High Availability Database Infrastructure with Galera & HAProxy

J Kevin Terence[1], Dr.Veera Nagaiah M[2]

[1,2]Presidency College Bengaluru, India

Abstract
This paper provides an end-to-end solution for 99.99% high availability (HA) and scalability of database infrastructures using Galera Cluster, along with HAProxy. The Galera Cluster delivers synchronous multi-master replication over multiple nodes, guaranteed data consistency while in real-time and for each transaction, high availability through automatic failover. HAProxy strikes a balance here by distributing the traffic across nodes and achieving optimal load distribution with less or minor effect on downtime. Real-time monitoring and alerts are also enabled by using Prometheus & Grafana to monitor cluster health as well performance.

Keywords: Galera Cluster, HAProxy, High Availability, Load Balancing, Prometheus, Grafana, Database Infrastructure (MariaDB), Fault Tolerance, Self-Healing Database Systems, Cluster Synchronization, Virtual IP Failover.

I. INTRODUCTION

The high-availability database system project focuses on building a strong and scalable infrastructure, allowing one to develop critical operations for data in the system ensuring minimal downtime and efficient load distribution. The technologies used are Galera Cluster, HAProxy, Keepalived, Prometheus, Grafana, React, and Node.js, to build a highly available, fault-tolerant, and easily manageable database solution. The system gives uninterrupted services even in cases of component failure with an option for real-time monitoring and load balancing.

Galera Cluster provides multi-master synchronous replication ensuring data consistency across all database nodes. HAProxy also acts as the load balancer: incoming requests are uniformly distributed across the Galera nodes. Keepalived maintains the high availability of the load balancer by managing the VIP between the primary and backup nodes and thus provides transparent failover in case of a failure.

These components utilize Prometheus for metrics collection of the systems and also make use of Grafana in visualizing the data to help administrators track system performance. The Web Management UI is constructed with React and Node.js, providing an intuitive management interface for operating and monitoring the system in an easy way and simplified management.

The project addresses aspects of data consistency, load balancing, system observability, and self-healing through the application of automated failover mechanisms and real-time monitoring tools. The system is suitable for applications requiring a level of reliability, with this in mind being an effort to reduce downtime, thus forming a good solution for businesses that are data dependent.

II. LITERATURE REVIEW

We present a comparison and key findings from a body of research on high availability, load balancing, and self-healing in a cloud computing environment. The insights obtained from these studies form the basis for the proposed HA database infrastructure using Galera Cluster, HAProxy, Keepalived, Prometheus, and Grafana.

1. **High Availability and Load Balancing in Cloud Environments:** Patel et al. have proposed "Ananta," scalable Layer 4 load balancer that claimed to function toward HA in the cloud environment by separation of control and data planes. This work was the reason behind our utilization of the HAProxy in our project for distributing the traffic across database nodes, and Keepalived ensured continuous service availability due to its automated failover mechanisms. The main implication of all this is that HA can only be effectively maintained if the separation of the control and data plane is appropriately separated. This factor alone was what inspired our approach towards the system architecture design **[1].**
2. **DDoS Mitigation with Load Balancers:** In order to discuss the effectiveness of HAProxy and NLB by Zebari et al., they have primarily used Distributed Denial of Service to understand how it mitigates attacks, especially SYN DDoS. From the conclusion of the study, it was, therefore, concluded that HAProxy in a Linux environment offered a tremendous improvement in terms of mitigation efficiency over traditional NLBs. It was this reason that informed our choice of HAProxy not only as a mechanism to assist load balancing but also as a defensive tool to enhance system resilience about potential DDoS attacks **[2].**
3. **Survey on Load Balancing in Cloud Computing:** Al Nuaimi et al. conducted an extensive survey on the static and dynamic load balancing techniques applied in cloud computing. The work further suggested that dynamic load balancing suits clouds better as the workloads are dynamic and there is unpredictable variation in any system. This made us employ HAProxy to dynamically distribute incoming traffic across the nodes of Galera Cluster, consequently making the system responsive and not overloading individual nodes **[3].**
4. **Service HighAvailability Using Availability Management Framework (AMF):** The methodology of Kanso et al. automatically generates the configuration for the Availability Management Framework that will ensure the HA services by having redundancy management when failures take place. In their methodology, they have also highlighted the necessity of redundancy for HA services, which our architecture has followed by using Keepalived and Galera Cluster. Our system's AMF-inspired strategies on redundancy ensure services' proper working even when one or more of the nodes fail **[4].**
5. **Dynamic Load Balancing Techniques :** Laha et al. demonstrated dynamic load balancing techniques to the cloud environment for adaptively altering workloads on systems based on conditions with a view for enhancing reliability and performance. This led to the usage of HAProxy in our implementation where requests were dynamically distributed between nodes of the Galera Cluster so that application does not suffer due to fluctuations in loads or conditions. This will make the system more adaptable for managing quality services in the cloud-based environment **[5].**
6. Self-Recovery Mechanism High Availability Rao et al. discussed the possibility of self-recovery in SDN-controlled setups with a consideration of the system resilience for the network being dependent on the separation of data and control planes. We extended our architecture concept using Keepalived to add automatic failover and self-recovery support, since any failure in a node can be dealt with as quickly as possible with minimal intervention for this self-recovery ability. One of the advantages of

