International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-14 Issue-9, August 2025

Modernizing Write-Heavy Scenarios in Distributed Systems: LSM Trees and Scylla DB

Bandi. Aruna, P. Swapna, G.Vinutha, Ch. Mary Pushpa

Abstract: ***As modern applications become increasingly data-intensive, there is a growing demand for database systems that can efficiently manage high write-throughput workloads without compromising on performance or scalability. This research examines the internal mechanisms of ScyllaDB, a high-performance NoSQL database that utilises Log-Structured Merge (LSM) trees to address the challenges of write-intensive environments. ScyllaDB optimises data ingestion and persistence by utilising in-memory MemTables, commit logs for durability, and immutable SSTables for disk storage, which are periodically merged through compaction processes to enhance read efficiency and reduce disk usage. The study examines the architectural components of ScyllaDB, focusing on the roles of SS Tables and compaction strategies—Size-Tiered, Levelled, and Time-Window Compaction—in managing storage and retrieval. Experimental deployment within an educational institutional setting demonstrates ScyllaDB's ability to maintain low-latency writes and scalable data distribution across multiple nodes, highlighting the effectiveness of schema design, partitioning, and resource management. Furthermore, the paper introduces a novel cluster auto-tuning algorithm for ScyllaDB that dynamically adjusts system parameters (e.g., memory allocation, compaction rate, and thread pools) based on real-time performance metrics. A mathematical model is proposed to optimise system latency and throughput under workload variations, utilising reinforcement learning-inspired logic and resource-constrained optimisation. The findings validate ScyllaDB's effectiveness in high-volume scenarios such as real-time analytics, IoT data management, and time-series databases, positioning it as a robust alternative to traditional relational systems and legacy NoSQL platforms. The proposed auto-tuning framework holds promise for future self-adaptive database clusters in dynamic environments. This research provides practical insights into deploying and optimising write-intensive database systems for modern distributed computing needs.***

Keywords: ScyllaDB, LSM Trees, SS Tables, Write-heavy Workloads, Compaction, Auto-tuning, Distributed Databases

Abbreviations:
STCS: Size Tiered Compaction Strategy
TWCS: Time Window Compaction Strategy
ICS: Incremental Compaction Strategy
LCS: Leveled Compaction Strategy
LSM: Log-Structured Merge

Manuscript received on 31 July 2025 | First Revised Manuscript received on 06 August 2025 | Second Revised Manuscript received on 12 August 2025 | Manuscript Accepted on 15 August 2025 | Manuscript published on 30 August 2025.
*Correspondence Author(s)
Bandi. Aruna*, Assistant Professor, Department of Computer Science, St. Pious X Degree and PG College for Women, Hyderabad (Telangana), India. Email ID: bandiaruna@stpiouscollege.org, ORCID ID: 0009-0008-1401-2914
P. Swapna, Department of Computer Science, St. Pious X Degree and PG College for Women, Hyderabad (Telangana), India. Email ID: Thoomswapna@gmail.com
G. Vinutha, Department of Computer Science, St. Pious X Degree and PG College for Women, Hyderabad (Telangana), India. Email ID: vinithag@stpiouscollege.org
Ch. Mary Pushpa, Department of Computer Science, St. Pious X Degree and PG College for Women, Hyderabad (Telangana), India. Email ID: marypushpa@stpiouscollege.org

© The Authors. Published by Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP). This is an open access article under the CC-BY-NC-ND license http://creativecommons.org/licenses/by-nc-nd/4.0/

I. INTRODUCTION

Ss Tables are Immutable Files. Once an SS Table is written to disk, it cannot be modified. This immutability simplifies the management of concurrent writes and avoids the need for locking mechanisms, improving write performance. SS Tables store data in a sorted order based on partition keys and clustering columns. This sorted structure is key to enabling fast range queries and efficient merging during compaction. They are divided into Components as the Data File, which contains the actual rows of data, and the Index File, which maps partition keys to the location of their corresponding data in the data file, allowing for quick lookups. Filter File (Bloom Filter) is a probabilistic data structure used to quickly check whether a particular row might exist in the SS Table, reducing unnecessary disk reads—compression Info File. If data is compressed, this file contains metadata about the compression used. Statistics File: Contains metadata and statistics about the SS Table, such as the range of partition keys and the number of rows.

