Knowledge Engineering and Data Science (KEDS)
Vol 6, No 1, April 2023, pp. 79–91

pISSN 2597-4602
eISSN 2597-4637

Round-Robin Algorithm in Load Balancing for National Data Centers

I Kadek Wahyu Sudiatmika [1,*], Gede Indrawan [2], Sariyasa [3]

Universitas Pendidikan Ganesha,
Jl. Udayana No.11, Buleleng 81116, Indonesia
[1] wahyu.sudiatmika@undiksha.ac.id; [2] gindrawan@undiksha.ac.id; [3] sariyasa@undiksha.ac.id*
** corresponding author*

ARTICLE INFO

Article history:
Received 21 July 2023
Revised 21 August 2023
Accepted 18 September 2023
Published online 22 September 2023

Keywords:
Load-balancing
Round-Robin Algorithm
Server on-premise
Performance
Public service application

ABSTRACT

The Provincial Government of Bali assumes a crucial role in administering various public service applications to meet the requirements of its community, traditional villages, and regional apparatus. Nevertheless, the escalating magnitude of traffic and uneven distribution of requests have resulted in substantial server burdens, which may jeopardize the operation of applications and heighten the likelihood of downtime. Ensuring efficient load distribution is of utmost importance in tackling these difficulties, and the Round Robin algorithm is often utilized for this purpose. However, the current body of research has not extensively examined the distinct circumstances surrounding on-premise servers in the Bali Provincial Government. The primary objective of this study is to address the significant gap in knowledge by conducting a comprehensive evaluation of the Round Robin algorithm's effectiveness in load-balancing on-premise servers inside the Bali Provincial Government. The primary objective of our study is to assess the appropriateness of the algorithm within the given context, with the ultimate goal of providing practical and implementable suggestions. The observations above can optimize system efficiency and minimize periods of inactivity, thereby enhancing the provision of vital public services across Bali. This study provides essential insights for enhancing server infrastructure and load-balancing strategies through empirical evaluation and comprehensive analysis. Its findings are valuable for the Bali Provincial Government and serve as a reference for other organizations facing challenges managing server loads. This study signifies a notable advancement in establishing reliable and practical public service applications within Bali.

I. Introduction

The Bali Provincial Government currently operates many public service applications integral to the lives of its residents, local villages, and regional apparatus. Prominent among these systems are the Traditional Village Financial Management System (in Indonesian, Sistem Informasi Keuangan Desa Adat or SIKUAT), the Civil Service System (in Indonesian, Sistem Manajemen Kepegawaian or SIMPEG), the Virtual Office (e-Office), and the Electronic Procurement System (in Indonesian, Sistem Pengadaan Secara Elektronik or SPSE). With these systems operating on single, on-premise servers, the challenge of resource limitation becomes increasingly apparent. A single server has finite CPU, RAM, storage, and bandwidth. Overloading a server with multiple systems can lead to performance degradation or crashes. Furthermore, the security of all the systems becomes jeopardized if one system on the server is compromised.

Robust server management techniques, like load-balancing and virtualization, become crucial to alleviate these issues. Load balancing, by definition, is the distribution of a workload across multiple servers, ensuring that no singular server bears an overwhelming load [1]. This process optimizes and stabilizes system performance, ensuring maximum uptime and consistent service delivery. Among the strategies employed for load-balancing, the Round Robin algorithm stands out. This algorithm systematically assigns incoming server requests to the next server in line, ensuring an equitable

https://doi.org/10.17977/um018v6i12023p79-91

distribution [2]. However, a significant gap exists: no existing research evaluating the performance of the Round Robin algorithm specifically within the Bali Provincial Government's on-premise server context exists.

This work undertakes a novel and groundbreaking investigation to address a significant gap in load balancing. The primary aim of this study is to examine the efficacy of the Round Robin algorithm within the specific context of on-premise servers used by the province government of Bali. This particular domain has been noticeably underrepresented in previous research efforts.

Our research aims to offer the Bali Provincial Government carefully crafted recommendations based on rigorous information and specifically customized to their distinct server environment. This study gives particular attention to assessing and examining the round-robin methodology. This study aims to precisely construct a framework that maximizes the operational efficiency of the National Data Centers managed by the Bali Provincial Government.

This work distinguishes itself via its innovative approach, as it explores hitherto unexplored domains to tackle the urgent requirement for server optimization within a specific and intricate real-world context. Through a thorough examination of the Round Robin algorithm's appropriateness for this unique context, our objective is to offer fresh perspectives and remedies that can be utilized not only by the Bali Provincial Government but also serve as a valuable point of reference for comparable entities grappling with comparable obstacles in their management of server infrastructure. This research has the potential to impact the domain of load balancing and server optimization substantially, hence facilitating the development of more efficient and robust server environments in the coming years.

II. Method

The research design commences with the collection of data, which is subsequently followed by the formulation of test cases, the testing of these test cases, and the analysis of the obtained test results [3][4][5][6]. The initial phase entails the identification of the system environment and infrastructure that will undergo testing, the collection of pertinent information regarding the application to be evaluated, and the establishment of the test's objectives and requirements. The subsequent phase involves the formulation of test cases for every test scenario, with the objective of including all crucial facets of the application and system environment. The third phase entails the execution of test cases based on specified scenarios, the documentation of test outcomes, and the verification of their alignment with the anticipated results. The concluding phase involves the examination of the test outcomes and their comparison with the objectives and requirements of the test. This process entails the identification of any issues or flaws in the application or system environment, followed by the implementation of the requisite enhancements or optimizations. In general, the study design is implemented to assure the systematic and rigorous execution of tests, hence generating dependable outcomes that can be utilized for the advancement of system development. The processes outlined in Figure 1 provide a more comprehensive and deep understanding.