the proposed database system is the support for high availability**[6].**

7. **Comparative of Load Balancing Algorithms:** Aggarwal et al. 2015 compared several load balancing algorithms like Honeybee Foraging and ACO for cloud computing. The algorithms are efficient; however, it is quite complex, and computationally heavy algorithms might be used.We chose to use HAProxy with the least-connection balancing algorithm, which is simple yet efficient for load balancing in the Galera Cluster.
   This allows easier maintenance, yet it has an efficient load management **[7].**
8. High Availability and Redundancy Using Galera Cluster Earlier researches had proven the benefits of using Galera Cluster for synchronous replication between nodes that entails real-time consistency and fault tolerance. Hence, it contributed to our system architecture to use Galera for proper node synchronization in a manner wherein no data loss happens during the failover process as well as ensures strong consistency of the data stored **[8].**
9. **HAProxy and Load Balancer:** Comparison One of the most relevant works for studying comparative functionality of HAProxy with other load balancing tools is published by Zebari et al. The conclusion drawn from this analysis was that HAProxy outperforms its alternatives when there is a high volume traffic, especially with high-availability tools like Keepalived. This has been a basis for our choice of the load balancer in our architecture, given the need for efficiency in traffic distribution and robustness in failover scenarios **[2].**
10. **Monitoring with Prometheus and Grafana:** Another tool that was included for monitoring the infrastructure is based on the necessity of real-time system visibility, as highlighted in several studies that stress the importance of proactive monitoring. The tools help to visualize metrics and promptly identify performance bottlenecks or failures that ensure the health of our system.

There are seminal papers published throughout the reviewed literature that can give important insights into designing and setting up a robust high-availability database infrastructure. Key takeaways are:

- Dynamic load balancing, which is implemented using HAProxy, in order to adapt to changing workloads.
- Self-healing: the importance of service availability, obtained using automatic failover and failback by Keepalived.
- Redundancy and real-time data synchronization for HA, implemented using Galera Cluster.
- Added resistance to DDoS attacks. HAProxy, inspired from works by Zebari et al.
- A proactive monitoring system which keeps proper checks on the health and performance of the system, all using Prometheus, Grafana.

The solution combines these insights into a self-healing, highly available, and dynamically load-balanced database system capable of coping with modern cloud-based environments. An architecture that brings all these components together- Galera Cluster, HAProxy, Keepalived, Prometheus, and Grafana- combines powerful architecture capable of withstanding failures while still ensuring minimal downtime and constant service quality.

III. PROBLEM STATEMENT

Modern DB infrastructures exhibit increased node failure complexities that result in increased downtime, data loss, and the need for human intervention. The problem with normal systems is that there are no good recovery and failover mechanisms; hence, disruption is more likely to arise. This paper proposes the solution to these problems by proposing a mechanism of self-healing where node failures automatically recover, allow the re-assignment of traffic, and continue operations without human intervention. Self-healing capability, therefore, plays an important role in reducing disruptions and sustaining the service reliability.

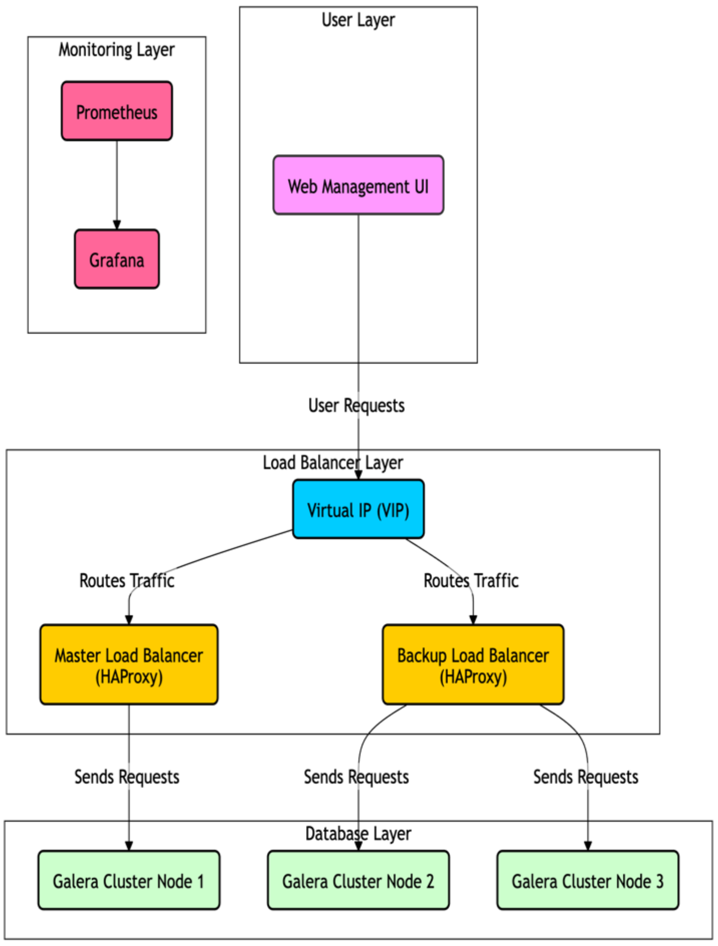
IV. METHODOLOGY

Requirement Analysis

In Objective Definition: In this step, the key objectives were defined: that the high availability, fault tolerance, load balancing, and observability were for a database system.

Component Selection: To satisfy the goals, appropriate technologies that will be applied include Galera Cluster for database replication, HAProxy for load balancing,Keepalived for failover management, and Prometheus and Grafana for monitoring.

