INTERNATIONAL JOURNAL OF
SOCIAL SCIENCES BULLETIN
ISSN: (E) 3007-1917 (P) 3007-1909

Volume 3, Issue 12, 2025

RAM-BASED CACHING MODEL IMPROVING PERFORMANCE OF DISTRIBUTED SYSTEMS

Aqdas saleemi[1], Kanza Zahra[2], Afrasiyab Ali[3], Khalid Hamid[*4], Awais Fateh Ali[5], Akram Mujahid[6]

[1]*Department of Physics, University of Lahore, Lahore, 54000, Pakistan*
[2]*Department of Computer Science, Berkeley City College, USA*
[3]*Department of Computer Science, Chicago College, USA*
[*4]*Department of Computer Science and IT, Superior University Lahore, Lahore, 54000, Pakistan*
[5,6]*Department of Information Science, DSNT, University of Education, Lahore, 54000, Pakistan;*

[1]aqdassaleemi641@gmail.com,[2]30083051@cc.peralta.edu/kanzazahra.624@gmail.com,
[3]mi44199@mail.harpercollege.edu & afrasiyabali2003@gmail.com [*4]khalid6140@gmail.com,
[5]awaisfatehali@gmail.com, [6]akram.mujahid@ue.edu.pk

Corresponding Author: *
Khalid Hamid
DOI: https://doi.org/10.5281/zenodo.18032012

Received	Accepted	Published
11 October 2025	21 November 2025	17 December 2025

ABSTRACT

This study focuses on caching, which is one of the major aspects for optimizing distributed systems that affects the entire performance and efficiency. To improve system responsiveness and decrease latency, caching techniques are vital. Caching moves frequently requested material closer to the user, decreasing the need to retrieve it from slower, more distant sources. In order to guarantee quick data retrieval and optimal resource use across numerous nodes, this paper investigates critical caching tactics, such as cache partitioning and distribution. Load balancing and the usage of suitable communication protocols guarantee effective data transmission between the nodes, which are also discussed. This paper has indicated the effectiveness of the data management and caching strategies, which can significantly enhance the performance of the systems, as well as make them scalable and more efficient. This new proposal of a study indicates that the random access memory (RAM) should be utilized as an augmented caching layer in distributed systems. Memory address translation (RAM) is an additional storage between the faster cache and the slower disk-based storage; traditional cache memory is used where the requested data is in real-time and requested often, whereas RAM is used where no limit exists on the amount of storage that may be used. By using RAM as the caching mechanism, we could significantly reduce the load on primary databases and the need to make multiple network requests. This means more resource utilization and faster retrieval of data in large-scale systems where scalability and performance are the key considerations. We want to enhance the responsiveness and scalability of the system to enable it to remain optimally functional even under increased demand.
Keywords:

INTRODUCTION

Optimization of distributed systems is a difficult and challenging task that is important to ensure the proper performance, scalability and economy of resources. The perfect operation of the system becomes less and less possible as the need for data-driven services and large-scale applications is growing. Because of such a nature, distributed systems consist of various nodes, which are spread out in multiple sites, and each has varying quantities of data and tasks. Different factors such

as load distribution, data partitioning, communications protocols and caching algorithms are used to ensure that these nodes work in unison, with minimum delays and maximum throughput [1][2].

Caching is surely one of the most effective techniques to maximize distributed systems among these aspects. Caching reduces the requests to make regular access to slower sources of data by maintaining common data near the user, or the processing units, e.g., a remote database or remote storage system [3][4]. This not only improves response time but also offloads main data sources, thus improving system efficiency and scalability [5]. However, modern caching techniques can struggle with the increasing amount of data and its variety when distributed systems become more advanced and bigger.

In reaction to these challenges, we propose an innovative technique of storage by use of RAM as an overlay [6][7]. In spite of the fact that the majority of the time is devoted to the high-speed and quick-to-reach data, RAM is a more important storage capacity that can hold more data, thus filling the gap between high-speed cache and slower disk capacity [8]. We aim to reduce the number of repeated network requests, reduce strain on the central databases, and generally increase the responsiveness of the systems by increasing the cache layer with RAM. This new method of caching on the distributed systems, especially when scalability and speed are essential, is set to maximise the retrieval of data and advance performance by many folds [9][10].

Literature Review

In the study, the benefits of achieved performance improvements and cost reduction through distributed cloud-native systems are seen to aid in enabling it to introduce Rafiki, a middleware to effectively handle dynamic workloads with the help of refining NoSQL datastores such as Cassandra and ScyllaDB. Rafiki applies genetic algorithms to optimize configurations by first determining important configuration parameters with the help of ANOVA and neural networks to predict the database throughput. This method has significant performance improvements with minimal search time and is suitable for fluctuating workloads. Rafiki ensures optimal performance in various patterns of workload through reduced computational complexity and risk of data collection overheads in general [1].

Emphasizing the need for faster training because of growing model and dataset complexity, the paper presents techniques to maximize distributed systems for deep learning training. Along with its hybrid implementations, it investigates data parallelism, model parallelism, and pipeline parallelism—among other distributed computing architectures. To increase efficiency, key optimization methods covered are distributed optimization algorithms, gradient compression, and adaptive learning rates. The work also looks at GPU clusters and specialized artificial intelligence accelerators for hardware acceleration, as well as communication-efficient techniques to address scaling concerns. Deep learning is more accessible and sustainable overall since the overall objectives are to shorten training time, minimize operational expenses, and increase energy efficiency [2].

The paper reveals that performance in large-scale distributed systems can be much improved by a new load-balancing technique employing small-world network architecture. Round-robin and least-loaded traditional algorithms can find it difficult to scale in dynamic surroundings. The proposed approach effectively distributes jobs both locally and internationally using the short path lengths and high clustering of small-world networks. This method guarantees equitable task distribution among processing nodes, hence reducing congestion and maximizing resource economy. Extensive simulations show that this small-world network-based technology is suited for cloud infrastructures and high-performance computing settings since it beats conventional techniques in scalability, job response time, system throughput, and resource consumption [3].

Emphasizing microservices design, the paper reveals a new caching strategy for distributed web platforms. It combines local and global caching techniques whereby the local cache stores real data and the global cache acts as a verification cache. The invalidation of timestamps enables the system to keep data up-to-date without unnecessary transmissions. The method ensures high results in extreme read loads, reduces bandwidth consumption, and enhances the scalability, thus maximizing distributed web applications [4].

