International Journal of Computer Engineering in Research Trends
ISSN: 2349-7084/ https://doi.org/10.22362/ijcert/2025/v12/i7/v12i703
Volume 12, Issue 7, July 2025, Pp.27-35

Research Paper

Comparative Analysis of Efficient Load Balancing Techniques in Split-Join Blockchain Architecture

Vemula Harish[1], R. Sridevi[2]

[1,] *Jawaharlal Nehru Technological University, Hyderabad, India, vemula.harish31@gmail.com*
[2] *JNTUH College of Engineering, Hyderabad, India, sridevirangu@jntuh.ac.in*

**Corresponding Author: vemula.harish31@gmail.com*

Received: 18/02/2025, Revised: 23 /04/2025, Accepted: 23/07/2025 Published: 31/07/2025

Abstract: - The demand for blockchain technology is growing rapidly in several application areas such as finance, healthcare, and cross-border payment systems. Despite its potential in offering security, the technology is facing limitations related to scalability and performance. The Split-Join blockchain architecture is introduced to address the scalability challenges through enabling parallel block processing and using parallel chains called split-chains, each with a dedicated memory pool. While conducting the performance study of the split-join platform, it is observed that there is a bottleneck with load distribution among the memory pools. This issue is addressed in our previous work by introducing the load balancing unit (LBU), which distributes the incoming transactions based on round robin scheduling strategy among the available memory pools. Further to understand the impact of an efficient load balancing strategy within the split-join blockchain framework, this study presents a comparative evaluation of three load balancing strategies: Round Robin, Least-Loaded, and Predictive Assignment. Experiments were conducted using different transaction loads ranging from 100 to 5000, and the results are analysed based on key performance metrics, Transactions Per Second (TPS), and processing time. Results indicate that Predictive Assignment consistently delivers the highest throughput, achieving up to 250 transactions per second, around 13.4 percent improvement over Round Robin strategy with 216 TPS. Least-Loaded Assignment strategy has shown a little improvement up to 226 TPS. These findings signify that the predictive assignment can boost the transaction throughput, especially in a split-join architecture, offering insights into more efficient and scalable network designs.

Keywords- Split-Join Blockchain, Load Balancing, Mempool Assignment, Transaction Throughput, Parallel Chain Architecture

1. Introduction

The traditional blockchain systems process the block of transactions in a sequential manner, which is one of the potential performance bottlenecks. Several alternative solutions are proposed by the researchers, but they have their limitations, such as the complexity of handling transactions is also increased in proportion to the improvement in scalability. To address these issues and offer parallelism in handling the block of transactions, the Split-Join blockchain framework introduced parallel processing of blocks. This framework categorises blocks into two types they are known as split blocks and join blocks. A split block is a block of transactions created by one of the split chains, which are also known as parallel chains having a dedicated memory pool. Each split chain operates in parallel and independently, generating its own set of split blocks. These split chains are synchronised after a certain number of split blocks have been added based on the split-chain-length configuration parameter, and subsequently, the join block is created and appended to the ledger. The join block maintains the reference of all the split-chains available in the network, responsible for validating the transactions included in the parallel chains and keeps the blockchain ledger simple. Ensuring that the ledger is always consistent and up to date. This design lets the blockchain handle many transactions, increasing scalability. This architecture allows multiple blocks to be validated at the same time and keep the ledger consistent by making sure that all the parallel chains are in sync. The research objectives of this study include understanding the performance impact of using different load balancing strategies in the framework environment and identifying the efficient strategy to boost the performance and scalability of the split-join framework.

1.1 Split-Join Architecture Overview

The Split-Join blockchain architecture has been designed to address the scalability issues which arise with conventional blockchain networks. It lets transactions happen simultaneously across multiple chains, which are referred to as "split chains". Each split chain has a separate memory pool that maintains a queue of incoming transactions. This allows processing of multiple blocks in parallel and boost the performance.

Key Components:

- Memory Pool (mempool): A memory pool that temporarily stores unconfirmed transactions before they are added to a block.
- Split Chain: An independent chain associated with a separate memory pool, allows parallel processing.
- Load Balancing Unit (LBU): The component responsible for assigning incoming transactions to the most appropriate memory pool based on the selected load balancing strategy.
- Join Mechanism: A synchronization process that is useful in maintaining a single coherent ledger.

The Split-Join architecture can handle many transactions simultaneously by using split chains and carefully controlling how transactions are assigned to memory pools through the Load Balancing Unit (LBU). Blockchain transactions are secure and can't be tampered but they exhibit high latency and processing times due to global consensus requirement. As the demand for decentralized applications (dApps) are growing, it has become evident that the blockchain environment has some inherent problems. Most traditional blockchain systems use global consensus mechanisms and process blocks in sequential manner, which reduces transaction throughput. Several architectural designs for blockchain systems have come up to address these issues. Some examples are sharding [1], sidechains [2], and multi-chain architectures. The parallel blockchain architecture is one of these new ideas. It spreads the work of processing transactions across several shards that are happening at the same time. In Split Join architecture every validator needs to sign up with one of the memory pools that are available. These pools keep track of and organize incoming transactions. This architecture lets more transactions be processed, and the transactions are spread out across memory pools. Earlier research on scalability [3] has shown that the naive distribution can lead to split chain overloading issues, idle split chains, and longer wait times because of overloading.