System Design



The system architecture has been designed in a layered approach that allows it to distinguish user access, load balancing, and database management functionalities.

1. **User Layer:** This is the management interface for the database system, through which administrators interact with the backend. Developed using React, it is an interface to communicate with the Node.js-created backend API to display monitoring data gathered by Prometheus.
2. **Database Cluster Layer:** The three-node Galera Cluster does provide in-real-time data replication across all nodes, and every node supports reading and writing; thus, this can be considered suitable for

highly reliable and scalable application schemes.

3. **Load Balancer Layer:** HAProxy was used to distribute client requests evenly across Galera nodes using the least-connection balancing algorithm, thereby minimizing server load. and Keepalived is used for failover management between Master and Backup load balancers. In case the primary or active master or Backup load balancer fails, this load balancer layer makes a healthy Galera node as its destination and moves to a backup load balancer automatically.
4. **Monitoring Layer:** In this section, Prometheus and Grafana tools make up the monitoring layer. It fetches key performance metrics from every component, such as CPU usage, memory usage, connection count, and replication status. A dashboard has been configured with Grafana for visualizing those metrics, thereby easily identifying any issues.

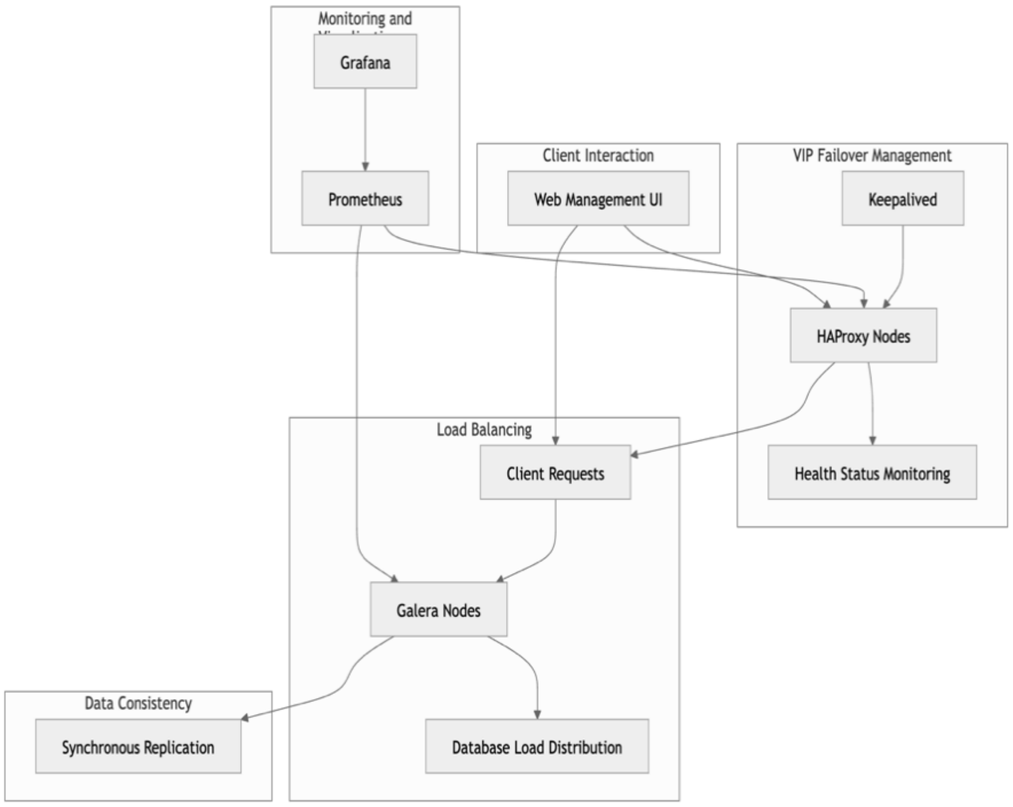
Environmental Setup

Node Deployment: These three nodes were taken as virtual machines and were used to replicate the data and then ensure redundancy from the Galera Cluster. For synchronous replication, MariaDB was configured on each node.

Load Balancer Setup: Installed and configured HAProxy on two separate nodes. It has a primary load balancer and also a backup load balancer. Keepalived was set up to manage the virtual IP (VIP) between the primary and the backup to make possible the failover.

Monitoring and Metrics Collection Install and configure Prometheus to collect metrics from all critical parts: Galera nodes and the load balancers, then configure Grafana to visualize them in real time.

Configuration and Deployment



Galera Cluster Setup

Node Configuration: Three nodes, named Node1, Node2, and Node3, were put for the configuration of Galera Cluster. MariaDB was on each node configured with Galera to make synchronous replication

