Ridong Jiang – 32095597
Tutor：Jay Zhao && Mohammad Goudarzi

In this report, I explore the factors influencing distributed computing by presenting the experimental results used to evaluate the performance of an object detection web service deployed on a Kubernetes cluster. The performance metric used in these experiments is the average response time of the service. The experiments are designed to investigate the impact of varying the number of threads in the client and available resources (number of pods) in the cluster on the service's average response time. Experiments are conducted for two sets of clients: a local client and a client hosted on the Nectar cloud.
The experimental results are presented in the form of two two-dimensional line charts. One of the charts represents the relationship between the average response time of the service for the local client with different numbers of threads and the number of pods. The other chart shows the corresponding data for the Nectar client.

Local Client Results

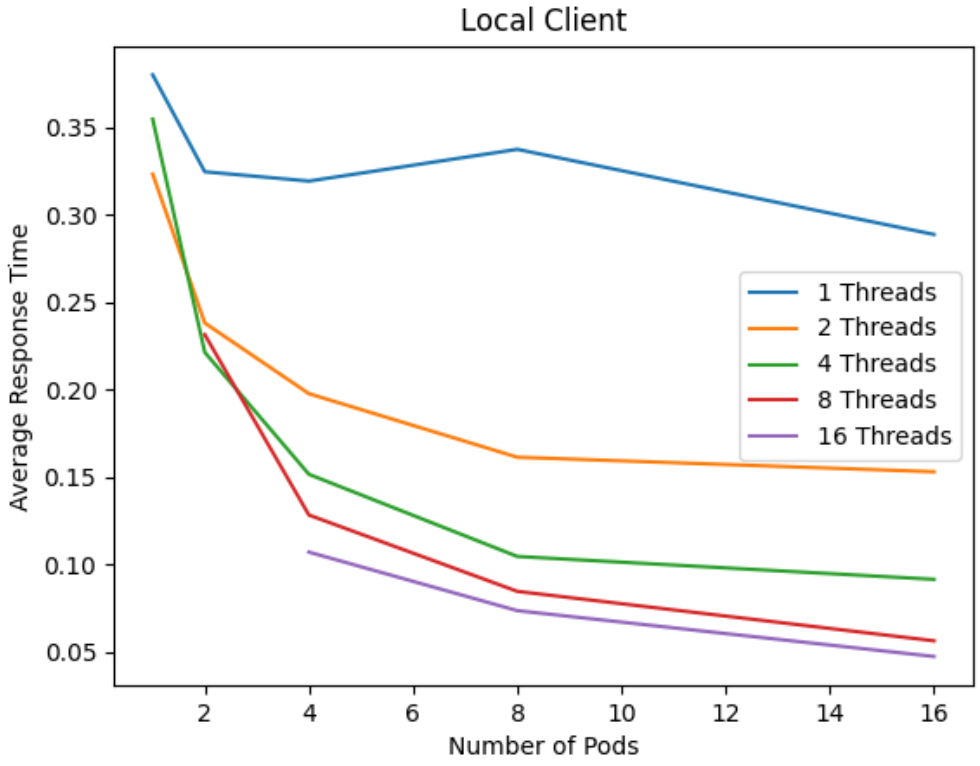

Figure 1: Relationship between average response time and number of pods (different thread counts, local client)

Nectar Client Results

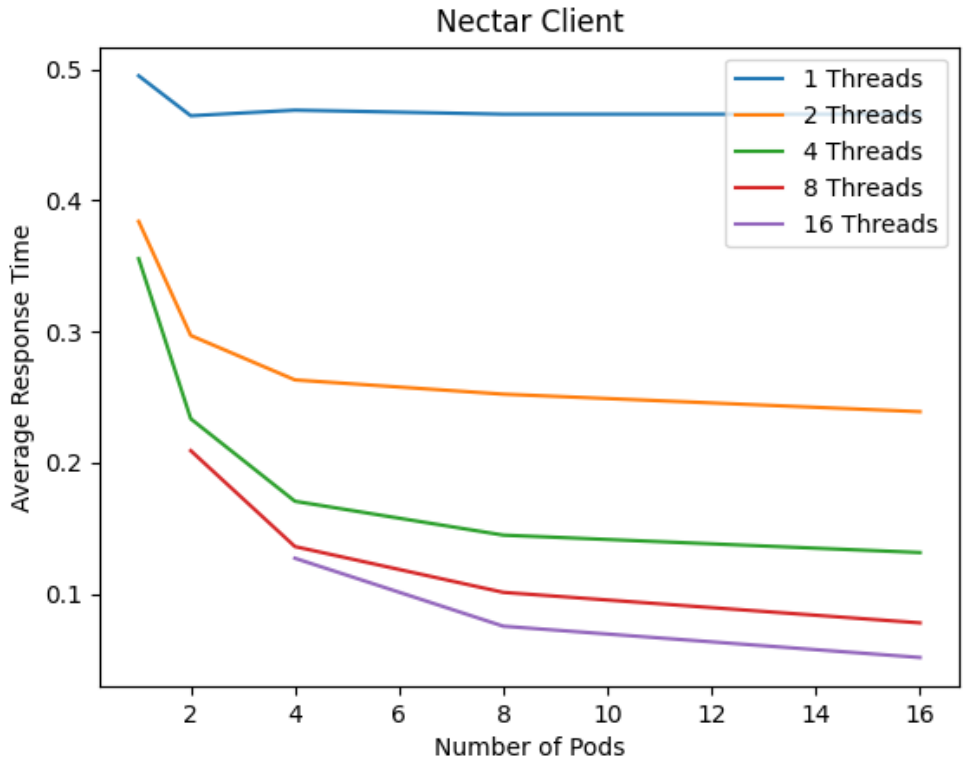

Figure 2: Relationship between average response time and number of pods (different thread counts, Nectar client)

