Global Journal of Engineering and Technology [GJET].
ISSN: **2583-3359** (Online)
Frequency: Monthly
Published By GSAR Publishers
Journal Homepage Link- https://gsarpublishers.com/journal-gjet-home/

A method of increasing the reliability and scalability of web applications using distributed caching

By

Volodymyr Kozub

National Aviation University 03058, 1 Liubomyr Husar Ave., Kyiv, Ukraine https://orcid.org/0009-0007-6740-5300

Article History

Received: 05/12/2024
Accepted: 16/12/2024
Published: 18/12/2024

Vol – 3 Issue – 12

PP: - 12-18

Abstract

This paper presents a new approach to improving the reliability and scalability of web applications by implementing distributed caching. Traditional centralized caching systems are often limited by a single point of failure and limited scalability, which makes them vulnerable to heavy loads and rapid traffic growth. To solve these problems, the proposed method uses a distributed architecture with multiple proxy servers that dynamically balance the load and replicate cache data across the network. This design provides high availability due to failover capabilities, where backup servers can maintain continuous service during individual server failures. The distributed caching approach simplifies scaling, allowing additional servers to be seamlessly integrated to meet growing user demand without significant infrastructure changes. Experimental results show that this method improves response time, reduces server downtime, and optimizes resource utilization, making it a reliable solution for modern, high-traffic web services. Traditional caching methods, such as centralized caching, often face load and single point of failure issues, which limits the system's ability to quickly adapt to changes in traffic and requests. To address the issues mentioned above, the distributed caching proxy model proposes to distribute cached content across multiple servers, increasing availability by eliminating a single point of failure. It also improves reliability by balancing the load across multiple servers, preventing any single server from being overloaded.

Keywords: *distributed caching, web application scalability, centralized caching, load balancing*

1. Introduction

Ensuring reliability and scalability of distributed systems is a challenge for modern web applications. One study emphasizes that key mechanisms such as fault detection, replication, and monitoring are essential to maintain high availability in cloud environments (Mesbahi et al., 2018). These techniques align well with distributed caching strategies, where replication plays an important role in maintaining service continuity across multiple nodes.

Research on reactive microservices for increasing availability in IoT applications has shown that distributed systems can be more resilient than centralized ones. In such systems, microservices can dynamically adapt to changes, making them highly effective in environments that require constant availability (Santana et al., 2020). Such adaptive behavior is necessary in distributed caching, where node failures must be handled without affecting overall system performance.

A reinforcement learning approach to optimize distributed caching for IoT environments has been investigated, demonstrating how intelligent caching can reduce latency and improve response time in edge computing scenarios (Tian et al., 2021). The ability to dynamically tune caching methods is key to ensuring that data is always available, even when demand fluctuates. This makes this strategy directly applicable to web applications that depend on distributed caching.

In 5G cloud-based mobile networks, caching-as-a-service models have been developed to improve performance and cost-effectiveness. These models emphasize the importance of dynamic caching solutions that scale according to user demand while minimizing operational costs (Niyato et al., 2020). Distributed caching aligns with this approach as it allows for scalable solutions. They reduce network congestion and improve data retrieval time.

The analysis of caching requests between network nodes emphasizes the role of distributed caching in load balancing and improving access times (Pasyeka et al., 2019). By distributing cache memory across multiple servers, web applications can avoid performance limitations. This provides faster response times and higher reliability, especially during traffic spikes.

*Corresponding Author: Volodymyr Kozub

The impact of caching on the energy consumption of progressive web applications (PWAs) has been studied and found that efficient caching mechanisms can significantly reduce energy consumption while maintaining performance (Malavolta & Kuppusamy. 2020). This finding is important for large-scale distributed caching systems, where energy efficiency becomes a significant factor in maintaining performance at lower operating costs.

Distributed caching algorithms have also been applied to streaming videos. They have been shown to optimize delivery and improve user experience by reducing latency (Poularakis et al., 2019). The use of intelligent caching strategies in media-intensive applications illustrates how distributed systems can scale efficiently while maintaining quality of service. A comparative analysis of popularity-based caching strategies has demonstrated that these methods improve content delivery by adapting cache memory based on user demand (Naeem et al., 2020). This strategy is effective for web applications that handle large amounts of traffic, where distributed caching can optimize resource allocation. Also, ensure that frequently accessed data is always easily accessible.

