Name: Akash Balakrishnan
Student ID: 32192886
Tutors Name: Jay Zhao and Mohammad Goudarzi

FIT5225: Cloud Computing and Security

Assignment 1 Report

Introduction:
This report aims to present a comprehensive analysis of the performance of Cloudiod, a web-based object detection system utilizing YOLO and OpenCV libraries. The Cloudiod system is designed to provide real-time object detection to end-users. It is built using YOLO and OpenCV libraries for image processing and Docker containers and Kubernetes for container orchestration. The main objective of this project is to evaluate the performance of the Cloudiod system under varying loads and number of pod conditions. In this report, I will present my findings and observations regarding the experiments conducted to test the Cloudiod system.

Experimental Design:
To test the Cloudiod system, we varied the number of threads in the client and the number of pods in the Kubernetes cluster. We conducted two sets of experiments, one where the client was executed locally on the master node of Kubernetes and another where the client was executed on a VM instance in our project in Nectar. In each experiment, we sent 128 images to the server, and the average response time was collected. We ran each experiment multiple times to ensure that the average response time values were consistent. We varied the num_threads argument of the Cloudiod_client.py script to 1, 2, 4, 8, and 16 and the number of pods to 1, 2, 4, 8, and 16, resulting in a total of 50 experiments.

Results:
I have plotted the average response time of the web service versus the number of pods for different numbers of threads for both local and Nectar clients in the following figures:

Local Client Plot

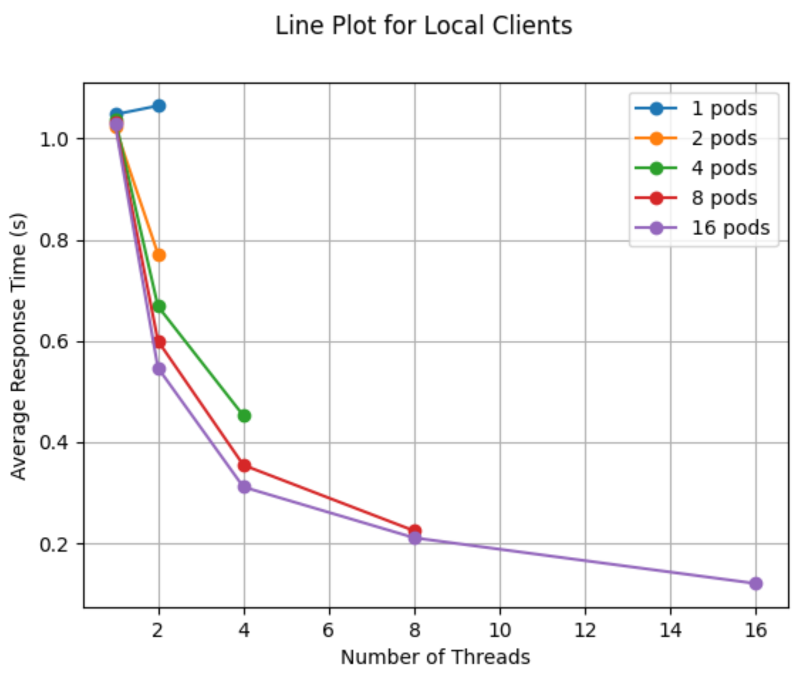

Figure 1: Average response time of the web service versus the number of pods for different numbers of threads for the local client.

