Cloud Computing Architecture

Semester project

March 15, 2021

Overview

The semester project consists of four parts, two of which are described in detail in this handout. In this project, you will explore how to schedule latency-sensitive and batch applications in a cloud cluster. You will deploy applications inside of containers and gain experience using a popular container orchestration platform, Kubernetes. Containers are a convenient and lightweight mechanism for packaging code and all its dependencies so that applications can run quickly and reliably from one computing environment to another.

You will work in groups of three students and submit a single report per group. **Please submit your report in the format of the project report template**. We will be assigning groups for the project, however you will have a chance to optionally let us know your preferences for teammates. If you know one or two other students in the class that you would like to work with on the project, please submit your group preference by March 12th, 2021. To do so, each student in your preferred group should sign up for the same group number in the Project Group Selection page on Moodle. We will notify you about final group assignments on March 15th and then you may redeem your cloud credits and begin working on the project.

Important Dates

March 12th, 2021: Deadline to submit group preferences (optional, since we will assign groups).
March 15th, 2021: Groups are assigned and announced. Start working on project.
April 13th, 2021: Deadline to submit Part 1 and 2 of the project.
May 21st, 2021: Deadline to submit Part 3 and 4 of the project.

We will release Part 3 and 4 of the project mid-April. Parts 3 and 4 are more open-ended and will require more time to complete than Part 1 and 2. Please plan your time accordingly.

Cloud Environment and Credits

To run experiments for the project, you will use Google Cloud. We will provide you with Google Cloud credits for your project. To redeem your cloud credits, please follow the steps in Part 1 (Section 1.1) *after March 15th*, when your project group assignment is confirmed. Each group member should create a Google Cloud account at `https://accounts.google.com`. Please use your ETH email address to create the account.

1 Part 1

In Part 1 of this project, you will run a latency-critical application, memcached, inside a container. Memcached is a distributed memory caching system that serves requests from clients over the network. A common performance metric for memcached is the tail latency (e.g., 95th percentile latency) under a desired query rate. You will measure tail latency as a function of queries per second and explore the impact of hardware resource interference. To add different types of hardware resource contention, you will use the iBench microbenchmark suite to apply different sources of interference (e.g., CPU, caches, memory bandwidth).

Follow the setup instructions below to deploy a Google Cloud cluster using the `kops` tool. Your cluster will consist of four virtual machines (VMs). One VM will serve as the Kubernetes cluster master, one VM will be used to run the memcached server application and iBench workloads, and two VMs will be used to run a client program that generates load for the memcached server.

In your project report (see report template), answer the questions in Section 1.2.

1.1 Setup Instructions

Installing necessary tools

For the setup of the project, you will need to install kubernetes, google-tools and kops. Instructions based on the operating system on your local machine are provided in the links above. Having installed all the tools successfully, the following three commands should return output in your terminal (for the rest of the document the $ symbol is there to declare a bash command and you shouldn't type it explicitly):

1. `$ kubectl --help`
2. `$ kops --help`
3. `$ ./google-cloud-sdk/bin/gcloud --help`

Note that the final command is relative to where you have downloaded the google cloud tools. If you have installed via a package manager or have added the gcloug tools to your `$PATH` you don't need the prefix and you can just type `gcloud`. Note that you have to open a new terminal or refresh your shell using `source` for your `$PATH` to be updated.

All the scripts that you will need for both parts of the project are available here:

```
git clone https://github.com/eth-easl/cloud-comp-arch-project.git
```

Redeeming cloud credits and creating Google Cloud project

Each group member should create a Google Cloud account at `https://accounts.google.com`. Use your ETH email address to create the account. Each group will receive a $100 Google Cloud coupon code. Select **one** group member to enter their name and ETH email address at this link. This student must use their @student.ethz.ch or @ethz.ch email address for the form. A confirmation email will be sent to the student with a coupon code. You can redeem the coupon using the following link, **using your ETH email address**.