The analysis shows that distributed caching is an effective way to improve the reliability and scalability of web applications. By using replication, load balancing, and dynamic resource management, distributed systems can respond flexibly to failures and scale efficiently, making them the best choice for modern web environments.

2. Methodology

The methodology covers several key aspects: analyzing existing caching topologies, defining and testing architectural approaches to distributed caching, and comparing their effectiveness with centralized solutions. This makes it possible to assess the impact of each topology on the performance, reliability, and scalability of web applications under high load conditions.

The main stages of the methodology include:

1. Selection and analysis of topologies: first, existing caching approaches (centralized, distributed) and their variations in the context of web applications are considered. Each topology is evaluated in terms of its advantages and disadvantages to ensure reliability and scalability.
2. Designing a test environment: to test the selected topologies, an environment is created that simulates real-world load conditions on the web application. This includes setting up servers, virtual machines, and proxy programs for data caching.
3. Define key parameters and metrics: set the parameters by which each topology will be evaluated: availability, reliability, scalability, system response time, and ability to withstand large volumes of simultaneous requests. These metrics are used to compare the performance and stability of each approach.
4. Testing of caching topologies: Each topology is tested to assess its ability to handle heavy load and ensure service continuity. Tests cover scenarios with a large number of simultaneous users, different levels of requests, and system load.
5. Formulation of recommendations: based on the results obtained, recommendations are made on the use of different topologies depending on the requirements for the web application, the type of traffic and the required level of scalability and reliability.

We will test the system for different topologies.

The centralized architecture consists of an ISP connection, a modem, a router, and a proxy server that is responsible for caching (Figure 1).

All client requests go through this single proxy server, which caches web content to reduce bandwidth usage and speed up response times. However, since this server is a single point of caching, it is a critical point of failure.

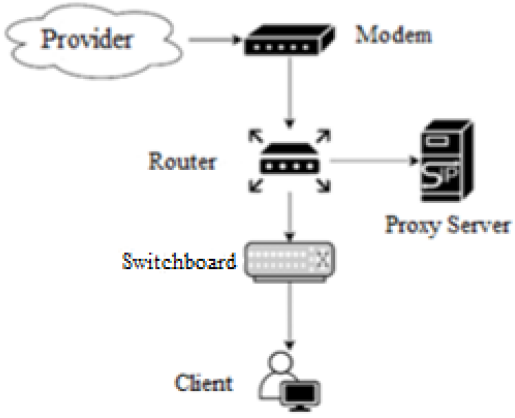

Figure 1. Physical topology with a centralized cache

This approach has the following features:

1. Single point of access to the cache: all client requests go to one cache server, which makes it easy to control and administer the caching system.
2. Simplified management and monitoring: A single caching server greatly simplifies configuration, monitoring, and maintenance processes. This reduces the need to coordinate multiple servers and makes it easy to make changes.
3. Scalability issues: there is a limited ability to scale, as the performance of a single server quickly decreases with the number of requests. Therefore, this solution is suitable for small systems with predictable and relatively low traffic.
4. Single Point of Failure: If the cache server fails, the entire system becomes unavailable, which can negatively affect the availability of the web application. This is especially critical for highly loaded web applications where stability and reliability are key factors.

During the test, a connection was established between the ISP and the modem, and then the modem was connected to the

*Corresponding Author: Volodymyr Kozub

router. The router, in turn, is connected to a centralized proxy server that serves as an intermediary for all requests from customers. Clients access the Internet through this proxy server. Performance metrics, such as response time and server load, are monitored to assess the ability of the centralized system to handle increasing traffic.

A distributed caching system uses a more robust physical topology (Figure 2), where multiple proxy servers are deployed to distribute the caching load across multiple nodes to improve performance and reliability. The components are connected in a distributed manner. This ensures that if one proxy server fails, the system can seamlessly redirect traffic through another server, thereby maintaining availability.

