Performance Efficiency Evaluation based on ISO/IEC 25010:2011 applied to a Case Study on Load Balance and Resilient

Diego S. Pereira[1,2], Lucas F. V. Bezerra[1], Jacyana S. Nunes[1], Itamir M. Barroca Filho[1], Frederico A. S. Lopes[1]

[1]Metrópole Digital Institute (IMD)
Federal University of Rio Grande do Norte (UFRN)
Natal, Brazil.

[2]Federal Institute of Rio Grande do Norte (IFRN)
Parnamirim, Brazil.

diego.pereira@ifrn.edu.br, lucas.vital.701@ufrn.edu.br,

{jacyana, itamir.filho, fred}@imd.ufrn.br

Abstract. *The client-server architecture makes it necessary to implement techniques to overcome the single point of failure problem (only one server node). One of the most commonly used techniques is load balancing. Load balancing consists in distributing requests between nodes that provide a service (clusters). In this paper, we performed an evaluation based on the concept of efficiency included in ISO25010:2011. The balancers evaluated were HAProxy (High Availability Proxy) and ARR (Application Request Routing) with and without caching. The project requirements are limited using the IIS (Internet Information Services) web server. Load Balances based on caching presented the best performance in most scenarios, while HAProxy obtained better values related to CPU utilization.*

1. Introduction

Currently, web applications are considered one of the most widely used means of communication and data exchange. More than 2.5 quintillion bytes are uploaded to the Internet daily through different applications, which offer a wide diversity of services to more than 4.5 billion users [Sadqi and Maleh 2022]. According to a survey by W3Techs[1] conducted in 2022, Nginx [Reese 2008], Apache [Hu et al. 1999], Cloudflare [Dewi Estri et al. 2019], LiteSpeed [Setiawan et al. 2019], and Microsoft IIS [Chitra and Satapathy 2017] are the five most widely used web servers today to provide online services.

A server is an element that receives requests from clients (users) and responds to them appropriately. Therefore, designing a server is a challenging task as it must be able to handle multiple connections and provide correct responses as quickly as possible. Thus, it is realized that a high access rate by clients directly impacts the server's performance and, consequently, the applications it supports. Such a situation

[1]https://w3techs.com/technologies/overview/web_server

can trigger an overload and generate a scenario where the server cannot respond to clients as expected [Data et al. 2019].

This architectural model is called a Client-Server. The server underlies web applications and is a crucial component. Any improper functioning of this component when a client requests it can result in an error. This issue is known in the literature as the Single Point of Failure (SPF) [Noveck 2011].

One of the most commonly used strategies for addressing this issue involves increasing the number of servers and utilizing load balancing and failover techniques. By creating a cluster, for instance, the workload generated by a large number of user accesses can be distributed across multiple servers, thereby increasing the system's availability. In addition, if one server fails, the application will not be interrupted [Domanal and Reddy 2014]. Load balancing techniques are responsible for distributing the workload among the nodes in the cluster, while failover techniques detect which server is not functioning and redirect traffic to active servers [Kostadinov et al. 2017].

In this context, this study presents a performance evaluation based on a case study that forms a part of the development of a high-availability solution for a web application of the Brazilian Ministry of Regional Development. The primary differences from other works, in addition to the application-specific requirements, were the standardization of tests using ISO/IEC 25010:2011, the creation and assessment of a failure recovery strategy in a multi-platform cluster, and traffic analysis.

The paper is organized as follows: Section 2 presents an overview of related works on the proposed topic. Section 3 provides a theoretical background about the solutions for load balancing in web scenarios and a comprehensive explanation of ISO/IEC 2510:2011. Section 4 presents the proposed solution design and justifies the decisions. Section 5 outlines the methodology adopted for the evaluation. Section 6 presented the results. Section 7 provides final remarks and suggests future directions for research.

2. Related Works

Several investigations have been conducted about load balancers for web applications. For example, Chitra et al. [Chitra and Satapathy 2017] conducted a study to compare Node.js and IIS servers to determine the impact of their architectures on applications. The study involved performing tests using performance metrics such as throughput and response time, which were measured using the Apache Jmeter[2] tool. The results indicated that Node.js outperformed IIS in most cases, except for tasks that involved intensive CPU usage, where IIS had an advantage.