This study examines three different ways of distributing transaction loads they are Round Robin, Least-Loaded, and Predictive Assignment. These algorithms are integrated into the split-join framework, the aim is to find the best way to load distribution, which increase transaction throughput and decrease transaction processing time, and keep load balance across split chains. Predictive Assignment is a way to balance load before they happen. It uses historical data on when transactions arrive in and statistical forecasting to guess how much load each memory pool will have in the future. Predictive Assignment is not the same as reactive methods like Least-Loaded. Instead of sending traffic to memory pools that aren't being used as much, it looks ahead to future bottlenecks. This method is useful in places where traffic patterns change frequently because it uses system resources more efficiently. Previous studies on predictive load management [4] and dynamic distribution in systems that change [5] support it. Figure 1 below shows the Split-Join architecture.

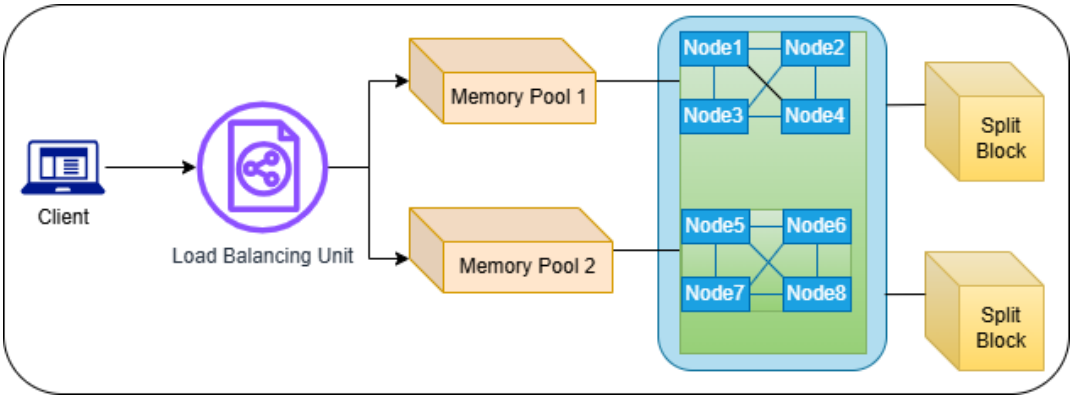

Fig 1. Split-Join Blockchain Architecture.

Figure 1 shows how blocks are generated in a Split-Join blockchain architecture. The first step is creating a genesis block, which then splits up into multiple split chains. Every chain handles its transactions and generates its split blocks. Later, these chains synchronize at join blocks, which ensure a single ledger that is easy to maintain. This makes sure that the state is consistent throughout the system and transactions included in the ledger are final. This architecture utilizes concurrency to boost transaction throughput, and the Load Balancing Unit makes sure that transactions are distributed fairly.

1.2 Load Balancing Unit (LBU) workflow in Split-Join Architecture

Figure 2 below depicts the Load Balancing Unit (LBU) in the Split-Join blockchain architecture. Incoming transactions (T_x) are first routed through the LBU, which applies one of the configured load balancing strategies: Round Robin, Least-Loaded, or Predictive Assignment. Based on these strategies, LBU assign each transaction to the appropriate memory pools (MP1, MP2, etc.).

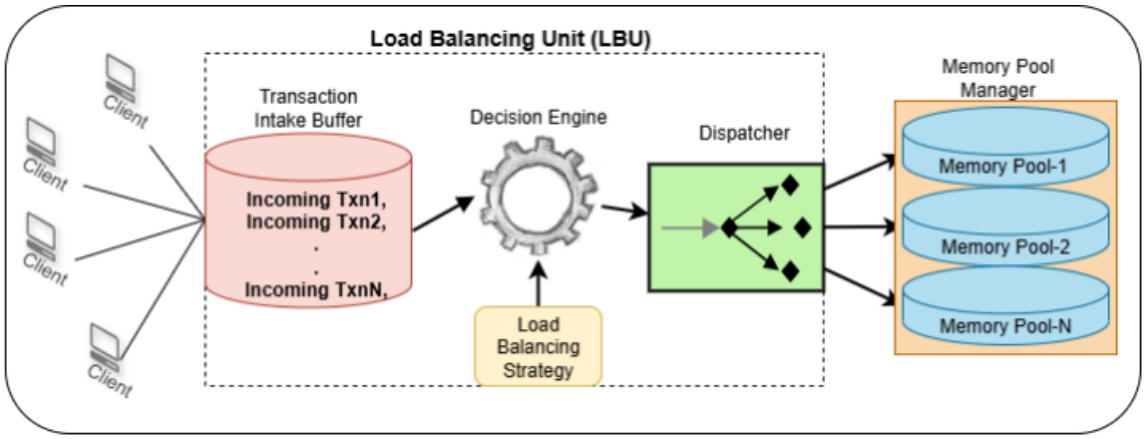

Fig 2. Load Balancing Unit in Split-Join Blockchain.

The LBU consists of three key components:

1. Transaction Intake Buffer (TIB), which temporarily stores incoming transactions for evaluation.
2. Decision Engine (DE) uses logic based on the configured strategy to determine the appropriate destination:
3. Dispatcher forwards transactions to the selected memory pool.

After distribution, the transactions are processed in parallel by their respective split chains.