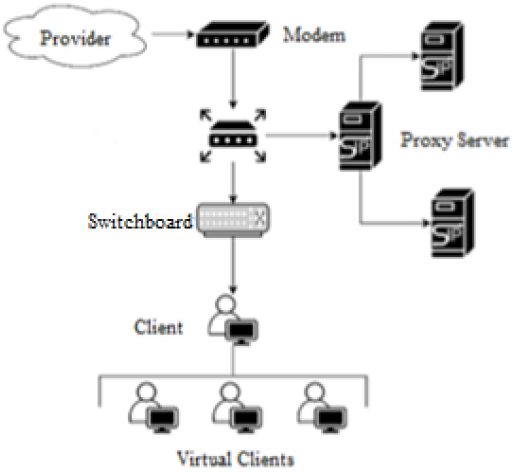

Figure 2. Physical topology with distributed cache

Each proxy server is connected to a central network switch, which is connected to a router and modem. This topology not only improves reliability by distributing the cache, but also supports scalability by allowing more proxy servers to be added as needed. The distributed cache system is designed to avoid the limitations and points of failure typical of a centralized architecture.

Features of the distributed cache topology include:

1. Load balancing: Cached data is distributed across multiple servers, which allows for more efficient load balancing on each individual node. Requests are distributed across servers using balancing algorithms, such as Nginx's ip_hash, which ensure that client requests are consistently redirected to specific servers.
2. High availability: the problem of a single point of failure is eliminated, as requests can be redirected to other nodes if one server fails. This ensures continuity of customer service even in the event of individual node failures.
3. Scalability: Caching can easily add new servers to support growing traffic or requests, allowing it to remain stable even at peak loads. New servers are integrated into the system dynamically, expanding the overall network capacity.
4. Geographic distribution of the cache: you can place servers in different geographical locations, which reduces access delays for customers, providing faster content delivery by storing the cache closer to users.
5. Reduced latency and increased processing speed: by distributing cached content and processing requests on different servers, latency is reduced and overall processing speed is increased.

For server virtualization, the star topology is used, where all server nodes are connected to each other through a central switch using UTP cables (Figure 3).

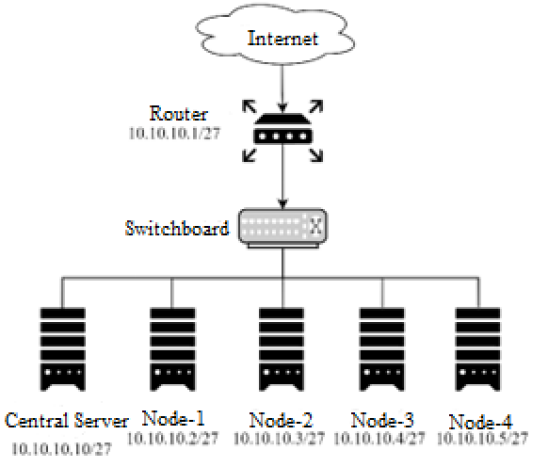

Figure 3. Network topology with virtualization

Features of this topology:

1. Centralized management and control: the central server manages the operation of the peripheral nodes, which allows you to maintain a single point of control over all virtualization processes, including caching, load balancing and redundancy.
2. Scalability through virtual machines: new virtual machines can be easily added to edge nodes to expand the system as demand grows. Virtual machines can quickly adapt to changes in workload, providing high flexibility.
3. Load balancing and redundancy: the central server can distribute requests among the peripheral nodes to efficiently use resources and reduce the risk of overload. If one of the nodes fails, requests are automatically redirected to other nodes, which ensures high reliability.
4. Flexible resource management: virtualized nodes can perform different functions (caching, query processing, data storage) depending on the load, allowing you to customize the system to meet current requirements. The central node monitors and allocates resources between nodes according to current needs, optimizing system performance.
5. Isolation and security: Each node of the star performs its tasks in isolation, which increases the security of the system. In case of a threat to one node, the others remain protected, reducing the likelihood of failure propagation.

*Corresponding Author: Volodymyr Kozub

This configuration ensures efficient processing of requests on all nodes of the system, reducing the likelihood of restrictions and increasing overall system performance.

To implement the caching system, key software components were installed and configured on each server. The Squid proxy program was deployed on each virtual machine to cache and process client requests through a proxy server, which provides fast access to popular content and distributes the load across multiple servers. Using local caches at the level of each virtual machine, the system is able to respond quickly to repetitive requests, which reduces latency and optimizes the use of network and server resources.

