International Journal of Innovation and Applied Studies
ISSN 2028-9324 Vol. 41 No. 3 Jan. 2024, pp. 859-866

http://www.ijias.issr-journals.org/

Development of a web server load balancing system

Thomas LOBA[1], N'Golo KONATE[1-2], Mory Richard BATIEBO[2-3], and Romaric De Judicael BITA[2-4]

[1]Mathématiques- Informatiques, Université Félix Houphouet Boigny, Abidjan, Côte d'Ivoire

[2]Informatiques et Sciences du Numérique, Université Virtuelle de Côte d'Ivoire (UVCI), Abidjan, Côte d'Ivoire

ABSTRACT: The exponential growth of Internet traffic generated by a plethora of interconnected apps poses a size challenge, making effective management of incoming requests by a single server difficult, even for the most reputable businesses. To ensure uninterrupted service delivery, IT teams are turning to the deployment of many servers operating inside a distributed system framework.
Charge balancing appears to be the best strategy for capitalizing on increasing data traffic, with the dual goal of distributing computation costs over several servers and improving overall infrastructure performance. In order to achieve this goal, a range of solutions, including specialized hardware, dedicated software, or a combination of the two, may be envisaged.
The combined use of keepalived with HAProxy has shown a notable reduction in recovery time following a server panel, minimizing stop time to only one second. Furthermore, our investigation reveals that in low-traffic scenarios, the Round Robin algorithm performs better than HAProxy and keepalived, but in high-traffic scenarios, the source IP technique leads. This idea emphasizes how wise it is to evaluate three algorithms and select the best one based on the traffic's fluctuating bit rate.

KEYWORDS: High Availability, Load balancing, HA Proxy, keepalived.

1 INTRODUCTION

Daily business ruling faces the use of a bunch of remote software. Those softwares are mostly hosted in remote software. Moreover, the Covid-19 crisis has increased the number of remote work. From 2019 to early 2023, the share of postings that say new employees can work remotely one or more days per week rose more than three-fold in the U.S and by a factor of five or more in Australia, Canada, New Zealand and the U.K (Hansen, 2023). Since the computers making up the infrastructure have different characteristics, it's better to take advantage of each server's strength. The quest for optimum responsiveness to these requests calls for establishing a configuration with multiple web assigned a volume of requests per its specifications, to ensure efficient use of their respective capacities.

Load balancing is dedicated to distributed and parallel systems to manage the web cluster (Singh, 2022) (Mbarek, 2018). The entrance point distributes the load between servers using various algorithms. However, the increasing of remote services has made load balancing and, in turn, High Availability (HA) crucial. Therefore, this study is to enhance HA through the joint use of HAproxy as a load balancer and keepalived for failover. To achieve this objective, the different scenarios will be compared. Indeed, a study conducted by Google shows that, after the three first seconds of a request, the probability that a user will close the application is 32%.

The remainder of this paper is organized as follow, section II will be dedicated to a review of different method to achieve HA. Then the testing environment will be described. Finally, the different results from each scenario will be analysed and compared.

2 LITTERATURE REVIEW

Load balancing, a fundamental task in the realm of web server clusters, is the process of equitably distributing infrastructure workloads among diverse servers. The primary objective is to ensure the efficient utilization of resources within the cluster system, ultimately enhancing its performance by expediting request processing (Patil, 2013).

Corresponding Author: Thomas LOBA

At the heart of load balancing lies the load balancer, a critical component capable of identifying the attributes of each server based on the underlying infrastructure architecture (Anselmi, 2020). When a request is received by the load balancer, it skillfully routes it to an appropriate server using a predefined algorithm. The selection criteria employed in this process determine the server best suited to handle the incoming request.

Subsequently, every subsequent request is likewise routed based on these established criteria.

Within the context of a web server cluster, the load balancer serves as the central processing hub. It is endowed with a Virtual IP (VIP), which serves a dual purpose—enabling the forwarding of requests and discerning incoming requests through a private address. The placement of this address within the protocol hierarchy varies depending on the architecture. This arrangement results in the system being perceived as a unified entity by end-users, even though it comprises multiple servers working in unison.

