Scenario-based synthetic traffic generation for web applications using workload patterns

Sana Khurram
s6409106860082@email.kmutnb.ac.th
The Sirindhorn International Thai-German Graduate School of Engineering, King Mongkut's University of Technology North Bangkok
Bangkok, Thailand

Alex R. Sabau
sabau@swc.rwth-aachen.de
Research Group Software Construction, RWTH Aachen University
Aachen, Germany

Horst Lichter
lichter@swc.rwth-aachen.de
Research Group Software Construction, RWTH Aachen University
Aachen, Germany

Sansiri Tanachutiwat
sansiri.t@tggs.kmutnb.ac.th
The Sirindhorn International Thai-German Graduate School of Engineering, King Mongkut's University of Technology North Bangkok
Bangkok, Thailand

ABSTRACT

Due to the growing number of internet users, Web applications face increasing challenges in providing constant availability, resilience to software failures, and rapid responses to user requests. Therefore, performance testing of web applications and monitoring them during high traffic are essential activities to evaluate the behavior of a system and ensure user satisfaction even during periods of high demand. However, the effectiveness of performance tests depends on the modeled user behavior in the tests. The closer the scenarios modeled in performance tests correspond to actual user behavior, the more valuable are the insights into the quality of the system that result from the performance tests performed. This paper presents a novel approach to traffic generation that leverages the concepts of workload patterns from cloud computing to model different scenarios of user interaction with a web application. The traffic generation can be used to test the performance and observe the behavior of web applications under real conditions of specific user interaction scenarios. The concepts are demonstrated in a proof-of-concept implementation and evaluated in a case study. The results of the case study show that the concepts do work as intended and the generated traffic follows the selected workload pattern. As a result, the resource usage of the case study system behaves differently in each scenario of user interaction with the web application.

CCS CONCEPTS

• **Software and its engineering** → **Object oriented architectures**; **Software performance**; **Software testing and debugging**; • **Information systems** → **Web applications**; • **Computer systems organization** → *Cloud computing*.

APIT 2024, January 29–31, 2024, Bangkok, Thailand
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1621-8/24/01
https://doi.org/10.1145/3651623.3651638

KEYWORDS

Traffic Generation, Web Applications, Performance Testing, Load Testing, Software Quality Assurance

ACM Reference Format:
Sana Khurram, Alex R. Sabau, Horst Lichter, and Sansiri Tanachutiwat. 2024. Scenario-based synthetic traffic generation for web applications using workload patterns. In *2024 6th Asia Pacific Information Technology Conference (APIT 2024), January 29–31, 2024, Bangkok, Thailand.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3651623.3651638

1 INTRODUCTION

Modern web applications must overcome unique challenges. Being available at all times and resilient to software failures, as well as responding quickly to user requests - the challenges of web applications meeting user requirements increased heavily in recent years due to the sheer amount of users of nowadays software. Moreover, they must be able to run stably and provide rapid feedback to the user even when client requests increase rapidly. This leads not only to major challenges in the design of the software but also in the infrastructure on which the software is deployed and operated.

To overcome these challenges, several approaches exist such as the microservice architecture pattern, which is optimized for high scalability of individual software components [9, 10], as well as IaaS and PaaS solutions in the cloud that allow high elasticity of IT resources for operated software components to be scaled[6]. However, performance testing and monitoring of productive software systems remains an utmost important task, as it provides indispensable information about the runtime behavior of a system under heavy load, which in turn heavily correlates with the software users' satisfaction [5, 7].

One way to define scenarios to test the performance of a system under real conditions can be found in the concepts of workload patterns [3]. The Collins dictionary defines a pattern as "the repeated or regular way in which something happens or is done" [1]. Workload patterns thus describe recurring scenarios in which the IT resources of a system are used in a certain way, e.g. due to recurring scenarios in the behavior of users with a system.

To assess the performance of web applications under varying levels of load, one effective approach is to generate synthetic messages that are then directed at the APIs of the web application being tested. In this research paper, we introduce a new method for creating workloads in web applications. This method is inspired by the workload patterns identified by Fehling et al. [3] and involves generating HTTP messages that are shaped as the desired workload pattern. Moreover, it adds a "human touch" to the generated traffic, as human behavior is uncertain and unpredictable and never follows a perfect pattern. Our contribution aims to improve the realism of performance testing for web applications. We formulate the following two research questions:

RQ1: How can workload patterns be leveraged to model real-world user behavior scenarios for performance testing web applications?

RQ2: How can synthetic traffic that is modeled on workload patterns be meaningfully generated in such a way that it contains fluctuations that reflect human behavior?

In this paper, we first discuss a comprehensive review of related work in traffic pattern generation to highlight their limitations in Section 2. Section 3 describes the proposed solution for generating and shaping traffic as workload patterns to mimic real-world user interactions using Django and modern technologies. Section 4 covers the evaluation and results, verifying the fidelity of generated traffic patterns. In Section 5, we discuss the effectiveness of our approach in simulating real user behavior and its implications. The closing section 6 offers a conclusion summarizing key contributions and future work.

