Name: Sahaj Kishore Jain
Student Number: 32313152
Tutor: Mohammad Goudarzi

Assignment 1

FIT5225 Cloud Computing and Security

This project is to build a web application for an object detection python application that is hosted using docker containers and deployed using Kubernetes. The web application is created using python flask and Oracle Cloud Infrastructure (OCI) is the cloud service provider.

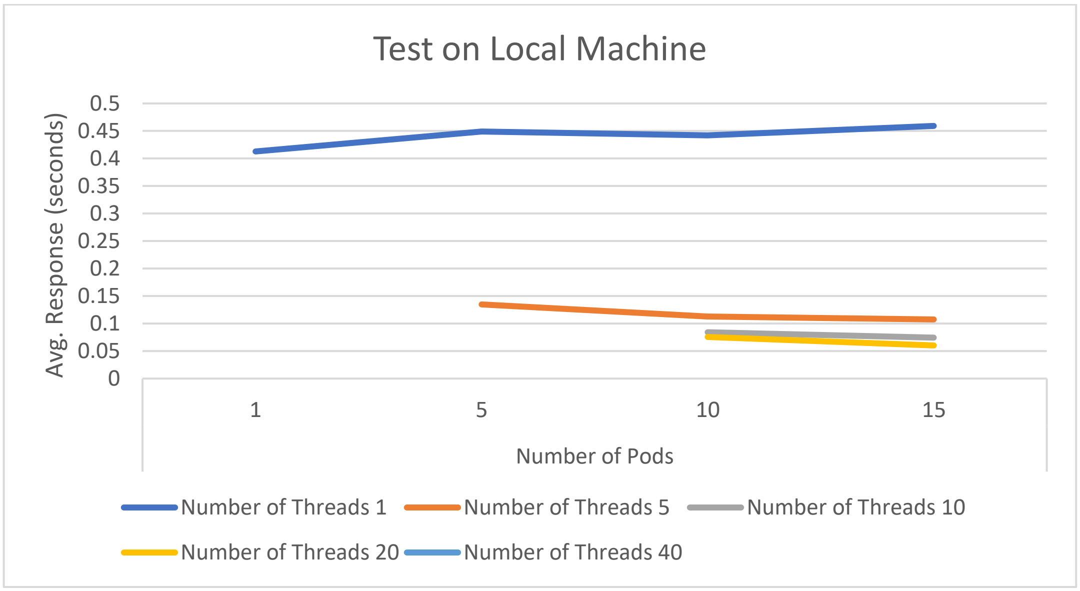
In this experiment, I am running the web application on a Kubernetes cluster with two worker nodes and a master node and am checking the average response time taken by the application for varying worker threads and pods to check the average response time. Average response time is the time taken by the application to process one image and detect all the objects that can be identified by it and returns a JSON response with its UUID and details regarding the objects on that image. I am also checking what performance differences can be observed while running the application on my local system as compared to it being run on the cloud virtual network in OCI.

Observations

Our test runs the web application 120 times with the number of pods varying from 1, 5, 10, 15 and the number of worker threads as 1, 5, 10, 20 and 40. This test is run on both the local machine and the virtual machine on the cloud.

On Local Machine:

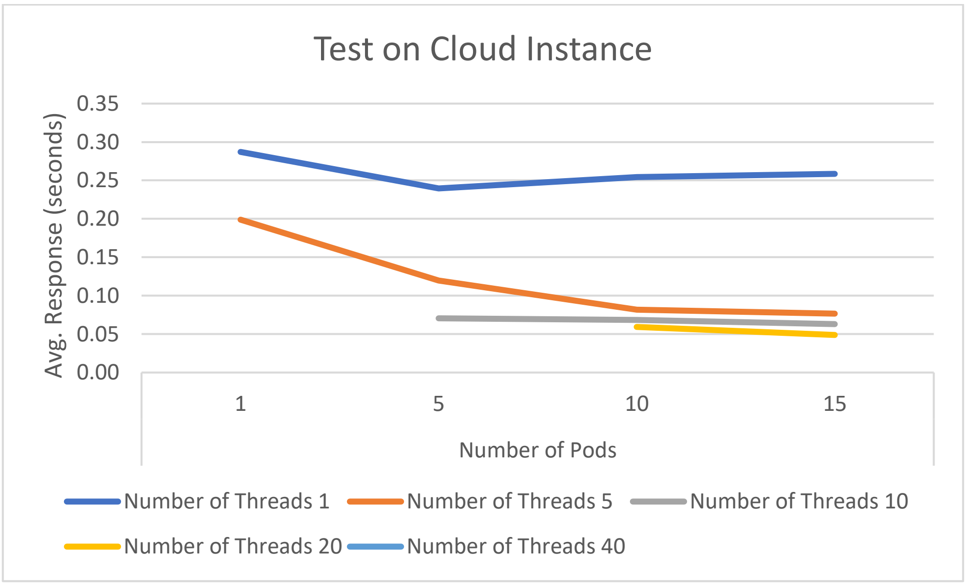
		Number of Pods			
		1	5	10	15
Number of Threads	1	0.41	0.45	0.44	0.46
	5	#N/A	0.13	0.11	0.11
	10	#N/A	#N/A	0.08	0.07
	20	#N/A	#N/A	0.08	0.06
	40	#N/A	#N/A	#N/A	0.06



