CS5052 Data-Intensive System Practical Report

Cloud Benchmarking for Container-based Applications

SID 180024570

25/04/2019

Contents

1 Objective

This report compares a bunch of cloud instances on Google Cloud Platform (GCP) and recommends one with the highest processing speed and the lowest average cost for a container-based application. The container-based application is a BigQuery controller that streams tweets data from Redis into Google BigQuery.

2 Method

2.1 Application Containerisation

The application by Google Inc. (2017) is a 'pipeline' to stream Twitter data into BigQuery via Redis and will be containerised into three deployments with Kubernetes. The three deployments are

- *twitter-stream*, a pod that runs Python scripts that fetch data from Twitter and dumps tweets into Redis.
- *redis-master*, a pod that runs Redis to buffer data coming in from Twitter.
- *bigquery-controller*, pods that group data in small batches and stream them into BigQuery. Unlike the other two deployments, there can will be 3 replicas running in parallel to increase data throughput.

2.2 Instances

Candidates are 11 types of VM instance with 1, 2, and 4 virtual CPUs and memory capacity varies from 1.7 to 26 GB. Instances are all based in the default region, us-central1 (lowa), which is also cheaper than most of other regions. Details are listed in table 1.

Table 1: Candidate VM Instances

Instance Type	vCPUs	Memory (GB)	Price ($/hour)
g1-small	1	1.70	0.025700
n1-stantard-1	1	3.75	0.047500
customized-1	1	2.50	0.044289
n1-stantard-2	2	7.50	0.095000
n1-highcpu-2	2	1.80	0.070900
n1-highmem-2	2	13.00	0.118400
customized-2	2	3.00	0.079686
n1-standard-4	4	15.00	0.190000
n1-highcpu-4	4	3.60	0.141800
n1-highmem-4	4	26.00	0.236800
customized-4	4	6.00	0.159372

2.3 CPU Benchmarking

The benchmark tool utilised to evaluate the CPU speed of candidate instances is *SysBench* version 1.0.15 by Kopytov (2018), which is also containerised as a docker image by Docker Hub (2018) and will be running as a Kubernetes job on each type of instance. Specifically, the CPU benchmarking time is set to be 10 seconds for instantaneous performance and 15 minutes for long-term performance. The instances with multiple CPUs are benchmarked with the same number of threads. The speed metric is the number of events processed per second. See appendix for benchmark commands.

The benchmark results are listed in table 2 above, where "overall" is the performance of the whole VM and "per CPU" is "overall" divided by the CPU number of the VM.

Table 2: CPU Performance of Candidate Instances

		Instantaneous speed (10s)		Long-term speed (15m)	
Instance Type	vCPUs	overall	per CPU	overall	per CPU
g1-small	1	308.78	308.78	194.01	194.01
n1-stantard-1	1	333.55	333.55	338.29	338.29
customized-1	1	332.25	332.25	336.00	336.00
n1-stantard-2	2	546.04	273.02	523.10	261.55
n1-highcpu-2	2	534.75	267.38	520.36	260.18
n1-highmem-2	2	523.34	261.67	524.31	262.15
customized-2	2	532.71	266.36	513.45	256.73
n1-standard-4	4	1058.63	264.66	1056.99	264.25
n1-highcpu-4	4	1064.58	266.14	1066.89	266.72
n1-highmem-4	4	1025.98	256.50	1052.90	263.23
customized-4	4	1066.79	266.70	1070.03	267.51

