How Deep Learning Model Architecture and Software Stack Impacts Training Performance in the Cloud

Egor Bykov(✉), Evgeny Protasenko, and Vladimir Kobzev

RocketCompute, 541 Jefferson Avenue Ste 100, Redwood City, CA 94063, USA
dev@rocketcompute.com

Abstract. Choosing the right instance on a public cloud for model training is not an easy task. There are hundreds of different virtual machines available with a wide variety of CPU core counts (i.e., how many tasks can be performed in parallel), memory, disk type, network speed, and of course graphics card (GPU). The latter has often become a differentiating factor for choosing one virtual machine (VM) over another. On top of that, containerization technology has greatly simplified GPU computations for machine learning, wrapping the software stack above the Kernel level in containers and allowing to juggle with different combinations of frameworks, lower lever libraries, and hardware drivers. Technologies like Nvidia-docker has even unlocked new stack combinations (driver plus low lever Compute Unified Device Architecture libraries, most of the time referred as CUDA) that were not feasible before. This, however, adds another dimension to the performance optimization problem, and now you not only need to choose an optimal hardware for your machine learning task, but also a variate driver-CUDA combination to fine-tune the performance further.

The goal of this work is to scrutinize how neural network architecture, different versions of NVIDIA drivers, and CUDA libraries will influence training performance and cost on different cloud VMs. We compared BERT, Mask R-CNN, and DLRM architectures using instances available on Amazon Cloud Services via Elastic Compute service (AWS EC2 for short) and showed that architecture, model implementation, and software stack can cause significant variation in training time and cost with a different optimal configuration for different architectures and software stacks.

Keywords: BERT · DLRM · Mask R-CNN · Deep learning training performance

1 Introduction

This chapter is the first step in our research program aimed at scrutinizing how different components of server hardware and software stacks affect a deep learning model's training time and cost. Our hypothesis is that even in a very simple setup where we need to choose a cloud VM with a single GPU secondary parameter (like RAM, CPU performance, disks, etc.) might have a significant impact on the cost and duration of the training, and this impact will vary from one network architecture to another.

W. F. Lawless et al. (Eds.): Engineering Artificially Intelligent Systems, LNCS 13000, pp. 109–121, 2021.
https://doi.org/10.1007/978-3-030-89385-9_7

Another hypothesis that we are testing here is that different combinations of GPU drivers and low-level libraries providing GPU acceleration for ML frameworks like PyTorch will cause significant deviations in training cost and time. There is anecdotal evidence that drivers might cause a performance degradation of the GPU from the consumer market, however, we have not seen papers investigating whether there is any performance implication for ML workloads on professional GPUs.

2 Set-Up

2.1 Benchmarks

As a base for testing machine-learning workloads, we chose the latest set of reference implementations for MLPerf training benchmarks [1]. Out of eight different neural network implementations available, we picked up one model for each domain that is widely relevant for the industry. The resulting reference model set is:

1. For image processing – an object detection Mask R-CNN model trained on COCO data set [2].
2. For natural language processing – a BERT model trained on a Wikipedia dump [3].
3. For recommendations – a DLRM model trained on a 1Tb Kaggle AdDisplay Challenge dataset [4].

2.2 Changes to the Reference Implementations

Our motivation to make changes to the reference implementations was driven by the following:

- The Take into account outdated hardware still available in the cloud. Users still can find cloud instances with GPUs as old as the Kepler-family Nvidia accelerators. The reference implementation of MLPerf benchmarks, tuned towards measuring the performance of current and future generations of a GPU, is not only time-consuming and costly on older generations, but sometimes even that cannot be performed due to insufficient GPU memory or other reasons.
- The Speed up the testing process. This chapter is the first step in the lengthy program developed to research how different elements in the technology stack influence performance and cost of deep learning model training. Having this in mind, building a representative set of fast and cost-efficient benchmarks is particularly important.
- The Ease of reproducibility of these results. Speed and cost-efficiency of the benchmarks provide more reliability of the result, as any test can be reproduced by any member of the community, and all results can be verified.

