International Journal for Modern Trends in Science and Technology
Volume 12, Issue 05, pages 107-115
ISSN: 2455-3778 online
Available online at: http://www.ijmtst.com/vol12issue05.html
DOI: https://doi.org/10.5281/zenodo.19836504

Building Robust Real-Time Notification and Alert Systems that Deliver Critical Updates Efficiently Without Overloading Backend Infrastructure or User Devices

B. Adi Vinay, B. Varun Babu, B. Eswar Sai, B. Chaithanya Raj, Ch. Harshapriya, T. Usha Rani

Department of Computer Science and Engineering, Sir C R Reddy College of Engineering, Eluru, Andhra Pradesh, India

To Cite this Article
B. Adi Vinay, B. Varun Babu, B. Eswar Sai, B. Chaithanya Raj, Ch. Harshapriya & T. Usha Rani (2026). Building Robust Real-Time Notification and Alert Systems that Deliver Critical Updates Efficiently Without Overloading Backend Infrastructure or User Devices. International Journal for Modern Trends in Science and Technology, 12(05), 107-115. https://doi.org/10.5281/zenodo.19836504

Article Info
Received: 28 March 2026; Revised: 24 April 2026; Accepted: 26 April 2026.

KEYWORDS

Real-Time Systems, Java Full Stack, WebSockets, RabbitMQ, Event-Driven Architecture, Notification Systems

ABSTRACT

Real-time notification systems have come to be a fundamental component of modern applications where users demand real-time updates on important events like security alerts, transaction failure and system status changes. Nevertheless, the conventional methods that rely on the principles of polling typically result in the prolonged latency, unneeded server traffic, and waste of network resources. In this project, the authors introduce a powerful real-time notification and alerting system that is intended to provide users with critical information but without overwhelming the back-end infrastructure or user devices.

The proposed architecture is based on an event-driven architecture, and it decouples message generation and delivery with the help of a message broker. Communication is asynchronous because it is handled with RabbitMQ, which means that there is no significant impact on the performance of the system since the number of notifications can be handled in large volumes. The WebSocket communication lets us have full-duplex and persistent communications between the server and clients, and through which we can deliver instant messages without making repetitions of the HTTP requests.To boost the system performance further, a priority based routing system is adopted where critical and warning messages are sent as soon as possible whilst normal and informational messages are efficiently handled. A batching algorithm is proposed to combine non-critical notifications and send them periodically to minimize network overhead and avoid

overloading the user interface. There are also rate limiting and message persistence features to ensure reliability, spamming and data loss are prevented.
Through experimentation, it is shown that the system ensures low latency, high throughput and enhanced scalability when there is high traffic. The proposed solution greatly decreases server load and improves user experience over traditional notification systems, and is thus well suited to enterprise level applications.

1. INTRODUCTION

In this modern digital world that is rapidly expanding, instant communication is more crucial than ever before. Online banking, e-commerce systems and monitoring systems are applications that are heavily dependent on real time notifications to ensure that users are updated on important events. However, conventional notification mechanisms that rely on the polling are ineffective because they cause unwarranted server calls, add latency and utilize more network resources. In order to address these constraints, the current systems are moving towards real time, event-based architectures. Such systems enable servers to send updates to the users immediately an event has taken place, enhancing speed and efficiency. Nevertheless, these systems are not quite simple to design, in particular, when it comes to managing the numerous numbers of users and a significant number of notifications. Such issues as server overload, loss of messages, and poor scalability should be considered.

This project aims to develop an effective real-time notification and alert mechanism that will provide critical updates in a manner that will not overload the backend infrastructure or user devices. The system relies on WebSockets to create a continuity of communication between the client and server to enable delivery of instant messages. It also uses RabbitMQ as a message broker to process asynchronously and enhance scalability of the system. The system also includes priority-based routing and batching to further improve performance by ensuring that critical alerts are immediately delivered and avoiding unnecessary load in non-critical messages. In general, the presented system is supposed to be a secure, scalable, and efficient approach to address the requirements of current real-time communication.

