UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Research of traffic balancing algorithms and their implementation into RSocket

Author:
Dmytro BILUSYAK

Supervisor:
Oleh DOKUKA

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences and Information Technologies
Faculty of Applied Sciences

Lviv 2023

Declaration of Authorship

I, Dmytro BILUSYAK, declare that this thesis titled, "Research of traffic balancing algorithms and their implementation into RSocket" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Research of traffic balancing algorithms and their implementation into RSocket

by Dmytro BILUSYAK

Abstract

Load balancing algorithms play a crucial role in enhancing system scalability, optimizing resource utilization, and improving overall performance. However, the effectiveness of load balancing depends on selecting the appropriate strategy that aligns with the specific requirements of the system. With a wide range of available options, it is essential to consider various parameters and system needs when making the choice. By carefully evaluating these factors, an optimal load balancing strategy can be selected to achieve the desired outcomes.The goal of this research is to compare the performance of various load balancing algorithms in different metrics. The test results are able to highlight specific performance metrics in which each algorithm outperformed others, as well as identify the traffic parameters that affect these metrics. The test environment was carefully designed to ensure consistent and reliable performance data in each test case. The test system consisted of one load balancing node and three servers responsible for handling requests, along with a traffic loader(also a performance testing node). The majority of the data was generated by a traffic loader implemented using the Locust load testing framework. The testing environment was developed using the Python programming language and employed the RSocket transport protocol.GitHub Repository...

Acknowledgements

I would like to express my sincere gratitude to my project advisor, Mr. Oleh Dokuka, for his invaluable guidance, support, and mentorship throughout the course of this research. His expertise, insightful feedback, and constant encouragement have been instrumental in shaping the direction and quality of this paper.

I would also like to extend my heartfelt thanks to my university teachers for their knowledge-sharing and dedicated efforts in imparting valuable education. Their teachings have provided a strong foundation for my academic journey and have contributed significantly to the development of this paper.

I am grateful to my friends for their unwavering support, encouragement, and fruitful discussions during the course of this research. Their insightful inputs and constructive feedback have been instrumental in refining my ideas and expanding my understanding of the subject matter.

Last but not least, I would like to express my deepest appreciation to my family for their unconditional love, understanding, and encouragement throughout my academic pursuits. Their constant belief in my abilities and their unwavering support have been the driving force behind my achievements.

Without the contributions and support of these individuals, this paper would not have been possible. I am truly fortunate to have had such incredible guidance, mentorship, and support from my project advisor, university teachers, friends, and family...

Contents

List of Figures

List of Abbreviations

LB	**L**oad **B**alancer
LC	**L**east **C**onnections load balancing strategy
DWRR	**D**ynamically **W**eighted **R**ound **R**obin load balancing strategy
WRR	**W**eighted **R**ound **R**obin load balancing strategy
RR	**R**ound **R**obin load balancing strategy
R	**R**andom load balancing strategy

Chapter 1

Introduction

The increasing need for high-performance and scalable network applications has led to a demand for efficient traffic and resource management. Satisfying the performance requirements simply can be achieved by enhancing capacities or optimizing the software, among other factors. However, software optimization has its limits and the high cost of scaling the system may not always justify the desired results. The same performance with a lower cost could be established by introducing load balancers to the system.

Load balancers allow for horizontal scaling of the system. This scalability enables the system to handle increasing loads and accommodate growing user demands without sacrificing performance or experiencing bottlenecks.Additionally, load balancers contribute to cost-effectiveness by optimizing resource utilization, reducing the need for expensive hardware upgrades, and maximizing the efficiency of existing infrastructure.

Load balancing techniques have undergone significant evolution in response to the increasing demands of modern network applications. Traditional approaches, such as round-robin and random load balancing, have paved the way for more sophisticated algorithms and adaptive strategies that consider factors like server capacities, network conditions, and user demands. Moreover, novel approaches, including machine learning-based load balancing and application-aware routing, have emerged to address the complexities of modern network environments and deliver enhanced performance, scalability, and resource utilization.

Distributing traffic across servers, load balancing algorithms aim to prevent overloading of any single server, minimizing response times and increasing the availability of network services. However, the effectiveness of these enhancements depends on a wide range of system parameters. A wide variety of application architecture surely makes finding the universal approach difficult. Fortunately, many different load balancing algorithms exist. Choosing the appropriate algorithm is sensitive to various factors such as the type of application, available resources, network topology, and performance requirements. To determine the most suitable load balancing algorithm for a given scenario, it is necessary to conduct performance testing and analysis.

Chapter 2

Background Information

2.1 Load balancing

Load balancing is an approach to manage the processing of incoming requests among the existing resources with a purpose of acquiring better performance-based characteristics of the system. The basic concept is as follows: the unit receiving the initial task (load balancer) decides with a forehand set algorithm, to which unit the job will be passed and then handled.

The logic behind selecting the node responsible for handling each request may be based either on a real-time state of the system (load, capacity and number of connections of the individual nodes), or a static algorithm(i.e. random selection of pre-defined order). The architecture of load balancers also plays its role. One approach is to design the system with hierarchical load balancing, which involves multiple layers of balancing nodes. Each layer balances the traffic among nodes, which may not necessarily process the requests. This may be particularly useful for large-scale applications where there are a large number of servers and the load is not evenly distributed. Contrary to this is the flat or non-hierarchical load balancing, where all nodes are considered equal and traffic is distributed evenly among them.

Performance testing of the algorithms is usually conducted on different setups and configurations and then the measurable data are compared based on parameters such as latency, throughput and low packet loss. In subsequent analysis, cogent conclusions can be drawn, including recommended approaches (such as a specific load balancing algorithm) for desirable performance requirements in a given environment.

2.2 RSocket

A relatively new binary protocol for use on different byte stream transports. It is designed to provide reactive and scalable communication between applications running on different systems. It is worth a mention that this paper focuses more on the load balancing algorithms, rather than the RSocket protocol itself. However, it is also recommended by the author not to assume applicability of any results of this paper to other transport protocols. Meanwhile, some protocols may show similar performance, but the difference of their under-the-hood implementations is without question, significant enough not to expect similar approaches to result in the same.

Chapter 3

Load balancing strategies

This section provides a concise overview of the theoretical details of each load balancing algorithm tested in this research. While the descriptions provided offer a necessary understanding of the concepts behind each algorithm, it is important to acknowledge that the suitability and performance of these algorithms can vary based on specific system requirements. Factors such as server capacities, traffic load, performance goals, and desired levels of fairness or resource utilization may significantly impact their effectiveness. Therefore, the actual properties and potential use cases of these algorithms may differ from the descriptions provided, underscoring the need for this research to assess their performance in real-world scenarios.

Load balancing algorithms are commonly divided into two main categories: static and dynamic. Static algorithms offer a straightforward implementation approach and maintain consistent behavior regardless of any changes in the system or traffic. On the other hand, dynamic algorithms provide a more flexible method to balance traffic, performing differently, based on various parameters, however being relatively complex to implement.

3.1 Round Robin (RR):

- **Concept**: RR operates by evenly distributing incoming requests across available servers in a sequential manner, looping back to the first server once all servers have received a request. Algorithm is static.
- **Pros:**: Simple implementation, fair distribution of requests among servers, suitable for systems with similar server capacities.
- **Cons**: Does not consider server load or capacity, may result in imbalanced resource utilization, not suitable for systems with varying server capacities.
- **Perfect scenario:**: A system with multiple servers of equal capacity, moderate traffic load, and where fairness in request distribution is a priority.

3.2 Weighted Round Robin (WRR):

- **Concept**: WRR assigns a weight to each server based on its capacity or performance, and requests are distributed in proportion to these weights. Servers with higher weights receive a larger share of requests. Algorithm is static.
- **Pros:**: Allows for better utilization of server capacities, enables prioritization of higher-performing servers, suitable for systems with varying server capacities.

- **Cons**: Static weight assignment may require fine-tuning, does not dynamically adjust weights based on real-time server conditions.
- **Perfect scenario:**: A system with servers of different capacities or performance levels, high traffic load, and the need to allocate requests proportionally to server capabilities.

3.3 Least Connection (LC):

- **Concept**:LC directs new requests to the server with the fewest active connections, aiming to distribute the load evenly across servers based on connection counts. Algorithm is dynamic.
- **Pros:**: Balances load based on active connections, suitable for systems with varying connection loads, avoids overwhelming a single server.
- **Cons**: Ignores server capacities or performance, may result in imbalanced resource utilization based on varying connection durations or workloads.
- **Perfect scenario:**: A system with varying connection durations, high traffic load, and the requirement to evenly distribute load based on active connections.

3.4 Dynamic Weighted Round Robin (DWRR):

- **Concept**:DWRR dynamically adjusts weights assigned to servers based on real-time server performance metrics, such as CPU usage or response times. It aims to optimize load distribution by considering server capacities and performance. Algorithm is dynamic.
- **Pros:**: Dynamically adapts to changing server conditions, improves resource utilization, suitable for systems with varying server capacities and performance.
- **Cons**: Requires monitoring and adjustment of performance metrics, additional complexity compared to static algorithms.
- **Perfect scenario:**: A system with servers of different capacities and performance levels, fluctuating traffic loads, and the need for load balancing that adapts to real-time server conditions.

3.5 Random (R):

- **Concept**: R randomly selects a server from the available pool to handle each incoming request. Algorithm is static.
- **Pros:**: Simple implementation, distributes requests randomly among servers.
- **Cons**: Does not consider server load or capacity, may result in imbalanced resource utilization, not suitable for systems with specific load balancing requirements.
- **Perfect scenario:**: A system where equal distribution of requests among servers is not a priority, and load balancing requirements are not critical.

Chapter 4

Test Design

4.1 Approach

Specifics of performance testing different load balancing algorithms required laying out all the performance parameters, as well as system configurations that effect those parameters.

The Performance metrics were selected to be:

- Response Time
- Failure Percentage
- Traffic rate
- Servers' CPU usage:

System's performance and behavior depends on:

- Specific load balancing algorithm.
- System capacity
  - Variety in hardware specifications of each server
  - Amount of nodes/servers
- System Load :
  - Spawn rate of requests
  - Maximum number of requests/s
- Request parameters
  - Size of package(request being transferred) in bytes
  - Volume of work in a single request (Complexity of operation)

The capacity and workload of a system are interdependent parameters. When designing or scaling the system, it is imperative to consider the necessity of additional capacity. Without sufficient capacity at your disposal, you will be unable to process higher workloads. Likewise, loading the system significantly lower than its processing capability is unnecessary. This conclusion has led to the approach of designing performance tests that neglect various combinations of system capacities and loads, focusing solely on request parameters and types of load balancing.

4.2 Test parameters

The type of work selected for the test was data serialization and deserialization (in json format). The final test set for each load balancing algorithm consisted of combinations of different package sizes and volumes of work per request (Figure 1.). To to simulate a load, similar to real-world scenarios and capture the diverse conditions, in the last test, the workload and package size was randomly selected for each new request from a pool of all possible combinations.

	small request (10kb)	medium request (100Kb)	large request (1MB)
small package(100b)	sp+sr	sp+mr	sp+lr
medium package(10Kb)	mp+sr	mp+mr	mp+lr
large package(1Mb)	lp+sr	lp+mr	lp+lr

Figure 1. Different package sizes and work volumes used in performance tests

Each test run was designed to last 20 minutes, with 3 replications(repetition), to mitigate the influence of anomalous data.

4.3 System configurations

The testing was performed on a system composed of three server nodes, a load balancing node, and a traffic loader node, each running on separate machines, using the Google Cloud platform. Two of the server nodes were configured with identical hardware specifications, specifically 0.25-2 vCPU (1 shared core) and 1 GB of memory. The third server node had 2 vCPU and 8 GB of memory. The traffic loader node was responsible for spawning up to 50 concurrent user instances, which sent requests to the load balancer. It was developed using Locust load testing framework. .

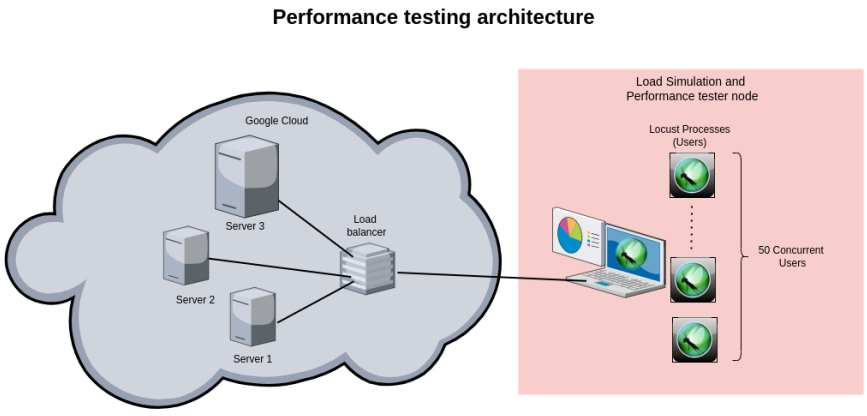


FIGURE 4.1: Performance testing architecture

Chapter 5

Test Results

The following section presents an analysis based on a comprehensive set of metrics, including failure rates, response time, and proper usage of available capacity data, in all test cases. Each metric not only provides specific performance information but also offers valuable insights into other key parameters within each test scenario. By carefully examining the patterns and trends in these parameters, we can uncover underlying factors and relationships that significantly influence the behavior of the system. This approach allows us to evaluate system efficiency, and uncover opportunities for improvement. In the subsequent sections, we will delve into the detailed analysis of each metric, in all five proposed load balancing algorithms, highlighting possible interdependencies, patterns and potential ways to change implemented load balancing algorithms to further increase system performance. .

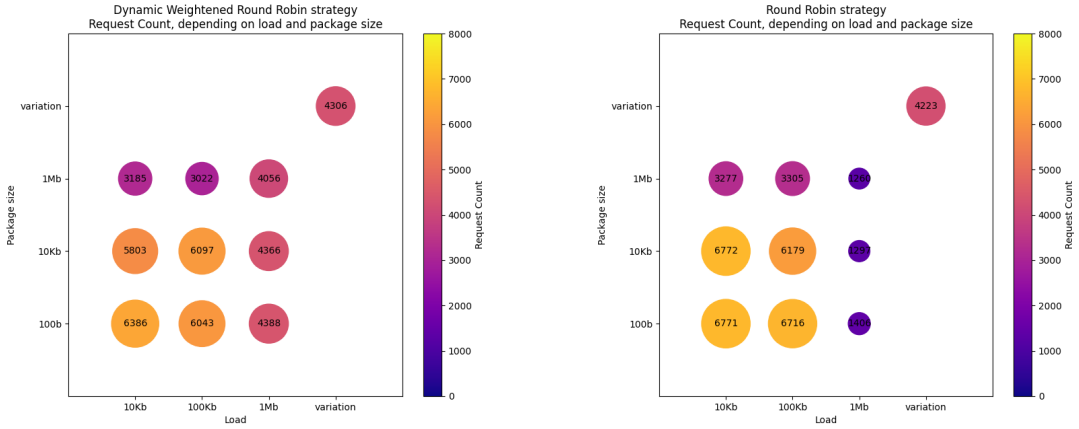
By evaluating performance of each load balancing algorithm in various metrics, alalysis aims to define strong and weak sides of each load balancing algorithm in said metrics. This information guides the selection of the most suitable load balancing approach for specific performance requirements, depending on parameters of data, such as package size and complexity of opperation per request. .

