Report on Faast-A-Faas-Framework

Arijit Saha (210050017) and Aryan Mathe (210050021)

1 Introduction

Faast-A-Faas-Framework is a project designed to evaluate and compare different cluster configurations for Function as a Service (FaaS) platforms. The goal is to analyze performance metrics such as response time and resource utilization across various setups. This report provides an overview of the project's approach, including the types of clusters tested, the workloads used for evaluation, and instructions for setting up and running experiments. By offering insights into the impact of different configurations, this project aims to guide best practices for optimizing serverless infrastructure

2 Cluster Configurations

The project considers the following cluster configurations:

2.1 Single Pod Cluster

This configuration consists of a single pod with a single container deployed on a single-node cluster. The pod hosts the FaaS service, and all incoming requests are directed to the single container within the pod. This simple setup serves as a baseline for comparison with other configurations.

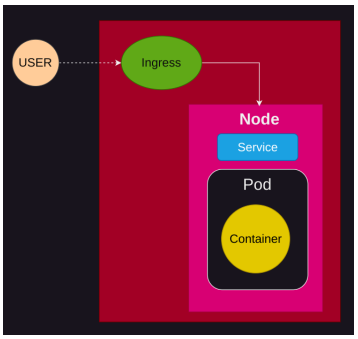

Figure 1: Single Pod Cluster

2.2 Single Pod with Multi-Container

In this configuration, a single pod contains multiple containers that run the FaaS service on a single-node cluster. An Nginx load balancer pod is also deployed to route incoming requests to the containers within the pod. This setup allows testing the efficiency of using multiple containers within a single pod.

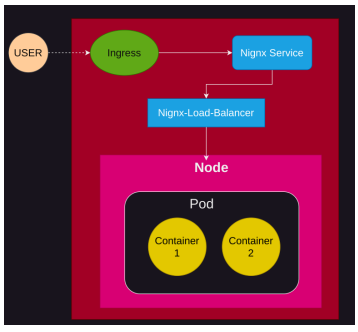

Figure 2: Single Pod with Multi-Container

2.3 Multi-Pod with Single Node

This configuration includes multiple pods, each with one container running the FaaS service, deployed on a single node. An Nginx load balancer pod manages incoming requests, routing them to the appropriate FaaS service pods. This setup tests how multiple pods interact within a single node and the impact of load balancing.

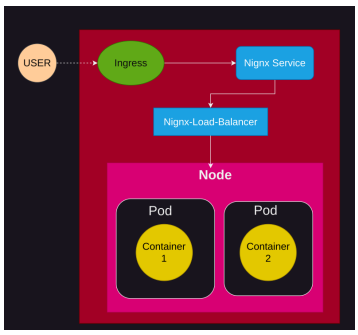

Figure 3: Multi-Pod with Single Node

2.4 Multi-Pod with Multi-Node

In this configuration, multiple pods are distributed across two different nodes, with each pod containing one container running the FaaS service. An Nginx load balancer pod manages incoming requests, distributing them evenly across the different nodes and their pods. This setup tests how the system performs when multiple pods are deployed across multiple nodes.

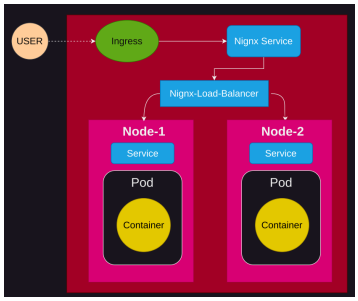

Figure 4: Multi-Pod with Multi-Node

2.5 Horizontal Pod Autoscaler (HPA)

The Horizontal Pod Autoscaler (HPA) is a Kubernetes feature that automatically scales the number of pods in response to changes in resource utilization or other metrics. In this configuration, multiple pods (initially one) are deployed on a single node. The HPA monitors resource usage (such as CPU or memory) and dynamically adjusts the number of pods based on predefined thresholds, ensuring efficient resource usage and consistent performance.

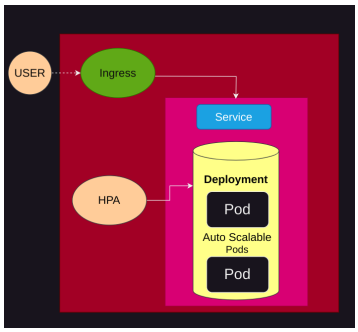

Figure 5: Horizontal Pod Autoscaler

2.6 Vertical Pod Autoscaler (VPA)

The Vertical Pod Autoscaler (VPA) is a Kubernetes feature that adjusts the resource limits and requests of a pod based on its actual usage. In this configuration, a single pod is deployed with VPA enabled in a single-node cluster. The VPA monitors the pod's resource usage and dynamically adjusts its resource requests and limits to ensure optimal performance and resource efficiency.

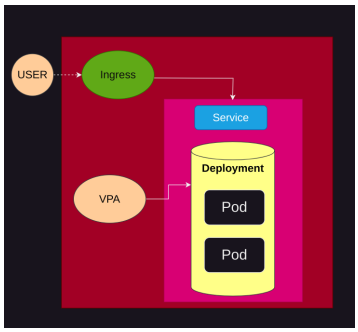

Figure 6: Vertical Pod Autoscaler

3 Metrics

This section discusses the metrics used to evaluate the performance and resource utilization of the Function as a Service (FaaS) platform across different cluster configurations. The two main types of workloads tested were a simple loop and a large Lorem Ipsum generator.