The resulting changes to the reference benchmarks are:
Object Detection (R-CNN)

- For the object detection model, the number of interactions was capped at 3,000 which resulted in a benchmark duration of around 15 min for the Volta family accelerators (Nvidia Telsa V100).

Recommender (DLRM)

- The The dataset was cut from 1Tb down to 20 Gb to reduce the burden of fetching training data for each test.
- The The number of epochs was limited by two, reducing the test duration to circa 45 min for virtual machine with one Nvidia Tesla V100 GPU.

Natural Language Processing (BERT)

- The architecture of the neural network was changed from BERT Large to BERT Base [3] to revise the benchmark for accelerators with 7 Gb of available VRAM.
- Just-in-time CUDA code compilations were turned off to reduce the benchmark start-up time.
- The number of training steps was reduced to 15k, as a result one Tesla V100 performed the test in circa 25 min.

2.3 Benchmarking Software Stack

To test the performance of different GPU driver/CUDA combinations, we administered the DLRM benchmark. Its implementation was based on PyTorch 1.7.1 which allowed us to use several CUDA versions (in this chapter, we present only 9.2, 10.1, 10.2, and 11.0 versions) without changing the benchmark source code. For each test we used Ubuntu 16.04 (with kernel version 4.4) as an operating system, on top of it we installed seven different GPU drivers obtained from the official Nvidia site. These drivers are:

- 410.129
- 418.165
- 440.118
- 450.80
- 455.32
- 460.32

This particular choice of drivers was guided by two things:

1. The compatibility of a driver with a Linux kernel (we decided not to drop below version 4.4); and,
2. The compatibility of a particular driver with different CUDA libraries

The benchmark was run inside of a container using docker v19.3 and nvidia-docker2. It is worth mentioning that using the nvidia-docker was critical for the whole experiment because when running a framework without a container, one should match the CUDA version installed on a host with the version used during the compilation of the framework's libraries. Another important note is that a particular version of the CUDA toolkit requires a certain version of the device's driver. This driver is usually installed simultaneously with the toolkit. Therefore, using a GPU enabled container engine is crucial, without it, this experiment is not feasible.

2.4 Infrastructure

To test our hypothesis, we took the most popular cloud provider Amazon Web Services (AWS) and ran our set of benchmarks on all of the available single GPU instances. It is also important to note that all VMs were booked in one availability zone (US-East-2 according to AWS zone naming), and that we used only general-purpose volumes (gp2) for benchmarking the driver/CUDA combinations (Table 1).

Table 1. The AWS instances with a single GPU used for benchmarking.

GPU	CPU family/model	AWS instance	vCPU (#)	RAM (Gb)	VRAM[1] (Gb)	Disk (type)	Price ($/hour)
K80	E5-2686 v4	p2.xlarge	4	61	11	gp2/io1	0.9
M60	E5-2686 v4	g3s.xlarge	4	30.5	7	gp2/io1	0.75
M60	E5-2686 v4	g3.4xlarge	16	122	7	gp2/io1	1.14
V100	E5-2686 v4	p3.2xlarge	8	61	16	g2p/io1	3.06
T4	Cascade Lake	g4dn.xlarge	4	16	15	gp2/io1	0.53
T4	Cascade Lake	g4dn.2xlarge	8	32	15	gp2/io1	0.75
T4	Cascade Lake	g4dn.4xlarge	16	64	15	gp2/io1	1.2
T4	Cascade Lake	g4dn.8xlarge	32	128	15	gp2/io1	2.18

3 Benchmark Results

3.1 GPU Instances

Figure 1, Fig. 2, and Fig. 3 show the summary of the benchmark runtimes and costs in relative terms. All pf the values are normalized to the best result (shown as equal to 1.00) across all of the different instances. The benchmarks with the lowest duration and costs are highlighted with dashed boxes, and all other results are multiples of these best results. Blue bars represent the cost of running a benchmark, and red bars show the duration of the benchmark.