In another study [Zeebaree et al. 2020], the performance of web servers with and without load balancing was analyzed, as well as the impact and availability of the servers under DDoS attacks. To conduct a more precise evaluation, the researchers built and configured a 1 Gbps network, using a D-Link 24-port switch and CAT 6 UTP cables. In the Windows environment, two nodes were grouped as a cluster, and the NLB feature provided by Windows Server was activated. In the Linux environment, HAProxy was installed on a separate server offering load balancing. The performance was evaluated using Apache Jmeter, which generated loads of HTTP requests, while HPing3 was used

[2]https://jmeter.apache.org/

to create and measure DDoS attacks. The results demonstrated that using load balancing helped to overcome the massive traffic of HTTP requests.

The study presented in [Zebari et al. 2018] analyzed the impact of DDoS attacks (SYN flood and HTTP flood) on IIS and Apache2 web servers. A real network was used consisting of 17 workstations divided into four types: four workstations used to generate the attacks, two hosts configured as web servers, ten computers generating legitimate HTTP requests via Apache Jmeter, and the last one acted as a test controller, and result collector. Hping3 was used to create and measure DDoS attacks. The results show that IIS 10.0 is more efficient and responsive than Apache2. However, the Apache2 is more stable when exposed to flooding attacks.

The paper of [Johansson 2022] examined the performance of open-source load-balancing software such as HAProxy, NGINX, Traefik, and Envoy. The researchers used a virtual environment based on VMware ESXi version (7.0U3) as the hypervisor on an HP EliteDesk 800 G2 with an Intel i5-6600 4-core processor and 32 GB RAM. They deployed five virtual machines running Debian 11, three of which served as backend web servers, one as a load-balancing server, and one as a client. Each server VM had 4 CPUs, 4 GB of RAM and the client had 4 CPUs, 8 GB of RAM. The key performance metrics were throughput and response time, and Apache JMeter was used to forge traffic and measure the results. The experiment demonstrated consistent performance differences between the software in both scenarios.

The research conducted by [Hosseini et al. 2021] proposed a persistent session load balancer that does not require a reverse proxy, resulting in a lighter and more efficient system. The experiment was conducted using seven Linux virtual machines, one of which acted as a load balancer, another hosted a database, and the remaining servers acted as web servers. To compare the proposed approach with other methods, Cheetah was implemented as a layer four load balancer, while Nginx and HAProxy were used as layer seven load balancers. The preliminary evaluation of the study showed that the performance of the proposed method is significantly better than previous approaches in terms of transaction rate and response time. Table 1 summarizes the related works and their main features.

Our article is the first to address the Application Request Routing (ARR) of an IIS 10 cluster running the Windows Server 2022 operating system. In addition, we evaluated the metrics throughput (transactions/seconds), errors (%), response time (ms), and CPU utilization (%) with the goal of aligning with ISO/IEC 25010:2011.

3. Performance Evaluation for Web Servers

This section is intended to provide a theoretical background on the topics of performance evaluation for servers and high-availability solutions. These topics are essential to the context in which this work is conducted.

3.1. Web Servers

A web server is the software that processes HTTP protocol requests and provides services to clients [Liu et al. 2018]. With the significant growth of web services, the flow of information between websites and users has increased exponentially. Therefore, there is a need for more efficient technologies that can provide fast responses to requests and

Table 1. Comparison of Related Works

Paper	Load Balancer	Web Servers	Tools	Metrics
[1]	-	IIS Node.js	Apache-JMeter 2.13	Throughput (bytes/minute) Response Time (ms)
[2]	HAProxy NLB (IIS)	IIS Apache2	Apache-Jmeter Hping3	Throughput (kB/s) Response Time (ms) Error (%) CPU Usage (%)
[3]	-	IIS Apache2	Apache-Jmeter Hping3	Average Response Time (ms) CPU Usage (%) Standard Deviation as Responsiveness Efficiency and Stability
[4]	HAProxy NGINX Traefik Envoy	Apache2	Apache-Jmeter	Throughput (Transactions/seconds) Response Time (ms)
[5]	Cheetah HAProxy NGINX	-	Own Application	Throughput (Transactions/seconds) Response Time (ms)
Proposed	ARR HAProxy	IIS	Apache-JMeter Netdata Wireshark	Throughput (Transactions/seconds) Error Rate (%) Response Time (ms) CPU Usage (%)