The evaluation of load balancing algorithms predominantly hinges on the measurement of workload (Alssaheli, 2022), although multiple other factors come into play when assessing the algorithm's overall performance.

The workload evaluation is based on the following criteria (Song, 2022):

- The number of open connections.
- The current usage or the processor total capacity: It measure the processor workload in percentage.
- The current memory or the total memory: this indicates the ratio of memory used to total memory.
- The bandwidth: this measure the quantity of data transmit throughout the server.
- The time to respond.
- The number of process.

Based on these evaluation criteria, load balancing is classified as statics and dynamics (Talaat, 2022):

- Static load balancing algorithms used the compilation environment to estimate the time for each task. Moreover, the workload is shared equitably between the servers. However, it cannot manage the workload change during the process. This algorithm is attractive because of their ease of implementation and low running costs. But they are limit by their incapacity to predict the load balancing and it's necessary to know the resources stability.
- Dynamic load balancing: Theses algorithms decisions are based on the system current state. They use information's from the different servers as the processor capacity to share the requests dynamically. These algorithms are suitable for heterogeneous networks.

If the dynamic algorithm are more efficacies than the static one, the execution time in dynamical algorithms can increase due the complexity of the algorithms and the times to collect the information on the system (Mena, 2022).

There are several technic for load balancing. It can be achieved through a materiel-based approach with F5 BIG-IP, Citrix ADC, Barracuda Load Balancer ADC. The cited brand offers a bunch of hardware with built in features for load balancing. As this paper is not devoted to this approach, we won't go deep in the configuration.

Coming to algorithm based approach in load balancing, Random is the simplest one, however it has several flaws which.

3 PROPOSED ARCHITECTURE

In order to construct a software model that will allow the incoming client load to be efficiently dispersed throughout the web server system, we will need to include several architectural design-specific characteristics and other information. Figure 1 is design for a load balancing scenario and figure 2 is designed for a high availability scenario

In Figure 1, the HTTP request coming from a client will be dispatched on the cluster using HAProxy or Keepalived. The selection of an algorithm other the other will depend on the test result. However, there is some technical specifications on each algorithms configuration