From Figure 1, in more detail, the research steps are described as follows. First, the data collection stage, is one of the initial stages in conducting research. Data collection is carried out to collect information and data relevant to the research problem to be solved. In this stage, the method of data collection that will be carried out is a literature study and interviews. Second steps is preparation of test cases. The test case preparation stage is essential in load-balancing research using the Round Robin algorithm on an on-premise server in the Bali provincial government. This stage aims to create a series of test cases that are used to test the performance of the Round Robin algorithm under various conditions. Next step is test case testing, at this stage, the researcher will test several test cases that have been prepared previously in the test case preparation stage. From the results of the test cases, it is hoped that information will be obtained about the performance and suitability of the Round Robin algorithm in an on-premise server environment at the Bali Provincial government so that it can provide recommendations to the Bali Provincial government regarding the most suitable load-balancing algorithm to use.

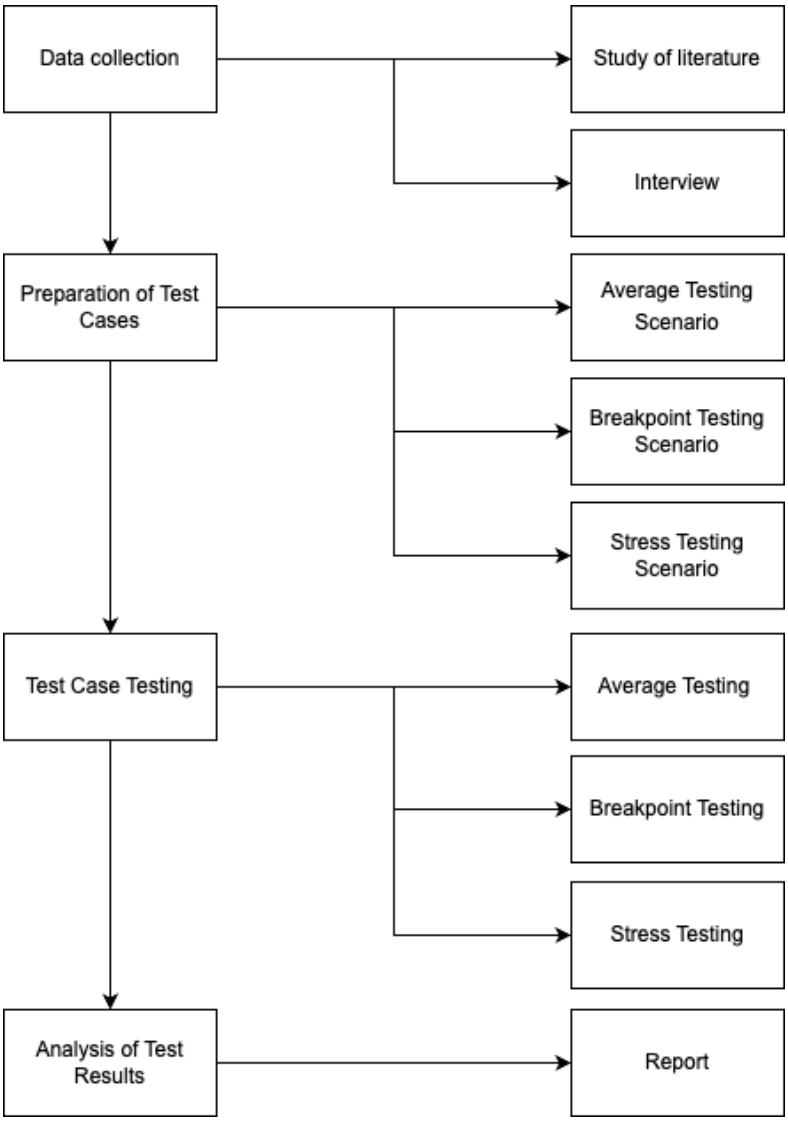

Fig. 1. Research steps

In the last stage, results of the test case test that have been carried out before will be analyzed in depth to determine the performance of the Round Robin algorithm in load-balancing on the server on-premise of the Bali Provincial government. This analysis will include an evaluation of load testing, failover testing, robustness testing, and security testing. Based on the results of the analysis from this stage, the researcher will conclude the advantages and disadvantages of the Round Robin algorithm in load-balancing on the server on the premise of the Bali Provincial Government and provide recommendations regarding the most suitable load-balancing algorithm used in an on-premise server environment.

III. Result and Discussion

This study aims to analyze the performance of the round-robin algorithm in a load balancer. In this study, we tested the performance of the round-robin algorithm in selecting the destination server for each incoming request.

The Test Case will use standard testing from Grafana Labs K6 [7][8][9][10][11][12][13]. Based on Grafana Labs documentation, K6 is an open-source load balancer testing tool that simplifies and increases performance testing productivity for cloud technicians and engineers. The following tests will be carried out based on the Grafana Labs K6 standard.

The assessment of a load balancer's performance under conditions that roughly resemble its regular workday load is a crucial benchmark, also referred to as average-load testing. This testing method offers significant insights into the ability of the load balancer to achieve its performance targets during regular operations continuously [14][15][16][17][18].

Figure 2 visually depicts the outcomes derived from the Average-load testing, illustrating our findings. The presented testing scenario portrays an environment that exhibits a typical workload, which closely resembles the load balancer's actual use during regular weekdays. Moreover, it illustrates a moderate labor duration, providing insight into the time required to handle and allocate incoming requests effectively.

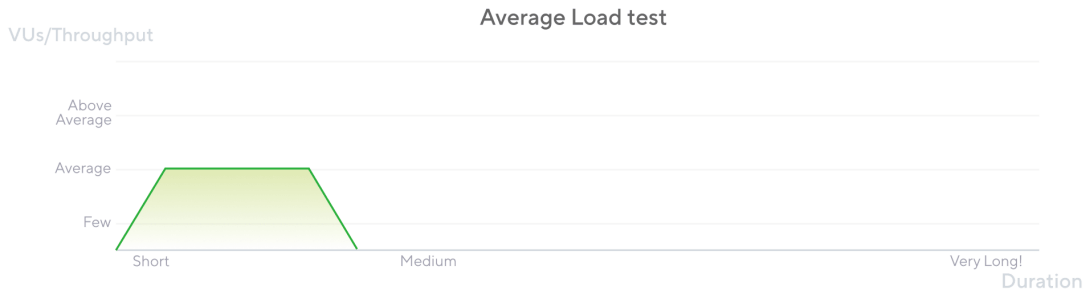

Fig. 2. Average-load test

The insights obtained by doing Average-load testing provide a practical understanding of the load balancer's capacity to manage the routine demands it faces effectively. Through simulating common use patterns, researchers can get a more comprehensive knowledge of the load balancer's performance within a context that closely aligns with its practical reality. Understanding this information is crucial in guaranteeing that the load balancer can continuously and effectively fulfill the requirements of the systems it assists throughout regular operations, hence improving the system's overall stability and user satisfaction.