[1] [Chitra and Satapathy 2017], [2] [Zeebaree et al. 2020], [3] [Zebari et al. 2018], [4] [Johansson 2022], [5] [Hosseini et al. 2021]

ensure high availability of applications. The web application used in this work is hosted on an Internet Information Service (IIS) server that operates in this context.

IIS, short for Internet Information Service, is a versatile and secure web server that can host various applications and technologies, including media streaming and service portals. It has an open and scalable architecture that provides flexibility and high availability. The latest version of IIS is 10.0, and since version 7.0, it works with a fully modular architecture. This architecture has three main features: Extensibility, componentization, and ASP.NET integration [Chitra and Satapathy 2017].

3.2. Load Balancers

In most Web services infrastructures, the load balancer is the first component to interact with requests from users or other systems. Its main function is to distribute requests among available servers so that they can process them and generate the appropriate responses. In such scenarios, a load balancer has two main goals: The first is to maximize the overall utilization of resources, while the second is to minimize the time required to process each request and provide its response [Rawls and Salehi 2022].

There are several approaches to providing load-balancing solutions that can be implemented through hardware or software. Hardware solutions are typically expensive and difficult to scale. Software solutions, on the other hand, involve installing additional programs or nodes and using a load-balancing algorithm. In general, this approach is more flexible and cost-effective than hardware-based solutions [Ibrahim et al. 2021].

Software-based load balancing mechanisms are divided into static and dynamic. In static load balancing mechanisms, the distribution of requests is based on a

predetermined configuration, disregarding the amount of load and the current number of connections of each node in the server pool. So, the predetermined information of the system is necessary for this strategy and is not updated according to the state of each node, such as the predetermined weight of each node [Mbarek and Mosorov 2018]. Round Robin is one of the most popular algorithms for this type of implementation.

Dynamic load balancing mechanisms, on the other hand, use performance information collected at runtime, such as accepted connections and load volume, to distribute requests among nodes that have more resources available. This approach requires more server resource consumption, such as CPU and RAM, and is recommended for heterogeneous server environments with machines of different capacities.

In the context of this study, particular attention has been paid to static balancing mechanisms, since this is a homogeneous server pool. Therefore, this section provides a brief overview of two such mechanisms, namely Application Request Routing (ARR) and High Availability Proxy (HAProxy).

3.2.1. Application Request Routing (ARR)

ARR[3] is a reverse proxy-based routing module from Microsoft that provides load balancing and reverse proxy services for web applications. Besides distributing incoming traffic between servers, it also provides additional features such as content caching, HTTP traffic management and rule-based routing. Its flexibility through various configuration strategies allows integration with many other IIS resources, such as SSL for data encryption and Web Application Firewall (WAF) for protection against attacks.

3.2.2. High Availability Proxy (HAProxy)

HAProxy (*High Availability Proxy*) is a widely used free and open source load balancer that provides availability and performance enhancement for web applications. In general, HAProxy acts as an intermediary between clients and web servers, distributing traffic to backend or frontend web servers. The goal is to ensure that each request is handled by the server with the most available resources. Its main features include low latency, advanced error management (failover), and a monitoring interface [Tarreau et al. 2012].

3.3. ISO/IEC 25010:2011

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) work in partnership to develop and define standards for various areas. ISO /IEC 25010:2011 Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models is a standard for describing product quality and usage models for software.

The quality model for software products includes both the static attributes of the program and the dynamic attributes of the computing environment in which the software is executed. The quality of use model consists of five key attributes, while the product quality model includes eight: Functionality, Reliability, Ease of Use, Efficiency,

[3]https://www.microsoft.com/en-us/download/details.aspx?id=47333

Maintainability, Portability, Security, and Compatibility. The current study focuses on parameters associated with efficiency.

