[1]Kewal Krishan,

[2]Gaurav Gupta

[3]Gurjit Singh Bhathal

# A Consistent and Scalable Double Layered Hash Locator Framework for NoSQL Data Storage Systems


***Abstract:*** - Cultural nuances in translating between Japanese and Vietnamese present unique challenges. This research aims to explore the benefits and drawbacks of utilizing widely-used translation tools in overcoming these challenges, particularly for students at FPT University. The purpose of this article is to examine the effectiveness of translation tools in bridging the linguistic and cultural gaps between Japanese and Vietnamese, with a focus on facilitating communication for FPT University students. Qualitative and quantitative methods are employed to assess the reliability and accuracy of translation tools such as Google Translate and specialized software tailored to Japanese-Vietnamese translation. These methods include evaluating the tools' ability to preserve cultural nuances, accuracy in translating idiomatic expressions, and overall coherence in conveying messages. The study reveals promising results, indicating that translation tools offer expedited and consistent translations. However, limitations are also identified, including the potential loss of subtle cultural nuances and inaccuracies in context-sensitive translations. For FPT University students, these findings underscore the importance of leveraging translation tools as supplementary aids in language learning and cross-cultural communication. While these tools can enhance efficiency, students should remain mindful of their limitations and strive to develop their linguistic and cultural proficiency through other means. In conclusion, while translation tools offer undeniable benefits in facilitating communication between Japanese and Vietnamese, students should approach them as complementary resources rather than sole reliance. A balanced approach that integrates technology with traditional language learning methods is crucial for cultivating comprehensive language skills and cultural awareness among FPT University students.

***Keywords:*** scalable distributed structure, NOSQL, consistency, MongoDB, MemCached

## I. INTRODUCTION

Modern applications manage ever-increasing amounts of data, frequently surpassing the capacity of conventional SQL Relational Database Management Systems (RDBMS), especially when several clients are utilizing the system concurrently or when it is being used on an internet-scale basis. As a result, more straightforward yet effective data stores—referred to as NoSQL—have surfaced as RDBMS substitutes. This NoSQL framework prioritizes high performance, efficiency, and scalability over features such as complex relational models and ACID connections. An important aspect of reporting systems is stability, which is especially important in dispersed environments to avoid data loss [1-3].

Since it can be difficult to achieve complete convergence, models have been developed that show how a system can withstand anomalies but require a distributed system. The given paper resulted in the development of a scalable, two-tier NoSQL framework for monitoring system operations and preserving data integrity. This model includes a systematic framework for monitoring system operations and preserving data integrity. It provides several contributions, such as effective testing of synchronization problems to build robust synchronization programs, theoretical construction that emphasizes practical application, and performance testing against well-known NoSQL frameworks such as MemCached and MongoDB.

## II. LITERATURE REVIEW

NoSQL systems include multiple data models, such as document (e.g., MongoDB), graph (e.g., Neo4j), key-value (e.g., memcached), and column-oriented (e.g., Cassandra) weak-consistency models a Many NoSQL implementation systems.[4] One example is the Basically Available, Soft State, Eventually Consistent (BASE) model. According to this model, policy conditions will eventually converge, even though there may be distinct differences. [5] Typically, these models use anti-entropy protocols such as epidemic algorithms to reduce the heterogeneity between data store nodes. Meanwhile, a stronger stability criterion is needed, as simple

[1] Research Scholar, Department of Computer Science and Engineering, Punjabi University, Patiala, Punjab, India and Assistant Professor, School of Computer Science and Engineering, Lovely Professional University, Phagwara, India, kakkarkewal@gmail.com

[2]Assistant Professor, Department of Computer Science and Engineering, Punjabi University, Patiala, Punjab, India, gaurav.shakti@gmail.com

[3] Assistant Professor, Department of Computer Science and Engineering, Punjabi University Patiala, Punjab, India, gurjit.phathal@gmail.com

association models are not sufficient for practical application. Social media giants like Facebook have also seen a greater need for consolidation in their internal data storage systems. New SQL systems emerged in response to the need for greater robustness. [6] This system is designed to provide online transaction processing (OLTP) solutions using distributed data storage. Strong development models such as basic availability, scalability, and instant consistency (BASIC) are introduced by NewSQL frameworks such as Calvin, Megastore, and Spanner. Eventually, these systems will break down as they stabilize in favor of increased reliability. New SQL systems can impact system performance because, despite improvements, they rely heavily on robust mechanisms such as multi-version concurrency control, and some systems prioritize robustness over scalability or capacity use, such as VoldDB and RubatoDB [7–12]. The choice of stability model has a significant impact on the efficiency of these systems. For example, RubatoDB supports BASE, BASIC, and ACID instances but performs worse than traditional systems. [13]