The stress testing process, as seen in Figure 3, is a crucial stage in assessing the resilience and performance of a load balancer under extreme loads that exceed standard usage patterns. This testing methodology comprehensively evaluates the load balancer's capacity to uphold system stability and consistent reliability while subjected to intense stress levels [19][20][21][22][23][24][25][26][27].

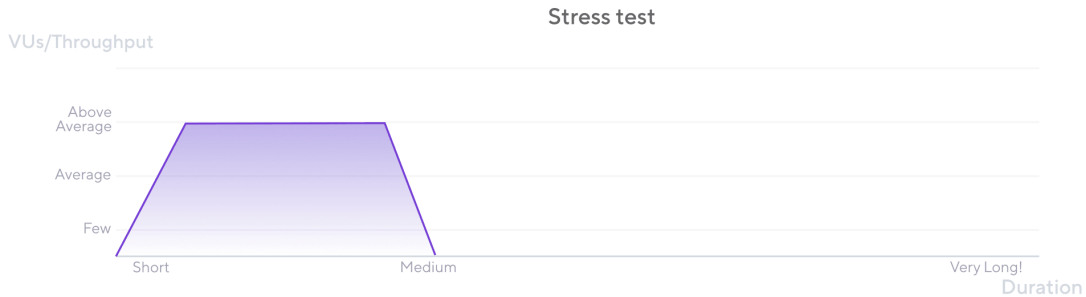

Fig. 3. Stress test

Figure 3 provides a visual representation of the stress testing results, effectively illustrating the responsiveness of the load balancer under extreme conditions. This scenario's purpose is to impose excessive demands on the system deliberately, so replicating instances of high usage or unanticipated surges in traffic to identify vulnerabilities, bottlenecks, or possible failure sites.

By putting the load balancer to these increased conditions, researchers can obtain vital insights regarding its resilience and ability to manage unexpected increases in user activity, ensuring the continuous provision of services. The stress testing process is of utmost importance in enhancing the load balancer's performance characteristics, ensuring its ability to withstand and remain robust in highly demanding use scenarios. These insights are crucial for enterprises aiming to uphold high

availability and ensure smooth user experiences, particularly in times of increased demand or unforeseen swings in traffic.

The process of breakpoint testing, as illustrated in Figure 4, is a critical undertaking aimed at precisely identifying the underlying constraints inside a system. The justification for doing breakpoint testing is complex and involves a range of compelling factors, all of which contribute considerably to the overall durability and strength of the system [28][29][30][31][32][33][34][35][36].

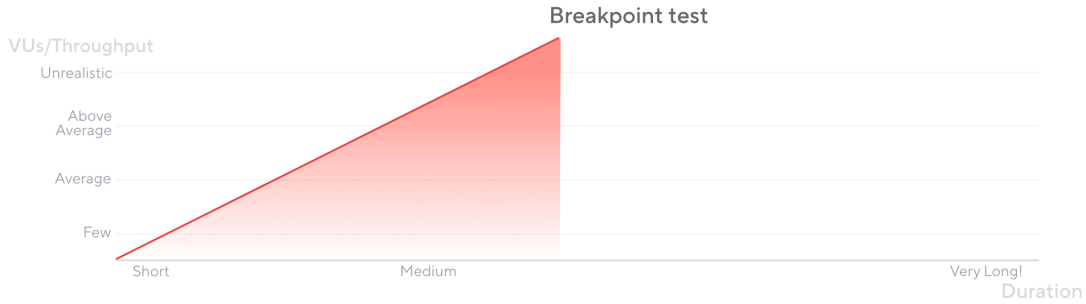

Fig. 4. Breakpoint test

Primarily, breakpoint testing plays a crucial role in proactive planning. Organizations can obtain valuable insights into the operating boundaries of the system by intentionally submitting the load balancer to progressively significant loads until it approaches its breakpoint. This information is the basis for developing thorough remediation techniques for load balancer failures or catastrophic system overloads. With this understanding, companies can establish predetermined measures for mitigating risks, resulting in decreased periods of inactivity, limited service disruptions, and the assurance of a prompt and efficient reaction to obstacles.

Moreover, the utilization of breakpoint testing is of utmost importance in the process of protocol creation. This capability enables businesses to optimize response protocols by refining the methods and procedures necessary to address prospective challenges. The use of a proactive strategy is crucial in the identification and preventative resolution of vulnerabilities, hence enhancing the overall dependability and stability of the system.

It is essential to acknowledge that breakpoint testing is a methodical and regulated procedure. The demand is gradually augmented until the load balancer nears its breakpoint, at this juncture, the test is manually terminated to mitigate any potential server harm. This cautious strategy guarantees the system's reliability while allowing enterprises to collect vital data about system performance and constraints.

Breakpoint testing is fundamentally a strategic endeavor that enhances system resilience and optimizes performance. This technology enables enterprises to effectively manage the intricacies of load balancing, instilling them with a sense of assurance in their ability to address obstacles proactively, mitigate interruptions, and provide uninterrupted service quality to their consumers.

Before conducting the test, a scenario will be created for each test case. The test scenarios are Average-load testing, stress testing, and breakpoint testing.

In the context of load testing, our objective is to accurately simulate the dynamic patterns of user interactions with the load balancer through a meticulously constructed average-load testing scenario. This process is conducted with a high degree of control and methodical precision. The initial stage, which involves the progressive inclusion of people individually over 5 minutes, closely resembles the natural accumulation of user engagement during typical usage. This phase enables a detailed observation of the load balancer's response to incremental requests, enabling an assessment of its capacity to effectively distribute resources and sustain minimal delay as the number of users progressively increases. Additionally, this provides valuable information regarding the load balancer's handling of the initial surge of connections, which is a critical factor in guaranteeing a smooth user experience during times of increased demand.