Unit: events per second

3 Evaluation

3.1 Performance of Single vCPU

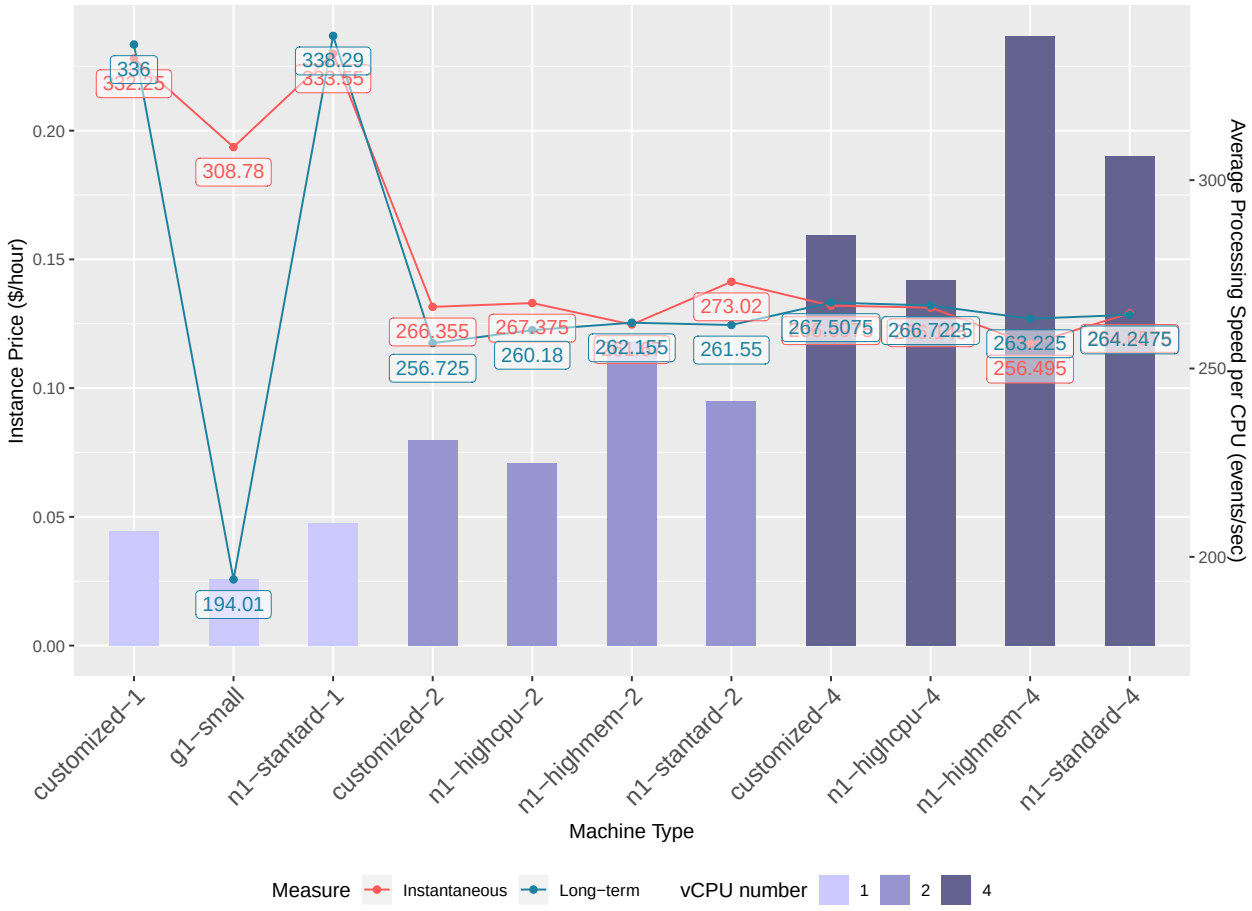

Figure 1: Instance Hourly Price and Average Speed of a Single vCPU

Figure 1 depicts the hourly prices of all instances and their speed of a single vCPU on average. It shows that the more the vCPUs, the higher the instance price, and among instances with the same number of vCPU, the high memory instance is clearly the most expensive type but does not show the fastest single vCPU processing speed.

The high memory instance shows the slowest speed in the group of 4 vCPU type for both instantaneous and longterm performance. However, the differences are negligibly small and might be caused by randomness. Overall, the average speed of multi-CPU types are basically in the same level. In other words, the performance scales linearly with the number of vCPU, except single vCPU type.

Surprisingly, the instances with single vCPU, also the cheapest group, appear to have considerably higher average processing speed. While the performance of the cheapest type *g1-small* is very unstable, showing largely difference between instantaneous and long-term processing speed.