In addition, Squid supports bandwidth optimization functions that reduce the amount of traffic transmitted between clients and primary servers, ensuring efficient use of network resources. In the event of a failure of one of the virtual machines, the others continue to serve requests, ensuring system continuity and increasing its availability. This distributed architecture also makes it easy to scale: as the number of users or traffic grows, you can easily add new virtual machines with Squid configured to quickly increase overall throughput.

This approach not only improves the reliability and resilience of the system, but also provides greater flexibility in configuring cache policies for individual content segments, which allows you to optimize application performance in dynamic web environments. As a result, using Squid as part of distributed caching is becoming one of the most effective methods of increasing the performance and adaptability of web applications, allowing you to quickly respond to changes in traffic volume and maintain stability even during peak loads.

Squid was chosen for its flexibility and support for both centralized and distributed caching environments. In addition to Squid, an Nginx web server was configured on each server to act as a reverse proxy and load balancer.

The ISO domain serves as a repository that contains the necessary ISO images needed to install operating systems on virtual machines. In this study, CentOS 7 was selected and the ISO was installed on oVirt node-1. The ISO installation process ensures that the same operating system environment is installed on each server node. This ensures consistency across all virtualized instances..

Additionally, a reverse proxy setting was implemented as an intermediary between client requests and web servers. The reverse proxy helps to distribute traffic efficiently. It also ensures that user requests are directed to the correct server, optimizing response time and ensuring uninterrupted operation.

The system includes configurations for both forward and reverse DNS zones. The forward zone is responsible for mapping domain names to IP addresses when a client makes a request. This process allows the server to find the appropriate IP address associated with a particular domain. Reverse zone, on the other hand, resolves IP addresses back to domain names. This is necessary for registration, tracking, and ensuring accurate processing of client requests.

For load balancing and caching, an Nginx server was installed on each virtual machine. It functions as a reverse proxy, HTTP cache, and load balancer, playing a key role in managing web traffic. The configuration uses the ip_hash algorithm, a static scheduling method that ensures that requests from the same client are consistently directed to the same server. This method is effective for maintaining session persistence and processing large volumes of requests, as it evenly distributes the load between servers.

Figure 4 illustrates the use of the IP hashing method in the Nginx configuration. It shows how the algorithm assigns client requests to specific servers, maintaining consistency and reducing latency.

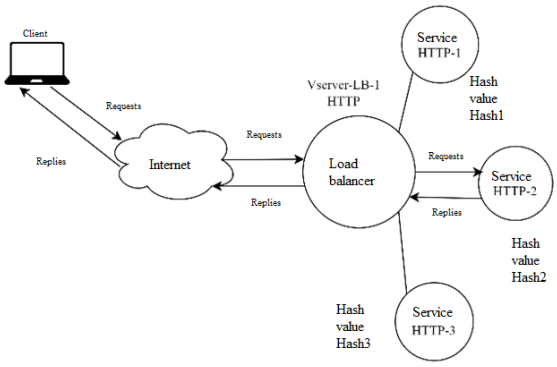

Figure 4. Visualization of the hashing mechanism

The combination of DNS, reverse proxy, and Nginx configuration allows the system to efficiently manage traffic, distribute the load between multiple proxy servers, and provide fast domain name and IP address resolution.

3. Results and Discussion

Figure 5 shows the results of the availability testing of a distributed caching system with 3000 concurrent users. This graph illustrates how the distributed system maintains high availability despite the increase in the number of user requests.

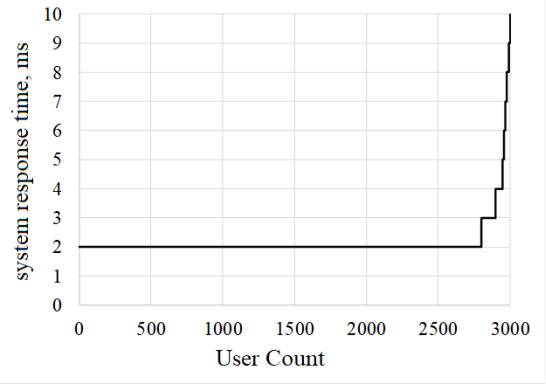

Figure 5. Results of testing the availability of a distributed system for 3000 users

*Corresponding Author: Volodymyr Kozub

At the same time, system availability remains unchanged for all 3000 users, demonstrating the effectiveness of the failover mechanism. The distributed architecture eliminates the problem of a single point of failure, ensuring that backup servers seamlessly take over when needed. The system's ability to handle a large number of requests without downtime is a clear advantage over centralized systems.