2 RELATED WORK

MACE, a framework for malicious traffic generation, provides a unique environment for recreating a wide range of malicious packet traffic in laboratory testbeds. It defines a model for the flexible composition of malicious traffic, enabling the creation of both known attacks and new attack variants. This mainly consists of three main components: exploits, obfuscators, and propagation elements [13]. It is written in Python and facilitates the creation of attack vectors with interpretation, execution, and exception handling functions. It supports dynamic exploit and obfuscation generation, utilizing payload and header construction elements.

On a different note, SURGE is an application-aware traffic generator specifically designed for testing web server performance. It complements MACE as it can generate background traffic in a controlled and configured manner. This tool relies on historical data to create references that closely mirror real-world measurements of various aspects of web workloads. These include characteristics like server file size distribution, request size distribution, relative file popularity, embedded file references, temporal locality of reference, and idle periods of individual users. These measurements are drawn from actual observations of web server usage, underlining the tool's foundation in real-world data. [2].

In the context of media service streaming, generating synthetic and shaped traffic is pivotal for understanding user behavior and system performance. MediSyn is a synthetic streaming media service traffic generator developed to capture the characteristics of new workloads in shared hosting services. By leveraging extended-term traces from real streaming media services, MediSyn models the persistent behavior of these services, including the introduction of new files and changes in file popularity. It introduces a novel generalized Zipf-like distribution that accurately represents the popularity of web objects and streaming media. The generator validates the statistical models by comparing the generated workloads with data from two representative streaming media server logs collected over an extended period [14].

Another notable paper focuses on the generation of synthetic network traffic data using the STAN approach, which utilizes autoregressive neural models. The goal of traffic generation is to create nearly realistic synthetic data that can be used as an alternative to real network traffic data, which is often difficult to obtain due to privacy concerns. STAN integrates convolutional neural layers (CNN), mixture density layers (MDN), and softmax layers to capture both temporal dependence and dependence between attributes in the generated traffic data [15]. This architecture allows STAN to capture both temporal dependence and dependence between attributes in the generated traffic data. The results show that models trained solely on synthetic data have only a small decline in accuracy compared to models trained on real data.

Traffic generation is crucial for testing intrusion detection systems (IDS), which play a vital role in protecting networks against potential threats and attacks. MUCUS IDS simulator is designed for black-box testing of network intrusion detection systems, where the internal workings of the IDS are not disclosed to the tester. This simulator enables security researchers and developers to evaluate the effectiveness and accuracy of intrusion detection systems under realistic network traffic scenarios. By utilizing signatures of established network-based intrusion detection systems like Snort, the tool generates synthetic network traffic that emulates malicious behavior and various types of attacks [8].

Another noteworthy method involves using statistical models and algorithms to simulate resource utilization. This tool, known as BURSE, generates and shapes traffic that imitates real-world workloads, featuring bursty and self-similar characteristics, such as sudden resource spikes and repetitive resource usage patterns. Such synthetic traffic shaped as workloads can be used to evaluate the performance, scalability, and reliability of an application under different loads and conditions [16].

All aforementioned generators are specifically designed for a particular type of system. Additionally, the existing traffic generators lack the ability to realistically simulate varied traffic patterns, as they mostly provide only one type of traffic pattern and rely heavily on historical data. Our approach differs from the ones presented, as it focuses on the various recurring user interaction scenarios of web application users. Thus, it can be used out of the box without the need for historical data while still allowing web applications to be tested in various well-known user interaction scenarios.

3 PROPOSED SOLUTION

Fehling et al. classify five workload patterns in the domain of cloud computing. Each workload pattern follows a certain course of how IT resources are utilized. The workload patterns are briefly described in the following [3]:

- ***Static:*** Resources exhibit consistent utilization, steady access, and minor fluctuations. Static workloads, while exhibiting slight variations, maintain consistent resource use.
- ***Periodic:*** Recurring resource utilization patterns, evident in a cyclic activity like retail orders peaking throughout a year. Analyzing recurring intervals, spike intensity, and duration reveals these patterns.
- ***Random:*** Unpredictable resource use fluctuations, challenging management. Factors like user behavior or external events induce this randomness. Social media spikes due to viral content exemplify random workload.
- ***Once-in-a-lifetime:*** Resources experience equal use until a one-time peak. Triggered by unique events, e.g., product launches, such spikes disrupt regular usage, unlikely to recur.
- ***Continuously changing:*** Dynamic utilization shifts, gradual and persistent. Adaptation to evolving demands poses challenges. Live streaming platforms with varying traffic due to user behavior and content trends exemplify this pattern.

A synthetic traffic generator that conforms to the presented traffic patterns must be able to generate and send a series of messages to the system under test during the course of a test run. During a test run, the number of requests sent in each second must evolve according to the shape described by the desired traffic pattern. In the following, we describe how we developed our solutions.