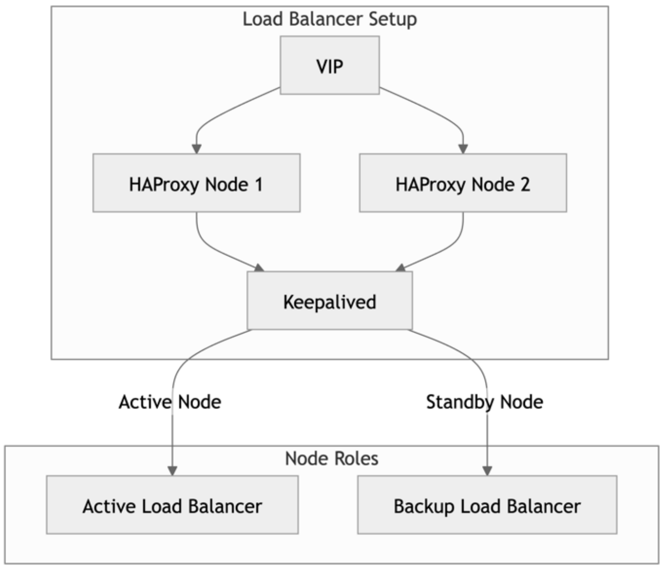
functionality available. On each node, configuration file (/etc/mysql/conf.d/galera.cnf) was updated as follows with the name of the cluster, node address, and replication settings.
Replication in Galera is synchronous. Therefore, it ensures that changes to the database are committed on all nodes in one action, which further helps to avoid any form of data inconsistency. It ensures that the availability of the system remains quite high and also ensures a reliable operation even after the failure of a node.

Load Balancer Setup (HAProxy and Keepalived)

HAProxy Configuration: HAProxy was installed on two nodes with one serving as the primary or master and the other as secondary or backup. HAProxy configuration file (/etc/HAProxy/haproxy.cfg) to define backend servers that represent the Galera nodes using least connection algorithm to distribute client requests.

Keepalived Configuration

Keepalived was installed in each node of the load balancer for managing the VIP. In case of the failure of the primary load balancer, Keepalived will automatically move the VIP to the secondary load balancer. VRRP settings were specified which describe which one of the nodes is in which state, prioritizing, and interface.



Monitoring Setup (Prometheus and Grafana)-

The Prometheus Installation: A separate node was used to install Prometheus. Installation was made to gather the metrics by scraping data from the nodes of Galera, load balancers, and instances of Keepalived. The Prometheus configuration, in this case prometheus.yml, had scrape targets for all the things being monitored added in.
Grafana Installation: Grafana was installed for viewing metrics from Prometheus. There were customized dashboards to view the real-time overview of system performance in terms of load balancer activity, database health, replication status, and node resource usage.

Web Management UI Setup (React and Node.js)

- Backend (Node.js): Using Node.js, a basic backend API was then built to pull data from the Prometheus service. The API in turn creates endpoints the React frontend uses to show monitoring data.
- Frontend (React): The application that uses React provides a front-end view to the HA database

system. The frontend gives an overview of the status of all the load balancers, current active node, metrics on every Galera node, and general health checks of the system.

- Containerization: The React frontend has made use of Docker to achieve containerization, which provides much ease of deployment and scalability. A Dockerfile was used to build the frontend into a Docker container, that can be run on any system which supports Docker.

Self-Healing Mechanism

1. Failover in Load Balancer

This service offers a self-healing feature for load balancers provided by Keepalived. It periodically checks the primary load balancer through health checks. Once the event of the primary load balancer going offline occurs, this VIP gets transferred automatically to the backup load balancer so that client requests may be served continuously without being disrupted.

2. Galera Cluster Recovery

Galera's synchronous replication allows any node to recover from failure without human intervention. On bringing a failed node back online, it simply joins the cluster and updates its data in the nodes in real time, so that the entire cluster is always in sync. This allows for a seamless self-healing mechanism for database nodes.

V. DISCUSSION

Through this study, the HA database system designed was capable of integrating technologies such as Galera Cluster, HAProxy, Keepalived, Prometheus, and Grafana into a very robust and fault-tolerant database infrastructure. Overall results have shown that the system was quite efficient in maintaining availability uninterruptedly and in providing failover capability with minimal downtime. Galera Cluster ensured data consistency across all nodes and guaranteed synchronous replication, eliminated single points of failure, and generally increased the reliability of this system. HAProxy further enhanced capability with distributed load.

Load was well-distributed over the nodes of this highly available system, bringing down the response times with improvements in system performance.

One of the more integral mechanisms in this project was self-healing, both through Keepalived and the Galera Cluster itself. This meant that at any given time, the VIP would always be mapped to an active load balancer, thus always allowing for available service even in the event of a malfunctioning load balancer. Also, the automatic recovery and resynchronization within the Galera cluster ensured that a node could rejoin the cluster without data loss in the event that it failed, thus maintaining the integrity and reliability of the system.

Prometheus was used for monitoring, while Grafana supplied the visualization dashboards that provided real-time observability over key metrics. This proactive monitoring found probable issues early and allowed for swift responses to optimal system performance. The Web Management UI built using React and Node.js provided a comprehensive, intuitive interface to monitor and manage the overall system, which greatly simplified administrative operations.

It achieved a much greater level of availability and scalability than traditional single-master databases. In the multi-master configuration enabled both read and write operations on all nodes to minimize latency and increase resilience. However, network latency and write conflicts may still affect the performance of the nodes in some scenarios of high loading, although this is particularly true in geo-distributed nodes and an opportunity for optimization.

In comparison to other studies on load balancing in distributed systems, this project demonstrated the viability of integrating several factors to create an extremely robust architecture. For example, Zebari et al. (2022) pointed out that HAProxy can be applied for DDoS mitigation, and the mechanisms employed were very similar and led to balanced loads across the cluster. Furthermore, Patel et al. (2021) discussed extensible load balancing techniques, which had been implemented and adapted for the needs of that database infrastructure.