Figure 6 shows the results of testing the reliability of a distributed cache system for a different number of users.

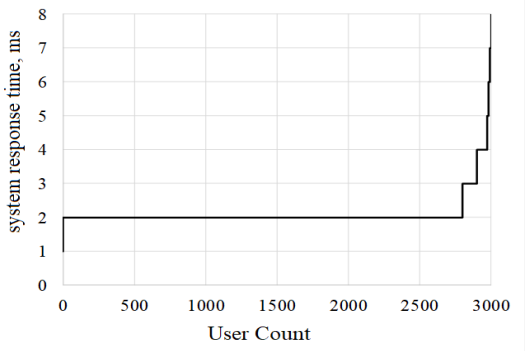

Figure 6. Results of reliability testing of a distributed system for 3000 users

The reliability of the system implies its ability to consistently process user requests without errors and delays. In a distributed system, proxy servers work in parallel. This means that several servers can process different requests at the same time, reducing the likelihood of overloading any single server.

The graph shows that the distributed system continues to process requests efficiently even under heavy loads. The lower failure rate compared to centralized systems can be explained by the load balancing features of the distributed architecture, where incoming requests are distributed among several nodes. This distribution not only increases fault tolerance but also improves the overall performance of the cache system.

Figure 7 shows the results of scalability testing of a distributed system, this time with 5000 users. Scalability refers to how well a system can expand to handle increasing traffic without sacrificing performance. In distributed caching systems, additional proxies can be added dynamically to handle more traffic.

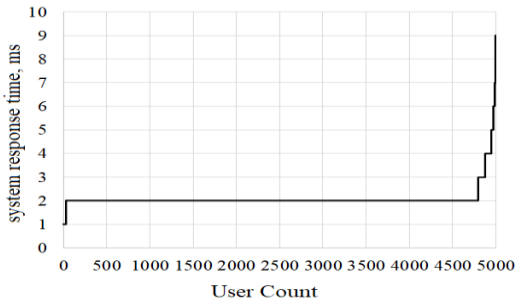

Figure 7. Scalability testing results of a distributed system for 5000 users

The graph demonstrates that the distributed caching system scales efficiently, maintaining low response times and minimizing errors as the number of users increases. This performance is achieved thanks to a flexible architecture that makes it easy to integrate new servers into the network. The distributed cache can grow with user demand, making it highly adaptable to web environments with fluctuating traffic.

In this case, the connection was lost. The single proxy server failed because the system had no backup or failover mechanism. This highlights a major drawback of centralized caching systems - they rely entirely on a single server to handle all traffic. If this server goes down, the entire system becomes unavailable and all user requests go unprocessed.

Figure 8 shows the results of reliability testing of a centralized system with 3000 users. The centralized proxy server tries to maintain reliability under high load, which leads to delays and an increase in the number of errors. When one server handles all requests, the system quickly becomes overloaded, which leads to poor performance.

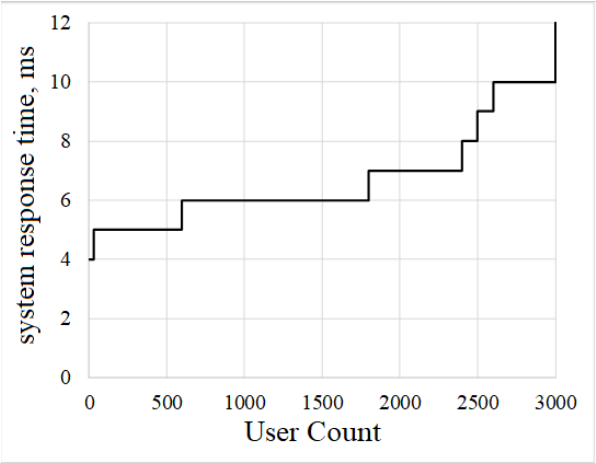

Figure 8. Reliability testing results of a centralized system for 3000 users

As the number of users grows, centralized caching faces significant challenges, as seen in the graph. This confirms the idea that centralized caching is less suitable for environments with heavy or unpredictable traffic. Figure 9 shows the results of testing the scalability of a system built on a centralized architecture and serving 5000 users. As the number of users increases, the centralized caching system cannot scale effectively. A single proxy server becomes overloaded, resulting in slower response times and more frequent errors. Unlike distributed systems, where additional servers can be connected to manage growing traffic, centralized systems are limited by the capabilities of a single node.