3.2 Performance of Multi-vCPU Equivalent

To compare the performance and costs of the same vCPU resource for all types, here use four vCPUs equivalent as example, assuming clusters of four single-vCPU instances and clusters of two 2-vCPU instances. Figure 2 illustrates the hourly cost and processing speed for all types, ordered by the cost from left (cheap) to right (expensive).

The instances *customized-1* and *n1-standard-1*, which show disctinctly good performance and stability in figure 1, also show fair costs with equivalent vCPU resource.

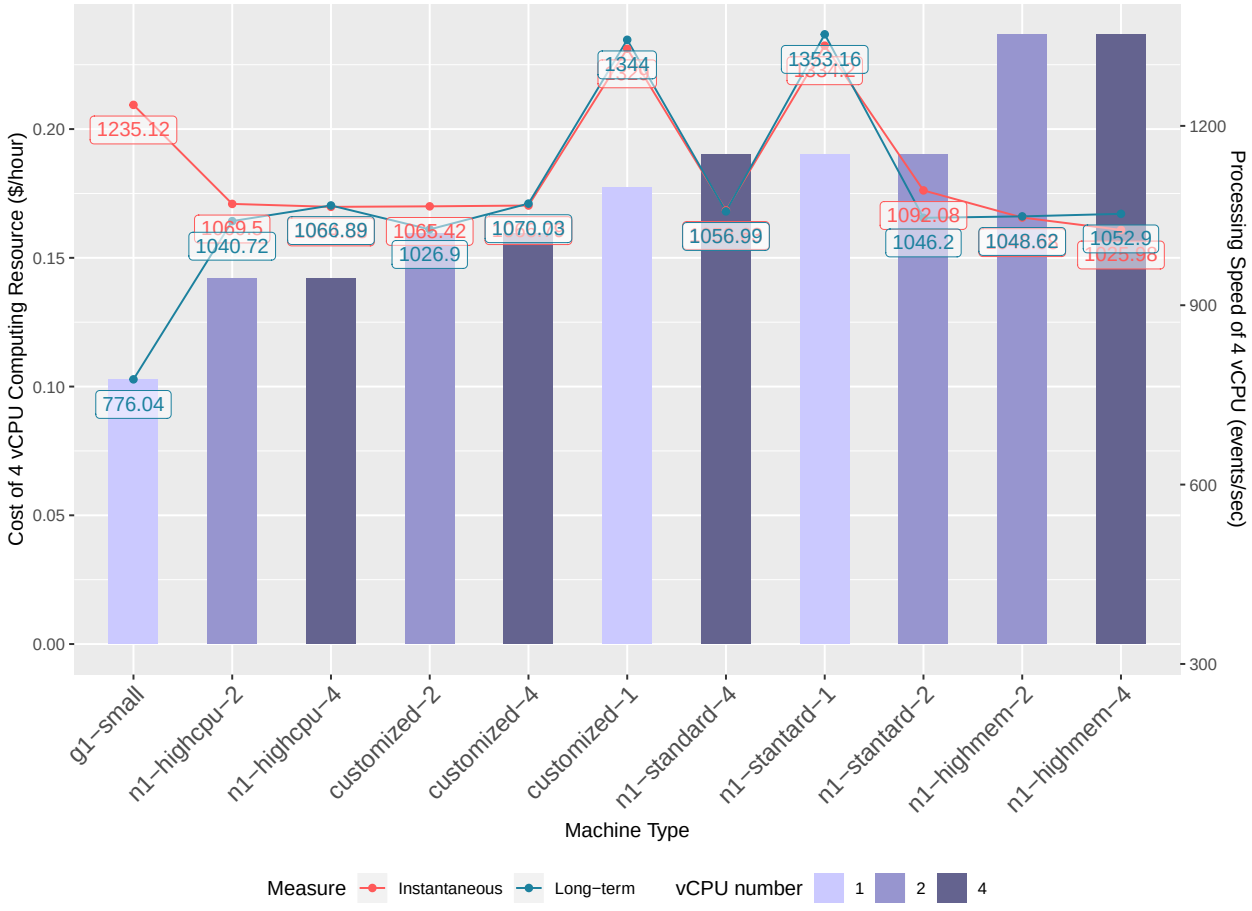

Figure 2: Processing Speed of 4 vCPUs Equivalent, Ordered By Hourly Cost

4 Conclusion

The evaluation demonstrates that, on Google Cloud Platform, renting multiple instances of single-vCPU type (*customized-1* and *n1-standard-1*) can achieve relatively better performance and lower cost than renting one/multiple multi-vCPU instances with equivalent resource. Especially for container-based applications, one can run replicas across several single-core instance nodes via Kubernetes, instead of placing all on fewer multi-vCPU instances, to save cost and guarantee performance at the same time.