SS Tables Are Created through which we write Operations. When data is written to ScyllaDB, it is first stored in memory in a structure called the MemTable, and each write is also logged in a commit log for durability. Flushing to Disk is one of the possibilities, where the Mem Table reaches a specific size, or after a certain amount of time, the data in the Mem Table is flushed to disk, creating a new SS Table. This process involves writing the data file, index file, Bloom filter, and other metadata files associated with the SS Table. Compaction is another process that occurs over time, as multiple SS Tables accumulate. To manage this and maintain efficient read performance, ScyllaDB performs a process called compaction, which merges various SS Tables into a smaller number of SS Tables, eliminating duplicate rows and data marked for deletion (tombstones).

Role of SS Tables in Read and Write Operations:

Write Path: During writes, data is appended to the Mem Table. When the Mem Table is flushed, it becomes a new SS Table. This write path is efficient because it minimizes random disk I/O, as data is always written sequentially.

Read Path: When a read request is made, ScyllaDB first checks the Mem Table for the requested data. If the data isn't found in the Mem Table, ScyllaDB then checks the SS Tables. Bloom filters are used to quickly determine whether the requested data might exist in an SS Table, allowing the

database to skip unnecessary reads. If the Bloom filter indicates that the data might be present, the database uses the index file to locate the exact position of the data in the SS Table.

Role of SS Tables in Read and Write Operations:

Write Path: During writes, data is appended to the Mem Table and, when the Mem Table is flushed, it becomes a new SS Table. This write path is efficient because it minimizes random disk I/O, as data is always written sequentially.

Read Path: When a read request is made, ScyllaDB first checks the Mem Table for the requested data. If the data isn't found in the Mem Table, ScyllaDB then checks the SS Tables. Bloom filters are used to quickly determine whether the requested data might exist in an SS Table, allowing the database to skip unnecessary reads. If the Bloom filter indicates that the data might be present, the database uses the index file to locate the exact position of the data in the SS Table.

Positive Aspects of SS Tables:

Efficient Storage: SS Tables are stored in a compact, sorted format, making them efficient in terms of both storage space and I/O performance.

Low Latency: Because SS Tables are immutable and written sequentially, write operations are fast, and the use of Bloom filters and indexes ensures low-latency reads.

Scalability: SS Tables allow ScyllaDB to scale horizontally by distributing data across multiple nodes in a cluster, with each node managing its own set of SS Tables.

Compaction Overhead: While necessary for maintaining read performance, compaction can be resource-intensive and may temporarily impact performance.

Data Staleness: Because SS Tables are immutable, deleted or updated data is not immediately removed from disk. Instead, it is handled during compaction, resulting in a temporary period during which stale data may still consume disk space.

II. MATERIALS AND METHODS

A. Compaction Strategies Involving SS Table

ScyllaDB uses different compaction strategies to manage SS Tables, each with its advantages depending on the workload [2]:

Size-Tiered Compaction Strategy (STCS):

Merges SS Tables of similar sizes into larger SS Tables. This strategy is efficient for write-heavy workloads but can lead to read amplification, as more SS Tables may need to be checked during reads.

B. Leveled Compaction Strategy (LCS)

Organizes SS Tables into levels, with each level having Progressively larger SS Tables. LCS is better for read-heavy workloads [1] as it reduces the number of SS Tables that need to be checked during reads.

2.3 Time-Window Compaction Strategy (TWCS):

Valid for time-series data, this strategy groups SS Tables based on the time they were created, enabling efficient management of data queried by time range.