3.1 Defining Workload Pattern Parameters

As a first step, we identified a set of parameters that effectively describe the distinct characteristics of workload patterns and named them shaping parameters. This approach allowed us to map the concepts of workload patterns to the concepts of object orientation, keeping the development effort low as well as the extensibility of the concepts high. Moreover, we introduced behavior-related parameters aimed at incorporating realistic human behaviors into the workload patterns. We identified the following common shaping parameters for all workload patterns:

- **Total Time**: This parameter is the total time taken by one traffic generation run.

Apart from the static workload, the two parameters below are introduced across all other workload patterns

- **Minimum Number of Requests**: Indicates the lowest number of requests that can be transmitted.
- **Maximum Number of Requests**: Indicates the highest number of requests that can be transmitted.

3.2 Generating traffic from workload patterns

In order to arrange traffic as a distinct workload category, a precise arrangement of parameters is required to replicate it in a real-world scenario.

3.2.1 Static Traffic: A static traffic pattern, when depicting a constant workload, can exhibit either consistent resource utilization over time or show variations within defined limits, resembling the characteristics of an approximate static sinusoidal wave. In this traffic shaping approach, we model user behavior using sine waves, as this approach affords us greater control over user behavior while also incorporating an approximation of human interaction. To generate a static workload trend, we introduce the following additional shaping parameters:

- **Number of Requests**: When the "Type" parameter, which is introduced next, is set to "Straight", it results in the creation of a traffic pattern characterized by consistent resource utilization, maintained through uniform sleep intervals and sending a constant number of requests that is equal to the number of requests entered by the user. On the other hand, when the "Type" parameter is set as "Fluctuated", a distinct approach is adopted. Such a static trend representing traffic assumes the shape of a stable sine wave, oscillating around a midpoint that is equal to the number of requests entered by the user.

In addition to the shaping parameter we introduce a behavior-related parameter, that adds a human touch to the workload:

- **Type**: This parameter can take two values: "Straight" or "Fluctuated". When set to "Straight", the generated traffic forms a straight line representing constant resource utilization. Conversely, when set to "Fluctuated", the traffic takes on the characteristics of a static workload with occasional fluctuations, resembling an approximation of a static sinusoidal wave.

In order to represent traffic as a fluctuated static trend, we divide total time into time intervals. These gaps in time mark the times at which sets of requests happen. For each interval, we create a series of requests using a curved pattern, like a sine wave. This wave tells us how many requests to make at each point. The pattern gradually goes up and then down, always staying within the initial range. Within each interval, and for every request group or data point identified on the sinusoidal range, the precise time gap between each request is calculated. This calculation is rooted in the overall interval duration and the specific request count pertaining to that particular data point. To incorporate a more authentic and realistic static behavior in the traffic, a brief sleep time is introduced between each request.

3.2.2 Periodic Traffic: To model traffic conforming to the periodic workload pattern, several supplementary parameters come into play. To generate a periodic workload trend, we introduce the following additional shaping parameters.

- **Number of Elements on each Side of the Wave**: This parameter plays a pivotal role in shaping the periodic wave. It defines the number of requests in each group that represent data points, spanning from the trough of the sine wave to its peak or vice versa.

We also introduce behavior-related parameters that add a human touch to the workload as follows:

- **Difference Variable**: Adjusts the range of viable request counts by subtracting it from the “Maximum Number of Requests” and adding it to the “Minimum Number of Requests”.
- **Rest Time at Lowest Peak**: Extends the resting phase at the lowest peak of the wave by elongating the width of the trough. This manipulation enables the replication of scenarios where fewer requests are transmitted during this resting period, adding a touch of human behavior.

Suitable intervals are computed for the sine waves based on the parameters "Total Time" and "Rest Time at Lowest Peak". Deducting the "Rest Time at Lowest Peak" from each interval yields the sleep time during the trough. Introducing a sleep time or more simply a delay, between requests helps to shape the trend. Subsequently, a well-ordered but randomized list of request numbers is formulated, mirroring sinusoidal waves with gradual inclines and declines. This list consists of the group of requests or data points of sine waves within each interval. The equitable distribution of the interval's duration among specific points regulates the temporal spacing, determining and documenting the interval for each point.

For each interval and its corresponding request count, a loop orchestrates the process of generating and dispatching requests. Following preparation, each request is sent, and a brief pause is introduced between them, replicating the temporal dynamics of real-world workloads. This inter-request pause substantiates the intervals between requests, eventually giving rise to the periodic wave-like pattern.

3.2.3 Random Traffic: To generate traffic conforming to the random workload pattern relies on the three fundamental parameters that are already discussed i.e. minimum and maximum number of requests, and total time. Nevertheless, we also introduce another behavior-related parameter as follows:

- **Total Data Points**: Divides the Total Time into equal intervals. This inherent randomness is achieved by employing the Poisson probabilistic distribution, which governs the number of requests dispatched within each interval. This parameter plays an important role in adding a touch of human-like characteristics. This parameter divides the Total Time into equal intervals. This inherent randomness is achieved by employing the Poisson probabilistic distribution, which governs the number of requests dispatched within each interval.

The Poisson distribution's ability to model random events within constraints makes it ideal for predicting events within a fixed time or space based on an average occurrence rate. The simulation uses these time intervals to progress iteratively. In each interval, a random number of request groups is generated using the Poisson distribution. To add realism, brief sleep durations are inserted between requests, mimicking the pacing of real workloads.