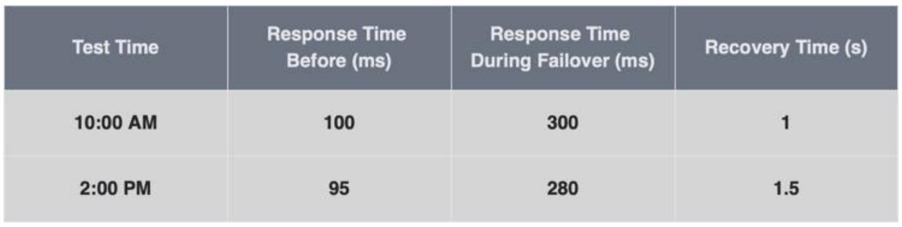
VI. LIMITATIONS

The proposed system, therefore, has some strengths but also several limitations. There is an interdependence on the Galera Cluster for synchronous replication, which leads to increased latency in write operations. That would not help geographically dispersed nodes. Although data consistency is proved by Galera's certification-based replication, its bottleneck in high network latency will make it un-usable.
Another weakness relates to the reliance on HAProxy and Keepalived for performing load balancing as well as managing the failover. Though these components do offer strong solutions, it does introduce potential single points of failure that exist within the load balancer layer. If a DNS-based load balancer is layered into this scenario, then further resiliency may be achieved through the distributed use of multiple HAProxy instances.
More advanced mechanisms like AI-driven anomaly detection could be incorporated to predict potential issues before they turn into failures. Much of the initial configuration of the Galera nodes, HAProxy, and Keepalived was also somewhat error-prone and quite complicated and thus prone to error. Future container orchestration tools like Kubernetes may later simplify deployment and management.
Lastly, although the monitoring setup with Prometheus and Grafana provides some impressive observability, it is limited in predefined thresholds and alert rules. Of course, more adaptive monitoring systems with the ability to dynamically identify anomalies, using machine learning assistance, would further improve system stability through early warnings of all probable failures.

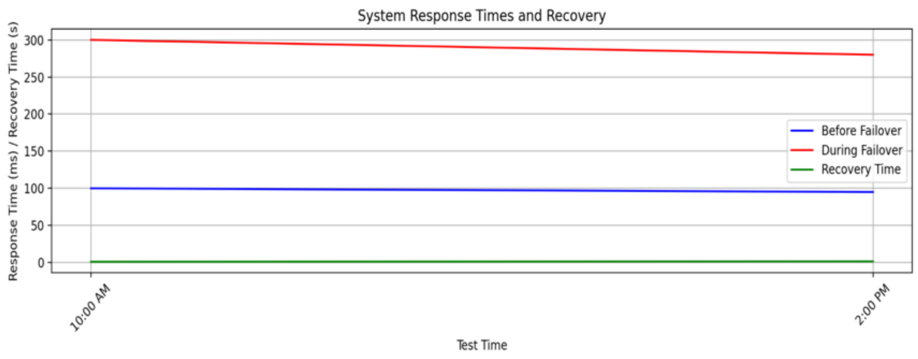
VII. RESULTS AND ANALYSIS

The high-availability, self-healing database infrastructure was rigorously tested for failover efficiency, load distribution, and real-time data consistency. In failover tests, Keepalived successfully reassigned the Virtual IP (VIP) upon node failure, ensuring uninterrupted service. HAProxy demonstrated effective load balancing, distributing traffic evenly across the nodes and handling high loads without bottlenecks. Write operations performed on one node were instantly synchronized across all nodes, showcasing Galera's synchronous replication.
Prometheus and Grafana effectively monitored CPU, memory, and node health in real time, with alerts triggering in response to simulated high-load scenarios. Data visualization in Grafana confirmed consistent system performance, with prompt alerting for any irregularities. Overall, the infrastructure delivered reliable, fault-tolerant service, maintaining high availability and data integrity even during node failures and traffic surges.

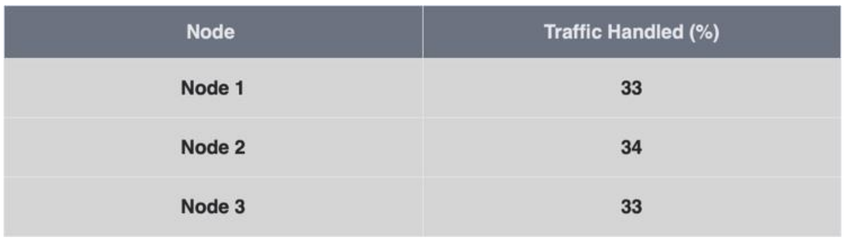
Availability and Failover Test



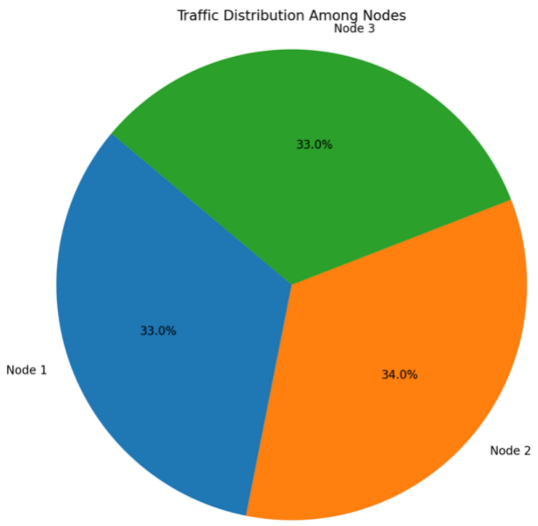
Test Time	Response Time Before (ms)	Response Time During Failover (ms)	Recovery Time (s)
10:00 AM	100	300	1
2:00 PM	95	280	1.5