- **Responce Time:** A Python script was used to measure response time in three phases: Sends 500 individual requests one at a time and records the average response time. Sends 500 groups of 10 simultaneous requests and records the average response time for each group. Sends 500 groups of 50 simultaneous requests and records the average response time for each group. This approach allows us to evaluate how each configuration performs under different levels of request concurrency, providing insights into the scalability and efficiency of the system.

3.1 Resource Utilization (CPU and Memory)

Resource utilization is another important aspect of evaluating FaaS services:

- **CPU Usage:** The amount of CPU resources consumed by the FaaS service. Lower CPU usage is preferred as it indicates efficient use of processing power.
- **Memory Usage:** The amount of memory consumed by the FaaS service. Lower memory usage is desirable as it implies efficient use of available memory resources.

Resource usage was measured using the Kubernetes 'metrics-server' API, which provides real-time data on the CPU and memory usage of pods and containers within the cluster. This data was collected and analyzed for each cluster configuration.

4 Requirements

To run the experiments, the following tools need to be installed:

- docker
- kubectl
- minikube
- helm

5 Running Instructions for Setting up a Cluster Environment

To set up the environment for running cluster configurations, run the following command:

```
bash setup.sh
```

The supported app-types based on the cluster configurations defined above are:

```
single-pod  two-pod-same-node  two-pod-diff-node  hpa  vpa  two-container
```

```
bash deploy_app.sh <app_name> <app_type> <docker_image_name> <python-app-file> <requirements-file> <port> <map_url>
```

6 Generating Analysis Results

To generate analysis results, follow these steps:

1. Set up the 'metrics-server' REST-API by running the following command in two different terminal windows:

   ```
   minikube dashboard --port=20000
   ```

2. Run the following command to perform the analysis for different cluster configurations and generate logs for response-time and resource utilization:

   ```
   bash src/analysis/perform_analysis.sh <host> <url> <app-type> <app-name>
   ```

   This will generate logs for response-time and resource utilization based on the defined workload. The file names will be in the following format:

   ```
   <logs-dir>/<app-name>-<app-type>-response_time.csv
   <logs-dir>/<app-name>-<app-type>-resource_usage.csv
   ```

3. Execute the following Python file to generate plots for the analysis:

```
   python3 analysis/get_plot_from_log.py
   usage: script to generate plot from log files [-h] --app-type APP_TYPE
                                                 [--response-log RESPONSE_LOG]
                                                 [--resources-log RESOURCES_LOG] --output-
folder
                                                 OUTPUT_FOLDER --app-name APP_NAME

   options:
     -h, --help            show this help message and exit
     --app-type APP_TYPE   single-pod/two-pod-same-node/two-pod-diff-node/hpa/vpa/two-
container
     --response-log RESPONSE_LOG
     --resources-log RESOURCES_LOG
     --output-folder OUTPUT_FOLDER
     --app-name APP_NAME
```

7 Analysis

7.1 HPA

7.1.1 HPA cpu-utilization

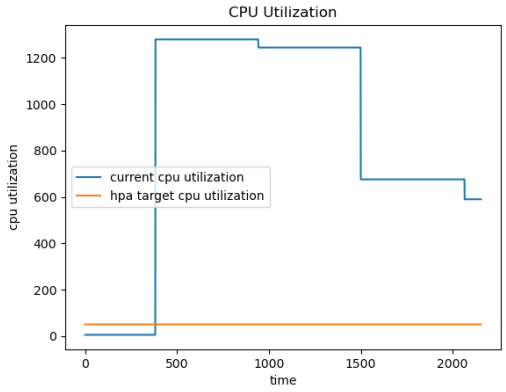

Figure 7: Loop

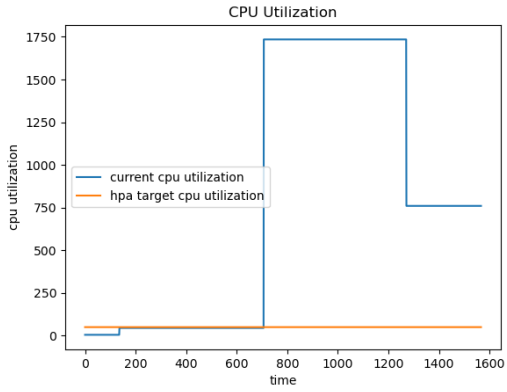

Figure 8: Lorem

- Initial CPU utilization for the HPA configuration is very high due to the limited number of pods available at the start.
- As the HPA scales out and increases the number of pods, the workload is distributed across more pods.
- Consequently, the CPU utilization per pod decreases over time as more pods are added to handle incoming requests.
- This adaptive scaling helps maintain performance while balancing resource usage efficiently.