3.2.4 Once-in-a-lifetime Traffic: To replicate this distinctive pattern, several supplementary parameters are introduced including various shaping parameters as follows:

- **Number of Elements on each Side of the Wave**: This is a key parameter influencing the formation of periodic waves. It creates the number of points representing request groups, from the trough to the peak of the sine wave, or vice versa.
- **Once-in-a-Lifetime Peak Occurrence Time**: This parameter pinpoints the exact moment of the unique peak's appearance.
- **Once-in-a-Lifetime Peak Occurrence Number of Requests**: This is a parameter that denotes the number of requests that are to be sent during this peak. This parameter plays a significant role in shaping the traffic as a once-in-a-lifetime pattern.

Shaping the traffic as a Once-in-a-lifetime workload pattern adheres to the underlying structure of a typical periodic pattern but omits any rest periods at its lowest points. While rest times at troughs were integrated into the periodic workload to mimic human dynamics, the focus in this workload shape is to craft a distinctive pattern of the traffic defined by an exceptionally high peak.

In order to send the traffic in this type of workload, appropriate intervals are computed for the sine waves based on the total time. This is followed by the creation of a sorted but randomized list of request numbers, ascending and descending alternatively. This list emulates sinusoidal waves, gradually increasing and then decreasing the number of requests similar to shaping the periodic workload. The numbers in the list determine the count of requests within each interval.

Within each interval, corresponding to the designated request count, a loop manages the generation and dispatch of requests. These requests are then prepared and transmitted, with a brief interval inserted between each to mimic real workload dynamics. Alongside the standard periodic transmission pattern, a unique event unfolds as time coincides with the once-in-a-lifetime occurrence peak. At this moment, a sharp and substantial increase in requests occurs creating an exceptional once-in-a-lifetime peak.

3.2.5 Continuously changing Traffic: A continuously changing workload pattern is characterized by an uninterrupted and gradual rise or fall in resource utilization[3]. For shaping a continuously changing workload pattern, the objective is to replicate a dynamic sequence of requests that either escalates or gradually diminishes in a straight or exponential manner within a defined time frame. To simulate traffic in such a continuously changing trend, the common parameters are used along with an addition of a behavior-related parameter:

- **Trend**: This parameter classifies what type of continuously changing workload is to be shaped. Four values have been proposed for this variable: straight increase, straight decrease, exponential increase, and exponential decrease.

When simulating an exponential trend, the number of requests steadily grows or shrinks based on a calculated exponential rate. For exponential increase, requests continuously and exponentially increase until they reach the maximum. For exponential decrease, they systematically decrease from the maximum using an exponential rate. The rate for both increase and decrease is determined as follows:

$$\text{rate} = \frac{\text{max\_requests\_num}}{\text{total\_time}} \tag{1}$$

This choice of the above equation offers several advantages for our study. Firstly, its simplicity and intuitive nature make it accessible for both researchers and practitioners. By adjusting the maximum number of requests and the total time, we can easily control the behavior of exponential growth or decay, facilitating a continuously changing traffic trend.

For the trends, straight increase and straight decrease there is a linear increase or decrease in the number of requests sent over time. The rate of increase and decrease is calculated as follows:

$$\text{rate} = \frac{\text{max\_requests\_num - min\_requests\_num}}{\text{total\_time}} \tag{2}$$

By adjusting the maximum and minimum number of requests and the total time, we can easily control the behavior of linear growth or decay.

3.3 Implementation

In the implementation process using Django, we integrated a range of contemporary technologies and frameworks, with the core being the Django web development framework itself. Highly regarded for its "batteries included" philosophy, Django greatly simplified the creation and management of a traffic generator for workload patterns [4].

We also utilized supplementary libraries including NumPy, pandas, and Django Rest Framework to enhance our implementation. A key transformation came through the incorporation of the factory design pattern, which enabled flexible and decoupled object creation [11]. As a client, we developed a simple frontend with HTML and CSS to provide a user-friendly system for initiating traffic generation through straightforward button interactions.

4 EVALUATION & RESULTS

To evaluate our prototype, our evaluation strategy is composed of two tests. First, we designed an end-to-end (E2E) test to evaluate the integration of the traffic generator and the system under test. Then we ran test runs for each traffic pattern. In each test run, we logged the number of requests per second and plotted them as line diagrams using the Python library Plotly. By this we could evaluate that the patterns generated by the traffic generator indeed matched the workload patterns by Fehling et al [3]

For the system under test, we implemented a case study application that models a basic student enrollment system. The application comprised four distinct microservices, sequentially processing requests through each of these services. In each test, the traffic generator sent the generated messages to the API of the Gateway of the case study application. Figure 1 shows the case study system as a component diagram, where each microservice is represented as a single component.

The test environment included an 11th Gen Intel(R) Core(TM) i5-1135G7 CPU with a base clock speed of 2.40GHz (Turbo Boost up to 4.20GHz), 4 cores, 8 threads, and 8GiB of DDR4 RAM running at 3200 MHz.