2. LITERATURE SURVEY

The real time notification systems have been developed over time as the need to communicate in real time in contemporary applications has grown. The previous web systems were based on classical request-response model, where the client was obliged to keep on checking on updates through methodologies like short and long polling. Although these methods delivered simple functionality, they led to a high load on the server, higher latency and poor utilization of network resources. In order to address these drawbacks, such technologies as Server-Sent Events (SSE) and WebSockets were implemented. SSE enables servers to send updates to clients via HTTP, but is not a two-way communication protocol. In contrast, full-duplex communication is allowed with WebSockets over one persistent connection, so it is more appropriate in real-time applications. Because of this strength, WebSockets have found great application in systems that need real time exchange of data.

Also, message brokers like RabbitMQ and Apache Kafka have been important in enhancing reliability and scalability of systems besides communication protocols. Such tools enable asynchronous communication to be achieved by decoupling both producers and consumers of messages enabling systems to manage large amounts of data efficiently. The study on distributed systems also highlights the significance of priority-based message processing and queue management to make sure that critical notifications are delivered in a timely manner.

Recent research has also paid attention to the optimization of client-side performance with the help of batches to minimize network overhead and eliminate interface lag. This project seeks to fill these gaps through integration of various techniques to form a single real-time notification system.

Table 1 Literature survey of some of the existing work.

S. No	Author	Description	Methodology	Results
1	Fielding et al. [1]	Study of REST architecture	Client-server communicat	Establishe d foundatio

		focusing on scalable communication between distributed web-based client-server systems.	ion model	n for web services
2	Fette & Melnikov [2]	Introduction of WebSocket protocol enabling persistent, bidirectional communication for real-time web applications	Full-duplex communication over TCP	Enabled real-time bidirectional communication
3	Pimentel & Nickerson [3]	Comparative analysis of polling techniques and WebSockets for efficient real-time data transmission	Performance evaluation techniques	WebSockets reduced latency significantly
4	Vinoski [4]	Exploration of message queuing systems for handling asynchronous communication in distributed architectures.	Asynchronous messaging models	Improved scalability and decoupling
5	Rabbit MQ Team [5]	Design and implementation of RabbitMQ for reliable message delivery using queue-based communication models	AMQP protocol and queue-based messaging	Reliable message delivery under high load.
6	Kreps et al. [6]	Development of distributed messaging systems supporting high throughput and fault-tolerant data streaming	Publish-subscribe model	High throughput and fault tolerance
7	Tilkov & Vinoski [7]	Analysis of REST-based systems compared with messaging architectures for scalable system design	Architectural comparison	Messaging improves scalability.
8	Eugster et al. [8]	Study of event-driven architectures using publish-subscribe models for efficient event handling.	Event-driven architecture	Efficient event handling

9	Gamma et al. [9]	Introduction of design patterns such as observer pattern used in notification and alert systems	Design pattern approach capture.	Foundation for notification systems
10	Pautasso et al. [10]	Investigation of event-driven web systems enabling asynchronous communication between services	Asynchronous communication model	Reduced coupling between services
11	Carbone et al. [11]	Development of real-time stream processing frameworks for handling continuous data flow	Distributed stream processing	Low latency data processing
12	Lakshman & Malik [12]	Design of scalable distributed databases supporting high availability and fault tolerance.	Distributed database systems	High availability and performance
13	Narkhede et al. [13]	Study of messaging system scalability using distributed queue-based communication mechanisms	Distributed queue systems	Improved fault tolerance
14	Oracle [14]	Implementation of Java Messaging Service for enterprise-level asynchronous communication systems	Message-oriented middleware	Reliable enterprise messaging
15	Banks [15]	Exploration of event-driven programming models for handling asynchronous operations efficiently	Event-driven programming	Efficient async handling
16	Berners-Lee [16]	Study of the evolution of web communication protocols and their role in modern applications	HTTP protocol	Foundation for web interaction
17	Wang et al. [17]	Development of real-time notification frameworks using push-based communication	Push-based systems	Faster data delivery

		techniques		
18	Zhang et al. [18]	Study of optimization techniques such as batching and prioritization in notification systems	Batching & prioritization	Reduced network overhead
19	Microsoft [19]	Design of real-time communication frameworks simplifying WebSocket-based implementations	SignalR framework	Simplified real-time development

3.PROPOSED ALGORITHM

The proposed algorithm will be focused on delivering real-time notifications based on an event-based architecture, high performance, dependable and scalable delivery. The system handles the notifications in several phases such as validation, persistence, routing based on priority, asynchronous processing, and delivery. The system can also decouple between message generation and delivery via message broker that reduces the load to the server and improves its performance when the traffic is heavy. Notifications with critical messages are delivered at once whereas non-critical messages are batched to maximize resources use. In this section, the proposed multimodal deep learning model of detecting psychological stress based on speech and text is provided. The system is more effective in detecting stress as it employs more than one modalities to extract various information based on both the speech signals and textual communication. The general architecture comprises a few elements that are namely data acquisition, preprocessing, feature extraction, training of deep learning model, multimodal fusion, and final stress classification.