Below is the replica count graph that illustrates the increase in the number of pods over time, supporting the trend of decreasing CPU utilization per pod:

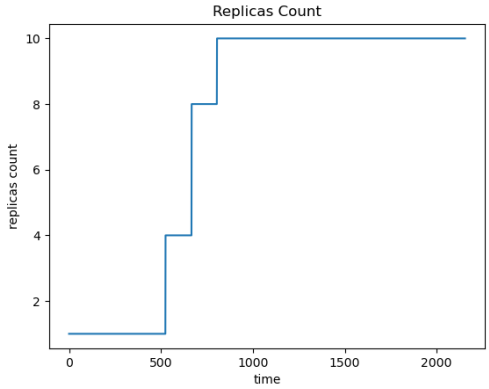

Figure 9: Loop

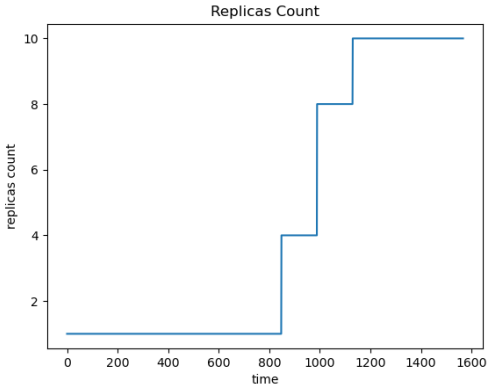

Figure 10: Lorem

7.1.2 HPA Response Time Observations

- Initial response times in the HPA configuration may be higher due to limited pod availability at the start.
- As HPA scales out the number of pods in response to increased load, response times start to decrease.
- The additional pods allow the service to handle more requests concurrently, resulting in improved response times.
- HPA helps maintain consistent response times even as demand fluctuates, by adjusting the number of pods to match the workload.

Below is the response time for HPA.

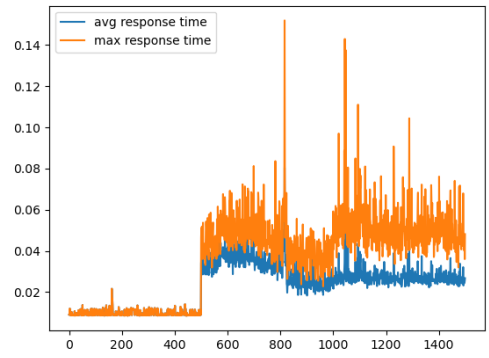

Figure 11: Loop

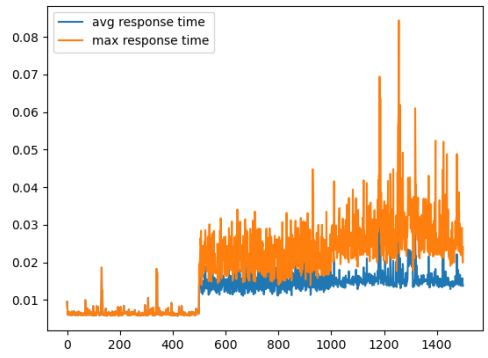

Figure 12: Lorem

7.2 VPA

7.2.1 VPA CPU Utilization

- VPA initially allocates CPU resources based on current requirements but can increase or decrease them dynamically.

- As demand changes, VPA adjusts CPU allocations, potentially causing spikes or dips in utilization as resources are adapted.
- These adjustments aim to maintain efficient resource usage and consistent application performance.

Below is the graph that illustrates the adjustments in resource allocation for VPA:

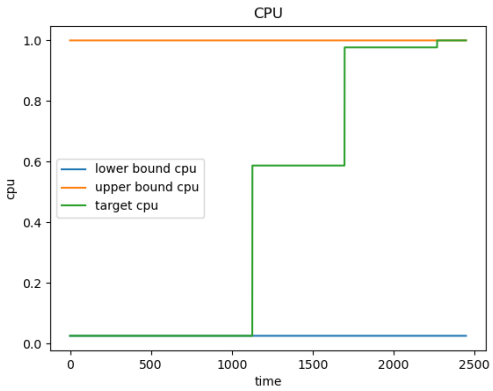

Figure 13: Loop

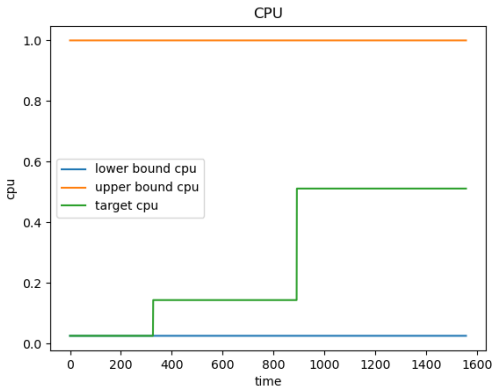

Figure 14: Lorem

7.2.2 VPA Response Time Observations

- Response times in the VPA configuration can vary as resources are adjusted to meet demand.
- Dynamic adjustments in pod resources can lead to fluctuations in response times.
- As the VPA optimizes CPU allocations, response times may stabilize, especially when demand is consistent.
- VPA aims to balance resource utilization with consistent response times, adapting to varying workloads.