After installing kubernetes tools, connect your local client to your google cloud account using:

`gcloud init`

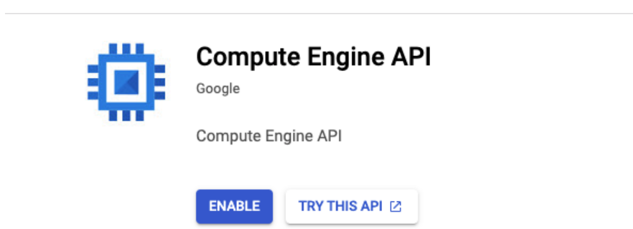
A browser window will open and you will have to login in with your ETH address. Afterwards, you will give `google-cloud-sdk` permissions to your account and then in the command line you will pick a name for the project. When creating the project name use `cca-eth-2021-group-XXX` (where XXX is your group number). **Only one group member (who also redeemed the cloud credit coupon) should create the Google Cloud project.** This person will add other group members as Project Owners (see instructions below). After the other group members are added as Project Owners, they will simply select the existing project name when they run the `gcloud init` command. All group members will have access to the project and share the cloud credits.

Do not configure any default computer region and zone. For deploying a cluster on Google Cloud we will follow the instructions here with some modifications.

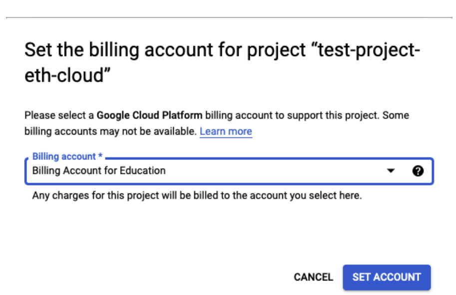
After creating the project, you will use the `gcloud compute zones list` to redeem your coupon. After typing the command you will get an error message like the following:

```
ERROR: (gcloud.compute.zones.list) Some requests did not succeed:
 - Project 610127079583 is not found and cannot be used for API calls. If it is recently created,
 enable Compute Engine API by visiting <YOUR_GOOGLE_PROJECT_LINK> then retry.
 If you enabled this API recently, wait a few minutes for the action
 to propagate to our systems and retry.
```

Follow the link provided. You will be redirected to a page with the following output on the top left:



There you will have to click the `Enable` button and then a window that has the option `Enable billing` will pop-up. Choose `Billing account for education` as below and click `Set account`:



Afterwards, if you type the `gcloud compute zones list` command again, you should see an output similar to the one below:

```
$ gcloud compute zones list
NAME                    REGION                  STATUS
us-east1-b              us-east1                UP
us-east1-c              us-east1                UP
us-east1-d              us-east1                UP
us-east4-c              us-east4                UP
us-east4-b              us-east4                UP
us-east4-a              us-east4                UP
us-central1-c           us-central1             UP
us-central1-a           us-central1             UP
us-central1-f           us-central1             UP
us-central1-b           us-central1             UP
us-west1-b              us-west1                UP
us-west1-c              us-west1                UP
us-west1-a              us-west1                UP
europe-west4-a          europe-west4            UP
europe-west4-b          europe-west4            UP
europe-west4-c          europe-west4            UP
europe-west1-b          europe-west1            UP
europe-west1-d          europe-west1            UP
europe-west1-c          europe-west1            UP
europe-west3-c          europe-west3            UP
europe-west3-a          europe-west3            UP
europe-west3-b          europe-west3            UP
europe-west2-c          europe-west2            UP
europe-west2-b          europe-west2            UP
europe-west2-a          europe-west2            UP
```

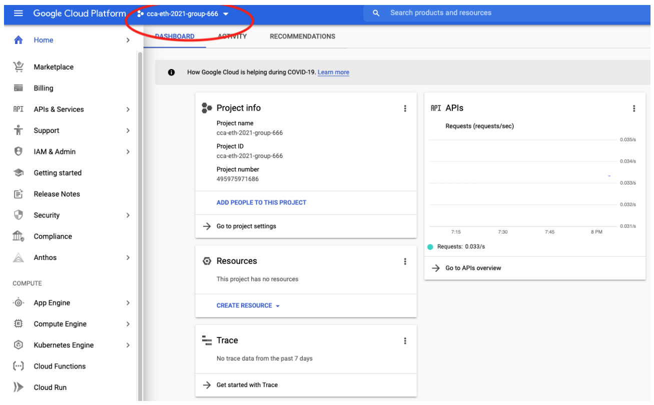
Then you will need to configure your default credentials using:

```
$ gcloud auth application-default login
```

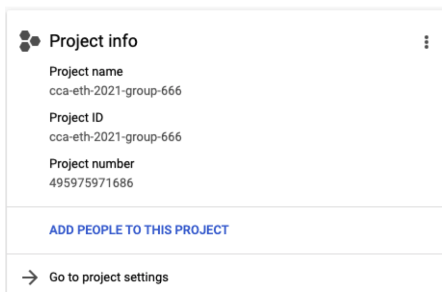
This will redirect you to a browser window where you will login with the same account you used when you setup the `gloud init` command.