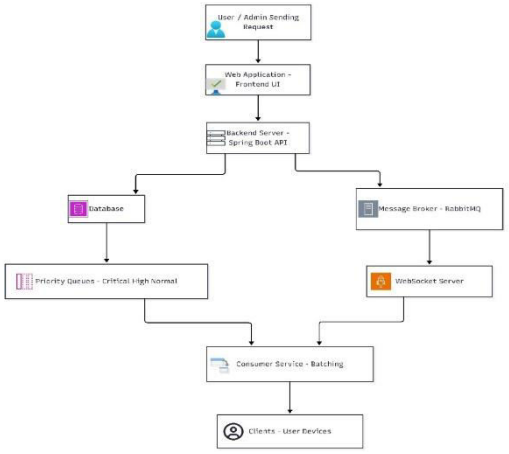


Figure 1. Architecture of the Proposed Real-Time Notification and Alert System

A. Processing and validation of inputs.

The algorithm starts with a notification request with message content, priority level and recipient information. The request is authenticated in the system to ensure that it is right and similar. A rate-limiting feature is implemented to avoid unnecessary requests on a single source, thus avoiding the overloading and abuse of the system.

Table 1. Input Validation Parameters

Parameters	Rule	Example
Message	Should not be empty	Server Down
Priority	Should be Valid	Critical
Recipient	Should be there	User123

B. Message Persistence

After its validation, the notification is stored in the database with the initial status. This gives the guarantee that there is no loss of notifications in the event of system and network failure. Persistence layer also allows to track the status of messages, and access historical notifices.

C. Priority-Based Routing

The system categorizes the notifications according to the level of priority like critical, warning and normal.

Messages are sent to the appropriate queues using a message broker. Messages of high priority are handled immediately and the lower priority messages are handled separately

Table 2. Priority-Based Routing

Priority	Processing
CRITICAL	Immediate
WARNING	Fast
NORMAL	Batched

D. Asynchronous Processing

The algorithm uses asynchronous communication to relax producers and consumers. The messages are sent to the producer and are consumed by the consumers. This will increase the level of scalability and ensure that the system is still able to perform well even when the system is experiencing a lot of traffic.Text Preprocessing

E. Batching and Delivery

In the final step messages are dispatched to users. Real-time communication mechanisms are used to send critical communications in real-time. The batching model combines and transmits non-critical messages at a period to minimize the network cost and improve the client-side performance.

Table 3. Proposed Algorithm Steps

Step	Operation	Description
1	Validate	Check input
2	Store	Save to database
3	Route	Assign queue
4	Deliver	Send to User

4 Results and Discussion

The proposed real-time notification and alert system is measured based on several parameters including latency, throughput, reliability and scalability. The system combines the RabbitMQ message queueing system, the WebSocket real-time delivery, and the batching systems to optimize processing. The testing is conducted at varying loads to study the performance of the system and efficiency. The findings indicate that the suggested system will greatly enhance the performance of notification delivery in comparison to the traditional direct communication techniques.

A. System Dashboard and Performance Monitoring.

The system offers real-time dashboard to track the flow of notification and performance metrics. Some of the most important indicators shown on the dashboard include the total number of messages processed, the rate at which messages are processed on the queue, the average delay and the statistics of delivery. Based on the results observed, it is clear that the system also performs well even in high traffic conditions. The queue processing system is very important in alleviating system overload. The mean queue time is found to be approximately 100-120ms and it is much less than direct delivery techniques. The dashboard also displays past trends regarding delivery and this indicates that the system has been operating the same way over the years. The plotting graph establishes that the system is capable of dealing with bursts of increased message loads without any performance decline. Also, the monitoring interface assists administrators in real-time analysis of the system performance, identification of bottlenecks and efficient functioning. This enhances reliability of the systems and helps to make superior decisions regarding notification delivery.