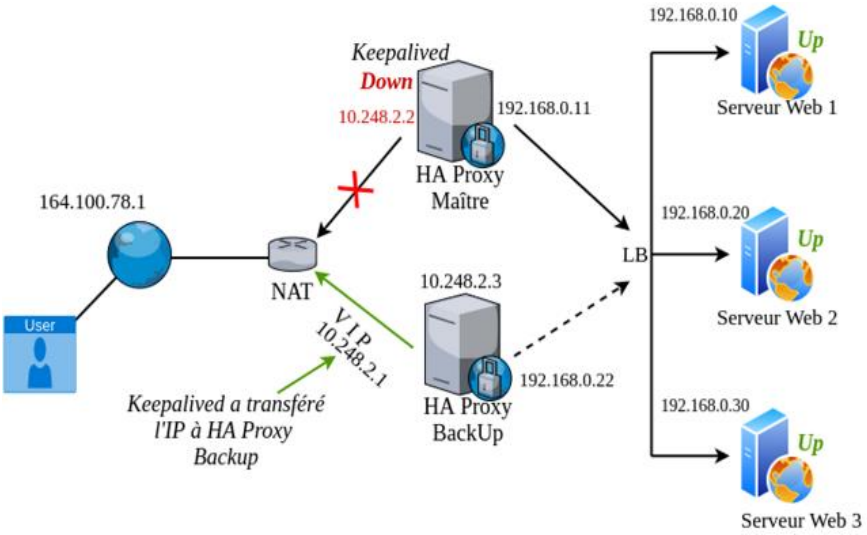

Fig. 1. Test Environnemnt Architecture

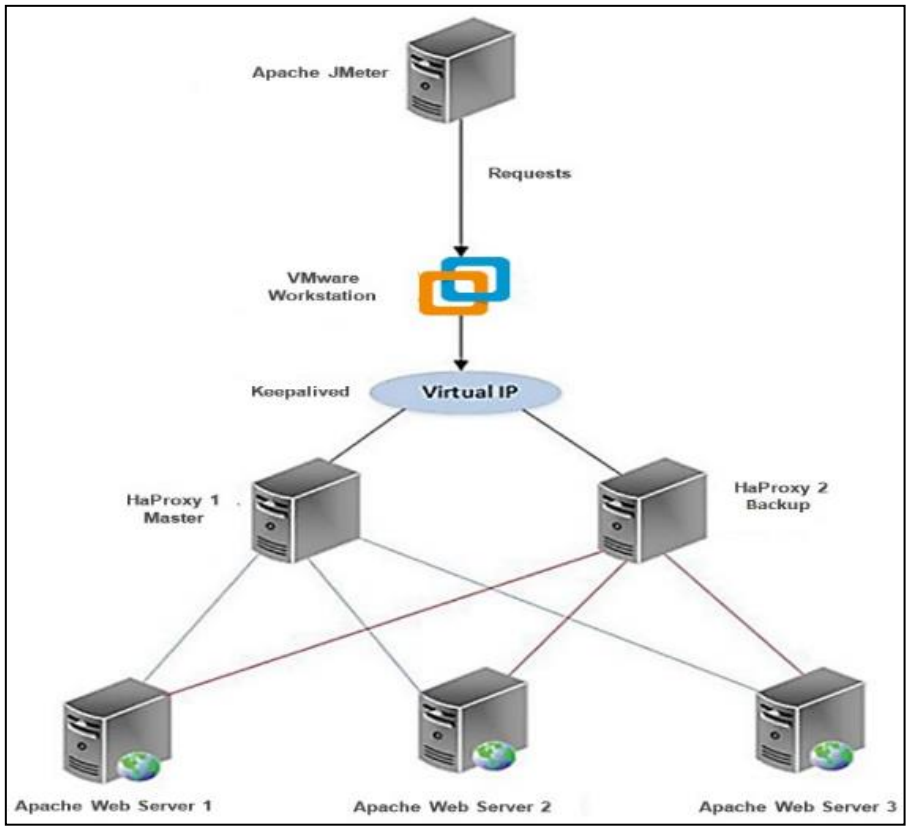

Fig. 2. High Availability Test

3.1 HA PROXY

HAProxy can be thought of as a two-half architecture, comprising a front-end and a back-end. The front-end is responsible for direct communication with the client, listening to incoming requests and applying the rules defined at this level. These rules may include blocking certain requests, modifying headers or collecting statistics.

In our configuration shown in figure 2, once the request has been received by the HAProxy front-end, it is passed on to the back-end, which is connected to the Web servers. This is where the load balancing strategy is applied, allowing the appropriate server to be chosen for the request. Once the server has been chosen, the request is sent to it.

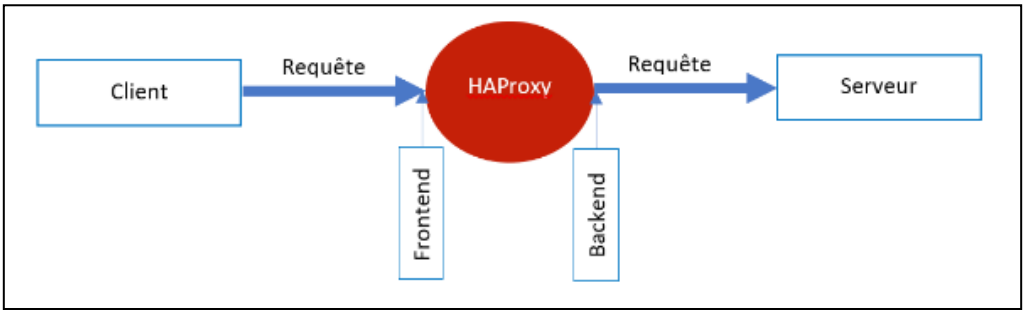

Fig. 3. HA Proxy operation

After the server has processed the request, the response is sent back to HAProxy. At this point, HAProxy can perform some additional processing on the response, or send it directly back to the client via the frontend.