*Corresponding Author: Volodymyr Kozub

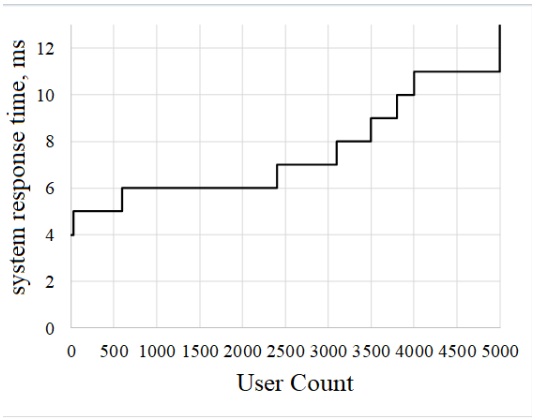

Figure 9. Scalability testing results of a centralized system for 5000 users

The graph shows a sharp drop in performance with an increase in the number of users, which emphasizes the main drawback of centralized caching - it is not designed to scale effectively. The inability to adapt to growing demand makes centralized systems less favorable for modern web services. After creating centralized and distributed caching systems, experiments were conducted to compare their performance in terms of availability, reliability, and scalability.

The first comparison focused on availability. In a centralized caching system, a single point of failure posed a significant risk. When the primary server went down, the entire system became unavailable because there were no backup servers to replace it. This led to significant downtime and negatively impacted overall system performance. In contrast, a distributed caching system worked much better in this regard. By distributing the cache across multiple servers, the system could continue to serve requests even when one or more servers went down. This failover mechanism ensured minimal disruption and significantly increased availability. This is evidenced by the significantly lower average time of the distributed system. Detailed results can be seen in Table 1.

Table 1. Results of system comparison (experiment 1)

Parameters	Centralized (average) / ms	Distributed (average) / ms
Availability	x	2
Reliability	7	2
Scalability	7	2

When assessing reliability, the centralized system experienced difficulties with high traffic loads. As the number of requests increased, the single server became overloaded, resulting in longer response times and an increase in the number of errors. This limitation was inherent in the centralized architecture, where all requests were processed by a single node, creating a bottleneck. On the other hand, the distributed system was more efficient in handling the same traffic load. The ability to distribute requests across multiple nodes allowed for better load balancing, faster response times, and increased overall system reliability. The distributed system consistently outperformed the centralized system both in terms of response time and error rate (Table 2).

Table 2. Results of system comparison (experiment 2)

Parameters	Centralized (average) / ms	Distributed (average) / ms
Availability	X	4
Reliability	9	3
Scalability	7	5

Scalability was another key area where the distributed system showed clear advantages. With a centralized caching system, scaling the system to serve a growing number of users was a challenge. As the number of users increased, a single server quickly reached its capacity, and upgrading the system required a significant investment in hardware. In contrast, the distributed system was designed to be easily scalable by adding additional cache nodes. This flexibility allowed the distributed cache to maintain consistent performance even as the number of users increased. Simply by adding more servers to the network, the system could handle the growing demand without significantly increasing response time or decreasing performance (Table 3).

Table 3. Results of system comparison (experiment 3)

Parameters	Centralized (average) / ms	Distributed (average) / ms
Availability	X	5
Reliability	11	3
Scalability	9	4

The experiments demonstrated that the distributed caching system was more efficient than the centralized cache in all tested aspects - availability, reliability, and scalability. The ability of the distributed architecture to effectively handle server failures, balance traffic loads, and scale makes it a more reliable and flexible solution for environments with high performance and fault tolerance requirements. These results confirm the theoretical advantages of distributed caching systems and emphasize their practical advantages over centralized approaches.

4. Conclusion

Traditional centralized caching systems often face significant limitations, such as single points of failure, performance degradation under high traffic loads, and limited ability to scale in response to growing user demands. These issues can lead to limitations, reduced user satisfaction, and increased latency. This hinders the operation of modern web applications that require continuous availability and high response speed. The proposed distributed caching method effectively solves these problems by using a network of interconnected proxies that jointly manage cached data. This approach has several important advantages.