According to the standard, software efficiency is defined as performance in terms of the number of resources used under specified conditions. It is divided into three subcharacteristics:

- **Temporal Behavior**: Degree to which the response times, processing times, and throughput rates of a product or system in performing its functions meet requirements
- **Resource Utilization**: Degree to which the quantities and types of resources used by a product or system in performing its functions meet requirements;
- **Capacity**: Degree to which the maximum limits of a parameter of a product or system meet the requirements;

Against this background, the next section evaluates the case study in terms of its efficiency. Some information was withheld due to the confidentiality agreement with the Ministry of Regional Development in Brazil.

4. Case Study

The objective of the case study presented in this paper was to evaluate the efficiency of a computer system in an operating environment configured to provide high availability, even in failure scenarios, for a strategic application of the Brazilian Ministry of Regional Development. The results were obtained in a test environment to validate the methodology and perform a proof of concept of the solution based on the application requirements. The Non-functional requirements (NFR) are detailed in Table 2. The functional requirements and some NFRs have been omitted due to a confidentiality agreement.

Table 2. List of non-functional requirements.

ID	Description
N1	The cluster of web servers **have** function and provide services to users even if one server in the cluster fails.
N2	The cluster of web servers **have** store user data reliably and consistently, even if one server in the cluster fails.
N3	The web server cluster includes a failover service to solve the single point of failure (SPF) problem.
N4	The web server cluster has service load balancers to distribute traffic among the individual servers in the cluster.
N5	The web server cluster provides the IIS web server service to HTTP requests from users.
N6	The cluster of web servers can provide a caching service to improve the performance of the web application.

A topology has been proposed based on the requirements for building an IIS web cluster that uses load balancers to distribute the load among the nodes. Figure 1 illustrates this topology along with its main components, which include IIS web servers, HAProxy, and ARR load balancers. The JMeter was used to generate virtual clients.

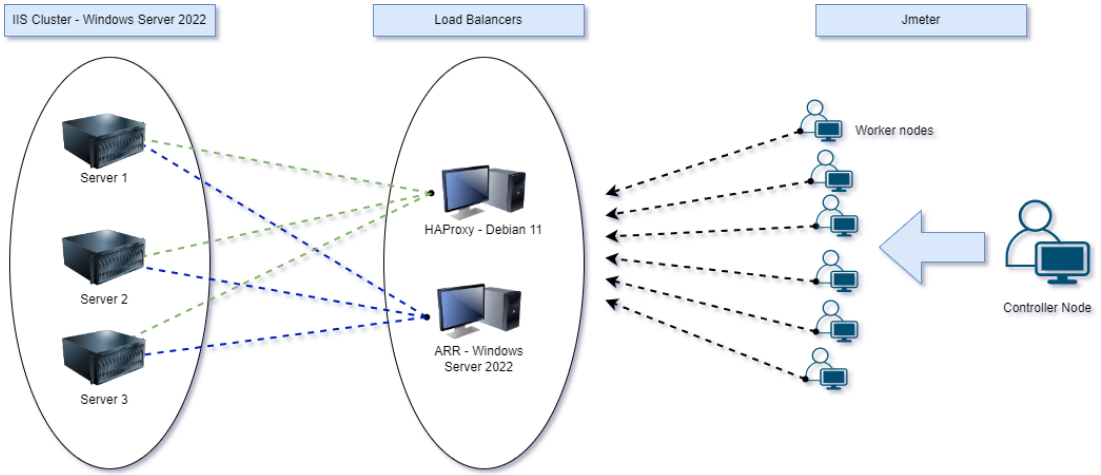

Figure 1. Representation of the experimental scenario of the case study.

Implementation of the scenarios required 12 physical computers arranged in a research lab, with traffic restricted to the local network via a HP -A5120 switch connected via Category 6 UTP network cables. The web server cluster consisted of three machines running the IIS 10.0 service, using two load balancers, one HAProxy 2.8.4 LTS, the other ARR 3.0. Client traffic was generated by seven machines configured with Apache JMeter 5.5, one acting as a controller and the others as remote nodes. For a detailed specification of the hosts, see Table 3.

