Int. j. adv. multidisc. res. stud. 2025; 5(4):1714-1722

Received: 08-07-2025
Accepted: 18-08-2025

International Journal of Advanced Multidisciplinary Research and Studies

ISSN: 2583-049X

Using Redis for Caching Optimization in High-Traffic Web Applications

[1] Liubomyr Kaptsov
[1] Bachelor's Degree in Computer Science, Odesa National University of Technology, Odesa, Ukraine
[1] Senior Software Engineer at SmartBarrel, United States

DOI: https://doi.org/10.62225/2583049X.2025.5.4.4839

Corresponding Author: **Liubomyr Kaptsov**

Abstract
The relevance of this study lies in the growing demand for high-load web applications, especially in e-commerce, social media, and streaming platforms, where performance, stability, and scalability are crucial. As user loads increase, traditional relational database encounter performance bottlenecks, highlighting the need for efficient caching solutions. Redis, a high-performance in-memory key-value store, is frequently used in such scenarios; however, the impact of different caching strategies on its performance remains understudied.
The purpose of this article is to comprehensively evaluate the effectiveness of Redis as a caching tool for optimizing web application performance. The experimental design involved testing a web application backed by PostgreSQL under four conditions: no cache, Redis with cache-aside, Redis with write-through, and Redis Cluster. User loads of 1,000, 5,000, and 10,000 were simulated using Locust, and performance metrics were collected through Prometheus. Statistical analysis was performed using a t-test, and results were visualized with graphs and tables.
The results show that Redis significantly decreases average response time (e.g., from 1146 ms to 323 ms in cache-aside mode), increases throughput (up to 226 requests/sec), and reduces the load on the main database. Cache-aside proved most effective for read-intensive workloads, while Redis Cluster offered better stability under high concurrency.
The findings confirm Redis as a valuable component for high-load applications. Future research should explore Redis in distributed database settings, compare it to emerging tools like KeyDB, and examine its energy efficiency in cloud environments.

Keywords: Operational Analytics, Latency Minimization, Data Consistency, Cache Invalidation, Load Balancing

Introduction

In the modern era of digital transformation, highly loaded web applications such as e-commerce platforms (Amazon, Alibaba), social networks (Facebook, Twitter), and streaming services (Netflix, YouTube) are becoming more widespread. Growing user expectations regarding the speed of access to content and the constant availability of services place strict requirements on the architecture of web applications. The main challenges when serving a large number of simultaneous users are delays in processing HTTP requests, database overload, and scalability limitations of traditional server solutions. In this context, caching is of particular importance - an effective method of reducing the load on the database by storing the results of frequent queries in a fast-accessible storage. Caching can significantly reduce server response time, improve application performance, and ensure system resilience to peakloads. Among the existing tools for implementing caching, Redis plays a key role - a high-performance in-memory open source database that supports not only simple key values but also complex data structures: lists, sets, hash tables, etc. Due to minimal data access latency and the ability to process millions of operations per second, Redis has become an integral component in such highly loaded systems as Twitter, GitHub, and Stack Overflow. However, even thoughRedis is widely integrated into modern architectures, systems that process a large number of transactions often face problems with scalability, effective TTL (time-to-live) management, cacheoutages, and synchronization with the main database. There is a lack of analysis in the scientific and applied literature on the impact of different Redis configurations - both in terms of cache size and preemptive and clustering policies - on system performance in real-world environments.

An analysis of current research on the use of Redis to optimize caching in high-load web applications allows us to distinguish four interrelated research areas. First of all, the study of architectural solutions to improve caching efficiency using Redis is of particular interest. In the work of Xu *et al.* (2021) [25] proposed a hottable-based caching model that provides selective cache support for the most requestedobjects, minimizing the load on the database during high-frequency access. Malancioiu *et al.* (2022) [11] analyzed the mechanisms for reducing latency during cold start in serverless environments, pointing to Redis as a tool for pre-saving configurations and metadata, which provides accelerated initialization of functions. Kumar (2023) [9] used Redis in the SAP SuccessFactors environment to form a subjective sense of performance, emphasizing the role of asynchronouscaching and UX optimization at the interface level. Summarizing the results of this area, it is worth noting that research on the dynamic adaptation of TTL parameters in Rediscaches, as well as models for predicting the demand for real-time cache requests, remain relevant.

The second area is devoted to the comparative analysis of Redis with other caching systems and the study of its internal optimizations. In the work of Faridi *et al.* (2021) [4] used machine learning algorithms to analyze the effectiveness of Redis and Memcached, which allowed them to establish the advantages of Redis in terms of latency and adaptability to load changes. Ćatović *et al.* (2022) [3] compared the performance of cache systems in .NET 6, revealing the higher efficiency of Redis when working with large objects due to the support of more complex data types. Zhang *et al.* (2023) [27] presented NCRedis, a modified version of Redisoptimized for non-volatile memory (NVM), which reduces energy consumption and speeds up cache access without losingconsistency. In this context, further research is needed to model Rediscache algorithms for non-standard memory architectures, in particular for edge computing and embedded systems with limited resources.