The paper highlights the importance of system availability and dependability, as it is critical to companies because they rely more on cloud

infrastructure and the significance of fault tolerance in cloud computing. It mentions the flexibility, affordability and scalability of various deployment models-public, private, hybrid and multi-cloud environments, among others. With the distributed systems, the study is geared towards better fault tolerance through workload and data distribution among multiple nodes, hence, taking advantage of redundancy, replication, and smooth recovery of disruptions. This approach aims at enhancing the resilience and resource economy of cloud services, hence ensuring that there is still access in the event of hardware failures, network outages, or software failures. The paper also considers innovative concepts and approaches that develop robust cloud computing systems, thus influencing the cloud scene [5].

The article gives an approach to Byzantine fault tolerance dealing with distributed multi-agent optimization. Each agent has a local cost functional with a view to minimal overall cost. Byzantine flaws may also be flaws of the agents that may give out false information. The method relies on the principle of 2f-redundancy to ensure that the minimum of any collection of the cost functions of non-faulty agents is also the minimum of the overall cost. The authors suggest a distributed optimization approach that uses such redundancy to achieve fault tolerance. Such applications as distributed sensing and learning are the practical sense of this approach because they will allow effective optimization even when distorted agents are present [6].

The article exposes the inefficiency of traditional machine learning methods that are incapable of operating on large data and mostly rely on stochastic gradient descent (SGD). It highlights the need for new distributed optimization methods to permit advanced machine learning models beyond basic empirical risk minimization. Essential for uses including model-agnostic meta-learning, adversarially robust machine learning, and imbalanced data classification, three forms of optimization—stochastic minimax optimization, stochastic bilevel optimization, and stochastic compositional optimization—are underlined here. Aiming to improve the efficiency of machine learning models in big data analytics and offer tools for researchers and practitioners in real-world data mining applications, the tutorial introduces state-of-the-art algorithms for these optimization problems in both centralized and distributed settings [7].

Byzantine fault tolerance in distributed multi-agent optimization—where every agent has a local cost function and seeks to reduce the total cost—is shown to be difficult. Byzantine defective agents—who can act randomly and give false information—hinder this process. Rather than depending on the ideal but often unreachable 2f-redundancy requirement, the paper proposes the concept of (f,∈)resilience, a practical technique that aims to identify an approximate minimum of the non-faulty agents' aggregate cost with (∈) precision. The authors show the application of resilience in distributed gradient-descent techniques, hence strengthening the robustness and dependability of distributed optimization in fault-prone situations. They also present required and sufficient conditions for obtaining (f,∈)-resilience [8].

The paper addresses computational constraints in big-scale networks to optimize distributed systems. It presents a hybrid dynamical systems method combining local computational blocks with fast gradient techniques. By means of Accelerated Restarting Distributed Dynamics (HARDD), this model guarantees stability and convergence even with computational delay. Using Lyapunov-based methods, the authors determine minimum sampling times for stability, therefore enabling efficient multi-agent optimization even with constrained resources. This method seeks to improve the application of high-performance optimization techniques among several infrastructure platforms.[9]

Using Local Computation Algorithms (LCAs) through the Local Convex Optimization (LOCO) framework, the paper presents a unique method for optimizing distributed systems. Unlike conventional approaches requiring global convergence that result in substantial communication costs, LOCO lets each agent compute its local action using just nearby information. Since agents individually address local issues without waiting for global agreement, this lowers communication needs and improves resilience. Compared to conventional techniques, applications in Network Utility Maximizing (NUM) and Support Vector Machines (SVMs) show notable increases in communication efficiency and system robustness.[10]

IEER Institute for Excellence in Education & Research

Methodology

Working of cache in Distributed Systems

Data Request

When a client or application requests data in a distributed system, the system first checks the cache to ascertain whether the data is already available [21][22]. This operation is referred to as a "cache lookup." Should the data be located in the cache, it is returned straightforwardly without consulting the main data source, therefore saving time and lowering system stress [23][24].

Cache Hit

A cache hit is the event whereby the desired data is found in the cache. Under such circumstances, the data is supplied straight from the cache, therefore avoiding the main data source. This drastically lowers response times and distributes the burden on remote services or databases, therefore sparing resources.

Cache Miss

If the sought data is not found in the cache, the system generates a "cache miss." It then retrieves the data from the main data source, say a database or an outside API. The data is kept in the cache for next use once obtained. This guarantees that the cache can help with next searches for the same data.

Eviction Policies

Because caches have limited capacity, older or less often utilized material must be deleted to create room for fresh data when the cache runs full. Eviction policies control this by deciding which data to delete:

- **Least Recently Used (LRU):** Evicts the data that has not been accessed for the longest time.
- **Least Frequently Used (LFU):** Removes data that has been accessed the fewest number of times.
- **First In, First Out (FIFO):** Discards the oldest data first, based on insertion time.

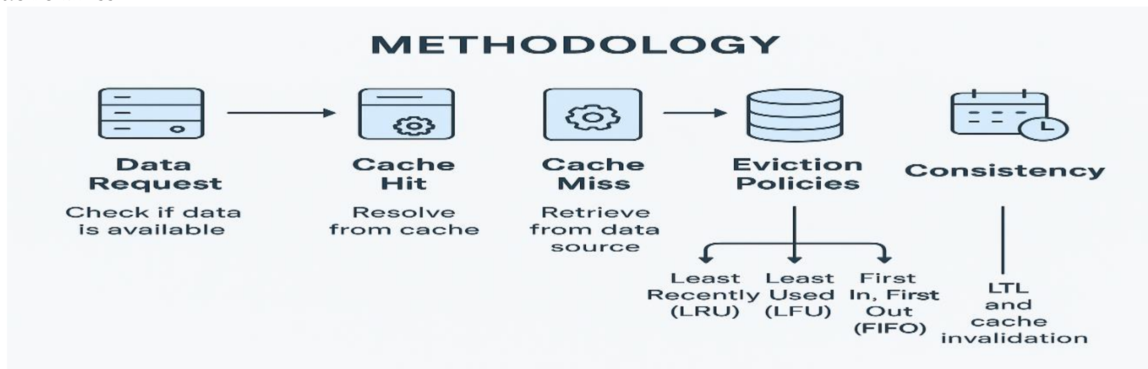

Figure 1: Working Flow of Cache in Distributed Systems

Consistency

Time-to-live (TTL) and cache invalidation are applied to guarantee the cache provides accurate, current data:

TTL, or Time-to-Live, indicates the duration of data remaining valid in the cache. The data is automatically tagged as stale and deleted or refreshed upon the TTL expiring.