The succeeding step involves the simultaneous engagement of 100 users with the load balancer for 10 minutes, which acts as a critically significant stress test. This rigorous phase simulates situations in which the system becomes overwhelmed due to abrupt increases in traffic, such as the dissemination of viral content or the execution of marketing campaigns. By putting the load balancer to a period of high demand, we can evaluate its capacity to effectively manage substantial workloads while ensuring optimal performance, uptime, and resource allocation. This phase assesses not only the technical capabilities of the load balancer but also its ability to maintain service quality under challenging circumstances, therefore mitigating the risk of service interruptions during periods of high demand [37].

Figure 5 provides a comprehensive visual depiction of the dynamic scenario, effectively illustrating the entire testing procedure, facilitating comprehension of the many stages of testing and serving as a framework for interpreting and analyzing results. By employing carefully designed testing scenarios and utilizing visual aids, businesses can obtain an in-depth understanding of the load balancer's functionality, enabling them to make informed decisions based on data analysis, aiming to improve system performance and resilience.

```
import http from 'k6/http';
import {sleep} from 'k6';

export const options = {
    // Key configurations for avg load test in this section
    stages: [
        { duration: '5m', target: 100 }, // traffic ramp-up from 1 to 100 users over 5
minutes.{ duration: '10m', target: 100 }, // stay at 100 users for 10 minutes
        { duration: '5m', target: 0 }, // ramp-down to 0 users
    ],
};

export default () => {
    http.get('https://apps1.baliprov.go.id');
    sleep(1);
};
```

Fig. 5. Scenario average-load testing

Figure 6 represents a pivotal juncture when intentional and significant pressure is applied to the load balancer, resulting in a massive surge of incoming traffic. The simulation provided in this study aims to recreate real-world scenarios where sudden and quick surges in user activity can place substantial pressure on the system's resources and capabilities. The initiation of stress testing entails users systematically consecutively accessing the load balancer for an extended duration of 10 minutes, progressively augmenting the user count to a substantial aggregate of 200 individuals. The progressive incorporation of users underscores the load balancer's capacity to adjust to an ever-expanding user load while maintaining consistent performance metrics [38].

```
import http from 'k6/http';
import {sleep} from 'k6';

export const options = {
    // Key configurations for Stress in this section
    stages: [
        { duration: '10m', target: 200 }, // traffic ramp-up from 1 to a higher 200 users over 10
minutes.
        { duration: '10m', target: 200 }, // stay at higher 200 users for 10 minutes
        { duration: '5m', target: 0 }, // ramp-down to 0 users
    ],
};

export default () => {
    http.get('https://apps1.baliprov.go.id');
    sleep(1);
};
```

Fig. 6. Stree testing scenario

Once the user count exceeds the critical threshold of 200, the situation transitions into a phase marked by a prolonged duration of heightened demand, wherein intensive demands persist for 10 minutes. This phase replicates scenarios involving high stress levels, during which system resources are entirely used. During this phase, a comprehensive analysis is performed on critical performance indicators, encompassing reaction times, resource utilization, and error rates. The data produced presents valuable information into the load balancer's capacity to effectively manage heavy workloads while maintaining service quality at a satisfactory level [39].

The culmination of the stress testing scenario occurs when users systematically complete their requests within a 5-minute timeframe, resulting in a gradual decrease in user burden. The phenomenon that has been noticed demonstrates a decline in user involvement that naturally occurs after increased demand. This observation offers valuable insights about the load balancer's capacity to manage the reduction in incoming requests efficiently. Organizations can enhance their comprehension of the load balancer's performance in high-stress conditions by employing visual representations of stress testing scenarios. These insights are of paramount importance for companies seeking to enhance the resilience of their systems against unexpected surges in user traffic and ensure uninterrupted service delivery, especially in peak demand.

The Break Point Test scenario, as illustrated in Figure 7, is a critical stage within our extensive testing protocol. This scenario aims to methodically evaluate the capabilities and thresholds of the load balancer, especially when confronted with a continuous and substantial increase in user traffic. The process commences with a notable influx of 20,000 users consistently visiting the load balancer, persistently exerting pressure on its capacities until a threshold is reached. This phase aims to determine the specific threshold at which the load balancer's performance begins to deteriorate or is compromised when subjected to high-load situations [40].

```
import http from 'k6/http';
import {sleep} from 'k6';

export const options = {
    // Key configurations for breakpoint in this section
    executor: 'ramping-arrival-rate', //Assure load increase if the system slows
    stages: [
        { duration: '60m', target: 20000 }, // just slowly ramp-up to a HUGE
load],
};

export default () => {
    http.get('https://apps1.baliprov.go.id');
    sleep(1);
};
```

Fig. 7. Breakpoint testing scenario

In order to do thorough examinations and verify the results, we utilize the advanced Grafana Labs K6 testing tool. This tool facilitates the precise execution of tests, ensuring adherence to specified scenarios that faithfully replicate real-world usage patterns. The use of Grafana Labs K6 guarantees that both control and representation of genuine user behavior characterize our testing methodology, allows us to extract significant insights into the load balancer's performance in diverse scenarios.

In addition, the testing procedure on the server side is closely monitored by utilizing Kibana Data Analytics tools. Using a dual-monitoring technique functions as a reliable validation mechanism, enabling cross-referencing and verifying the outcomes derived by Grafana Labs K6. By utilizing the sophisticated analytics features of Kibana, a comprehensive understanding of the load balancer's performance can be obtained, including evaluating resource use, response times, and error rates [41].

The testing protocol we employ is characterized by its rigorous nature, resulting in substantial data and valuable insights. These findings are meticulously arranged and effectively communicated

through a collection of tables. The tables above encompass the Test Results in Average-load Testing (Table 1), Test Results of Stress Testing (Table 2), and Breakpoint Testing Results (Table 3). Utilizing a tabular format facilitates the seamless comparison and analysis of crucial performance parameters, enabling the process of making well-informed decisions and implementing optimization methods for the load balancer and its connected systems.