Below are the response time graphs for VPA:

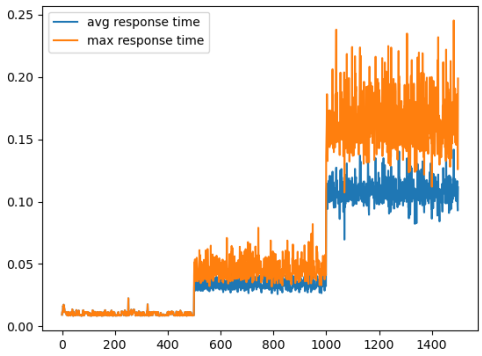

Figure 15: Loop

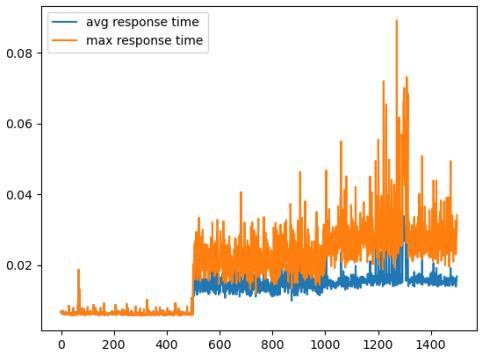

Figure 16: Lorem

7.2.3 VPA Memory Utilization

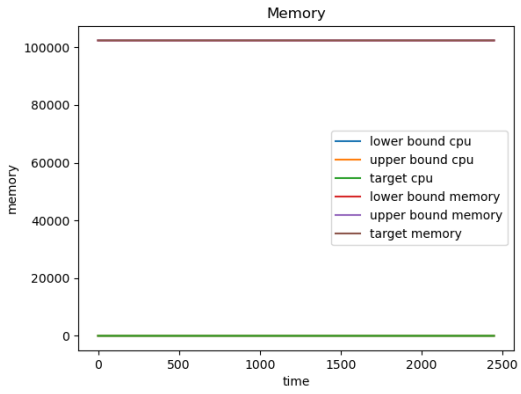

Figure 17: Loop

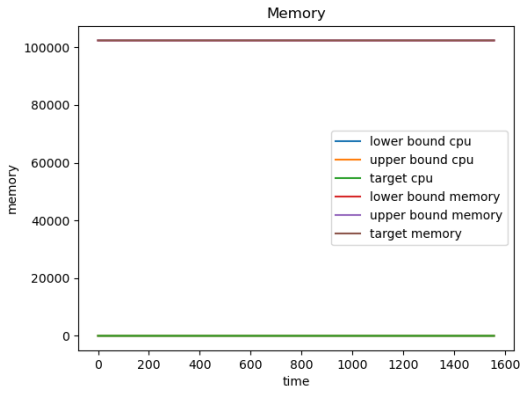

Figure 18: Lorem

- VPA manages memory allocations dynamically, adjusting resources according to application needs.
- Initial memory usage may vary based on initial resource requests set by VPA.
- As the VPA adjusts memory allocations, memory utilization stabilizes according to application requirements.
- This adaptive approach optimizes memory usage and aims to maintain consistent application performance.

7.3 Single Pod

7.3.1 Single Pod CPU Utilization

- CPU utilization in the Single Pod configuration can be relatively stable, as all workload is handled by a single pod.
- The pod may experience higher CPU usage during peak demand since it manages all requests.
- Over time, as workloads fluctuate, CPU usage may stabilize as the single pod adjusts to the demand.

Below is the graph that illustrates the CPU utilization for Single Pod:

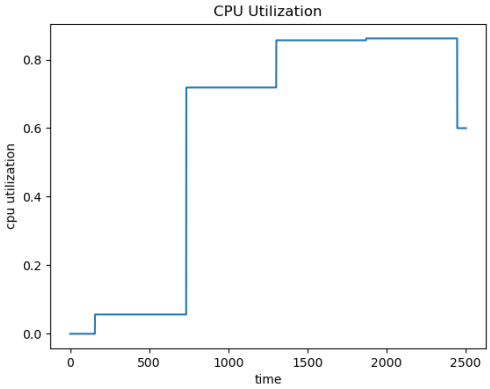

Figure 19: Loop

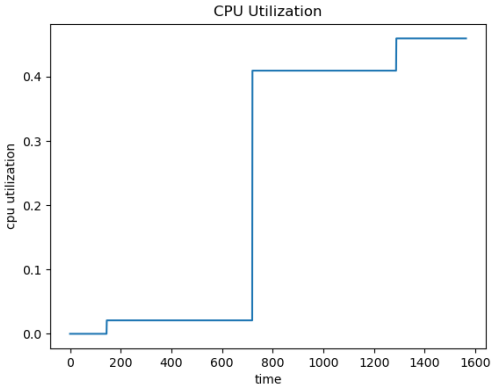

Figure 20: Lorem

7.3.2 Single Pod Response Time Observations

- Response times in the Single Pod configuration can be consistent, as a single pod handles all requests.
- Under heavy load, response times may increase due to the single pod being responsible for all tasks.
- As workloads stabilize, response times tend to remain steady, provided the pod is not overwhelmed.