Table 3. Configuration of the machines used in the case study.

Type	Operational System	Application	CPU	RAM
Web Server	Windows Server 2022	IIS 10.0	Intel Core I5 @ 3.20GHz	16 GB
Load Balancers	Debian 11/Windows Server 2022	HAProxy 2.8.4 LTS/ARR 3.0	Intel Core I5 @ 3.20GHz	16 GB
Clients	Ubuntu 22.04	JMeter 5.5	AMD Phenom II @ 3.20GHz	8 GB

The load balancers were configured using the round-robin algorithm with equal weight distribution between each node, since all machines have the same amount of available resources. To solve the problem of node failures in the cluster, node state checking was enabled with an interval of 1.0 seconds.

5. Methodology

The methodology of this paper was guided by the recommendations in ISO/IEC 25010:2011. Thus, the evaluation of load balancers considered metrics related to system efficiency, in particular response time, throughput (requests per second), error rate, and usage of CPU.

Response time refers to the time it takes an application to respond to a service request, such as loading a requested web page. Throughput indicates the number of transactions or requests that can be processed within a given time interval. It is usually used to check server utilization. Error rate is the percentage of legitimate requests that were not processed correctly. Finally, CPU usage refers to processor consumption when processing a given workload [Jader et al. 2019].

In general, the infrastructure described in the previous section was load tested. To generate HTTP requests, a JMeter cluster with seven nodes, one controller node, and six

nodes configured as remote nodes (slaves), was used. The JMeter script consisted of four HTTP GET requests interrupted by constant timers of 300 ms. The virtual users were 25,000, 50,000, 75,000, 125,000, and 250,000, so the tests generated 600,000, 1,200,000, 1,800,000, 3,000,000, and 6,000,000 requests, respectively.

In addition to JMeter reports, Netdata[4] and Wireshark[5] software were used. Netdata was used as a collector of resource usage parameters and requires the agent to be installed on each node. It is important to note that at the time of writing, there is no version for the Windows platform. Therefore, we used an Ubuntu image over WSL (Windows Subsystem for Linux). To further investigate the scenarios, we recorded all traffic on the load balancers and cluster nodes using Wireshark.

As for the load balancers, ARR 3.0 and HAproxy 2.8.4 were selected. Since the Microsoft platform is a requirement for this work, the focus was on the IIS 10.0 web server configured with and without the use of the caching feature as a cluster. Table 4 details the scenarios evaluated.

Table 4. Organization of evaluated scenarios.

Scenario	Load Balancer	Users
1	HAProxy	25000, 50000, 75000, 125000 e 250000
2	ARR	
3	HAProxy (Caching)	
4	ARR (Caching)	

6. Results and Discussion

The results are summarized in Tables 5, 6, and 7. Table 5 gives the throughput in requests per second. ARR (with and without caching) showed higher throughput than HAProxy in almost all scenarios. In the configuration without caching, ARR showed better performance. In caching mode, the results were very similar, but ARR was better. In the most critical scenario with 250,000 users per node, the throughput was very similar, showing that the use of load balancing techniques is important to achieve robustness. Figure 2 illustrates this behavior as a function of the number of virtual users.

Table 5. Throughput results in transactions per second for the scenarios evaluated.

Users (10^3)	HAProxy	ARR	HAProxy (Caching)	ARR (Caching)
25	2,093.70	3,804.35	3,476.00	1,982.77
50	4,069.56	4,719.65	5,297.10	5,616.40
75	4,514.94	5,687.04	4,541.30	6,755.55
125	7,148.15	8,323.88	7,731.80	7,397.89
250	9,606.64	9,795.52	9,360.95	9,535.34

Table 6 shows the values obtained for the error rate. ARR in ”disable caching” mode was the only load balancer that showed errors in the scenario with 25,000 users.

