Analysis and Exploitation of Dynamic Pricing in the Public Cloud for ML Training

Deepak Narayanan†, Keshav Santhanam†, Fiodar Kazhamiaka†,
Amar Phanishayee⋆, Matei Zaharia†
⋆Microsoft Research †Stanford University

ABSTRACT

Cloud providers offer instances with similar compute capabilities (for example, instances with different generations of GPUs like K80s, P100s, V100s) across many regions, availability zones, and on-demand and spot markets, with prices governed independently by individual supplies and demands. In this paper, using machine learning model training as an example application, we explore the potential cost reductions possible by leveraging this cross-cloud instance market. We present quantitative results on how the prices of cloud instances change with time, and how total costs can be decreased by considering this dynamic pricing market. Our preliminary experiments show that a) the optimal instance choice for a model is dependent on both the objective (e.g., cost, time, or combination) and the model's performance characteristics, b) the cost of moving training jobs between instances is cheap, c) jobs do not need to be preempted more frequently than once a day to leverage the benefits from spot instance price variations, and d) the cost of training a model can be decreased by as much as 3.5× compared to a static policy. We also look at contexts where users specify higher-level objectives over collections of jobs, show examples of policies for these contexts, and discuss additional challenges involved in making these cost reductions viable.

1. INTRODUCTION

Cloud providers like AWS, GCP, and Azure provide an opportunity for users to rent instances of many different types, in multiple regions and availability zones. In addition to reserved and on-demand cloud markets for long-term and guaranteed instances, many cloud providers offer a market for accessing unclaimed machines at lower cost, often referred to as the *spot market*. These instances are priced independently and dynamically, according to instance-specific supply and demand. In this paper, we explore the following question, which to our knowledge has not been broadly studied yet: **how much can a user benefit from a dynamic multi-cloud instance market?**

The primary challenge in taking advantage of spot pricing is that spot instances can be reclaimed or preempted at any time. Applications running on spot instances thus need to be easily stoppable; applications would then be restarted on another instance. Deep Neural Network (DNN) model training is a good example of an application suitable for spot instances; its iterative nature makes it conducive to preemption. DNN training is also compute-heavy, uses expensive instances with accelerators, and often uses a static read-only training data set that can be easily copied across clouds and availability zones. Consequently, we use DNN training as a target workload through the rest of this paper, and focus on answering three important questions.

How should cloud instances be chosen? A DNN model can be trained in the cloud using many instance types, with different accelerators (e.g., GPU generations like the K80, P100, V100; dedicated ML chips like the TPU [20]) and varying prices. DNN models are extremely diverse with many operator types, and show widely different performance behavior across instance types. The most appropriate choice of instance type depends on the *model* as well as the *user's objective* (e.g., throughput, cost, or a combination of the two, such as minimizing cost subject to a performance SLO like "complete job X in 10 hours").

Furthermore, spot instances, which are a cheap alternative to on-demand instances, are dynamic:

- Instances are priced differently across regions, availability zones, and cloud providers. These prices change with time as supply and demand change.
- A spot instance may be preempted at any time.
- Instances with multiple accelerators may be in less demand compared to an instance with a single accelerator of the same type, and consequently may be cheaper on a per-accelerator basis.

All these factors influence the optimal choice of instance.

How should higher-level objectives over multiple jobs be taken into account? Many organizations use public cloud instances to train models with the latest data on a repeated (e.g., daily) schedule. In such a use case, cost may not be the only objective to optimize for, e.g., some important jobs might have strict deadlines that must be met, even at a higher cost.

We present examples of *policies* that optimize for various higher-level objectives that can be specified over multiple jobs. Policies can be re-run whenever spot prices change to ensure that per-job allocations are computed using the

latest pricing information. Different policies can lead to dissimilar allocations: the allocation from optimizing cost alone with no job deadlines might place computation on slower instances that are cheaper, while minimizing cost with strict per-job deadlines might require some computation be executed on more expensive but faster instances to ensure these strict deadlines are met. Allocations must also respect real-world constraints, such as instance quotas.

