Problem Framing

Other studies have focused on finding the best cloud configurations for recurring batch jobs, based off a user supplied workload (Yadwadkar, et al., 2017; Alipourfard, et al., 2017). This is not necessarily helpful for a common use case for cloud deployments: Running a webserver expected to deliver an end-user service rapidly and effectively. In this case, a user most likely wants the configuration that provides the best service (e.g. lowest latency) for the price per hour of uptime, rather than the lowest cost cluster to perform a specified workload. While mimicking end-user requests with workloads is possible, this may sometimes be based on false assumptions, such as optimal scheduling from prior knowledge of an entire job trace, lack of limiting network bandwidth, and non-stochastic request intensity. Other studies have looked at benchmarking a container-based application itself on different virtual machine configurations (Varghese, et al., 2016), but do not examine how these containers would perform under a stochastic set of end-user requests.

The aim of the prototype developed here was to explore and assess the performance of different configurations for any web-based Kubernetes deployment that would receive end-user requests. It should be generalisable to any user-defined container or metric to determine the best cluster.

Prototyping

Algorithm 1 *Configuration Exploration* Algorithm

```
1:    Procedure CONFIGEXPLORE(testConfigs, TestDeployment,
      PingDeployment, logsToData, Analysis, PingCluster)
2:       while True:
3:          for config in testConfigs:
4:             TestCluster <- MakeCluster(config)
5:             Run TestDeployment(TestCluster)
6:             TestDeploymentIP <- TestCluster.ExternalIP
7:             Run PingDeployment(PingCluster, TestDeploymentIP)
8:             While isActive(PingDeployment)
9:                DeploymentLogs <- getLogs(TestCluster)
10:            PingLogs <- getLogs(PingCluster)
11:            Data.append(logsToDataProc(PingLog, DeploymentLogs))
12:         testConfigs <- Run Analysis(Data)
13:         if testConfigs is empty:
14:            break;
```

The *Configuration Exploration* algorithm would utilise a given PingCluster and the following user-supplied files:

- ***testConfigs*** – A headerless csv file of the configs to immediately sample from
- ***TestDeployment*** – A start-up script and associated yaml file to deploy the container-based application being tested

- ***PingDeployment*** – A start-up script and associated yaml file to deploy the container replicating user's job requests to the test deployment and recording the latency. Would be passed the External IP of the tested deployment
- ***LogsToDataProc*** – A script to transform the collected logs into a readable form, and append it to a file of all the relevant data collected so far
- ***AnalysisProc*** – A script to use the data collected so far to work out what configurations are left to test

For each configuration in the original testConfig file, a test cluster would be created, the application being tested then deployed on it, and a Ping cluster set up to generate logs based on pings to the test cluster representing simulated end-user requests. The logs would be collated and passed through a user-supplied analysis script to determine what, if any, further cluster configurations need to be explored.

To make the solution as universal as possible, the prototype was developed to be run entirely on the Google Cloud Shell. No authentication or configuration of non-user-supplied files is required by the user. The git repository would be cloned into one's Google Cloud Shell console, the relevant deployment files and user-supplied analysis scripts would be copied into the root folder, and the main script run.

A prototyped implementation of Algorithm 1 is provided in the Github linked in the appendix and is evaluated below.

Evaluation

For evaluation, a container for a simple Go based webserver was built which computed for end-users the highest Prime number up to a given value passed as part of the url. The algorithm used was intentionally slow to give exactly $O(n^2)$ complexity. The following arguments were used:

- ***testConfigs*** – See appendix
- ***TestDeployment*** – Yaml of the webserver and a single while loop to expose deployment and wait for an External IP.
- ***PingDeployment*** – A script to set up a Daemonset that repeatedly curls a given URL with normally distributed numbers around 10,000, and logs the URL used and time taken.
- ***LogsToDataProc*** – A python script to extract input number and time taken from ping logs.
- ***AnalysisProc*** – No further analysis was performed.
- ***PingCluster*** – A 4-node cluster of g1-small machines with disk size of 30GB in europe-west1-b

As mentioned, no analysis was provided for the example case, simply a set of configurations to test and collect data from. Figures 1 and 2 show plots of the results obtained, while Table 1 and Figure 3 show details of a model derived from the collected data which could be used to decide on an optimal configuration.

