Analyzing the Differences between NYU Prince Cluster and Google Cloud

Alankrith Krishnan, Ruining Sun, Xifei Li

New York University

{ak7380, rs6565, xl2715}@nyu.edu

Abstract

The idea behind this project is to analyze and test all the differences between NYU's Prince HPC Cluster and Google Cloud from a general student's perspective – this includes performance differences, ease of access, UI/UX experience, wait time to acquire resources, and differences in raw hardware and storage. Analyzing these differences will give us a fair idea of which aspects of Google Cloud are better or worse than the Prince Cluster and will help the NYU IT team realize whether migrating to Google Cloud from our existing cluster is a viable option. Analyzing the performance differences can also help with optimizing the Prince Cluster more, based on the areas where loss of performance is seen.

Keywords: User Interface, Performance, Cluster, Cloud, Hardware

Table of Contents

Introduction

The Prince Cluster serves as our main source of compute here at NYU, and NYU's IT department puts in a lot of resources and time into making the cluster functional and operate normally, which includes addition of new hardware, monthly maintenance, and managing user access. Now, we look at public offerings (in this case, Google Cloud) and see if there are any differences or improvements in porting our High-Performance Computing needs directly to a public cloud instead, and if the cost differences are worth the change overall.

The contributions of each member in this project are mostly even. The analysis of User Interface and Access was managed by all of us (based on our feelings of the User Experience in the initial weeks of testing). Performance testing was split based on area of research – Computer Vision (Ruining Sun), Natural Language Processing (Xifei Li), Benchmarking and Multi-GPU tests (Alankrith Krishnan). Storage related differences were handled by Xifei, Access Time Analysis and other Google Cloud Features by Alankrith, and Hardware and Cost Differences by Ruining.

Methodology

We split our tests into multiple sections:

1. Ease of Access – User Interface and Jupyter Notebook Access
2. Storage Comparisons – Disks and Buckets
3. Hardware Comparisons – CPUs and GPUs
4. Access Times – Requesting resources at different times on multiple days
5. Cost Comparisons – Total cost (theoretical estimate)
6. Performance Testing – Computer Vision, Natural Language Processing, Benchmark and Multi-GPU Tests

All tests were conducted in isolated environments, with no elevated permissions to make sure it is as comparable to a standard user as possible. Separate instances were created for each job on Google Cloud, and GPUs were requested directly for each job on Prince. Environments were isolated as well, with each job having its own Anaconda Environment to avoid any kind of dependency issues.

Ease of Access

Ease of Access is determined by the experience the user has during their time on the either cloud. Let us look at each one.

Prince

Login

Logging into Prince is a terminal based process, since there is no GUI linked to Prince. Depending on whether you are on the NYU network, you might need to go through a Bastion Host/VPN to log in – multiple hops.

Requesting Resources

Requesting resources happens through the Slurm workload manager, which has two different options to run jobs – srun and sbatch. srun is used to request nodes for interactive access, and sbatch is used to submit batch jobs that run on a node when it is allocated. The commands for these are non-trivial – they need you to know the exact arguments and flags which are available from examples on the NYU Prince wiki.

File Transfer

SCP is straightforward, as it is like SSH, except with the path to the files to be copied. (scp <source> <destination>)

Jupyter Notebook Access

Running the notebook server and accessing it is not as simple, since you need to sbatch a specific jupyter script and forward the ports to your local system before accessing the notebook on the web browser. This takes a while to figure out and set up for the first time, after which most people just script the process instead.

Google Cloud

Login

Login is as simple as logging into your Google account and accessing the Google Cloud Console and setting the correct project. No hassles or multiple hops of any sort.

Requesting Resources

Google Cloud allows you to create VM's from the GUI directly, with the specifications being clear and completely available on the instance creation page. You choose what you need and boot up the instance. After the instance is created, you can SSH to the instance in multiple ways – directly through the browser or through the GCloud SDK. Connecting through the GCloud SDK directly has a command on the GUI as well.

File Transfer

Once you have the command, you can replace SSH with SCP and add a source path (like SCP on Prince), so there is no difference in file transfer. If you do choose to upload directly through the web browser however, you can just choose upload file on the browser and choose your file to upload it without SSH.

Jupyter Notebook Access

Jupyter notebook can be created and tied to an instance under the 'AI Platform – Notebooks' section. It allows you to create a notebook that comes with your favorite Deep Learning Framework and creates a JupyterLab proxy for direct notebook access.