3.1.1 SCHEDULING'S ALGORITHMS

HAProxy offers more than nine scheduling algorithms, among which the most commonly used are:

- Round robin (RR): This is the most frequently used algorithm, distributing requests equally among the different servers, one by one in a loop. By default, each server is automatically addressed in turn, but it is possible to weight the servers to optimize performance in the case of heterogeneous servers.
- Least connection: This algorithm selects the server with the lowest number of active connections. In this way, the request is forwarded to the server with the lightest load, enabling more dynamic load balancing.
- Source: This algorithm uses the client's source IP address to determine the destination server. By hashing the source IP address, you can send the client to the same server for each request. This can be useful for session persistence, although its scope is limited.
- URI: In this algorithm, the server is selected according to the URI of the HTTP request. A hash of the URI is calculated (either the left-hand part of the URI only, before the question mark, or the entire URI including all parameters). This determines the server to which the request should be sent. This method is commonly used in caching proxies to maximize hits.

These algorithms offer a variety of possibilities for request scheduling in HAProxy, allowing users to choose the one best suited to their specific needs

3.1.2 SESSION PERSISTENCE

HAProxy offers several methods for ensuring session persistence, also known as "Stickiness". These methods ensure that the client remains connected to the same server for each request, thus avoiding disconnections caused by landing on a different server that does not contain the client's session information.

Session persistence can be based on various parameters, such as the source IP address, URL, cookies, sessions, etc. These parameters are used to determine the correct session. These parameters are used to determine the correct server to which the client should be redirected, thus ensuring continuity in the client-server relationship.

To manage this persistence, HAProxy uses a table called the "stickiness table" at server level. This table contains the mapping information between the client and the associated server, enabling client-specific data to be stored. When a new request arrives, HAProxy uses this information to route the request to the appropriate server, thus guaranteeing the persistence of the client's session. This table contains mapping information between the client and the associated server, enabling client-specific data to be stored. So, when a new request arrives, HAProxy uses this information to route the request to the appropriate server, thus guaranteeing the persistence of the client's session.

3.1.3 HIGH AVAILABILITY

Simple load balancing algorithms increase the risk of server downtime, as they increase the likelihood that a client will be directed to a server that is unavailable. To ensure that servers are always available to customers, and to avoid any interruption in service, HAProxy offers mechanisms to ensure high availability.

HAProxy implements high-availability mechanisms by periodically testing the status of servers through so-called "health checks". These tests verify that servers are operating correctly before sending requests. In this way, only fully operational servers actually receive client requests.

What's more, HAProxy makes it possible to remove a failed server (a node) without affecting the overall operation of the architecture. When a failure is detected, HAProxy automatically activates backup servers, ensuring continuity of service despite a server outage.

By default, when a server goes down, HAProxy distributes the load evenly over the remaining servers. This maintains a balanced load distribution and prevents excessive overloading of the remaining servers.

3.2 KEEPALIVED

Keepalived is an open-source application used to ensure high availability and failover in network environments. It operates as a Virtual Router Redundancy Protocol (VRRP), enabling multiple routers to form a group and work together as a virtual entity to provide a shared, redundant IP address.

Keepalived is used to ensure a smooth transition between load balancers (Saeid & Yahiya, 2018). It avoids downtime and major interruptions by quickly detecting failures and automatically switching traffic to a working backup router.

The Keepalived operating process is based on the election of a master router within the VRRP group. The Master router manages the shared virtual IP address and handles network traffic. The other routers in the group act as backup routers, ready to take over if the master router fails.

Keepalived uses VRRP Advertisements to periodically signal that it is still active as the master router. Other routers listen to these advertisements to determine whether the master router is still available. If the master router fails (for example, if it stops sending advertisements), the other routers in the group quickly detect the failure and trigger the election process to choose a new master router from among the backup routers.

In addition to providing high availability and failover, Keepalived can be used with LVS (Linux Virtual Server) to provide load balancing between several active servers, improving performance and optimizing resource utilization.