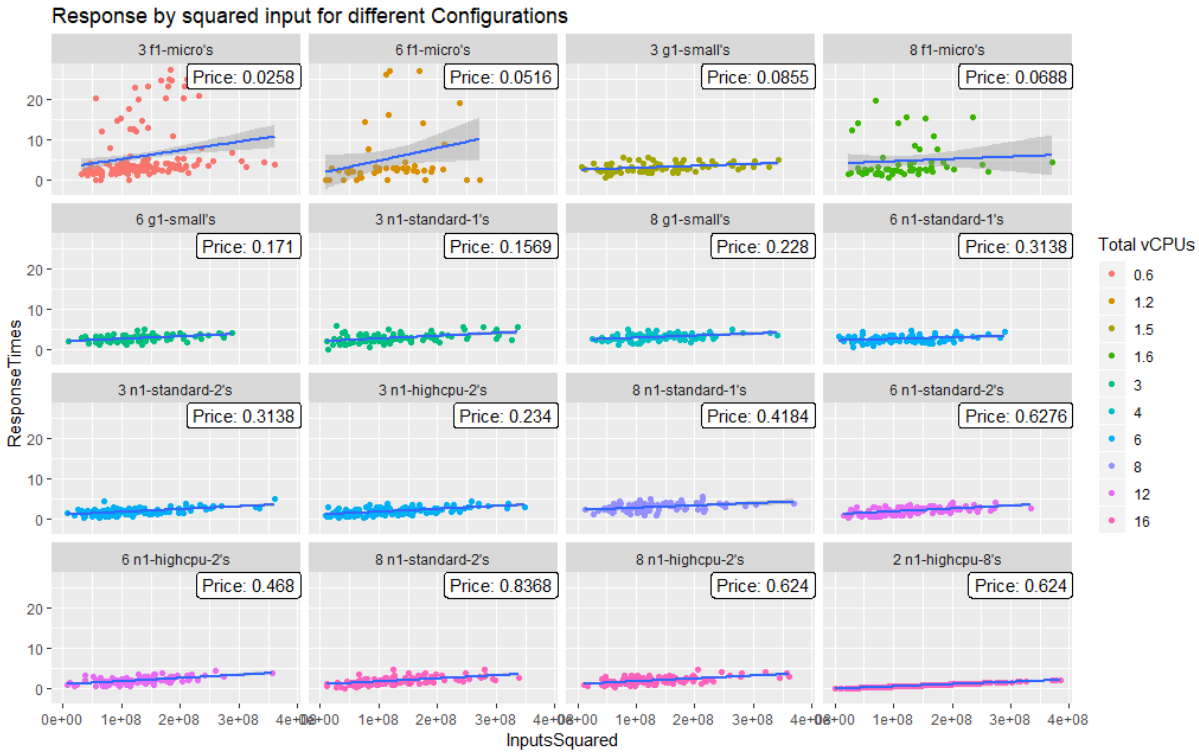

Figure 1. Plots of the total time taken by the curl command against the square of the input number given in the URL, across different configurations, coloured by total number of vCPUs. The hourly price of the cluster is provided. Lines and 95% confidence intervals overlaid are from linear models.