Consequent sections about each performance metric will not include graphs for all load balancing algorithms. However, all the data and visualizations generated during the research can be found in the appendix. The appendix provides access to the complete dataset and accompanying visual representations.

5.1 Request Count

The following section focuses on the analysis of the number of processed requests within a specified time period, a crucial performance metric that provides insights into the system's efficiency and throughput. The number of processed requests serves as a fundamental indicator of the system's ability to handle incoming workload and meet user demands. By examining patterns in the number of processed requests, depending on a requests parameters and load balancing algorithm, we can gain a deeper understanding of opportunities of different load balancing strategies .

In this parameter, performance of DWRR and LC strategies was almost identical, except than LC outperformed DWRR in variation test by 12%. R and RR were also quite similar in this metric, differing only slightly in 10kb load test, where RR processed 34% more requests, which were transported as 100b packages, but R processed 25% more requests in 10Kb package test. .



(a) Request Count, using DWRR strategy

(b) Request Count, using RR strategy

FIGURE 5.1: Side to side comparison of the Request Count data

When considering the comparison between dynamic and static algorithms (excluding WRR) in terms of the number of processed requests, static algorithms demonstrate superior performance in small and medium loads. This disparity arises from the increased complexity of dynamic algorithms compared to static ones, resulting in a slightly longer average duration for each request. However, in large load tests, dynamic strategies significantly outperform them. .

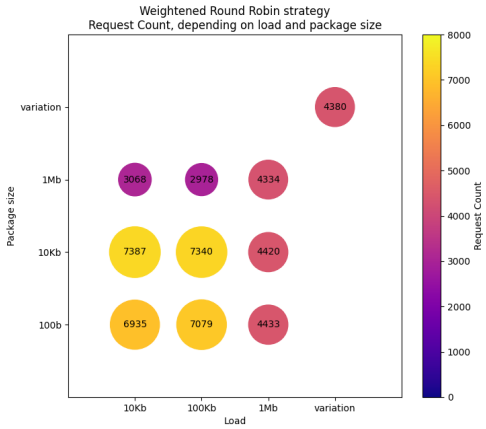


FIGURE 5.2: Request count, depending on load and package size using WRR strategy.

WRR exhibits comparable performance to DWRR in large load and/or package size tests, as well as in variative tests. However, it surpasses dynamic algorithms by processing 16-21% more requests in scenarios involving small and medium loads and package sizes. WRR excels in handling small and medium-sized requests more effectively than other static algorithms, while also demonstrating comparable performance to dynamic algorithms in processing large requests. Consequently, WRR emerges as the most effective static algorithm overall in terms of the number of requests processed per unit of time..

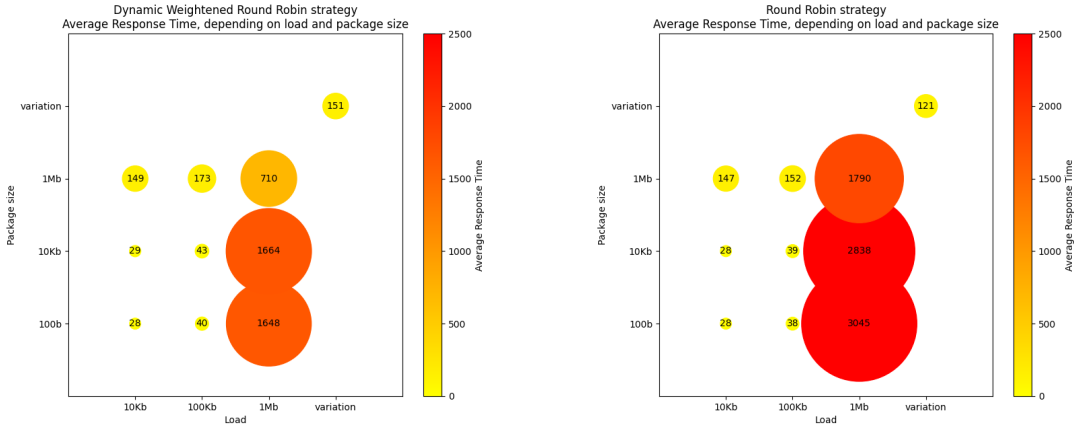
The observation that both dynamic load balancing algorithms presented in this research process fewer requests than WRR suggests that they may not fully exploit the capacity of the servers. Consequently, there is potential for further refinement and improvement of dynamic algorithms, which could lead to different performance outcomes when compared to WRR. Furthermore, it is worth noting that the weights assigned to server nodes in the WRR approach were roughly estimated, implying that WRR has the potential to achieve even better performance with more accurate weight assignments.

5.2 Conclusion of Request Count

In systems prioritizing throughput, but where the requests are not excessively time-intensive (up to a complexity of processing 100KB of data) sent in packages of up to 10KB, the system could opt for either the R or RR algorithm. By conducting performance testing and appropriately configuring a WRR load balancer, the system can achieve slightly better performance in this metric. However, in the event of a potential high workload, it becomes crucial to implement a dynamic load balancer or WRR that can effectively utilize system capacities. .

5.3 Average Response Time

In this metric, RR and R performed similarly, except RR had up to 30% slower response time with small and medium load and package size tests. Conversely, R algorithm performed faster by up to 30% in the large load category. Similarly, among the dynamic algorithms, the statistical differences were relatively similar, with variances of up to 20%, except for the variation test, where DWRR displayed a 50% shorter response time compared to LC. This is directly linked to difference between approaches to utilize system capacities in these dynamic algorithms, on which we will focus later. .



(a) Average Response Time, using DWRR strategy

(b) Average Response Time, using RR strategy

FIGURE 5.3: Side to side comparison of the Request Count data

In terms of latency, the static algorithms demonstrate similar performance to the dynamic ones, except that the dynamic algorithms have a huge advantage when it comes to processing a large number of requests. Interestingly, both R and RR algorithms yield a slightly 20% smaller average response time in the variation test then dynamic strategies. This difference in performance can be attributed to the fact that dynamic algorithms are more complex, resulting in additional time required for determining the appropriate node to handle each response. As a consequence, dynamic algorithms tend to have slightly longer average response times compared to static algorithms. .

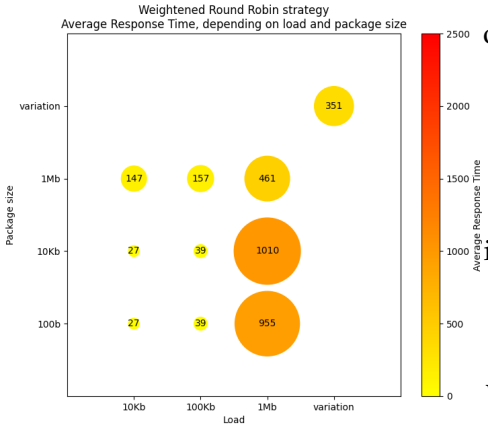


FIGURE 5.4: Average response time using WRR algorithm

In small and medium processing load tests, there is minimal difference in performance between WRR and DWRR. However, in large processing tests, WRR exhibits a significantly better average response time compared to DWRR, outperforming it by 30-70%. This contrast can be explained by the difference between WRR and DWRR algorithms, with the former employing a more complex node selection algorithm than the latter. However, the average latency in WRR for the variation test is more than twice as long as that of DWRR. Despite also working similarly to RR, but effectively utilizing server capacities, WRR performs worse than R or RR in the variation test. This observation suggests that there may be specific rare scenarios that need to be further investigated through prolonged testing. Although the median response time in WRR is 17ms, the 90th and 95th percentiles are 433ms and 3726ms, respectively. These higher percentiles indicate the existence of rare scenarios that requires further investigation upon running longer tests.

5.4 Conclusion on latency data

- **Static Solution**: In terms of shortening average response time in small and medium complexities of operation per requests, as well as variative traffic, both R and RR result perform similarly. Alternatively, WRR is much more capable of managing traffic with larger requests.
- **Dynamic Solution**: As was stated before, in variation test, DWRR showed 50% shorter average response time, then LC. Howerver, up to 90th percentile, data is very similar. Therefore there exist scenarios, where LC, not taking into consideration capacity of the individual server in any way(by constant as WRR, or dynamically by RT as DWRR), overloads server/s with less capacities with many large requests. In which case, variation of capabilities of nodes in a system suggests DWRR being better option than LC in scenario with large variation in size of package and complexity of operation per request.
- **Dynamic and Static Comparison**: Despite the fact, that WRR is great with all types of package sizes and operation complexity, dynamic algorithms give decent responce time on average in all categories, which make them be percieved as more flexible.

5.5 Resource utilization

This section aims to compare how different load balancing algorithms use available server capacity. Desirable parameters are equivalency in usage amoung nodes and smoothness. .

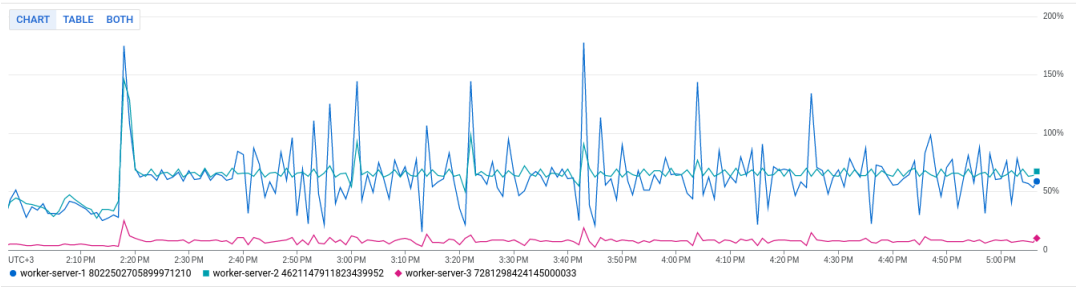


FIGURE 5.5: CPU utilization chart during processing large amount of data per request, using RR algorithm

CPU usage in RR and R test proved these algorithms to be uncapable of utilizating the available capacity of the system properly, as they aren't designed for it. .

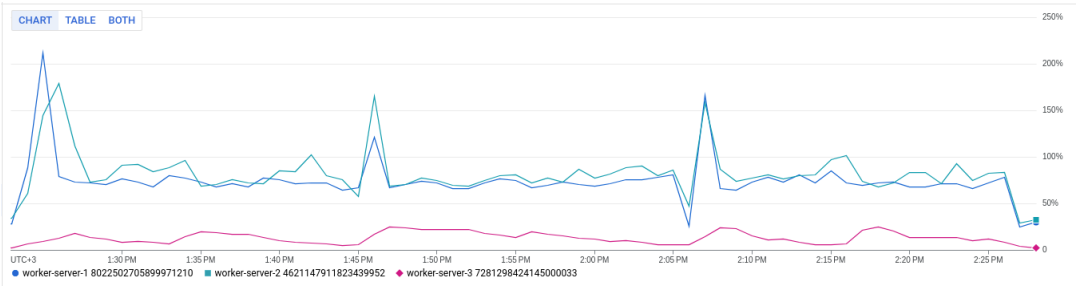


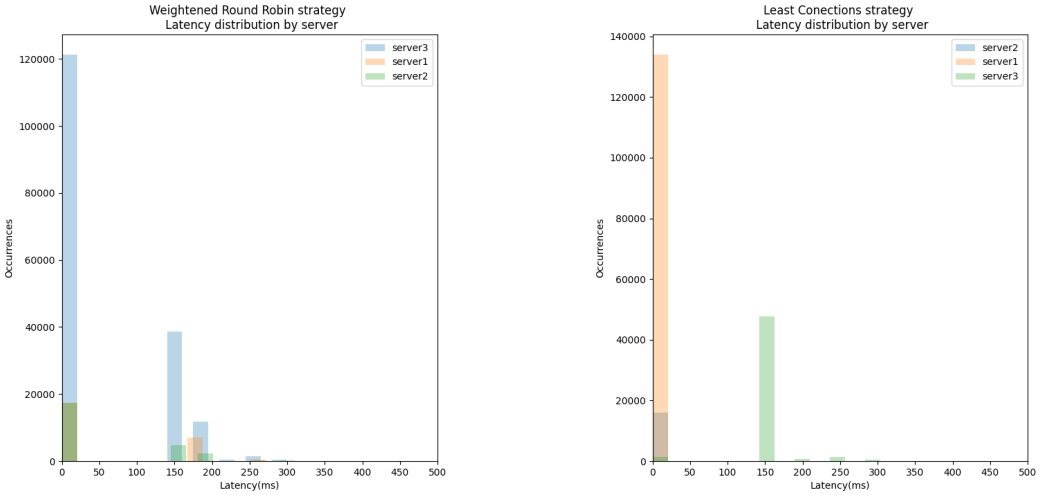
FIGURE 5.6: CPU utilization chart during variation test using LC algorithm

FIGURE 5.7: CPU utilization chart during variation test using WRR algorithm

WRR, LC, and DWRR demonstrated the ability to effectively utilize the diverse capacities of the system(Figure 5.6 and 5.7). However, both WRR and DWRR consistently distributed the workload evenly across all nodes, regardless of the package size or operational complexity. In contrast, LC did not load the node with higher capacity when the traffic could be handled by the first two servers. In other words, traffic is infrequent enough and consists of relatively small requests,

LC could completely neglect nodes with more capacities.This occurred due to the absence of a priority-based selection mechanism for assigning requests to nodes.

.



(a) Latency distribution by server in WRR algorithm

(b) Latency distribution by server in LC algorithm

FIGURE 5.8: Side to side comparison of latency distribution by server in WRR and LC alorithms(aggregated across all tests)

Figure 5.8 provides further evidence supporting the previous statement regarding the difference in approach to resource utilization between the LC and WRR strategies. From LC algorithm perspective, 6 out of 9 tests involved traffic that did not require the utilization of the third server. As a result, the majority of requests in all of these tests were handled by the first two servers in the LC strategy. Conversely, in both WRR and DWRR strategies, the node with the highest capacity handled the majority of requests across all tests. .

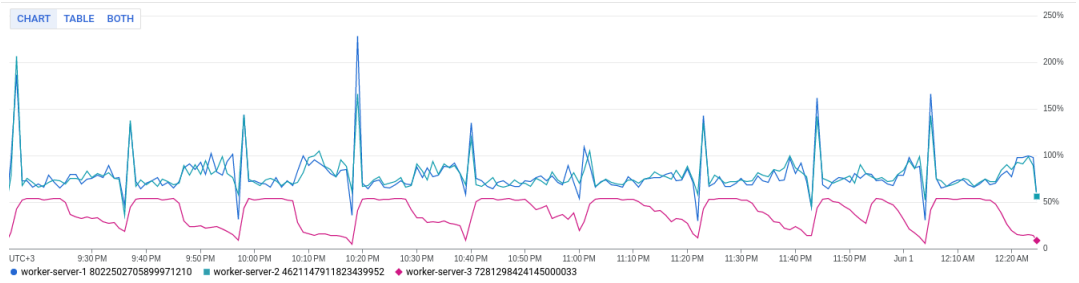


FIGURE 5.9: CPU utilization chart during processing requests requiring processing of larger amount of data using DWRR algorithm

A peculiar observation(Figure 10) was the unexpected decrease in CPU usage on the third server in tests involving a high complexity of operations per request for DWRR, LC, and WRR. In the case of DWRR, this led to an increase in CPU usage on the first two servers during several replications, while LC did not exhibit a similar pattern. Further investigation is needed, requiring additional prolonged

tests. It is possible that this phenomenon is connected to the previously observed higher failure rates, comparing to static algorithms in tests involving large requests. .

