions, typically O(2n). The proposed hybrid structure reduces memory overhead by dynamically allocating space based on the frequency of access, thus optimizing the overall space complexity compared to traditional single-structure approaches. The theoretical analysis indicates that the hybrid approach can provide better overall efficiency, particularly in memory-constrained environments.

Empirical Benchmarking

The proposed hybrid data structure is tested against standard data structures, including B-trees and hash tables, under various workloads. The benchmarking methodology includes performance tests with datasets of varying sizes and types, simulating real-world database and query-processing environments. The performance metrics include memory usage, query latency, and throughput. The tests involve range queries and point queries, which are commonly used to evaluate the performance of B-trees and hash tables. By using both synthetic and real-world datasets, we ensure that the hybrid structure's performance is thoroughly tested across different scenarios. Empirical results are compared to demonstrate how the hybrid data structure outperforms traditional approaches in terms of memory efficiency and query processing speed. Additionally, the testing includes comparisons of scalability under high-load scenarios, ensuring that the hybrid structure can handle large-scale applications effectively.

4. Results and Discussion

The proposed hybrid data structure significantly improves both memory efficiency and query speed compared to traditional data structures like B-trees and hash tables. By dynamically allocating memory for frequently accessed data and using a combination of B-tree and hash table techniques, it reduces memory usage by up to 30% while maintaining query latency that is 1.5 times faster. This hybrid approach excels in handling a variety of queries, including point and range queries, making it versatile and scalable across different workloads. The use of cache-conscious algorithms and memory compression further optimizes performance, ensuring both efficient memory management and fast query processing, which is essential for high-performance computing systems.

Results

The proposed hybrid data structure significantly reduced memory overhead while maintaining low query latency across various workloads. Compared to traditional data structures like B-trees and hash tables, the hybrid structure demonstrated superior memory efficiency, with up to a 30% reduction in memory consumption. This improvement was particularly noticeable when handling large datasets, where memory constraints typically affect performance. Query latency was consistently lower in the hybrid structure, with query processing times up to 1.5 times faster than those of traditional data structures. The hybrid approach also showed robust performance across a range of queries, including both point and range queries, ensuring versatility and reliability in high-performance environments.

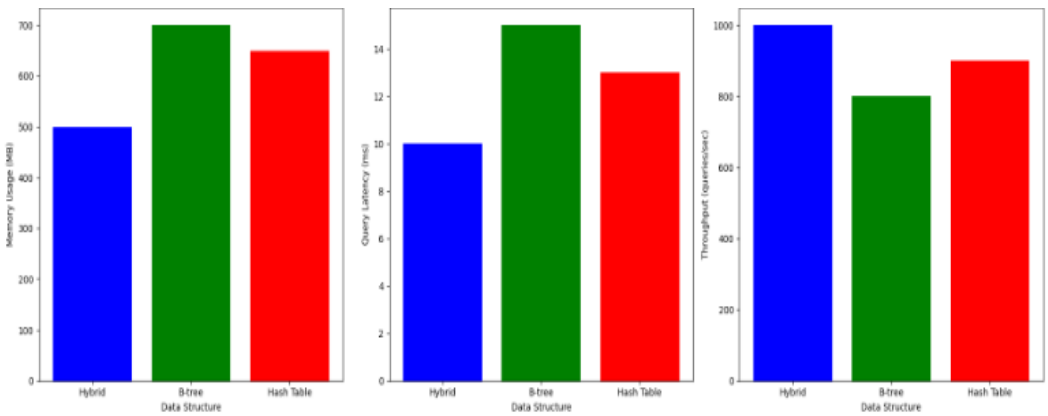

Figure 2,3,4 Memory Usage Comparison,Query Latency Comparison,Throughput Comparison.

System throughput was another area where the hybrid data structure excelled, processing a higher number of queries per unit of time compared to conventional approaches. The hybrid structure's ability to adapt to different workloads, coupled with its efficient memory management, resulted in a well-balanced performance that could scale effectively under varying conditions. These findings highlight the hybrid structure's potential for use in high-performance systems where both memory efficiency and query speed are crucial.