1.3 Complexity Analysis for LBU Strategies

TABLE 1. Comparison of time complexities and their latency

Strategy	**Decision Logic**	**Complexity**	**Assignment Latency**
Round Robin	Index-based cycling	O (1)	Low
Least-Loaded	Find min queue length	O(n)	Medium
Predictive	Forecast + min selector	O(n+f)	High

Where 'n' is the number of memory pools, and 'f' is the cost of forecasting.

The computational characteristics of the Load Balancing Unit (LBU) significantly influence the system's responsiveness under varying transaction loads. Each strategy integrated into the LBU introduces a distinct computational overhead and latency profile. The Round Robin technique requires no system state evaluation and relies solely on a cycling index counter, resulting in constant time complexity O(1) and minimal assignment latency. In contrast, the Least-Loaded strategy necessitates querying the length of all active memory pool queues to determine the least occupied one. This introduces a linear time complexity of O(n), where 'n' represents the number of memory pools, and incurs moderate latency due to real-time queue polling. Predictive Assignment, the most advanced of the three, leverages historical data and forecasting models, such as moving averages or machine learning techniques, to predict future queue states. Its computational cost is therefore O(n+f), where f represents the time required for forecasting. This additional layer introduces higher latency, particularly in systems with limited computational bandwidth or under bursty workloads.

2. Literature Survey

Active research is currently being conducted by the researchers to find ways to make blockchain networks work better and to boost their performance. There have been many solutions given, but the problem is still open. This study focuses on some of the most important research aspects that are related to the scalability of blockchain systems.

2.1 Static Load Balancing Techniques

Static load balancing strategies, such as Round Robin, are among the simplest and most widely used techniques in distributed systems. In these approaches, transactions are assigned to processing units in a fixed sequence, regardless of current system state. Round Robin, in particular, has been studied in the context of permissioned blockchains and split-memory architectures, where it is valued for its low latency and constant time complexity. However, its inability to respond to dynamic workload variations often leads to load imbalance under bursty traffic conditions.

2.2 Reactive Load Balancing Approaches

Reactive load balancing methods adapt to real-time system conditions by monitoring the state of resources, such as queue lengths or CPU load. The Least-Loaded strategy is a common example, where each incoming transaction is assigned to the memory pool with the shortest queue. This strategy has been explored in sharded blockchain systems, where it has demonstrated performance improvements over static methods. However, reactive approaches incur monitoring overhead and may struggle with rapid load fluctuations without predictive context.

2.3 Predictive and Adaptive Methods

Predictive assignment strategies leverage historical data and forecasting techniques to estimate future load distributions. These methods have been proposed in systems such as TxAllo and dynamic DAG blockchains. By anticipating congestion before it occurs, predictive models can significantly reduce latency and queue variance. The trade-off, however, lies in computational cost, particularly in

environments with limited resources or unpredictable transaction patterns.

2.4 Transaction Load Distribution in Sharded Systems

There are issues with load balancing in blockchain systems, like load imbalance and resource bottlenecks, a number of studies conducted to address these issues by the researchers. Huang et al. [6] conducted a research study on how stable queues are in permissioned blockchains that use sharding. They propose a Lyapunov-based elastic resource allocation model that changes how transactions are assigned depending on how busy the system is at the time. This is similar to the Least-Loaded assignment strategy, which sends transactions to the memory pool that is the least busy. Zhang et al. [7] suggest TxAllo, a dynamic, predictive transaction assignment method that uses graph-based community detection to assign transactions in a way that reduces cross-shard dependencies. Predictive Assignment is based on the model that uses transaction histories and expected workload trends to figure out how to divide up work. Toulouse et al. [8] talk about a consensus-based load-balancing algorithm that lets shards work together to change transaction loads based on how activity has been going recently. Their distributed model makes it easier to adjust to changes in traffic patterns and backs up the theoretical bases of both the Least-Loaded and Predictive approaches.

The Split-Join model is based on a parallel architecture that has separate memory pools, associated with split chains, in contrast to traditional architectures that use shared memory pools. Gai et al. [9] study has about Stratus, a shared memory pool system that separates the process of submitting transactions from the process of reaching consensus. This makes sure that all validators are looking at the same transactions at the same time. This method doesn't use Split-Join, but it does help us learn how to sync memory pools and make transactions visible. This is important for knowing the benefits and drawbacks of using isolated and global memory pool structures.

TABLE 2. Key Contributions of Various Load Balancing Strategies

Study	Domain	Strategy	Key Contribution
Huang et al.	PBFT Sharded	Least-Loaded (Elastic)	Maintains queue stability under bursts
Zhang et al.	Ethereum	Predictive Clustering	Reduces cross-shard transactions
Li et al. [10]	Sharding	Account-Migration-Based Load Balancing	Balances workload by dynamically migrating high-activity accounts across shards to minimize transaction latency and improve throughput.
Prior Work [11]	Split-Join	Round Robin	Parallel validation with fixed routing

Table 2 shows that these studies have laid a solid foundation for dynamic load balancing in blockchain systems. There is still a considerable research gap in comparing load balancing strategies, such as Round Robin, Least-Loaded, and Predictive Assignment etc. This study tries to fill the research gap by using and testing these three distribution strategies in a Split-Join environment. It focuses on how they affect transaction throughput, latency, and memory pool utilization. It gives useful insights on how to make transaction distribution better in parallel blockchain architectures. Researchers have been working on different approaches to speed up, make more reliable, and make blockchain transactions more efficient. Bitcoin and Ethereum are examples of traditional blockchain systems that have sequential bottlenecks because they are monolithic and need global consensus. Sharding [12] is a well-known way to divide the blockchain network into smaller parts called shards. Each shard is handled by a different group of nodes. This parallelism lets transactions happen on their own, but it makes it harder for shards to talk to each other and agree on the common state of transactions.