Below are the response time graphs for Single Pod:

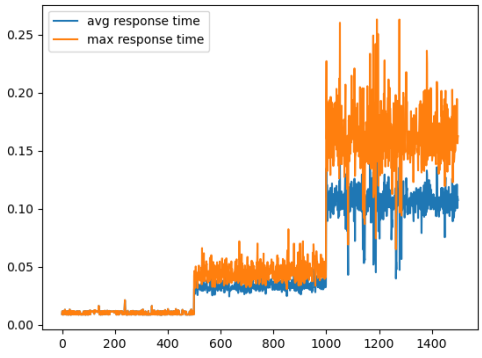

Figure 21: Loop

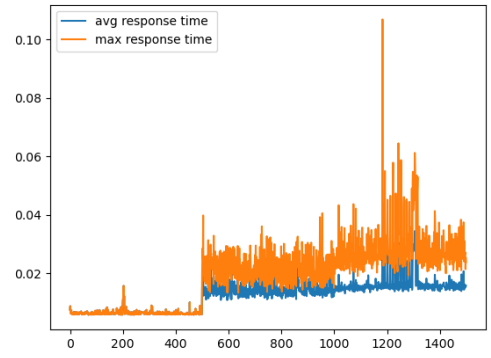

Figure 22: Lorem

7.3.3 Single Pod Memory Utilization

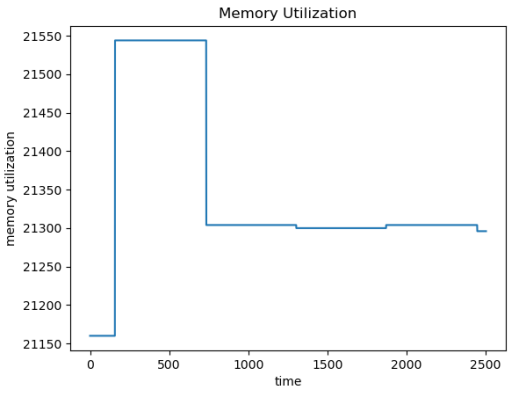

Figure 23: Loop

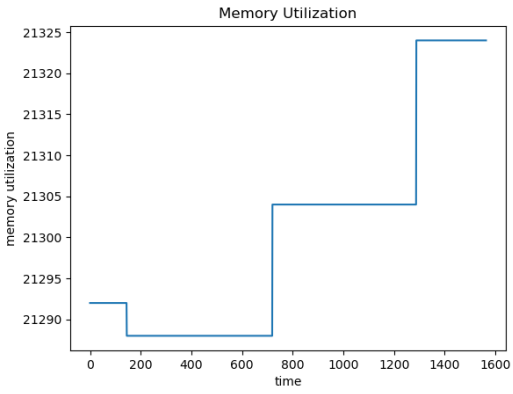

Figure 24: Lorem

- Memory utilization in the Single Pod configuration can vary based on the complexity of the workloads.
- The single pod manages memory allocation for all requests, which can lead to fluctuations in memory usage.
- As workloads stabilize, memory utilization tends to even out, provided the single pod is adequately resourced.

7.4 Two-Container

7.4.1 Two-Container CPU Utilization

- CPU utilization in the two-container configuration may vary depending on the distribution of workload between the containers.
- Both containers can handle requests, potentially balancing the load and leading to lower overall CPU utilization.
- Under peak demand, CPU usage may increase as both containers manage incoming requests simultaneously.

Below is the graph that illustrates the CPU utilization for two-container configuration:

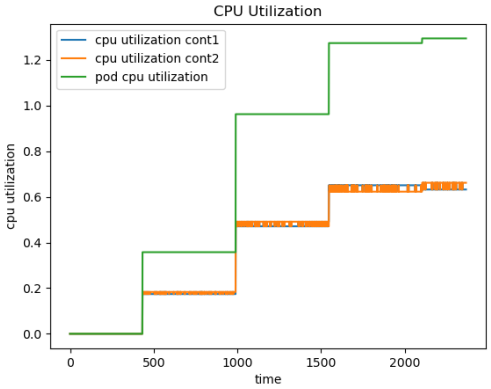

Figure 25: Loop

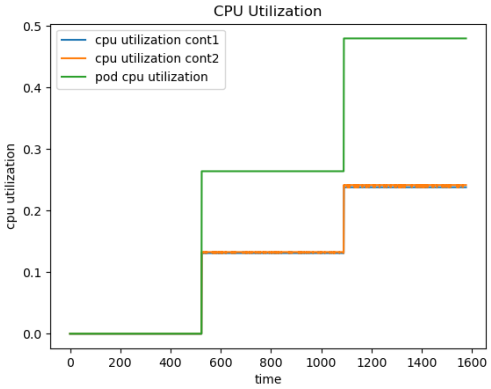

Figure 26: Lorem

7.4.2 Two-Container Response Time Observations

- Response times in the two-container configuration can be lower, as multiple containers share the workload.
- If one container becomes overwhelmed, the other can handle the load, potentially balancing response times.
- During peak demand, response times may fluctuate depending on the distribution of requests between the containers.