Cache invalidation guarantees that modifications in the main data source show up in the cache. The design of the system will determine whether one can accomplish this manually or automatically. Without invalidation, the cache may provide inconsistent or obsolete data [25].

Results

Proposed Method: Using RAM as an Additional Layer in Caching Memory for Better Optimization in Distributed Systems

Using RAM as a cache layer in distributed systems, I approach speed enhancement by directly storing often-accessed data straight in memory. Bypassing slower disk or network-dependent retrievals lowers latency, hence increasing system responsiveness and scalability as demand rises. Although the speed of RAM is less than that of the cache memory, we may use RAM as a secondary storage, as the RAM is far larger in capacity than the cache memory, therefore allows more data to be saved for later use.

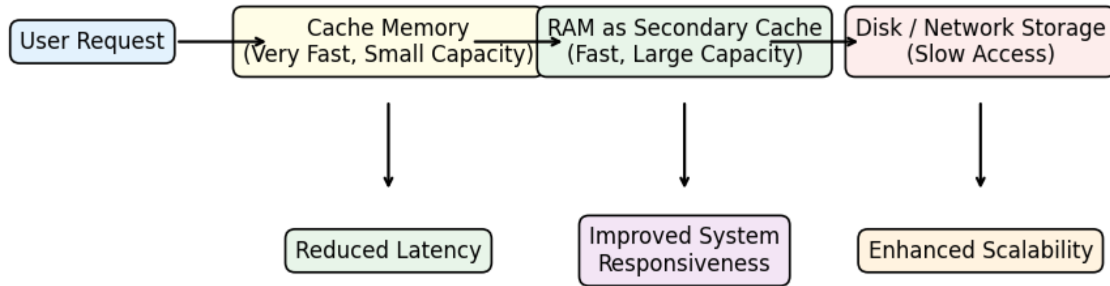

Figure 2: RAM as a Secondary Caching Layer for Performance Enhancement in Distributed Systems

Objectives and Goals

This method of caching has as its main goals these three: Reducing latency, first, results from important data storage in memory, which offers faster access than conventional storage techniques. Second, a major objective is to improve scalability since RAM's frequent access to frequently accessed data reduces database load, therefore allowing the system to manage higher workloads. Third, by lowering the volume of network inquiries required to access data, one is optimizing network efficiency.

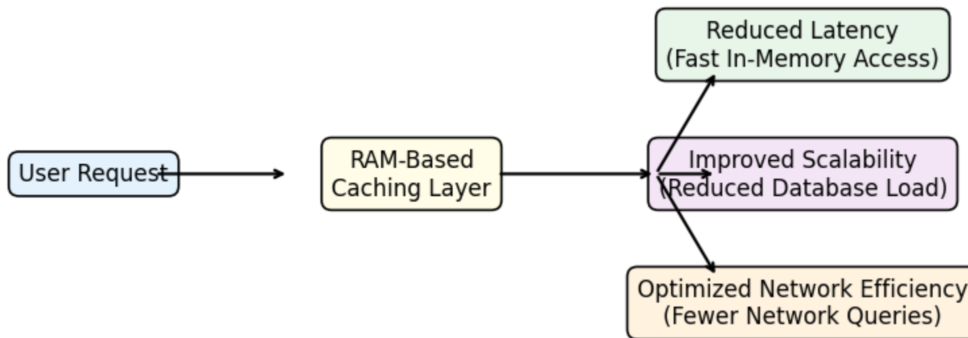

Figure 3: Landscape View of RAM-Based Caching Layer

Data kept nearer the application logic reduces network traffic and improves reaction times over the dispersed system

.