Giving your teammates owner permission to the project

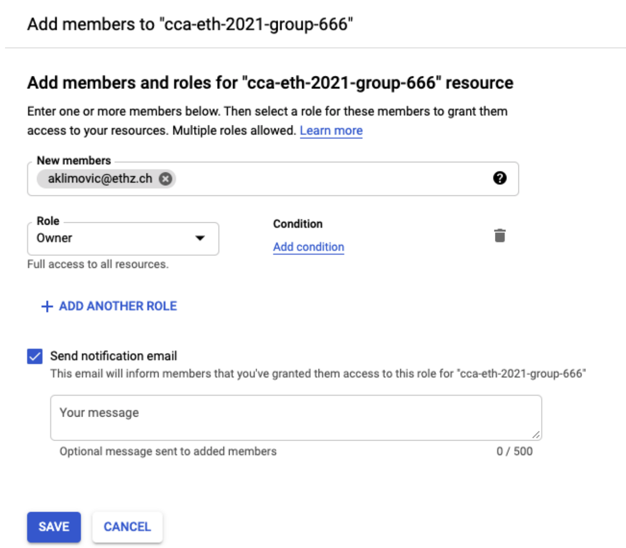
After creating the `cca-eth-2021-group-XXX` project on Google Cloud, give your group members access to the project and cloud credits by navigating to the Google Cloud console menu. Make sure your project is properly displayed on the top left as below:



In the project info click `Add people to this project`.



Type the email addresses of your teammates, select `Owner` as a role and click `Save`. **Note that your teammates should have created a google cloud account with their ETH address in advance to put them as project owners.**



Deploying a cluster using `kops`

At this point you will deploy a cluster using `kops`. First of all you will need to create an empty bucket to store the configuration for your clusters. Do this by running:

```
$ gsutil mb gs://cca-eth-2021-group-XXX-ethzid/
```

where `XXX` is your group number and `ethzid` is your ETH username. Then run the following command to have the `KOPS_STATE_STORE` command to your environment for the subsequent steps:

```
$ export KOPS_STATE_STORE=gs://cca-eth-2021-group-XXX-ethzid/
```

If you open another terminal this and other environmental variables will not be preserved. You can preserve it by adding it with an `export` command to your .bashrc You should substitute the number of your group and your ETH username as before.

For the first part of the exercise you will need a 3 node cluster. Two VMs will have 2 cores. One of these VMs will be the node where `memcached` and `iBench` will be deployed and another will be used for for the `mcperf` memcached client which will measure the round-trip latency of memcached requests. The third VM will have 8 cores and hosts the `mcperf` client which generates the request load for the experiments.

Before you deploy the cluster with `kops` you will need an ssh key to login to your nodes once they are created. Execute the following commands to go to your `.ssh` folder and create a key:

```
$ cd ~/.ssh
$ ssh-keygen -t rsa -b 4096 -f cloud-computing
```

Once you have created the key, go to lines 16 and 43 of the `part1.yaml` file (provided in the github link above) and **substitute the placeholder values with your group number and ethzid**. Then run the following commands to create a kubernetes cluster with 1 master and 2 nodes.

```
$ PROJECT=`gcloud config get-value project`
$ export KOPS_FEATURE_FLAGS=AlphaAllowGCE # to unlock the GCE features
$ kops create -f part1.yaml
```

We will now add the key as a login key for our nodes. Type the following command:

```
$ kops create secret --name part1.k8s.local sshpublickey admin -i ~/.ssh/cloud-computing.pub
```

We are ready now to deploy the cluster by typing:

```
$ kops update cluster --name part1.k8s.local --yes --admin
```

Your cluster should need around 5-10 minutes to be deployed. You can validate this by typing:

```
$ kops validate cluster --wait 10m
```