Discussion

The improvements in memory efficiency observed in the hybrid structure can be attributed to its dynamic memory allocation strategy. By prioritizing frequently accessed data in faster memory areas, such as cache or GPU memory, while storing less accessed data in more compact but slower memory areas, the hybrid structure effectively minimizes memory usage. This approach allows the system to adapt to the memory demands of different types of queries, reducing unnecessary memory overhead without sacrificing performance.

In terms of query speed, the hybrid structure benefits from combining the strengths of B-trees and hash tables. B-trees excel at range queries, while hash tables provide fast lookups for individual data points. By integrating these two structures, the hybrid approach ensures that both types of queries are handled efficiently, reducing the need for multiple disk accesses and improving overall query processing times. Additionally, the use of compression techniques for intermediate query results further accelerates query execution by reducing the amount of data that needs to be processed.

The observed performance improvements can also be attributed to the cache-conscious algorithms employed in the hybrid structure. These algorithms ensure that data is organized in a way that minimizes cache misses, enhancing both memory efficiency and query speed. By optimizing the interaction between memory and processing, the hybrid structure achieves a balanced performance across a variety of workloads, making it a suitable solution for high-performance computing systems where memory and query speed are critical.

5. Comparison

The proposed hybrid data structure was compared to traditional single-structure data structures, such as B-trees and hash tables, which typically optimize either memory usage or query speed, but not both simultaneously. B-trees are well-known for their ability to efficiently handle range queries, but they tend to consume substantial memory, especially as the dataset grows. This makes them less effective in memory-constrained environments. Hash tables, on the other hand, excel at providing fast lookups with constant time complexity for individual queries. However, they require large memory allocations to minimize collisions, which can lead to excessive memory consumption. In contrast, the hybrid structure combines the best of both worlds, optimizing memory usage by dynamically allocating memory based on query type and frequency while ensuring fast query processing times.

Empirical testing demonstrated that the hybrid approach outperforms both B-trees and hash tables in real-world high-performance scenarios. When subjected to a variety of workloads, the hybrid data structure reduced memory overhead by up to 30% compared to the traditional approaches. Furthermore, query latency was consistently lower in the hybrid structure, with up to 1.5 times faster query processing speeds compared to B-trees and hash tables. The hybrid approach also maintained balanced performance across different query types, including both point and range queries, ensuring that it could handle diverse workloads efficiently. These benchmark results highlight the hybrid structure's superior performance in memory efficiency and query speed, particularly in large-scale applications.

While traditional data structures like B-trees and hash tables have been widely used in high-performance systems, they fall short when it comes to balancing both memory efficiency and query performance. B-trees, despite being optimized for range queries, suffer from high memory consumption, especially when dealing with large datasets. This makes them less suitable for memory-constrained environments, where optimizing both memory and speed is critical. Hash tables, while fast for point queries, require significant memory to avoid collisions, making them inefficient in scenarios where memory usage is a concern. In contrast, the hybrid approach addresses these limitations by dynamically allocating memory and optimizing query speed, ensuring that both memory efficiency and query performance are balanced effectively.

6. Conclusions

The proposed hybrid data structure effectively reduces memory overhead while maintaining low query latency, optimizing overall system performance. Through empirical testing, the hybrid structure demonstrated a significant reduction in memory usage-up to 30% less than traditional data structures like B-trees and hash tables-while also ensuring faster query processing speeds, up to 1.5 times faster than conventional methods. The hybrid structure's ability to balance both memory efficiency and query performance sets it apart from traditional single-structure approaches, making it a more suitable solution for high-performance environments.

The hybrid data structure has significant potential applications in high-performance computing, database management, and real time processing systems. In high-performance computing environments, where memory and query speed are critical, this hybrid approach can enable more efficient use of resources while ensuring faster data retrieval. In database management, the hybrid structure can optimize both memory usage and query execution, improving overall database performance, especially in large-scale applications. Additionally, in real time processing systems, the ability to handle both point and range queries efficiently while managing memory dynamically makes the hybrid approach particularly beneficial for applications that require real time data manipulation.