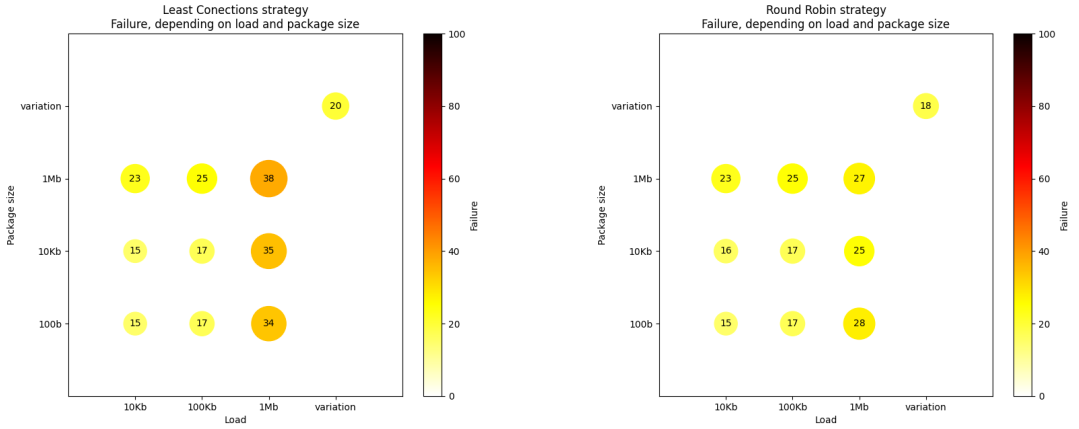
5.6 Conclusion on resource utilization section

- **Static Solution**: In resource utilization, R and RR algorithms perform terribly, which indicates that they are not designed for systems with variation in capacities of nodes. On the other hand, WRR demonstrates resource utilization comparable to dynamic algorithms. However, it is crucial to carefully configure static weights for nodes, which necessitates thorough research into the system's performance. It is important to note that in scenarios involving dynamic resources, such as the ability to add/remove servers from the load balancer's pool or server downtime, any static load balancing algorithm will result in significantly higher failure rates, response times, and overall degradation of the user experience.
- **Dynamic Solution**: Both DWRR and LC effectively utilize system resources despite variations in node capacities. However, they demonstrate contrasting approaches for distributing traffic among servers. LC's approach may be considered a disadvantage compared to DWRR. Nevertheless, by establishing the correct static order for node selection and conducting prolonged tests, it could help identify unnecessary nodes for a given traffic loadthrough further analysis of individual node capacities.
- **Dynamic and Static Comparison**: In systems where nodes have equal capacities and there is moderate traffic with a high demand for small response time, static load balancing strategies such as RR or R can provide significant benefits. On the other hand, in environments where there is a large variation in node capacities, dynamic load balancing algorithms and properly configured WRR are better choices.

5.7 Failure rate

The section focusing on server failure rates provides an analysis of the occurrence of server failures in the context of the studied system. This metric is crucial to determing system reliability and performance, as server failures can lead to service disruptions, data loss, and compromised user experience. Understanding the patterns and underlying causes of server failures is essential for devising effective strategies load balancing. Examinination of server failure rates, depending on a load balancing algorithm, aims to shed light on potential ways to enhance the reliability and availability of existing load balancing algorithms.

Dynamic strategies exhibit remarkably similar failure rates across all test cases. Round Robin and Random algorithms also demonstrate a significant degree of similarity in terms of failure rates. Therefore, based solely on this metric, there is limited differentiation between the RR and R algorithms. .



(a) Failure rate, using LC strategy

(b) Failure rate, using RR strategy

FIGURE 5.10: Side to side comparison of the failure rate data

The results obtained from the analysis indicate that all five algorithms exhibited nearly identical failure rates when 1Mb of data per request. However, there was a notable increase in failure rates for the dynamic algorithms when processing 1Mb of data. Despite this, the dynamic algorithms demonstrated superior performance in processing larger volumes of such requests within a given time period, exhibiting lower average response times. Further details regarding these findings will be discussed in subsequent sections of the paper. .

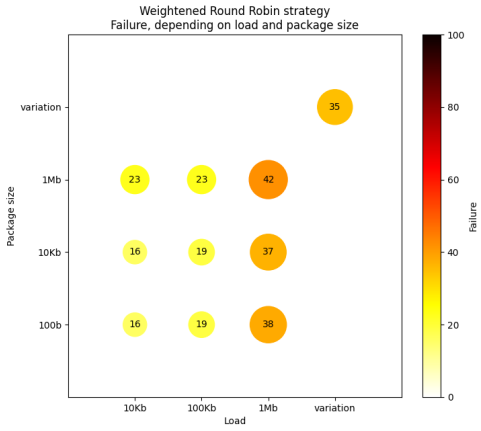


FIGURE 5.11: Failure rate in WRR load balancer

WRR had comparable performance to dynamic algorithms in scenarios involving small to medium package request complexities. However, it showed slightly poorer performance in requests that processed 1Mb of data compared to dynamic algorithms. Additionally, WRR had twice as many failures as DWRR in the variation test. Considering that the WRR relationship was derived from analyzing the statistics of DWRR (based on the number of requests processed by each server), this case requires further research and running more prolonged tests to better understand the cause of it.

5.8 Conclusion of Failure rate

- **Static Solution**: The failure rate between RR and R algorithms does not exhibit a significant difference. However, WRR can yield a failure rate that is more than twice as high in certain scenarios. It is important to note that WRR is capable of processing a larger volume of traffic, including operations with greater complexity. However, when considering failure rate alone, WRR falls behind R and RR algorithms.
- **Dynamic Solution**: Dynamic algorithms resulted in almost identical performance in failure rate metric.
- **Dynamic and Static Comparison**: While dynamic algorithms excel in processing large request volumes, they may experience increased failure rates in scenarios with more complex operations, however providing better avarage response time. Conversely, WRR demonstrates comparable performance in smaller-scale scenarios but exhibits limitations when processing 1Mb data requests. Therefore, for systems with a high demand for a low failure rate, implementing R and RR algorithms would be the appropriate solution. However, as mentioned earlier, dynamic algorithms and WRR have potential for improvement. With further enhancements, they could demonstrate reduced failure rates, as well as efficient utilization of system capacities, and the ability to process larger requests effectively.

Chapter 6

Research conclusions

6.1 Balancing Algorithm Comparison

The Round Robin (RR) and Random (R) algorithms performed almost identically in every metric, which was expected. Despite their different node selection methods for individual requests, over time, there is an equal average number of requests handled by each node in both RR and R. These static algorithms are best suited for environments where node capacities are equally or quite similarly distributed, with a moderate level of traffic consisting of relatively small package sizes and low complexity per request. In such systems, they ensure better system performance in terms of response time, failure rate, and the number of responses per time unit compared to other considered algorithms. However, the improper utilization of system capacity leads to poorer performance compared to dynamic algorithms when the traffic intensity increases. Additionally, the writer suggests that this paper may not have showcased every flaw of R and RR algorithms due to the relatively short test duration, which does not fully represent real-world scenarios. This also applies to dynamic algorithms, but due to inability to utilize system capacities properly, R and RR have much higher risk of potential pitfalls in this matter.

LC and DWRR are quite similar in the majority of performance metrics. However, LC, being a faster algorithm than DWRR, results in slightly faster performance, particularly in scenarios with low traffic. This makes LC capable of performing well even with larger traffic. Therefore, LC is a desirable algorithm for systems with unequal capacities and varying traffic, including variations in package size and complexity of operations, as long as the majority of the requests are relatively simple. On the contrary, DWRR specializes in handling heavier loads, despite its slower selection algorithm compared to LC. Nonetheless, DWRR still provides good performance and can effectively handle lighter traffic.

WRR exhibited mixed performance compared to other algorithms. It showed superiority in certain metrics, such as average response time, particularly in traffic with large complexity of requests, despite a slightly higher failure rate in this category. However, unexpectedly, it performed slower in variation tests compared to all other algorithms. WRR demonstrates effectiveness in handling consistently large workloads, properly utilizing resources, and processing more requests per time unit, particularly with consistently small to medium package size and request complexity. Nevertheless, its unpredictable behavior in specific cases suggests that it is difficult to make definitive recommendations regarding the system configurations where it would be most suitable.

Research confirmed expected differences in performance among different static and dynamic load balancing algorithms. The evaluation provided empirical evidence that each algorithm has its own strengths and weaknesses, which align with the theoretical expectations. These findings reinforce the importance of carefully selecting the appropriate load balancing algorithm based on specific traffic parameters and performance requirements. Furthermore, the research highlights the need for continuous evaluation and comparison of load balancing techniques to optimize system performance.

6.2 Other findings

- **WRR performance**: As mentioned in previous analysis sections, WRR exhibited a higher failure rate compared to dynamic algorithms and slower response time in the variation test, particularly when compared to DWRR (on which WRR's weights were based). However, it showed superior performance in other tests compared to DWRR. While potential explanations for these observations were discussed in the corresponding sections, further investigation is needed. Retesting the WRR strategy with more precisely chosen weights for nodes, preferably in a similar environment but with longer test cases, would be beneficial in gaining deeper insights into its behavior and performance.

- **Possible WRR, DWRR, LC decrease of consistent balance in utilizing capacity**: Possible scenarios for WRR, DWRR, and LC resulting in a decrease in consistent capacity utilization were observed during the tests. CPU usage graphs indicated a point in time where the overall usage of all nodes dropped (except for DWRR). Upon further investigation, no significant drops in requests per second or an increase in failure rate were observed during that period. This suggests that there was no apparent impact on the overall performance of the system. However, this remains an open case that requires further investigation to understand the underlying reasons for the observed behavior and its potential implications.

- **LC alternative usage**: LC takes a different approach to capacity utilization compared to DWRR. Instead of prioritizing the more capable nodes, LC focuses on utilizing nodes that are sufficient for achieving competent performance. This unique characteristic of LC enables it to be used as an analytical algorithm that aids in identifying unnecessary nodes in the system for handling specific traffic. With the suggested enhancements, the analysis performed by LC can become even more beneficial in optimizing resource allocation and improving overall system efficiency.

Appendix A

Appendix

FIGURE A.1: CPU utilization chart during small request complexity test using LC algorithm

FIGURE A.2: CPU utilization chart during medium request complexity test using LC algorithm

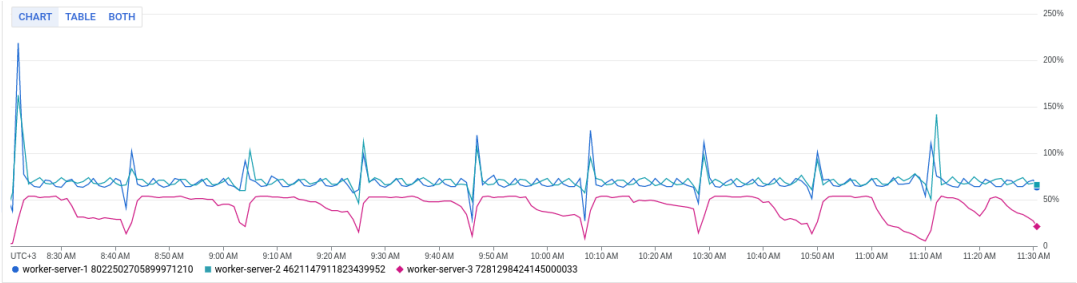


FIGURE A.3: CPU utilization chart during large request complexity test using LC algorithm

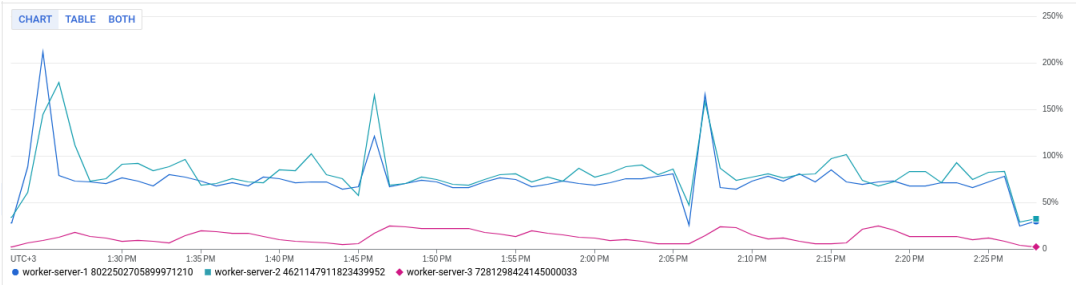


FIGURE A.4: CPU utilization chart during variation test using LC algorithm

FIGURE A.5: CPU utilization chart during small request complexity test using wrr algorithm

FIGURE A.6: CPU utilization chart during medium request complexity test using wrr algorithm

FIGURE A.7: CPU utilization chart during large request complexity test using wrr algorithm

FIGURE A.8: CPU utilization chart during variation test using wrr algorithm

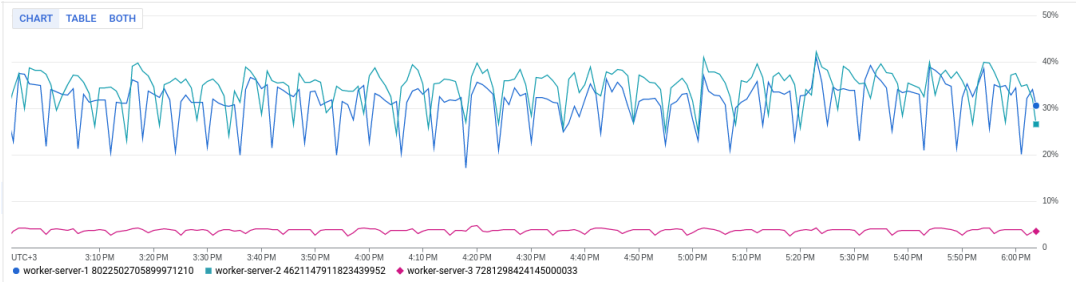


FIGURE A.9: CPU utilization chart during small request complexity test using dwrr algorithm

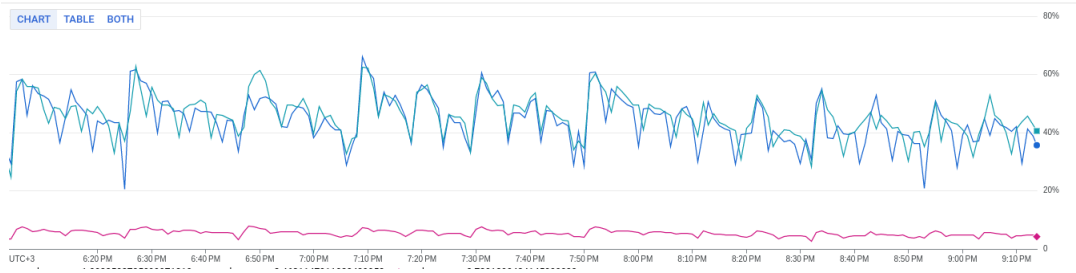


FIGURE A.10: CPU utilization chart during medium request complexity test using dwrr algorithm