IOTA and Nano are examples of Directed Acyclic Graph (DAG)-based blockchains [13]. They use a graph-based structure instead of a linear chain to let transactions confirm each other. This model does away with miners and allows for high-throughput, feeless operations. However, it can have problems with finality and security when there aren't many transactions. Off-chain scaling solutions [14], such as payment channels and sidechains, try to keep the main chain decentralized and secure while moving transaction processing to a different chain. These mechanisms help ease congestion, but they may come with trade-offs in terms of trust assumptions and locking up liquidity.

Multi-chain environments [15], such as Polkadot [16] and Cosmos [17], connect separate blockchains so that they can talk to each other and grow. These systems are based on the idea of modular interoperability and shared security layers, which makes them a strong alternative to application-specific chains. Split-Join architectures have become more popular in this range because they can process transactions at the same time by using distributed memory pools and running chains on their own. The first work in this area set up the basic framework for sending transactions to multiple memory pools and keeping their outputs in sync using a join protocol to keep things consistent. Building on these ideas, Zhang et al. came up with TxAllo, a method for predicting transaction assignments that cuts down on dependencies between chains and balances workload by looking at past traffic patterns. Toulouse et al. also made a consensus-aware dispatcher that changes transaction loads based on how well the chain is performing. These studies show how important adaptive load balancing strategies are in real life and strongly support this approach. At the same time, research on sharded blockchains and hybrid chains has been focused on finding a balance between the workloads of validators. Toulouse et al.'s work, which used a consensus-adjusted dispatcher across independent chains, can also be used to compare different load balancing methods in distributed systems. Their method is very similar to the Least-Loaded assignment policy in this study. Also, traffic routing [18], cloud computing [19], and blockchain simulation frameworks [20] are using predictive methods like time-series forecasting and machine learning algorithms more and more.

Wang et al. [21] also made important contributions to this field by looking into adaptive sharding protocols that change shard boundaries based on validator activity and workload skew. Their framework used a cost-benefit model

to figure out when to start re-sharding, which helped them understand how to realign dynamic loads. Kim et al. [22] suggested a way to spread blockchain transactions using localized memory pool prioritization and a distributed queuing system. This method sped up the spread of data and increased the amount of data that could be sent over a single chain in multi-chain networks. Harish et al. (2023) [23] conducted a comparative evaluation of public and private blockchain systems, focusing on key performance metrics such as transaction throughput, scalability, and consensus efficiency. Their study highlights the trade-offs between openness and control, where public blockchains offer higher decentralization but suffer from lower performance, while private blockchains achieve better speed and efficiency due to restricted participation. The authors also emphasize the importance of consensus mechanisms in influencing performance outcomes and suggest future research should focus on optimizing consensus protocols for enterprise use cases and large-scale deployments.

These studies show that there are many different ways to distribute transactions and stress how important it is to change strategies based on how the network changes. These new studies build on older ones and show how important adaptive load balancing is. Most of these methods, on the other hand, are either specific to one application. The literature survey gives a full picture of the architectural and algorithmic approaches that have been tried to solve the scalability and load distribution problems in blockchain systems. It starts with basic methods like sharding, DAG-based structures, off-chain scaling, and multi-chain ecosystems. These set the stage for parallel transaction execution. the study stresses how important it is to assign transactions using techniques such as Round Robin, Least-Loaded, and Predictive Assignment etc. Huang et al., Zhang et al., and Toulouse et al. have all done research that adds useful models that are similar in concept to the methods looked at in this study. These include elastic resource allocation, predictive clustering, and consensus-aware load balancing. Gai et al.'s research on shared memory pools gives us more information about the trade-offs involved in coordinating. The chapter goes on to talk about more recent ideas like elastic sharding, distributed queueing, hybrid assignment methods, and load-aware block assembly. Each of these ideas helps us learn more about how to balance loads in distributed systems.

3. Research Methodology and Results

This study compares the performance of three load balancing techniques, Round Robin, Least-Loaded, and Predictive Assignment, in the Split-Join blockchain architecture. The experiments are conducted to see how each strategy affected the system performance, like throughput (TPS), average transaction latency, and queue balance. The Load Balancing Unit (LBU) sends transactions to the available memory pools that feed transactions into parallel split chains. Integrated each load balancing method to the LBU of the split-join system and conducted experiments with five different transaction volumes, i.e., 50, 100, 500, 1000, and 5000. The goal is to see how the system handled more load and whether each strategy kept balance and performance across the split chains. A separate memory pool is assigned to each split chain, and a central Load Balancing Unit (LBU) distributes the transactions to the memory pools based on the integrated balancing technique. Transaction volumes are generated in an increasing fashion, and the results, especially block processing time, transaction throughput. To make sure the results were consistent, each experiment was run in a controlled environment for multiple rounds, for each configuration. The final analysis used the average results. Logging modules recorded performance data like throughput (TPS), latency (ms), and transaction queue variance across split chains.