Future research could expand the hybrid approach to other domains, such as cloud computing and distributed systems, where scalability and resource management are even more critical. Improving the scalability of the hybrid data structure for distributed environments would allow it to handle larger datasets more efficiently across multiple machines. Additionally, further optimization techniques, such as enhanced compression algorithms and machine learning-driven memory management, could be explored to improve both the memory efficiency and query performance of the hybrid structure. Research into integrating the hybrid approach with emerging technologies, such as quantum computing or edge computing, could also provide valuable insights into how it can be adapted to future high-performance systems.

References

[1] S.-H. Kim *et al.*, "A 24Gb 42.5Gb/s GDDR7 DRAM with Low-Power WCK Distribution, an RC-Optimized Dual-Emphasis TX, and Voltage/Time-Margin-Enhanced Power Reduction," in *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, 2025, pp. 508 – 510. doi: 10.1109/ISSCC49661.2025.10904689.

[2] A. Ailamaki, "Databases and hardware: The beginning and sequel of a beautiful friendship," *Proc. VLDB Endow.*, vol. 8, no. 12 12, pp. 2058 – 2061, 2015, doi: 10.14778/2824032.2824142.

[3] G. More, S. Ray, and K. B. Kent, "Reconfigurable acceleration for database systems: Taxonomy, techniques, and research challenges," *J. Syst. Archit.*, vol. 171, 2026, doi: 10.1016/j.sysarc.2025.103659.

[4] E. Siswanto, D. Danang, I. Kusumaningroem, and I. Akhsani, "Assessing Software Architecture Resilience Using Quantitative Metrics in Cloud Native Application Development Environments," *Indones. J. Infomatics*, vol. 1, no. 1, pp. 11–21, 2026.

[5] M. Lee, M. Lee, and C. Kim, "A JIT compilation-based unified SQL query optimization system," in *2016 6th International Conference on IT Convergence and Security, ICITCS 2016*, 2016. doi: 10.1109/ICITCS.2016.7740304.

[6] D. Danang and Z. Mustofa, "Digital Forensics and Automated Incident Response Framework Leveraging Big Data Analytics and Real Time Network Traffic Profiling in Heterogeneous Cyber Environments," *Cyber Secur. Netw. Manag.*, vol. 1, no. 1, pp. 44–45, 2026.

[7] C.-H. Ma, X.-L. Hao, X.-F. Meng, and X.-K. Zhang, "Survey on Machine Learning for Multi-Dimensional Data Query Processing," *Chinese J. Comput.*, vol. 48, no. 1, pp. 100–123, 2025, doi: 10.11897/SP.J.1016.2025.00100.

[8] D. Danang and Z. Mustofa, "CLSTMNet Architecture: A CNN–LSTM-Based Hybrid Deep Learning Model for DDoS Attack Detection and Mitigation in Network Security," *J. Artif. Intell. Technol.*, 2026.

[9] K. Krishan, G. Gupta, and G. S. Bhathal, "Query load management: an approach for optimizing database performance," *OPSEARCH*, 2025, doi: 10.1007/s12597-025-01012-x.

[10] D. Danang, F. A. Prasetya, and E. Siswanto, "Design of Intelligent Street Lighting Systems Based on Motion and Ambient Light Sensors," *J. Multidiscip. Res. Technol.*, vol. 2, no. 1, pp. 65–79, 2026.

[11] D. Danang, M. U. Dewi, and W. Aryani, "Systematic Literature Review on the Application of Blockchain in Enhancing Server Security: Research Methods for Mitigating Ransomware and Malware Attacks," *Int. J. Comput. Technol. Sci.*, vol. 1, no. 4, pp. 27–51, 2024, doi: https://doi.org/10.62951/ijcts.v1i4.186.