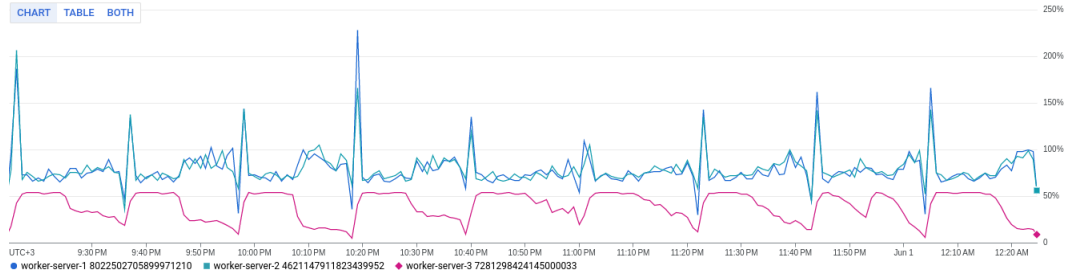


FIGURE A.11: CPU utilization chart during large request complexity test using dwrr algorithm

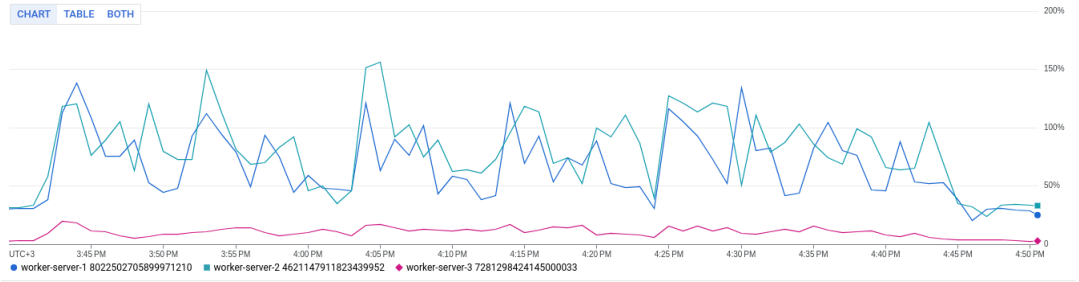


FIGURE A.12: CPU utilization chart during variation test using dwrr algorithm

FIGURE A.13: CPU utilization chart during small request complexity test using rr algorithm

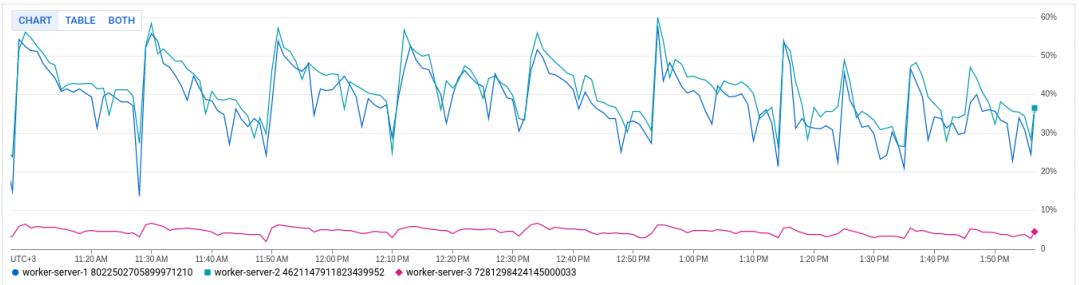


FIGURE A.14: CPU utilization chart during medium request complexity test using rr algorithm

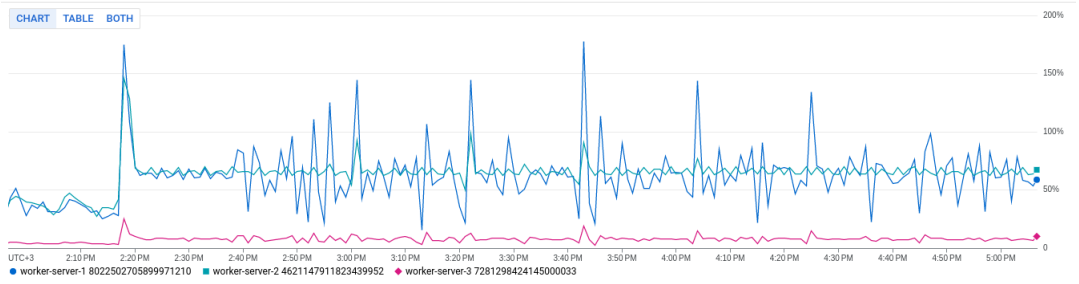


FIGURE A.15: CPU utilization chart during large request complexity test using rr algorithm

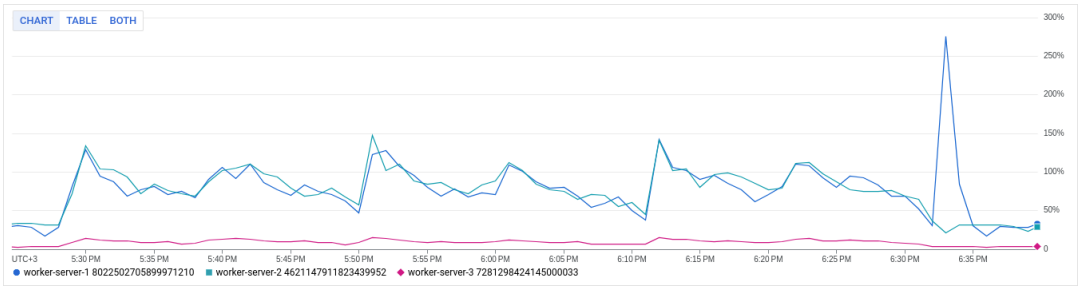


FIGURE A.16: CPU utilization chart during variation test using rr algorithm

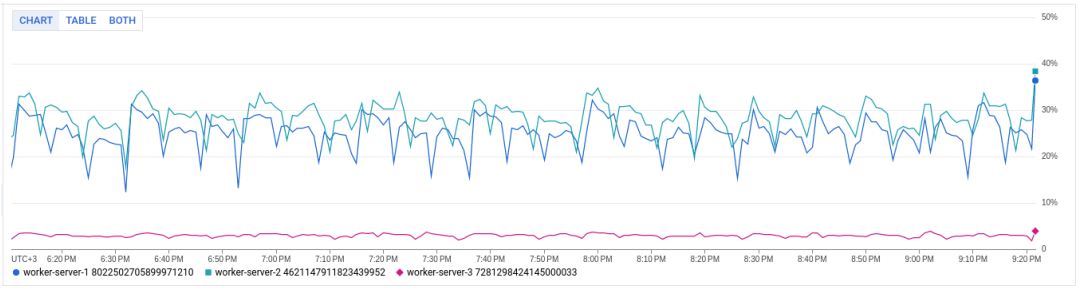


FIGURE A.17: CPU utilization chart during small request complexity test using r algorithm

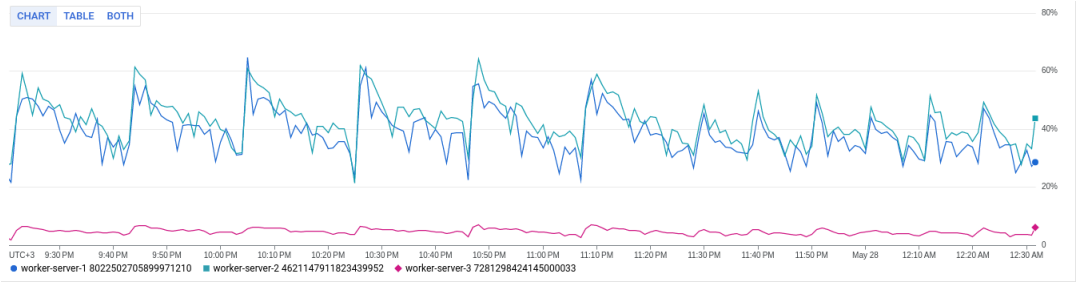


FIGURE A.18: CPU utilization chart during medium request complexity test using r algorithm

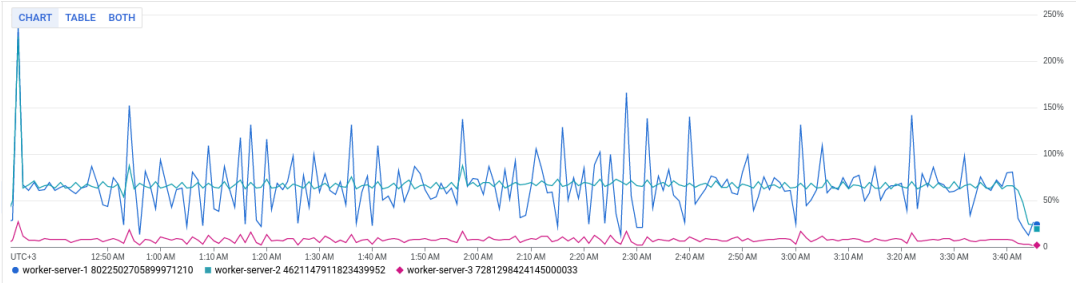


FIGURE A.19: CPU utilization chart during large request complexity test using r algorithm

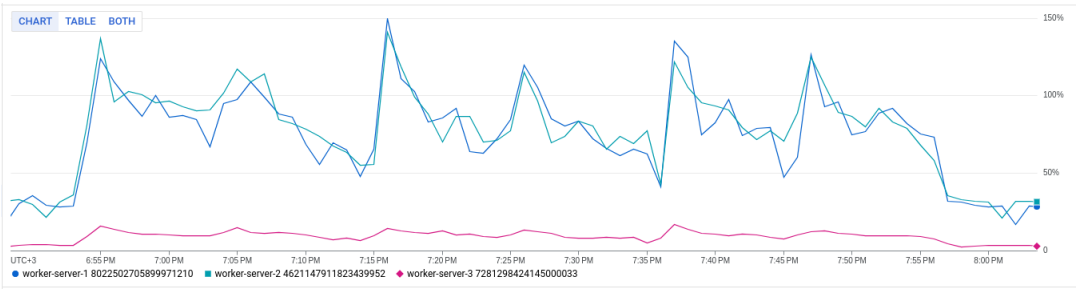


FIGURE A.20: CPU utilization chart during variation test using r algorithm

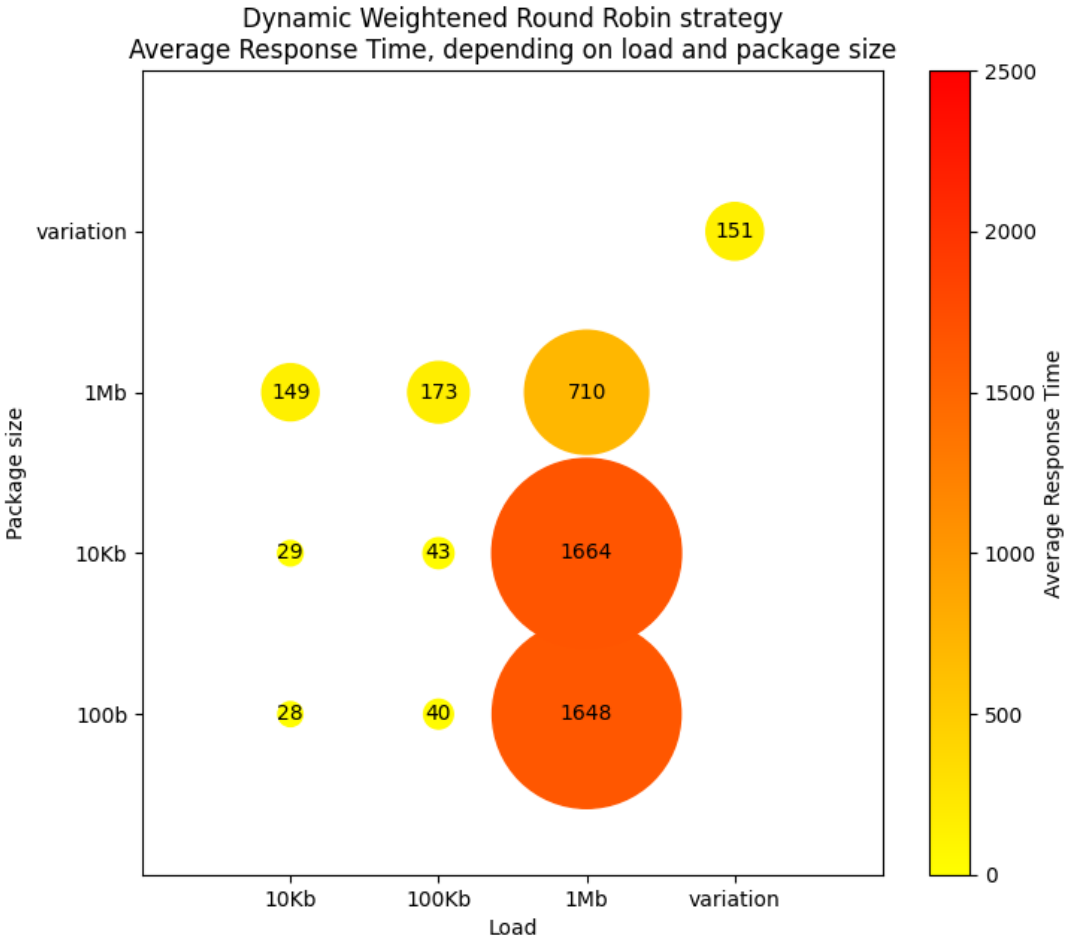


FIGURE A.21: Average responce time, using dwrr algorithm, depending on traffic parameters