[1] GPU memory available inside virtual machine.

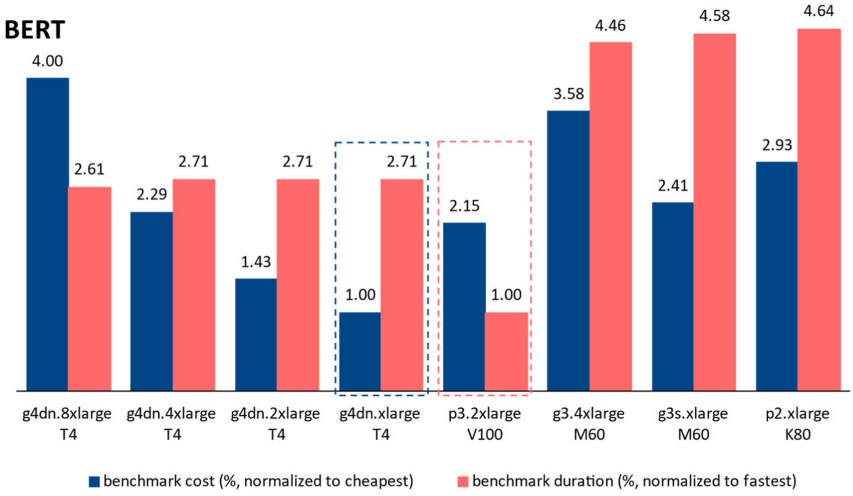

Fig. 1. Relative performance results for the BERT

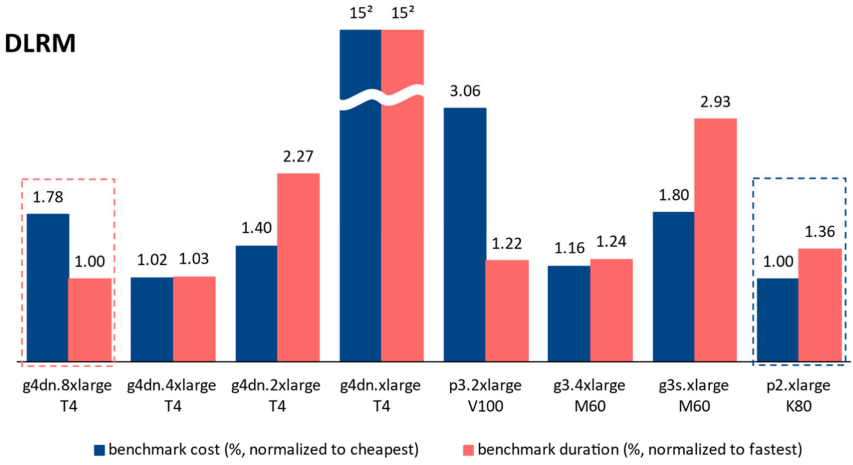

Fig. 2. Relative performance results for the DLRM[2]

[2] reflects the best performance duration or cost estimated based on data from aborded test. The actual test was stopped after 300 min and then duration/cost was extrapolated to get estimate for the full successful test (e.g., all successful runs for other configurations took between 15 and 120 min).