4.1 Evaluating integration by E2E test

To test the integration we logged all messages on both ends to verify that they were sent and received in the same manner and in the same order. For this, we added a payload size of a specific size to each message. The payload size was further scrutinized to identify any alterations upon reception.

We used our periodic traffic generation to send the messages. Due to the design of our traffic generator, we did not repeat the tests for the other traffic patterns, as we logged the data right before the traffic generator sent the messages out. Since all traffic generator modules use this same endpoint to send the messages, it suffices

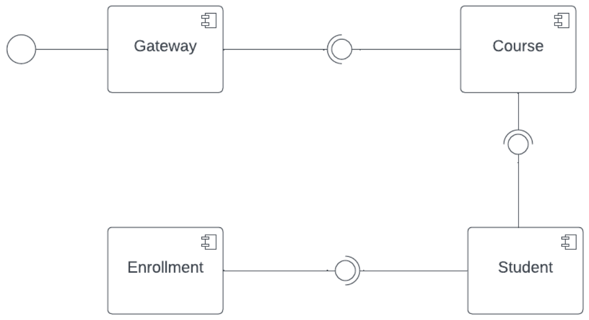


Figure 1: Component diagram of the case study system

to implement an E2E test for any of the traffic generators. Table 1 shows the parameters of the E2E test:

Parameters	Values
Payload Size Min	1
Payload Size Max	4
Min Requests Num	1
Max Requests Num	10
Total Time	60
Number of Elements on Each Side of the Wave	5
Rest Time at Lowest Peak	2
Difference Variable	3

Table 1: Parameters for periodic workload trend

In total 126 requests were sent from the traffic generator to the case study system. Throughout all 126 requests, the order in which these requests were received corresponded exactly to the order in which they were initially sent by the traffic generator. Additionally, no anomalies in the data of the sent and received messages could be observed. In conclusion, the E2E test has shown that the received messages correspond exactly to the sent messages in every way. Thus, we were able to continue our evaluation with the second evaluation approach.

4.2 Evaluating the traffic shapes

To evaluate that the generated traffic shapes conform to the desired workload shape, we manually logged the number of requests per second in the generator component at runtime. For each traffic generator, we ran multiple tests with the same test configuration to ensure that the traffic shape resulting from the test configuration could repeatedly be generated. However, if a test configuration included a certain amount of randomness in the traffic shape, we accepted deviations from other test runs and compared the overall traffic shape of the test run with one of the others, i.e. in such cases we did not enforce strict equality of the traffic shapes.

4.2.1 Test configurations. Table 2 shows the parameters for each test of traffic generation. The parameters of total time, minimum, and maximum requests are consistent across all test runs, set at 60 seconds, 1, and 10 requests, respectively. However, this range for

Traffic Patterns	**Parameters**	**Values**
Static Traffic (Straight)	Number of Requests	5
Static Traffic (Fluctuated)	Number of Requests	9
Periodic Traffic	Number of Elements on each side of the Wave	5
	Rest Time at Lowest Peak	3
	Difference Variable	2
Random Traffic	Total Data Points	20
Once-in-a-lifetime Traffic	Number of Elements on each side of the Wave	5
	Once in a Lifetime Peak Occurrence Time	50
	Once in a Lifetime Peak Occurrence Num of Request	20
Continuously Changing Traffic (straight increase)	Trend	straight increase
Continuously Changing Traffic (straight decrease)	Trend	straight decrease
Continuously Changing Traffic (exponential increase)	Trend	exponential increase
Continuously Changing Traffic (exponential decrease)	Trend	exponential decrease

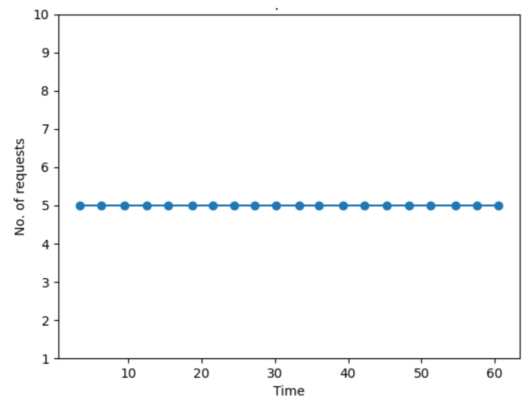
Table 2: Evaluation test configurations

minimum and maximum request values is from 1 to 20 requests for continuously changing workload patterns characterized by exponential increments and decrements. The total time is 60 seconds for the static workload test run whereas it doesn't consist of parameters related to minimum and maximum requests. Static workload trend consists of separate parameter values for straight and fluctuated types.