3.1 Simulation Environment and Configuration

The Split-Join blockchain framework used in this study was implemented using the Go programming language (Golang), chosen for its concurrency support and execution speed. No external simulators such as SimBlock or BlockSim were employed; instead, a custom lightweight simulation environment was built from scratch to model the Load Balancing Unit (LBU), mempools, split chains, and join logic.

Experiments were executed on a system configured with an Intel Core i5 processor, 8 GB RAM, and a 250 GB SSD, running a standard Linux-based operating system. The testing methodology involved injecting transactions into the system in batches of varying volumes (e.g., 100, 500, 1000, 5000) and recording the total processing time in milliseconds.

Transactions Per Second (TPS) is calculated using the formula:

$$TPS = \frac{Number\ of\ Transactions}{Processing\ Time\ (in\ seconds)} \quad (1)$$

To ensure consistency and repeatability, each experiment was conducted using a fixed input. Each test case was executed 5 times, and average metrics were reported to smooth out system-level variability.

TABLE 3. Key Contributions of Various Load Balancing Strategies

Parameter	**Description / Value**
Programming Language	Go (Golang)
Simulator Type	Custom-built Split-Join blockchain simulator
Number of Mempools	2 (one per split chain)
Number of Split Chains	2 (parallel processing units)
Load Balancing Strategies	Round Robin, Least-Loaded, Predictive Assignment
Mempool Size	Dynamically adjusted (not capped during simulation)
Transaction Volumes Tested	50, 100, 500, 1000, 5000
Repetitions per Scenario	5 runs per configuration
System Specifications	Intel Core i5, 8 GB RAM, 250 GB SSD, Ubuntu 20.04
Timing Metric	Total processing time (ms) and TPS calculated

Randomization	Fixed structure; deterministic sequence; no RNG used

3.2 Load Balancing through the Round Robin technique

The Round Robin method is a load-balancing strategy that is both static and deterministic. It evenly spreads transactions across the available memory pools by giving each new transaction to the next memory pool in a cycle, no matter what the current queue state is. This method is computationally lightweight and implementation-friendly.

Algorithm:

Initialize index = 0

For each incoming transaction Tx:

Assign Tx to mempool[index]

index = (index + 1) % total_mempools

Formula:

$$Memory\ Pool\ Index = (\ Current\ Index + 1) mox\ N \quad (2)$$

The worst-case time complexity of the Round Robin algorithm is O(1), as it simply cycles through available mempools in a fixed order without inspecting their current state. This approach ensures constant-time transaction assignment and incurs minimal computational overhead, resulting in low latency. However, its lack of responsiveness to real-time load conditions can lead to imbalance under bursty or uneven transaction flows, results are shown in figure 3 below.

TABLE 4. Transaction Throughput using Round Robin based Load Balancing Technique

Transaction Volume	Avg Processing Time (ms)	Avg Transaction Throughput (tx/sec)
50	392.802	127.291
100	515.687	193.916
500	2312.241	216.240
1000	4843.568	206.459
5000	24876.161	200.996

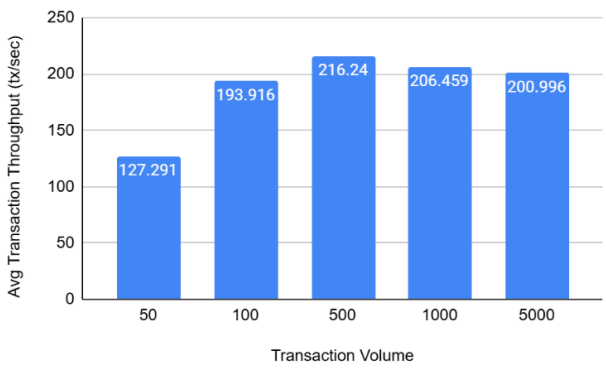

Fig 3. Transaction Throughput using Round Robin Scheduling.

These results show that Round Robin works well when there is little to moderate traffic. It gives a fair distribution of work and a predictable processing time. However, its throughput doesn't go up in direct proportion to the number of transactions because it doesn't respond to changes in individual memory pool states. The performance drop isn't too bad, but the method doesn't work as well when there are a lot of people using it because dynamic load shifts become more obvious. This restriction makes Round Robin a good choice only for environments with light traffic or traffic that is evenly distributed. As the number of transactions goes up, the throughput goes down a little and the processing time goes up, which shows how limited a static allocation model works best when the traffic is evenly spread out and the system doesn't need to be very responsive. With low to moderate transaction loads (50, 100, and 500), Round Robin kept the queue sizes fair and the throughput good. But when the volume was higher (1000 and 5000), it didn't take temporary load spikes into account, which led to longer wait times and sometimes full queues.

3.3 Load Balancing through the Least-Loaded Assignment Technique

Least-Loaded is a dynamic strategy that assigns each incoming transaction to the memory pool with the smallest current queue length. It requires constant monitoring of queue states and responds to real-time load conditions.

Algorithm:

For each incoming transaction Tx:

Evaluate queue length for all memory pools

Assign Tx to the memory pool with the minimum queue size

Formula:

$$Memory\ Pool\ Index = Argmin\ (Q1, Q2, \dots, Qn) \quad (3)$$

The Least-Loaded algorithm exhibits a worst-case time complexity of O(n), where n denotes the number of memory pools. For each incoming transaction, the algorithm performs a full scan of all memory pool queues to identify the one with the least load. While this method adapts to dynamic traffic conditions and improves distribution accuracy, it introduces moderate latency due to the need for real-time state inspection, figure 4 shows the results using least loaded strategy.

