MONASH University

FIT5225 Assignment 1 Report – iWebLens application

Full Name: Hao Xu

Student Number: 32767919

Tutor Name: Mohammad Goudarzi and Hasanul Ferdaus

Tutorial Time: 10:00 am – 12:00 pm, Monday

1. Abstract

For this project, two series of image objects detection experiments are conducted to collect average response time after local clients and cloud clients send multi-thread requests to the web-based system, iWebLens. Through visualizing and explaining these data, the report identifies and justifies that the average response time can be shortened by increasing the number of pods and threads without "OOMKilled" and "CrashLoopBackOff" pods states. The programmer discussed 3 different challenges and corresponding ways to address them.

2. Result & Discussion

This section presents the results of 40 experiments.[1] The two major variables for these experiments are the number of threads sent by clients and the number of pods on the server-side, which is limited to 0.5 CPU and 512 MiB memory. The programmer also conducts an extra experiment with pods limited on 0.5 CPU.

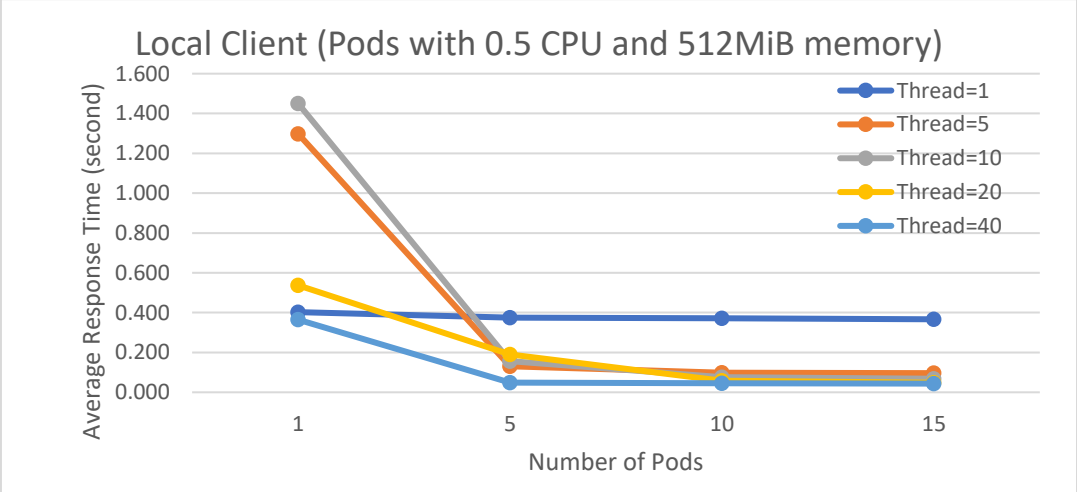

Figure 1: Plot of Local Client Test on Pods with 0.5 CPU and 512 MiB memory

[1] The 40 experiments are 2 (local clients/cloud clients) x 4 (pods - server side) x 5 (requests sent by clients) groups.