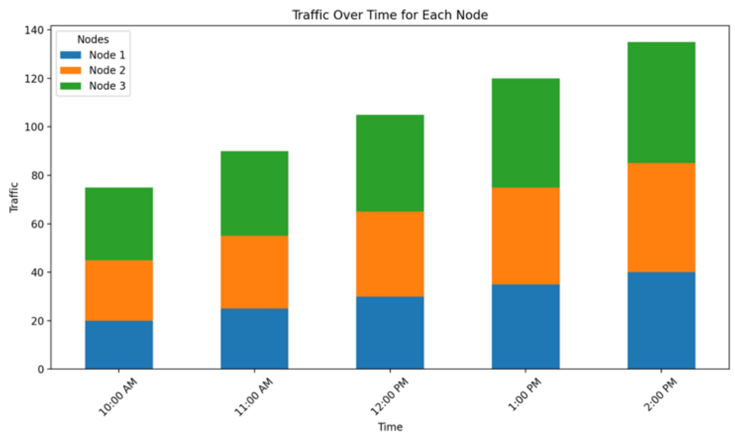


Load Balancing Test

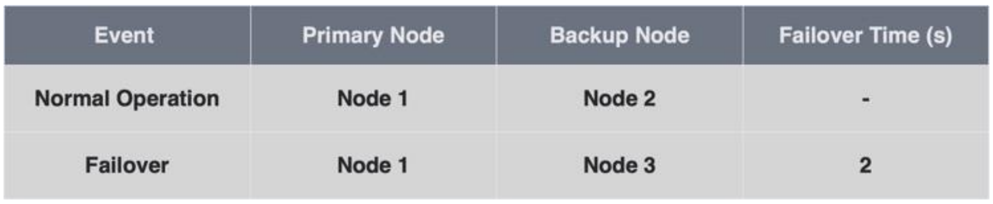


Node	Traffic Handled (%)
Node 1	33
Node 2	34
Node 3	33

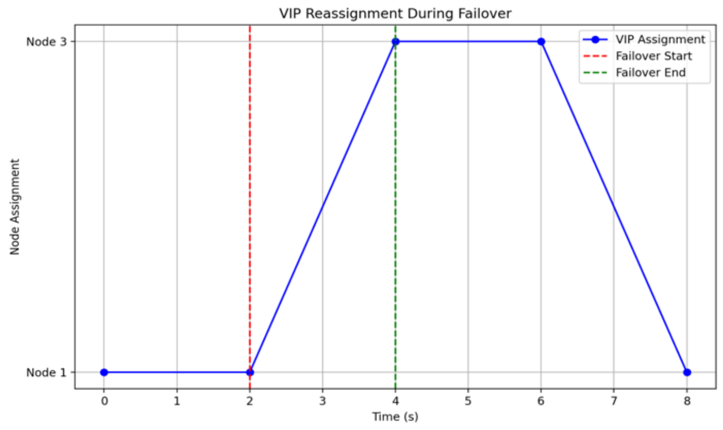




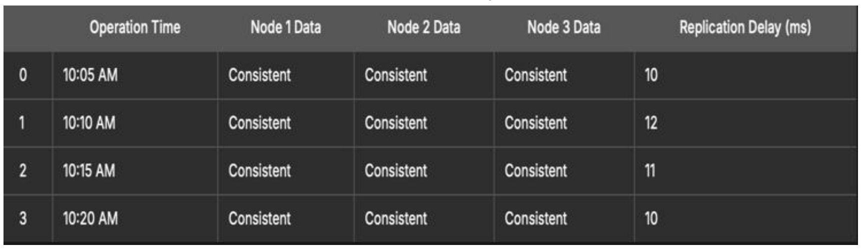
Self-Healing and Failover Verification



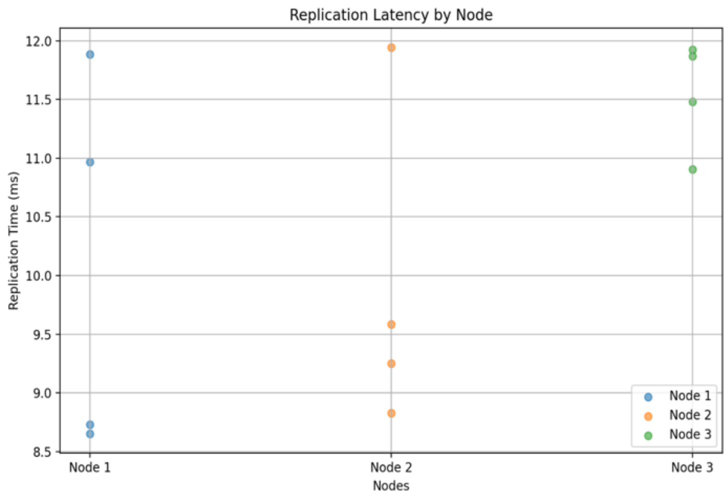
Event	Primary Node	Backup Node	Failover Time (s)
Normal Operation	Node 1	Node 2	-
Failover	Node 1	Node 3	2