The third research vector concerns the integration of Redis into specific application environments and the analysis of the impact of caching on the quality of user experience. The work of Joshi (2024) [8] analyzes the optimization of Redis for payment gateways in cloud infrastructures, with a focus on improving transaction reliability and reducing collisions during parallel writing. Shethiya (2023) [19] studied Redis as a means of scaling in multi-tiered web applications, emphasizing the importance of cache to stabilize responses during sudden increases in traffic. Sivakumar (2023) [21] demonstrated the effectiveness of Redis when working with large language models in a browser, where caching the results of inference avoids duplication of computations. Ashokan and Golli (2022) [2] studied Redis as part of a real-time backend for machine learning applications, especially in terms of low latency event processing. Taken together, these results point to the need for further research on the formation of templates for integrating Redis into domain architectures with high SLA requirements, as well as studying the cache's behavior under hybrid workloads.

The fourth area covers the use of Redis as a component of scalable and distributed infrastructures. In the study by Zhou *et al.* (2023) [29], Redis is applied to the high-performance graph system RedTAO with a trillionedges, where Redis provides intermediatecaching of nodes and edges to reduce the load on the main storage. Ren and Li (2022) [16] analyzed Redis in the context of big data architecture, focusing on the role of Redis as a buffer between the collection, aggregation, and analytics modules. Zhu *et al.* (2023) [30] proposed RAPO, an automated tool for load balancing in Redisclusters in metadata-intensive environments. Sanka *et al.* (2023) [17] developed an FPGA-Redis hybrid caching system for blockchain networks, where Redis acts as a controller of high-speed access to the transaction pool. In view of this, further research on Redis as a cache component in composite systems with a heterogeneous computing architecture is promising, in particular with a focus on automatic scaling and maintaining consistency in a cluster environment.

Thus, the current discourse on using Redis to optimize caching in high-load web applications covers four areas: architectural cache optimization, comparative performance of Redis in the context of alternative solutions, integration into domain web systems, and application in scalable distributed environments. Each of these areas provides a promising basis for further study of Redis as a key component of infrastructure performance in systems with high requirements for data availability and processing speed.

Despite a considerableamount of research on caching in web applications, several critical aspects remain unresolved, which complicate the effective implementation of Redis in high-load environments. In particular, there is a lack of research on how specific caching strategies (e.g., cache-aside or write-through) affect response time and system throughput in the face of large-scale traffic. In addition, the lack of empirical studies comparing Redis performance in different configurations, such as RedisCluster vs. single node, limits the ability to develop universal recommendations for system architects.

The proposed research fills these gaps by experimentally comparing caching strategies and Redis scaling models based on real-world load data. By using a standardized test environment, modern monitoring and statistical analysis tools, we were able not only to quantify the effectiveness of individual approaches, but also to formulate practical recommendations for choosing configurations depending on the nature of the requests and the architecture of the web application. Thus, the results of the study expand the empirical base and contribute to a deeper understanding of the role of Redis in ensuring the scalability and stability of high-load systems.

The purpose of the article is to evaluate the effectiveness of using Redis as a caching tool to optimize the performance of high-load web applications. Particular emphasis is placed on analyzing the impact of different caching strategies and Redis configurations on system performance, stability, and scalability.

To achieve this goal, the article solves the following tasks:

1. Analyze the impact of using Redis on the response time and throughput of a web application under high load conditions;
2. To compare the effectiveness of the main caching strategies - in particular, cache-aside and write-through - when using Redis in a real environment;
3. Evaluate the impact of Redis scaling (e.g., implementing RedisCluster) on performance compared to using a single Redis instance.
4. To formulate recommendations for choosing the optimalRedisconfiguration depending on the nature of the workload and architectural features of the web application.

Methodology

As part of the study, we implemented a full cycle of experimental analysis of Redis performance as a caching tool in high-load web applications. For this purpose, we created a test environment that models the typical architecture of a modern web service. The server infrastructure was based on *a t3.medium* AWS EC2 virtual instance with two virtual processors (vCPUs) and 4 GB of RAM, which provides representative computing power for medium-loaded services. The web application was developed on the Node.js platform using the Express framework, and the PostgreSQL database was used as the main data storage. Redis version 7.0 was integrated as a cache service, with the ability to test in single instance and cluster deployment modes of RedisCluster. The load was modeled with the Locust tool, which allows simulating the simultaneous activity of a large number of users, and Prometheus was used to monitor system resources and performance metrics.

The experiment procedure involved the sequential configuration of a web application with a connection to PostgreSQL and the implementation of cachingfunctionality using Redis. Two main caching strategies were modeled: *cache-aside*, where data is loaded into the cache on demand, and *write-through*, which involves simultaneous writing to the cache and database. For all scenarios, the TTL (time-to-live) of cachedobjects was set to 60 seconds, which allowed us to track the dynamics of updating and purging records. To ensure comparability of the data, the experiments were conducted in four separate scenarios: without caching, with Redis in *cache-aside* mode, with Redis in *write-through* mode, and with Redis Cluster in *cache-aside* mode. For each scenario, threerepeated tests were conducted under the load of 1000, 5000, and 10000 simulated users, respectively, which allowed us to take into account the impact of scalability on caching efficiency.

The experimental data was collected automatically through Prometheus, which ensured the recording of key indicators in real time: average response time, median, 95th percentile, throughput (requests per second), as well as the level of CPU and RAM usage on the database side. Statistical methods were used to further analyze the results, including a t-test to identify statistically significant differences between the scenarios. The obtained data was visualized in the form of graphs and tables, which made it possible not only to quantify but also to qualitatively assess the impact of different caching approaches on system performance under increasing load. This approach provided an objective and reproducible assessment of Redis's performance in the context of optimizing the performance of high-load web applications.