Table 1 provides a comprehensive analysis demonstrating the constant and reliable performance of the Round Robin algorithm across several vital parameters. Notably, this algorithm exhibits exceptional proficiency in processing HTTP requests and establishing secure connections. The consistent capacity to produce expected outcomes highlights its appropriateness for the server environment of the Bali Provincial Government, where the utmost importance is placed on stability and dependability.

Table 1. Test results in average-load testing

Algorithm	Round Robin	IP Hash
data_received.	243 MB 270 kB/s	218 MB 330 kB/s
data_sent.	32 MB 35 kB/s	108 MB 43 kB/s
http_reg_blocked	avg=131.88ms min=3us med=115.75ms med=115.75ms max=2.35s p(90)=165.48m p(95)=190 .69ms	avg=140.85ms min=1µs med=117.38ms max=57.57s p(90)=185.54ms p(95)=223.84ms
http_req_connecting	avg=9.12ms min=0s med=0s max=1.035 p(90)=17.99ms p(95) -24.65 ms	avg=7.76ms min=0s med=0s max=3.03s p(90)=14.91ms p(95)=22.45ms
http_req_duration	avg=109.32ms min=21.92ms med=114.79ms max=2.39s p(90)=199. 14ms p(95)=226 .37 ms	avg=154.45ms min=20.54ms med=121.04ms max=1m0s p(90)=216.97ms p(95)=262.81ms
{expected_response: true}	avg=109.32ms min=21.92ms med=114.79ms max=2.39s p(90)=199. 14ms p(95)=226.37 ms	avg=153.07ms min=20.54ms med=121.04ms max=57.6s p(90)=216.96ms p(95)=262.77ms
http_req_failed	0.00% / 0 × 51290	0.00% ✓ 4 ✗ 172777
http_req_receiving	avg=44.16ps min=5us med=25us max=6.13ms p(90)=100 us p(95)=123us	avg=37.75µs min=0s med=18µs max=21.62ms p(90)=78µs p(95)=126µs
http_req_sending	avg=72.87ms min=3us med=86.91 ms max=2.36s p(90)=159.74ms p(95)=182.22ms	avg=116.84ms min=3µs med=92.41ms max=57.58s p(90)=177.1ms p(95)=214.64ms
http_req_tls_handshaking	avg=54.33ms min=0s med=0s max=1.49s p(90)=126.68ms p(95)=141.39ms	avg=53.93ms min=0s med=0s max=45.91s p(90)=133.34ms p(95)=157.43ms
http_req_waiting	avg=36.41ms min=21.8ms med=31.65ms max=2s p(90)=49.24ms p(95)=59.91ms	avg=37.57ms min=20.5ms med=30.21ms max=1m0s p(90)=49.2ms p(95)=63.94ms
http_regs	51290 56.925729/s	172781 69.609188/s
iteration duration	avg=1.17s min=1.07s med=1.15s max=3.39s p(90)=1.21s p(95)=1.25s	avg=1.21s min=1.04s med=1.15s max=1m1s p(90)=1.24s p(95)=1.3s
iterations	51290 56.925729/s	172781 69.609188/s
vus	1 min=1 max=100	1 min=1 max=100
vus max	100 min=100 max=100	100 min=100 max=100

On the other hand, the IP Hash algorithm demonstrates its advantages in data transmission rates and its ability to handle a larger number of requests per second effectively. These characteristics make it appealing when the primary focus is on swift data delivery. Nevertheless, it is essential to acknowledge that compromises in other aspects of performance accompany these benefits.

The data obtained from these experiments provides a comprehensive understanding of the performance of both methods, demonstrating distinct strengths in various load-balancing aspects. Although both Round Robin and IP Hash have their advantages, the predominant data indicates that Round-Robin's constant and dependable performance establishes it as the preferable option inside

the server ecology of the Bali Provincial Government. Nevertheless, it is essential to consider the individual deployment and use-case needs, as they may necessitate a more nuanced conclusion. Therefore, more research should be conducted to examine these aspects and offer more customized advice the government's servers.

The findings reported in Table 2 demonstrate the superior performance of the round-robin algorithm compared to the IP Hash technique across all critical performance criteria. Significantly, the Round Robin algorithm demonstrates exceptional performance in connection times, request lengths, and overall efficiency in effectively handling HTTP requests. Consistent superior outcomes across all crucial factors establish Round Robin as the optimum solution for enhancing performance in the tested setting.

Table 2. Test results of stress testing

Algorithm	Round Robin	IP Hash
data_received.	787 MB 525 kB/s	1.1 GB 162 kB/s
data_sent.	102 MB 68 kB/s	151 MB 22 kB/s
http_reg_blocked	avg=194.77ms min=0s med=166.53ms max=5.68s p(90)-246.5ms p(95)=315.45ms	avg=920.09ms min=0s med=589.27ms max=16m48s p(90)=1.06s p(95)=1.12s
http_req_connecting	avg=16.67ms min=0s med=0s max=1. 1s p(90)=36.26ms p(95)=49.9ms	avg=277.56ms min=0s med=0s max=16m48s p(90)=68.84ms p(95)=118.92ms
http_req_duration	avg=171.3ms min=0s med=141.69ms max=59.97s p(90)=302.22ms p(95)=377.21ms	avg=4.72s min=0s med=228.01ms max=56m54s p(90)=1.22s p(95)=1.38s
{expected_response: true}	avg=167.37ms min=21.42ms med=141.66ms max=5.75s p(90)=302.16ms p(95)=377.06ms	avg=2.16s min=20.51ms med=225.44ms max=40m7s p(90)=1.22s p(95)=1.32s
http_req_failed	0.00% / 14 × 166217	0.66% ✓ 1592 ✗ 236692
http_req_receiving	avg=38.17us min=0s med=18us max=579.45ms p(90)=68us p(95)=94us	avg=25.75µs min=0s med=19µs max=11.5ms p(90)=38µs p(95)=53µs
http_req_sending	avg=114.06ms min=0s med=93.2ms max=5.7s p(90)=241.33ms p(95)=298. 96ms	avg=2.04s min=0s med=136.15ms max=40m7s p(90)=1.12s p(95)=1.17s
http_req_tls_handshaking	avg=75.22ms min=0s med=0s max=4.77s p(90)=172.42ms p(95)=198.03ms	avg=183.01ms min=0s med=0s max=10m54s p(90)=863.72ms p(95)=963.06ms
http_req_waiting	avg=57.2ms min=0s med=43.39ms max=59.52s p(90) =80. 37ms p(95)=100.45ms	avg=2.68s min=0s med=60.85ms max=56m54s p(90)=181.39ms p(95)=325.93ms
http_regs	166231 110.778429/s	238284 34.517602/s
iteration duration	avg=1.26s min=1.06s med=1.22s max=1m1s p(90)=1.34s p(95)=1.44s	avg=2.2s min=1.03s med=1.87s max=1m1s p(90)=2.24s p(95)=2.55s
iterations	166231 110.778429/s	238282 34.517312/s
vus	1 min=1 max=200	1 min=1 max=200
vus max	200 min=200 max=200	200 min=200 max=200