The command will terminate when your cluster is ready to use. If you get a `connection refused` or `cluster not yet healthy` messages, wait while the previous command automatically retries. When the command completes, you can type

```
kubectl get nodes -o wide
```

to get the status and details of your nodes as follows:

```
NAME                        STATUS   ROLES    AGE    VERSION   INTERNAL-IP   EXTERNAL-IP
master-europe-west3-a-2s21  Ready    master   3m2s   v1.19.7   10.156.0.63   34.107.107.152
memcache-server-jrk4        Ready    node     102s   v1.19.7   10.156.0.61   34.107.94.26
client-agent-vg5v           Ready    node     98s    v1.19.7   10.156.0.62   34.89.236.52
client-measure-ngwk         Ready    node     102s   v1.19.7   10.156.0.60   35.246.185.27
```

You can connect to any of the nodes by using your generated ssh key and the node name. For example to connect to the client-agent node, you can type:

```
$ gcloud compute ssh --ssh-key-file ~/.ssh/cloud-computing ubuntu@client-agent-vg5v \
                     --zone europe-west3-a
```

Running memcached and the mcperf load generator

To launch memcached using Kubernetes, run the following:

```
$ kubectl create -f memcache-t1-cpuset.yaml
$ kubectl expose pod some-memcached --name some-memcached-11211  \
                                    --type LoadBalancer --port 11211 \
                                    --protocol TCP
$ sleep 60
$ kubectl get service some-memcached-11211
```

Then run:

```
$ kubectl get pods -o wide
```

The ouput should look like:

```
NAME             READY   STATUS    RESTARTS   AGE   IP           NODE
some-memcached   1/1     Running   0          42m   100.96.3.3   memcache-server-zns8
```

Use the IP address above (`100.96.3.3` in this example) as the `MEMCACHED_IPADDR` in the remaining instructions. Now ssh into both the `client-agent` and `client-measure` VMs and run the following commands to compile the `mcperf` memcached load generator:

```
$ sudo apt-get update
$ sudo apt-get install libevent-dev libzmq3-dev git make g++ --yes
$ sudo cp /etc/apt/sources.list /etc/apt/sources.list~
$ sudo sed -Ei 's/^# deb-src /deb-src /' /etc/apt/sources.list
$ sudo apt-get update
$ sudo apt-get build-dep memcached --yes
$ cd && git clone https://github.com/shaygalon/memcache-perf.git
$ cd memcache-perf
$ make
```

On the `client-agent` VM, you should now run the following command to launch the `mcperf` memcached client load agent with 16 threads:

```
$ ./mcperf -T 16 -A
```

On the `client-measure` VM, run the following command to first load the memcached database with key-value pairs and then query memcached with throughput increasing from 5000 queries per second (QPS) to 55000 QPS in increments of 5000:

```
$ ./mcperf -s MEMCACHED_IP --loadonly
$ ./mcperf -s MEMCACHED_IP -a INTERNAL_AGENT_IP  \
          --noload -T 16 -C 4 -D 4 -Q 1000 -c 4 -t 5 \
          --scan 5000:55000:5000
```

where `MEMCACHED_IP` is from the output of `kubectl get pods -o wide` above and `INTERNAL_AGENT_IP` is from the Internal IP of the client-agent node from the output of `kubectl get nodes -o wide`. You should look at the output of `./mcperf -h` to understand the different flags in the above commands.

Introducing Resource Interference

Now we are going to introduce different types of resource interference with iBench microbenchmarks. Run the following commands:

```
$ kubectl create -f interference/ibench-cpu.yaml
```

This will launch a CPU interference microbenchmark. You can check it is running correctly with:

```
$ kubectl get pods -o wide
```

When you have finished collecting memcached performance measurements with CPU interference, you should kill the job by running:

```
$ kubectl delete pods ibench-cpu
```

Then repeat the three steps in this section with `ibench-l1d, ibench-l1i, ibench-l2, ibench-llc`, and `ibench-membw` interference microbenchmarks.

Deleting your cluster

IMPORTANT: you must delete your cluster when you are not using it! Otherwise, you will easily use up all of your cloud credits! When you are ready to work on the project, you can easily re-launch the cluster with the instructions above.
To delete your cluster, run on your local machine the command:

```
$ kops delete cluster part1.k8s.local --yes
```

1.2 Questions

Please answer the following questions in the report you submit.