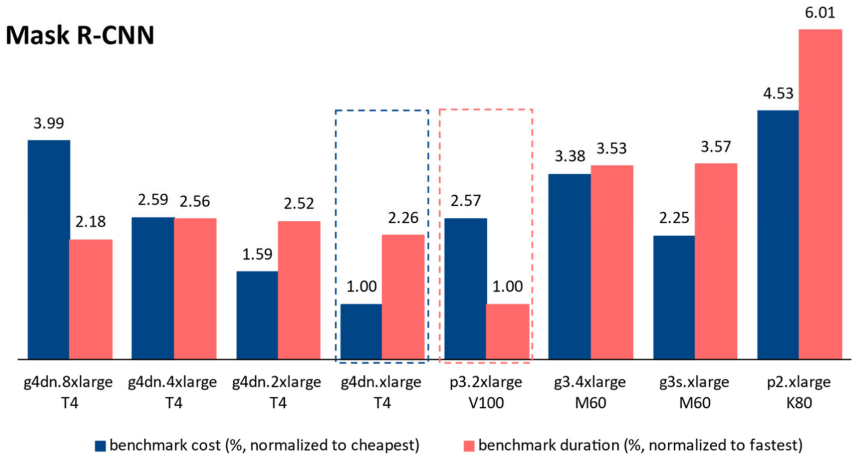

Fig. 3. Relative performance results for the Mask R-CNN

The above graphs in the figure clearly show that there is a notable performance difference between instances across different neural networks. To illustrate this better, let us use a different representation of the same data. Table 2 below shows a summary across all VMs and neural networks. There are four interesting observations from this table (highlighted). From top to bottom:

- Goofy VM configurations can give both the best price and performance. An example, a g4dn.4xlarge instance (Tesla T4) which gives mediocre results for both BERT and R-CNN appears to be the best option for DLRM in terms of price/performance.
- The cheapest instances for one network can be the most expensive for others. An example, a g4dn.xlarge (Tesla T4) which gave the lowest training costs for BERT and R-CNN is the most expensive for DLRM training by at least an order of magnitude.
- The most powerful GPUs are not always the fastest, and never cost-efficient. The Tesla V100 (p3.2xlarge instance) is the most capable GPU in our set and provided the lowest training time for BERT and R-CNN, but it was 22% slower than the significantly weaker Tesla T4 when used for the DLRM network.
- Legacy GPUs can still be the cheapest to train certain networks. Tesla K80 was the cheapest for training the DLRM network.

Table 2. Benchmark results summary (relative values, lower is better)

GPU	AWS instance	Price ($/hour)	BERT		DLRM		Mask R-CNN	
			Cost	Time	Cost	Time	Cost	Time
T4	g4dn.8xlarge	2.18	4.00	2.61	1.78	1.00	3.99	2.18
T4	g4dn.4xlarge	1.2	2.29	2.71	1.02	1.03	2.59	2.56

(*continued*)

Table 2. (*continued*)

GPU	AWS instance	Price ($/hour)	BERT		DLRM		Mask R-CNN	
			Cost	Time	Cost	Time	Cost	Time
T4	g4dn.2xlarge	0.75	1.43	2.71	1.40	2.27	1.59	2.52
T4	g4dn.xlarge	0.53	1.00	2.71	15.00	15.00	1.00	2.26
V100	p3.2xlarge	3.06	2.15	1.00	3.06	1.22	2.57	1.00
M60	g3.4xlarge	1.14	3.58	4.46	1.16	1.24	3.38	3.53
M60	g3s.xlarge	0.75	2.41	4.58	1.80	2.93	2.25	3.57
K80	p2.xlarge	0.9	2.93	4.64	1.00	1.36	4.53	6.01

To illustrate the last two bullet points further, let us consider Fig. 4 and Fig. 5. The highlighted instances in these figures are for GPUs with substantially different performance, but still showing very similar results in terms of training times. Moreover, Fig. 5 highlights the case where the GPU from the latest generation available shows a training time close to GPUs which are several generations earlier.