In sharp contrast, despite its higher overall data processing and repetition, the IP Hash algorithm has a significantly distinct profile. It is characterized by significantly reduced speeds, prolonged waiting periods, and an increased frequency of request failures. The deficiencies above prove the system's constraints in providing expeditious and prompt service, a crucial factor for consumers in a rapidly evolving digital environment.

The information presented in Table 3 is quite explicit and without significant ambiguity. Due to its demonstrated dependability and effectiveness in managing key activities, the round-robin scheduling algorithm is unequivocally favored for optimizing performance within the specific context under examination. Nevertheless, it is advisable to consider the precise operational demands

and use circumstances since these factors may need a more intricate decision-making process when deploying load-balancing solutions. It is imperative to do more investigation into these intricate situations to offer complete and customized suggestions for selecting an ideal load balancer.

Table 3. Breakpoint testing results

Algorithm	Round Robin	IP Hash
data_received.	20 MB 37 kB/s	20 MB 27 kB/s
data_sent.	2.4 MB 4.4 kB/s	2.4 MB 3.1 kB/s
http_reg_blocked	avg=114.64ms min=75.82ms med=101.49ms max=1.13s p(90)=148.8ms p(95)=212.63ms	avg=106.5ms min=0s med=92.05ms max=3.46s p(90)=106.75ms p(95)=117.64ms
http_req_connecting	avg=30.68ms min=20.57ms med=25.37ms max=328.1ms p(90)=41.57ms p(95)=56.4ms	avg=13.77ms min=0s med=5.45ms max=2.01s p(90)=9.76ms p(95)=13.83ms
http_req_duration	avg=32.07ms min=21.58ms med=26.52ms max=441.87ms p(90)=42.18ms p(95)=55.14ms	avg=72.18ms min=0s med=27.29ms max=18.38s p(90)=31.97ms p(95)=36.02ms
{expected_response: true}	avg=32.07ms min=21.58ms med=26.52ms max=441.87ms p(90)=42.18ms p(95)=55.14ms	avg=72.19ms min=21.89ms med=27.29ms max=18.38s p(90)=31.97ms p(95)=36.02ms
http_req_failed	0.00% ✓ 0 ✗ 4259	0.02% ✓ 1 ✗ 4258
http_req_receiving	avg=156.23µs min=31µs med=139µs max=4.04ms p(90)=198µs p(95)=242µs	avg=159.25µs min=0s med=139µs max=8.04ms p(90)=214µs p(95)=266.09µs
http_req_sending	avg=140.53µs min=23µs med=125µs max=7.45ms p(90)=190µs p(95)=229µs	avg=43.14ms min=0s med=132µs max=18.35s p(90)=209µs p(95)=294.19µs
http_req_tls_handshaking	avg=83.66ms min=53.25ms med=72.74ms max=1.07s p(90)=101.44ms p(95)=144.09ms	avg=92.13ms min=0s med=85.78ms max=3.45s p(90)=97.23ms p(95)=105.03ms
http_req_waiting	avg=31.77ms min=21.36ms med=26.2ms max=441.6ms p(90)=41.79ms p(95)=54.88ms	avg=28.87ms min=0s med=26.96ms max=839.28ms p(90)=31.53ms p(95)=35.32ms
http_regs	4259 7.884463/s	4259 5.59725/s
iteration duration	avg=1.14s min=1.09s med=1.13s max=2.18s p(90)=1.19s p(95)=1.26s	avg=1.19s min=1.09s med=1.12s max=1m1s p(90)=1.14s p(95)=1.15s
iterations	4259 7.884463/s	259 5.59725/s
vus	1 min=1 max=13	1 min=1 max=25
vus max	50 min=50 max=50	50 min=50 max=50

The results shown in Table 3 highlight the Round-Robin method's superior performance compared to the IP Hash alternative across several essential criteria. Significantly, the Round-Robin algorithm demonstrates exceptional proficiency in data transmission speed, the duration of requests, and the efficient execution of iterations. The constant and excellent performance of Round-Robin in these crucial areas makes it a tempting option for optimizing load balancing within the dataset under evaluation.

The IP Hash method demonstrates notable strengths in specific measures such as request blocking and connection delays. However, the Round-Robin algorithm emerges as the most advantageous option when examining the overall performance profile. The selection between the two algorithms is contingent upon the particular priorities and exigencies of the given use case since each method possesses distinct strengths and trade-offs.

Based on the extensive data provided, it can be concluded that the Round-Robin algorithm has superior efficiency across all dimensions, rendering it a highly appealing alternative for enterprises aiming to optimize their load-balancing techniques. Nonetheless, it is crucial to ensure that the choice of algorithm follows the unique performance goals and operational limitations of the given context,

emphasizing the significance of customized approaches in the load-balancing domain. Additional inquiry and contextual analysis can potentially enhance this judgment's precision significantly.