1. **[10 points]** Using the instructions above, run memcached alone (i.e., no interference), and with each iBench source of interference (cpu, l1d, l1i, l2, llc, membw). Plot a single line graph with 95th percentile latency on the y-axis (the y-axis should range from 0 to 10 ms) and QPS on the x-axis (the x-axis should range from 0 to 55K) for each configuration (7 lines in total). Label your axes. State how many runs you averaged across (we recommend a minimum of 3) and include error bars. The readability of your plot will be part of your grade. **The shape of the curves depends on many factors and may differ across groups. However, that does not imply that your solutions are wrong.**

2. **[6 points]** How is the tail latency and saturation point (the "knee in the curve") of memcached affected by each type of interference? Why? Briefly describe your hypothesis.

3. **[2 points]** Explain the use of the taskset command in the container commands for memcached and iBench in the provided scripts. Why do we run some of the iBench benchmarks on the same core as memcached and others on a different core?

4. **[2 point]** Assuming a service level objective (SLO) for memcached of up to 2 ms 95th percentile latency at 40K QPS, which iBench source of interference can safely be collocated with memcached without violating this SLO?

2 Part 2

In Part 2 of this project, you will run six different throughput-oriented ("batch") workloads from the `PARSEC` benchmark suite: `blackscholes`, `fft`, `dedup`, `ferret`, `freqmine`, and `canneal`. You will first explore each workload's sensitivity to resource interference using iBench on a small 2 core VM (`e2-standard-2`). This is somewhat similar to what you did in Part 1 for `memcache`. Next, you will investigate how each workload benefits from parallelism by measuring the performance of each job with 1, 2, 4, 8 threads on a large 8 core VM (`e2-standard-8`). In the latter scenario, no interference is used.

Follow the setup instructions below to deploy a Google Cloud cluster and run the batch applications. In your project report, answer the questions in Section 2.2.

2.1 Setup

In order to complete this Part of the project, we will have to study the behavior of `PARSEC` in two different contexts. For both, we will require that `kubectl`, `kops` and `gcloud sdk` are set up. This should already be the case if you have completed Part 1.

We have provided you with a set of `yaml` files which are useful towards spawning `kubectl` jobs for workloads and interference. The interference files are the same as in Part 1, **but you must change the nodetype from memcached to parsec**. The `PARSEC` workloads are in the `parsec-benchmarks/part2a` folder in the github repo. All these files cover the workloads in the `PARSEC` suite, as well as the `iBench` interference sources relevant for this part: `cpu`, `l1d`, `l1i`, `l2`, `llc`, `memBW`.

2.1.1 `PARSEC` Behavior with Interference

For the first half of Part 2, you will have to set up a single node cluster consisting of a VM with 8 CPUs. For this, we will employ `kops` and make use of the `part2a.yaml` file (make sure to update the file with values for your GCP project):

```
$ export KOPS_STATE_STORE=<your-gcp-state-store>
$ export KOPS_FEATURE_FLAGS=AlphaAllowGCE
$ PROJECT=`gcloud config get-value project`
$ kops create -f part2a.yaml
$ kops update cluster part2a.k8s.local --yes --admin
$ kops validate cluster --wait 10m
$ kubectl get nodes -o wide
```

If successful, you should see something like this:

```
NAME                          STATUS   ROLES    AGE    VERSION   INTERNAL-IP    EXTERNAL-IP
master-europe-west3-a-9nxl    Ready    master   3m2s   v1.19.7   10.156.0.46    34.107.0.118
parsec-server-s28x            Ready    node     104s   v1.19.7   10.156.0.47    35.234.110.58
```

Now you should be able to connect to the `parsec-server` VM using either `ssh`:

```
$ ssh -i ~/.ssh/cloud-computing ubuntu@35.234.110.58
```