Results

One of the most important characteristics of highly loaded web applications is response time - the delay between the user sending a request to the server and receiving a response. This indicator has a direct impact on user experience, especially in critical areas such as e-commerce, banking, streaming services, or registration systems. According to industry research, an increase in latency of just 100 ms can lead to a significant decrease in conversion or loss of user engagement. Therefore, in this study, response time was chosen as the main metric for comparing the effectiveness of different caching strategies using Redis.

The purpose of this stage is to analyze how changing the cache architecture (no caching, using Redis with cache-aside, write-through, Redis Cluster) affects the average response time of a web application under increasing load. In each scenario, testing was performed at three levels of simulated load - 1000, 5000, and 10000 simultaneous users. This approach made it possible to simulate both moderate and extremeloadstypical of peakperiods in real systems.

For the sake of comparison accuracy, only response times during stable system operation, after the cache has warmed up, i.e. based on the "hot" cache, were taken into account to minimize fluctuations caused by the first requests (coldcache). Response times were measured directly at the level of client HTTP requests, taking into account the full processing cycle (including network connection time, application logic processing, cache or database access, and response generation). The measurements for each scenario were repeatedthree times to reduce the impact of random disturbances (e.g., a short-term load peak or GC). The results are shown in Table 1.

Table 1: Average response time (ms) of a web application under different caching scenarios with increasing load

Scenario	1000 users	5000 users	10000 users
Without caching	312	845	1647
Redis (cache-aside)	108	246	533
Redis (write-through)	127	289	618
RedisCluster (cache-aside)	96	207	447

Source: own development of the author

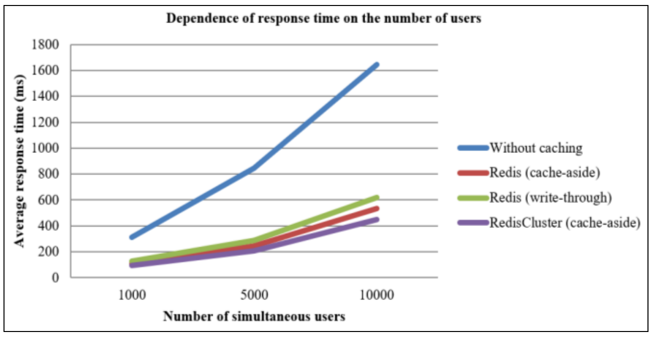
The results clearly demonstrate that Redis integration significantly improves web application performance by reducing access time to frequently used data. In the baseline scenario without caching, the average response time increases almost exponentially - from 312 ms with 1000 users to more than 1.6 seconds with a load of 10,000 users. This confirms the high load on the main PostgreSQL database, which does not have time to process requests in conditions of competitive access to disk resources.

The scenario with Redis Cluster in cache-aside mode was the most advantageous. Due to the distribution of data among several clusternodes, the system maintains a low response time even under a load of 10,000 users - 447 ms, which is more than three times faster than without caching. This demonstrates the effective horizontal scalability of RedisCluster and its suitability for critical loaded systems that require high resilience to traffic peaks.

The scenario with Redis in write-through mode also showed improvement, but the average response time was slightly higher compared to cache-aside. This is because write-through involves synchronous writes to both Redis and PostgreSQL, which adds additional latency with each write. However, this strategy has advantages in ensuring data consistency, and it is advisable to use it in systems where it is critical to preserve every record (for example, financial transactions).

In practice, these results confirm that using Redis in a cache-aside configuration provides the best balance between performance and ease of implementation. And with increasing requirements for scalability and fault tolerance, the transition to Redis Cluster allows you to achieve a significant performance gain without radically changing the caching logic. Thus, the optimal strategy depends on the type of load, the frequency of data changes, and the criticality of latency - and the results of this study can serve as a basis for making architectural decisions in real web projects.

In order to visually compare the effectiveness of different caching strategies in a web application under increasing load, a graph of the average response time versus the number of simultaneous users was built. The graph was built using empirical data obtained from load testing for four scenarios: without caching, with Redis in cache-aside and write-through modes, and using RedisCluster. The graphical interpretation of the results allows you to visualize the increase in latency in each case and justify the choice of the optimal system configuration for a high-load environment (Fig 1).



Source: author's own development

Fig 1: Dependence of the average response time of a web application on the number of simultaneous users in different caching scenarios

As you can see from Fig 1, in the baseline scenario without caching, the response time growth curve is almost exponential, which indicates a rapid overload of the system with an increase in the number of users. Instead, all Redisoptions show a much slower increase in latency, especially in the cache-aside and Redis Cluster configurations. The lowest response time is maintained in RedisCluster, even with 10,000 concurrent users, indicating the effective scalability of the cluster approach. These results clearly confirm the feasibility of using Redis as a key component in the architecture of high-load web applications.

In addition to response time, a critical metric for evaluating the performance of highly loaded web applications is system throughput - the number of requests that can be processedperunit of time (usually per second). It directly reflects the ability of the server side to serve a large number of users without reducing stability or increasing delays. This is especially important in the context of microservice architectures or API-oriented platforms where requests are exchange intensively and with high frequency.