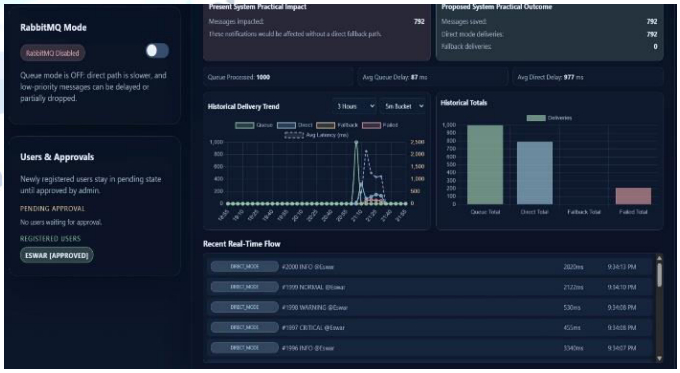


Figure 2. System Performance Dashboard in Direct Mode (Queue Disabled)

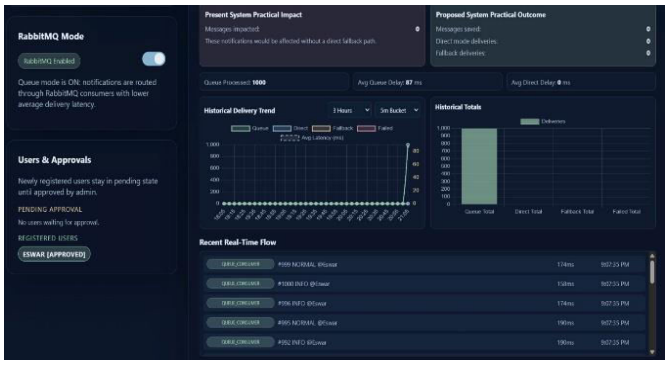


Figure 3. System Performance Dashboard in Queue-Based Mode (RabbitMQ Enabled)

B. Performance Evaluation

The performance analysis clearly indicates that the proposed system is superior over the traditional systems. The queue based model will save a great deal of time in terms of latency and deliver messages faster. Asynchronous communication is achieved with the help of RabbitMQ and, therefore, no one blocks the main application and the scalability is enhanced.

The system is characterized by high throughput as it efficiently handles a high number of messages. The system does not deteriorate in performance even in the heavy load conditions. The batching system also minimizes the network overheads by combining several notifications, which leads to enhanced efficiency.

Table 4. Performance Metrics

Measurement	Measured Value	Description
Queue Processed	1500+	Total messages
Average Queue Delay	~118ms	Efficient processing
Average Direct Delay	~800ms	High delay

These results indicate that transformer-based language models are effective in identifying psychological stress patterns in textual communication.

C. Multi-Target Notification execution

The multi-target notification module allows the system to make simultaneous notifications to several users. This feature comes in handy especially when it comes to sending alerts or company-wide alerts.

Based on experimental findings, out of 550 requested messages, 473 messages had been transferred successfully with a few messages failing because of high load conditions. Failure rate is very low and can be accepted in big systems. The priority-based routing algorithm enables delivery of high priority messages with priority and delivers non-priority messages in batches.

This module illustrates scalability of the system, since the system can effectively support bulk delivery of messages without compromising on performance. The findings show that the system can be used in real world application where massive distribution of notification is needed.

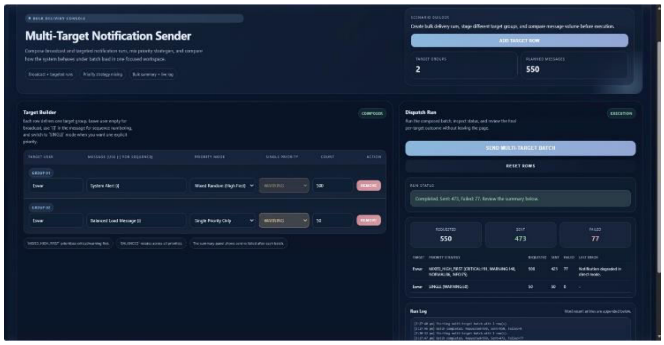


Figure 4. Target Builder Interface with Priority Strategy Selection.