Overall, the Ease of Access section seems to favor Google Cloud here – since most of the work (other than running the code itself of course) is seamlessly done through the GUI itself.

Storage Comparisons

In this section, we look at the differences in storage capability and availability across the two clouds. On Prince, we are offered access to only disks with different filesystems, and on Google Cloud, we are offered disks on demand, along with object store buckets as well on demand.

The quota on Prince for disks is generous – 20GB/user on home, 5TB/user on scratch, 2TB/user on beegfs and 2TB/user on archive. On Google Cloud, disks offered are tied to an instance directly. We also get separate persistent disks for storage. All of the disks come with an option of being either standard hard disks or SSD's, so read/write performance can be boosted by using an SSD instead if necessary.

Buckets are not available on Prince, although Google Cloud offers them. But since there is usually a latency with respect to accessing data from buckets and writing to them, we do not consider them for our testing purposes in this report.

Overall, both Prince and Google Cloud have similar storage capabilities. Google Cloud can scale on demand however and has SSD support – which can be used if necessary, for certain jobs.

Hardware Comparisons

Hardware offerings on Prince and Google Cloud are mostly similar, and we will categorize them into CPUs and GPUs

Hardware Comparisons Table

Hardware	Prince	Google Cloud
CPU Architecture	Broadwell	Broadwell
	Haswell	Haswell
	Skylake	Skylake
	Ivy Bridge	**Sandy Bridge**
GPUs	NVIDIA V100	NVIDIA V100
	NVIDIA P100	NVIDIA P100
	NVIDIA K80	NVIDIA K80
	NVIDIA P40	**NVIDIA P4**
		NVIDIA T4
TPUs	**Not Provided**	**Provided**

CPU Architectures was used instead of CPUs since Google Cloud does not specify the CPUs, only guarantees the architecture of the CPU during instance creation. Looking at them, we can see that there aren't many big differences. Prince has Ivy Bridge instead of Sandy Bridge on Google Cloud, with very similar performance.

On the GPU front, we can see that the V100, P100 and K80 are the same. Prince has a P40, which is a GPU used for training, whereas Google Cloud has P4 and T4, GPUs that are used for virtual workstation (graphics) and inference. The P40 is expected to have higher performance than P4 and T4.

TPUs are made and managed only by Google and are not available on the Prince cluster. Since there is no direct comparison to be made here, we will be ignoring the use of TPUs in this report.

Access Times

Average access times on Google Cloud is always ~2 minutes (this includes VM setup and driver installation time). If resources are not enough, the VM fails to create and you can create the VM in a different zone instead, which only takes a minute to fill out the specifications for the new zone.

Access times on Prince vary based on the time of day and the day itself – Afternoons and Evenings are times when requesting resources takes a long time. During heavy days (like finals week), there is a wait time for almost all GPU resources, with high wait times during afternoons and evenings. Here is the full list:

Day	Time	Avg Access Time
Normal Days	Morning	Instant
	Afternoon	~2 minutes
	Evening	~5 minutes
	Night	Instant
Heavy Days (Finals Week)	Morning	~5 minutes
	Afternoon	~13 minutes
	Evening	~25 minutes
	Night	~20 minutes

Since the quota on Prince is limited, there is a queue for resource allocation whenever the requested resources are unavailable. On Google Cloud, it expands dynamically – and hence there are no wait times, you can provision an instance and get it up and running in very little time.

Cost Comparisons

When using Committed Use Discounts for 3 years, Google Cloud gives more than 50% off on most of its resources. When trying to put in a committed use reservation for the exact number of cores, GPU's and disks, these are the results.

1. 10,000 Cores + 60TB Memory = $188,000/month
2. K80 = Free (Does not increase monthly cost)
3. P100 = ~$16,000/month
4. V100 = ~$22,000/month
5. P4 (50 GPUs) = ~$10,000/month
6. T4 (50 GPUs) = ~$16,000/month
7. SSD storage disks (2.25 PB) = Free (Does not increase monthly cost)

This adds up to $252,000/month in charges for the entire month. This should be cheaper than Prince for the short term, but in the long term, the fact that NYU owns the hardware might make it cheaper than Google Cloud overall. Since we do not know the purchase, operation and maintenance costs for the cluster, we are going to leave that to the NYU IT team to find out whether the cost is feasible.