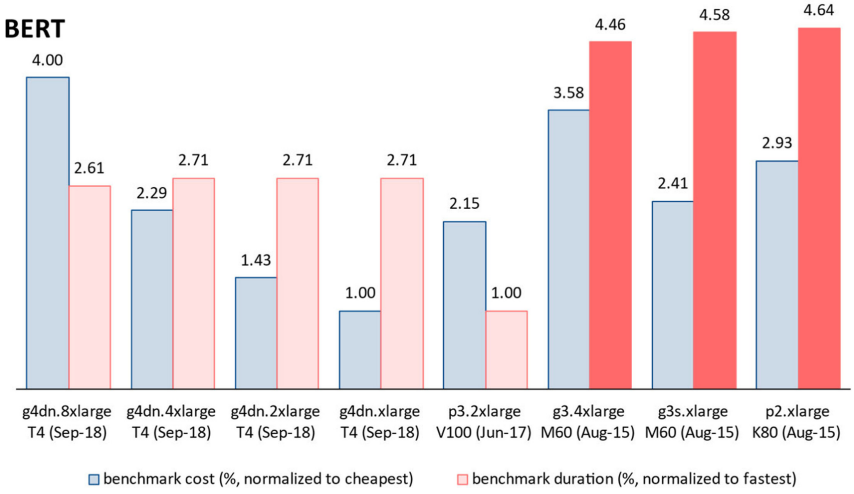

Fig. 4. Cloud instances with Tesla K80 give very similar training times for BERT as the instances for the Tesla M60 (next-generation compared to K80).

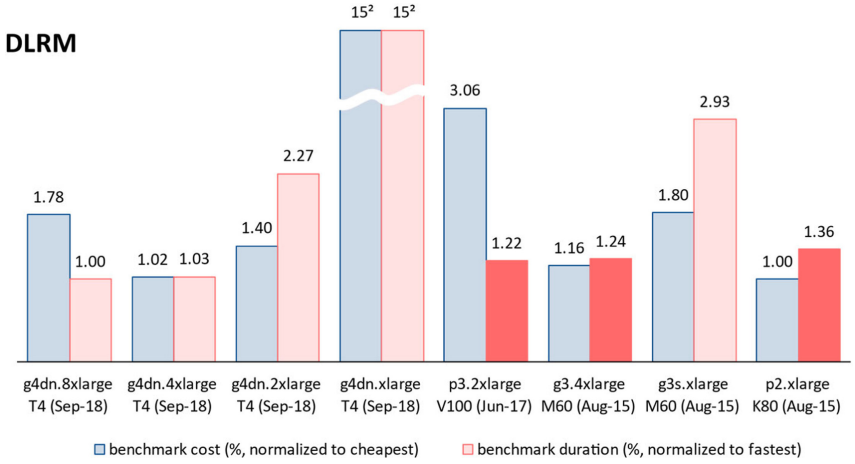

Fig. 5. Instances with Tesla V100 (the most powerful GPU in the set) show training times for DLRM that are marginally better than the weakest Tesla K80 and the second weakest Tesla M60.

To summarize the evidence presented above, it is clear that choosing the most performant GPU guarantees neither the fastest training time, nor the lowest training costs.

In the following section, we scrutinize the evidence to explore why the DLRM benchmark is so different comparing to BERT and Mask R-CNN.

Model Implementation Implications

According to [4], a DLRM architecture implies feature embedding where categorical data (e.g., gender, geography, etc.) is transformed to a vector representation before being submitted to a neural network. Feature embedding requires a significant amount of RAM (44 Gb), and it forces instances with less memory to use the swap-space on a hard drive to perform the task. As a result, the most efficient instances for BERT and R-CNN have become the most expensive solution for a DLRM because of the lack of memory. We can see this effect on the utilization graphs in Fig. 6, where the DLRM model consumes all of the available RAM up to 44 Gb required to perform the training.

Another remarkable consequence of heavy RAM usage is low GPU utilization as part of the computation is shifted to a CPU. Moreover, the benchmark implementation used only one CPU thread. Both features led to a clear bottleneck on the CPU side which can be seen in the monitoring data (Fig. 7), where the more powerful GPUs had less utilization than did the less powerful.

As a result, the best performance was shown by instances with the best single-threaded CPU performance (i.e., the g4dn instances in our set).

For both R-CNN and BERT, although their architectures differ significantly, there is no evidence that their architecture leads to a significant loss of GPU performance.

Fig. 6. Memory utilization charts for the DLRM benchmark. The green area represents the memory reserved by the benchmark, the yellow area for the memory used by the OS as a cache for IO operations (irrelevant for our analysis). From top to bottom: 1) g3s.xlarge instance with 31 Gb of RAM; 2) g4dn.4xlarge with 64 Gb of RAM; and, 3) g4dn.8xlarge with 128 Gb RAM.