Between the two candidates with the best performance, *customized-1* and *n1-standard-1*, *customized-1* is the more recommended. Because it is cheaper than *n1-standard-1* and there are discount for long-term commitment (1 year and 3 year). In addition, one can also adjust memory capacity on demand.

5 Reference

Docker Hub (2018) SysBench docker image. *Docker Hub*. [Online]. Available from: https://hub.docker.com/r/severalnines/sysbench/.

Google Inc. (2017) Real-time data analysis using kubernetes, redis, and bigquery. *GitHub repository*. [Online]. Available from: https://github.com/GoogleCloudPlatform/kubernetes-bigquery-python/tree/master/redis.

Kopytov, A. (2018) SysBench. *GitHub repository*. [Online]. Available from: https://github.com/akopytov/sysbench.

6 Appendix

6.1 SysBench Command

```
# Instantaneous performance
sysbench cpu --cpu-max-prime=20000 --threads=1 --time=10 run
sysbench cpu --cpu-max-prime=20000 --threads=2 --time=10 run
sysbench cpu --cpu-max-prime=20000 --threads=4 --time=10 run

# Long-term performance
sysbench cpu --cpu-max-prime=20000 --threads=1 --time=900 run
sysbench cpu --cpu-max-prime=20000 --threads=2 --time=900 run
sysbench cpu --cpu-max-prime=20000 --threads=4 --time=900 run
```