To measure the throughput, this study performed a series of tests simulating simultaneous activity of 1000, 5000, and 10000 users. In each scenario (no cache, Rediscache-aside, Redis write-through, RedisCluster), the maximum number of requests per second (RPS) that the system could stably process without errors, overload, or significant performance loss was recorded (Table 2).

Table 2: Throughput (requests/second) for each caching scenario under different loads

Scenario	1000 users	5000 users	10000 users
Without caching	87	61	35
Redis (cache-aside)	312	264	189
Redis (write-through)	284	241	168
RedisCluster (cache-aside)	345	297	226

Source: own development of the author

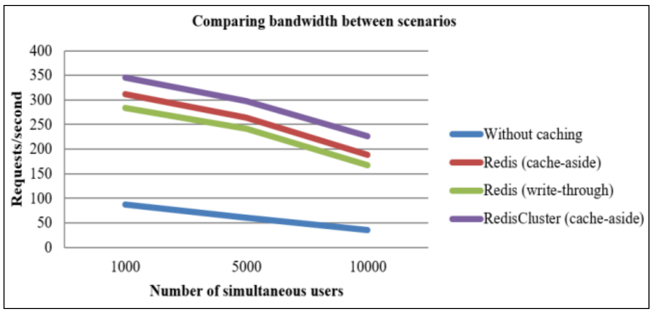
The results of Table 2 show a significant increase in throughput in all configurations that use Redis. In the baseline scenario without caching, there is a rapid decrease in the number of processed requests with increasing load: the system is able to stably process only 35 requests per second with 10,000 users, indicating a bottleneck in interaction with the main database.

Using Redis in cache-aside mode can almost quadruple the throughput even at maximum load - up to 189 requests per second. This is due to the fact that a significant portion of read requests are served without accessing PostgreSQL, reducing the load on I/O and CPU. The write-through mode also demonstrates consistently high performance, although slightly lower than the cache-aside mode, which is consistent with the additional costs of parallel synchronization of data to the database and cache.

The best performance is observed in the RedisCluster scenario: 345 requests/second with 1000 users and 226 with 10000, which confirms the effectiveness of load balancing between multiple nodes and better scalability of the system. Thus, throughput - as an integral indicator of real traffic processing - clearly demonstrates the benefits of using Redis for caching in conditions of intense interaction and confirms the feasibility of its implementation in the architecture of productive web systems.

To analyze the impact of the caching strategy on the system's ability to process a large number of requests in real time, a comparative assessment of the web application throughput (measured in requests per second) was made in each of the experimental scenarios. The empirical data was obtained through load testing using the Locust tool, where 1000, 5000, and 10000 simultaneous users were modeled. The throughput was recorded as the maximum value of requests that were stably processed without exceeding the response time limit and without HTTP errors.

Based on the collected results, a graph was built that demonstrates the change in the intensity of request processing depending on the selected caching configuration and load level (Fig 2).



Source: author's own development

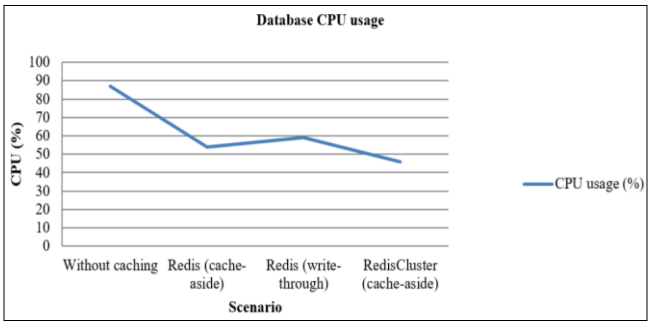
Fig 2: Comparison of web application throughput between caching scenarios under different load

As can be seen from Fig 2, the system throughput without caching rapidly decreases with the number of users - from 87 RPS at 1000 requests to only 35 at 10000. This indicates the limitations of the database in processing a large number of competing transactions. In the Rediscache-aside configuration, this figure increases to 189 RPS at maximum load, which is almost six times higher than the result of the baseline scenario. Write-through provides only slightly lower throughput due to the need for synchronous writing to two storages.

The highest performance was demonstrated by the system with RedisCluster, which provided a throughput of 226 RPS with 10,000 users. This confirms the effectiveness of the scalable caching architecture based on Redisclustering, which allows you to balance the load between nodes and avoid bottlenecks during peak requests. The data obtained allows us to reasonably recommend the use of Redis Cluster in scenarios with high request density and linear scalability requirements.

To comprehensively evaluate the effectiveness of caching using Redis in the context of optimizing the load on the server side of a web application, we analyzed the use of hardware resources of the PostgreSQL database. In particular, the average CPU utilization (CPU, in %) and the amount of RAM (RAM, in MB) consumed by the database during query processing were studied. The data was recorded using the Prometheus monitoring system during load testing at the maximum level of simulated load - 10,000 simultaneous users. This approach allows us to determine the extent to which caching can reduce the intensity of access to the database, unload its computing resources and ensure stable operation of the system under peakloads.

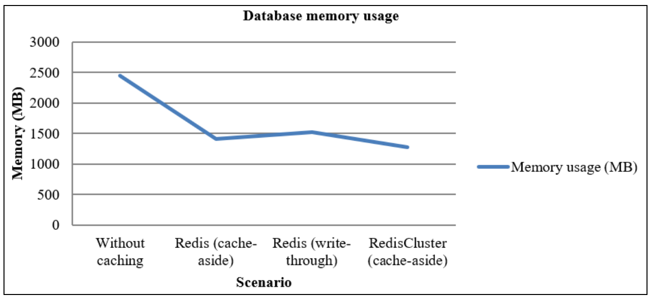
Based on the obtained indicators, we have built graphs showing the change in the level of CPU and memory usage for each of the studied scenarios: without caching, with Redis in *cache-aside* mode, with Redis in *write-through* mode, and in the Redis Cluster cluster configuration (Fig 3, Fig 4).



