Name: Tural Mehtiyev
Project Title: Scalability experiment of microservice architecture on an online bookstore application
Date: 03.07.2023

To measure the effect of resource scaling on a microservice application's performance, I will employ several measurement strategies.

Load Testing:

Characteristics: Simulation of real-world loads and measure the system's response.
Variables: Concurrent users/requests, response time, throughput, CPU and memory usage.
Roles:

- **Concurrent users/requests**: Represents the number of users or requests accessing the system simultaneously.
- **Response time**: Measures the time taken for the system to respond to a request.
- **Throughput**: Indicates the number of requests the system can handle per unit of time.

Key Assumption: The load test environment closely resembles the production environment, and the user behavior is representative.

Expected Outcome: Increasing the number of concurrent users/requests will negatively impact response time and throughput.

- **Null Hypothesis**: Increasing the number of concurrent users/requests does not impact response time and throughput.

- **Alternative Hypothesis**: Increasing the number of concurrent users/requests does affect impact response time and throughput.

Identification of bottlenecks

To identify the main bottleneck in the overall application, a manual approach will be adopted whereby each application will be individually scaled and the response rate of the entire application will be measured. This methodology is similar to the one described in [1], which can be referenced for further details.

The dataset used in [1] consists of 4,483,537 rows (requests) and 9 features, with the primary focus being the **request response time**. To gain insights into the distribution of response times within the dataset, histograms were utilized as visualizations. These visualizations provide a clear understanding of how the response times vary across different scenarios.

For instance, Figure 1 depicts the response time distribution when App 2 is scaled three times, showing an average response time of 17.7 seconds. Similarly, Figure 2 illustrates the response time distribution when App 3 is scaled three.

distribution when all Apps are scaled three times, indicating an average response time of 10.9 seconds.

These visualizations aid in comprehending the variations in response times across different scaling scenarios, thereby providing valuable insights for further analysis and interpretation.

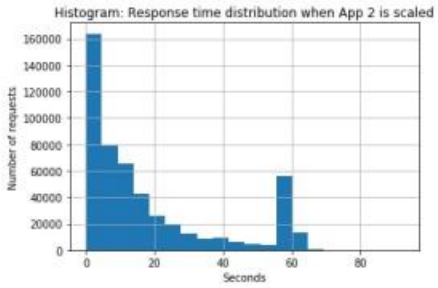


Fig. 1: Histogram: Distribution across requests where App 2 was scaled up

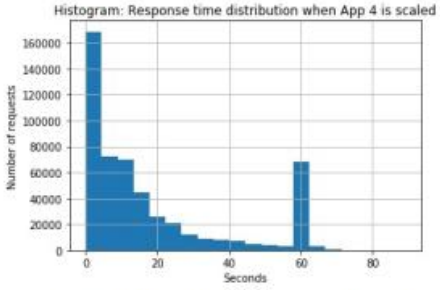


Fig. 3: Histogram: Distribution across requests where App 4 was scaled up

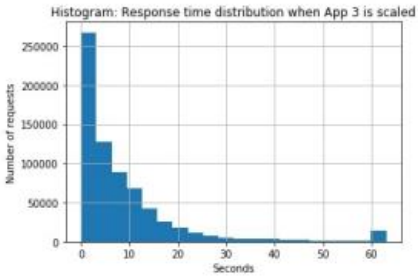


Fig. 2: Histogram: Distribution across requests where App 3 was scaled up

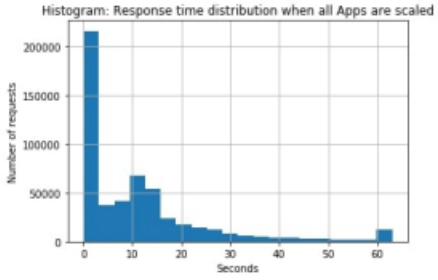


Fig. 4: Histogram: Distribution across requests where all Apps were scaled up

Based on the provided experiment results in paper [1], the following observations can be made:

- From the analysis of the visualizations, it appears that App 3 is a significant bottleneck within the microservice. This can be inferred from the fact that scaling App 3 leads to a mean response time reduction of over 50%, and the distribution of response times clearly skews towards the lower range, with a noticeable decrease in the number of requests falling within the 60-second range.