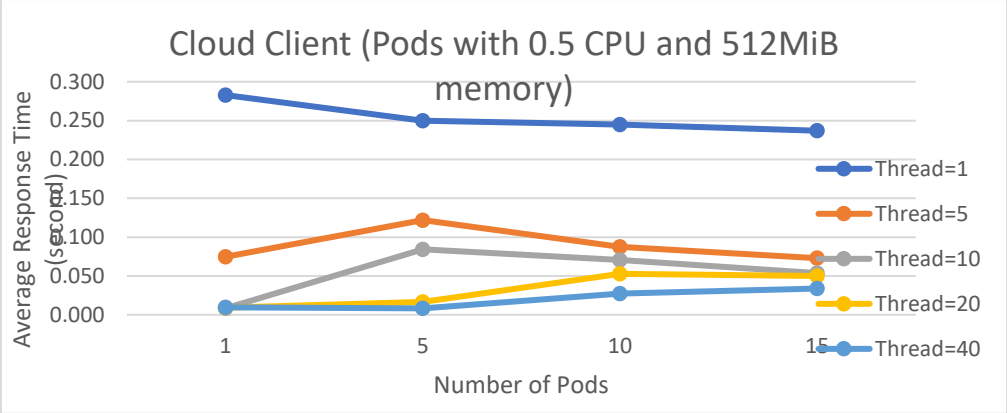

Figure 2: Plot of Cloud Client Test on Pods with 0.5 CPU and 512 MiB memory

a) Normal behavior pods without "OOMKilled" state

The report first demonstrates pods more than 5 and threads less than 40, where the "OOMKilled" and "CrashLoopBackOff" states do not occur. By looking at the plots of both local and cloud client tests (Figure 1 and Figure 2), several trends can be detected.

We can fix the number of threads to analysis the effects of different pods. When the client sends a single-thread request, the average response time slightly decrease as the number of pods increase, while this amount of time difference is minimal. Quantitatively speaking, in the local client test, all response times of 1 thread with multiple pods are around 0.40 seconds. However, when the client sends a multiple-thread request, the average response time reveals an obvious drop as the number of pods are increased. For example, in cloud client test, when threads are 10, the average response time of 5 pods is 0.08 second (s). This duration is reduced to 0.05s when the number of pods is increased to 15. That's because Kubernetes deploys pods equally in two worker nodes and then distributes tasks to the pods in a balanced way so as to increase the speed of processing requests and decrease the resources pressure of each pod.

When the number of pods is fixed, the average response time decrease when the number of threads is increased. In Figure 2, when the number of pods is 15 and the client sends a single-thread request, the average response time is around 0.2s; When the client sends 40 threads request at the same time, the average response time is reduced to 0.03s. This difference can be explained by one key feature of Kubernetes. This software balances tasks among various pods so that these requests can be processed concurrently. Moreover, each pod is running a whole python flask application inside the container, non-interfering to each other. That is to say, every pod can be regarded as an independent web service processing the request. This quality serves as one key advantage for using Kubernetes to do container orchestration, which easily scales up the web-based application to handle more client requests.

b) Pods with "OOMKilled" and "CrashLoopBackOff" state

However, we can see there are several odd points on both local client plot and cloud client plot. Looking at the data in detail, in local client test (Figure 3), when there are only 5 pods, but the local client sends 40-thread requests at the same time, the average response time is around 1.5s which is extremely high. That is because when all pods are running out of resource, all of them are in "CrashLoopBackOff" state and local client needs to wait for responses. On the other hand (Figure 5), when cloud client sends 40-thread requests to 5 pods, the average response time is around 0.008 which is extremely low. That is because webservice refuses cloud client requests immediately. During these high-pressure tests, the single pod experiences several lifecycle states, which are "Running", "OOMKilled", "CrashLoopBackOff" and back to "Running." This process is caused by resource starvation. When this single pod is deployed with the deployment YAML file, the coder manually sets the memory request and limits it to 512MiB in the first place. In this case, when the client sends so many threads concurrently (20 threads, for instance) to the web service, there are still excessively many local variables created in the container's memory of this pod, even though the coder reduces the number of local variables and uses global variables as many as possible. If the pod's container memory limit is reached, "OOMKilled" state will occur, and the pod may crash. If the pod is trapped in this endless cycle of starting and crashing, the state will become "CrashLoopBackOff."

There are several ways to address this issue. Firstly, since the limitations are assigned to pods' CPU and memory, creating more pods can increase the number of accessible resources. Secondly, assigning more resources to each pod can raise the memory limit. The third method is using fewer local variables and more global variables so that the memory limit will not be reached so quickly.