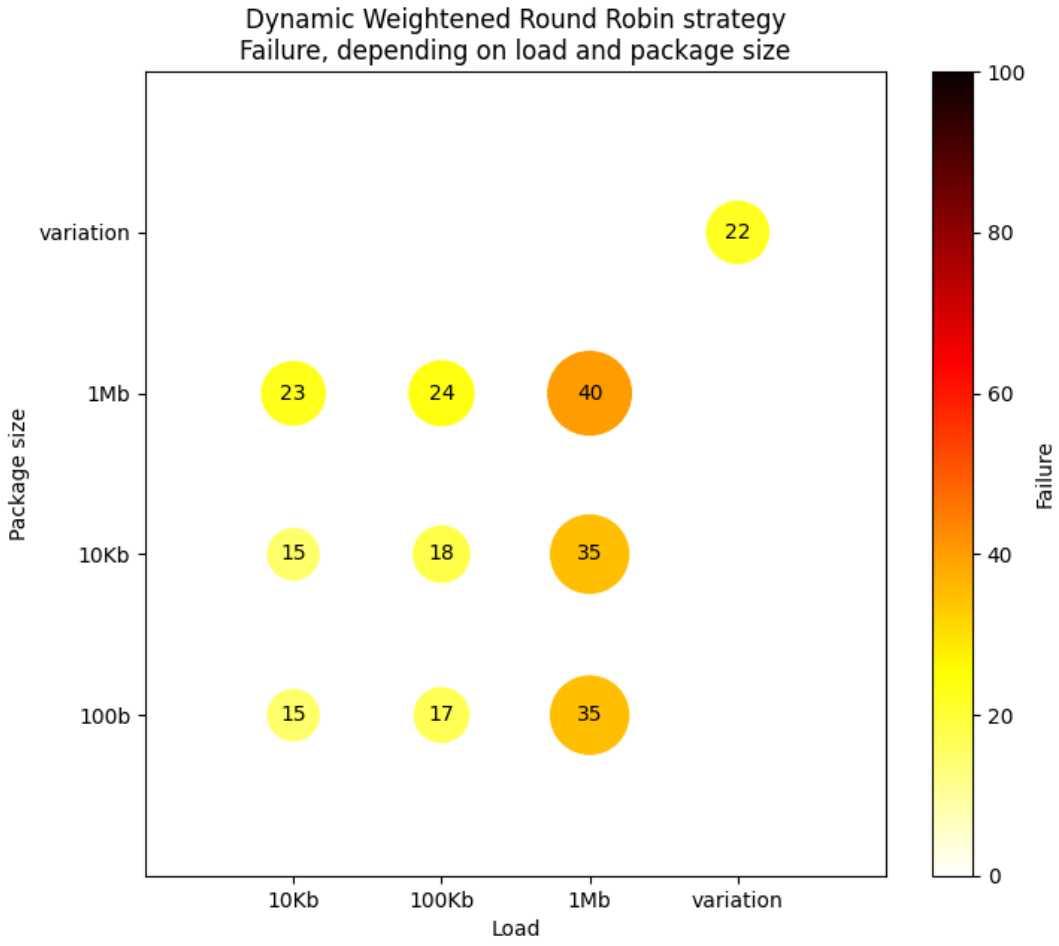


FIGURE A.22: Failure rate, using dwrr algorithm, depending on traffic parameters

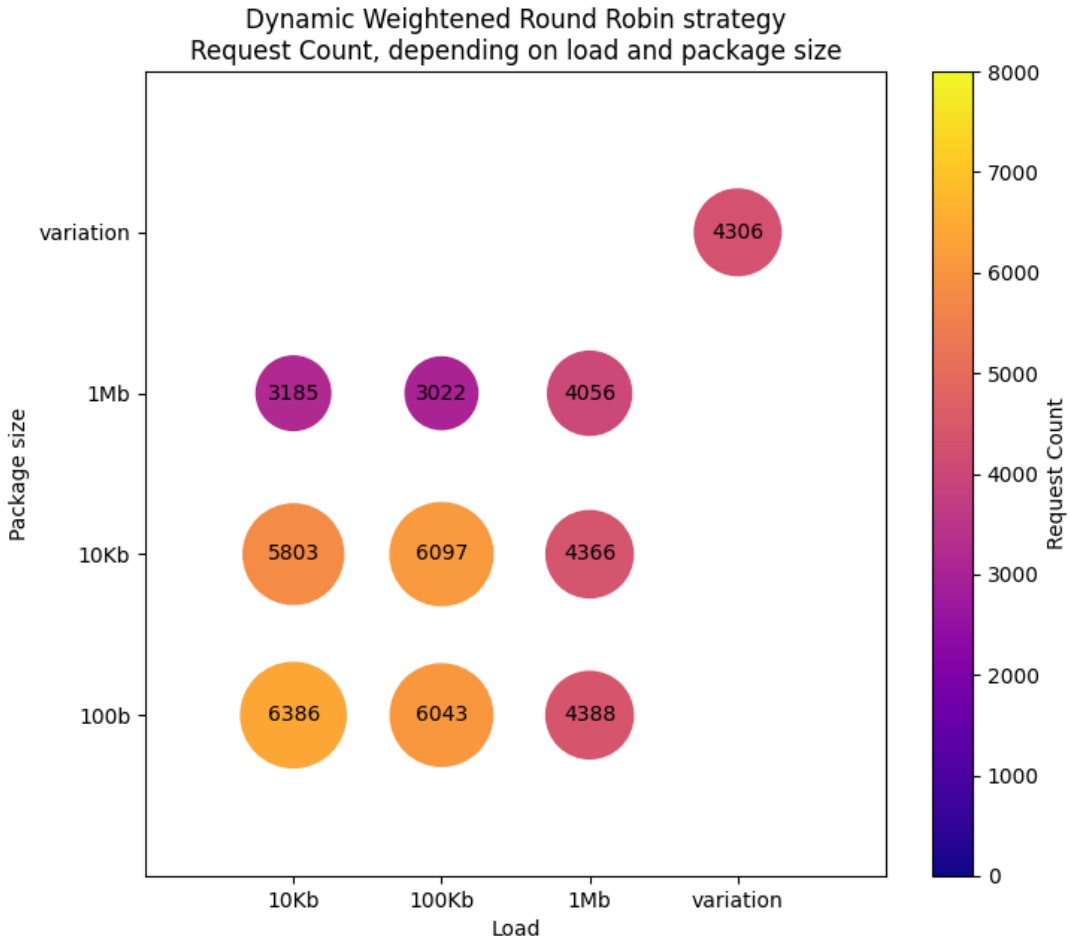


FIGURE A.23: Number of request in a time unit , using dwrr algorithm, depending on traffic parameters

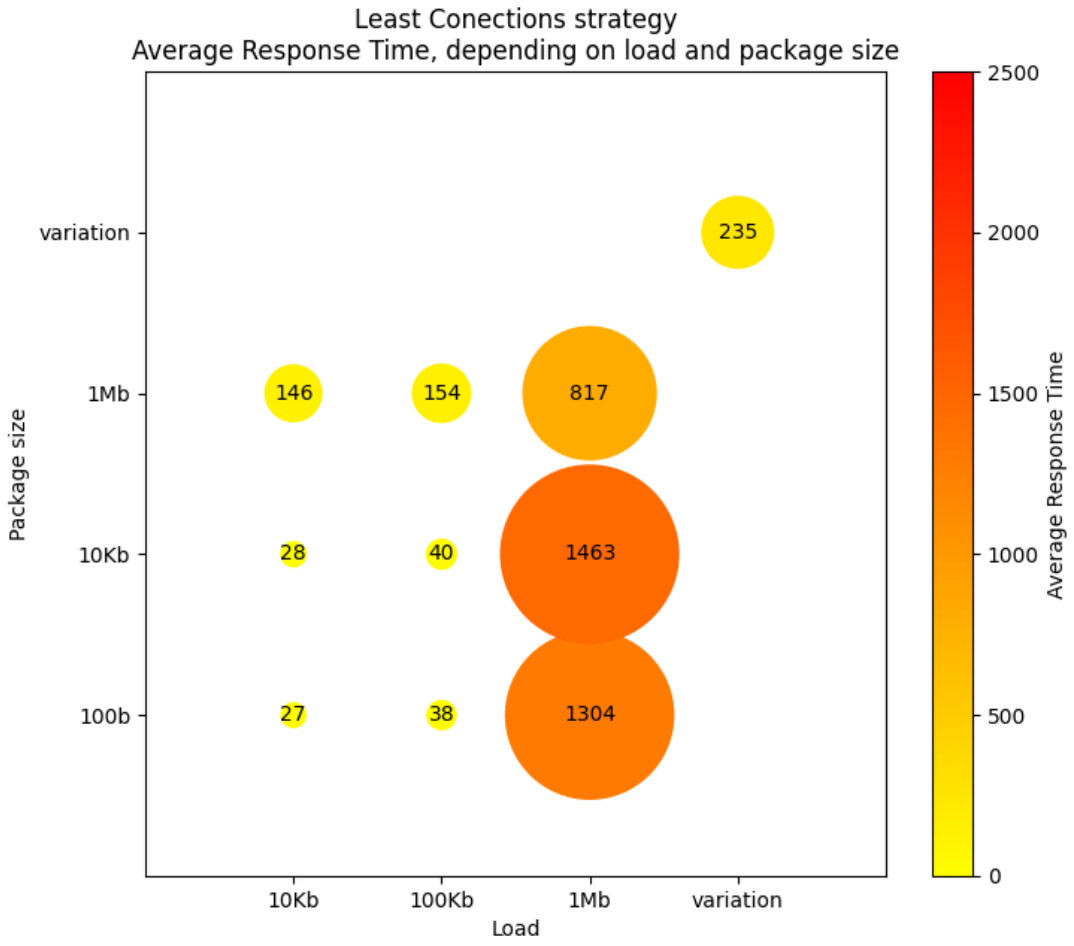


FIGURE A.24: Average response time, using lc algorithm, depending on traffic parameters

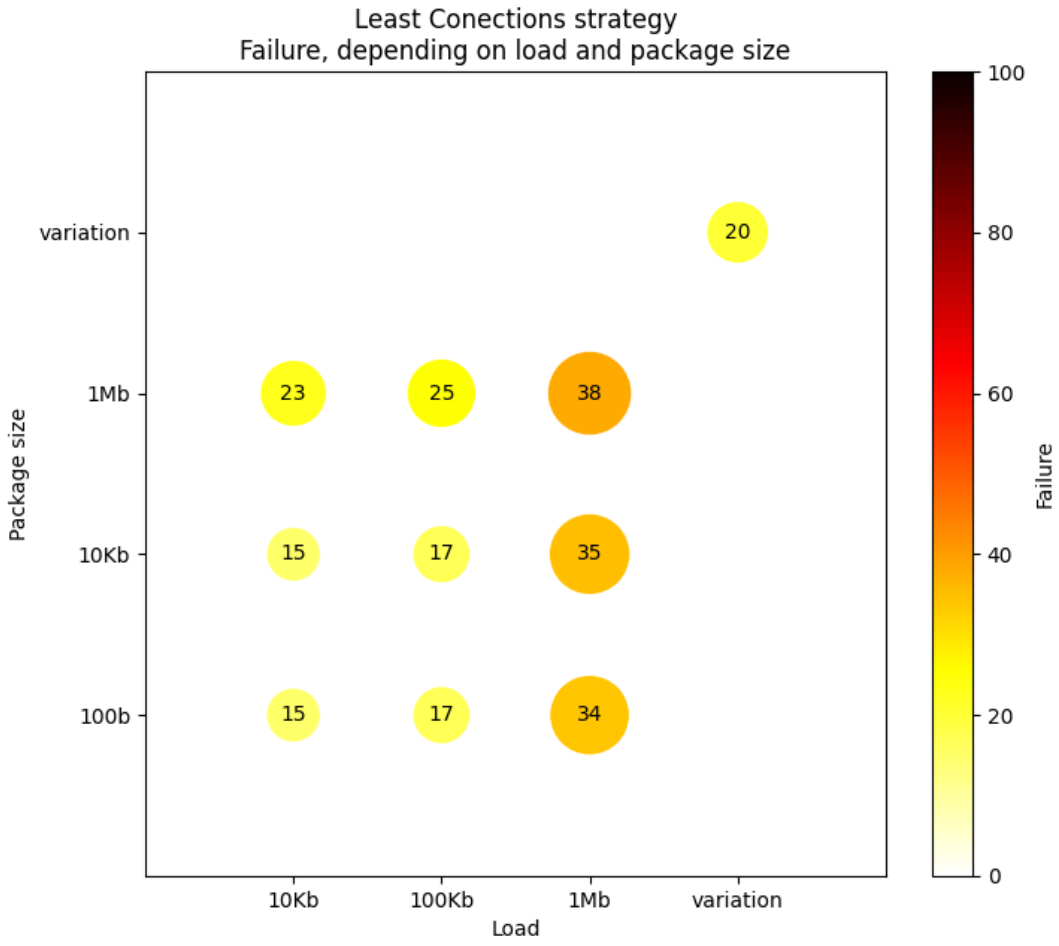


FIGURE A.25: Failure rate, using lc algorithm, depending on traffic parameters

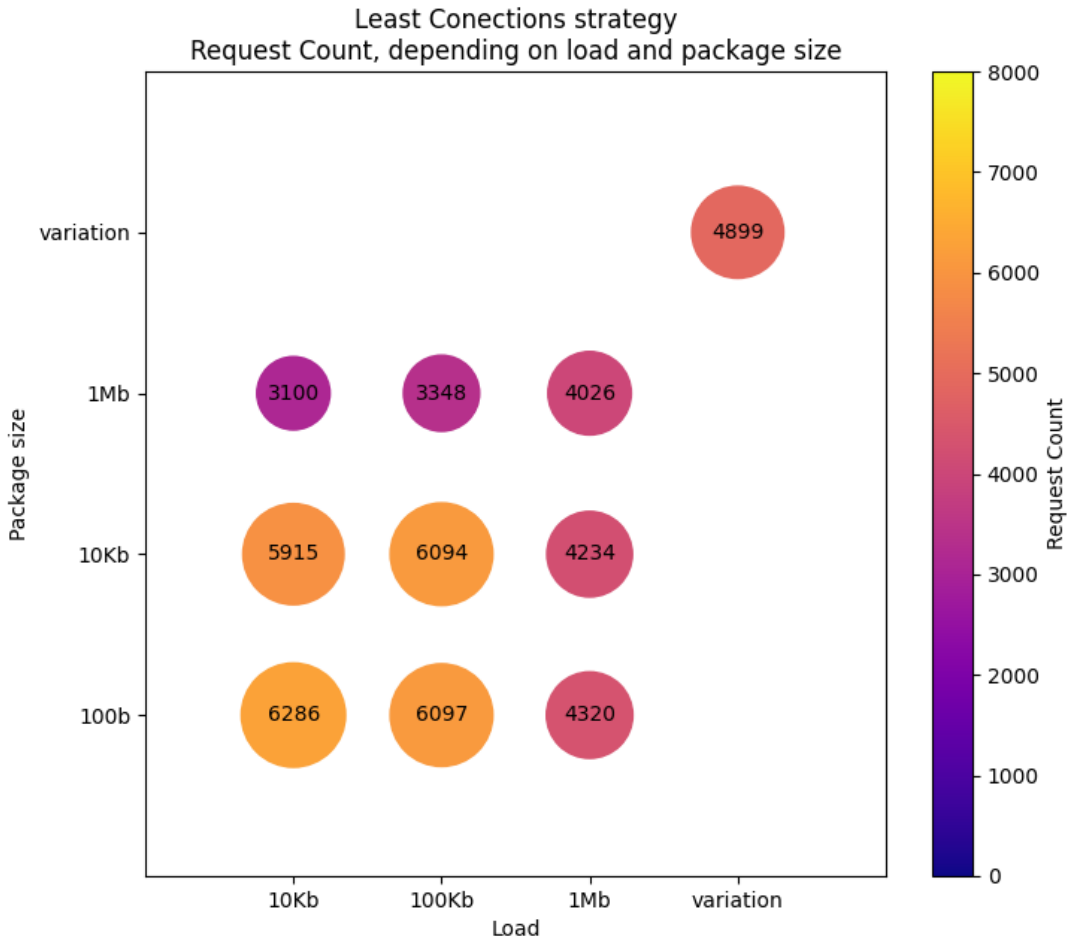


FIGURE A.26: Number of request in a time unit , using lc algorithm, depending on traffic parameters