## III. PROPOSED FRAMEWORK

- The division of headers and bodies into separate entities, enabling autonomous management, is a key optimization in the proposed framework. By reducing transfer problems during split operations, this method improves storage performance as a whole.
- Notably, a component's header and body are not always placed on the same server, even if a server may contain both kinds of buckets. Headers may move as a result of split operations, but bodies stay in one place. INSERT, UPDATE, RETRIEVE and DELETE are the four main actions used in the proposed framework. These are essential for effectively managing components. The distributed data store's data accessibility and integrity are guaranteed by these activities.
- A consistent structure known as a file is formed by each node in the distributed system storing a finite amount of entries in fixed-size containers called buckets.
- Range Partitioning (RP*), and Linear Hashing (LH*) are some of the techniques used in the proposed framework to handle records. LH* is especially useful for efficient addressing in the two-layer version of framework.
- To access data, each client keeps a file image, or local directory, of its own. But changes in file structure might make these pictures out of date and cause address issues. It is handled by LH* which transmits messages to the right bucket.
- In order to overcome the drawbacks of single-layered structures, a two layer proposed distributed framework is proposed, providing a method for effectively handling big files. Apart from improving split speed, it adds features like anonymity and throughput scalability.

Figure 1 shows the idea behind the two tier layered framework. The first layer comprises metadata as header ($\mathbf{h_k}$) while the second layer stores the actual data location as body ($\mathbf{b_k}$). Collectively, it forms a component represented by $\mathbf{C_k} = (\mathbf{h_k}, \mathbf{b_k})$.

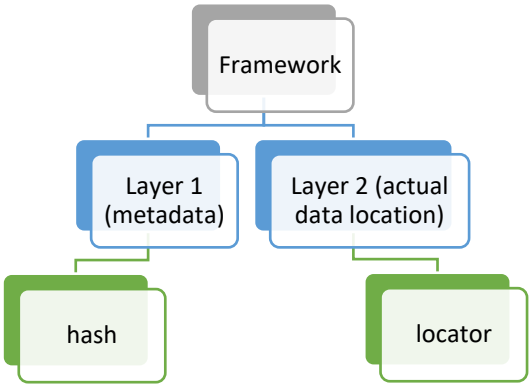


Figure 1: Structure of double layered scalable distributed framework

Each of the components is placed in different containers called buckets. Both layers have their respective buckets depicted as ($W_1$, h) and ($W_2$, b) in the form of header and body respectively. The algorithms connected with four basic operations used in the proposed framework are as follows:

***Algorithm 1: INSERT component 'm' in the body ($b_k$) of framework***

For i = hash (k)

{

If (ɔ $h_k$ belongs to $W_1 \wedge h_k$ . key = m) then stop

Else

{

$h_k$ . key = m;

$h_k$ . locator = $m_{new}$

$W_1 = h_k \vee m_{new;}$

End if

}

If ($W_1$, h) > threshold of the first layer bucket then

Split ($W_1$, h) // *to improve storage by reducing transfer issues//*

End if

}

***Algorithm 2: RETRIEVE component 'm' from the body ($b_k$) of framework***

For i = hash (k)

{

If (ɔ $h_k$ belongs to $W_1 \wedge h_k$ . key = m) then l = $h_k$

Else

{

stop

End if

}

Outcome = $W_1$. Retrieve ($W_1 \wedge h_k$ . key = m)

}

***Algorithm 3: DELETE component 'm' from the body ($b_k$) of framework***

For i = hash (k)

{

If (ɔ $h_k$ belongs to $W_1 \wedge h_k$ . key = m) then l = $h_k$

Else

{

Stop

End if

}

$W_1$.Delete ($W_1 \wedge h_k$ . key = m)

}

***Algorithm 4: DELETE component 'm' from the body ($b_k$) of framework***

For i = hash (k)

{

If (ɔ $h_k$ belongs to $W_1 \wedge h_k$ . key = m) then l = $h_k$

Else

{

Stop

End if

}

$W_1$.Delete ($W_1 \wedge h_k$ . key = m)

$W_1$.append ($b_{m)}$

}

## IV. RESULTS AND DISCUSSION

We performed a theoretical investigation of the proposed framework data store's performance in this part, concentrating on the computational cost of the key operations described in Algorithms 1-4.

Message passing time (T), component processing times on the first layer bucket (P1), and component processing times on the second layer bucket (P2) are found to be the three main performance-influencing parameters. The transmission time (T) is not affected by the internal organization of the data storage and stays constant.