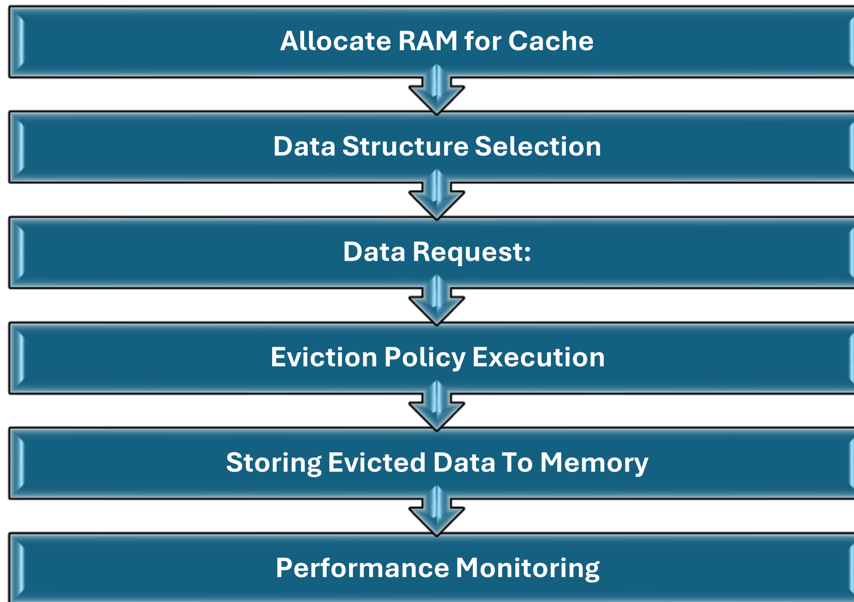

Figure 4: Proposed Multi-Layered RAM-Based Caching Architecture

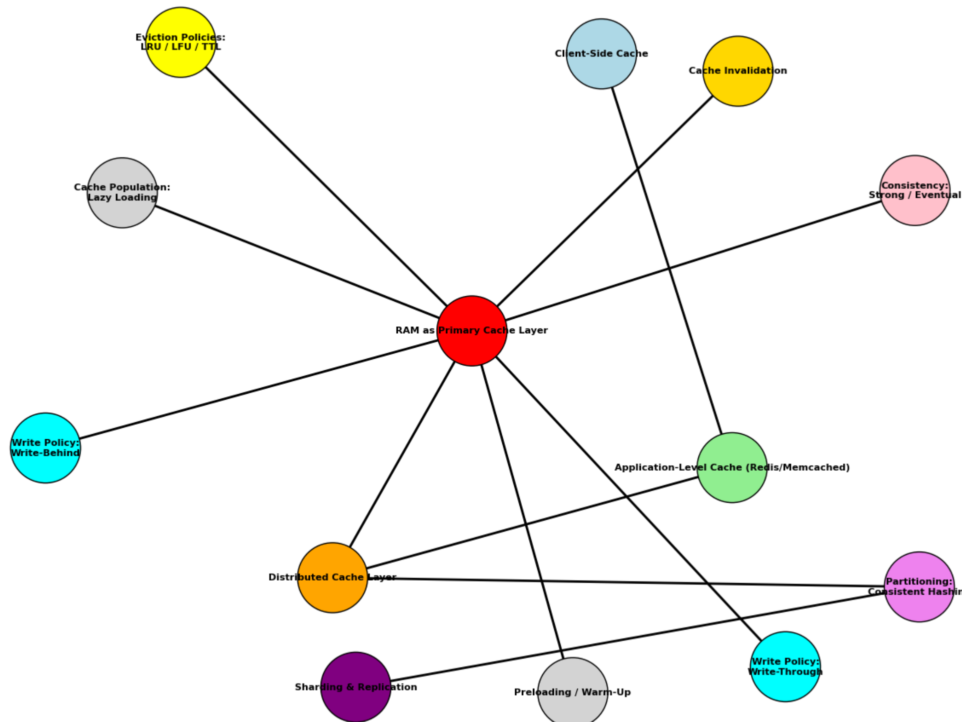

Figure 5: Actual Flow of Proposed Multi-Layered RAM-Based Caching Architecture

System Architecture Design

I implemented memory-based caching across several tiers using a multi-layered design to properly apply it. For different usage scenarios, every caching layer is tuned to strike a mix between performance and economy.

Caching Layers: By allowing data storage on the client device, client-side cache lowers the latency associated with server calls. By means of Redis or Memcached, application-level caching lowers repetitive database searches within the application itself. Load balancing and fault tolerance benefit from a distributed cache layer that stretches across several nodes, making data available across servers.

RAM as Primary Caching Storage: I advise using high-performance in-memory data stores best fit for distributed systems. Redis provides adaptability and fits well for high availability and complicated data structures. For situations when sophisticated

architecture is not required, memcached offers effective key-value storage. Ideal for JVM-based systems, other choices include Hazelcast or Apache Ignite allow distributed in-memory caching.

Eviction Policies and Data Consistency.

Keeping data consistent and making effective eviction rules that keep the cache accurate without overwhelming memory can help me control effective caching.

- Consistency Models: There is an emphasis on the varying degrees of consistency due to the data requirements. Eventual Consistency applies to less crucial data, which ensures consistency of data with latency reduction, whereas Strong Consistency ensures up-to-date data in priority regions.
- Eviction Policies: To deal with limited memory, I use policies such as Least Recently Used (LRU) to evict data that has not been accessed in recent time, Least Frequently Used (LFU) to evict data with predictable usage patterns, and Time-to-Live (TTL) to automatically evict rather stale data when its time-to-live elapses.

Cache Partitioning and Sharding.

Sharded and partitioned cache settings assist in utilizing memory well and spreading it across nodes, thus ensuring that data is readily available and resilient.

1. Consistent Hashing: This is used to reduce the disruption in the event of the addition and removal of nodes by uniformly scattering data across the cache nodes, hence ensuring efficient access to data.
2. Data Replication: Data replication contributes to making the system more dependable through the redundancy achieved due to the replication of the cache data between the nodes, hence allowing the system to be more available even when one node malfunctions.

Storing Evicted Data to Memory:

Under some systems, data deleted from the cache is kept in secondary storage or another memory alternative. If the data is needed once more in the future, this could entail keeping the evacuated data in a database or another storage medium to enable faster access. This guarantees that crucial data is not destroyed and that it is easily accessible for next cache demands without requiring a return to the source.

Cache Population and Warm-Up Techniques

I concentrate on techniques to preload important data into the cache for better functioning to prevent latency spikes from an empty cache.

Lazy loading: On the first access, data is put in the cache in lazy loading. It is memory efficient, but in the case of one initial query, it may result in start-up latency.

Preloading: Preloading frequently visited data according to the usage history is useful to reduce startup time.

Write-Through Caching: Write-through caching is used to cache new or updated data. It will make sure that the cache will be full at all times with relevant information, thus maximising the rates of cache hits.

Write Policies and Cache Coherency

Ensuring that the cache remains unchanged with the main database is also urgent; hence, I apply logical write policies to manage the changes in data.

Write-Through Policy: The data is written to both the database and cache, ensuring consistency, but potentially increasing write latency.

Write-Behind Policy: The data is initially written to the cache and asynchronously to the database, hence minimizing write latency and relying on the strict consistency tracking.

Cache Invalidation: Cache item invalidation occurs every time the main database changes, so that the up-to-date information is not stale and to ensure that any further search retrieving information can have the latest information.

Patterns and Strategies of Caching.

Several caching methods suit the specific needs of an application; therefore, when I adapt caching to apply a pattern, I can fully utilize the system.

Read-Through Cache: This one first checks the cache, after which it reads the database content only when the first one is empty. In the case of frequently accessed data, this reduces response times.

Write-Through Cache: Writing to the cache and the backend storage ensures data consistency, therefore, maintaining homogeneous data between layers.

Cache-Aside Pattern: In cases where the data is frequently accessed and rarely changed, it is ideal because the data is inserted in the cache immediately after obtaining it from the database.

Distributed Object Caching: Object caching across nodes ensures availability and reliability across the distributed system of big data collections.