Below are the response time graphs for two-container configuration:

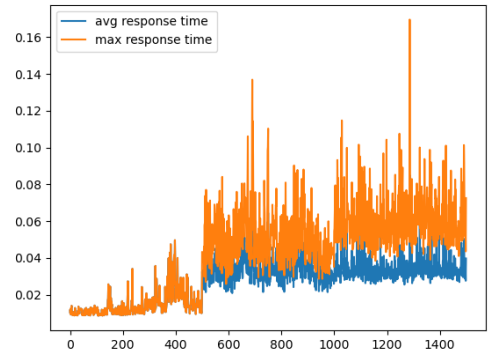

Figure 27: Loop

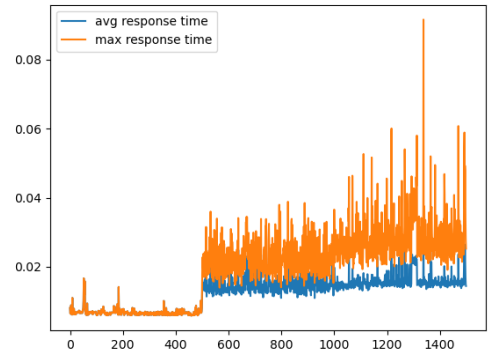

Figure 28: Lorem

7.4.3 Two-Container Memory Utilization

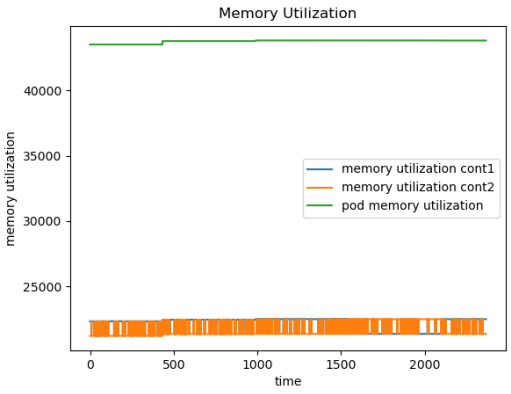

Figure 29: Loop

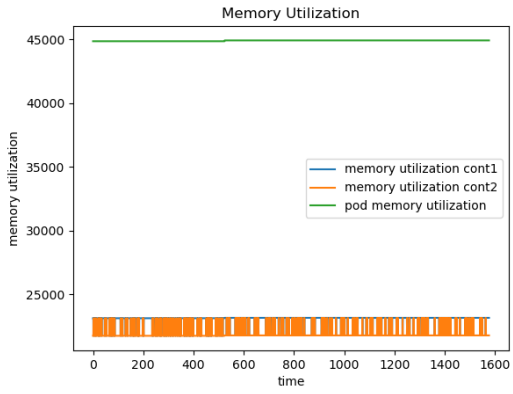

Figure 30: Lorem

- Memory utilization in the two-container configuration can vary depending on the memory demands of the workloads.
- Each container manages its own memory allocation, which may lead to more efficient usage when workloads are balanced.
- As workloads stabilize, memory utilization should even out as both containers manage their respective resources.

7.5 Two Pods Same Node

7.5.1 Two Pods Same Node CPU Utilization

- In the two-pods-same-node configuration, CPU utilization may be more stable compared to the single pod setup, as the workload is distributed across two pods.
- With two pods handling incoming requests, the load is shared, potentially reducing the CPU usage of each pod.
- During peak demand, CPU usage may increase as both pods manage concurrent requests on the same node.

Below is the graph that illustrates the CPU utilization for the two-pods-same-node configuration:

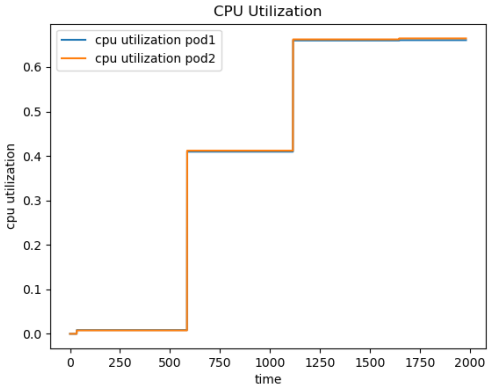

Figure 31: Loop

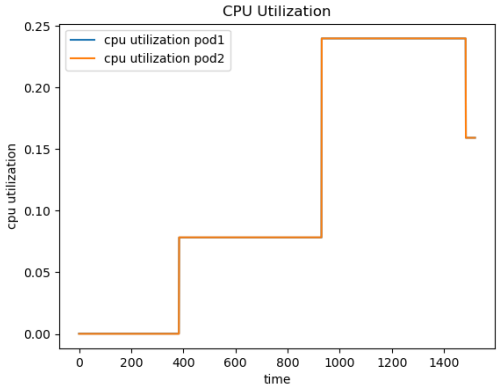

Figure 32: Lorem

7.5.2 Two Pods Same Node Response Time Observations

- Response times in the two-pods-same-node configuration can be consistent, as two pods share the workload.
- If one pod is under heavy load, the other can manage additional requests, balancing response times.
- This setup helps maintain consistent response times as the workload is distributed across both pods.