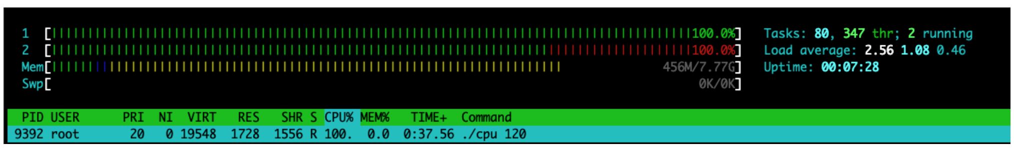
Or by using `gcloud`:

```
$ gcloud compute ssh --ssh-key-file ~/.ssh/cloud-computing ubuntu@parsec-server-s28x \
                     --zone europe-west3-a
```

For this part of the study we will sometimes require to set up some form of interference, and also deploy a `PARSEC` job. For this example, we will use the `PARSEC fft` job together with `iBench` CPU interference. Here is where we will use `kubectl` together with some of the `yaml` files we provide. The following code snippet spins up the interference, and runs the `PARSEC fft` job:

```
$ kubectl create -f ibench-cpu.yaml  # Wait for interference to start
$ kubectl create -f parsec-fft.yaml
```

Make sure that the interference has properly started **before** running the `PARSEC` job. One way to see if the interference and the `PARSEC` job has started refers to `ssh`-ing into the VM and using the `htop` command to inspect running processes. You should see an image like below:



You can get information on submitted jobs using:

```
$ kubectl get jobs
```

In order to get the output of the `PARSEC` job, you will have to collect the logs of its pods. To do so, you will have to run the following commands.

```
$ kubectl logs $(kubectl get pods --selector=job-name=<job_name> \
        --output=jsonpath='{.items[*].metadata.name}')
```

Run experiments sequentially and wait for one benchmark to finish before you spin up the next one. *Since fft, ferret, freqmine, and canneal jobs take over 5 minutes to complete with the native dataset, you must use a smaller dataset called simlarge (use -i simlarge in the YAML file) for these jobs. Use the native dataset (-i native) for dedup and blackscholes jobs. The .yaml files provided have the correct configuration.* Once you are done with running one experiment, make sure to terminate the started jobs. You can terminate them all together using:

```
$ kubectl delete jobs --all
$ kubectl delete pods --all
```

Alternatively, you can do so one-by-one using the following command:

```
$ kubectl delete job <job_name>
```

IMPORTANT: you must delete your cluster when you are not using it! Otherwise, you will easily use up all of your cloud credits! When you are ready to work on the project, you can easily re-launch the cluster with the instructions above. To delete your cluster, use the command:

```
$ kops delete cluster part2a.k8s.local --yes
```

2.1.2 PARSEC Parallel Behavior

For the second half of Part 2, you will have to look into the parallel behavior of PARSEC, more specifically, how does the performance of various jobs in PARSEC change as more threads are added (more specifically 1, 2, 4 and 8 threads). For this part of the study, no interference is used.

You will first have to spawn a cluster as in section 2.1.1, however, this time use the `part2b.yaml` file we provided (make sure to update the file with values for your GCP project). Once more, this will be a single node cluster with an 8 CPU VM. You will have to vary the number of threads for each PARSEC job. To do so, change the value of the `-n` parameter in the relevant `yaml` files. You should run all these benchmarks with the *native* dataset. The corresponding .yaml files are in `parsec-benchmarks/part2b` of the github repo.

Other relevant instructions for this task can be found in section 2.1.1.

IMPORTANT: you must delete your cluster when you are not using it! Otherwise, you will easily use up all of your cloud credits! When you are ready to work on the project, you can easily re-launch the cluster with the instructions above. To delete your cluster, use the command:

```
$ kops delete cluster part2b.k8s.local --yes
```

2.2 Questions

Please answer the following questions in the report you submit.

1. **[12 points]** Using a Kubernetes cluster with a single 2-core node, fill in the following table with the normalized execution time of each batch job with each source of interference. The execution time should be normalized to the job's execution time with no interference. For the execution time, consider the compute time only (not the initialization time) from the PARSEC container logs. Color-code each field in the table as follows: green if the normalized execution time is less than or equal to 1.3, yellow if the normalized execution time is greater than 1.3 and up to and including 2, and red if the normalized execution time is greater than 2. The setup for this question is detailed in Section 2.1.1.

Workload	none	cpu	l1d	l1i	l2	llc	memBW
dedup	1.0						
blackscholes	1.0						
ferret	1.0						
freqmine	1.0						
canneal	1.0						
fft	1.0						

   Summarize in a paragraph the resource interference sensitivity of each batch job.