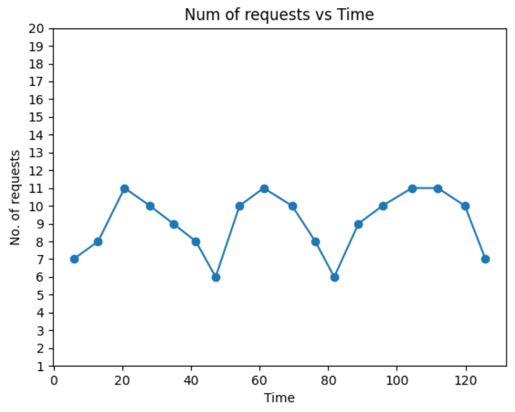
4.3 Results

For all traffic shapes, we were able to reproduce the desired shapes several times. Moreover, we did not have a test run in which the traffic shape did not have the desired shape or did not show the expected characteristics. It has to be mentioned that in the case of the random traffic generation, in contrast to all others, we were not able to reproduce the exact shape again. This lies in the nature of the random character of the random traffic generator. Thus, in the case of the random traffic generation we did not compare the traffic shapes of each test run with each other. We compared whether the outcomes of all test runs showed a random behavior, which they did.

Since all our tests have shown that the behavior of our traffic generators is reproducible, we can ensure that the test results are not false positives. Therefore, we will now present the results of a test run for each traffic pattern. Figure 2a represents a straight line for traffic shaped as a static workload pattern, which occurs when the minimum and maximum request number parameters are set to the same value. In contrast, Figure 2b illustrates a static sinusoidal wave that oscillates around a central point due to distinct parameter values.



(a) Static workload with a constant trend



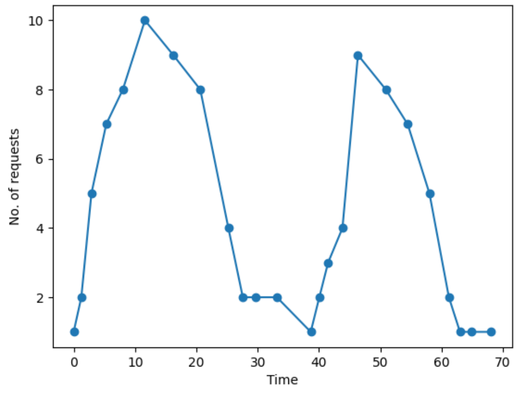
(b) Static workload with variation

Figure 2: Result for static workload

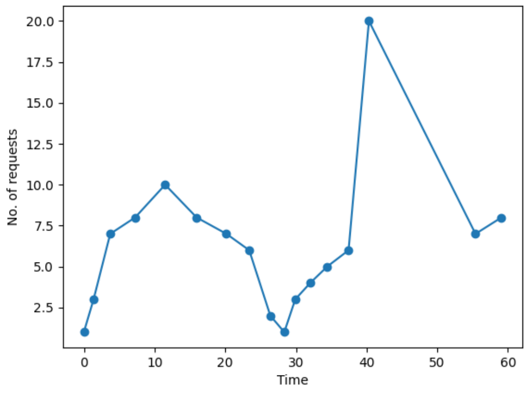
In Figure 3a the outcome of a test run of the periodic traffic generation is presented. The monitored data aptly depicts a periodic workload pattern, including deviations in the shapes conforming to human behavior, by elongating the trough of the sinusoidal curve and keeping the spectrum for maximum and minimum request numbers variable.

Figure 3b illustrates the results of a test run for the Once-in-a-lifetime traffic pattern. The diagram clearly shows an exceptionally distinct and high peak within the course of the test run. Moreover, it also shows not a complete artificial course of the number of user requests per second, as after 10 seconds there was already a small peak in the number of requests received before the exceptionally high demand of the case study system began. This reflects the human touch we developed in our

Figure 4a and 4b show the outcome of two separate test runs of the random traffic generation. In both figures, it can clearly be