Below are the response time graphs for the two-pods-same-node configuration:

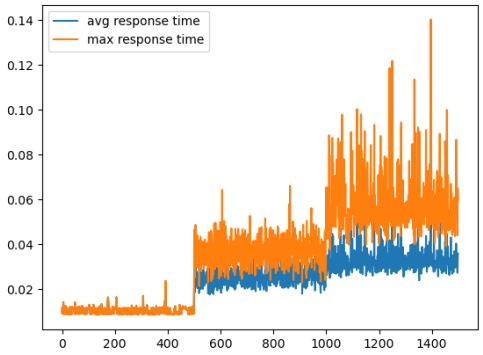

Figure 33: Loop

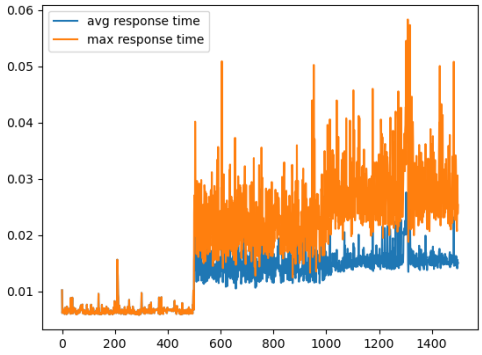

Figure 34: Lorem

7.5.3 Two Pods Same Node Memory Utilization

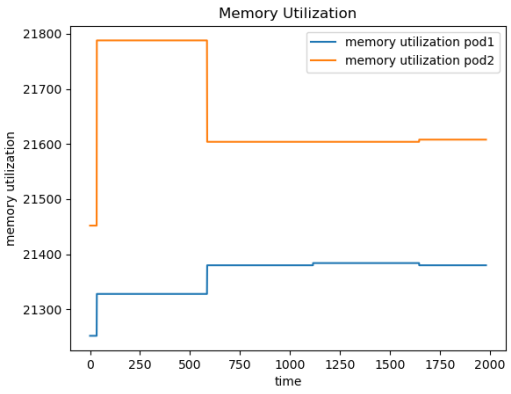

Figure 35: Loop

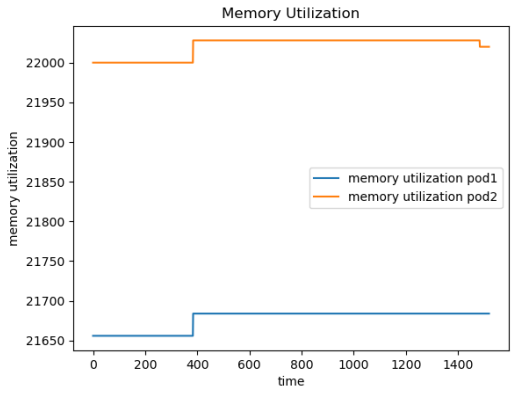

Figure 36: Lorem

- Memory utilization in the two-pods-same-node configuration can vary based on the workloads and the memory requirements of each pod.
- Since two pods share the same node, memory usage might be affected by how each pod manages its resources.
- As workloads stabilize, memory utilization tends to balance out as both pods handle their own resource management.

7.6 Two Pods Different Node

7.6.1 Two Pods Different Node CPU Utilization

- In the two-pod-diff-node configuration, CPU utilization may vary depending on the distribution of workload across the two pods located on different nodes.
- Since each pod is on a different node, the workload is distributed evenly, potentially balancing the overall CPU usage across nodes.
- CPU utilization may vary as each pod handles its respective requests, but overall utilization may stabilize over time.

Below is the graph that illustrates the CPU utilization for the two-pod-diff-node configuration:

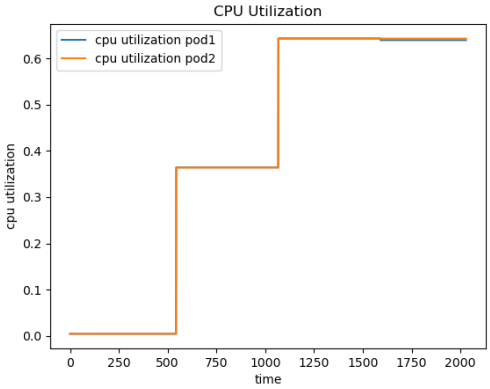

Figure 37: Loop

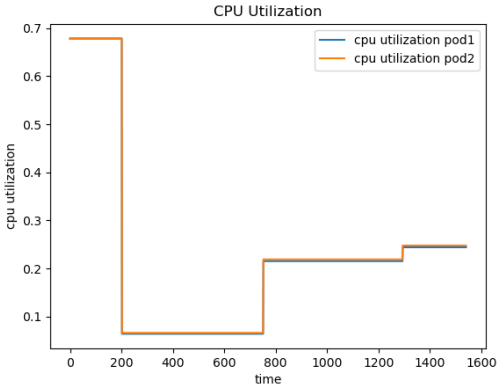

Figure 38: Lorem

7.6.2 Two Pods Different Node Response Time Observations

- Response times in the two-pod-diff-node configuration may be consistent due to the even distribution of workload across the two pods on different nodes.
- The setup allows each pod to handle requests independently, possibly balancing the load and resulting in more consistent response times.
- Under peak demand, response times may increase if either pod becomes overwhelmed, but overall, the configuration tends to provide more consistent response times.