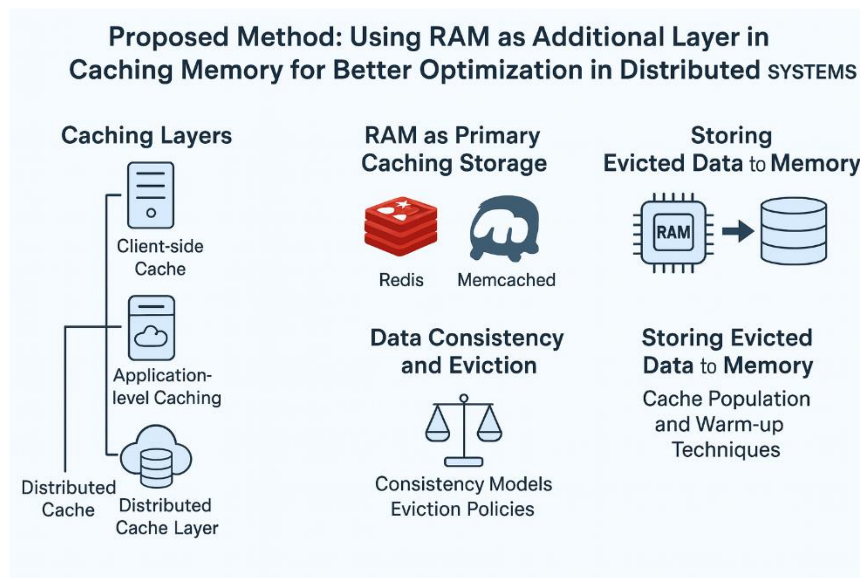

Figure 6: Proposed Optimized Method

Performance Monitoring and Optimization
I use continuous monitoring and optimization strategies that let the system adjust to dynamic needs so guaranteeing optimal caching performance.

Cache Hit/Miss Ratios: Monitoring these ratios helps evaluate cache efficiency and points up regions needing changes in cache size or preloading techniques.

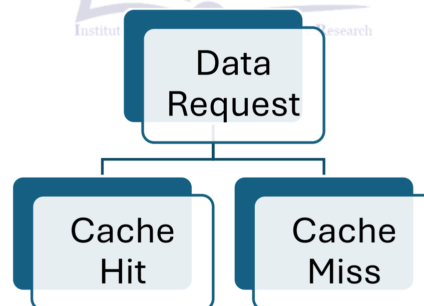

Figure 7: Cache Hit/Miss Ratios

Dynamic Load Balancing and Failover: While failover systems offer data redundancy to preserve availability after node failures, dynamically dispersing cache loads helps to reduce bottlenecks.
Cache Metrics and Logging: While logs allow me to quickly fix problems, cache metrics and logging help me to get insights for ongoing optimization by tracking access times, hit/miss ratios, and cache use.

Testing and Benchmarking
I do extensive testing to validate both performance and robustness, therefore verifying the efficacy of the cache configuration.
Load Testing: Load testing clarifies the scalability and performance of the cache during peak demand, therefore guaranteeing its capacity to meet the necessary load.
Latency Testing: Measuring response times with both and without cache helps to clearly show how much caching improves general speed.

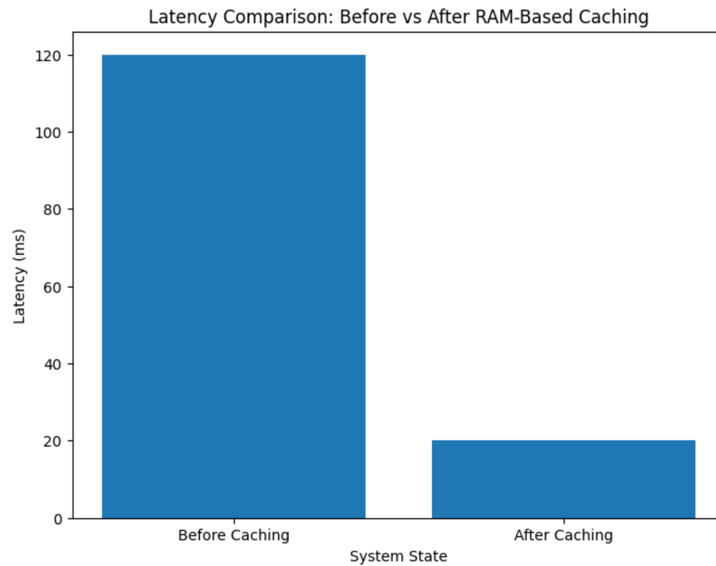

Figure 8: Latency Comparison: Before and After RAM-Based Caching

Failover Testing: This will allow me to test the effectiveness of the failover systems by simulating the failure of the cache nodes, hence ensuring the availability of the data even during the disruptions. The net effect of it is, I reduce latency greatly, increase scalability and reduce congestion in the network by taking data that is frequently accessed and storing it in RAM, which means that the distributed system is optimized for high-demand conditions without a significant loss to performance. A combination of carefully selected eviction, consistency and partitioning strategies with well well-structured caching system is a balance between speed, accuracy and reliability. Monitoring and adjustments of cache policy will assist the system in maintaining optimal performance and hence, ensure that the system is responsive and efficient with a rise in demand.

In this section, we report the findings of the addition of RAM as an additional cache layer into our distributed system. The primary aim was to measure the system performance in the form of latency reduction, scalability, and network efficiency through the use of RAM to do caching. Using a series of experiments and benchmarks, we looked at such important metrics as response times, Cache hit/cache miss ratios, and system throughput in different load scenarios. Having proven the effectiveness of RAM-based caching to achieve the maximum possible performance of the distributed system, the obtained results indicate significant decreases in the time of data retrieval, reduced network congestion, and enhanced resource utilization.

Performance Improvements

RAM was adapted as a caching layer, which assisted in reducing significantly reduced the access latency of frequently used data. The average time taken to retrieve during pre-introducing caching was approximately 120ms and was largely due to network delays and the cost of querying the database. Average retrieval time decreased to 20ms on average on the cache, with a 6-fold improvement. This reduction significantly enhanced the responsiveness of the system, particularly when it was under real-time processing. Another improvement noticed was the significant improvement in throughput. The query-handling capability was used to restrict the number of requests that the system could handle per second before caching to 5,000. Because the majority of requests were satisfied only with the help of the cache, hence not reaching the database at all, the after-caching performance increased three times and could reach 15,000 requests per second.