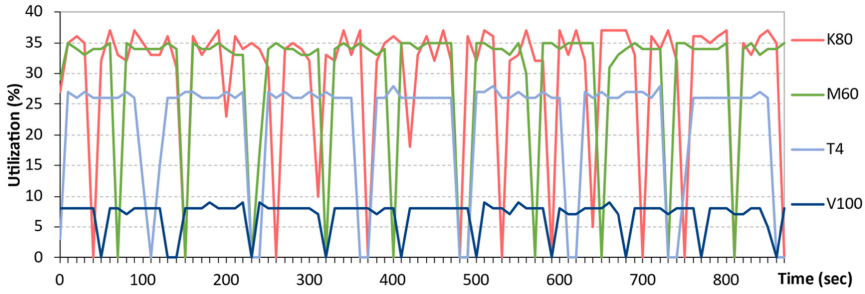

Fig. 7. GPU Utilization. Top to bottom: Tesla K80 (p2.xlarge) and Tesla M60 (g3.4xlarge) with 30% utilization on average, Tesla T4 (g4dn.8xlarge) with 22% on average, and V100 (p3.2xlarge) with 7% on average.

Comparing Training on a CPU Versus a GPU

To prove the point that the GPU is better for neural network training, we ran BERT and DLRM benchmarks on a subset of AWS CPU instances optimized for computing and storage (Table 3) and compared the training time and cost with results for GPU instances.

CPU instances appeared roughly seven times (7×) slower and five to six times (5–6×) more expensive (Table 4) when comparing to the best-performing GPU instances.

Table 3. AWS CPU instances chosen for benchmarking

CPU family/model	AWS instance	Optimized for	vCPU (#)	RAM (Gb)	Disk (type)	Price ($/hour)
E5 2666 v3	c4.4xlarge	Compute	16	30	gp2/io1	0.90
E5 2666 v3	c4.8xlarge	Compute	36	60	gp2/io1	0.75
Cascade Lake	c5.metal	Compute	96	192	gp2 io1	4.08
Cascade Lake	c5.12xlarge	Compute	48	96	gp2/io1	4.08
Cascade Lake	c5.18xlarge	Compute	72	144	gp2/io1	3.06
Cascade Lake	c5.24xlarge	Compute	96	192	gp2/io1	4.08
Cascade Lake	c5d.4xlarge	Storage	16	32	gp2/io1	0.77
Cascade Lake	c5d.12xlarge	Storage	48	96	gp2/io1	2.30
Xeon Platinum	c5d.9xlarge	Compute	36	96	gp2/io1	1.94
Xeon Platinum	m5d.metal	Storage	96	384	gp2/io1	5.42

Table 4. Comparing best performance for CPU instances versus GPU instances

Architecture	CPU			GPU		
	Runs (#)	Min cost ($)	Min time (mins)	Runs (#)	Min cost ($)	Min time (mins)
BERT	6	3.62	182	8	0.63	26
DLRM	9	3.59	237	12	0.74	36

The Effect of Storage

We also tested the influence of storage type for two instances. The results appeared to be controversial and to require additional, but thorough research. It appears that adding more performance storage can both increase and decrease results depending on the instance type (Table 5).

Table 5. Mask R-CNN benchmark storage type variations, Δ

Instance	Type	Time (mins)	Δ	Cost ($)	Δ
g4dn.xlarge	gp2	36	8.4%	0.32	8.4%
	io1	33		0.29	
p2.xlarge	gp2	88	−5.2%	1.32	−5.2%
	io1	93		1.39	

3.2 Performance Implications of GPU Drivers and CUDA Libraries

Figure 8, Fig. 9, Fig. 10, and Fig. 11 show the results of our benchmark performance for the software stacks. The vertical axis represents the duration of the DLRM benchmark in seconds, the horizontal axis represents the NVIDIA driver version, and each line on the graph represents a particular version of the CUDA libraries (version 11.0 is not supported by drivers older than 418, therefore, the yellow lines have fewer data points).