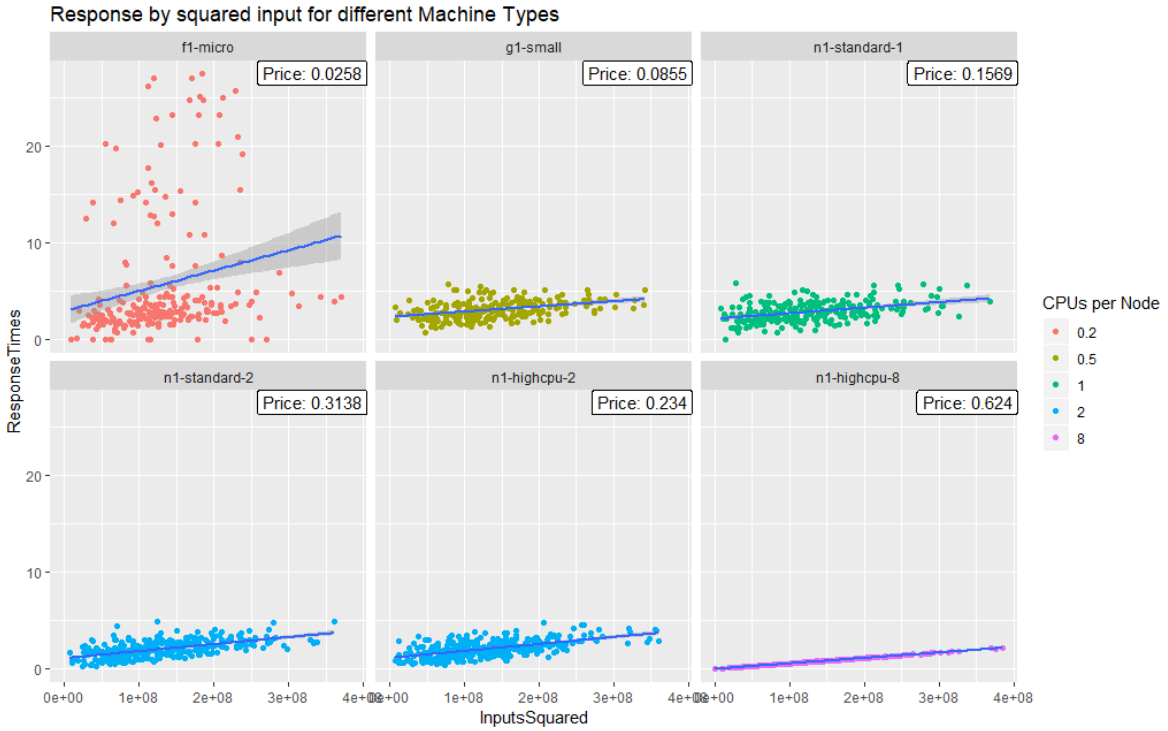

Figure 2. Plots of the total time taken by the curl command against the square of the input number given in the URL, across different Machine Types, coloured by vCPU per node of that machine type. The hourly price of three nodes of the given machine type is provided. Lines and 95% confidence intervals overlaid are from linear models.

To interpret relationships between the configurations and the main relationship between Response Time and and Input, a linear model was estimated based on formula 1, where n is the input number and CPU is a value determined by the machine type used. n1-standard-1 was used as a baseline to compare CPU values. Due to the nature of shared vCPUs in f1-micro and g1-small machines, these machine types were excluded from the model to improve other estimates, although plots show that only f1-micro machines had particularly high variability in latency results.

Formula 1 $$ResponseTime = NetworkDelay + CPU.n^2$$

While it is hard to tell from Figures 1 and 2, the Go webserver used did not benefit from having additional nodes in its cluster, as no parallelisation between machines was utilised. This lack of interaction of input and number of nodes with the response time was shown in the model ($F = 0.1335$, $p = 0.9401$). Table 1 shows the estimates from the model in practical terms. The Network delay was shown to be almost exactly 1 second (1.055s) in the model, which ended up meaning that any improvement in CPU had almost no practical effect on end-user latency. As only CPU speed was important in this example, n1-highcpu-2 gave significantly better value for money than n1-standard-2. n1-highcpu-8 was both extremely quick and had almost no variation in response time, however it was 6 times more expensive than n1-standard-1.

Machine Type	Change in latency $*10^{-8}$ per $n^2 \pm$ standard error	Proportion of Latency before network delay	Proportion of Latency after network delay	Hourly Price Increase	Price Increase Ratio
n1-standard-1	+0.000 ± 0.0412	1.000	1.000	+0.00	1.00
n1-standard-2	-0.462 ± 0.0423	0.623	1.000	+0.0523	2.00
n1-highcpu-2	-0.453 ± 0.0420	0.630	1.000	+0.0257	1.49
n1-highcpu-8	-1.276 ± 0.0427	-0.042*	1.000	+0.2597	5.97

*Table 1. The effects of different machine-types on latency of end-user requests from ping cluster. * Negative value results from inccorect estimation by model, suffice to say that the value was likely very low.*

Figure 3 shows that the distribution of residuals resulting from this model is positively skewed, even after the removal of f1-micro and g1-small machine-types. The residuals were shown to be non-normal ($W=0.961$, $p<<0.001$).

It is also worth noting that this process, despite only using minute-long pinging traces, took an extended period of time due to the lengthy cluster creation and deletion durations. However, there is no reason that all the clusters that are to be used could not be created simultaneously, and simply utilised one by one, dramatically

speeding up the process. Multiple pinging clusters could even be used to allow traces to be performed simultaneously.