TABLE 5. Transaction Throughput using Least-Loaded based Load Balancing Technique

Transaction Volume	Avg Processing Time (ms)	Avg Transaction Throughput (tx/sec)
50	361.254	138.407
100	475.288	210.399
500	2205.853	226.670
1000	4471.356	223.646
5000	23052.671	216.895

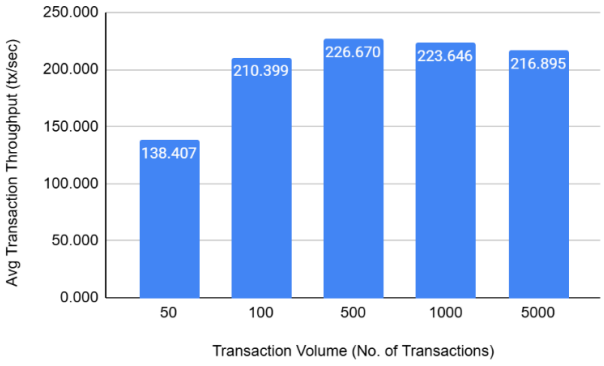

Fig 4. Transaction Throughput using Least-Loaded Scheduling

The results in Table 5 show that the Least-Loaded strategy works well at adjusting to traffic conditions in real time. This method is different from Round Robin because it actively responds to congestion by sending transactions to the memory pool with the shortest queue. This reactive behaviour makes processing times more consistent and throughput noticeably better as the system grows. The Least-Loaded method is a good balance of performance and practicality for all transaction volumes, even though it is a little more complicated because it needs to keep an eye on the queue all the time. This strategy worked well for all transaction amounts, including 50, 100, 500, 1000, and 5000. It worked well to smooth out bursts by sending traffic to memory pools that weren't being used much. It cost a little more to run because it had to be monitored, but it had better throughput and lower processing time than the Round Robin based model.

3.4 Load Balancing through the Least-Loaded Assignment Technique

Predictive Assignment is a way to guess what the future state of a queue will be by looking at past data and using forecasting methods. It gives out transactions based on the expected load for the next few time steps instead of just the current state.

Algorithm:

Maintain time-series data for each memory pool

For each transaction Tx:

Predict near-future load using a moving average or regression

Assign Tx to the memory pool with the lowest predicted load

Formula:

$$L_i = \left(\frac{1}{W}\right) * \Sigma\, Tx_{it}, for\ t\ in\ [1, W] (W = window\ size) \quad (4)$$

The Predictive Assignment algorithm has a worst-case time complexity of O(n+f), where n is the number of memory pools and f represents the computational cost associated with load forecasting. By leveraging historical data and predictive models, such as moving averages or time-series forecasts, the algorithm anticipates future queue states and allocates transactions accordingly. Although this approach yields the most balanced distribution and highest throughput, it incurs the highest latency due to its reliance on forecast computations, the following figure 5 depicts the results using predictive assignment algorithm.

TABLE 6. Transaction Throughput using Predictive Assignment-based Load Balancing Technique

Transaction Volume	Avg Processing Time (ms)	Avg Throughput (tx/sec)
50	334.751	149.365
100	439.382	227.592
500	1995.853	250.519
1000	4291.356	233.027
5000	21925.671	228.043

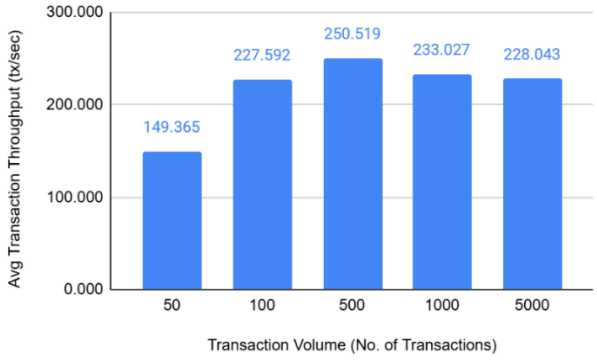

Fig 5. Transaction Throughput using Predictive Assignment Scheduling

Table 6 shows that Predictive Assignment works best in places where traffic patterns change all the time. Its forecasting system accurately predicts and makes up for load imbalances that will happen in the future, which leads to higher throughput and shorter processing times at all levels of transactions. The gain is most clear when there are a lot of heavy loads, because its proactive routing cuts down on latency spikes that happen with other methods. This makes Predictive Assignment a great choice for high-performance blockchain systems that need to process transactions quickly and reliably.

Predictive Assignment could achieve the best results, especially when there were many transactions ranging from 1000 and 5000. It actively sent transactions away from places where they were likely to get stuck, keeping the queue variance low, the throughput high, and the latency low. The following figure 6 shows how the transaction throughput varies for the different load balancing strategies as the number of transactions goes up.