Nectar Client Plot

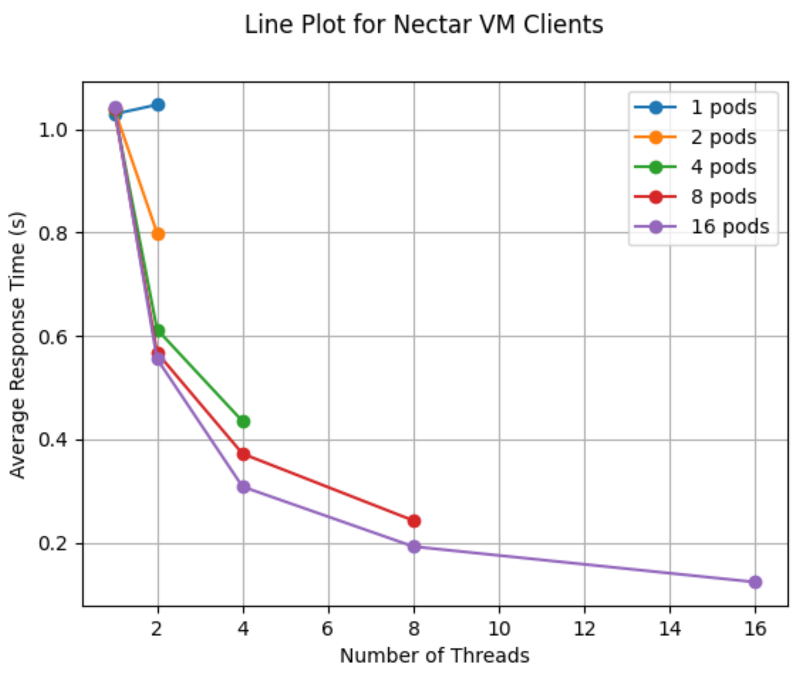

Figure 2: Average response time of the web service versus the number of pods for different numbers of threads for the Nectar client.

Observations:
From the results, we can observe that the average response time of the web service decreases with the number of threads and the number of pods. This is because increasing the number of threads and pods allows the system to handle more requests concurrently, which can improve its throughput and reduce the response time. During my experiments, I observed that when the number of threads exceeded the number of pods in the system, the web service was unable to generate output. In these scenarios, the system became overwhelmed with incoming requests, leading to increased response times and, in some cases, the service becoming unresponsive. This highlights the importance of properly configuring, optimising the web service and scaling the number of threads and pods to ensure that the system can handle the incoming workload.

Another observation is that the response time of the Nectar client is a little higher than that of the local client. This is likely due to the network latency between the Nectar VM and the Kubernetes cluster, which adds to the response time. However, the difference is not significant, and the overall trend remains the same for both clients.

Challenges:
The Cloudiod system faces several challenges during the experimentation. Those three challenges are:

- **Scalability**: In the context of the Cloudiod experiment, scalability refers to the ability of the system to handle an increasing number of requests from end-users while maintaining acceptable performance levels. To address this challenge, Kubernetes was used as the container orchestration system. This allowed the system to scale horizontally by adding more pods to the cluster as demand increased.

During the experiment, the number of pods was gradually increased from 1 to 16, while also increasing the number of threads in the client script. This helped to simulate an increasing demand for the object detection web service. By varying the number of pods and threads, the experiment was able to analyze the impact of different levels of demand on the average response time of the service.

Overall, the use of Kubernetes and horizontal scaling helped to ensure that the Cloudiod system was able to handle a large number of requests from end-users simultaneously, without being limited by a fixed number of resources. As demand increased, more pods could be added to the cluster to ensure that performance levels were maintained, demonstrating the effectiveness of horizontal scaling in addressing the scalability challenge in distributed systems.

- **Failure Handling:** In terms of failure handling, the Cloudiod system encountered some challenges when the number of threads exceeded the number of pods, leading to an overwhelming number of incoming requests that made the service unresponsive. But Kubernetes provides fault-tolerance features such as automatic pod restarts and node failover, which helped the system to handle and recover from failures in individual pods or nodes. These features ensured that the system remained resilient even when individual pods or nodes failed, thereby improving the overall reliability of the Cloudiod system.

- **Security**: As part of the Kubernetes service configuration, the Cloudiod system uses NodePort to expose its deployment and enable communication with the web service running inside the pods. NodePort allows the system to map a well-known port (such as 80 or 8080) to the Kubernetes service port, making it easier to access the service from various locations. In this experiment, I used 30000 as my NodePort to communicate with the web service.

  Though, it is worth noting that NodePort usage does come with certain security considerations, as it exposes the service to the external network. To mitigate these concerns, the Cloudiod system makes use of various security measures, such as network security configurations to ensure that the exposed service remains secure and protected against potential security threats. Overall, the use of NodePort in the Cloudiod system has enabled more efficient and accessible communication with the web service while also maintaining a high level of security.

Conclusion:
In conclusion, this report presented a comprehensive analysis of the performance of the Cloudiod system. The experiments conducted aimed to evaluate the system's performance under varying loads and the number of pods. The results showed that the average response time of the web service decreased with the number of threads and the number of pods, demonstrating the effectiveness of horizontal scaling in distributed systems. Nevertheless, the experiments also revealed that the system could become overwhelmed with incoming requests when the number of threads exceeded the number of pods, highlighting the importance of proper configuration and optimization of the web service. Overall, the results of the experiments demonstrate the feasibility of the Cloudiod system and its potential for real-time object detection applications.

References:
I acknowledge the use of ChatGPT (https://chat.openai.com/) to generate the report for this assessment. I entered the following prompts on 19 April 2023:

Write a report summary about the following specification. Write it in an academic style.
The output from the generative artificial intelligence was adapted and modified for the final response.