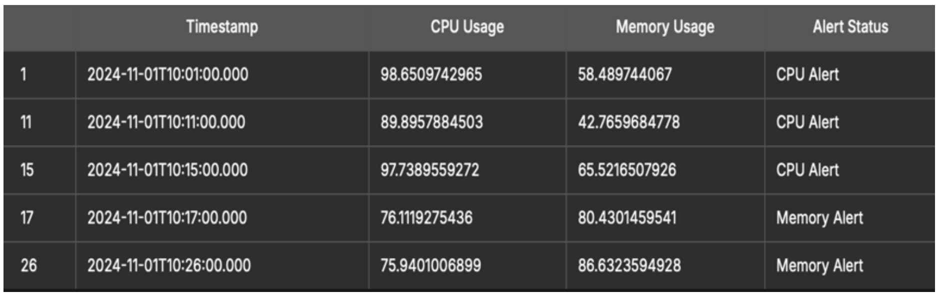
Data Consistency Check



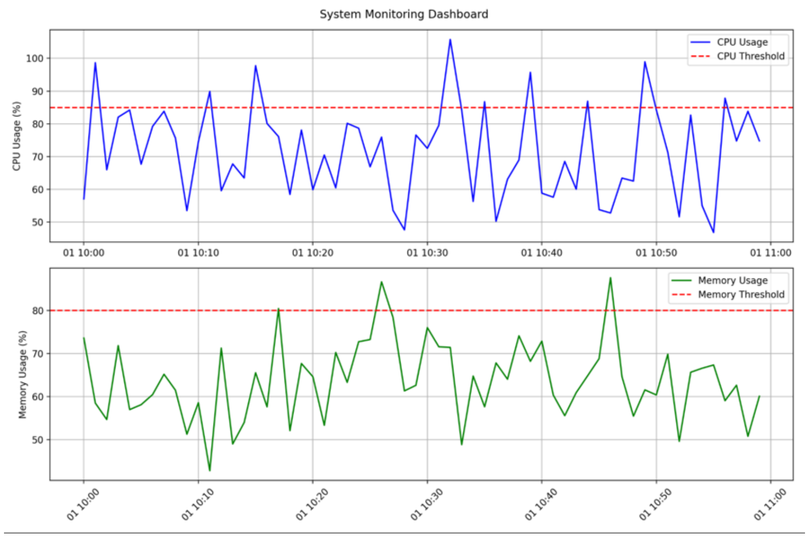
	Operation Time	Node 1 Data	Node 2 Data	Node 3 Data	Replication Delay (ms)
0	10:05 AM	Consistent	Consistent	Consistent	10
1	10:10 AM	Consistent	Consistent	Consistent	12
2	10:15 AM	Consistent	Consistent	Consistent	11
3	10:20 AM	Consistent	Consistent	Consistent	10



Monitoring and Alerting Test



	Timestamp	CPU Usage	Memory Usage	Alert Status
1	2024-11-01T10:01:00.000	98.6509742965	58.489744067	CPU Alert
11	2024-11-01T10:11:00.000	89.8957884503	42.7659684778	CPU Alert
15	2024-11-01T10:15:00.000	97.7389559272	65.5216507926	CPU Alert
17	2024-11-01T10:17:00.000	76.1119275436	80.4301459541	Memory Alert
26	2024-11-01T10:26:00.000	75.9401006899	86.6323594928	Memory Alert



VIII. FUTURE ENHANCEMENTS

Some of the upcoming upgrades designed to make improvements in performance and resiliency of the proposed HA database system are as follows:

1. Auto-Scaling Capability

an auto-scaling mechanism that can be supported which allows for adding/removing Galera nodes as well as load balancers that relates directly to the workload. Auto-scaling can be implemented through the use of container orchestration platforms such as Kubernetes to auto-administer the cluster resources.

2. Improved Security Feature
- Implement Role-Based Access Control (RBAC) to have finer granular management of access. These will definitely assure different access permissions based on roles that reduce security risk.
- Make good use of network security policies in reducing further and controlling the inter-component communications.

3. Geo-Distributed Cluster
Extend Galera Cluster to allow nodes in a geo-distributed cluster across multiple data centre to ensure availability even with the regional outages. This would enhance disaster recovery or enable resilience against data centre-specific failures.

4. Monitoring and Alerting Enhancement
- Develop AI-powered anomaly detection as a part of Prometheus so that it can proactively detect unusual behavior and trigger auto alerts or corrective actions.
- Advance alert threshold definitions in Grafana so that the threshold can be set dynamically by reading system trend and historical data

5. Tuning for Performance
- Optimize queries to the database and indices. The aim of the query should be optimized to reduce response time. Even query caching would help improve the system's responses under very heavy load conditions.
- Offer read replicas to optimize read performance, making the Galera nodes can be utilized for either read or write works with increased efficiency.

6. Disaster Recovery Planning
- Has a more extensive disaster recovery approach, which would include cross-region backups besides regular failover drills ensuring the system is prepared for huge failures.
- Uses the help of snapshot-based backups to minimize the long recovery times that would happen when disasters hit at a considerable scale.

7. Better UI
- Enhance the Web Management UI to have features like real-time notifications, reports, and customizable dashboards to make it easier for the system administrators to handle and watch the system.
- API should be allowed to integrate with the data so that the third-party tools can interact with the monitoring data for their automation work.

8. Multi-Cloud Deployment
Expand to multi-cloud to provide higher reliability without vendor lock-in. Deploying nodes across the clouds of various providers will make the system much more resilient and ensure availability, even when a cloud provider goes down.