Performance Testing

Imagenet

Prince		Google Cloud	
K80	**89m 29s**	K80	92m 54s
P100	54m 25s	P100	**25m 18s**
V100	57m 8s	V100	**19m 31s**
P40	55m 38s	P4	59m 25s
		T4	**52m 55s**

10 epochs of training on PyTorch

GAN (MNIST)

Prince		Google Cloud	
K80	**32m 55s**	K80	36m 44s
P100	**10m 28s**	P100	12m 18s
V100	**7m 42s**	V100	8m 11s
P40	**15m 7s**	P4	24m 48s
		T4	20m 21s

50 epochs of training on TensorFlow

Glove (20newsgroups)

Prince		Google Cloud	
K80	**44m 53s**	K80	48m 47s
P100	**18m 12s**	P100	19m 23s
V100	**12m 44s**	V100	14m 21s
P40	**27m 37s**	P4	44m 51s
		T4	28m 52s

100 epochs of training on PyTorch

Sentiment Analysis (IMDB Listings)

Prince		Google Cloud	
K80	**34m 38s**	K80	40m 17s
P100	**14m 2s**	P100	23m 21s
V100	**19m 48s**	V100	21m 32s
P40	**20m 37s**	P4	29m 50s
		T4	24m 22s

50 epochs of training on PyTorch

Deep Learning Benchmark

Prince		Google Cloud	
K80	**8m 8s**	K80	8m 31s
P100	**3m 5s**	P100	3m 13s
V100	**2m 10s**	V100	2m 12s
P40	**3m 23s**	P4	4m 51s
		T4	3m 49s

20 epochs of training (without data) on TensorFlow and PyTorch

Multi GPU (Imagenet)

Prince		Google Cloud	
K80	8m 43s	K80	**4m 40s**
P100	8m 51s	P100	**3m 20s**
V100	9m 53s	V100	**3m 15s**
P40	8m 9s	P4	**3m 38s**
		T4	**3m 24s**

1 epoch of training on PyTorch (4 GPUs)