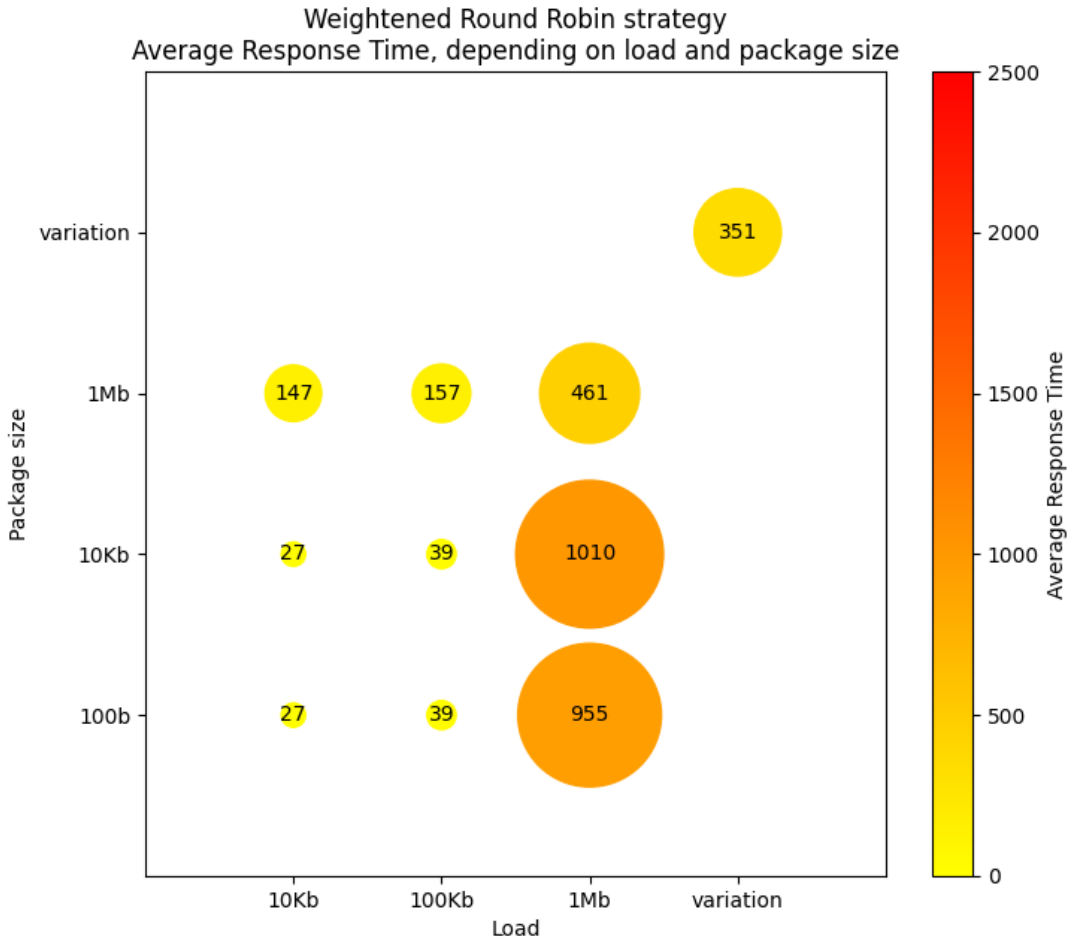


FIGURE A.27: Average response time, using wrr algorithm, depending on traffic parameters

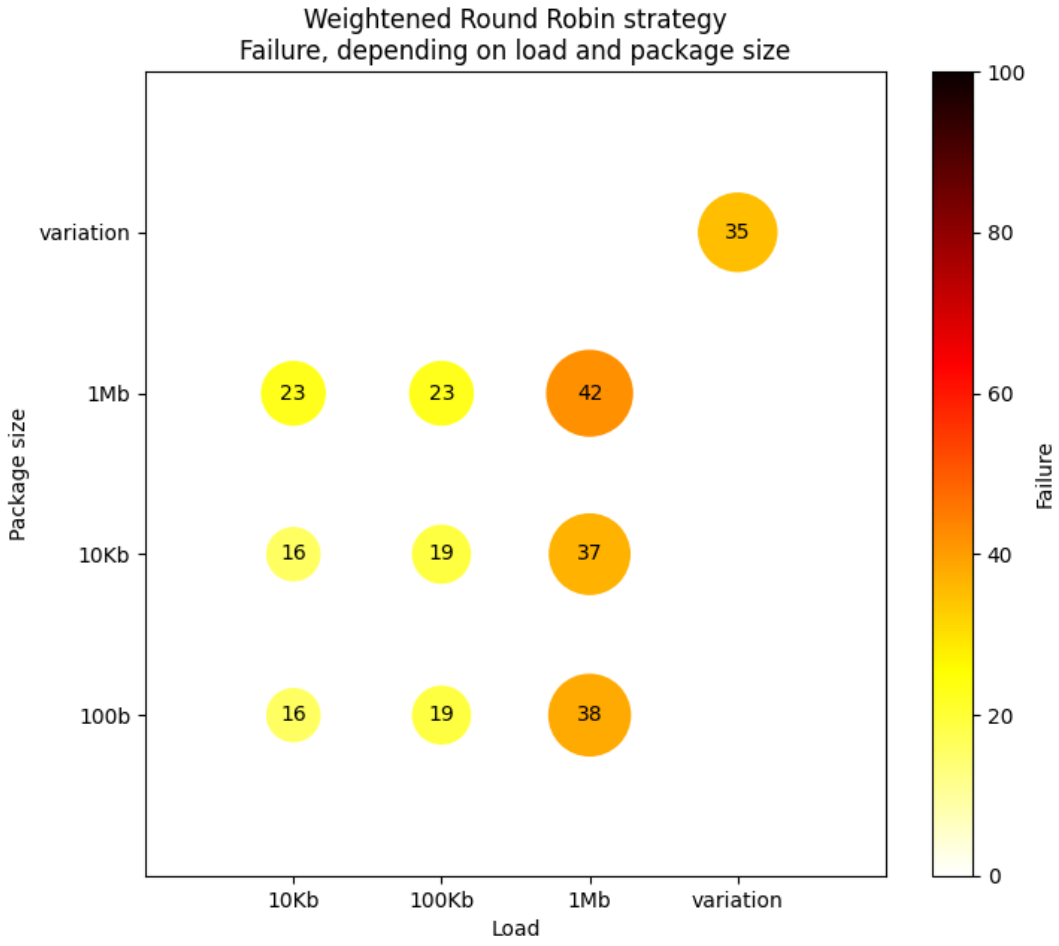


FIGURE A.28: Failure rate, using wrr algorithm, depending on traffic parameters

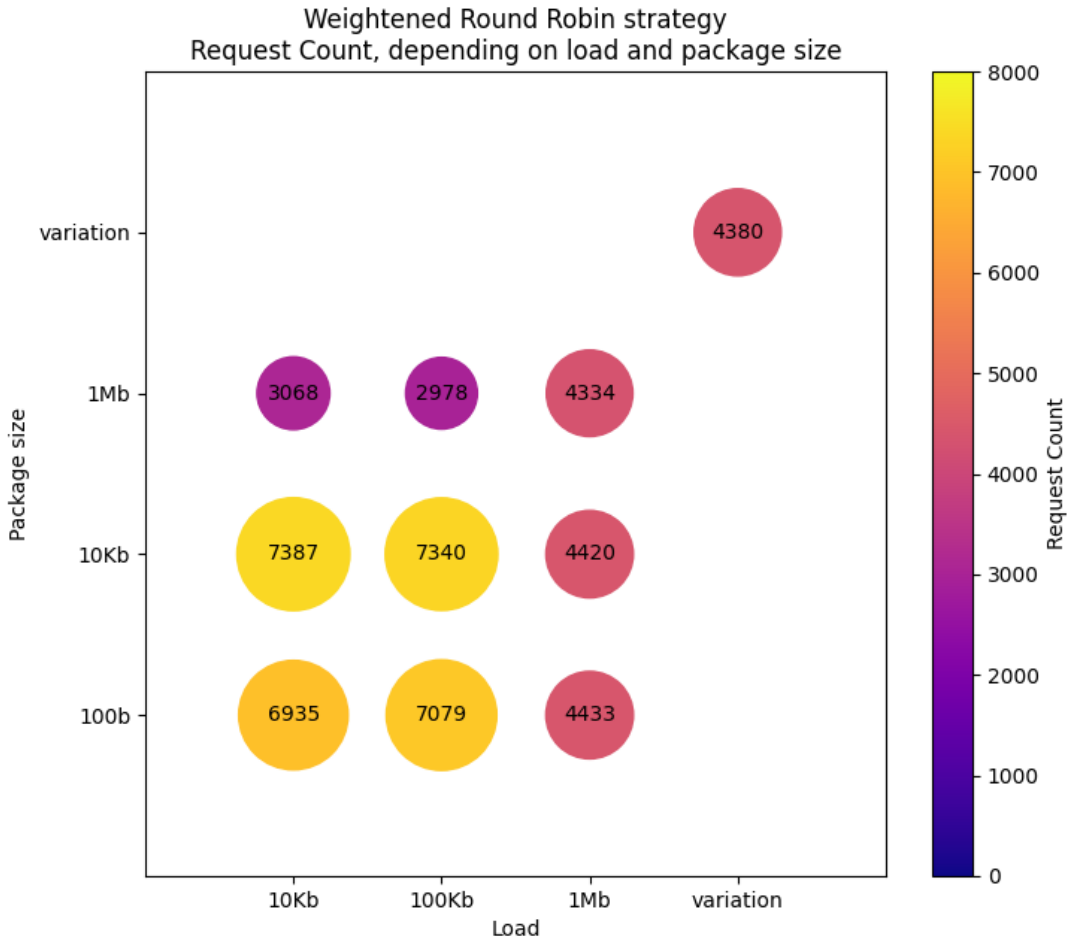


FIGURE A.29: Number of request in a time unit , using wrr algorithm, depending on traffic parameters

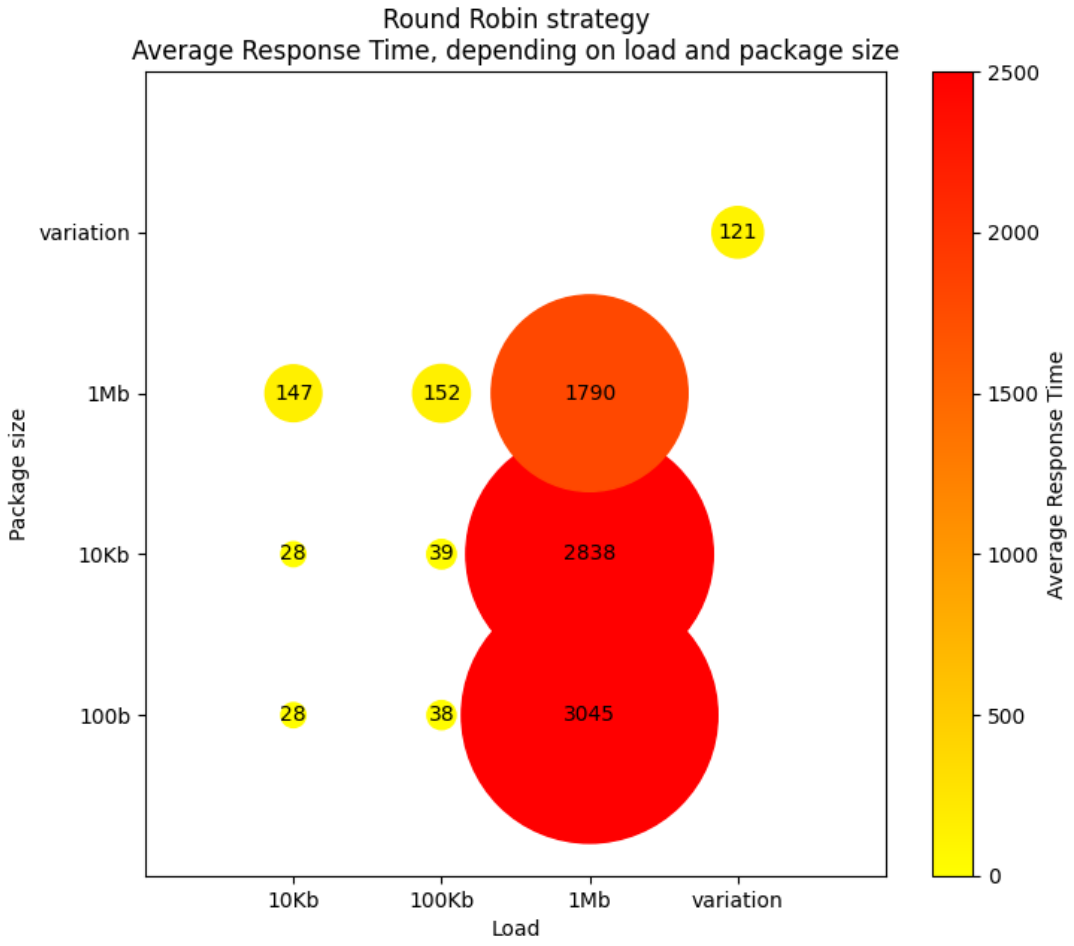


FIGURE A.30: Average responce time, using rr algorithm, depending on traffic parameters