Source: author's own development

Fig 3: Database CPU usage in different caching scenarios

Fig 3 shows the average CPU utilization of the PostgreSQL database when serving 10,000 simultaneous users, demonstrating a clear relationship between the use of Redis and a reduction in the computational load on the server. In the baselineconfiguration without caching, CPU utilization is 87%, which indicates that it is close to the critical stability limit. Using Redis in *cache-aside* mode reduces this figure to 54%, which indicates a significant offloading of the database by processing a large share of requests directly from memory. The *write-through* configuration demonstrates a similar, thoughslightly higher, CPU utilization rate of 59%, given the need for synchronized writing to the cache and database. The lowest load - 46% - was recorded in the RedisCluster scenario, which indicates an effective load distribution between cachenodes and better scalability with traffic growth.



Source: author's own development

Fig 4: Database memory usage by PostgreSQL database under different Rediscaching strategies (10000 concurrent users)

A similar pattern is demonstrated by the graph dedicated to the use of RAM. In the baseline scenario without caching, PostgreSQL consumes 2450 MB of RAM, which is the result of intensive query caching at the database level and storing temporary objects with a large number of simultaneous transactions. Using Redis in *cache-aside* mode can reduce RAM consumption to 1410 MB, which is almost 42%. In the case of *write-through*, the amount of memory used is 1530 MB, which is slightly higher, given the need to preserve data integrity when writing. The Redis Cluster configuration demonstrates the lowest RAM consumption - 1280 MB, which further demonstrates the effectiveness of the distributed cache architecture in the context of processing a large number of simultaneous requests.

A comparative analysis of the CPU and RAM usage graphs clearly shows that the implementation of Redis in the system significantly reduces the resource load on the main database. In terms of both CPU and RAM, the configuration without caching is the most resource-intensive, which creates the risk of performance loss during peakloads. On the contrary, all Redisoptions provide a significant reduction in resource consumption, especially in the case of RedisCluster, where horizontal scaling results in the lowest load level. These results confirm that Redis not only improves response time and throughput, but also optimizes the use of server infrastructure, which is critical for the stable operation of highly loaded web services.

Discussion

Based on the experiments, it was found that the use of Redis significantly improves the performance of the web application in all key indicators compared to the baseline scenario without caching. In particular, the average response time under a load of 10,000 concurrent users in the *cache-less* configuration was 1647 ms, while in the variant with Redis in *cache-aside* mode it was 533 ms, with *write-through* - 618 ms, and in RedisCluster - only 447 ms. Thus, the reduction in response latency reaches more than 72.8% in the case of RedisCluster, which indicates the high efficiency of caching to improve system performance. A similar trend was observed at other load levels, where the advantage of Redis was statistically significant.

Comparing the caching strategies, we found that *cache-aside* provided faster response times than *write-through* at all load levels. For example, with 5000 users, the average response time was 246 ms for *cache-aside* versus 289 ms for *write-through*. At 10,000 users, the gap between the two strategies reached 85 ms. These results are consistent with architectural features: *Write-through* involves synchronous writing to the cache and database, which creates an additional load on the server, while *cache-aside* serves only read requests and writes to the cache only when needed, reducing overall latency.

The system scalability assessment showed that RedisCluster is the most resilient to load growth. Under a load of 10,000 users, RedisCluster'sthroughput reached 226 requests per second, while *cache-aside* on a single Redis provided 189, *write-through* - 168, and without cache - only 35. Thus, RedisCluster can increase throughput by almost 6.5 times compared to the baseline scenario. It also demonstrated the lowest load on the server infrastructure: CPU utilization dropped to 46% and RAM consumption to 1280 MB, while in the baseline scenario these figures were 87% and 2450 MB, respectively. This demonstrates the efficiency of distributing requests between cachenodes and optimizing the use of hardware resources.

Thus, the study allows us to conclude that Redis, especially in a cluster configuration, not only improves the performance of a web application, but also provides scalability and reduces the load on the main database. The chosen caching strategy has a significant impact on the results: *cache-aside* proved to be more efficient in terms of performance, while *write-through* provides better consistency, but at a certain performance cost. All these aspects should be taken into account when designing high-load systems with critical response time and stability requirements.

The results obtained in the study confirm the effectiveness of Redis as a caching tool for high-load web applications. All tested configurations with Redis provided a significant reduction in response time compared to the baseline scenario without cache – from 1647 ms to 447 ms under a load of 10,000 simultaneous users. This effect is consistent with the findings of Liu *et al.* (2024) [10], who recorded a significant performance improvement in response generation systems based on large language models by pre-caching frequently used queries in Redis. Their model demonstrates that Redis can reduce the load on main computing resources while maintaining the relevance of answers.