*Deep Learning Benchmark - https://github.com/u39kun/deep-learning-benchmark

Performance Discussions

Performance on baremetal should always be better than the cloud, but this isn't always the case. Particularly, Imagenet on Google Cloud seems to outperform Prince, in both the single GPU and multi GPU cases. The only difference between Imagenet and the other datasets (and the benchmark) is the data. The platforms are same for most cases, and testing was done in similar conditions for all of them. The difference we find is in the size of the dataset (more importantly, the size of each image). Each image is bigger and the dataset overall is bigger, so we went questions the data loading times on Prince and Google Cloud for Imagenet. Here are the screenshots of the results for the first epoch.

```
[ak7380@log-1 ak7380]$ cat imagenet-test.txt
=> creating model 'resnet18'
Epoch: [0][  0/391]     Time  7.893 ( 7.893)    Data  5.609 ( 5.609)    Loss 7.0042e+00 (7.0042e+00)    Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.00)
Epoch: [0][ 10/391]     Time  0.193 ( 1.289)    Data  0.000 ( 0.928)    Loss 6.5072e+00 (6.8158e+00)    Acc@1   1.17 (  0.57)   Acc@5   3.12 (  1.78)
Epoch: [0][ 20/391]     Time  3.473 ( 1.231)    Data  3.367 ( 0.967)    Loss 5.8508e+00 (6.4890e+00)    Acc@1   0.39 (  0.73)   Acc@5   2.34 (  2.68)
Epoch: [0][ 30/391]     Time  0.205 ( 1.102)    Data  0.021 ( 0.868)    Loss 5.4927e+00 (6.2147e+00)    Acc@1   1.56 (  0.92)   Acc@5   5.47 (  3.44)
Epoch: [0][ 40/391]     Time  3.172 ( 1.096)    Data  3.073 ( 0.884)    Loss 5.3485e+00 (6.0139e+00)    Acc@1   1.17 (  1.11)   Acc@5   6.64 (  4.19)
Epoch: [0][ 50/391]     Time  0.227 ( 1.044)    Data  0.136 ( 0.846)    Loss 5.1578e+00 (5.8634e+00)    Acc@1   2.73 (  1.23)   Acc@5   8.20 (  4.86)
Epoch: [0][ 60/391]     Time  3.322 ( 1.052)    Data  3.236 ( 0.863)    Loss 5.1129e+00 (5.7466e+00)    Acc@1   3.91 (  1.37)   Acc@5  12.50 (  5.50)
Epoch: [0][ 70/391]     Time  0.111 ( 1.079)    Data  0.006 ( 0.895)    Loss 5.0209e+00 (5.6550e+00)    Acc@1   2.34 (  1.52)   Acc@5  12.11 (  6.17)
Epoch: [0][ 80/391]     Time  2.752 ( 1.077)    Data  2.662 ( 0.898)    Loss 5.0356e+00 (5.5783e+00)    Acc@1   1.17 (  1.64)   Acc@5   9.38 (  6.67)
Epoch: [0][ 90/391]     Time  0.404 ( 1.043)    Data  0.304 ( 0.871)    Loss 4.9291e+00 (5.5136e+00)    Acc@1   3.91 (  1.86)   Acc@5  16.80 (  7.29)
Epoch: [0][100/391]     Time  2.293 ( 1.041)    Data  2.205 ( 0.872)    Loss 4.9172e+00 (5.4593e+00)    Acc@1   2.73 (  2.01)   Acc@5  14.06 (  7.84)
Epoch: [0][110/391]     Time  1.070 ( 1.022)    Data  0.975 ( 0.855)    Loss 4.8944e+00 (5.4101e+00)    Acc@1   4.69 (  2.14)   Acc@5  12.11 (  8.37)
Epoch: [0][120/391]     Time  2.176 ( 1.015)    Data  2.081 ( 0.851)    Loss 4.7957e+00 (5.3681e+00)    Acc@1   4.30 (  2.28)   Acc@5  16.02 (  8.77)
Epoch: [0][130/391]     Time  1.357 ( 1.003)    Data  1.266 ( 0.841)    Loss 4.8848e+00 (5.3330e+00)    Acc@1   5.08 (  2.43)   Acc@5  13.67 (  9.12)
Epoch: [0][140/391]     Time  1.765 ( 0.995)    Data  1.671 ( 0.835)    Loss 4.7865e+00 (5.2994e+00)    Acc@1   5.08 (  2.60)   Acc@5  16.41 (  9.60)
Epoch: [0][150/391]     Time  1.744 ( 0.993)    Data  1.634 ( 0.835)    Loss 4.8230e+00 (5.2677e+00)    Acc@1   3.91 (  2.75)   Acc@5  15.62 (  9.98)
Epoch: [0][160/391]     Time  1.385 ( 0.985)    Data  1.286 ( 0.828)    Loss 4.7742e+00 (5.2403e+00)    Acc@1   3.91 (  2.83)   Acc@5  17.58 ( 10.32)
Epoch: [0][170/391]     Time  2.221 ( 0.982)    Data  2.124 ( 0.826)    Loss 4.7745e+00 (5.2166e+00)    Acc@1   3.91 (  2.96)   Acc@5  14.84 ( 10.61)
Epoch: [0][180/391]     Time  1.313 ( 0.974)    Data  1.216 ( 0.819)    Loss 4.7646e+00 (5.1928e+00)    Acc@1   5.86 (  3.08)   Acc@5  17.97 ( 10.91)