From the above graph and table for the tests on the local machine, we can infer that the application has the least response time when there are multiple nodes present. On a single-threaded application, we see that the response time is significantly higher than the rest at almost 0.5 seconds per image and is consistent regardless of the number of pods in the cluster. The lease response time is taken when there are 15 pods deployed into the cluster reducing the response tenfold to almost 0.05 seconds. When the number of pods is less, there are higher chances of failure introduced into the system. Using a single pod single thread instance would not be ideal for this kind of application as it would take a lot of time and has a high chance of the pods crashing. We can see a trend where if the number of threads is more than double the number of pods, the pod instances will have a higher likelihood to crash and restart.

On Cloud Instance:

		Number of Pods			
		1	5	10	15
Number of Threads	1	0.29	0.24	0.25	0.26
	5	0.20	0.12	0.08	0.08
	10	#N/A	0.07	0.07	0.06
	20	#N/A	#N/A	0.06	0.05
	40	#N/A	#N/A	#N/A	0.04



For the test that was done on the cloud virtual machine, we kind of see a similar trend where the single-threaded application is slower than the rest. This would be the case as the requests are processed successively. Although we can see that the response rates are lower than that of the local machine. This could be the case as the local machine is not on the Kubernetes cluster and is connected to a different network. This would introduce latency and other network issues that should be taken into consideration while building a web application.

Comparing the two tests, we can see that although the response rate reduces as the number of pods increases, after a certain level the response time is the same regardless of the number of pods. For instance, on increasing the number of pods and threads over 10, we notice that the average response time is very similar to each other implying that for limited users, excess processing power provides no benefit. We see that a lot of crashes occur when there are 40 worker threads in the system irrespective of the number of pods.

Challenges:

There are a lot of difficulties that are encountered when considering an application that is distributed. Some of these challenges that are faced in this application have to do with scalability, fault tolerance and heterogeneity. This application requires a lot of computing power as it needs to use computer vision and prediction algorithms to identify the type of objects in the image. The system also needs to process several images which causes a lot of load and image processing can be a very performance intensive application. To ensure that several image requests can be received at once without causing the system to crash, we have 2 worker nodes that are controlled by the master node. The master node controls these nodes and balances the work they must do and assigns work based on the application requests being sent by clients and other jobs to be done.

Scalability refers to how the application size can increase or decrease based on the needs of your users. This is an advantage of the application being deployed in the Kubernetes cluster. If more worker nodes or more pods are required, they can be created within oracle cloud and added to the cluster.

Heterogeneity talks about how the application can be run on many types of systems regardless of the type of client hardware architecture. As the application is contained within docker, it allows the image to have all the necessary tools for successful execution and can be accessed via the webserver. As we know, java applications are portable as Java provides a virtual environment for its applications to be run on any system through JDK and JVM. This is like how docker containers function as all the configuration files are within the docker image for that application.

Failure handing has to do with how the application will handle any errors or crashes on any pods or nodes in the system. As this system is in a container inside a Kubernetes cluster, there are 2 worker nodes and in case any nodes stop functioning, the other node is always available to pick up and continue running the application without any issues caused to the users. The other advantage of having several pods or nodes is that if there is any update or maintenance that needs to be performed on the system, you do not need to bring the whole system down. Instead, you can provision it on any one of the nodes, while the other node handles the application requests and once the update is completed it can be brought back online into the cluster and allow each part of the system to be updated one by one without downtime or stopping the service.