IV. Conclusion

Through our extensive examination of the IP Hash and Round Robin algorithms, we have garnered significant insights that can contribute to advancing future research endeavors and provide practical guidance for their implementation. Concerning data transfer rates, the IP Hash method demonstrated a marginal superiority based on the average outcomes of the conducted tests. Nevertheless, Round-Robin has shown to be a more reliable option, especially in terms of managing HTTP requests and secure connections. The stability and dependability of Round-Robin were further emphasized during stress testing, as it continually surpassed IP Hash across several performance parameters. Significantly, Round Robin exhibited enhanced connection times, request durations, and overall efficiency in managing HTTP requests. In the breakpoint test, the level of competition between the two algorithms was more evenly balanced. Both IP Hash and Round- Robin algorithms handled comparable data amounts. However, Round Robin exhibited superior data transmission rates. Although IP Hash showed superior performance in request blocking and connection delays, Round Robin once again showcased its proficiency in the crucial realm of HTTP request handling and iteration processing rates. Upon examining the collective results obtained from the three tests, it becomes apparent that the Round-Robin algorithm exhibits superior performance, consistency, and reliability compared to the IP Hash method. Although IP Hash showed capabilities in certain areas, Round-Robin consistently beat it across a broader range of performance criteria.

When businesses or entities are confronted with the decision between these two algorithms in prospective research, it is highly recommended that Round-Robin be given significant consideration due to its equitable and efficient performance, which holds particular significance when the utmost importance is placed on maintaining consistency and reliability. Nevertheless, it is important to acknowledge that individual use cases and distinct requirements may influence the final selection. Hence, it is recommended that future research endeavors undertake a more comprehensive investigation of these particular cases to offer additional insights and recommendations for the selection and implementation of algorithms.

Declarations

Author contribution

All authors contributed equally as the main contributor of this paper. All authors read and approved the final paper.

Funding statement

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Conflict of interest

The authors declare no known conflict of financial interest or personal relationships that could have appeared to influence the work reported in this paper.

Additional information

Reprints and permission information are available at http://journal2.um.ac.id/index.php/keds.

References

[1] A. Hanafiah, "Implementasi Load Balancing Dengan Algoritma Penjadwalan Weighted Round Robin Dalam Mengatasi Beban Webserver," IT J. Res. Dev., vol. 5, no. 2, pp. 226–233, Jan. 2021.

[2] Y. Arta, "Penerapan Metode Round Robin Pada Jaringan Multihoming Di Computer Cluster," IT J. Res. Dev., vol. 1, no. 2, pp. 26–35, Aug. 2017.

[3] T. D. Putra and R. Purnomo, "Average Max Round Robin Algorithm: A Case Study," Sinkron, vol. 8, no. 3, pp. 1230–1237, Jul. 2023.