Below are the response time graphs for the two-pod-diff-node configuration:

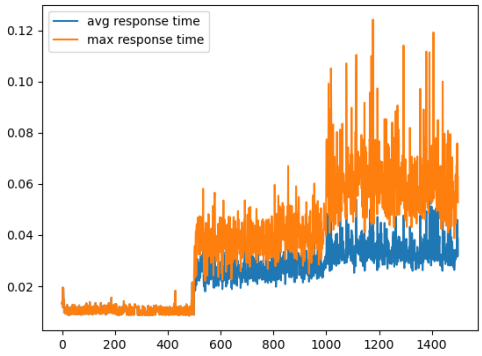

Figure 39: Loop

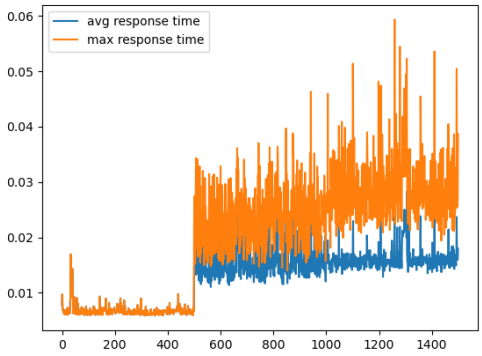

Figure 40: Lorem

7.6.3 Two Pods Different Node Memory Utilization

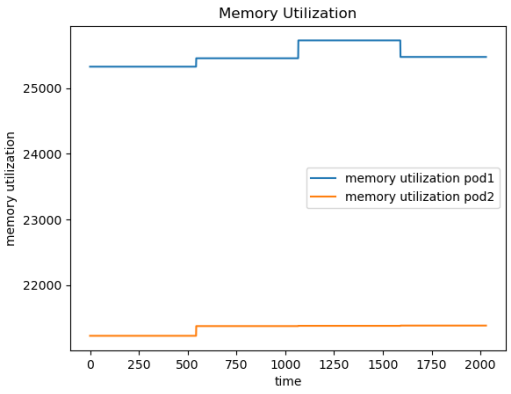

Figure 41: Loop

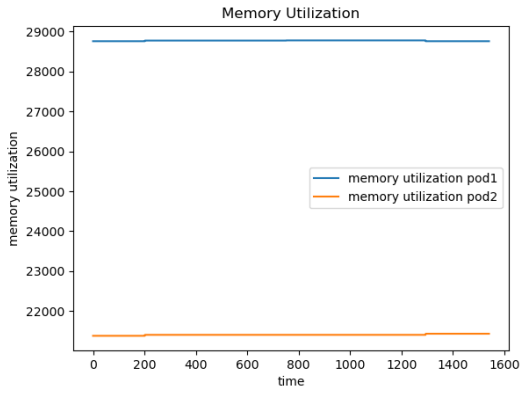

Figure 42: Lorem

- Memory utilization in the two-pod-diff-node configuration can vary depending on the memory requirements of each pod on different nodes.
- Since the pods are on separate nodes, memory usage is isolated per pod, potentially leading to efficient usage of resources.
- As workloads stabilize, memory utilization tends to balance out as each pod manages its own resources independently.

8 Comparison of Average Response Times

Table 1: Average Response Times across Configurations

Configuration	**Average Response Time (s)**
Horizontal Pod Autoscaler (HPA)	0.02 - 0.04
Vertical Pod Autoscaler (VPA)	0.1 - 0.15
Single Pod	0.1 - 0.15
Two-Container	0.04 - 0.05
Two-Pod Same Node	0.03 - 0.04
Two-Pod Different Node	0.04 - 0.045

This table summarizes the average response times across the six configurations:

- **HPA:** Offers the lowest average response time, ranging from 0.02 to 0.04 seconds. This configuration scales out the number of pods effectively to handle incoming requests, resulting in fast response times.
- **VPA:** The average response time for VPA is higher, ranging from 0.1 to 0.15 seconds, possibly due to dynamic adjustments in resource allocation.
- **Single Pod:** Similar to VPA, single pod configurations exhibit an average response time of 0.1 to 0.15 seconds, as a single pod manages all incoming requests.
- **Two-Container:** The two-container configuration offers an average response time between 0.04 and 0.05 seconds. This setup distributes the workload across multiple containers within a single pod, resulting in relatively low response times.

- **Two-Pod Same Node:** This configuration demonstrates average response times between 0.03 and 0.04 seconds, distributing the workload across two pods on a single node for efficient request handling.
- **Two-Pod Different Node:** This setup results in average response times ranging from 0.04 to 0.045 seconds. Distributing the workload across two pods on different nodes may lead to more balanced response times.

In summary, HPA demonstrates the lowest average response times, while VPA and single pod setups show higher response times. The other configurations offer varying levels of performance based on how the workload is distributed.