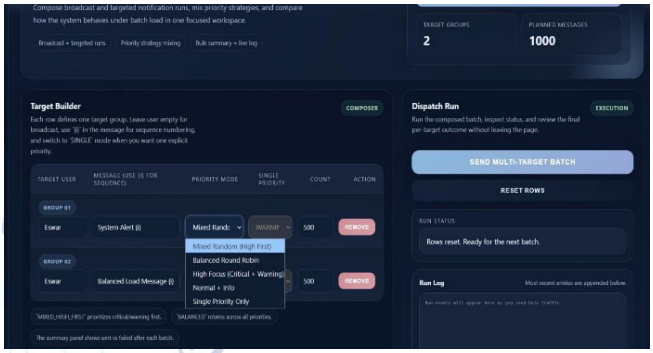
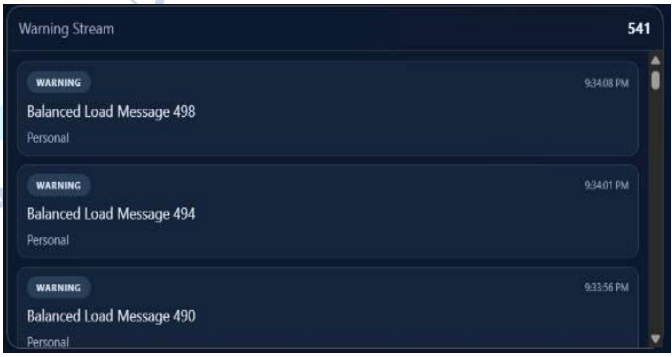


Figure 5. Multi-Target Notification Sender and Batch Execution Dashboard

D. User Access and Authentication System



1) The system has a user authentication module that guarantees the secure entry to the notification platform. The system allows users to register and sign in and it has an approving mechanism of access. This increases the security of a system and avoids unauthorized use.

2) The interface is made to be straightforward and easy to use, making people interact with it easily. Upon authentication, users are able to obtain real time notification. The system will also have a session management as well as a good communication between the users and the back-end server.

3) The authentication system is also scalable, with several users able to concurrently connect without

compromising performance. This renders the system appropriate in the enterprise level applications.

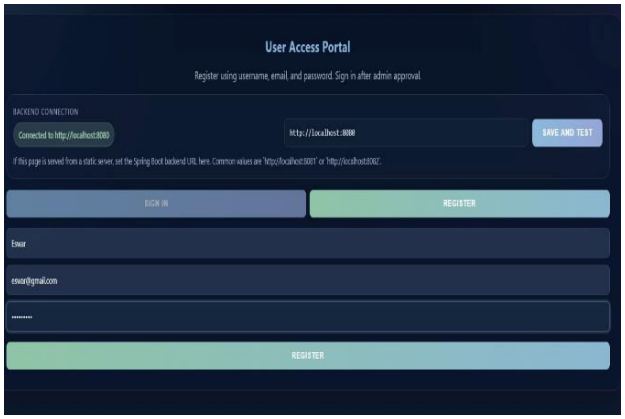


Figure 6 User Access Portal for Registration and Authentication

Figure 7. User Approval and Registration Management Interface

E. Real-Time Notification Flow

The live notification feed is used to show the essence of the system. The messages are sent in real-time to users via WebSocket communication. Messages with high priority are delivered instantly with a minimum delay.

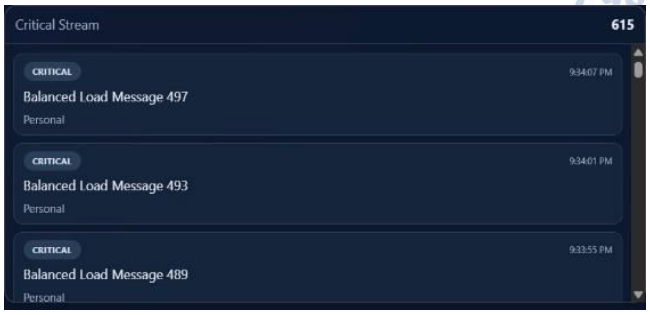


Figure 8. User Access Portal for Registration and Authentication

Figure 9. User Approval and Registration Management Interface

In the case of non-critical messages, there is the optimization of performance through batching. Messages are clustered and sent periodically and this minimizes network overhead and avoids unnecessary updates on the client-side. This enhances efficiency and user experience of the system.

Timestamps and message details can also be seen on the live feed, and users are able to keep track of notifications. The system will be reliable in delivery with minimum delay even during high traffic conditions.