Epoch: [0][190/391]     Time  1.508 ( 0.969)    Data  1.415 ( 0.815)    Loss 4.7903e+00 (5.1709e+00)    Acc@1   6.25 (  3.21)   Acc@5  18.75 ( 11.18)
Epoch: [0][200/391]     Time  1.497 ( 0.964)    Data  1.403 ( 0.811)    Loss 4.8604e+00 (5.1509e+00)    Acc@1   3.91 (  3.30)   Acc@5  11.72 ( 11.44)
Epoch: [0][210/391]     Time  2.618 ( 0.975)    Data  2.532 ( 0.822)    Loss 4.6898e+00 (5.1294e+00)    Acc@1   7.42 (  3.42)   Acc@5  20.70 ( 11.76)
Epoch: [0][220/391]     Time  1.456 ( 0.975)    Data  1.362 ( 0.823)    Loss 4.8093e+00 (5.1113e+00)    Acc@1   5.08 (  3.51)   Acc@5  15.62 ( 12.02)
Epoch: [0][230/391]     Time  0.198 ( 0.975)    Data  0.000 ( 0.824)    Loss 4.6395e+00 (5.0932e+00)    Acc@1   7.03 (  3.61)   Acc@5  19.92 ( 12.32)
Epoch: [0][240/391]     Time  1.664 ( 0.971)    Data  1.571 ( 0.820)    Loss 4.7447e+00 (5.0780e+00)    Acc@1   4.30 (  3.70)   Acc@5  15.23 ( 12.54)
Epoch: [0][250/391]     Time  0.200 ( 0.968)    Data  0.000 ( 0.818)    Loss 4.6881e+00 (5.0615e+00)    Acc@1   6.64 (  3.82)   Acc@5  21.09 ( 12.82)
Epoch: [0][260/391]     Time  1.807 ( 0.966)    Data  1.709 ( 0.815)    Loss 4.5320e+00 (5.0471e+00)    Acc@1   6.64 (  3.86)   Acc@5  23.83 ( 13.04)
Epoch: [0][270/391]     Time  0.203 ( 0.962)    Data  0.000 ( 0.812)    Loss 4.6413e+00 (5.0340e+00)    Acc@1   6.25 (  3.95)   Acc@5  16.41 ( 13.25)
Epoch: [0][280/391]     Time  1.278 ( 0.958)    Data  1.190 ( 0.808)    Loss 4.6666e+00 (5.0192e+00)    Acc@1   6.25 (  4.05)   Acc@5  19.53 ( 13.51)
Epoch: [0][290/391]     Time  0.204 ( 0.958)    Data  0.001 ( 0.808)    Loss 4.6502e+00 (5.0059e+00)    Acc@1   7.42 (  4.13)   Acc@5  17.19 ( 13.71)
Epoch: [0][300/391]     Time  0.802 ( 0.952)    Data  0.704 ( 0.802)    Loss 4.5676e+00 (4.9932e+00)    Acc@1   5.47 (  4.19)   Acc@5  21.88 ( 13.90)
Epoch: [0][310/391]     Time  0.193 ( 0.969)    Data  0.000 ( 0.820)    Loss 4.6667e+00 (4.9810e+00)    Acc@1   7.42 (  4.29)   Acc@5  20.70 ( 14.13)
Epoch: [0][320/391]     Time  1.186 ( 0.964)    Data  1.090 ( 0.815)    Loss 4.5690e+00 (4.9688e+00)    Acc@1   6.64 (  4.37)   Acc@5  20.70 ( 14.34)
Epoch: [0][330/391]     Time  0.202 ( 0.962)    Data  0.000 ( 0.813)    Loss 4.6888e+00 (4.9574e+00)    Acc@1   6.25 (  4.46)   Acc@5  16.80 ( 14.53)
Epoch: [0][340/391]     Time  1.774 ( 0.961)    Data  1.680 ( 0.812)    Loss 4.5750e+00 (4.9455e+00)    Acc@1   6.25 (  4.54)   Acc@5  20.31 ( 14.72)
Epoch: [0][350/391]     Time  0.191 ( 0.957)    Data  0.000 ( 0.808)    Loss 4.5918e+00 (4.9341e+00)    Acc@1   5.08 (  4.65)   Acc@5  21.88 ( 14.96)
Epoch: [0][360/391]     Time  1.968 ( 0.955)    Data  1.872 ( 0.806)    Loss 4.5998e+00 (4.9240e+00)    Acc@1  10.16 (  4.74)   Acc@5  21.88 ( 15.17)
Epoch: [0][370/391]     Time  0.204 ( 0.952)    Data  0.000 ( 0.803)    Loss 4.6140e+00 (4.9143e+00)    Acc@1   7.03 (  4.81)   Acc@5  19.14 ( 15.34)
Epoch: [0][380/391]     Time  2.599 ( 0.953)    Data  2.499 ( 0.804)    Loss 4.4078e+00 (4.9051e+00)    Acc@1   9.77 (  4.89)   Acc@5  24.22 ( 15.52)
Epoch: [0][390/391]     Time  0.805 ( 0.949)    Data  0.000 ( 0.799)    Loss 4.4114e+00 (4.8956e+00)    Acc@1  11.25 (  4.94)   Acc@5  25.62 ( 15.67)
Test: [ 0/40]   Time  4.350 ( 4.350)    Loss 7.1015e+00 (7.1015e+00)    Acc@1   0.00 (  0.00)   Acc@5   2.34 (  2.34)
Test: [10/40]   Time  0.274 ( 1.121)    Loss 7.1271e+00 (7.0948e+00)    Acc@1   0.39 (  0.36)   Acc@5   0.78 (  2.27)
Test: [20/40]   Time  2.579 ( 1.045)    Loss 7.0556e+00 (7.0879e+00)    Acc@1   1.95 (  0.50)   Acc@5   3.91 (  1.99)
Test: [30/40]   Time  0.746 ( 0.984)    Loss 7.3109e+00 (7.0825e+00)    Acc@1   0.39 (  0.57)   Acc@5   0.78 (  1.99)
 * Acc@1 0.630 Acc@5 2.080
```