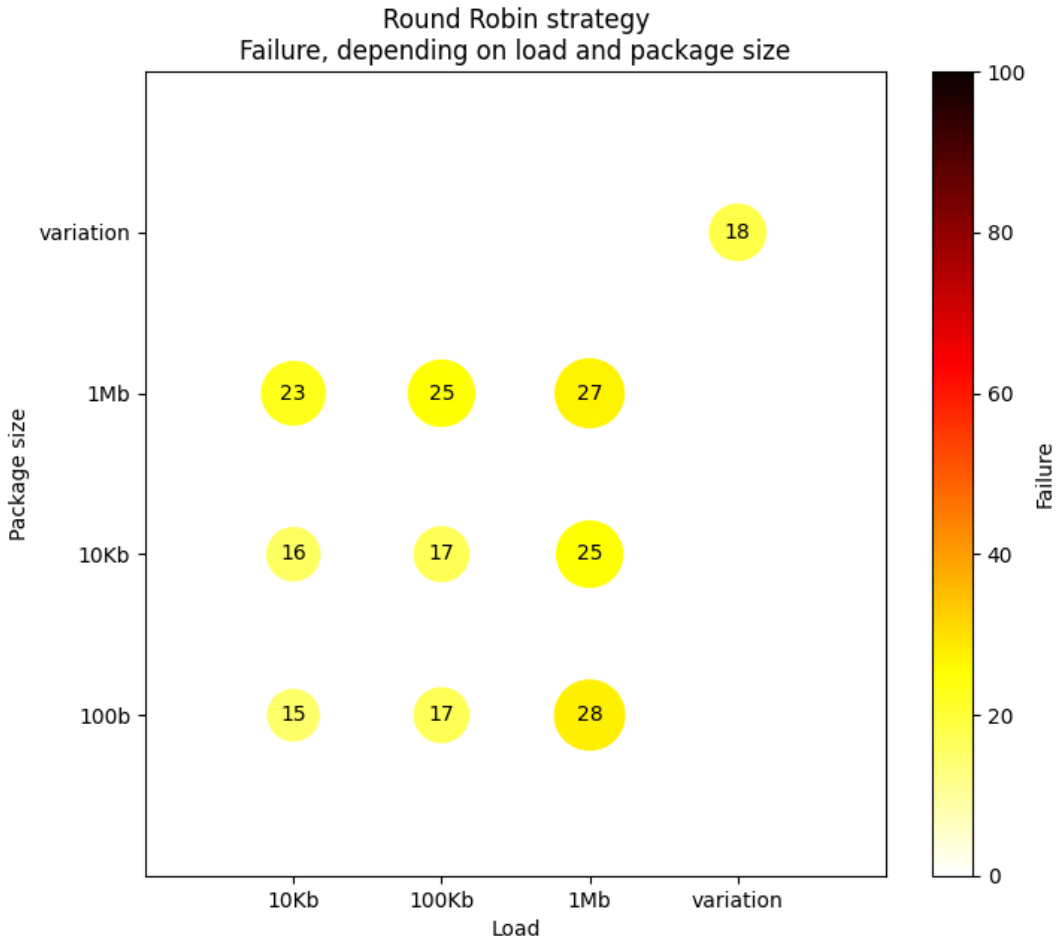


FIGURE A.31: Failure rate, using rr algorithm, depending on traffic parameters

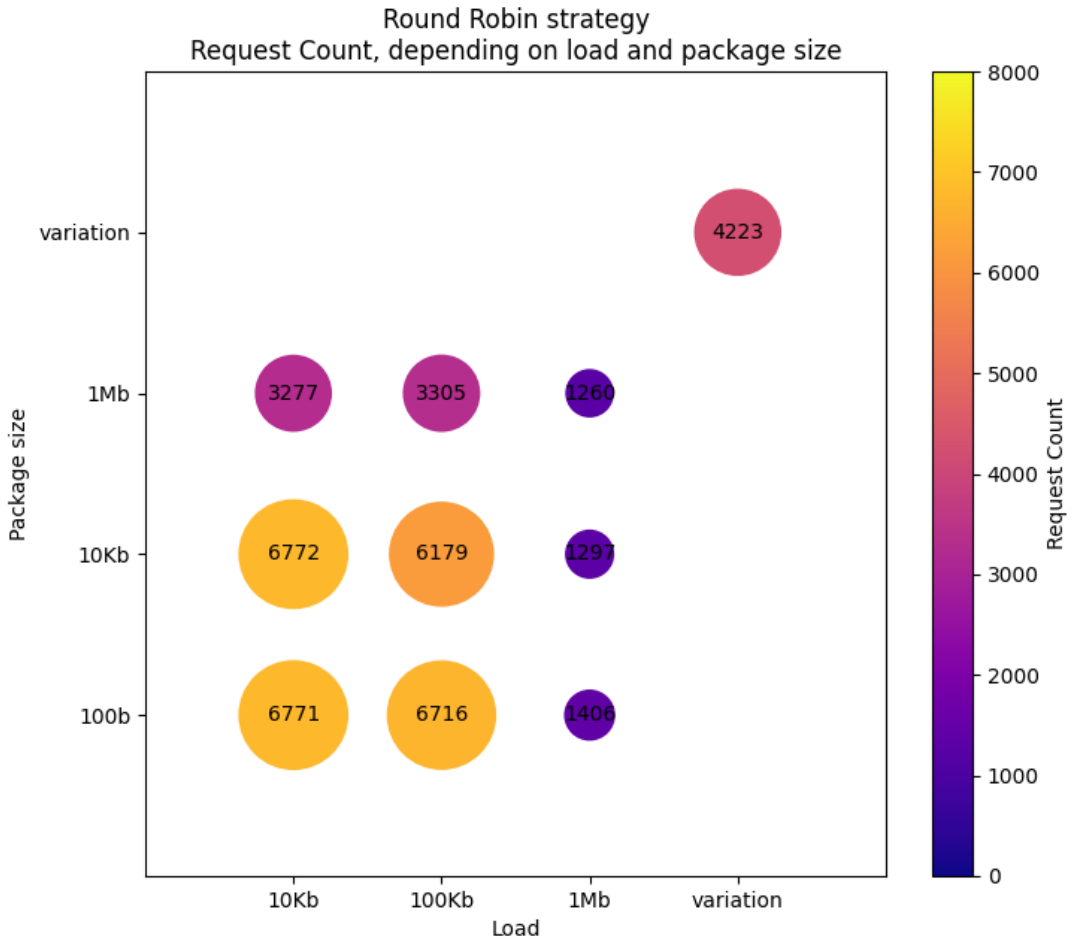


FIGURE A.32: Number of request in a time unit , using rr algorithm, depending on traffic parameters

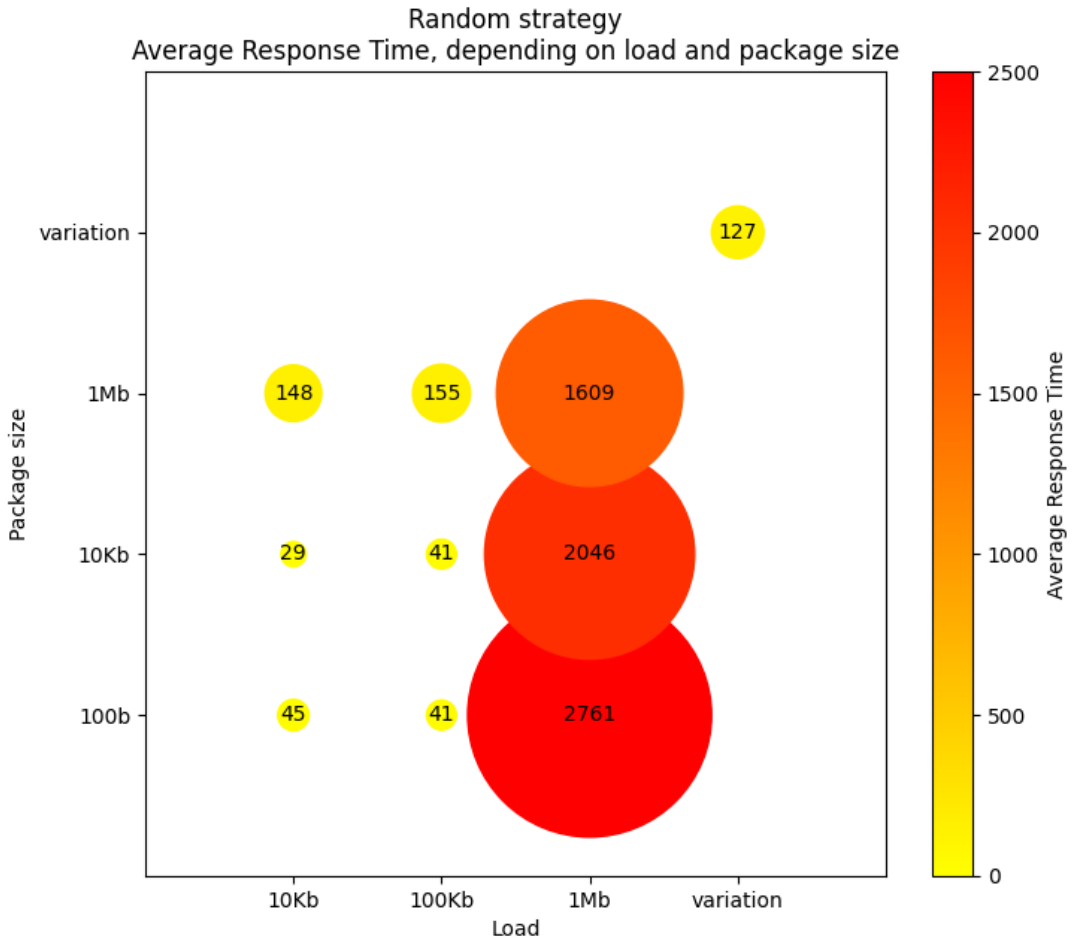


FIGURE A.33: Average response time, using r algorithm, depending on traffic parameters

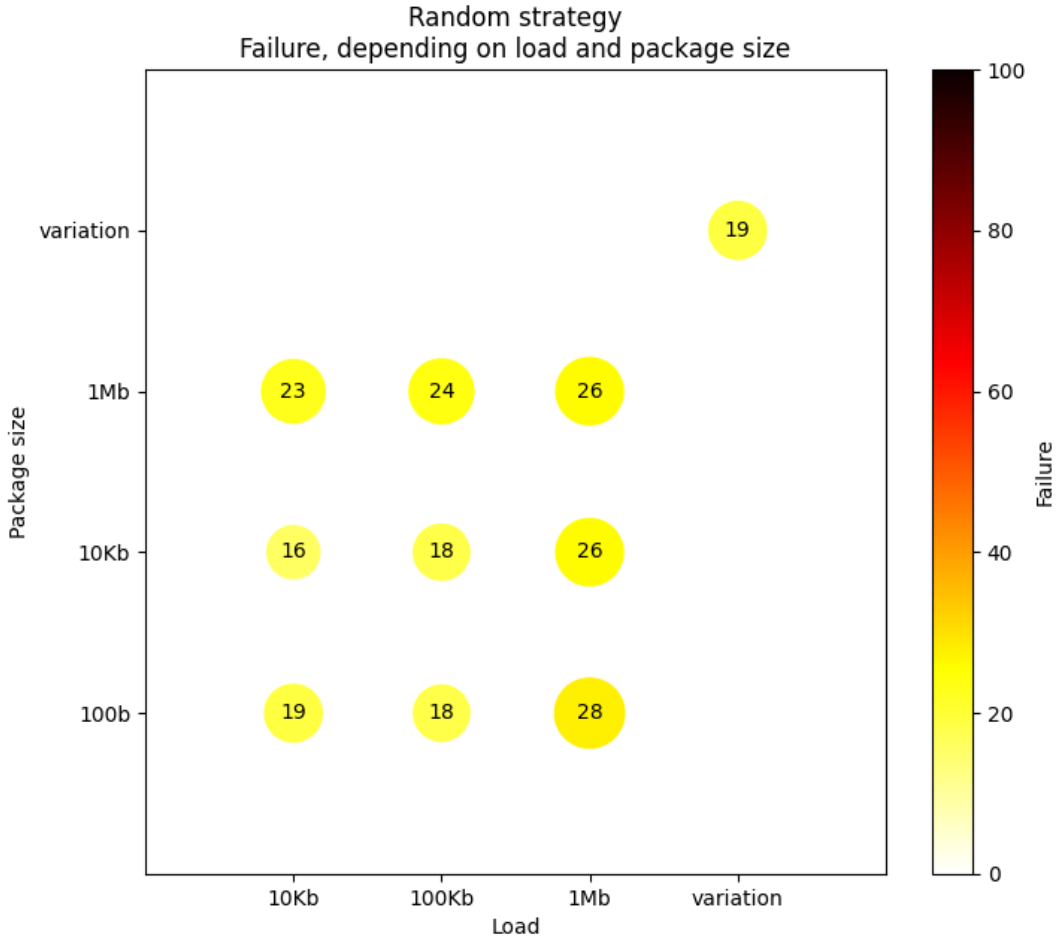


FIGURE A.34: Failure rate, using r algorithm, depending on traffic parameters

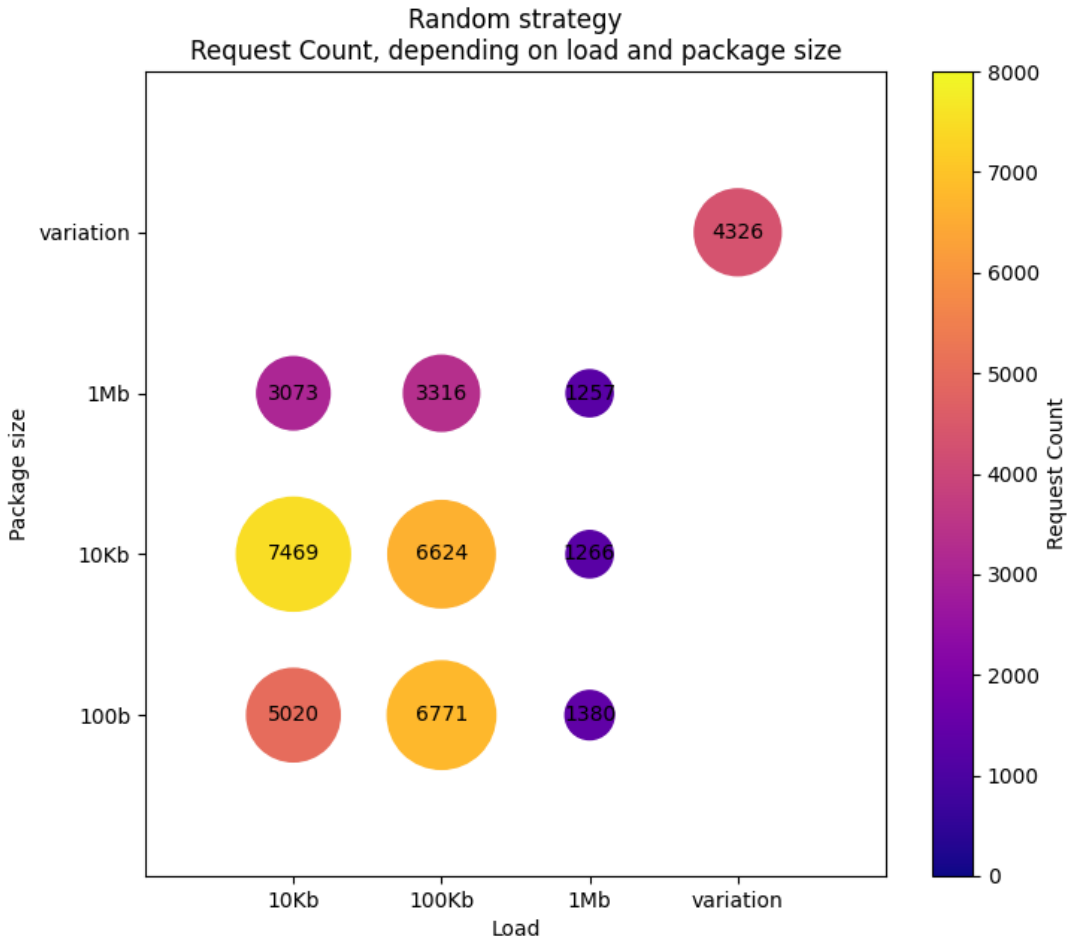


FIGURE A.35: Number of request in a time unit , using r algorithm, depending on traffic parameters