Keepalived is widely used in enterprise network environments, data centers and cloud infrastructures, where service availability is critical. Thanks to its flexible configuration, easy integration with Linux systems and open-source status, Keepalived has become an essential tool for guaranteeing the redundancy, reliability and resilience of critical infrastructures.

3.3 APACHE WEB SERVER

The long-established web server that has dominated the market for over a decade in terms of the number of web pages hosted. Its dominant position can be explained by the habits of administrators, but also by the plethora of optional modules available, including the Modsecurity application firewall (Guionnet, 2013).

3.3.1 EVALUATION'S TOOLS

(Sharma et al., 2016) focused on the comparison of four load testing tools, WebLOAD, Apache Jmeter, HP Load Runner and Grinder. The main objective of their paper is to study these load testing tools and select the best among them. They used certain parameters to evaluate these tools, such as unlimited load generation, server monitoring, ease of use and cost. At the end of their comparison, the authors selected JMeter as the best tool because it is free, has good load generation and user interface. For the above reasons, we have decided to use JMeter as a performance testing tool in the remainder of our work.

3.4 TEST SCENARIOS

To avoid congestion, we used a load balancing tool such as HAproxy to intercept user requests and distribute them evenly across the servers.

Three load-balancing algorithms (Round Robin, Least Connection and Source) will each be tested on the load balancer, then the results obtained with each algorithm will be analyzed in order to choose the most suitable according to the metrics selected. In general in this context, there is practically only one method for testing the performance of a load balancing algorithm (Saeid & Yahiya, 2018). The method involves generating loads in terms of requests to web servers. Query generation will take place via the Jmeter load testing tool. Depending on the algorithm design, requests will be forwarded to the appropriate servers (Madani & Jamali, 2018).

To avoid the load balancer being a single point of failure, we have duplicated it to ensure high availability. We will study the effect of failure of the master load balancer on end-user QoS.

Each time, display the results obtained in the form of graphs, curves, etc. Analyze and interpret the results to select the most appropriate algorithm: vary the number of simultaneous requests and use the graphs and statistics generated.

4 RESULT AND DISCUSSION

4.1 RESULT

The tests are conducted in three cases, 1000 requests, 1500 requests and 2000 request

For 1000 request in figure 4, the source algorithm performs less well than the other two. Load balancing with the Round-Robin algorithm performs best when handling traffic of 1,000 requests per minute. Compared to load balancing with the Round-Robin algorithm, the least connection algorithm has a shorter reaction time.