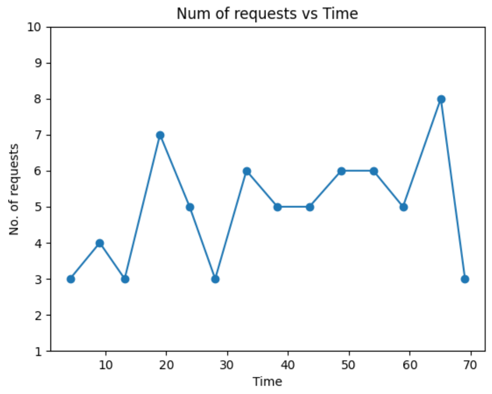


(a) Periodic workload pattern

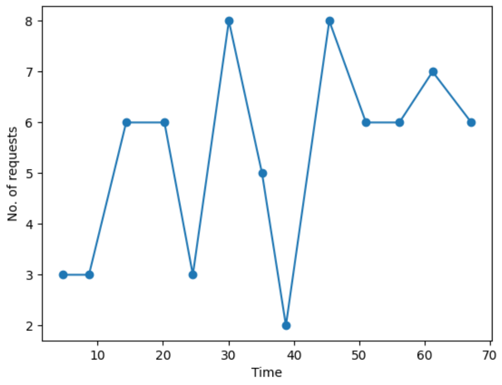


(b) Once-in-a-lifetime workload pattern

Figure 3: Result for periodic and once-in-a-lifetime workload patterns



(a) Random workload pattern - test run 1



(b) Random workload pattern - test run 2

Figure 4: Result for random workload pattern

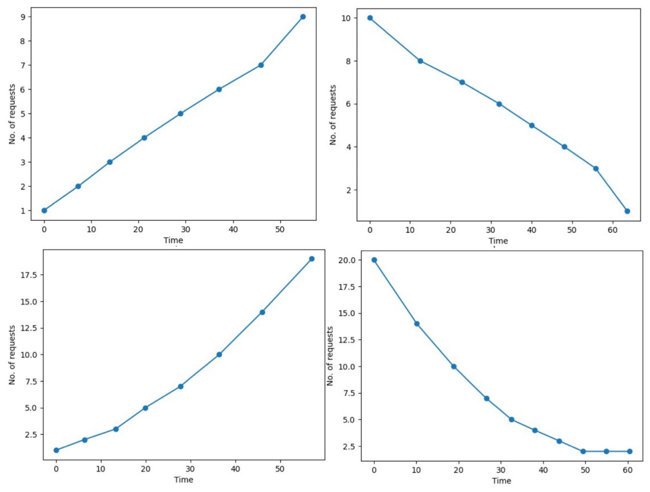


Figure 5: Continuously changing workload pattern

seen that the course of user requests per second does not follow any certain pattern and is, thus, characterized by a distribution of completely random and unpredictable numbers of user requests per second. The scenario-based traffic generator employed in this study exhibits the capability to produce four distinct categories of continuously changing workloads, characterized by both incremental and decremental patterns. The first row of Figure 5 portrays a consistent, yet linear, increase and decrease in traffic load. In contrast, the second row illustrates the manifestation of traffic exhibiting exponential growth and decrease trends.

Due to resource constraints, this scenario-based traffic generator was subjected to testing with predefined thresholds for specific parameters. These threshold values included a maximum number of requests set at 20 and a total time threshold limited to 120 seconds.

5 DISCUSSION

In addressing our first research question, our approach offers a multifaceted strategy. We replicate genuine user interactions by using synthetic traffic patterns, faithfully mimicking real user behaviors, including fluctuations corresponding to human behavior. For example, periodic workload patterns enable us to recreate scenarios like seasonal sales or product launches. This approach provides valuable insights into application performance during peak periods, enhancing the comprehensiveness of performance testing.

Furthermore, workload patterns are essential for assessing infrastructure suitability. Conducting load tests with these patterns helps organizations determine if their existing infrastructure can handle varying loads effectively. For example, simulating traffic peaks associated with once-in-a-lifetime events, such as Black Friday sales, helps ascertain whether the current infrastructure can scale to meet such demands or if additional resources are necessary. This proactive approach ensures web applications remain responsive during peak usage, ultimately improving the user experience. Workload patterns also introduce human-like behavior into synthetic traffic, adding another layer of realism.

To answer our second research question, our approach integrates behavior-related parameters into traffic shaping. Key parameters such as "Type" (straight or fluctuated) and "Number of Requests" play a pivotal role in simulating these fluctuations. Furthermore, in the case of random workload patterns, we employ probabilistic distributions such as the Poisson distribution to introduce unpredictability into request dispatch. This randomness closely mirrors scenarios where user behavior is influenced by external factors or spontaneous events, thereby replicating real-world fluctuations in user activity. This ensures that load-testing scenarios encompass the inherent uncertainties associated with human-driven interactions, providing a more accurate assessment of system performance and robustness.

6 CONCLUSION

In this paper, we presented the concept of generating synthetic traffic that corresponds to workload patterns observed in the operation of software systems in the cloud. Our approach enables performance testing of existing web applications under real conditions without the need for historical data. To answer our research questions, we demonstrated how workload patterns can be leveraged to model real-world user behavior scenarios for performance testing of web applications. We also addressed the challenge of generating synthetic traffic that includes fluctuations corresponding to human behavior. Our research has demonstrated the effectiveness and reproducibility of our scenario-based traffic generators in simulating a wide range of traffic patterns.

In our implementation, we realized a first prototype that allows for generating traffic conforming to the presented workload patterns including a human touch. We were able to evaluate the feasibility of our approach by plotting the course of user requests per second in multiple test runs per traffic pattern. Thus, the achieved flexibility in our concepts allows for a wide range of performance testing scenarios. Despite resource constraints, we employed predefined thresholds for specific parameters i.e maximum of 20 requests and total time limit of 120 seconds. This ensured that our testing approach remained robust and effective. For all tested traffic shapes, we consistently achieved the desired patterns, and we did not encounter any instances where the traffic shape deviated from our expectations.

The results of our test runs for various traffic patterns provided valuable insights. In the case of static workload patterns, we successfully replicated linear and sinusoidal shapes, illustrating the flexibility of our approach in capturing different characteristics. Periodic traffic generation accurately reflected real-world periodic workload patterns, introducing variations that mimic human behavior. The Once-in-a-lifetime traffic pattern exhibited a distinct and high peak, aligning with the demands of a case study system. The early peak in user requests highlighted the human-like unpredictability that we aimed to incorporate into our approach. Additionally, the random traffic generation yielded entirely unpredictable patterns, simulating scenarios where user behavior is influenced by external factors, thus introducing realism into the testing process. Our scenario-based traffic generator successfully created four distinct categories of continuously changing workloads, including incremental and decremental patterns. The ability to replicate these various traffic patterns has significant implications for performance testing and infrastructure assessment.