6.2 Source Code of Figures and Tables

```
knitr::opts_chunk$set(echo = FALSE,
                      include = TRUE,
                      message = FALSE)

library(tidyverse)
library(knitr)
library(kableExtra)
options(kableExtra.latex.load_packages = FALSE,
        knitr.kable.NA = '-')

instance <- read_csv("instances.csv")
instance <- instance %>% arrange(cpu) %>% select(type, cpu, mem, price)
kable(instance,
      format = "latex",
      booktabs = T,linesep = "",
      col.names = c('Instance Type','vCPUs','Memory (GB)','Price ($/hour)'),
      caption = "Candidate VM Instances") %>%
  kable_styling(latex_options = c("striped", "hold_position"), stripe_index =c(1:3,8:11))
instance <- read_csv("instances.csv")
instance <- instance %>% arrange(cpu) %>%
  mutate(instantpp=instant/cpu, longtermpp=longterm/cpu) %>%
  select(type, cpu, instant, instantpp, longterm, longtermpp)

kable(instance,
      format = "latex", digits = 2,
      booktabs = T,linesep = "",
      col.names = c('Instance Type','vCPUs','overall','per CPU','overall', 'per CPU'),
      caption = "CPU Performance of Candidate Instances") %>%
  add_header_above(c(" "=2, "Instantaneous speed (10s)"=2, "Long-term speed (15m)"=2)) %>%
  footnote(general = "events per second",
           general_title = "Unit:",footnote_as_chunk = T) %>%
  kable_styling(latex_options = c("striped", "hold_position"), stripe_index =c(1:3,8:11))
bench <- read_csv("instances.csv")
bench <- bench %>% arrange(cpu) %>%
```

```
  mutate(instantpp=instant/cpu, longtermpp=longterm/cpu) %>%
  select(type, cpu, price, instant, instantpp, longterm, longtermpp) %>%
  gather(key = "measure", value = "speed", instantpp, longtermpp)


y1 <- min(bench$price)
y2 <- max(bench$price)
x1 <- min(bench$speed)
x2 <- max(bench$speed)
b <- (y2 - y1) / (x2 - x1)
a <- y1 - b * x1

bench %>%
  mutate(scaled = a + b * speed, cpug= factor(cpu)) %>%
  ggplot(aes(x = reorder(type, cpu), y = price, fill = cpug, group = cpug)) +
  geom_col(position = "dodge", width=0.5) +
  geom_line(aes(x = reorder(type, cpu), y = scaled, color = measure, group = measure)) +
  geom_point(aes(x = reorder(type, cpu), y = scaled, color= measure, group = measure)) +
  geom_label(
        aes(reorder(type, cpu), scaled, label = speed, color = measure,
            fill = NULL, group = measure),
        alpha = 0.5, vjust = 1.4, show.legend = F) +
  scale_y_continuous(name = "Instance Price ($/hour)",
    sec.axis = sec_axis(~ (. - a) / b, name = "Average Processing Speed per CPU (events/sec)")) +
  labs(x = "Machine Type", fill = "vCPU") +
  scale_color_manual(name= "Measure",
                     labels=c("Instantaneous", "Long-term"),
                     values = c("#ff5959", "#1c819e")) +
  scale_fill_manual(name = "vCPU number",
                    values = c("#cbc9ff","#9795cf","#64638f")) +
  guides(fill = guide_legend(override.aes = list(shape = NA))) +
  theme(axis.text.x.bottom = element_text(angle = 45, hjust = 1,size = 13),
        legend.position = "bottom")

bench <- read_csv("instances.csv")
bench$price[bench$cpu==1] <- bench$price[bench$cpu==1]*4
bench$price[bench$cpu==2] <- bench$price[bench$cpu==2]*2
bench$instant[bench$cpu==1] <- bench$instant[bench$cpu==1]*4
bench$instant[bench$cpu==2] <- bench$instant[bench$cpu==2]*2
bench$longterm[bench$cpu==1] <- bench$longterm[bench$cpu==1]*4
bench$longterm[bench$cpu==2] <- bench$longterm[bench$cpu==2]*2

bench <- bench %>% arrange(cpu) %>%
  mutate(instantpp=instant/cpu, longtermpp=longterm/cpu) %>%
  select(type, cpu, price, instant,longterm) %>%
  gather(key = "measure", value = "speed", instant, longterm)


y1 <- min(bench$price)
y2 <- max(bench$price)
x1 <- min(bench$speed)
x2 <- max(bench$speed)
b <- (y2 - y1) / (x2 - x1)
```

```
a <- y1 - b * x1

bench %>%
  mutate(scaled = a + b * speed, cpug= factor(cpu)) %>%
  ggplot(aes(x = reorder(type, price), y = price, fill = cpug, group = cpug)) +
  geom_col(position = "dodge", width=0.5) +
  geom_line(aes(x = reorder(type, price), y = scaled, color = measure, group = measure)) +
  geom_point(aes(x = reorder(type, price), y = scaled, color= measure, group = measure)) +
  geom_label(
        aes(reorder(type, price), scaled, label = speed, color = measure,
            fill = NULL, group = measure),
        alpha = 0.5, vjust = 1.4, show.legend = F) +
  scale_y_continuous(name = "Cost of 4 vCPU Computing Resource ($/hour)",
    sec.axis = sec_axis(~ (. - a) / b, name = "Processing Speed of 4 vCPU (events/sec)")) +
  labs(x = "Machine Type", fill = "vCPU") +
  scale_color_manual(name= "Measure",
                     labels=c("Instantaneous", "Long-term"),
                     values = c("#ff5959", "#1c819e")) +
  scale_fill_manual(name = "vCPU number",
                    values = c("#cbc9ff","#9795cf","#64638f")) +
  guides(fill = guide_legend(override.aes = list(shape = NA))) +
  theme(axis.text.x.bottom = element_text(angle = 45, hjust = 1,size = 13),
        legend.position = "bottom")
```