**Time complexity after four operations**:

The time complexity of the **INSERT (m, bk)** operation varies from **3T + P1 + P2** in the most optimistic situation to **5T + P1 + P2** when a split event happens, according to our analysis of the time complexity in terms of the total number of components stored in the framework. Similarly, the time complexity of the **DELETE (m**) and **RETRIEVE (m)** operations is **3T + P1 + P2**, and the time complexity of **UPDATE (k, b0i) is 3T + P1 + 2P2**. Because P1 and P2 are independent of the number of components, the time complexity is written as O (1) assuming a fixed bucket size.

**Performance complexity**

It may be examined in relation to the number of customers (N) that are utilizing the data store at the same time. Since there are S total servers and each bucket is assumed to reside on a distinct server, the average load is represented as N/S. By letting several buckets operate on a single server or modifying bucket sizes to match server resources, the framework provides horizontal scalability [14–15].

Figure 4 shows the superiority of our proposed approach to existing NoSQL systems in the context of scalability. It is seen that the proposed two-layered framework achieves the least time to access resources, thereby ensuring the highest scalability among MongoDB and MemCached.

.

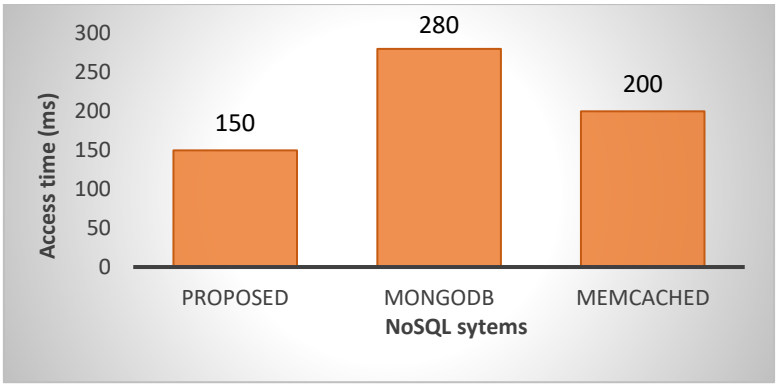


Figure 4: Scalability comparison while getting components

Figures 5 shows that even with different loads and component sizes, our suggested framework continuously outperformed the solutions that were already in place. Notably, because of the indirect component access brought on by layer separation, the proposed framework shows slower processing times for smaller components. Its complete horizontal scalability and excellent performance in highly trafficked areas, however, provide notable benefits over conventional systems. Figure 6 shows an efficiency comparison in relation to component body size.

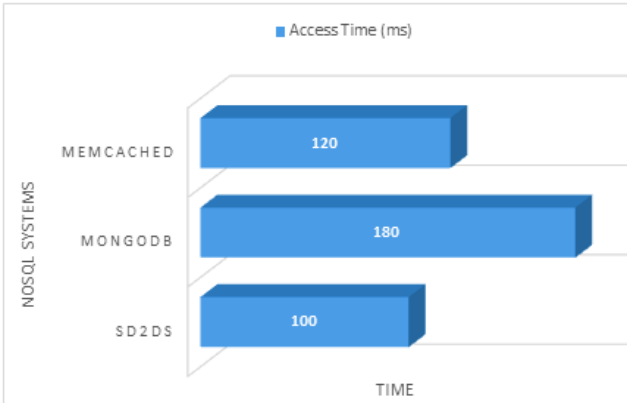


Figure 5: Efficiency comparison in response to access time

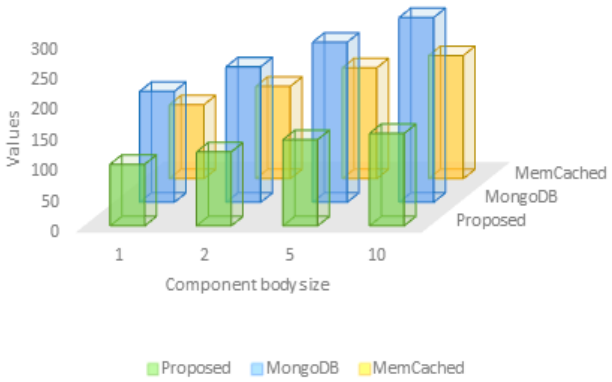


Figure 6: Efficiency comparison in relation to component body size

## V. CONCLUSION AND FUTURE WORK

In this work, we present the two-layered distributed architecture for NoSQL systems. We have implemented this design in a prototype and obtained encouraging testing results. The techniques used are range partitioning and linear hashing which are useful for accessing records and addressing data. To access data, each client keeps a

file image, or local directory, of its own. But changes in file structure might make these pictures out of date and cause address issues. It is handled by LH* which transmits messages to the right bucket. Interestingly, our tests confirm that maintaining consistency does not affect the datastore's capacity to scale. The results show that the proposed framework yields better results in terms of access time and component size as compared to the existing NoSQL systems such as MongoDB and MemCached. Our future research will focus on utilising the proposed architecture to design smart health monitoring systems that would ensure the higher credibility and scalability of patient records.