V100 Training on Prince (Epoch 0)

```
Amorian@imagenet-v100:~$ cat imagenet-output.txt
=> creating model 'resnet18'
Epoch: [0][  0/391]     Time  3.673 ( 3.673)    Data  1.635 ( 1.635)    Loss 7.0623e+00 (7.0623e+00)    Acc@1   0.39 (  0.39) Acc@5   0.39 (  0.39)
Epoch: [0][ 10/391]     Time  0.211 ( 0.510)    Data  0.000 ( 0.149)    Loss 6.5905e+00 (6.8337e+00)    Acc@1   0.39 (  0.36) Acc@5   1.17 (  1.17)
Epoch: [0][ 20/391]     Time  0.196 ( 0.367)    Data  0.000 ( 0.084)    Loss 5.8792e+00 (6.5095e+00)    Acc@1   0.00 (  0.48) Acc@5   2.34 (  2.10)
Epoch: [0][ 30/391]     Time  0.211 ( 0.335)    Data  0.075 ( 0.095)    Loss 5.5291e+00 (6.2344e+00)    Acc@1   1.56 (  0.63) Acc@5   5.47 (  2.87)
Epoch: [0][ 40/391]     Time  0.192 ( 0.313)    Data  0.000 ( 0.091)    Loss 5.3603e+00 (6.0378e+00)    Acc@1   1.56 (  0.71) Acc@5   5.86 (  3.42)
Epoch: [0][ 50/391]     Time  0.208 ( 0.319)    Data  0.006 ( 0.109)    Loss 5.2879e+00 (5.8889e+00)    Acc@1   1.17 (  1.04) Acc@5   5.47 (  4.27)
Epoch: [0][ 60/391]     Time  0.240 ( 0.309)    Data  0.125 ( 0.108)    Loss 5.1148e+00 (5.7735e+00)    Acc@1   1.56 (  1.28) Acc@5   9.38 (  4.99)
Epoch: [0][ 70/391]     Time  0.227 ( 0.304)    Data  0.000 ( 0.109)    Loss 5.1169e+00 (5.6838e+00)    Acc@1   1.95 (  1.38) Acc@5   9.77 (  5.56)
Epoch: [0][ 80/391]     Time  0.201 ( 0.298)    Data  0.000 ( 0.108)    Loss 5.0636e+00 (5.6122e+00)    Acc@1   4.30 (  1.57) Acc@5  10.16 (  6.00)
Epoch: [0][ 90/391]     Time  0.414 ( 0.293)    Data  0.309 ( 0.108)    Loss 5.0019e+00 (5.5489e+00)    Acc@1   1.95 (  1.75) Acc@5  11.33 (  6.66)
Epoch: [0][100/391]     Time  0.257 ( 0.287)    Data  0.076 ( 0.104)    Loss 4.9507e+00 (5.4922e+00)    Acc@1   3.52 (  1.94) Acc@5  10.94 (  7.28)
Epoch: [0][110/391]     Time  0.577 ( 0.288)    Data  0.471 ( 0.108)    Loss 4.9718e+00 (5.4484e+00)    Acc@1   3.12 (  2.00) Acc@5  13.28 (  7.65)
Epoch: [0][120/391]     Time  0.199 ( 0.283)    Data  0.000 ( 0.105)    Loss 5.0110e+00 (5.4053e+00)    Acc@1   4.69 (  2.21) Acc@5  13.67 (  8.16)
Epoch: [0][130/391]     Time  0.301 ( 0.282)    Data  0.202 ( 0.105)    Loss 4.8663e+00 (5.3693e+00)    Acc@1   5.86 (  2.32) Acc@5  16.41 (  8.56)
Epoch: [0][140/391]     Time  0.215 ( 0.280)    Data  0.000 ( 0.105)    Loss 4.8466e+00 (5.3356e+00)    Acc@1   3.91 (  2.47) Acc@5  12.89 (  8.92)
Epoch: [0][150/391]     Time  0.225 ( 0.278)    Data  0.048 ( 0.104)    Loss 4.7522e+00 (5.3025e+00)    Acc@1   3.91 (  2.58) Acc@5  16.02 (  9.36)
Epoch: [0][160/391]     Time  0.199 ( 0.276)    Data  0.000 ( 0.102)    Loss 4.8055e+00 (5.2759e+00)    Acc@1   4.30 (  2.67) Acc@5  17.97 (  9.69)
Epoch: [0][170/391]     Time  0.374 ( 0.276)    Data  0.257 ( 0.103)    Loss 4.8692e+00 (5.2491e+00)    Acc@1   4.69 (  2.78) Acc@5  13.28 ( 10.05)
Epoch: [0][180/391]     Time  0.202 ( 0.274)    Data  0.000 ( 0.101)    Loss 4.7960e+00 (5.2238e+00)    Acc@1   4.30 (  2.89) Acc@5  14.45 ( 10.42)
Epoch: [0][190/391]     Time  0.214 ( 0.273)    Data  0.000 ( 0.100)    Loss 4.8449e+00 (5.2031e+00)    Acc@1   1.56 (  2.96) Acc@5  12.50 ( 10.70)
Epoch: [0][200/391]     Time  0.203 ( 0.277)    Data  0.000 ( 0.105)    Loss 4.7634e+00 (5.1813e+00)    Acc@1   5.08 (  3.04) Acc@5  16.80 ( 10.98)