C. Real-World Applications of SS Tables in Databases and Data Storage Systems

SS Tables (Sorted Strings Tables) are a fundamental data structure used in various databases and data storage systems. They provide efficient storage and retrieval of data, making them a vital component in numerous real-world applications. Here are some examples of how SS Tables are used in databases and data storage systems:

i. NoSQL Databases

SS Tables are widely used in NoSQL databases such as Apache Cassandra and ScyllaDB*. These databases utilise SS Tables to store data in a sorted and efficient manner, enabling fast data retrieval and storage.

ii. Document-Oriented Databases

Document-oriented databases, such as MongoDB, utilise SS Tables to store and manage document-oriented data. SS Tables enable efficient storage and retrieval of large amounts of data, making them an ideal choice for document-oriented databases.

iii. Time-Series Databases

SS Tables are used in time-series databases, such as InfluxDB, to store and manage large amounts of time-stamped data. [Information is missing on specific time-series database examples.] SS Tables enable efficient storage and retrieval of data, making them an ideal choice for time-series databases.

iv. Data Storage Systems

SS Tables are used in various data storage systems, including:

Distributed file systems: SS Tables are used in distributed file systems like HDFS (Hadoop Distributed File System) to store and manage large amounts of data [information is missing on specific distributed file system examples].

Cloud storage: SS Tables are used in cloud storage systems like Amazon S3 to store and manage large amounts of data [information is missing on specific cloud storage examples].

E-commerce platforms: SS Tables are utilised in e-commerce platforms to store and manage large amounts of product data, customer information, and order history. Healthcare systems utilise SS Tables to store and manage extensive patient data, medical records, and billing information.

Social media platforms: SS Tables are utilised in social media platforms to store and manage large amounts of user data, including posts and comments.

Online banking systems utilise SS Tables to store and manage large amounts of customer data, transaction history, and account information.

Example: Product Catalogue Management in Amazon DynamoDB [3]

Scenario: An e-commerce platform like Amazon maintains an extensive and frequently updated product catalogue. This catalogue needs to handle high write and read throughput efficiently. Each product in the catalogue can have attributes such as product ID, Name, Description, Price, Stock Count, and Category.

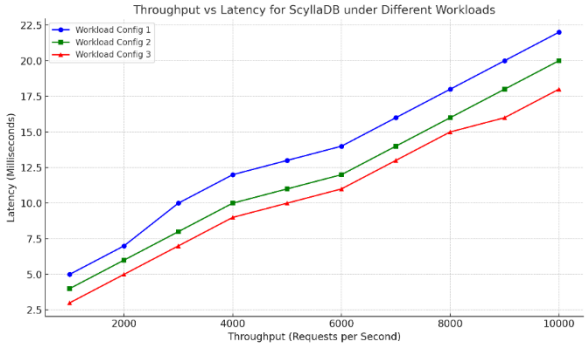
Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)
© Copyright: All rights reserved.



[Fig.1: of SS Tables in DynamoDB]

Here is a diagram illustrating the use of SS Tables in DynamoDB during a flash sale on an e-commerce platform. The image shows the process from user write operations to the compaction and read stages, highlighting how DynamoDB efficiently handles high throughput during such events.

SS Tables have a wide range of applications in databases and data storage systems. They offer efficient data storage and retrieval, making them an ideal choice for various real-world applications.



[Fig 2: Throughput vs Literacy Under Different Workloads:]

ScyllaDB supports several approaches for compacting SS Tables, ranging from size-based (Size Tiered Compaction Strategy, or STCS) to time-based (Time Window Compaction Strategy*, TWCS) to an incremental, consistently-compacting strategy (Incremental Compaction Strategy, or ICS), which can be tailored to the needs of the application.

Incremental Compaction Strategy (ICS) is unique to ScyllaDB Enterprise. It is not available in ScyllaDB Open Source or any version of Cassandra. ICS makes more efficient use of persistent disk. Whereas SS Tables using other compaction strategies require a reserve of 50% of free disk space to accommodate compactions, with ICS, you can use up to about 85% of your disk, and only need about 15% of free disk space to conduct compactions. This reduces overall costs and enhances the utilisation of your hardware resources.

An e-commerce platform that utilises an SS Table (Sorted String Table) with a Log-Structured Merge (LSM) tree implementation can be found in Amazon DynamoDB. Here's a real-time example of how this works:

Steps to keep in mind for effectively working with ScyllaDB.

v. Installation and Setup:

Deploy ScyllaDB: You can deploy ScyllaDB on several platforms — On-premise servers, Cloud environments, and as a Docker image. Cluster Setup — Create a cluster with multiple nodes to ensure high availability and fault tolerance.