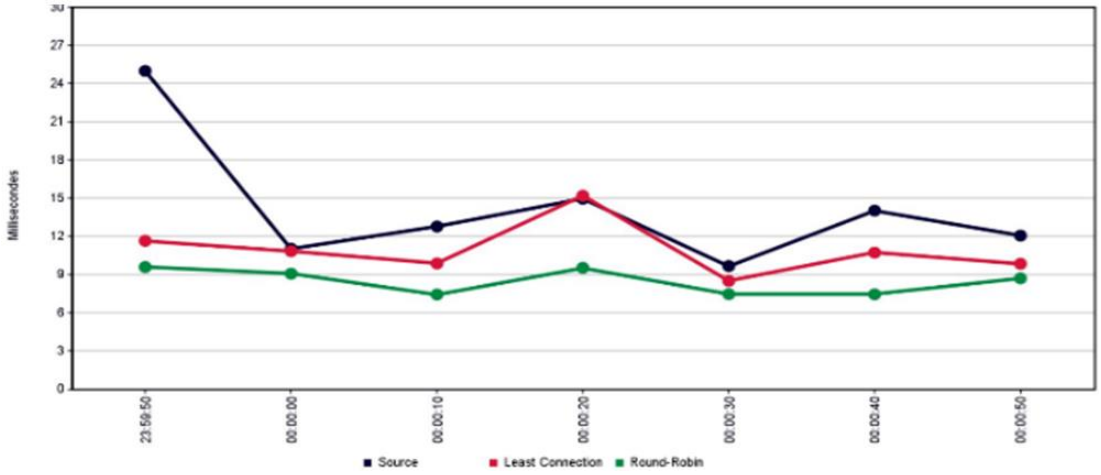

Fig. 4. Performance evolution for 1000 request

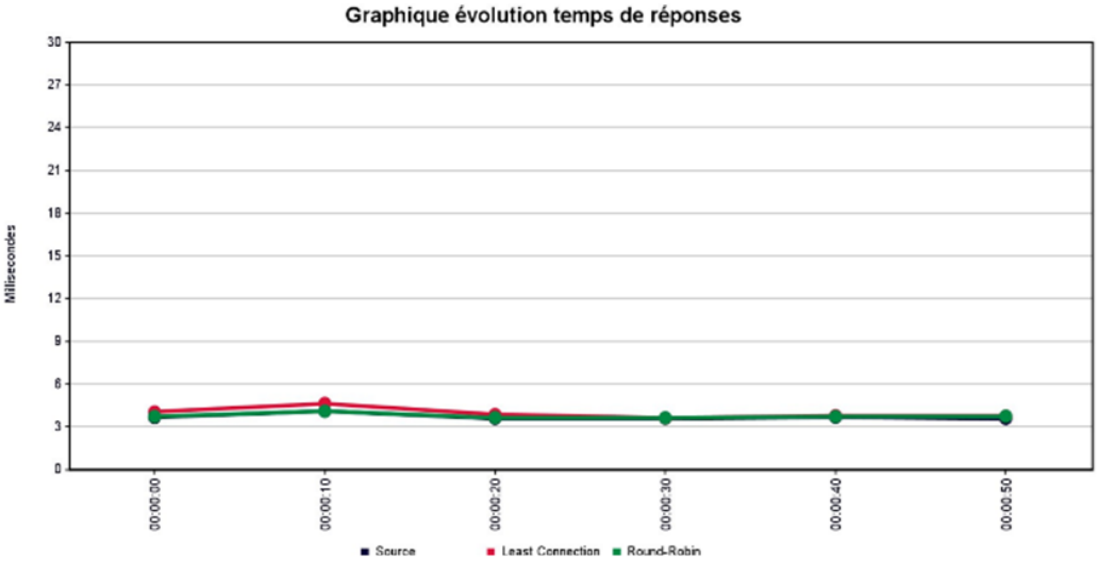

Fig. 5. Performance evolution for 1500 request

For 1500 request in figure 5, the algorithms performance are similar as time progress.

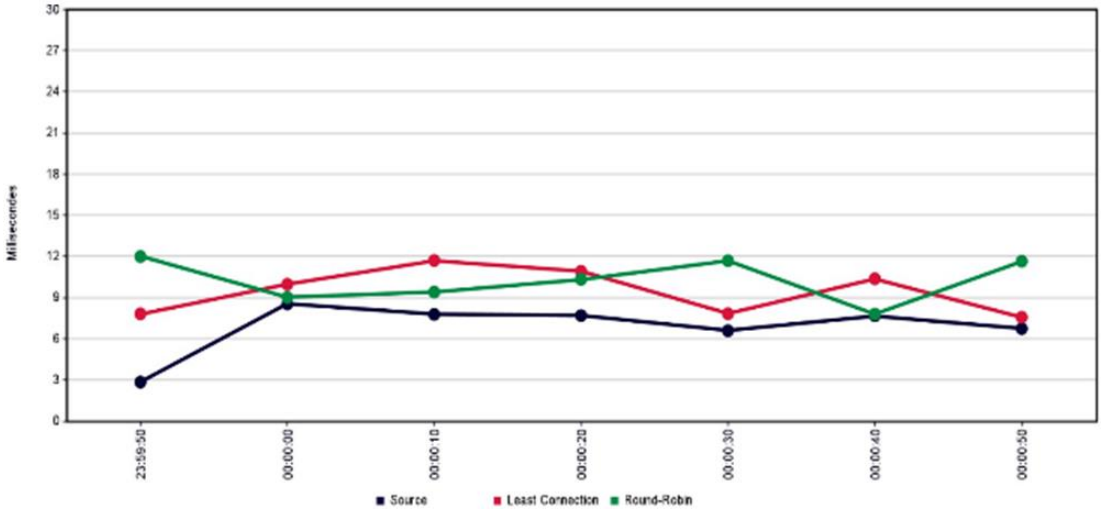

Fig. 6. Performance evolution for 2000 request

In figure 7, The IP Source algorithm outperformed the other two algorithms over the entire duration of the performance test.