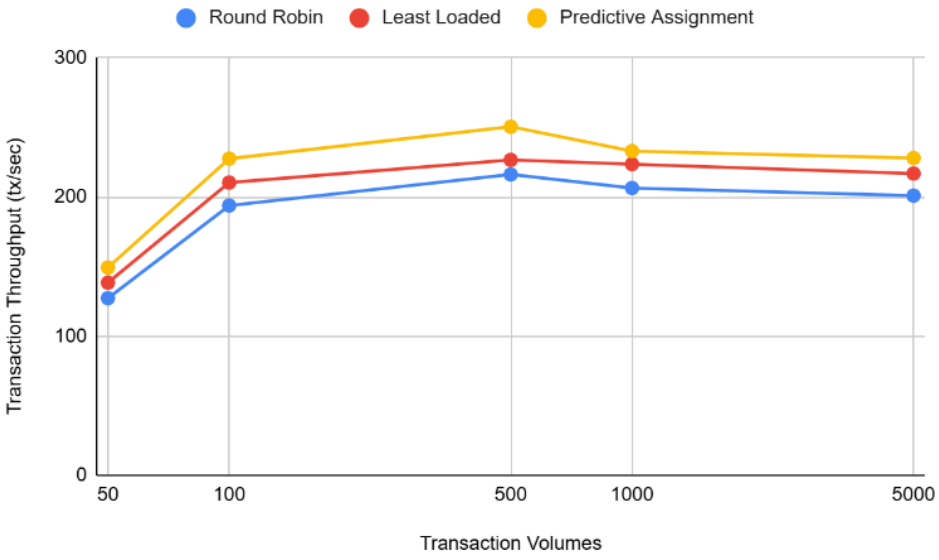

Fig 6. Transaction Throughput Comparison Across Load Balancing Techniques

Round Robin shows relatively stable but lower throughput, while Least-Loaded always gets better as traffic goes up. Predictive Assignment is better than both, especially when there are a lot of them, because it predicts loads ahead of time. The graph clearly shows that the Predictive method is better at scaling and responding, which confirms that it is better for blockchain environments with a heavy load.

3.5 Summary of Results

The results of the experiments on the three load balancing strategies signify about how well they work with different types of workloads. Round Robin is a simple and stateless way to distribute things, and as the load increased, it behaved in a predictable but not ideal way. It couldn't handle the congestion in the memory pool, which caused queues to build up and transaction throughput to drop slightly. Least-Loaded Assignment made this a lot better by changing the way transactions were routed based on the length of the queues in real time. Because it was responsive, it was able to keep processing times low and throughput high, even when transaction volumes went over 500. Predictive Assignment turned out to be the best method, beating the others in both TPS and latency. It balanced memory pool queues with more foresight by looking at past trends and making predictions about how the load would be distributed in the future. This reduced bottlenecks and made sure that split chains were used more evenly. The results support the idea that dynamic and predictive strategies are necessary for getting the most out of scalability in asynchronous blockchain architectures.

6. Conclusion

This study presented a comprehensive evaluation of three transaction assignment strategies: Round Robin, Least-Loaded, and Predictive Assignment, within the context of a Split-Join blockchain architecture. Although similar techniques have been analyzed in sharded or generic blockchain systems, the novelty of this work lies in performing a comparative analysis specifically within a Split-Join framework that utilizes parallel mempools and concurrently operating split chains. The findings indicate that Predictive Assignment achieves the highest throughput and responsiveness under varying transaction volumes, whereas Round Robin offers low computational overhead and latency. To enhance the decision-making capability of the Load Balancing Unit (LBU), future research will explore the integration of LSTM-based predictive models and reinforcement learning techniques. These models are expected to enable adaptive and context-aware transaction routing based on historical and real-time system behavior. Further extensions to the Split-Join framework will include support for cross-chain transaction execution, allowing interaction between parallel processing units. Additionally, the development of a fault-tolerant join block design will be investigated to ensure system continuity in the presence of delays or failures in individual split chains. These proposed enhancements aim to advance the adaptability, reliability, and scalability of transaction processing in Split-Join blockchain environments.

Author Contributions: Vemula Harish, contributed to the conceptualization, methodology, and data analysis of the study. R. Sridevi supervised the research and provided critical revisions to the manuscript.

Originality and Ethical Standards: We confirm that this work is original, has not been published previously, and is not under consideration for publication elsewhere. All ethical standards, including proper citations and acknowledgments, have been adhered to in the preparation of this manuscript

Data availability: Data available upon request.

Conflict of Interest: There is no conflict of Interest.

Ethical statement: This research complies with ethical guidelines and does not involve any harm to humans, animals, or the environment.

Funding: The research received no external funding.

Similarity checked: Yes.

References

[1] Y. Liu, J. Li, R. Xie, J. Yu, and J. Lin, "Building blocks of sharding blockchain systems: Concepts, approaches, and open problems," Computer Science Review, vol. 46, p. 100513, 2022. doi: 10.1016/j.cosrev.2022.100513.

[2] W. Li, Z. Li, H. Dai, F. Wang, and X. Luo, "Towards Blockchain Interoperability: A Comprehensive Survey on Cross-Chain Solutions," Blockchain: Research and Applications, p. 100286, 2025. doi: 10.1016/j.bcra.2024.100286.

[3] M. Bez, G. Fornari, and T. Vardanega, "The scalability challenge of ethereum: An initial quantitative analysis," in 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), 2019, pp. 122–128. doi: 10.1109/SOSE.2019.00028.

[4] M. Jodayree, M. Abaza, and Q. Tan, "A predictive workload balancing algorithm in cloud services," Procedia Computer Science, vol. 159, pp. 902–912, 2019. doi: 10.1016/j.procs.2019.09.234.