[4]https://www.netdata.cloud/

[5]https://www.wireshark.org/

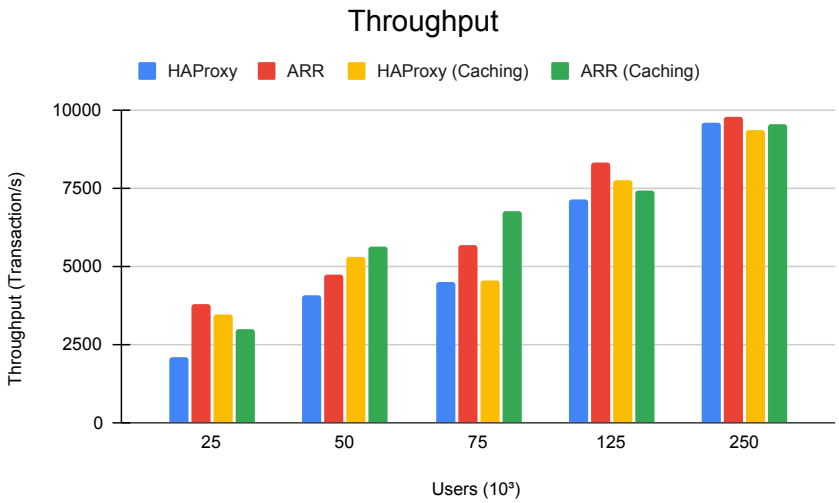

Figure 2. Throughput by the load balancer and the number of virtual users.

The load balancer failed to serve about 60,000 out of 600,000 HTTP requests, a high value compared to HAProxy and ARR (caching), which showed high values only in the last scene. ARR (caching) had a lower failure rate than HAProxy. This shows that the server makes good use of caching to respond correctly to users. HAProxy (caching) had a failure rate of 5.13%, and ARR (caching) had 5.35% in the most critical scenario. Figure 3 shows the results graphically. ARR (without cache) had the worst performance in this regard, and ARR (caching) had the best performance.

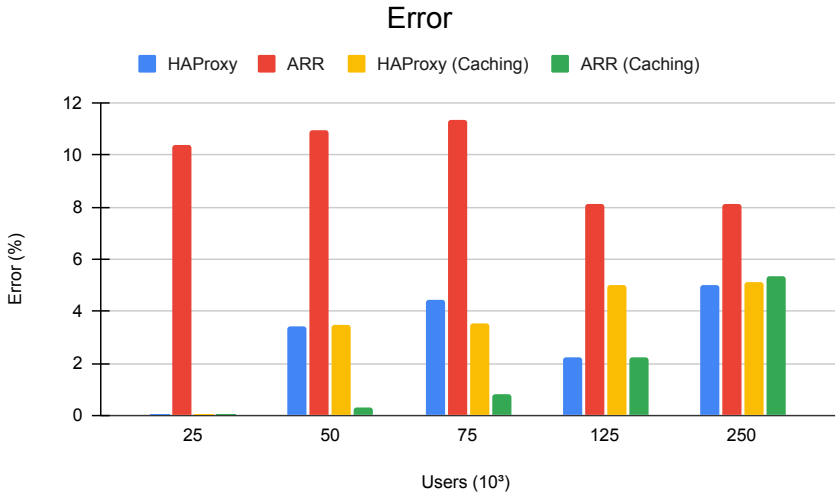

Figure 3. Error Rate by load balancer and number of virtual users.

Table 7 describes the results for the response time metric. The load balancers that performed best in this regard were ARR (caching) and HAProxy (caching), as the use of caching allows for faster delivery of responses. In this configuration mode, load balancing does not forward all requests to the web cluster, this strategy improves performance. HAProxy (caching) had lower performance in scenarios 1, 2, and 3, but the

Table 6. Results in percentage of error rate for the evaluated scenarios.

Users (10^3)	HAProxy	ARR	HAProxy (Caching)	ARR (Caching)
25	0,00	10,35	0.00	0,00
50	3,41	10,92	3.50	0,33
75	4,43	11,36	3.52	0,82
125	2,23	8,12	4.98	2,21
250	5,03	8,14	5.13	5,35

performance was almost constant in scenarios 4 and 5, which introduces scalability. A comparison between HAProxy and ARR with the feature disabled shows the superiority of HAProxy's performance, with values up to 56% lower than ARR. Figure 4 illustrates these results.