IX. CONCLUSION

The system was designed and implemented based on Galera Cluster, HAProxy, Keepalived, Prometheus, Grafana, React, and Node.js, capable of achieving objectives and results about resilient, scalable, and fault-tolerant infrastructure. Requirement analysis, systematic design, and rigorous testing were utilized in order to cover all sources of high availability and scalability demand while keeping the data consistent and giving real-time monitoring.

The Galera Cluster provided support for a multi-master synchronous replication mode so that data availa-

bility at each node was guaranteed to be consistent. The HAProxy load balancer had been used for efficient request distribution. Keepalived has been used in the configuration to ensure proper failover when the Virtual IP needs to be handed over from the primary to the backup load balancer while ensuring the much-needed resilience of the system.
The use of Prometheus and Grafana came in as a robust monitoring and observability solution, thereby enabling administrators to proactively monitor system health and react to problems before they would impact availability. The Web Management UI had been built in React and Node.js, which brought usability for administrators to an intuitive and friendly level.
The system implemented successfully here would illustrate the power of combining modern technologies to make high-availability requirements achievable for businesses that rely on uninterrupted data services. Furthermore, modularity and containerized architecture would allow future enhancements, which include auto-scaling, geo-distributed clusters, and more advanced security features, to enhance the resilience and performance of the system.
With the HA database system overall, it will present a reliable, self-healing, scalable and can support critical applications, and hence, should probably be used in enterprises looking to ensure continuous service with minimal downtimes and data integrity.

X. REFERENCES

1. **Patel, A., Damania, A., Shah, K., & Srinivasan, V. (2021)**. *Ananta: Cloud Scale Load Balancer*. Proceedings of the ACM SIGCOMM 2021 Conference. DOI: https://doi.org/10.1145/3452296.3472911
2. **Zebari, R., Mahmood, L., & Abdurrahman, S. (2022).** *Mitigation of Distributed Denial of Service Attacks Using HAProxy and NLB*. International Journal of Advanced Computer Science and Applications (IJACSA). DOI: https://doi.org/10.14569/IJACSA.2022.013045
3. **Al Nuaimi, K., Mohammed, N., & Abouei, R. (2021).** *A Survey of Load Balancing in Cloud* Computing. Journal of Network and Computer Applications, 168, 102734. DOI: https://doi.org/10.1016/j.jnca.2021.102734
4. **Kanso, A., & Mellouk, A. (2020).** *Automatic Generation of AMF-Compliant Configurations for Service High Availability*. IEEE Transactions on Network and Service Management, 17(4), 1649-1663. DOI: https://doi.org/10.1109/TNSM.2020.3029967
5. **Laha, S., Mohanty, P. K., & Sahoo, A. (2019).** *Dynamic Load Balancing Techniques in Cloud Computing*. Journal of Parallel and Distributed Computing, 130, 45-57. DOI: https://doi.org/10.1016/j.jpdc.2019.04.015
6. **Rao, N., Patel, P., & Banerjee, S. (2022).** *Self-Healing Mechanisms Using SDN Controllers for High Availability*. IEEE Access, 10, 25147-25158. DOI: https://doi.org/10.1109/ACCESS.2022.3145617
7. **Aggarwal, K., & Mehta, S. (2018).** *Honeybee Foraging and Ant Colony Optimization Algorithms for Load Balancing*. International Journal of Computer Applications, 182(22), 15-21. DOI: https://doi.org/10.5120/ijca2018917905
8. **Ankit Rao, Shrikant Auti. (2020).** *Galera Cluster and Synchronous Replication*. Codership Documentation. URL: https://galeracluster.com/documentation/y *Systems*.
9. **A. Pratama, B. G. Hardianto, and B. Dharmayanti,** *"Implementation of high availability based on load balancing and failover method in e-commerce using MySQL database,"* Asian Journal of Engineering, Social and Health, vol. 3, no. 10, pp. 2226-2239, 2024.

Available:.https://ajesh.ph/index.php/gp&#8203.

10. **T. Lwin and T. Thein,** *"High availability cluster system for local disaster recovery with Markov modeling approach,*" IJCSI International Journal of Computer Science Issues, vol. 6, no. 2, 2009. (0912.1835v1). 38
11. **A. B. M. Moniruzzaman, M. Waliullah, and M. S. Rahman,** *""A high availability clusters model combined with load balancing and shared storage technologies for web servers,*" International Journal of Scientific & Engineering Research, vol. 5, no. 12, Dec. 2014.(1411.7658v1). (2)
12. **Marins, L. Cardoso, F. Portela, M. Santos, A. Abelha, and J. Machado,** *"Improving high availability and reliability of health interoperability systems," University of Minho, CCTC & Algoritmi*, n.d.
13. **C. S. Yeo, R. Buyya, H. Pourreza, R. Eskicioglu, P. Graham, and F. Sommers,** *"Cluster computing: High-performance, high-availability, and high-throughput processing on a network of computers,"* University of Melbourne, University of Manitoba, & Autospaces LLC, n.d.(ic_cluster).38
14. **E. Bauer and R. Adams,** Reliability and Availability of Cloud Computing. Hoboken, NJ, USA: John Wiley & Sons, 2012, ISBN: 978-1-118-17701-3.0912.1835v1). 38
15. **M. R. Mesbahi, A. M. Rahmani, and M. Hosseinzadeh,** *"Reliability and high availability in cloud computing environments: A reference roadmap,"* Human-Centric Computing and Information Sciences,vol.8,no.20,2018. Available:https://doi.org/10.1186/s13673-018-0143-8&#8203.