[5] P. A. D. S. N. Wijesekara, "Load Balancing in Blockchain Networks: A Survey," International Journal of Electrical and Electronic Engineering & Telecommunications, vol. 13, no. 3, 2024. doi: 10.18178/ijeetc.13.3.225-233.

[6] R. Adhikari, C. Busch, and D. R. Kowalski, "Stable blockchain sharding under adversarial transaction generation," in Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures, 2024, pp. 325–336. doi: 10.1145/3617653.3656482.

[7] Y. Zhang, S. Pan, and J. Yu, "Txallo: Dynamic transaction allocation in sharded blockchain systems," in 2023 IEEE 39th International Conference on Data Engineering (ICDE), 2023, pp. 1389–1401. doi: 10.1109/ICDE55515.2023.00118.

[8] M. Toulouse, H.-K. Dai, and Q. L. Nguyen, "A consensus-based load-balancing algorithm for sharded blockchains," in Future Data and Security Engineering: 8th International Conference, FDSE 2021, Virtual Event, November 24–26, 2021, Proceedings 8. Cham: Springer International Publishing, 2021, pp. 254–270. doi: 10.1007/978-3-030-93247-3_16.

[9] F. Gai, W. Zheng, X. Liu, L. Su, and J. Wang, "Scaling blockchain consensus via a robust shared mempool," in 2023 IEEE 39th International Conference on Data Engineering (ICDE), 2023, pp. 1377–1388. doi: 10.1109/ICDE55515.2023.00117.

[10] M. Li, W. Wang, and J. Zhang, "LB-Chain: Load-balanced and low-latency blockchain sharding via account migration," IEEE Transactions on Parallel and Distributed Systems, vol. 34, no. 10, pp. 2797–2810, 2023. doi: 10.1109/TPDS.2022.3195045.

[11] V. Harish and R. Sridevi, "Enhancing Split-Join Blockchain Performance through Load Balancing," SSRG International Journal of Electrical and Electronics Engineering, vol. 11, no. 7, pp. 124–133, 2024. doi: 10.14445/23488379/IJEEE-V11I7P110.

[12] G. Yu, Y. Huang, Y. Guo, S. Wang, Z. Zheng, and J. Wu, "Survey: Sharding in blockchains," IEEE Access, vol. 8, pp. 14155–14181, 2020. doi: 10.1109/ACCESS.2020.2966128.

[13] A. Mahdi and F. Rabee, "DAGchains: An Improved Blockchain Structure Based on Directed Acyclic Graph Construction and Distributed Mining," Mesopotamian Journal of CyberSecurity, vol. 5, no. 2, pp. 375–394, 2025. doi: 10.34299/csmjcs.5.2.53.

[14] T. Cai, S. Huang, Y. Wang, T. Yu, and X. Chen, "On-chain and off-chain scalability techniques," in Blockchain Scalability, Singapore: Springer Nature Singapore, 2023, pp. 81–96. doi: 10.1007/978-981-99-3009-0_6.

[15] Z. Wu, Y. Wang, and L. Wang, "GAM: A scalable and efficient multi-chain data sharing scheme," Information Processing & Management, vol. 62, no. 3, p. 104004, 2025. doi: 10.1016/j.ipm.2024.104004.

[16] D. Smuseva, U. Mononen, P. Kuvaja, and M. Matinlassi, "Under the Space Threat: Quantitative Analysis of Cosmos Blockchain," in European Workshop on Performance Engineering. Cham: Springer Nature Switzerland, 2024, pp. 54–66. doi: 10.1007/978-3-031-47451-1_5.

[17] H. Abbas, M. Caprolu, and R. Di Pietro, "Analysis of polkadot: Architecture, internals, and contradictions," in 2022 IEEE International Conference on Blockchain (Blockchain), 2022, pp. 106–113. doi: 10.1109/Blockchain55522.2022.00025.

[18] T. Liu, S. Zhou, F. Zhao, J. Liu, J. Luo, H. Xu, C. Xu, and X. Meng, "Efficient algorithms for storage load balancing of outsourced data in blockchain network," The Computer Journal, vol. 65, no. 6, pp. 1512–1526, 2022. doi: 10.1093/comjnl/bxac033.

[19] K. Gai, M. Qiu, L. Tao, and H. Zhao, "Blockchain meets cloud computing: A survey," IEEE Communications Surveys & Tutorials, vol. 22, no. 3, pp. 2009–2030, 2020. doi: 10.1109/COMST.2020.2976257.

[20] M. Alharby and A. Van Moorsel, "Blocksim: a simulation framework for blockchain systems," ACM SIGMETRICS Performance Evaluation Review, vol. 46, no. 3, pp. 135–138, 2019. doi: 10.1145/3308897.3308955.

[21] G. Wang, A. Ponomarev, A. Barnes, G. Ren, and J. Xie, "SoK: Sharding on blockchain," in Proceedings of the 1st ACM Conference on Advances in Financial Technologies, 2019, pp. 41–61. doi: 10.1145/3318041.3355466.

[22] D. Kim and S. Park, "Blockchain-based Scalable and Reliable Social Networking Platform," IEEE Access, vol. 12, 2024. doi: 10.1109/ACCESS.2024.3341193.

[23] V. Harish and R. Sridevi, "Performance Analysis of Public and Private Blockchains and Future Research Directions," in International Conference on Network Security and Blockchain Technology. Singapore: Springer Nature Singapore, 2023, pp. 243–261. doi: 10.1007/978-981-99-6086-8_21.