Each data point in these graphs aggregates at least 3 independent benchmarks and equals the mean benchmark duration. Each figure represents one of the four GPU models (Tesla K80, M60, T4, and V100).

Instances with Tesla K80 and Tesla M60 show similar variations caused by the driver version (around 10%). For both of these GPUs, CUDA 11.0 on average resulted in longer training time than the other versions. For K80 9.2 and 10.1, almost every time are better than for the other versions. CUDA 9.2 and driver v.440 gave the lowest training time on average, however, other versions of the driver (except 450) together with CUDA 9.2 or 10.1 showed very close performance.

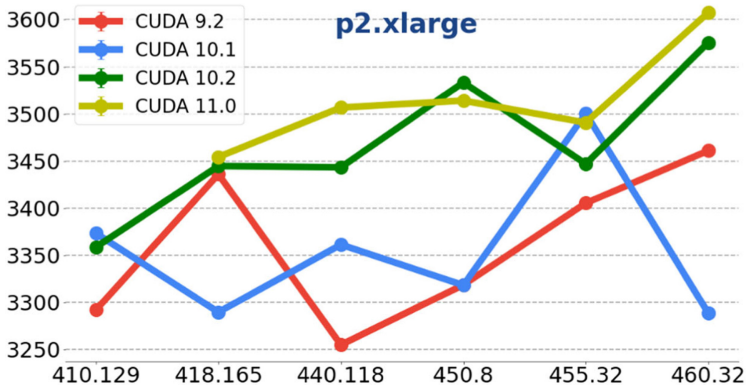

Fig. 8. Tesla K80. Benchmark duration (in seconds, vertical axis) versus the NVIDIA driver version (horizontal axis) for different CUDA library versions.

For the M60 performances, performance for all of the drivers lay within 3% from each other, but with a clear disadvantage in using CUDA 11.0.

Tesla T4 and Tesla V100 show bigger variations (14% and 15% on average, correspondingly). For T4, there is a clear optimum for CUDA 10.2 and driver v.410 and a clear worst performer (CUDA v.11.0 and driver v.450). It is safe to say that the recent two versions of CUDA gave longer training times compared to the older versions.

Tesla V100 instances show the highest variation of benchmark training times. It is remarkable that all CUDA versions except 11.0 reveal similar behaviors. On average, there is a clear optimum for the oldest version of the stack (CUDA v.9.2 and driver v.410), and a clear worst performing stack (CUDA v10.1 and driver v.450). The difference between these two combinations can exceed 20%. Another interesting observation is that V100 is slowed down by v.450 of the driver more than the other GPUs in the set.

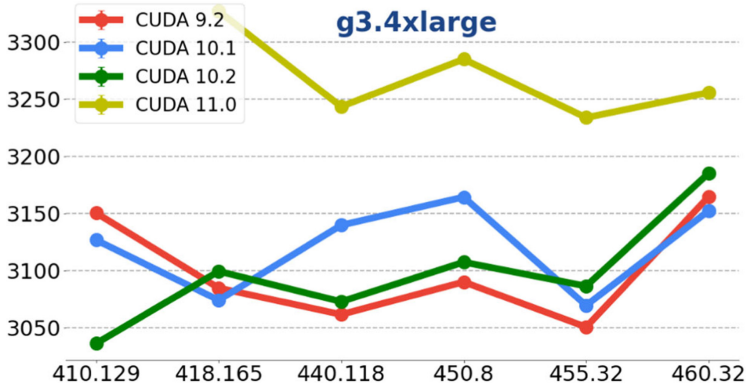

Fig. 9. Tesla M60: Benchmark duration (in seconds, vertical axis) versus the NVIDIA driver version (horizontal axis) for different CUDA library versions.