4.2 DISCUSSION

In Figure 2 scenario, we set up two HAProxy load balancers to guarantee high availability using the Keepalived tool. Keepalived assigned a virtual IP address to both load balancers and designated the first load balancer as master and the second as backup,

We use JMeter to send 10,000 requests to the master load balancer. During this operation, we'll simulate a failure by shutting down the server. The aim is to quickly detect the failover and measure the time elapsed between the main server becoming unavailable and the backup server taking over.

The server failover is illustrated in figure 8 and failed request in figure 9.

```
sept. 23 15:50:23 haproxy2 Keepalived_vrrp[1387]: VRRP_Script(reload_haproxy) succeeded
sept. 23 15:50:26 haproxy2 Keepalived_vrrp[1387]: VRRP_Instance(VI_1) Transition to MASTER STATE
sept. 23 15:50:27 haproxy2 Keepalived_vrrp[1387]: VRRP_Instance(VI_1) Entering MASTER STATE
```

Fig. 7. HA Proxy Failover

Fig. 8. Unsuccessful http requests

Figure 7 and figure 8 show that during the test over 10,000 requests, 60 requests failed, i.e. 0.6% of all requests, and it only took 1s for the second server to become the master.

Therefore, IP Source performs better for high traffic, while the Round Robin algorithm offers better response times for low to medium traffic. This underlines the importance of choosing the right load balancing algorithm according to workload characteristics.

However, the efficiency of the load balancing algorithm can vary considerably depending on the usage scenario. The choice of algorithm must therefore be carefully considered in relation to the specific needs of the application.

The fast failover time (1 second) and low number of failed requests (0.6%) are positive indicators of the responsiveness of our infrastructure. This shows that, despite the failure of the master HAProxy server, the user experience remains good. However, these results may vary depending on application complexity and workload. It is important to regularly monitor and test our infrastructure to ensure that it always meets high availability requirements

5 CONCLUSION

In this work, we used the HAProxy load balancer. More specifically, we unveiled the load-balancing algorithms it uses and examined three of them in real-life contexts with various workload intensities and under homogeneous back-end servers.

In general load balancing scenarios in a homogeneous server environment, the source and round robin algorithms give the lowest response times.

As part of our failover system design, we use two load balancing servers to guarantee high availability of our infrastructure. Our approach is to minimize downtimes, keeping them to a maximum of 1s, whether in the event of a loss of communication or the failure of one of the load balancing servers

REFERENCES

[1] Ali, S. A. (2023). Navigating the Multi-Cluster Stretched Service Mesh: Benefits, Challenges, and Best Practices in Modern Distributed Systems Architecture. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY,* 98--125.O. V. ADEOLUWA, O. S. ABODERIN, and O. D. OMODARA, «An Appraisal of Educational Technology Usage in Secondary Schools in Ondo State (Nigeria),» *International Journal of Innovation and Applied Studies,* vol. 2, no. 3, pp. 265–271, 2013.

[2] Anselmi, J. (2020). Combining size-based load balancing with round-robin for scalable low latency. *Ieee Transactions on Parallel and Distributed Systems,* 886-896.

[3] Hansen, S. a. (2023). *Remote work across jobs, companies, and space.* National Bureau of Economic Research.

[4] Patil, S. &. (2013). Cluster performance evaluation using load balancing algorithm. *International Conference on Information Communication and Embedded Systems (ICICES),* 104-108.

[5] Samir, R. a.-H. (2023). Cluster-based multi-user multi-server caching mechanism in beyond 5G/6G MEC. *Sensors: MDPI,* 996.

[6] Singh, P. a. (2022). A fog-cluster based load-balancing technique. *Sustainability: MDPI,* 7961.