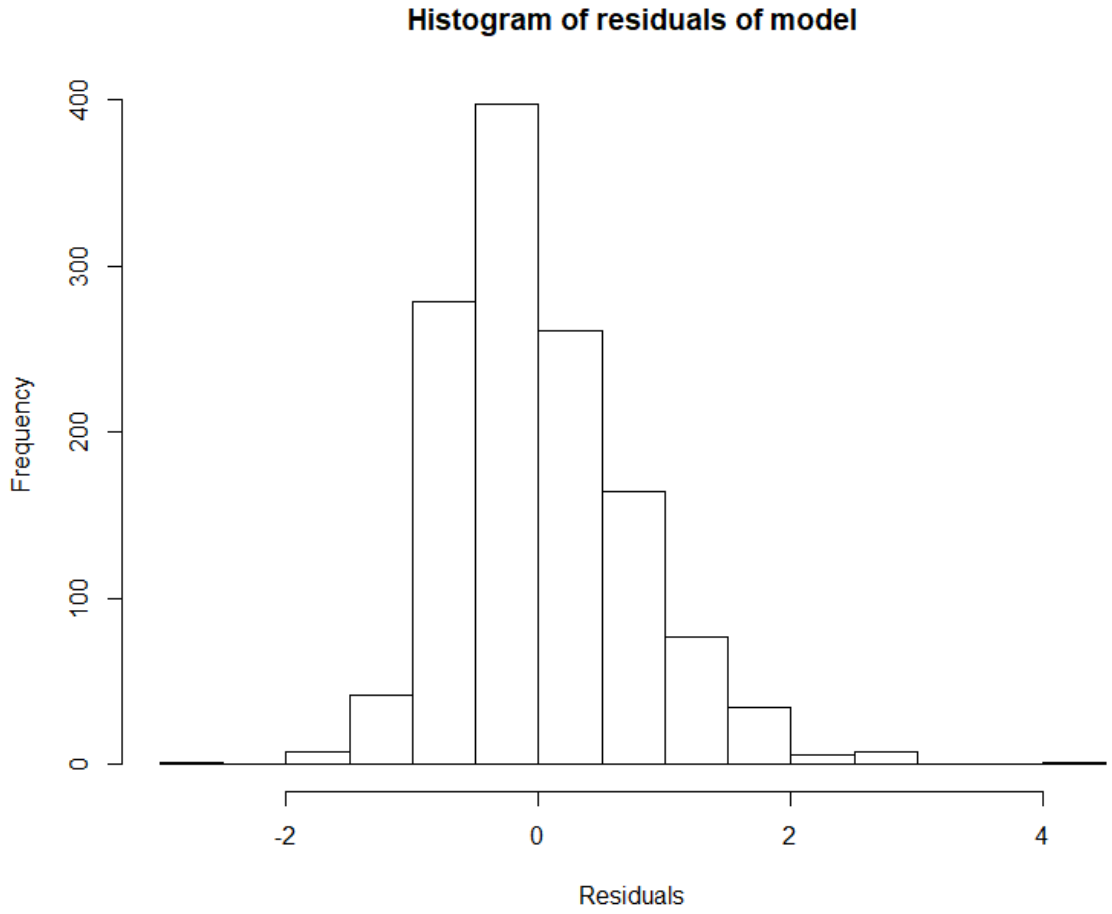

Figure 3. Histogram of residuals from the model used, showing a positively skewed distribution.

Appendix

Github - https://github.com/briggsby/cs5052practical

Testconfigs - Configurations tested for the evaluation:

Disk Size / GB	Disk Type	Machine Type	Nodes	Zone	Price	Ping trace duration
30	pd-standard	f1-micro	3	europe-west2-a	0.0285	60
30	pd-standard	f1-micro	6	europe-west2-a	0.0285	60
30	pd-standard	f1-micro	8	europe-west2-a	0.0285	60
30	pd-standard	g1-small	3	europe-west2-a	0.0285	60
30	pd-standard	g1-small	6	europe-west2-a	0.0285	60
30	pd-standard	g1-small	8	europe-west2-a	0.0285	60
30	pd-standard	n1-standard-1	3	europe-west2-a	0.0523	60
30	pd-standard	n1-standard-1	6	europe-west2-a	0.0523	60
30	pd-standard	n1-standard-1	8	europe-west2-a	0.0523	60
30	pd-standard	n1-standard-2	3	europe-west2-a	0.1046	60
30	pd-standard	n1-standard-2	6	europe-west2-a	0.1046	60
30	pd-standard	n1-standard-2	8	europe-west2-a	0.1046	60
30	pd-standard	n1-highcpu-2	3	europe-west2-a	0.078	60
30	pd-standard	n1-highcpu-2	6	europe-west2-a	0.078	60
30	pd-standard	n1-highcpu-2	8	europe-west2-a	0.078	60
30	pd-standard	n1-highcpu-8	2	europe-west2-a	0.312	60

References

Alipourfard, O. et al., 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation.*

Varghese, B., Subba, L. T., Thai, L. & Barker, A., 2016. Container-Based Cloud Virtual Machine Benchmarking. *2016 IEEE International Conference on Cloud Engineering.*

Yadwadkar, N. J., Hariharan, B. & Gonzalez, J. E., 2017. Selecting the Best VM across Multiple Public Clouds: A Data-Driven Performance Modeling Approach. *Proceedings of the 2017 Symposium on Cloud Computing,* pp. 452-465.