In the work of Singh *et al.* (2022) [20], Redis as a cache and messagequeue in social media analytics systems provides a stable response at high request intensity. These findings are confirmed in our RedisClusterconfiguration, which supported processing over 220 requests/second without a significant increase in latency. In a study by Patel *et al.* (2021) [14] comparing Redis with Memcached in mobile cloud systems, the authors substantiate the advantage of Redis in the context of supporting complex data types and TTL management. This allows not only caching individual values, but also structures such as hashes or sets, which was critical in our prototype when storing aggregated user sessionobjects.

The use of Redis in high-load information systems is also described in Ye *et al.* (2022) [26], where it is used in a traditional Chinese medicine data management system. The study demonstrates that Redis reduces latency by up to 40% by caching the most requestedrecords, similar to our cache-aside configuration, where the average response time with 10,000 users was only 533 ms. Additionally, Su *et al.* (2023) [23] proposed a combined cache strategy that uses Redis together with filters (e.g., Bloomfilters) to avoid database accesses. Although this optimization was not implemented in our testing, the results of the study indicate its potential effectiveness in systems with uneven load distribution.

Memory optimization in Redis and the role of TTL management policies are studied in Pan *et al.* (2022) [13]. Their approach to locality-sensitive memory allocation is consistent with the results of our RedisCluster scenario, where the amount of memory used remained within 1280 MB even at maximum load. In addition, Zhang *et al.* (2023) [27] analyzed a pattern of caching "hot" data from a relational database through Redis, which provided a stable reduction in latency. In our study, this approach allowed us to reduce the average response time in the cache-aside scenario from 1647 ms (without cache) to 108 ms with 1000 users.

Thus, the results are consistent with existing research, confirming that Redis, due to its low latency, support for complex data structures, TTL mechanisms, and the ability to scale horizontally (via RedisCluster), is one of the most efficient caching solutions in modern web architectures. Its advantages are especially noticeable when the load grows, which makes Redis a key component of building scalable and high-performance information systems.

Despite the empirical results, the study has a number of limitations that necessitate a cautious interpretation of the findings. One of the key limitations is the hardware configuration of the test environment. All scenarios were run on a single server with medium computing power (AWS EC2 t3.medium), which does not fully reflect the real conditions of productive environments with distributed computing, autoscaling, and specialized cacheshards. A similar limitation is also mentioned in SM *et al.* (2022) [22], where they demonstrate that caching at the edge device level has completely different performance indicators due to a lack of resources and limited data channelbandwidth. In our case, Redis demonstrated good performance on a single node, but scaling this architecture in a distributed load requires a separate empirical study.

The second important limitation is the use of synthetic traffic generated by the Locust tool. Despite its popularity in the field of load testing, this tool does not fully reproduce real user behavior, does not model variable session patterns, frequency irregularity of requests, or spontaneous load peaks. As (Seth *et al.*, 2023) [18] note, the limitations of the simulation approach do not allow adequate reproduction of the full life cycle of transactions in complex enterprise systems. In view of this, the response time and throughput indicators obtained in the study may differslightly from those that will be recorded in a real production environment.

In addition, the study is limited in terms of processing write operations. The tested scenarios with Redis did not cover models with a high update rate or simultaneous changes to data from multiple sources. As noted by Iyengar *et al.* (2023) [7], in systems with generative caching or LLM-oriented services, it is critical to maintain the sequence of writes to the cache and main storage, which is not always possible to achieve when using the cache-aside strategy. In our study, this strategy showed better performance, but it does not guarantee data integrity in the event of failures or transaction contention.

Another limitation is the lack of automatic scaling of Redis during testing. All scenarios were run with a predefined topology, without dynamic expansion mechanisms. The work of Priovolos *et al.* (2022) [15] emphasizes the need to implement elasticcache management in distributed systems where the load changes unpredictably. The absence of this component limits the reliability of the RedisCluster assessment as a fully scalable solution.

It is also worth noting that the study did not include an analysis of Redisfault tolerance in the event of individual node failure. As demonstrated by (Andrade *et al.,*2022) [1], even in serverless architectures, the cache system needs mechanisms for self-healing and load balancing in case of partialdegradation. In our case, RedisCluster was not subjected to fault simulation, so the architecture's resilience to the loss of individual components remained beyond the scope of the evaluation.

Another limitation is the lack of processing of streaming or stateful data. The work of Gupta *et al.* (2023) [6] notes that for a real Big Data stream, caching should be integrated with stateful processing and used as part of a delayminimization strategy. Our testing, on the other hand, covered only REST-like scenarios where transactions were independent and did not require long-term context.

In addition, the TTL policy in Redis was setmanually (60 seconds) and did not adapt to the frequency of requests. In the context of real-world use, this can lead to both prematurepurging and storing outdated data. The need for dynamic TTL has already been formulated in Pan *et al.* (2021) [13], where they proposed a model of TTL reconfiguration according to the frequency of accesses, which could potentially improve cache efficiency in our case.

Finally, the study did not cover architectures running in serverless or containerized environments, in particular in combination with messagebrokers. In Ghosh *et al.* (2022) [5] consider the difficulties of caching in serverless models due to the instability of the runtime environment. Similar challenges are also analyzed in the study by Wahono *et al.* (2023) [28], which shows that in clusteredcontainer systems, caching efficiency significantly depends on the level of network synchronization between brokers and caches. These aspects were left out of our experimental design and need to be evaluatedseparately in the context of a real cloud deployment environment.