Epoch: [0][210/391]     Time  0.204 ( 0.277)    Data  0.000 ( 0.104)    Loss 4.6605e+00 (5.1617e+00)    Acc@1   7.81 (  3.13) Acc@5  15.62 ( 11.21)
Epoch: [0][220/391]     Time  0.378 ( 0.277)    Data  0.262 ( 0.105)    Loss 4.8119e+00 (5.1432e+00)    Acc@1   6.64 (  3.23) Acc@5  17.97 ( 11.50)
Epoch: [0][230/391]     Time  0.217 ( 0.276)    Data  0.000 ( 0.104)    Loss 4.7111e+00 (5.1234e+00)    Acc@1   5.08 (  3.34) Acc@5  14.45 ( 11.76)
Epoch: [0][240/391]     Time  0.362 ( 0.276)    Data  0.263 ( 0.105)    Loss 4.6482e+00 (5.1073e+00)    Acc@1   5.86 (  3.42) Acc@5  21.09 ( 11.97)
Epoch: [0][250/391]     Time  0.209 ( 0.274)    Data  0.000 ( 0.105)    Loss 4.7770e+00 (5.0916e+00)    Acc@1   5.08 (  3.51) Acc@5  15.23 ( 12.21)
Epoch: [0][260/391]     Time  0.219 ( 0.274)    Data  0.046 ( 0.105)    Loss 4.7366e+00 (5.0770e+00)    Acc@1   3.52 (  3.59) Acc@5  15.23 ( 12.41)
Epoch: [0][270/391]     Time  0.228 ( 0.274)    Data  0.000 ( 0.106)    Loss 4.7629e+00 (5.0619e+00)    Acc@1   6.25 (  3.67) Acc@5  18.36 ( 12.65)
Epoch: [0][280/391]     Time  0.427 ( 0.274)    Data  0.319 ( 0.106)    Loss 4.6560e+00 (5.0482e+00)    Acc@1   5.08 (  3.74) Acc@5  16.80 ( 12.87)
Epoch: [0][290/391]     Time  0.205 ( 0.272)    Data  0.000 ( 0.104)    Loss 4.6955e+00 (5.0350e+00)    Acc@1   3.91 (  3.83) Acc@5  17.19 ( 13.07)
Epoch: [0][300/391]     Time  0.368 ( 0.272)    Data  0.261 ( 0.104)    Loss 4.5039e+00 (5.0223e+00)    Acc@1   8.59 (  3.93) Acc@5  20.31 ( 13.27)
Epoch: [0][310/391]     Time  0.206 ( 0.271)    Data  0.000 ( 0.102)    Loss 4.5604e+00 (5.0087e+00)    Acc@1   8.98 (  4.01) Acc@5  19.92 ( 13.50)
Epoch: [0][320/391]     Time  0.363 ( 0.271)    Data  0.262 ( 0.102)    Loss 4.3512e+00 (4.9942e+00)    Acc@1  12.11 (  4.11) Acc@5  27.34 ( 13.71)
Epoch: [0][330/391]     Time  0.254 ( 0.271)    Data  0.000 ( 0.101)    Loss 4.5540e+00 (4.9825e+00)    Acc@1   5.47 (  4.18) Acc@5  19.92 ( 13.91)
Epoch: [0][340/391]     Time  0.225 ( 0.271)    Data  0.113 ( 0.103)    Loss 4.7350e+00 (4.9714e+00)    Acc@1   5.08 (  4.25) Acc@5  14.45 ( 14.08)
Epoch: [0][350/391]     Time  0.225 ( 0.271)    Data  0.000 ( 0.102)    Loss 4.4558e+00 (4.9599e+00)    Acc@1   7.81 (  4.34) Acc@5  21.88 ( 14.30)
Epoch: [0][360/391]     Time  0.180 ( 0.271)    Data  0.039 ( 0.102)    Loss 4.6207e+00 (4.9498e+00)    Acc@1   6.25 (  4.42) Acc@5  19.53 ( 14.49)
Epoch: [0][370/391]     Time  0.203 ( 0.271)    Data  0.000 ( 0.102)    Loss 4.4132e+00 (4.9379e+00)    Acc@1   8.98 (  4.52) Acc@5  21.88 ( 14.73)
Epoch: [0][380/391]     Time  0.283 ( 0.271)    Data  0.192 ( 0.103)    Loss 4.6132e+00 (4.9278e+00)    Acc@1   6.64 (  4.60) Acc@5  20.31 ( 14.93)
Epoch: [0][390/391]     Time  0.876 ( 0.271)    Data  0.000 ( 0.102)    Loss 4.5251e+00 (4.9186e+00)    Acc@1   7.50 (  4.66) Acc@5  20.00 ( 15.10)
Test: [ 0/40]   Time  1.366 ( 1.366)    Loss 6.1331e+00 (6.1331e+00)    Acc@1   3.52 (  3.52)   Acc@5   7.81 (  7.81)
Test: [10/40]   Time  0.081 ( 0.328)    Loss 6.1381e+00 (6.1354e+00)    Acc@1   1.56 (  3.59)   Acc@5   7.81 (  9.27)
Test: [20/40]   Time  0.081 ( 0.287)    Loss 6.1771e+00 (6.1244e+00)    Acc@1   3.91 (  3.96)   Acc@5  10.55 (  9.95)
Test: [30/40]   Time  0.117 ( 0.286)    Loss 6.3797e+00 (6.1404e+00)    Acc@1   1.56 (  3.78)   Acc@5   7.81 (  9.48)
 * Acc@1 3.800 Acc@5 9.540
```