2. **[3 points]** What does the interference profile table tell you about the resource requirements for each application? Which jobs (if any) seem like good candidates to collocate with memcached from Part 1, without violating the SLO of 2 ms P95 latency at 40K QPS?

3. **[10 points]** In a Kubernetes cluster with a single 8-core node, run each of the six batch jobs individually and measure their execution time. Note that you should **not** use any interference for this part of the study. Vary the number of threads (1, 2, 4, and 8). Plot a single line graph

with the speedup on the y-axis (normalized time to the single thread performance: $\frac{t_1}{t_n}$ where t_i is the execution time for i threads) and the number of threads on the x-axis. Briefly discuss the scalability of each application, mentioning if it is linear, sub-linear or super-linear. Which of the applications, if any, gain a significant speedup with more threads? Explain what you consider to be "significant". The setup for this question is detailed in Section 2.1.2.

3 FAQ

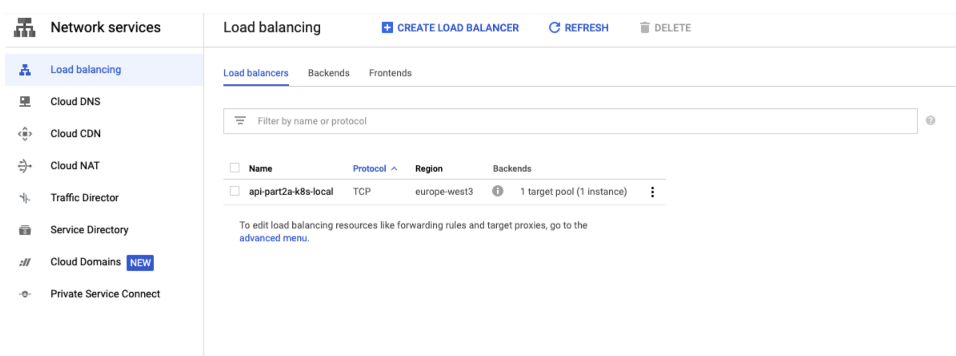
- When running kops create, if you get the following error: failed to create file as already exists: gs://cca-eth-2021-group-XXX-ethzid/part1.k8s.local/config. error: error creating cluster: file already exists, you need to delete the contents of your Google Cloud storage bucket, the recreate it with the following commands:

```
$ gsutil rm -r gs://cca-eth-2021-group-XXX-ethzid/
$ gsutil mb gs://cca-eth-2021-group-XXX-ethzid/
```

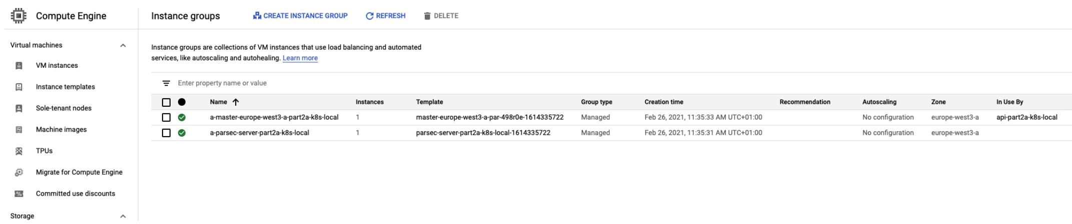
- When ssh-ing into a cluster node, if you get an error like
  WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!
  ...
  Offending ED25519 key in /Users/username/.ssh/known_hosts:9
  ...
  Host key verification failed
  then you need to run ssh-keygen -R <host> where <host> is the IP address of the server you want to access.

- If kubectl commands prompt you for a username and password, delete the cluster and recreate it from scratch.

- If for any reason you cannot delete the cluster with the kops command do the following:
  - Go to console.cloud.google.com
  - Type in the search bar the term "Load balancers". You should be redirected to a page similar to the one below:



  - Select and delete the load balancer.
  - Then type in the search bar the term "Instance groups". You should be redirected to a page similar to the one below:



- Select and delete all the instance groups.
- Delete your Google Cloud storage bucket by typing:
  `$ gsutil rm -r gs://cca-eth-2021-group-XXX-ethzid/`