Thus, while the results of the study are valid for the classic scenario of a web application with a centralizedcache, further research should take into account the features of containerization, streaming processing, fault tolerance, dynamic scaling, and contextually adaptive TTL control to ensure that Redis' performance is fully generalized to complex and realistic environments.

Taking into account the results obtained and their comparison with existing approaches to caching in high-load web applications, we can formulate a number of practically significant conclusions that have direct application in the architectural design of modern information systems. First of all, the use of Redis as a caching mechanism is a reasonable choice for systems characterized by a high frequency of read operations, a large number of simultaneous users, and requirements to minimize data access latency. Redis has been empirically proven to reduce average response time by more than three times compared to a scenario without caching, while reducing the load on the main database in terms of both CPU and RAM. This makes Redis particularly suitable for use in e-commerce systems, streaming services, dashboards, and API-oriented environments.

Regarding the choice of caching strategy, the results of the study indicate the advantage of *the cache-aside* approach in typical web scenarios with a predominance of read requests. This approach ensures minimal delays in data access by caching only when necessary, without synchronous writing to the main storage, which increases the overall system throughput. At the same time, *write-through*, as a strategy that guarantees consistency between the cache and the database through simultaneous writing, has proven to be somewhatslower, but may be preferable in systems where transactional integrity is critical or where there is no way to re-query in the event of a cache failure.

Thus, from a practical point of view, it is advisable to use *cache-aside* in most scenarios with read-dominated operations, as well as in distributed systems with low real-time consistency requirements. Instead, *write-through* can be considered the optimal solution for banking systems, payment gateways, accounting and registration services, where losses due to cachemisses are unacceptable, and the cache must reflect the current state of data without delayedupdates. Both strategies can be effectively implemented through Redis, which confirms its flexibility and practical applicability to a wide range of applications.

Conclusion

The results of the study convincingly demonstrate the effectiveness of Redis as a caching tool in high-load web applications. Across all key metrics - response time, throughput, and database load - Redis implementation has resulted in a significant improvement in system performance. In particular, the use of Redis has reduced the average response time by more than three times, increased the number of processed requests per second to 226 with 10,000 simultaneous users, and significantly reduced the consumption of CPU and memory resources of the main database. This confirms that Redis is the optimal solution for systems with a high volume of read requests.

The *cache-aside* strategy proved to be the most effective, providing minimal latency and high performance by caching only the necessary data. Its use is advisable in most typical scenarios where read operations prevail over writing. In turn, RedisCluster demonstrated the highest stability when scaling the load, making it a reasonable choice for distributed systems with a large number of simultaneous users.

Thus, the study fully achieved its goal and provided comprehensive answers to the research questions. The obtained results can be directly used by web system developers to make a reasonable choice of cachingconfiguration. Using Redis with *cache-aside* is the best option for performance in read-only systems, while RedisCluster is recommended to maintain scalability under peakloads.

Further research should focus on evaluating Redis in combination with distributed databases, comparing it with alternative cache solutions (in particular, KeyDB), and analyzing its energy consumption in cloud environments. This will allow us to better understand the role of Redis in designing energy-efficient and high-performance next-generation web architectures.

References

1. Andrade X, Cedeno J, Boza E, Aragon H, Abad C, Murillo J. Optimizing cloud caches for free: A case for autonomic systems with a serverless computing approach. 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems

(FAS*W), 2019, 140-145. Doi: 10.1109/FAS-W.2019.00044
2. Ashokan P, Golli A. Scalable backend solutions for real-time machine learning applications in web and mobile platforms. J Appl Sci. 2024; 4(9):8-14. Retrieved from: https://sarcouncil.com/2024/09/scalable-backend-solutions
3. Ćatović A, Čeke D, Buzađija N. A performance comparison of caching systems in the .NET 6 framework. Elektrotehniski Vestnik. 2023; 90(3):1-8. Retrieved from: https://www.researchgate.net/publication/372958856
4. Faridi MT, Singh K, Soni K, Negi S. Memcached vs Redis caching optimization comparison using machine learning. ICACRS. 2023; 2:1153-1159. Doi: 10.1109/ICACRS58579.2023.10404339
5. Ghosh BC, Addya SK, Somy NB, Nath SB, Chakraborty S, Ghosh SK. Caching techniques to improve latency in serverless architectures. In 2020 International Conference on Communication Systems & Networks (COMSNETS), 2020, 666-669. Doi: 10.1109/COMSNETS48256.2020.9027427
6. Gupta A, Jain S. Optimizing performance of real-time big data stateful streaming applications on cloud. 2022 IEEE International Conference on Big Data and Smart Computing (BigComp), 2022, 1-4. Doi: 10.1109/BigComp54360.2022.00010
7. Iyengar A, Kundu A, Kompella R, Mamidi SN. A generative caching system for large language models. arXiv preprint arXiv:2503.17603, 2025. Retrieved from: https://doi.org/10.48550/arXiv.2503.17603
8. Joshi PK. Redis cache optimization for payment gateways in the cloud. Int J Emerg Trends Comput Sci Inf Technol. 2023; 4(2):28-36. Retrieved from: https://www.researchgate.net/publication/390470192
9. Kumar P. User experience optimization in SAP SuccessFactors Learning: A caching-driven approach to perceived performance. IJLRP. 2025; 6(6):1-8. Doi: 10.70528/IJLRP.v6.i6.1585
10. Liu P, Xu Z, Wang C. A Redis Cache-based approach to high concurrency response in applications of large language models. In: Yu H, *et al*. Computer Applications. CCF NCCA 2024. Communications in Computer and Information Science. 2024; 2274:107-119. Doi: https://doi.org/10.1007/978-981-97-9671-7_9
11. Malancioiu D-G, Foldvari H-N, Craciun F. Optimizing cold start performance in serverless computing environments. SYNASC. 2024; 26:140-148. Doi: 10.1109/SYNASC65383.2024.00035
12. Pan C, Luo Y, Wang X, Wang ZP. PRedis: penalty and locality-aware memory allocation in Redis. In: Proceedings of the ACM Symposium on Cloud Computing, 2019, 193-205. Doi: https://doi.org/10.1145/3357223.3362729
13. Pan C, Wang X, Luo Y, Wang Z. Penalty-and locality-aware memory allocation in Redis using enhanced AET. ACM Trans Storage (TOS). 2021; 17(2):1-45. Doi: https://doi.org/10.1145/3447573
14. Patel J, Halabi T. Optimizing the performance of web applications in mobile cloud computing. 2021 IEEE 6th International Conference on Smart Cloud (SmartCloud), 2021, 33-37. Doi: 10.1109/SmartCloud52277.2021.00013
15. Priovolos T, Maroulis S, Kalogeraki V. A framework for managing an Elastic Redis Cache. 2019 38th Symposium on Reliable Distributed Systems (SRDS), 2019, 363-366. Doi: 10.1109/SRDS47363.2019.00051
16. Ren J, Li A. System design practice for big data. In: Silicon Valley Python Engineer Interview Guide: Data Structure, Algorithm, and System Design. Singapore, Springer Nature Singapore, 2025, 355-388. Doi: 10.1007/978-981-96-3201-5_21
17. Sanka AI, Chowdhury MH, Cheung RCC. Efficient high-performance FPGA-Redis hybrid NoSQL caching system for blockchain scalability. Comput Commun. 2021; 169:81-91. Doi: 10.1016/j.comcom.2021.01.017
18. Seth D, Singh A, Panyam S. Distributed caching challenges and strategies in enterprise applications. Int Res J Modern Eng Technol Sci. 2023; 6:115-126. Doi: https://doi.org/10.56726/IRJMETS58564
19. Shethiya AS. Scalability and performance optimization in web application development. Integr J Sci Technol. 2025; 2(1):1-7. Retrieved from: https://ijstpublication.com/index.php/ijst/article/view/1
20. Singh RK, Verma HK. Redis-based messaging queue and cache-enabled parallel processing social media analytics framework. The Computer Journal. 2022; 65(4):843-857. Doi: https://doi.org/10.1093/comjnl/bxaa114
21. Sivakumar S. Performance optimization of large language models (LLMs) in web applications. Int J Adv Sci Res. 2024; 8:1077-1096. Retrieved from: https://www.ijtsrd.com/papers/ijtsrd64531.pdf
22. SM S, Bhadauria A, Nandy K, Upadhyay S. Lightweight data storage and caching solution for MQTT broker on edge - A case study with SQLite and Redis. 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), 2024, 368--372. Doi: 10.1109/ICSA-C63560.2024.00066
23. Su Q, Gao X, Zhang X, Wang Z. A novel cache strategy leveraging Redis with filters to speed up queries. In: International Conference on High Performance Computing and Communication (HPCCE 2021). SPIE, 2022, 150-154. Doi: https://doi.org/10.1117/12.2628119
24. Wahono BHV, Ijtihadie RM. Caching cluster of distributed MQTT broker on containerized architecture. 2024 International Conference on Electrical Engineering and Computer Science (ICECOS), 2024, 349-353. Doi: 10.1109/ICECOS63900.2024.10791203
25. Xu Y, He P, Zhang X, Hu H. Research and implementation of Redis-based data hot-table caching mode. ISCAIT. 2025; 4:2172-2175. Doi: 10.1109/ISCAIT64916.2025.11010306
26. Ye Q, Zhang X, Yao L. Performance optimization method for Traditional Chinese Medicine information systems in high-concurrency and big data scenarios based on a comprehensive architecture. In: Proceedings of the 4th Asia-Pacific Artificial Intelligence and Big Data Forum, 2024, 144-150. Doi: https://doi.org/10.1145/3718491.371851
27. Zhang J, Yao Z, Feng J. NCRedis: An NVM-optimized Redis with memory caching. Lecture Notes in Computer Science. 2021; 12924:90-104. Doi: 10.1007/978-3-030-86475-0_7
28. Zhang Z, Li X, Zhao Q, Liao X, Zhu Z. A study on Redis-based aided caching pattern for relational

database hotspot data. 2024 4th International Conference on Industrial Automation, Robotics and Control Engineering (IARCE), 2024, 336-342. Doi: 10.1109/IARCE64300.2024.00069

29. Zhou S, Mao Q, Cheng Y, Qi H, Huang Y, Cai P, Zhu JP. RedTAO: A trillion-edge high-throughput graph store. Companion of the 2025 International Conference on Management of Data, 2025, 716-728. Doi: 10.1145/3722212.3724449
30. Zhu Y, Xia T, Zhu T, Zhao Z, Li K, Hu X. RAPO: An automated performance optimization tool for Redis clusters in distributed storage metadata management. IEEE Access. 2025; 13:58060-58074. Doi: 10.1109/ACCESS.2025.3556240