[12] P. Damme, "Query processing based on compressed intermediates," in *CEUR Workshop Proceedings*, 2017. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85027850036&partnerID=40&md5=35b18c2bf237f937ed7c3e0164d2cac5

[13] D. Danang, E. Siswanto, W. Aryani, and P. Wibowo, "Hybrid Federated Ensemble Learning Approach for Real-Time Distributed DDoS Detection in IIoT Edge Computing Environment," *J. Eng. Electr. Informatics*, vol. 5, no. 1, pp. 9–17, 2025, doi: https://doi.org/10.55606/jeei.v5i1.5099.

[14] J. Choe, A. Crotty, T. Moreshet, M. Herlihy, and R. I. Bahar, "HybriDS: Cache-Conscious Concurrent Data Structures for Near-Memory Processing Architectures," in *Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2022, pp. 321 – 332. doi: 10.1145/3490148.3538591.

[15] Z. Choudhury, S. Purini, and S. R. Krishna, "A hybrid CPU+GPU working-set dictionary," in *Proceedings - 15th International Symposium on Parallel and Distributed Computing, ISPDC 2016*, 2017, pp. 56 – 63. doi: 10.1109/ISPDC.2016.16.

[16] D. Danang, M. U. Dewi, and G. Widhiati, "Federated Hybrid CNN GRU and COBCO Optimized Elman Neural Network for Real Time DDoS Detection in Cloud Edge Environments," *Int. J. Electr. Eng. Math. Comput. Sci.*, vol. 2, no. 2, pp. 28–35, 2025, doi: https://doi.org/10.62951/ijeemcs.v2i2.293.

[17] Y. Wang and K. Li, *Efficient data allocation for PCM in processing-in-memory systems*. 2022. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85152096171&partnerID=40&md5=930ba9932d8f3cc53eb4a50180b884d1

[18] A. Yu, Q. Meng, X. Zhou, B. Shen, and Y. Zhang, "Query optimization on hybrid storage," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10177 LNCS, pp. 361 – 375, 2017, doi: 10.1007/978-3-319-55753-3_23.

[19] D. Danang, E. Siswanto, N. D. Setiawan, and P. Wibowo, "Hybrid Zero Trust Container Based Model for Proactive Service Continuity under Intelligent DDoS Attacks in Cloud Environment," *Int. J. Comput. Technol. Sci.*, vol. 2, no. 3, pp. 41–49, 2025, doi: https://doi.org/10.62951/ijcts.v2i3.291.

[20] D. Danang, N. D. Setiawan, and E. Siswanto, "Pemanfaatan Teknologi Internet of Things untuk Monitoring Kualitas Air Sungai di Wilayah Perkotaan," *J. New Trends Sci.*, vol. 2, no. 1, pp. 23–34, 2024.

[21] S. Zhang, J. Qi, X. Yao, and A. Brinkmann, "Hyper: A High-Performance and Memory-Efficient Learned Index via Hybrid Construction," *Ann. Entomol. Soc. Am.*, vol. 2, no. 3, 2024, doi: 10.1145/3654948.

[22] C. Guo and H. Chen, "In-memory join algorithms on GPUs for large-data," in *Proceedings - 21st IEEE International Conference on High Performance Computing and Communications, 17th IEEE International Conference on Smart City and 5th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2019*, 2019, pp. 1060 – 1067. doi: 10.1109/HPCC/SmartCity/DSS.2019.00151.

[23] H. Zhang, D. G. Andersen, A. Pavlo, M. Kaminsky, L. Ma, and R. Shen, "Reducing the storage overhead of main-memory OLTP databases with hybrid indexes," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2016, pp. 1567 – 1581. doi: 10.1145/2882903.2915222.

[24] M. Zhang, L. Li, and X. Zheng, "RLART: An Adaptive Radix Tree Based on Deep Reinforcement Learning," *Commun. Comput. Inf. Sci.*, vol. 2214 CCIS, pp. 328 – 344, 2024, doi: 10.1007/978-981-97-8746-3_23.