Hardware Configuration and Resource Checking: Properly configure ScyllaDB based on your hardware and workload. Watch Open File Descriptor Count. It may involve optimising memory allocation, disk I/O, and network settings to improve performance.

vi. Data Modelling:

Assign your schema: Define the columns for your schema with the Cassandra Query Language (CQL). Since ScyllaDB is a drop-in replacement that utilises the same data modelling concepts as Cassandra, you will define keyspaces, tables, partitions, and clustering keys. Partitioning: Define appropriate partition keys to guarantee uniform distribution of data on the cluster

vii. Writing Data:

Write CQL to insert data into ScyllaDB. The data will initially be written to an information Repository and a commit log. Optimal Writes: You can optimize write operations by tuning Repository, flush_writers, commitlog_sync, and some Disk I/O settings based on your hardware properties.

viii. Reading Data:

CQL Reads: Query with CQL. All of the following bits can be retrieved from the persistence layer of ScyllaDB, i.e., from Reference Data and Persistent Data Tables stored on disk. Based on Filters and indexes used by ScyllaDB, updates for read paths to minimize disk IO.Optimizing Reads. — Consistency Level like QUORUM, be watchful with Secondary Indexes and Materialized Views for avoiding extra I/O.

ix. Compaction:

Compaction parameters can be set at the time of table creation or updated dynamically using CQL commands.

x. Monitoring and Maintenance:

Monitoring [4] — Use Scylla Monitoring Stack, which comes with Grafana and Prometheus for observing performance metrics such as read/write latency, compaction throughput, and disk usage. Maintenance: Regularly monitor the SS Table size and count, disk space, and node health. Use Scylla Manager for tasks such as repairs and backups.

xi. Recovery Disaster Recovery Plan

Maintain backups & a recovery procedure so you can prepare for the worst.

xii. Testing and Optimization:

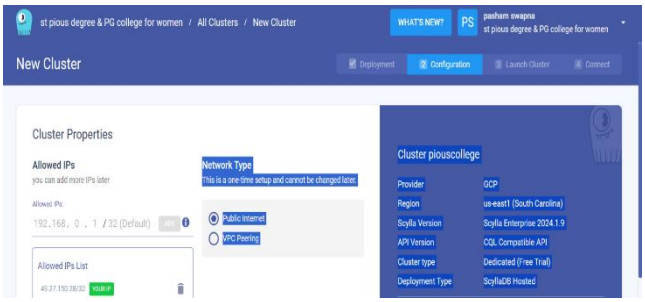
Conduct benchmarking: Run benchmarks on your ScyllaDB setup using various workloads to evaluate its performance. Fortunately, many tools exist, like Cassandra-stress and custom scripts.

III.EXPERIMENTAL IMPLEMENTATION

We tested our solution to implement a new ScyllaDB cluster for our college use case.

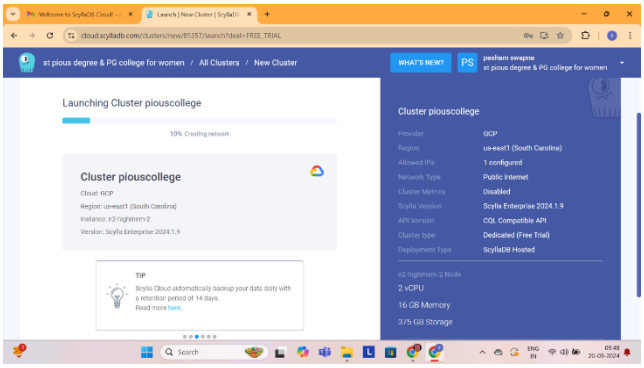
Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

Some of the requirements observed are Performance goals: Assess whether high availability, low latency, or distributed computing is a priority.