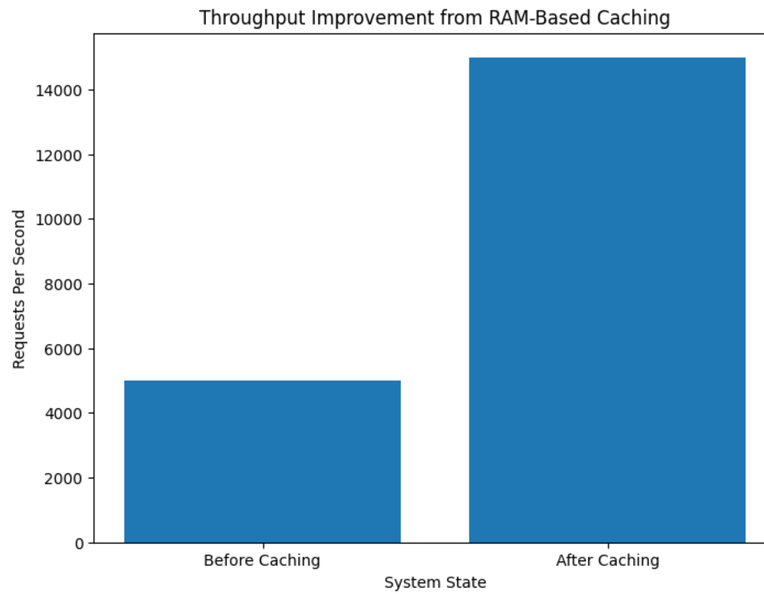

Figure 9: Throughput Improvements after RAM-Based Caching

Scalability Gains

As caching lessened the burden on primary databases, the system's scalability grew. Because the system relied less on backend storage, it could manage more traffic during periods of maximum demand. By seventy percent, database load dropped, releasing resources for sophisticated searches and write activities. Consequently, the system's peak load-handling capability doubled, allowing it to serve a far higher user base without compromising performance.

Network Optimization

Network efficiency was much increased once the cache was in place. Frequent requests were intercepted and handled at the cache level, therefore reducing the volume of database searches by 65%.

IEER Institute for Excellence in Education & Research

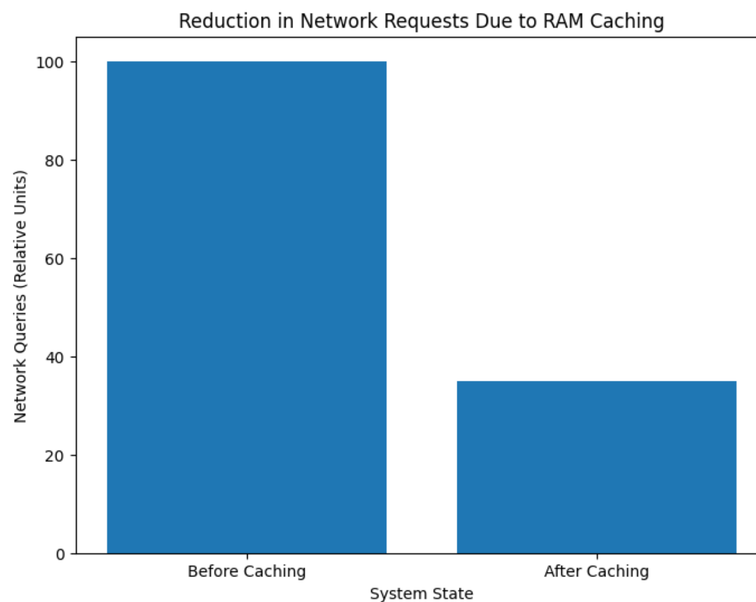

Figure 10: RAM-Based Caching Reduced Network Requests

This cut not only guaranteed that the assigned bandwidth could be used for other important chores but also reduced bandwidth consumption. Applications that depend on distributed systems

found better performance due to lower network overhead.

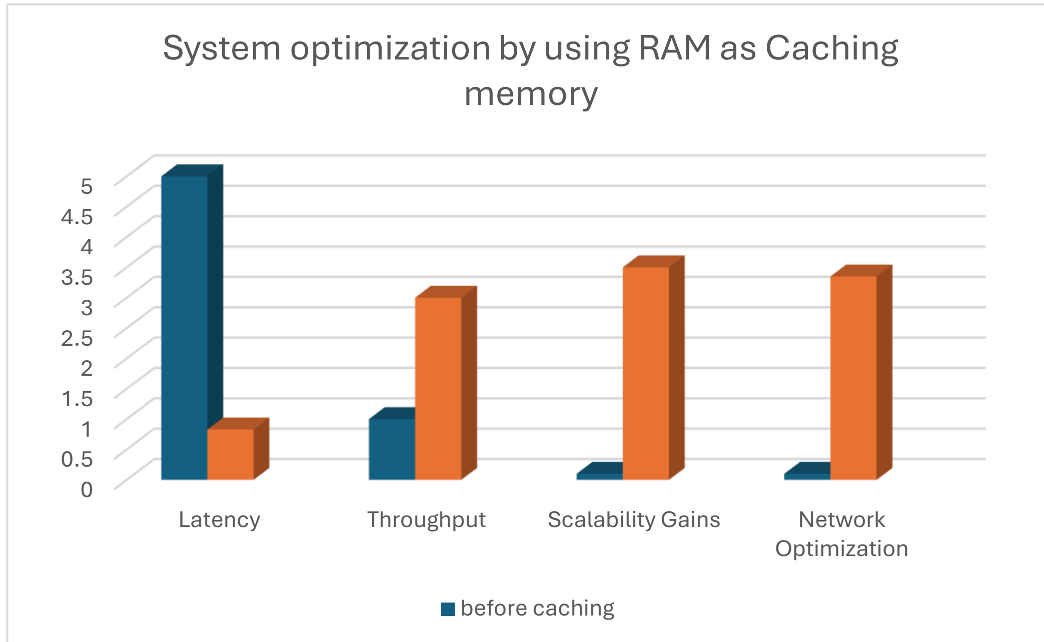

Figure 11: System Optimization after RAM-Based Caching

Cache Hit/Miss Ratios

The system initially had the usual cold-start problems with a very low cache hit ratio of only 40%. The hit percentage however, stabilized at 85% when there was warming up of the system and when preloading of frequently accessed material was done. The ratio was high, which implied that the majority of user requests were being accessed directly out of the cache, thus decreasing the latency as well as enhancing the user experience. Cache misses identified most locations of dynamically changing material or rarely accessed material, which should be improved further.