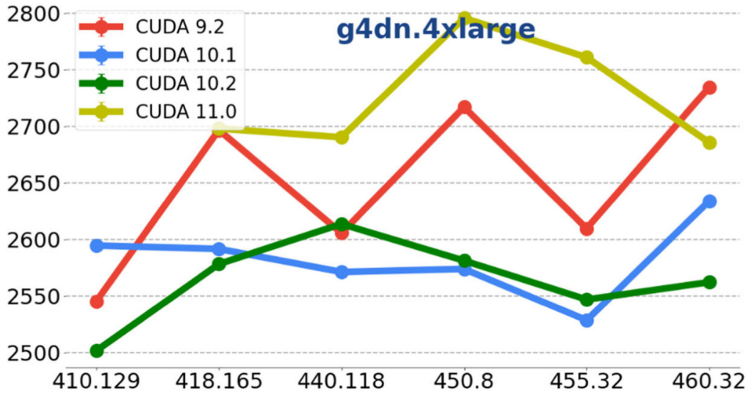

Fig. 10. Tesla T4: Benchmark duration (in seconds, vertical axis) versus the NVIDIA driver version (horizontal axis) for different CUDA library versions.

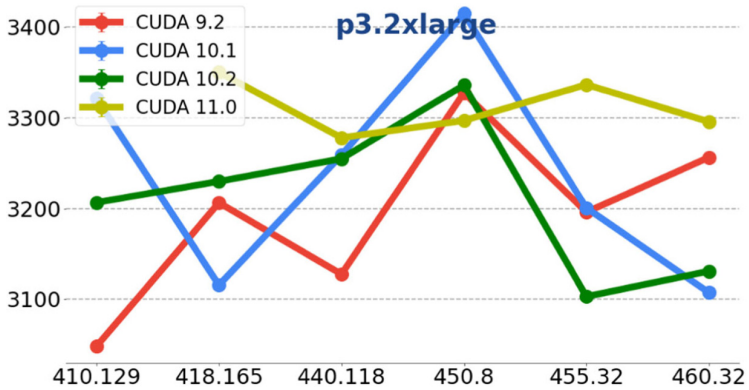

Fig. 11. Tesla V100: Benchmark duration (in seconds, vertical axis) versus the NVIDIA driver version (horizontal axis) for different CUDA library versions.

It is also remarkable that for almost all of the GPUs except for the Tesla K80 the driver v.410 is better on average than all of the newer drivers; and the new version of CUDA libraries is almost always worse than older versions. Another interesting observation is that v.450 is a worse performer than both of the previous and succeeding versions of the driver.

We believe that our data provide the evidence that optimizing a software stack can give meaningful benefit in terms of speed and cost of training. Although we need to thoroughly examine statistical significance, reproducibility, and routes causing this behavior in our future research.

4 Conclusion

In this chapter, we have investigated how a network's training performance is linked to a network's architecture, hardware components of the training machine, and software stack. We showed that a simple rule of thumb (e.g., always choosing the latest generation or the most powerful GPU) can increase training cost and time by an order of magnitude in the worst-case scenario. We also showed that components surrounding a GPU (e.g., the RAM and CPU) can cause significant performance bottlenecks and should be considered carefully in conjunction with a trained model architecture and implementation.

The overall results show that even in the case of a single GPU, the training setup costs can significantly vary by hundreds of percent.

We also showed that there is a meaningful variation of training time caused by the device driver version and the CUDA toolkit version, and that this variation is different for different GPU families.

References

1. MLPerf GitHub. https://github.com/mlperf/training
2. He, K., Gkioxari, G., Dollár, P., Girshick: R.: Mask R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2961–2969 (2017a)
3. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding, arXiv:1810.04805, 2019.
4. Naumov, M., et al.: Deep learning recommendation model for personalization and recommendation systems, arXiv:1906.00091 (2019)
5. Mattson, P., et al.: Mlperf training benchmark. arXiv:1910.01500 (2019)
6. Lin, T.-Y., et al.: Microsoft COCO: common objects in context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8693, pp. 740–755. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10602-1_48