	A	B	C	D	E	F	G
1	Pods (with 0.5 CPU and 512MiB) - tested by loacal client						
2			Average Response Time				
3	# of Pods	# of Threads	Experiment 1	Experiment 2	Experiment 3	**Avg. Experiment**	Comment
4	1	1	0.41535803	0.39729532	0.39481957	0.40249097	
5	5	1	0.38249490	0.36131623	0.38028212	0.37469775	
6	10	1	0.37209815	0.36556761	0.37801834	0.37189470	
7	15	1	0.37454604	0.37091845	0.35435769	0.36660739	
8	1	5	1.29176019	1.30549187	1.29758943	1.29828050	OOMKilled at middel of the testing
9	5	5	0.13512373	0.11920829	0.13739998	0.13057733	
10	10	5	0.09380531	0.09495767	0.10706998	0.09861099	
11	15	5	0.09878534	0.09218102	0.09755295	0.09617311	
12	1	10	1.41042215	1.50472614	1.43574911	1.45029913	OOMKilled at the beginning
13	5	10	0.16069432	0.19923415	0.11069272	0.15687373	OOMKilled at middel of the testing
14	10	10	0.07587058	0.07851114	0.07340713	0.07592962	
15	15	10	0.06741380	0.06580433	0.07113373	0.06811728	
16	1	20	0.53054548	0.52479302	0.55648220	0.53727357	OOMKilled at the beginning
17	5	20	0.18015090	0.20479322	0.18634824	0.19043079	OOMKilled at middel of the testing
18	10	20	0.05861847	0.05936694	0.05906399	0.05901646	OOMKilled at middel of the testing
19	15	20	0.04829235	0.04814560	0.04814560	0.04819451	
20	1	40	0.35125989	0.36323499	0.37982389	0.36477292	OOMKilled at the beginning
21	5	40	0.04948100	0.04954323	0.04509490	0.04803971	OOMKilled at the beginning
22	10	40	0.04622051	0.04457199	0.04651176	0.04576809	OOMKilled at middel of the testing
23	15	40	0.04100173	0.04486066	0.04507794	0.04364678	
24							
25	Note: Yellow cells meas when the test is running, pod(s) are facing OOMKilled and CrashLoopBackOff states multiple times						
26							

Figure 3: Detail Data of Local Client Test on Pods with 0.5 CPU and 512 MiB memory

	A	B	C	D	E	F	G
1	Pods (with 0.5 CPU and 512MiB) - tested by cloud client						
2			Average Response Time				
3	# of Pods	# of Threads	Experiment 1	Experiment 2	Experiment 3	**Avg. Experiment**	Comment
4	1	1	0.28346932	0.28568748	0.27938177	0.28284619	
5	5	1	0.24583643	0.25326415	0.25029815	0.24979958	
6	10	1	0.24462829	0.24473921	0.24480917	0.24472556	
7	15	1	0.23711511	0.23367767	0.24013006	0.23697428	
8	1	5	0.08626584	0.08315917	0.05490645	0.07477715	OOMKilled at middel of the testing
9	5	5	0.11539741	0.11642185	0.13374817	0.12185581	
10	10	5	0.08751444	0.08843219	0.08648638	0.08747767	
11	15	5	0.07623995	0.07271971	0.06954368	0.07283445	
12	1	10	0.00805410	0.00834462	0.00846187	0.00828686	OOMKilled at the beginning
13	5	10	0.08064833	0.08463813	0.08749371	0.08426006	OOMKilled at middel of the testing
14	10	10	0.06914671	0.07074682	0.07174917	0.07054757	
15	15	10	0.05399357	0.05317910	0.05346148	0.05354472	
16	1	20	0.01041880	0.00846282	0.00846316	0.00911493	OOMKilled at the beginning
17	5	20	0.01312734	0.01643819	0.01974319	0.01643624	OOMKilled at middel of the testing
18	10	20	0.05076505	0.05469822	0.05271491	0.05272606	OOMKilled at middel of the testing
19	15	20	0.04967452	0.05326419	0.04749322	0.05014397	
20	1	40	0.00970074	0.00963273	0.00965124	0.00966157	OOMKilled at the beginning
21	5	40	0.00851584	0.00879042	0.00746137	0.00825588	OOMKilled at the beginning
22	10	40	0.02912761	0.02473220	0.02749871	0.02711951	OOMKilled at middel of the testing
23	15	40	0.03035808	0.03643822	0.03453169	0.03377600	
24							
25	Note: Yellow cells meas when the test is running, pod(s) are facing OOMKilled and CrashLoopBackOff states multiple times						