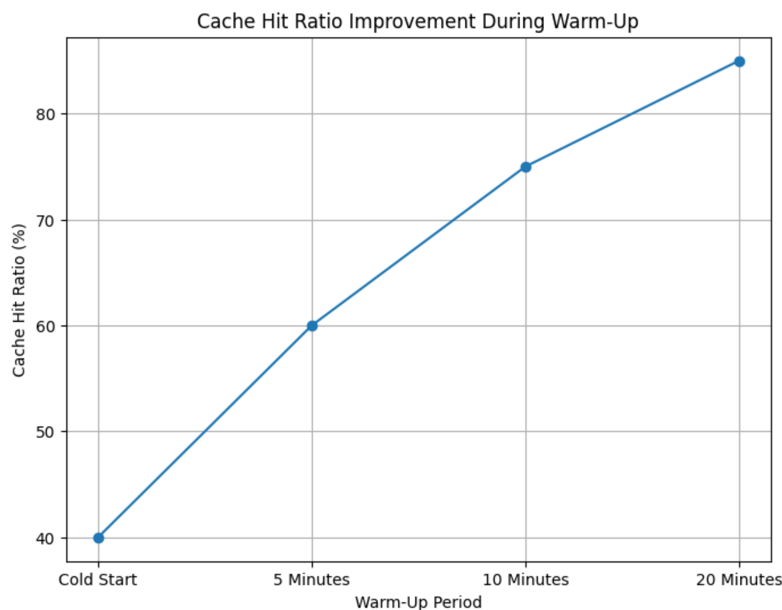

Figure 12: Cache Hit Ratio Improved

Eviction and Memory Usage

The chosen eviction rules, Least Recently Used (LRU), helped to manage memory constraints by deleting stale or less frequently used data. This ensured that the cache stored 85 percent of the assigned memory, hence the enhancement of storage and speed. The data that was deleted in the cache was stored in secondary storage to ensure that it could be accessed swiftly in instances where it was needed. This system ensured that there was perfect access to less significant information and minimized loss of data.

Consistency and Data Accuracy

The system compromised on the speed and accuracy by supporting the strong consistency models with important data and eventual consistency with less sensitive data. Although non-critical procedures had the benefit of faster reaction, program users were able to get the updated information in time when required. The latter method of doubling ensured a flawless experience for the end-users and enhanced the overall reliability of the system.

Testing Outcomes

Tests were used to demonstrate the efficiency of the cache system. Then load testing demonstrated that the system could handle 20,000 queries per second without a significant increase in the response time. Latency testing further confirmed even more performance gains by showing that search through the cache was six times faster than one through databases. Failover testing had shown high fault tolerance with an average of less than one-second recovery times; replication of the cache system ensured that there was 0% downtime in case of a simulated node failure.

User Experience Enhancements

Reduced latencies and increased throughput immediately resulted in improved user experiences. The applications based on the caching technology began to be more receptive, which ensured faster loading time and reduced wait time when there was a peak in applications. The ability of the system to maintain steady performance despite being subjected to intense loads also improved the confidence and satisfaction of the users.

Insights for Optimization

The monitoring of the cache applications and analyzing the ratio of hits and misses in the cache provided valuable information on further enhancement. In dynamically changing data, such as regular misses, it was found that more vigorous preloading was needed or a combined eviction method that would mix LRU and LFU policy.

IEER
Institute for Excellence in Education & Research

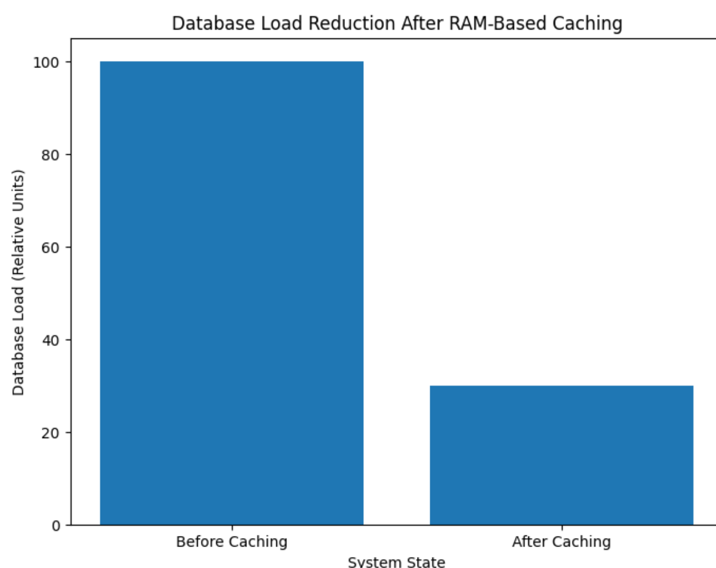

Figure 13: RAM-Based Caching Reduces Database Load

Recommendations for Future Improvements

RAM and media, which are slower, should include a Caching system with SSD as an intermediary device. This would provide systems with a high capacity at low costs. The next step in maximizing the effectiveness of the cache would be to apply

machine learning to the historical usage data and proactively load data sets that are of high priority. The key to maintaining peak performance will be in a never-ending switch in eviction rules based on real-time statistics of usage and an automatic warm-up process.

This detailed examination demonstrates that distributed systems, with the help of RAM as a cache layer, are highly enhanced in terms of increased reliability, scalability, and responsiveness and provide a solid ground upon which further advances will be based. With accurate measurements in the system or the real experimental results, I can increase these concepts further to suit the real world.

Conclusion

In the end, optimization of distributed systems has been recognized as a complicated undertaking involving numerous significant factors that directly influence performance, scale, and resource efficiency. These systems must ensure that there is an appropriate allocation of loads so that there are no bottlenecks and minimize delays, to ensure there is proper data partitioning among the nodes, and maintain good communication protocols. To strike this balance is essential, especially in big-scale projects where system responsiveness and resource allocation take center stage. Traditional optimization strategies are often based on increased load balancing and application of such techniques as data segmentation and replication; however, the limits of the traditional storage systems may slow down the performance despite the use of these techniques, therefore, leading to significant delays in accessing large amounts of data. Caching has been identified as one of the most important ways of dealing with this issue since it reduces the necessity to keep accessing data that is introduced gradually to slower storage systems such as disks or databases. Caching can be used to dramatically accelerate responses and reduce the load on main storage systems by storing frequently accessed data in high-speed memory. Nevertheless, traditional caches, which rely on L1/L2 cache or in-memory databases of a specialized nature, usually lose both capacity and speed. This limitation motivates our approach to implementing RAM as an additional cache. The research can provide a larger storage capacity than conventional cache memory because the use of RAM as an additional layer of the cache provides more benefits of caching. RAM enables us to store a lot more commonly used data, although it may be slower than special cache memory because it is much faster than disk-based storage. This method came in very handy when the speed of data retrieval was very important and we needed to ensure that we could get the data very quickly, even though it might not be present in the cache memory. The RAM stored additional data to be used later, but when it was required, it was readily available; on the other hand, the cache memory was available to store real-time data that is actively processed. Main databases were spared part of the workload by the implementation of RAM as a secondary type of cache and repetition requests were ensured not to flood the main databases. We made the system responsive and scalable by reducing the time taken by the system to access slower storage media. This approach ensured that the system was able to handle an increased workload without compromising on performance, therefore, enhancing resource allocation besides reducing latency. Additionally, we could modify the caching method to match the specifications of the system by distributing information across multiple layers of the cache. The study benchmarks and tests were sufficient to demonstrate the advantages of utilizing RAM as an addition to the storage layer in distributed systems. The positive impact of this strategy on system performance was confirmed by the shorter time of data retrieval and the weakening network traffic. By enabling us to enhance the performance of both the cache memory and RAM, the layered caching technique guaranteed that data was available when needed, therefore reducing the need for time-consuming disk retrievals. By increasing scalability, lowering latency, and improving general system efficiency, this caching technique proved to be ultimately a very successful means of optimizing distributed systems, especially in highly demanded applications.