V100 Training on Google Cloud (Epoch 0)

Explaining the two important metrics to note here –

1. Time – The total time taken to run, including data loading and processing.
2. Data – Data loading time

The value in parenthesis is the average time taken for the current epoch.

Here, we can see something very off – The data loading time is way higher on Prince (final average of 0.799s) when compared to Google Cloud (final average of 0.102). The total time taken is 0.159s on Prince and 0.169s on Google Cloud on average, when you remove the data loading time average. This suggests that the performance bottleneck isn't in the GPUs itself, but the data loading time. This might be due to the V100 GPUs being physically farther away from the data source (GPU nodes in the range of 70 – 90) in comparison to K80 (GPU nodes in the

range of 0 – 30). This would explain why the time taken on Prince is faster on K80, but slower on the rest for single GPU Imagenet training.

As for Multi-GPU, this data transfer issue becomes even more evident, as V100 and P100 start performing worse than K80, and P40 comes close to K80, as the data loading time to 4 GPUs is making parallel processing much slower.

Extra Features on Google Cloud

Google Cloud also comes with other features that aren't available on Prince, which are listed below:

1. Cloud TPU's - Deep Learning Accelerators made by Google.
2. Superuser/sudo rights - Root Access to download and use any libraries or applications necessary, without being restricted to only the available modules like on Prince.
3. Docker/Kubernetes - Google Kubernetes Engine and Docker Containers are usable since superuser access exists.
4. AI Platform - Auto ML/Predictions from Google based on your data
5. Easy JupyterLab access - Create notebooks with a VM attached
6. Static IP's for instances if necessary
7. Premade images to use in VM's for deep learning jobs

Conclusion

	Prince	Google Cloud
Ease of Access		✓
Storage		✓
Performance	✓	✓
Access Time	✓	✓
Hardware	✓	✓
Flexibility		✓

Google Cloud is the choice in our analysis - much better User Experience, Buckets and Disk based storage, comparable performance and hardware, and the access times are constant and do not vary and is much more flexible than Prince while offering more features. Can scale infinitely without needing to deal with the hardware directly (through the UI), making HPC admin work much easier for the future.

While Prince is great for the long run, Google Cloud offers a lot more utility and similar performance. If the cost is not too great, migrating certain groups of users to Google Cloud or giving priority access to researchers and professors on Google Cloud is a possibility, but Prince can continue to stay for students or other groups of people.