From the plotted charts, we can make the following observations:

1. With different numbers of threads, as the number of pods increases, the average response time for both the local client and the Nectar client generally decreases. This indicates that, under the experimental conditions, the performance of the distributed system improves with an increase in the number of pods.
2. If the number of pods is fixed, the response time decreases as the number of threads increases. This suggests that increasing the number of concurrent threads can further improve system performance.
3. For both clients, the most significant decrease in response time occurs at higher thread counts (e.g., 8 or 16 threads). This may indicate that the performance advantage of the distributed system is more pronounced under high concurrency conditions.
4. However, it is important to note that when the thread count is high but the

number of pods is low, errors occur during execution. In my experimental setup, a single pod can handle up to 4 threads. This reminds us that under high concurrency situations, distributed systems may face challenges related to concurrency control, load balancing, or resource allocation, leading to some request processing failures. We should be particularly cautious in these cases.

5. The average response time for the Nectar client is generally higher than that of the local client. This is likely due to the Nectar client and server being connected via a public network, resulting in greater network latency.
6. When the number of threads is 1, the response time for both the local client and the Nectar client fluctuates with the increase in the number of pods. This may suggest that, under low concurrency scenarios, the impact of increasing the number of pods on improving response time is limited. This could be due to the communication and coordination overhead between multiple pods offsetting performance gains.
7. For some specific thread counts (e.g., 8 and 16 threads), in both the local client and Nectar client, the response time decreases more significantly at higher pod counts (e.g., 16 pods) compared to lower pod counts (e.g., 8 pods). This indicates that when the concurrent thread count is high, the system may be better able to fully utilize distributed computing resources, further improving performance.

Through the analysis above, we can see that increasing the number of pods and client threads in a distributed system has a positive effect on improving system performance. This can help cope with a large number of concurrent requests, thereby enhancing the processing capabilities for compute-intensive tasks such as object detection. However, it is important to note that in practical deployments, there may be a need to balance the relationships among network latency, pod quantity, thread count, and system overhead to find the optimal balance between performance and resource utilization. Additionally, for the runtime errors encountered during the experiment, it is necessary to address potential issues related to concurrency, load balancing, and resource allocation within the system to ensure the stability and reliability of the distributed system.

In conclusion, this report has shed light on the factors influencing the performance of a distributed computing system deployed on a Kubernetes cluster for an object detection web service. The experimental results demonstrate the positive impact of increasing the number of pods and client threads on system performance, while also highlighting the need for careful consideration of network latency, system overhead, and potential issues related to concurrency and resource allocation. By understanding these factors and balancing them effectively, we can optimize their distributed

systems to achieve the best possible performance while ensuring stability and reliability.

In the implementation process of distributed systems, there are many challenges. Among them, Security, Openness, and Heterogeneity have left a deep impression on me. Here are some practical examples of these three challenges in this project and how to address them:

1. Security

   Example: In this experiment, the ports were protected by modifying the firewall and oci security group, which to some extent avoided server intrusion and subsequent data leakage. However, the experiment used the HTTP protocol for image content transmission, making it very susceptible to third-party theft and tampering.

   Challenge explanation: Security issues in distributed systems mainly include protecting data and communications from unauthorized access, tampering, and theft. In our object detection Web service system, an actual security challenge is ensuring the safe transmission of data between the client and the server.

   Solution: To ensure data transmission security, TLS can be used to encrypt communications. In this project, secure data transmission can be achieved by deploying TLS certificates and configuring the Web service to use HTTPS.

2. Openness

   Example: In this experiment, the object detection function was made into a standalone Flask web service. Through this web service and our custom commands, third-party developers can easily use this feature without having to investigate how object detection is implemented. However, the problem with this experiment is that the web service has not been encapsulated as a standardized API but uses special commands.

   Challenge explanation: Openness in distributed systems involves allowing third-party developers to easily integrate their components and applications into existing systems. In the object detection Web service project, an actual openness challenge is designing a system architecture that is easy to expand and integrate.

   Solution: To achieve openness, we can use a Service-Oriented Architecture (SOA) to encapsulate the object detection function into standardized API services such as RESTful API, making it easier for other developers to interact and promote the system's Openness.

3. Heterogeneity

   Example: This experiment uses the cross-platform programming language Python and the virtual environment container technology Docker to ensure maximum

compatibility. However, the experiment is only suitable for kubeadm=1.26.4, docker=23.0.4 or older versions.

Challenge explanation: Heterogeneity in distributed systems involves maintaining system consistency and availability across different types of hardware, software, and network environments. In the object detection Web service project, an actual heterogeneity challenge is ensuring that the service can run properly on different clients and computing environments.

Solution: First, use a cross-platform programming language (such as Python) to develop the object detection Web service to ensure the code's portability across different operating systems and hardware architectures. At the same time, containerization technologies (such as Docker) can be used to encapsulate the service, making its deployment in different environments simpler. After that, try to update our code quickly according to the updates of Docker and kubeadm to ensure the maximum compatibility.