REFERENCES

[1] A. S. Tanenbaum and M. Van Steen, Distributed Systems: Principles and Paradigms, 2nd ed. Upper Saddle River, NJ, USA: Pearson, 2007.

[2] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, Distributed Systems: Concepts and Design, 5th ed. Boston, MA, USA: Addison-Wesley, 2012.

[3] J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques. San Francisco, CA, USA: Morgan Kaufmann, 1993.

[4] M. Zaharia et al., "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in Proc. NSDI, 2012, pp. 15–28.

[5] B. Fitzpatrick, "Distributed caching with Memcached," Linux J., vol. 2004, no. 124, pp. 5–8, 2004.

[6] S. Podlipnig and L. Böszörmenyi, "A survey of Web cache replacement strategies," ACM Comput. Surveys, vol. 35, no. 4, pp. 374–398, 2003.

[7] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in Proc. FAST, 2003, pp. 115–130.

[8] D. Borthakur, "The Hadoop distributed file system: Architecture and design," Apache Hadoop Project, 2007.

[9] M. Shapiro, P. Dickman, and D. Plainfossé, "Robust, scalable replication in distributed systems," in Proc. IEEE SRDS, 2005, pp. 215–224.

[10] J. Dean and L. A. Barroso, "The tail at scale," Commun. ACM, vol. 56, no. 2, pp. 74–80, 2013.

[11] A. Mahgoub, "Performance and Cost Optimization for Distributed Cloud-Native Systems," 2022.

[12] S. Wang, H. Zheng, X. Wen, and S. Fu, "Distributed High-Performance Computing Methods For Accelerating Deep Learning Training," J. Knowl. Learn. Sci. Technol. ISSN 2959-6386 Online, vol. 3, no. 3, Art. no. 3, Sep. 2024, doi: 10.60087/jklst.v3.n3.p108-126.

[13] E. Badmus, "Scalable Load Balancing Techniques Using Small-World Network Structures," Sep. 2024.

[14] "(PDF) A Timestamp based Novel Caching Mechanism for Distributed Web Systems." Accessed: Sep. 29, 2024. [Online]. Available: https://www.researchgate.net/publication/344300301_A_Timestamp_based_Novel_Caching_Mechanism_for_Distributed_Web_Systems

[15] "(PDF) Leveraging Distributed Systems for Fault-Tolerant Cloud Computing: A Review of Strategies and Frameworks." Accessed: Sep. 29, 2024. [Online]. Available: https://www.researchgate.net/publication/380510044_Leveraging_Distributed_Systems_for_Fault-Tolerant_Cloud_Computing_A_Review_of_Strategies_and_Frameworks

[16] N. Gupta and N. H. Vaidya, "Fault-Tolerance in Distributed Optimization: The Case of Redundancy," in Proceedings of the 39th Symposium on Principles of Distributed Computing, in PODC '20. New York, NY, USA: Association for Computing Machinery, Jul. 2020, pp. 365–374. doi: 10.1145/3382734.3405748.

[17] "Distributed Optimization for Big Data Analytics: Beyond Minimization | Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining." Accessed: Sep. 29, 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3580305.3599554

[18] "Approximate Byzantine Fault-Tolerance in Distributed Optimization | Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing." Accessed: Sep. 29, 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3465084.3467902

[19] "Computation-aware distributed optimization over networks | Proceedings of the Workshop on Computation-Aware Algorithmic Design for Cyber-Physical Systems." Accessed: Sep. 29, 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3457335.3461710

[20] P. London, S. Vardi, and A. Wierman, "Logarithmic Communication for Distributed Optimization in Multi-Agent Systems," Proc ACM Meas Anal Comput Syst, vol. 3, no. 3, p. 48:1-48:29, Dec. 2019, doi: 10.1145/3366696.

IEER Institute for Excellence in Education & Research

[21] A. Tahir, K. Hamid, M. Ahmed, and S. Zubair, "Cyber Sovereignty Challenges: A Strategic Framework For National Data Protection Using Blockchain Authentication," Contemporary Journal of Social Science Review, vol. 03, pp. 1316–1327, Aug. 2025

[22] I. Aslam, W. Tariq, T. Waheed, K. Hamid, S. Rizwan, and A. Ahmed, "Enhancing Privacy and Security in Multi-Party Computation for Data Mining in Cryptographically Protected Environments," Annual Methodological Archive Research Review, vol. 3, pp. 510–531, Aug. 2025, doi: 10.63075/fxe5gg65.

[23] I. Manzoor, R. Ghani, and K. Hamid, "Challenges of Data Extraction from Facebook & WhatsApp Applications," Journal of Computing & Biomedical Informatics, vol. 09, Aug. 2025.

[24] H. Bibi, A. Rauf, and K. Hamid, "Comparative Study Of Requirements Prioritization Techniques," Contemporary Journal of Social Science Review, vol. 3, no. 3, pp. 1301–1314, Aug. 2025, doi: 10.63878/cjssr.v3i3.1099.

[25] S. Rafique, R. Mushtaq, L. Anum, K. Hamid, M. W. Iqbal, and S. Ruk, "Analytical Study of OLTP Workload Management in Database Management System," Journal of Computing & Biomedical Informatics, vol. 6, pp. 1–12, Apr. 2024.

IEER
Institute for Excellence in Education & Research