- When scaling the other Apps without scaling App 3, the result is an increased response time. Interestingly, the average request response time when all Apps were scaled is actually higher compared to when only App 3 is scaled.

- This observation could be attributed to the scarcity of bare metal resources available to the containers of App 3, which can be attributed to overcrowding on the virtual machine (VM).

Like the study mentioned as a reference, my research also involves conducting a similar experiment (including visualization and storytelling) to identify any potential bottleneck within my application.

Scalability Testing:

Characteristics: Assess the system's ability to handle increased workloads with growing resources.
Variables: Workload, scalability indicators (e.g., response time, throughput).
Roles:

- Workload: Represents the amount of work imposed on the system.
- System resources: Measure the allocation and utilization of resources necessary for the system's operation.
- Scalability indicators: Quantify the system's performance, such as response time and throughput.

Test Scenario

- Scenario 1: Single Node Configuration

  Microservice 1: 1 instance
  Microservice 2: 1 instance
  Microservice 3 (Bottleneck): 1 instance

- Scenario 2: Multiple Node Configuration (Scaling the Bottleneck Microservice)

  Microservice 1: 1 instance
  Microservice 2: 1 instance
  Microservice 3 (Bottleneck): 2 or more instances (scaled)

Design of Load Tests

Below is an initial draft outline of the load tests that will be performed using Apache JMeter:

Test 1: Baseline Performance Test (Low Load)

- ✓ Concurrent Users: 50
- ✓ Ramp-up Period: 10 seconds
- ✓ Test Duration: 5 minutes

Test 2: Scalability Test (Medium Load)

- ✓ Concurrent Users: 100
- ✓ Ramp-up Period: 10 seconds
- ✓ Test Duration: 5 minutes
- ✓

Test 3: Scalability Test (High Load)

- ✓ Concurrent Users: 200
- ✓ Ramp-up Period: 5 seconds
- ✓ Test Duration: 10 minutes

Note: The given all mentioned cases above will be applied to both scenario (with and without scaled versions)

Data Normality Assurance:

The principles of the Central Limit Theorem (CLT) and the Law of Large Numbers (LLN) will be applied in my experiment to achieve normalization with the following steps.

- **Determining the number of samples and the sample size:**

  I will have at least 50 observations in each sample with total 100 samples for each scenario.

- **Collection of Sample Data:**

  I will perform multiple runs experiment for each experiment scenario, where each run represents a sample. For example, I will collect response time and throughput data for each samples in each scenario.

- **Calculation of Sample Means:**

  I will calculate the mean of each sample (e.g., average response time and throughput for each sample) for 2 scenarios. These sample means will serve as data points for normalization. As a result, I will have a distribution of sample means which will guarantee the normal distribution.

Expected Outcome: Increasing system resources (e.g., adding more instances, nodes, or containers) will result in improved performance.

- **Null Hypothesis**: Increasing system resources does not affect scalability, resulting in the same response time and throughput.
- **Alternative Hypothesis**: Increasing system resources does affect system's scalability, resulting in the improved response time and increased throughput.