## REFERENCES


[1] Chin-Wei Wang and S. E. Chang, "Cloud service in stock trading game: Service virtualization, integration and financial application," 2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN), Vienna, Austria, 2016, pp. 857-862, doi: 10.1109/ICUFN.2016.7537158.

[2] A. Krechowicz, S. Deniziak and G. Łukawski, "Highly Scalable Distributed Architecture for NoSQL Datastore Supporting Strong Consistency," in IEEE Access, vol. 9, pp. 69027-69043, 2021, doi: 10.1109/ACCESS.2021.3077680.

[3] J. Magdum and R. Barhate, "Performance Analysis of DML Operations on NoSQL Databases for Streaming Data," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 2018, pp. 1-6, doi: 10.1109/ICCUBEA.2018.8697482.

[4] M. Scavuzzo, E. Di Nitto and S. Ceri, "Interoperable Data Migration between NoSQL Columnar Databases," 2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, Ulm, Germany, 2014, pp. 154-162, doi: 10.1109/EDOCW.2014.32.

[5] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas and N. Koziris, "Automated, Elastic Resource Provisioning for NoSQL Clusters Using TIRAMOLA," 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, Delft, Netherlands, 2013, pp. 34-41, doi: 10.1109/CCGrid.2013.45.

[6] S. Zareian, M. Fokaefs, H. Khazaei, M. Litoiu and X. Zhang, "A Big Data Framework for Cloud Monitoring," 2016 IEEE/ACM 2nd International Workshop on Big Data Software Engineering (BIGDSE), Austin, TX, USA, 2016, pp. 58-64, doi: 10.1145/2896825.2896828.

[7] E. Hossny, S. Khattab, F. A. Omara and H. A. Hassan, "STAGER: Semantic-Based Framework for Generating Adapters of Service-Based Generic-API for Portable Cloud Applications," in IEEE Transactions on Services Computing, vol. 14, no. 3, pp. 903-914, 1 May-June 2021, doi: 10.1109/TSC.2018.2831204.

[8] E. Hossny, S. Khattab, F. A. Omara and H. A. Hassan, "STAGER: Semantic-Based Framework for Generating Adapters of Service-Based Generic-API for Portable Cloud Applications," in IEEE Transactions on Services Computing, vol. 14, no. 3, pp. 903-914, 1 May-June 2021, doi: 10.1109/TSC.2018.2831204.

[9] E. Gupta, S. Sural, J. Vaidya and V. Atluri, "Enabling Attribute-Based Access Control in NoSQL Databases," in IEEE Transactions on Emerging Topics in Computing, vol. 11, no. 1, pp. 208-223, 1 Jan.-March 2023, doi: 10.1109/TETC.2022.3193577.

[10] J. Magdum and R. Barhate, "Performance Analysis of DML Operations on NoSQL Databases for Streaming Data," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 2018, pp. 1-6, doi: 10.1109/ICCUBEA.2018.8697482.

[11] P. Colombo and E. Ferrari, "Enhancing MongoDB with Purpose-Based Access Control," in IEEE Transactions on Dependable and Secure Computing, vol. 14, no. 6, pp. 591-604, 1 Nov.-Dec. 2017, doi: 10.1109/TDSC.2015.2497680.

[12] J. Mosquera and N. Piedra, "Use of Graph Database for the Integration of Heterogeneous Data about Ecuadorian Historical Personages," 2018 7th International Conference On Software Process Improvement (CIMPS), Guadalajara, Mexico, 2018, pp. 95-100, doi: 10.1109/CIMPS.2018.8625618.

[13] G. Daniel, A. Gómez and J. Cabot, "UMLto[No]SQL: Mapping Conceptual Schemas to Heterogeneous Datastores," 2019 13th International Conference on Research Challenges in Information Science (RCIS), Brussels, Belgium, 2019, pp. 1-13, doi: 10.1109/RCIS.2019.8877094.

[14] Rami Sellami; Bruno Defude, "Big Data Integration in Cloud Environments: Requirements, Solutions and Challenges," in NoSQL Data Models: Trends and Challenges , Wiley, 2018, pp.93-134, doi: 10.1002/9781119528227.ch4.

[15] H. Chihoub and C. Collet, "A Scalability Comparison Study of Data Management Approaches for Smart Metering Systems," 2016 45th International Conference on Parallel Processing (ICPP), Philadelphia, PA, USA, 2016, pp. 474-483, doi: 10.1109/ICPP.2016.61.