*Corresponding Author: Volodymyr Kozub

First, the distributed nature of the caching system ensures fault tolerance, even if one or more servers fail or are overwhelmed by requests, other servers in the network can seamlessly take over, maintaining service availability. This redundancy significantly reduces the risk of service interruptions, providing a more stable user experience.

Second, scalability is significantly improved by distributed caching. Unlike centralized systems that require significant structural changes to accommodate growing loads, distributed caching allows you to add new proxy nodes with minimal reconfiguration. This flexibility allows web applications to scale horizontally, dynamically adapting to changes in user traffic and ensuring that response times remain low even during peak periods.

In addition, distributed caching improves load balancing in the network. By intelligently distributing requests across multiple servers, the system prevents any one node from becoming a performance limitation. This not only improves overall application throughput, but also optimizes resource utilization by reducing the load on individual servers and minimizing the likelihood of server overload.

The experimental results presented in this study confirm the effectiveness of the distributed caching method, showing a marked improvement in response time and system uptime compared to centralized caching. The distributed model proved to be particularly effective in scenarios with high user demand, where the ability to distribute data and processing load across multiple nodes resulted in smoother performance and lower latency.

Consequently, the shift from centralized to distributed caching is a solid solution for web applications looking to achieve higher levels of reliability, scalability, and efficiency. As digital ecosystems continue to expand, the demand for scalable, high-performance solutions will only increase, making distributed caching an important component in the evolution of web application infrastructure. Future research will focus on exploring methods to optimize distributed caching, such as adaptive caching algorithms and machine learning-based load prediction. This helps to improve the performance and efficiency of distributed systems.

References

1. Ghoreishi, S. E., Karamshuk, D., Friderikos, V., Sastry, N., Dochler, M., & Aghvami, A.H. (2020). A cost-driven approach to caching-as-a-service in cloud-based 5G mobile networks. *IEEE Transactions on Mobile Computing*, *19*(5), 997-1009. https://doi.org/10.1109/TMC.2019.2904061
2. Malavolta, I., Chinnappan, K., Jasmontas, L., Gupta, S., & Soltany, K.A.K. (2020). Evaluating the impact of caching on the energy consumption and performance of progressive web apps. In *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems (MOBILESoft '20)* (pp. 109-119). New York: Association for Computing Machinery. https://doi.org/10.1145/3387905.3388593
3. Mesbahi, M. R., Rahmani, A. M., & Hosseinzadeh, M. (2018). Reliability and high availability in cloud computing environments: a reference roadmap. *Human-Centric Computing and Information Sciences*, *8*, 20. https://doi.org/10.1186/s13673-018-0143-8.
4. Naeem, M. A., Rehman, M. A. U., Ullah, R., & Kim, B. -S. (2020). A comparative performance analysis of popularity-based caching strategies in named data networking. *IEEE Access*, *8*, 50057-50077. https://doi.org/10.1109/ACCESS.2020.2980385.
5. Pasyeka, M., Sheketa, V., Pasieka, N., Chupakhina, S., & Dronyuk, I. (2019). System analysis of caching requests on network computing nodes. In *2019 3rd International Conference on Advanced Information and Communications Technologies* (pp. 1-5). New York: IEEE. https://doi.org/10.1109/AIACT.2019.8847909.
6. Poularakis, K., Iosifidis, G., Argyriou, A., Koutsopoulos, I., & Tassiulas, L. (2019). Distributed caching algorithms in the realm of layered video streaming. *IEEE Transactions on Mobile Computing*, *18*(4), 757-770. https://doi.org/10.1109/TMC.2018.2850818
7. Santana, C., Andrade, L., Delicato, F. C., & Prazeres, C. (2020). Increasing the availability of IoT applications with reactive microservices. *Service Oriented Computing and Applications*, *15*(2), 109-126. https://doi.org/10.1007/s11761-020-00308-8
8. Tian, H., Xu, X., Lin, T., Cheng, Y., Qian, C., Ren, L., & Bilal, M. (2021). DIMA: Distributed cooperative microservice caching for internet of things in edge computing by deep reinforcement learning, *World Wide Web*, *25*, 1769-1792. https://doi.org/10.1007/s11280-021-00939-7.

*Corresponding Author: Volodymyr Kozub