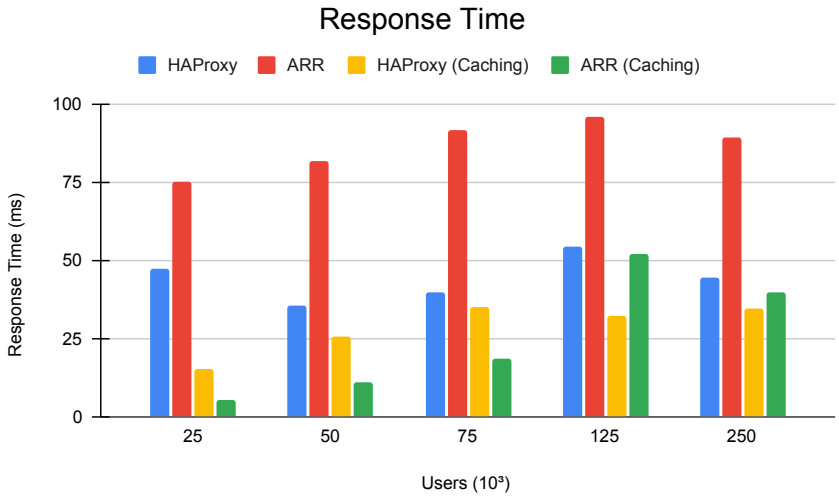

Figure 4. Response Time by load balancer and number of virtual users.

Table 7. Results from Response Times (ms) for the evaluated scenarios.

Users (10^3)	HAProxy	ARR	HAProxy (Caching)	ARR (Caching)
25	47.3	75.26	15.06	5.23
50	35.64	81.56	25.45	10.96
75	39.75	91.60	34.97	18.50
125	54.25	96.08	32.15	51.99
250	44.52	89.14	34.85	39.60

6.1. Crash Failure

To evaluate the load balancers in case of a crash failure, we interrupt a network interface of a cluster node. The interface is paused 10 seconds after the start of the test. The goal was to measure how HAProxy and ARR handle this situation. We use scenario 1 (25,000 virtual users per JMeter slave node) for this proposal. Table 8 shows the results.

Table 8. Results from crash failure experiment.

Metric		HAProxy	HAProxy (caching)	ARR(Caching)
Throughput (Transactions/s)		3,280.16	1,713.15	1,759.61
Error (%)		2.85	0.00	0.00
Response Time (ms)		1,149,87	7.21	7.54
CPU Utilization (%)	Minimum	3,35	4,32	5,29
	Maximum	16,65	33,32	37,59
	Average	9,52	12,77	17,75

HAProxy without cache activation reaches 3,280.16 transactions per second. Load balancers with caching enabled achieve lower throughput, but do not exhibit errors. HAProxy (caching) and ARR (caching) show better performance than HAProxy without caching. The results prove that the caching strategy is a good alternative to overcome single point of failure on cluster nodes.

Regarding the usage of CPU, solutions with caching had higher usage of CPU. For ARR, the average was 17.75%, while for HAProxy it was 12.77%. HAProxy without caching showed 9.52%. It is important to emphasize that solutions based on Windows Server consume more resources than most Linux distributions.

7. Conclusions

This paper presented the evaluation of a solution for high availability using load balancers for web applications. The load balancers used in the case study were HAProxy and ARR, while the web servers were implemented by IIS. The efficiency metrics proposed by the Quality-in-Use model, described in ISO 25010:2011, were used as a reference to perform the experiments.

Results have shown that the use of caching increases the ability to respond to requests and reduces the error rate. In scenarios without this technique, where all requests were forwarded to the servers, HAProxy performed better, especially when compared to ARR.

Thus, functional requirements F1 and F2 are about the evaluated solution providing services to users even if a server in the cluster fails, and storing user data reliably and consistently. Non-functional requirements N1, N2, N3, and N4 were met. The solution uses a failover service to solve the SPF problem, distribute the load among the servers, provide HTTP responses, and a caching service.