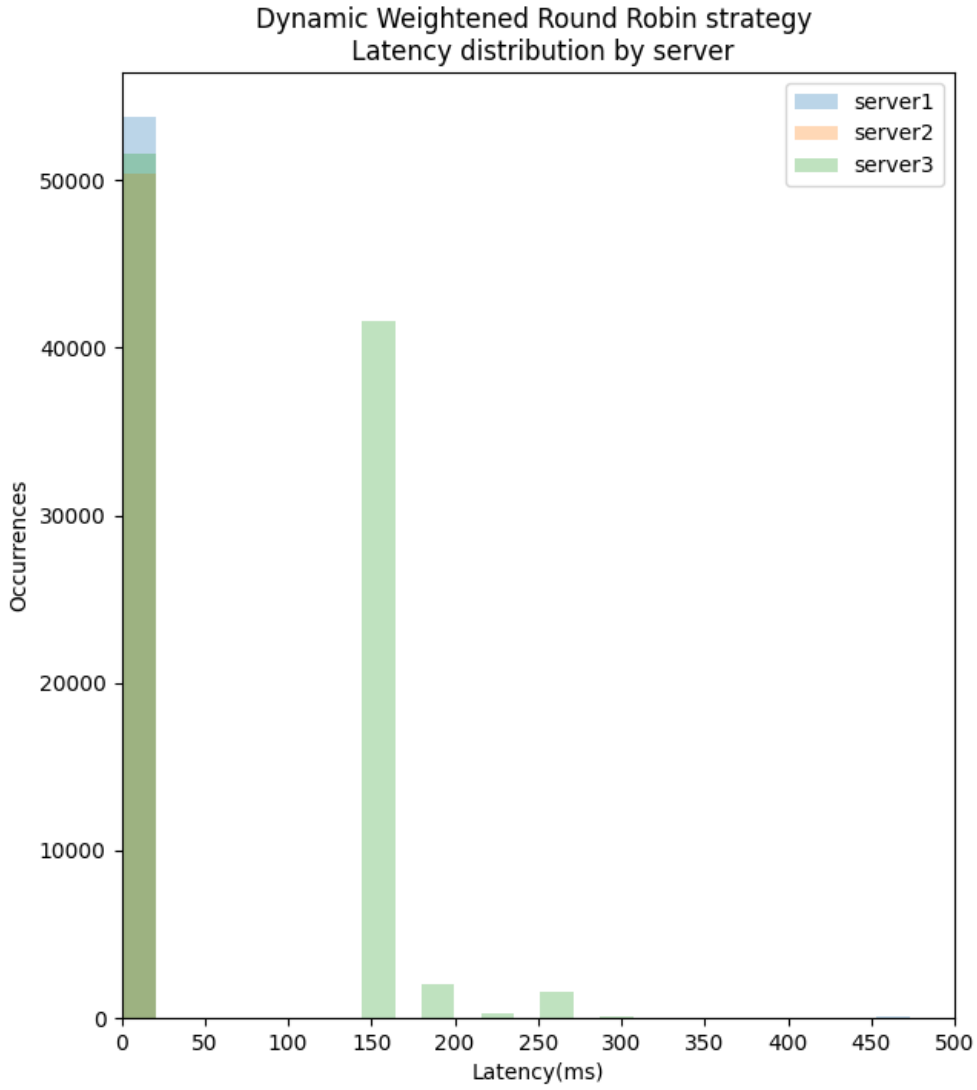


FIGURE A.36: Distribution of response time by server in dwrr algorithm

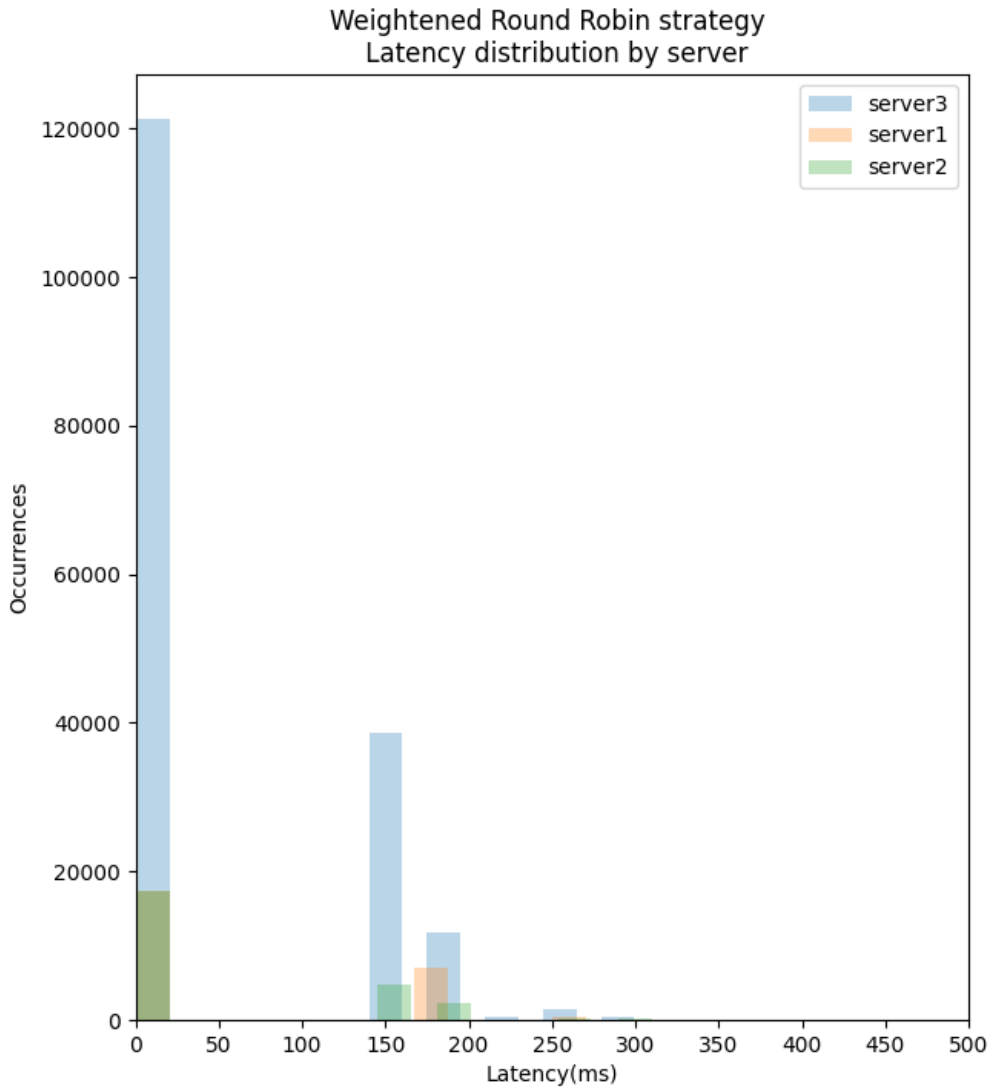


FIGURE A.37: Distribution of response time by server in wrr algorithm

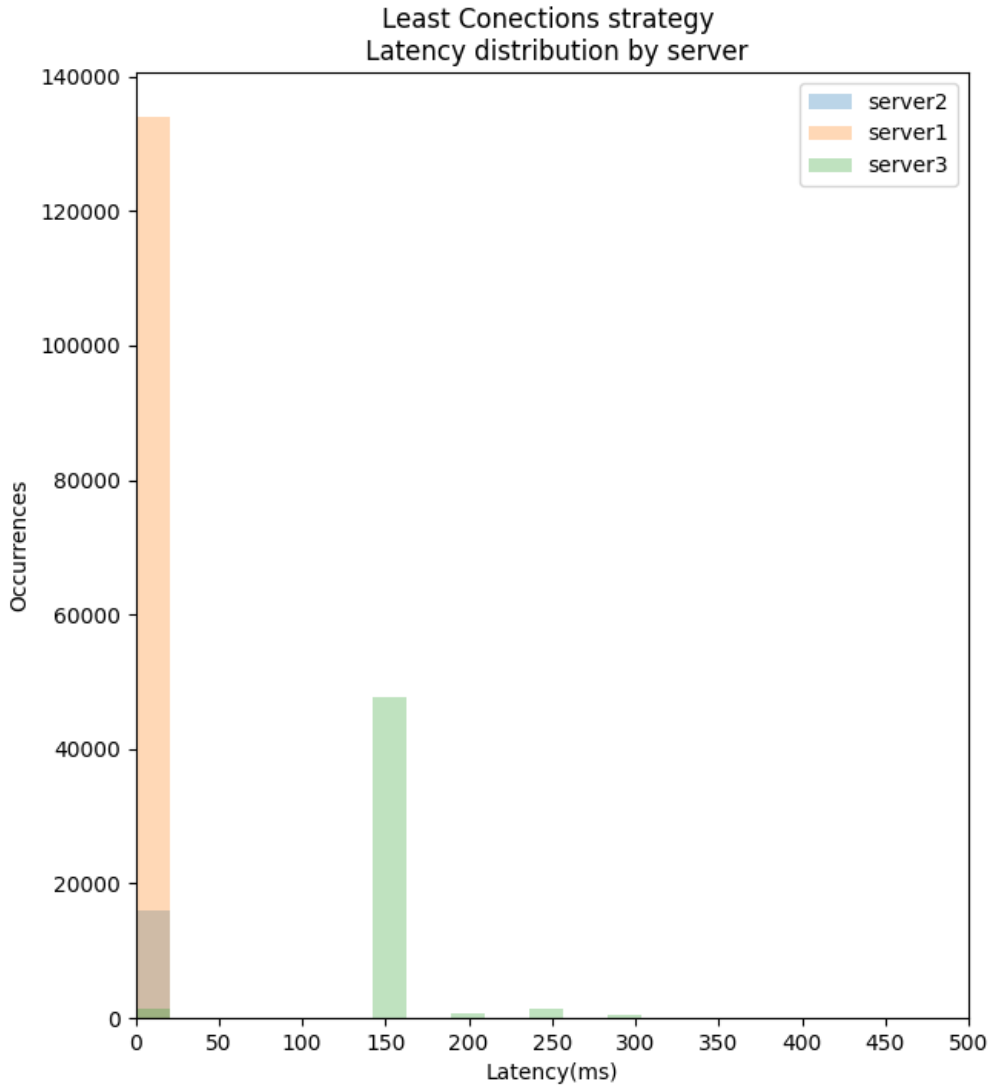


FIGURE A.38: Distribution of response time by server in lc algorithm

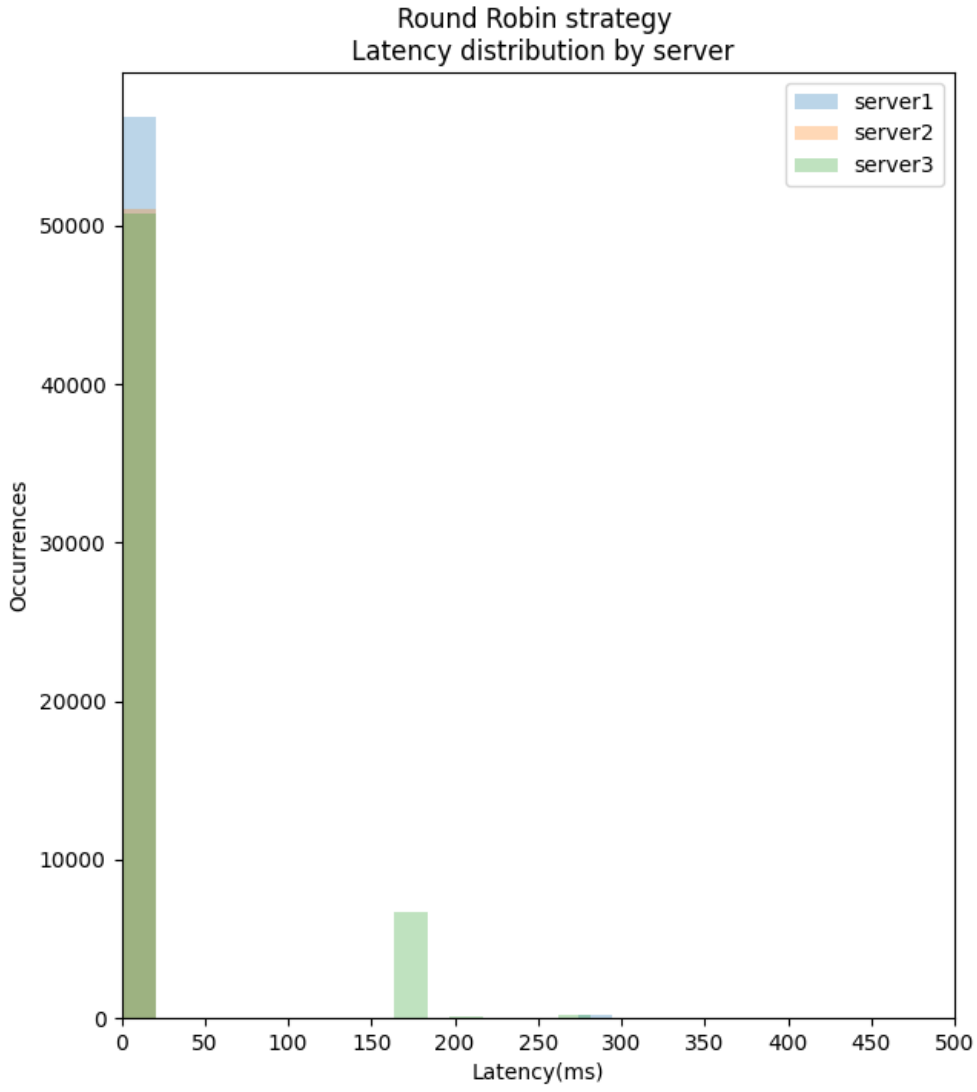


FIGURE A.39: Distribution of response time by server in rr algorithm

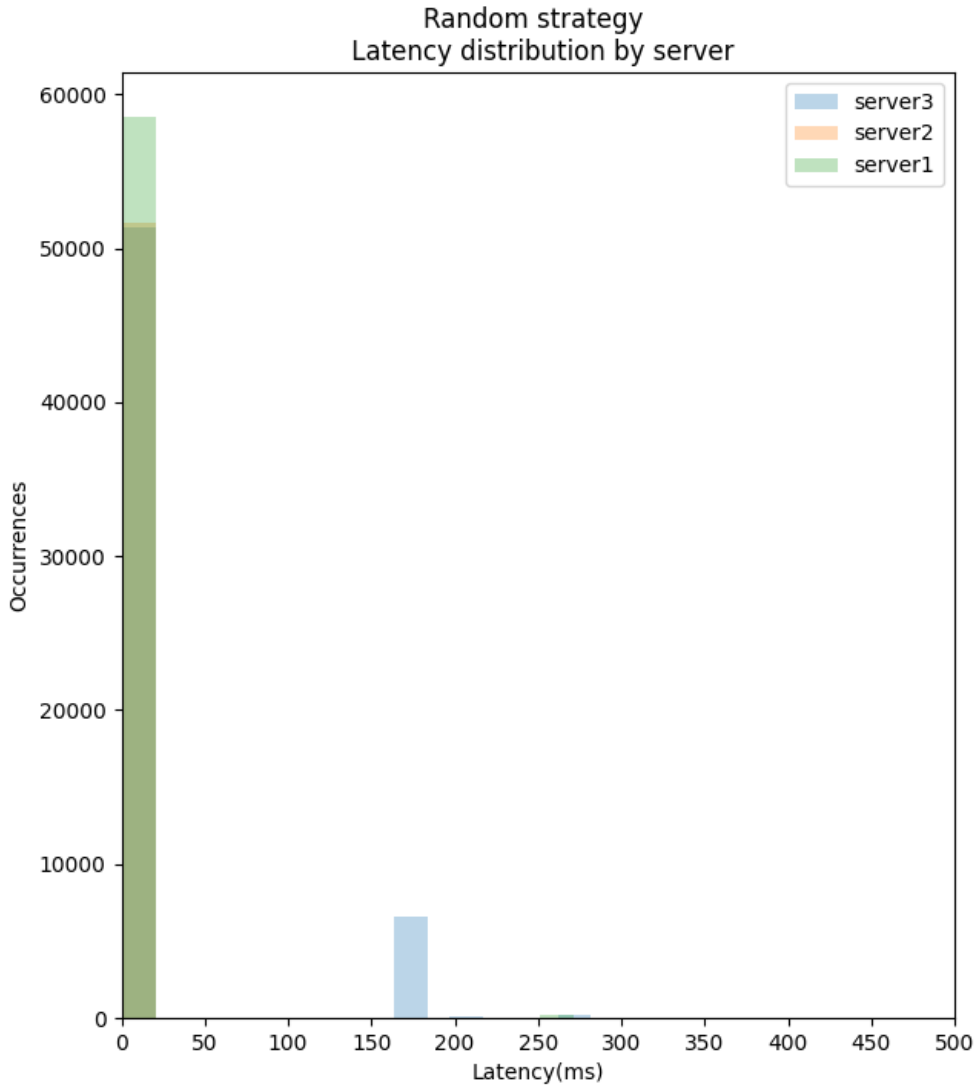


FIGURE A.40: Distribution of response time by server in r algorithm