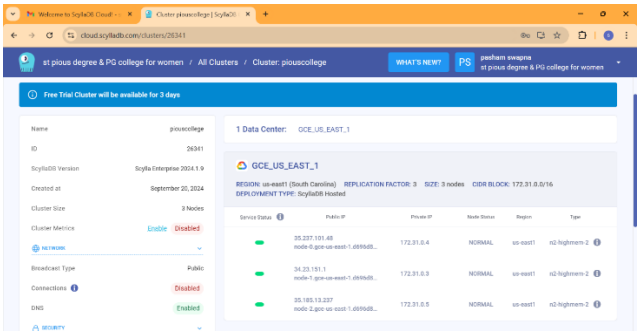


[Fig.3: Performance Goal Output 1]

Scalability: Plan for future growth in terms of both data size and concurrent users.

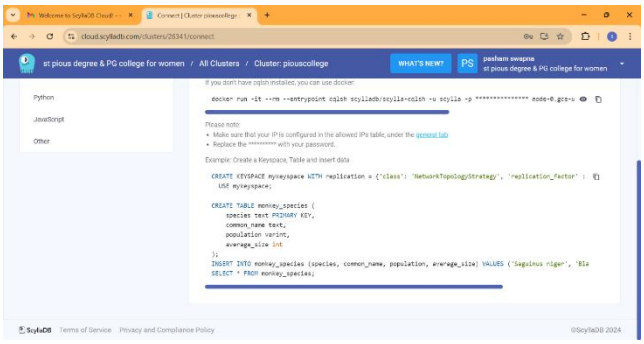


[Fig.4: Performance Goal Output 2]



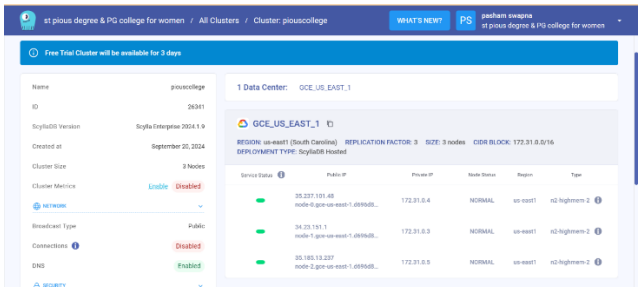
[Fig.5: Performance Goal Output 3]

ScyllaDB typically works with a set of nodes. For a college environment, start with at least three nodes to ensure replication and fault tolerance. Replication Factor: A replication factor of 3 is standard, meaning data is stored across three nodes for redundancy. Data Sharding: ScyllaDB automatically shards data across nodes for performance. Ensure nodes are distributed evenly to avoid hotspots.



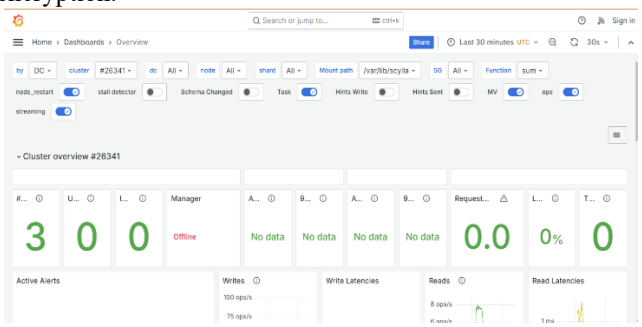
[Fig.6: Replication Factor]

Ey space: Define keyspaces for logical separation of data (e.g., one for student information, one for courses). Design tables for quick read/write operations, utilising ScyllaDB's column family structure—renormalise data when necessary for optimal performance.



[Fig.7: Private Networking]

Private Networking: Set up a private network for secure communications between nodes. Allow only necessary ports (e.g., 9042 for CQL) and secure the cluster with TLS encryption.



[Fig.8: Scylla Monitoring Stack]

Utilise the Scylla Monitoring Stack (Prometheus + Grafana) to track key performance metrics, including throughput, latency, and resource utilisation. Enable automatic backups and set up alerts for any node failure or performance degradation.