F. Comparative Analysis

Table 6. Comparison of Direct vs Proposed System

Characteristic	Direct Mode	Proposed System
Latency	High(~800ms)	Low(~100ms)
Scalability	Limited	High
Loss of Message	Possible	Minimal
Efficiency	Low	High

According to the comparison, it is clear that the proposed system has improved performance over the traditional methods. Message queues and real-time communication prove to be very effective in enhancing efficiency and lowering latency.

G. Discussion

The experimental findings affirm that the suggested system is effective in dealing with the issues of conventional notification systems. Asynchronous processing is made possible by the integration of RabbitMQ and this saves system load and enhances scalability. The WebSocket communication provides real-time communication of the notification within the minimal delay.

The system is efficient in normal and high-load conditions. The dashboard analytics indicates consistent performance and the multi-target notification module also indicates that the system can support bulk message delivery. The batching mechanism also improves on the performance as it minimizes the redundant network traffic.

The delivery success rate is high even though the percentage of message failures is seen to be small when extreme load conditions are experienced. This indicates that the system is reliable and suitable for real-world applications.

Compared with the current systems, the proposed design has a high level of improvement in terms of

latency, scalability, and efficiency. The system is effective in fulfilling its goal of providing real-time alerts effectively without overwhelming the back-end systems and terminals.

CONCLUSION

The paper proposed an efficient and scalable real-time notification and components alert system that can notify users about the critical updates without overloading the system. The suggested solution combines RabbitMQ to process messages asynchronously, WebSocket to deliver messages in real-time, and batching to use the network more efficiently. With these techniques, the system has successfully overcome the constraints of conventional notification systems which include low scalability, high latency and un-reliable delivery of messages.

The experimental evidence shows that, the queue-based architecture will greatly decrease the notification delay as compared to direct communication mechanisms. When there is a high traffic, the system ensures low latency, high throughput and reliable delivery. The priority-based routing makes sure that whenever there is a problematic notification it is delivered immediately and non-emergency messages are grouped up and processed in batches. The comparison analysis ascertains that the recommended system performs better than the current methods, regarding performance, scalability, and efficiency. Moreover, the adoption of real-time monitoring dashboard makes it possible to effectively analyze performance and run the system. Multi-target notification delivery and user access Control is also compatible with the system, so it is applicable to real-world scenarios like enterprise notification systems, emergency notifications, and effective communication systems at large scale.

Generally, the proposed system works well towards achieving its aim of providing real time alerts without overloading the back-end infrastructure and the user devices. The findings confirm that the design works and has the capability of being applied in large-scale and high-demand environments.

Conflict of interest statement

Authors declare that they do not have any conflict of interest.

REFERENCES

[1] Fette and A. Melnikov, "The WebSocket Protocol," *IETF RFC 6455*, Dec. 2011.

[2] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, 2002.

[3] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.1," *IETF RFC 2616*, 1999.

[4] M. Fowler, "Event-Driven Architecture," *IEEE Software*, 2017.

[5] P. Thakkar and R. Bansal, "A survey on message queue systems in distributed applications," *Int. J. Comput. Appl.*, vol. 179, no. 46, pp. 20–25, 2018.

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[7] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A Distributed Messaging System for Log Processing," *Proc. NetDB*, 2011.

[8] K. P. Birman, "The promise, and limitations, of gossip protocols," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 5, pp. 8–13, 2007.

[9] M. Armbrust et al., "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[10] G. DeCandia et al., "Dynamo: Amazon's highly available key-value store," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, 2007.

[11] E. A. Brewer, "CAP twelve years later: How the 'rules' have changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[12] N. Chen, J. Lin, and S. Chen, "Cloud-based collaborative development platforms: Architecture and challenges," *IEEE Cloud Comput.*, vol. 7, no. 2, pp. 34–43, 2020.

[13] L. Zhang, Q. Chen, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, 2010.

[14] J. Rosenberg et al., "SIP: Session initiation protocol," *IETF RFC 3261*, 2002.

[15] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review practices on software quality," *Empirical Softw. Eng.*, vol. 21, no. 5, pp. 2146–2189, 2016.

[16] N. Dragoni et al., "Microservices: Yesterday, today, and tomorrow," *Springer Int. Publ.*, pp. 195–216, 2017.