Figure 4: Detail Data of Cloud Client Test on Pods with 0.5 CPU and 512 MiB memory

3. Challenges and Solutions

In this section, the programmer raises 3 common challenges that are likely to appear in cloud computing and how these issues are addressed in his system.

a) Scalability

Scalability refers to the fact that a system should be able to handle the growth of the number of users. In the system for this project, the number of users to use this iWebLens app may shoot up at a particular time so that the pressure on the web service can be huge and the CPU or memory resource of 2 worker node VMs may be ran out. To address this issue, the programmer first scales up the system by creating more VMs to play the role of worker nodes. After that, these worker nodes can be easily joined to the master node and afford user's pressure, since Kubernetes Cluster provides an easy way to combine worker nodes into the master node. Another solution is scaling out. This means that when the VMs are created, more CPUs and memory can be assigned to a single VM so that it has more resources for client users to use.

b) Concurrency

Concurrency means that multiple clients can access the same resources at the same time. In this system, multiple users may send multiple images by multi-thread requests to the iWebLens app simultaneously. To address this potential issue, the coder uses two techniques: container and multi-thread programming. Firstly, the python-based application used Flask package which can handle multi-thread client request. Only variables related to neural network part stay in local variables while other variables, such as yolo-tiny-configs, yolo-

tiny-label, are global variables that can be shared by clients. Secondly, the coder encapsulates and packages his application into a container by using Docker, so that Kubernetes can create multiple pods, including a container, where an application can run independently in it. Based on these designs, when multiple users send requests to this system concurrently, Kubernetes can automatically assign these tasks to different pods in order to process them at the same time without slowing down the processing time.

c) Failure Handling

Failure handling requires the system to handle different types of failure, such as wrong data type sent by clients and the failure of the web service. In this system, it is possible that clients send a non-image data or send too many requests, crashing the web service. To address these types of problems, detecting, tolerating and recovery methods can be adopted. For the first problem, this python app is designed to detect the data type sent by clients and throw an exception if the data type is wrong. As for the second issue, if the webservice is down, the following message, "the connection runs out of time, please try again later," will be shown to the clients. Also, this Kubernetes application can restart the pods automatically if they meet potential crash states, such as "OOMKilled" and "CrashLoopBackOff".

4. References

Anderson, K. (2018, March 27). *What is a CrashLoopBackOff? How to alert, debug / troubleshoot, and fix Kubernetes CrashLoopBackOff events*. Sysdig. https://sysdig.com/blog/debug-kubernetes-crashloopbackoff/

Darshan, N. (2021, January 13). *How To Setup A Three Node Kubernetes Cluster For CKA: Step By Step.* K21Academy. https://k21academy.com/docker-kubernetes/three-node-kubernetes-cluster/?utm_source=onlineappsdba&utm_medium=referral&utm_campaign=kubernetes30_oct20

Komodor. (n.d.). *How to fix OOMKilled Kubernetes error (exit code 137).* https://komodor.com/learn/how-to-fix-oomkilled-exit-code-137/

Kumar, A. (2020, September 20). *Kubernetes Pods For Beginners.* K21Academy. https://k21academy.com/docker-kubernetes/kubernetes-pods-for-beginners/

Weaveworks. (n.d.). *Integrating Kubernetes via the Addon.* https://www.weave.works/docs/net/latest/kubernetes/kube-addon/#pod-network