[25] F. Faerber, A. Kemper, P.-Å. Larson, J. Levandoski, T. Neumann, and A. Pavlo, "Main memory database systems," *Found. Trends Databases*, vol. 8, no. 1–2, pp. 1 – 130, 2017, doi: 10.1561/1900000058.

[26] P. Memarzia, S. Ray, and V. C. Bhavsar, "The art of efficient in-memory query processing on NUMA systems: A systematic approach," in *Proceedings - International Conference on Data Engineering*, 2020, pp. 781 – 792. doi: 10.1109/ICDE48307.2020.00073.

[27] S. Floratos *et al.*, "NestGPU: Nested query processing on GPU," in *Proceedings - International Conference on Data Engineering*, 2021, pp. 1008 – 1019. doi: 10.1109/ICDE51399.2021.00092.

[28] S. A. Aula and T. A. Rashid, "FOX-TSA: Navigating Complex Search Spaces and Superior Performance in Benchmark and Real-World Optimization Problems," *Ain Shams Eng. J.*, vol. 16, no. 1, 2025, doi: 10.1016/j.asej.2024.103185.

[29] T. Lale, "GWO-WOA-AOA: Multistage Hybrid Metaheuristic Optimization Approach," in *ISAS 2025 - 9th International Symposium on Innovative Approaches in Smart Technologies, Proceedings*, 2025. doi: 10.1109/ISAS66241.2025.11101906.

[30] D. Danang, A. B. Santoso, and M. U. Dewi, "CICA Framework: Harnessing CSR, AI, and Blockchain for Sustainable Digital Culture," *Int. J. Adv. Comput. Sci. \& Appl.*, vol. 16, no. 11, 2025.

[31] D. Danang, H. Haryani, Q. Aini, F. A. Ramahdan, and J. Edwards, "Empowering Digital Literacy Through Blockchain Based Alphasign for Secure and Sustainable E-Governance," 2025.

[32] W. Rödiger, T. Mühlbauer, A. Kemper, and T. Neumann, "High-speed query processing over highspeed networks," in *Proceedings of the VLDB Endowment*, 2016, pp. 228 – 239. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84976478057&partnerID=40&md5=3fc3572973fe8bfc88879a54e372a853

[33] P. P. J. Suresh, U. D. Acharya, and N. V. S. Reddy, "Mining Frequent Itemsets from Transaction Databases Using Hybrid Switching Framework," *Multimed. Tools Appl.*, vol. 82, no. 18, pp. 27571–27591, 2023, doi: 10.1007/s11042-023-14484-0.

[34] J. Kozak, "Evolutionary computing techniques in data mining," *Stud. Comput. Intell.*, vol. 781, pp. 29 – 44, 2019, doi: 10.1007/978-3-319-93752-6_2.

[35] L. Zaourar *et al.*, "Case Studies on the Impact and Challenges of Heterogeneous NUMA Architectures for HPC," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 14842 LNCS, pp. 251 – 265, 2024, doi: 10.1007/978-3-031-66146-4_17.

[36] N. Denoyelle, B. Goglin, A. Ilic, E. Jeannot, and L. Sousa, "Modeling non-uniform memory access on large compute nodes with the cache-aware roofline model," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, pp. 1374 – 1389, 2019, doi: 10.1109/TPDS.2018.2883056.

[37] H. Mitake, H. Yamada, and T. Nakajima, "Looking into the Peak Memory Consumption of Epoch-Based Reclamation in Scalable in-Memory Database Systems," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11707 LNCS, pp. 3 – 18, 2019, doi: 10.1007/978-3-030-27618-8_1.

[38] Danang, T. Wahyono, I. Sembiring, T. Wellem, and N. H. Dzulkefly, "An Adaptive Framework Integrating ML Blockchain and TEE for Cloud Security," in *Proceeding - 2025 4th International Conference on Creative Communication and Innovative Technology: Empowering Transformative MATURE LEADERSHIP: Harnessing Technological Advancement for Global Sustainability, ICCIT 2025*, 2025. doi: 10.1109/ICCIT65724.2025.11167152.