IV. FUTURE SCOPE

The future scope for Scylla DB, particularly with its use of LSM trees, includes significant opportunities for optimization in areas such as real-time analytics, scalability, AI/ML integration, and multi-cloud deployments. These advancements will continue to make Scylla DB a competitive choice for applications that require high performance, scalability, and low-latency data processing.
Here's a step-by-step breakdown of how each can be implemented:

A. Cluster Auto-Tuning Algorithm

The auto-tuning system dynamically adjusts configurations to maximise performance based on real-time data, including CPU usage, memory utilisation, disk I/O, and workload patterns.

Logical Flow for Cluster Auto-Tuning:

Step 1: Surveillance Layer

- **Collect Metrics:** Use monitoring tools (e.g. ScyllaDB Monitoring Stack) to gather metrics from all nodes in the cluster continuously:
  - CPU utilization
  - Memory usage
  - Disk I/O

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

- Query latency
- Network bandwidth
- Read/write throughput

Analyze Patterns: The system should analyze trends over time (e.g., peak traffic times, periods of low activity). Spot outliers, spikes, or fluctuations in workload [5].

Step 2: Decision Support System

Define Thresholds and Rules: Set thresholds for key performance indicators (KPIs). For example, if CPU usage exceeds 80% for more than 10 seconds, an action must be triggered. Use predefined rules or machine learning models to detect patterns.

- **Thresholds:** Determine upper and lower bounds for CPU, memory, and disk usage.
- **Rules:** Define conditions when tuning is necessary (e.g., high compaction load, CPU nearing max capacity, or queries with high latency).

Decision Logic: Based on the collected metrics, the system makes decisions. The logic can follow these patterns:

- **Compaction Tuning:** If a node's disk I/O or CPU is saturated, delay non-critical compaction tasks or adjust compaction thresholds.
- **Memory Tuning:** If frequent cache misses are detected, allocate more memory to the cache on nodes with high read traffic.

Step 3: Enforcement Layer

- **Revise Compaction Parameters:** Dynamically adjust the compaction frequency, size, or strategy (e.g., levelled or size-tiered).
- **Thread Resource Adjustment:** Assign more CPU cores to high-priority tasks or specific queries based on observed bottlenecks.
- **Feedback Loop:** After tuning, continue to monitor the effects of the change. If performance improves, retain the new configuration. If it worsens, roll back the changes or try another adjustment.

The logical implementation of the cluster auto-tuning Algorithm involves monitoring key metrics, analysing workload patterns, and making decisions based on predefined rules or machine learning models. This algorithm ensures optimal performance by balancing load across the cluster and automatically tuning parameters in real-time [5].

To mathematically prove the cluster auto-tuning algorithm, we'll employ statistical models and techniques such as regression analysis, optimization, and reinforcement learning principles. The goal of the Cluster Auto-Tuning Algorithm is to automatically adjust the system configuration to minimise response time (latency) and maximise throughput under varying workloads.

B. Problem Definition

Cluster Auto-Tuning Algorithm: Mathematical Representation

The Cluster Auto-Tuning Algorithm dynamically adjusts system parameters to optimise performance (e.g., throughput and latency) in response to real-time changes in workload characteristics.

V. OBJECTIVE

To automatically adjust parameters like compaction rate, memory allocation, thread pools, and I/O distribution to minimize latency while maximizing throughput under varying workload demands.

System Parameters: Let θ be a vector representing the system parameters, including

$\Theta = \{C_{comp}, M_{cache}, P_{threads}, IO_{rate}, F_{flush}\}$

Where

1. C_{comp}-Compaction
2. M_{cache}-Memory allocation to cache (SS Table, Mem Table, Bloom Filters)
3. $P_{threads}$-Number of threads allocated
4. IO_{rate}- Disk IO read, write operations, computation
5. F_{flush} Write flush rate from MM Table to SS Table
6. System Metrics:
7. The performance of the cluster depends on several key metrics, denoted as Xt at time t
8. L_t-Latency
9. C_t-CPU Utilization
10. Q_t-Throughput
11. M_t-Memory Utilization
12. D_t-Disk Utilization

The goal is to optimize performance by adjusting θ based on these metrics.