For a more advanced assessment, future work targets the use of data center infrastructure and dynamic balancing strategies, especially considering pools of heterogeneous servers. Another future study is related to the inclusion of redundancy for load balancing servers and the creation of new scenarios with database operations.

References

Chitra, L. P. and Satapathy, R. (2017). Performance comparison and evaluation of node. js and traditional web server (iis). In *2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)*, pages 1–4. IEEE.

Data, M., Kartikasari, D. P., and Bhawiyuga, A. (2019). The design of high availability dynamic web server cluster. In *2019 International Conference on Sustainable Information Engineering and Technology (SIET)*, pages 181–186. IEEE.

Dewi Estri, J., Umar, R., and Riadi, I. (2019). Implementation of cloudflare hosting for speeds and protection on the website. *Universitas Ahmad Dahlan*.

Domanal, S. G. and Reddy, G. R. M. (2014). Optimal load balancing in cloud computing by efficient utilization of virtual machines. In *2014 sixth international conference on communication systems and networks (COMSNETS)*, pages 1–4. IEEE.

Hosseini, S. M., Jahangir, A. H., and Daraby, S. (2021). Session-persistent load balancing for clustered web servers without acting as a reverse-proxy. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 360–364. IEEE.

Hu, Y., Nanda, A., and Yang, Q. (1999). Measurement, analysis and performance improvement of the apache web server. In *1999 IEEE International Performance, Computing and Communications Conference (Cat. No. 99CH36305)*, pages 261–267. IEEE.

Ibrahim, I. M., Ameen, S. Y., Yasin, H. M., Omar, N., Kak, S. F., Rashid, Z. N., Salih, A. A., Salim, N. O., and Ahmed, D. M. (2021). Web server performance improvement using dynamic load balancing techniques: A review. *system*, 19:21.

Jader, O. H., Zeebaree, S., and Zebari, R. R. (2019). A state of art survey for web server performance measurement and load balancing mechanisms. *International Journal of Scientific & Technology Research*, 8(12):535–543.

Johansson, A. (2022). Http load balancing performance evaluation of haproxy, nginx, traefik and envoy with the round-robin algorithm.

Kostadinov, B., Jovanov, M., and Stankov, E. (2017). Cost-effective website failover through a cdn network and asynchronous replication. In *IEEE EUROCON 2017-17th International Conference on Smart Technologies*, pages 151–156. IEEE.

Liu, G., Xu, J., Wang, C., and Zhang, J. (2018). A performance comparison of http servers in a 10g/40g network. In *Proceedings of the 3rd International Conference on Big Data and Computing*, pages 115–118.

Mbarek, F. and Mosorov, V. (2018). Load balancing algorithms in heterogeneous web cluster. In *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, pages 205–208. IEEE.

Noveck, B. S. (2011). The single point of failure. *Innovating Government: Normative, policy and technological dimensions of modern government*, pages 77–99.

Rawls, C. and Salehi, M. A. (2022). Load balancer tuning: Comparative analysis of haproxy load balancing methods. *arXiv preprint arXiv:2212.14198*.

Reese, W. (2008). Nginx: the high-performance web server and reverse proxy. *Linux Journal*, 2008(173):2.

Sadqi, Y. and Maleh, Y. (2022). A systematic review and taxonomy of web applications threats. *Information Security Journal: A Global Perspective*, 31(1):1–27.

Setiawan, E., Setiyadi, A., and Wahdiniwaty, R. (2019). Quality analysis of mobile web server. In *IOP Conference Series: Materials Science and Engineering*, volume 662, page 022043. IOP Publishing.

Tarreau, W. et al. (2012). Haproxy-the reliable, high-performance tcp/http load balancer.

Zebari, R. R., Zeebaree, S. R., and Jacksi, K. (2018). Impact analysis of http and syn flood ddos attacks on apache 2 and iis 10.0 web servers. In *2018 International Conference on Advanced Science and Engineering (ICOASE)*, pages 156–161. IEEE.

Zeebaree, S., Zebari, R. R., and Jacksi, K. (2020). Performance analysis of iis10. 0 and apache2 cluster-based web servers under syn ddos attack. *TEST Engineering & Management*, 83(March-April 2020):5854–5863.