[4] R. Purnomo and T. D. Putra, "Comparison Between Simple Round Robin and Improved Round Robin Algorithms," JATISI (Jurnal Tek. Inform. dan Sist. Informasi), vol. 9, no. 3, pp. 2205–2221, Sep. 2022.
[5] R. Sharma, A. K. Goel, M. K. Sharma, N. Dhiman, and V. N. Mishra, "Modified Round Robin CPU Scheduling: A Fuzzy Logic-Based Approach," in Lecture Notes in Operations Research, 2023, pp. 367–383.
[6] A. Y. Ahmad, "An Attempt to Set Standards for Studying and Comparing the Efficiency of Round Robin Algorithms," J. Educ. Sci., vol. 32, no. 2, pp. 11–20, Jun. 2023.
[7] B. Manasa and A. R. Babu, "Dynamic Weighted Round Robin Approach in Software-Defined Networks Using Pox Controller," Int. J. Recent Innov. Trends Comput. Commun., vol. 11, no. 5, pp. 304–310, May 2023.
[8] S. E. Abubakar, "Modified Round Robin with Highest Response Ratio Next CPU Scheduling Algorithm using Dynamic Time Quantum," SLU J. Sci. Technol., pp. 87–99, Mar. 2023.
[9] D. Biswas, M. Samsuddoha, M. R. Al Asif, and M. M. Ahmed, "Optimized Round Robin Scheduling Algorithm Using Dynamic Time Quantum Approach in Cloud Computing Environment," Int. J. Intell. Syst. Appl., vol. 15, no. 1, pp. 22–34, Feb. 2023.
[10] M. A. S. Al-Mekhlafi and N. N. S. Al-Marbe, "Lower and Upper Quartiles Enhanced Round Robin Algorithm for Scheduling of Outlier Tasks in Cloud Computing," J. Eng. Technol. Sci. - JOEATS, vol. 1, no. 1, pp. 67–87, Mar. 2023.
[11] W. Ullah and M. A. Shah, "A novel resilent round robin algorithm based CPU scheduling for efficient CPU utlilization," in Competitive Advantage in the Digital Economy (CADE 2022), 2022, pp. 41–48.
[12] Y. Afrianto, H. Sukoco, and S. Wahjuni, "Weighted Round Robin Load Balancer to Enhance Web Server Cluster in OpenFlow Networks," TELKOMNIKA (Telecommunication Comput. Electron. Control., vol. 16, no. 3, p. 1402, Jun. 2018.
[13] H. M. Noman and M. N. Jasim, "A Comparative Performance Analysis for Static and Dynamic Load Balancing Techniques in Software Defined Network Environment," J. Phys. Conf. Ser., vol. 1773, no. 1, p. 012010, Feb. 2021.
[14] T. Chomsiri and D. Pansa, "Load Balancer Mechanism using Optimal Parameter based on Calculus," in 2018 International Conference on Information Technology (InCIT), Oct. 2018, pp. 1–6.
[15] M. A. N. Saif, S. K. Niranjan, B. A. H. Murshed, F. A. Ghanem, and A. A. Q. Ahmed, "CSO-ILB: chicken swarm optimized inter-cloud load balancer for elastic containerized multi-cloud environment," J. Supercomput., vol. 79, no. 1, pp. 1111–1155, Jan. 2023.
[16] K. K. Azumah, P. R. M. Maciel, L. T. Sørensen, and S. Kosta, "Modeling and Simulating a Process Mining-Influenced Load-Balancer for the Hybrid Cloud," IEEE Trans. Cloud Comput., vol. 11, no. 2, pp. 1999–2010, Apr. 2023.
[17] R. Uddin and F. Monir, "Performance Evaluation of Ryu Controller with Weighted Round Robin Load Balancer," in Communications in Computer and Information Science, 2021, pp. 115–129.
[18] K. Takahashi, K. Aida, T. Tanjo, and J. Sun, "A Portable Load Balancer for Kubernetes Cluster," in Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, Jan. 2018, pp. 222–231.
[19] O. Khoshaba, V. Lytvynov, V. Grechaninov, and K. Zavertailo, "Performance of the Reverse Load Balancer Method in Cluster and Cloud Infrastructures," in Advances in Intelligent Systems and Computing, 2021, pp. 186–196.
[20] X. Huang, Z. Guo, and M. Song, "FGLB: A fine-grained hardware intra-server load balancer based on 100 G FPGA SmartNIC," Int. J. Netw. Manag., vol. 32, no. 6, Nov. 2022.
[21] S. Mangalampalli, P. K. Sree, K. V. N. Rao, A. Rapaka, and R. T. Kocherla, "Prioritized Load Balancer for Minimization of VM and Data Transfer Cost in Cloud Computing," in Advances in Intelligent Systems and Computing, 2022, pp. 263–271.
[22] S. Atalla, A. Bianco, R. Birke, and L. Giraudo, "A Hardware Load Balancer for a Multi-Stage Software Router Architecture (Sep. 17)," in 2014 World Congress on Computer Applications and Information Systems, WCCAIS 2014, 2022, no. July.
[23] W. W. Mulat, S. K. Mohapatra, R. Sathpathy, and S. K. Dhal, "Improving Throttled Load Balancing Algorithm in Cloud Computing," in Algorithms for Intelligent Systems, 2022, pp. 369–377.
[24] M. Park, J. Seok, and K. Lee, "A SIP Load Balancer for Performance Enlargement," 2022.
[25] S. S. Tripathy, D. S. Roy, and R. K. Barik, "M2FBalancer: A mist-assisted fog computing-based load balancing strategy for smart cities," J. Ambient Intell. Smart Environ., vol. 13, no. 3, pp. 219–233, May 2021.
[26] N. G. Elnagar, G. F. Elkabbany, A. A. Al-Awamry, and M. B. Abdelhalim, "Simulation and performance assessment of a modified throttled load balancing algorithm in cloud computing environment," Int. J. Electr. Comput. Eng., vol. 12, no. 2, p. 2087, Apr. 2022.
[27] F. Mulyadi and K. Akkarajitsakul, "Non-Cooperative and Cooperative Game Approaches for Load Balancing in Distributed Systems," in Proceedings of the 2019 7th International Conference on Computer and Communications Management, Jul. 2019, pp. 252–257.
[28] M. Elveny, A. Winata, B. Siregar, and R. Syah, "A Tutorial: Load Balancers in a Container technology System using Docker Swarms on a Single Board Computer Cluster," Ilkogr. Online - Elem. Educ. Online, vol. 19, no. 4, pp. 744–751, 2020.
[29] S. Sahana, T. Mukherjee, and D. Sarddar, "A Conceptual Framework Towards Implementing a Cloud-Based Dynamic Load Balancer Using a Weighted Round-Robin Algorithm," Int. J. Cloud Appl. Comput., vol. 10, no. 2, pp. 22–35, Apr. 2020.
[30] T. Barbette, E. Wu, D. Kostic, G. Q. Maguire, P. Papadimitratos, and M. Chiesa, "Cheetah: A High-Speed Programmable Load-Balancer Framework With Guaranteed Per-Connection-Consistency," IEEE/ACM Trans. Netw., vol. 30, no. 1, pp. 354–367, Feb. 2022.
[31] O. Khoshaba, V. Grechaninov, A. Lopushanskyi, and K. Zavertailo, "Studying the Dynamic Bottlenecks of a Load Balancer in Distributed Systems," in Lecture Notes in Networks and Systems, 2022, pp. 199–211.

[32] J.-B. Lee, T.-H. Yoo, E.-H. Lee, B.-H. Hwang, S.-W. Ahn, and C.-H. Cho, "High-Performance Software Load Balancer for Cloud-Native Architecture," IEEE Access, vol. 9, pp. 123704–123716, 2021.
[33] S. Atalla, A. Bianco, R. Birke, and L. Giraudo, "NetFPGA-based load balancer for a multi-stage router architecture," in 2014 World Congress on Computer Applications and Information Systems (WCCAIS), Jan. 2014, pp. 1–6.
[34] F. Alharbi and M. Mustafa, "Two-Tier Load Balancer as a Solution to a Huge Number of Servers," J. Eng. Appl. Sci., vol. 9, no. 1, p. 1, 2022.
[35] K. I. Nikishin, "Load Balancer of Data in a Distributed Network via Nginx Proxy Server," Proc. Southwest State Univ., vol. 26, no. 3, pp. 98–111, Feb. 2023.
[36] A. K. Sinha, S. S. K. Singh, S. Sai, and M. Sivagami, "Implementing an Integrated Network Load Balancer for Minimizing Weighted Response," in Lecture Notes on Data Engineering and Communications Technologies, 2023, pp. 651–662.
[37] M. Lopez-Martin, B. Carro, J. I. Arribas, and A. Sanchez-Esguevillas, "Network intrusion detection with a novel hierarchy of distances between embeddings of hash IP addresses," Knowledge-Based Syst., vol. 219, p. 106887, May 2021.
[38] E. Osei Kofi and E. Ahene, "Enhanced network load balancing technique for efficient performance in software defined network," PLoS One, vol. 18, no. 4, p. e0284176, Apr. 2023.
[39] T. Isobe et al., "Areion: Highly-Efficient Permutations and Its Applications to Hash Functions for Short Input," IACR Trans. Cryptogr. Hardw. Embed. Syst., pp. 115–154, Mar. 2023.
[40] C. Rawls and M. A. Salehi, "Load Balancer Tuning: Comparative Analysis of HAProxy Load Balancing Methods," 2022.
[41] K. Takahashi, "A Study on Portable Load Balancer for Container Clusters," University for Advanced Studies (SOKENDAI), 2019.