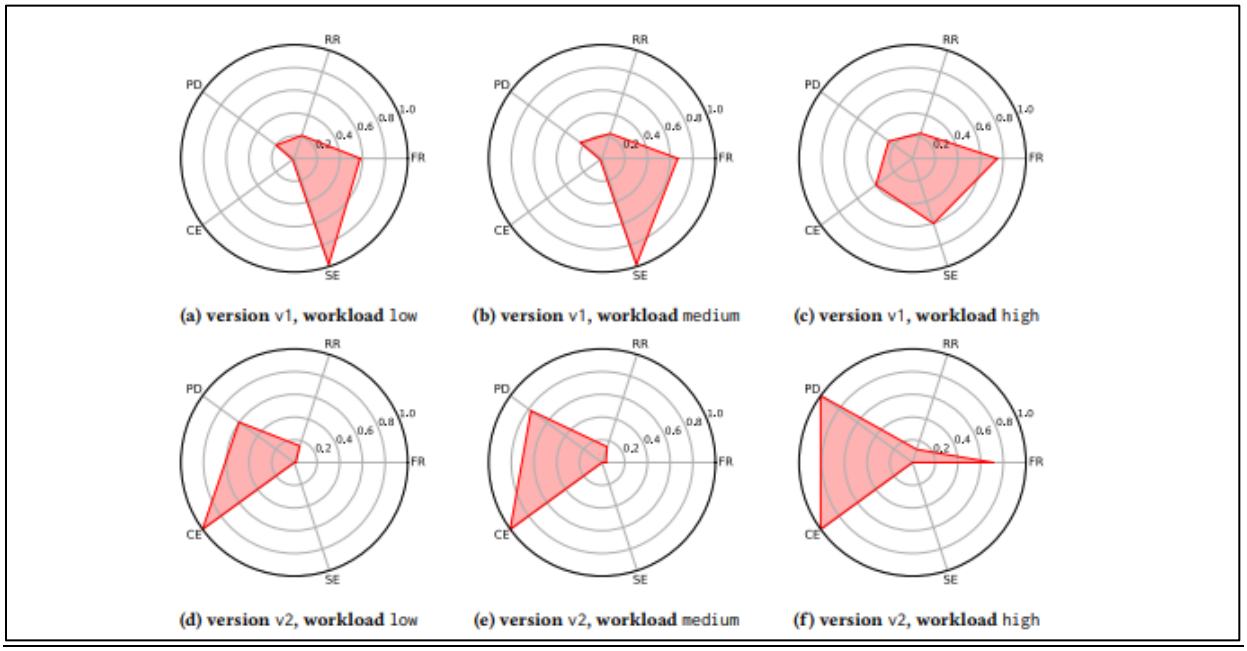
Accepting or Rejecting Hypothesis:

To determine whether to accept or reject your hypothesis, compare the performance metrics obtained from different test scenarios (e.g., with varying numbers of instances or resources) will be compared and the collected data will be assessed if there is a noticeable improvement in performance and throughput as the resources scale. Statistical analysis techniques such as t-tests will be used to evaluate the significance of the results and determine if they support your hypothesis.

Draft Visualization Ideas:

To effectively present the findings of my experiment, I can utilize radar plots as a visualization tool for each metric. The concept of radar plots, as demonstrated in a sample idea from a referenced paper [2], can be applied to my research paper. Specifically, the three radar plots above represent the load results for test scenario 1, while the three radar plots below represent the load scenarios (low, mid, high) for test scenario 2, which includes a scaled version.

Within these radar plots, I can showcase various metrics such as response time, throughput, resource utilization, and failure rate. By using radar plots, I can visually illustrate the performance characteristics of the different load scenarios and compare the metrics across the test scenarios. This approach allows for a comprehensive understanding of the experiment's outcomes and facilitates effective communication of the results to readers.



(a) **version** v1, **workload** low (b) **version** v1, **workload** medium (c) **version** v1, **workload** high

(d) **version** v2, **workload** low (e) **version** v2, **workload** medium (f) **version** v2, **workload** high

References:

1. Cruz Coulson, N., Sotiriadis, S., & Bessis, N. (2020). Adaptive microservice scaling for elastic applications. *IEEE Internet of Things Journal*, *7*(5), 4195–4202. https://doi.org/10.1109/jiot.2020.2964405

2. Camilli, M., Guerriero, A., Janes, A., Russo, B., & Russo, S. (2022). Microservices integrated performance and reliability testing. *Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test*. https://doi.org/10.1145/3524481.3527233