We define an Evaluation function to optimize the system performance as a Weighted Function of throughput Qt and latency L_t

Where:

1. α_t and β_t are user-defined weights, throughput or latency.
2. $Q_t(\theta t)$ and $\beta L_t(\theta_t)$ are functions of system parameter θ_t

$P(\theta_t) = \alpha_t Q_t(\theta t) - \beta L_t(\theta_t)$

The optimization factor for the system resources can be written as

1. $Mt(\theta t) \leq Mmax$, $Ct(\theta t) \leq Cmax$, $Dt(\theta t) \leq Dmax$

Where:

Mmax, Cmax, and Dmax represent the maximum allowable resource limits (memory, CPU, disk).

VI. CONCLUSION

ScyllaDB is a powerful choice for organisations that need a high-performance, scalable NoSQL database. Its modern architecture enables it to handle demanding workloads more efficiently than many of its peers, making it an excellent choice for real-time analytics, IoT, time-series data, and other applications. By being compatible with Cassandra, ScyllaDB enables easy migration and adoption, providing a significant performance boost with minimal disruption.

DECLARATION STATEMENT

After aggregating input from all authors, I must verify the accuracy of the following information as the article's author.

- **Conflicts of Interest/Competing Interests:** Based on my understanding, this article does not have any conflicts of interest.
- **Funding Support:** This article has not been sponsored or funded by any organization or agency. The independence of this research is a crucial factor in affirming its impartiality, as it was

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

conducted without any external influence.

- **Ethical Approval and Consent to Participate:** The data provided in this article is exempt from the requirement for ethical approval or participant consent.
- **Data Access Statement and Material Availability:** The data and materials supporting the findings of this article are publicly accessible.
- **Author's Contributions:** The authorship of this article is contributed equally to all participating individuals.

REFERENCES

1. Feng, L., Liu, L., Zhang, Z., & Xu, M. (2022). *A survey of compaction management in LSM-tree-based key-value stores.* Frontiers of Computer Science**,** 16(2), 1–23. DOI: https://doi.org/10.1007/s11704-021-0436-2
2. Chitransh, A., & Kalyan, B. S. (2021). ARM Microcontroller-Based Wireless Industrial Automation System. *Indian Journal of Microprocessors and Microcontroller*, 1(2), 8–11. DOI: https://doi.org/10.54105/ijmm.b1705.091221
3. Lakshman, A., & Malik, P. (2017). Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2), 35–40. DOI: http://dx.doi.org/10.1145/1773912.1773922
4. Wang, Y., Hu, Z., & Yang, X. (2023**).** *ElasticLSM: An elastic and adaptive compaction scheme for LSM-tree-based storage systems.* Future Generation Computer Systems**,** 139, 135–148. DOI: https://doi.org/10.1016/j.future.2023.01.012
5. Park, J., Lee, M., & Kim, J. (2022). *Write Optimization in Time-Series Databases Using Enhanced LSM Trees.* ACM Transactions on Storage (TOS)**,** 18(1), Article 3. DOI: https://doi.org/10.1145/3491027

AUTHOR'S PROFILE



Mrs. Bandi. Aruna received a BSc. Computer Science Degree from St. Theresa's Women's College, Eluru, A.P. in 1997. Received M.C.A. Degree from Madras University, Chennai, in the year 2006 and M.Phil from Anna University in 2008. She has been in the field of teaching for 26 Years. Her Areas of interest are cybersecurity and networks.



Mrs P. Swapna is a Computer Science Lecturer presently working at St. Pious X Degree & PG College for Women. Her Qualification is (M.C.A, MTech (CSE)) from Osmania University with 20 Years of Teaching Experience, Specializing in Machine Learning, AI, and Data Analytics.



Mrs. G. Vinutha completed my master's degree in MCA from Osmania University in the year 2007, working as an assistant professor for the past 13 years, having immense knowledge in research on Data Structures, User Interface applications, and developing a User Interface application.



Ch. Mary Pushpa completed her M. Tech at JNTU and has 11 years of teaching experience. Her area of interest is AI, Data mining, and Networks, having years of immense knowledge in research on Data Structures and User Interface applications.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of the Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)/ journal and/or the editor(s). The Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP) and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

© Copyright: All rights reserved.