6.1 Future Work

Our proof of concept implementation showed that our concepts work. However, the applicability of it is limited, due to the generic messages being created. To leverage our concepts for meaningful performance testing arbitrary web applications, the concepts of message generation must be extended. We see one way to do this in an HTTP message configuration component that provides a set of HTTP message stereotypes, such as messages with an image or video payload to upload or authenticated messages, as well as a set of configuration parameters to configure messages tailored to the APIs of the system under test.

Another avenue for future work lies in the prospect of conducting system tests within a production environment, specifically on a server infrastructure. This extension would enable comprehensive test runs using larger values of the number of requests over extended time spans. Such an endeavor holds promise for evaluating the performance of the traffic generator in real-world conditions and fine-tuning any potential delays in traffic generation processes. This valuable step can enhance the reliability and efficiency of the traffic generation system, further strengthening its applicability and practicality. Furthermore, another area for future research involves the development and testing of a more intricate case study-based system. This advanced system would be intentionally designed to assess its resilience under specific scenarios, to identify vulnerabilities or critical failure points. The creation of such a complex and challenging testbed could offer deeper insights into the performance and robustness of traffic generation techniques under adverse conditions.

Given the growing challenges web applications encounter in maintaining availability, resilience, and responsiveness, a valuable future work can the extension of our concepts by the concepts of resilient vector consensus to design test scenarios that protect web applications from malicious attacks and failures [12]. By incorporating the extended concepts for mitigating mobile malicious attacks, web applications can undergo more extensive testing. Furthermore, future research may explore innovative techniques to safeguard web applications, ensuring they remain robust even in the presence of security threats and disruptions.

ACKNOWLEDGMENTS

The author would like to thank the Research Group Software Construction at RWTH Aachen University, Aachen, Germany and The

Sirindhorn International Thai-German Graduate School of Engineering King Mongkut's University of Technology North Bangkok, Bangkok, Thailand.

REFERENCES

[1] 2023. "Collins English dictionary (2023)". https://www.collinsdictionary.com/english/pattern. [Online; accessed 04-September-2023].

[2] P. Barford and M. Crovella. 1998. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. 151–160.

[3] Christoph Fehling, Frank Leymann, Ralph Retter, Walter Schupeck, and Peter Arbitter. 2014. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer. https://doi.org/10.1007/978-3-7091-1568-8

[4] Devndra Ghimire. 2020. Comparative study on Python web frameworks: Flask and Django. (2020).

[5] Guoliang Jin, Linhai Song, Xiaoming Shi, Joel Scherpelz, and Shan Lu. 2012. Understanding and detecting real-world performance bugs. *ACM SIGPLAN Notices* 47, 6 (2012), 77–88.

[6] Peter Mell, Tim Grance, et al. 2011. The NIST definition of cloud computing. *Special Publication 800-145* (2011), 1–7.

[7] Ian Molyneaux. 2014. *The art of application performance testing: from strategy to tools*. " O'Reilly Media, Inc.".

[8] D. Mutz, G. Vigna, and R. Kemmerer. 2003. An experience developing an IDS stimulator for the black-box testing of network intrusion detection systems. In *19th Annual Computer Security Applications Conference, 2003*. 374–383.

[9] Sam Newman. 2021. *Building microservices*. " O'Reilly Media, Inc.".

[10] Chris Richardson. 2018. *Microservices patterns: with examples in Java*. Simon and Schuster.

[11] Vaskaran Sarcar. 2016. *Java Design Patterns*. https://doi.org/10.1007/978-1-4842-1802-0

[12] Yilun Shang. 2023. Resilient Vector Consensus Over Random Dynamic Networks Under Mobile Malicious Attacks. *Comput. J.* (2023), bxad043.

[13] J. Sommers, V. Yegneswaran, and P. Barford. 2004. A framework for malicious workload generation. In *Proceedings of the 2004 ACM SIGCOMM Internet Measurement Conference, IMC 2004*. 10.

[14] W. Tang, Y. Fu, L. Cherkasova, and A. Vahdat. 2003. Medisyn: A synthetic streaming media service workload generator. In *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*. 12–21.

[15] Shengzhe Xu, Manish Marwah, Martin Arlitt, and Naren Ramakrishnan. 2021. Stan: Synthetic network traffic generation with generative neural models. In *Deployable Machine Learning for Security Defense: Second International Workshop, MLHat 2021, Virtual Event, August 15, 2021, Proceedings 2*. Springer, 3–29.

[16] J. Yin, X. Lu, X. Zhao, H. Chen, and X. Liu. 2014. Burse: A bursty and self-similar workload generator for cloud computing. *IEEE Transactions on Parallel and Distributed Systems* 26, 3 (2014), 668–680.