How can real systems realize these cost-saving opportunities? Leveraging the spot market comes with many practical challenges, including dealing with instance preemption, determining how to schedule jobs on instances while respecting the computed allocation, responding to price changes, and transparently allowing movement of jobs between instances without user intervention. Our paper touches on these challenges in §5.

Summary of contributions. We measured the cost benefits of leveraging the dynamic multi-cloud instance market using AWS, GCP, and Azure instance prices collected over a month. We highlight the following key takeaways:

- The optimal instance type for a given model is dependent on both the target objective (cost, speed, or both) and performance characteristics of the model, even when using statically-priced instances.
- The cost of moving model checkpoints between instances is cheap. Moving input datasets is more expensive, but can be amortized over many jobs.
- Jobs do not need to be preempted more frequently than once a day to leverage the benefits from spot instance price variations. We observe that cloud providers today change instance prices at a much coarser granularity than before [3, 26]; this affects how systems leveraging the dynamic spot market should be designed.
- Instances themselves are usually preempted fairly infrequently (on the order of hours). In such cases, recent systems such as Spotnik [29], which provides fine-grained resilience to transient instance failures for distributed training, are not needed.
- The cost of training a model can be reduced by up to 3.5× (in practice, thousands of dollars) by making use of all available sources of price variation, including by up to 1.4× when enabling movement of applications across instances mid-computation.

Code and pricing data are open sourced at `https://github.com/stanford-futuredata/training_on_a_dime`.

2. BACKGROUND

In this section, we provide some background on DNN training and instance pricing in the public cloud.

Deep Neural Network (DNN) training. DNN training proceeds in iterations. In each iteration, the model processes a collection of training data inputs (called a minibatch), and subsequently updates its parameters using gradients derived from the minibatch. If training were interrupted, the model's parameters would need to be *checkpointed* to stable storage; state-of-the-art DNNs can have millions to billions of parameters. These model checkpoints then need to be loaded on the new worker to ensure that training progress is not lost. On-premise DNN schedulers leverage the fact that DNN training is iterative to suspend and resume training at iteration boundaries [16, 30].

Model	Throughput		Dollar-norm. Throughput	
	P100	V100	P100	V100
Transformer	**3.3**×	**3.3**×	**1.0**×	0.8×
A3C	1.2×	**2.2**×	0.4×	0.4×
CycleGAN	4.5×	**9.3**×	1.4×	**1.7**×
ResNet-18	4.0×	**6.8**×	**1.2**×	**1.2**×
ResNet-50	3.7×	**9.6**×	1.1×	**1.8**×

Table 1: Throughput and dollar-normalized throughput (using GCP on-demand prices) speedups with respect to a NVIDIA K80 GPU for various ML training workloads. The magnitude of speedup across GPU generations varies significantly across models, with later GPU generations (V100) faster. The V100 is no longer always optimal when considering dollar-normalized throughputs; dollar-normalized speedups are smaller across all models.

Pricing in public clouds. Cloud providers allow compute instances to be rented by users at fine granularities. The standard way to rent instances from public cloud providers involves using *on-demand* instances, which are guaranteed to be available at all times. Instances are hosted in different *regions*; each region has multiple availability zones.

Using on-demand instances for long durations can be expensive. As a cheaper alternative, cloud providers offer spot or preemptible instances, which can be preempted with little warning. Cloud providers usually price these instances in one of two ways: either the spot price changes (capped at the on-demand price) as demand changes (AWS and Azure), or the instances are offered at a constant price and can only be run for < 24 hours (GCP).

3. QUANTITATIVE ANALYSIS OF CLOUD PRICING

In this section, we pose two questions in the context of training various DNN models on instances with accelerators in the public cloud: (1) How should users go about picking which instance and accelerator type to use? (2) Instance pricing is dynamic, and changes across cloud providers, regions, availability zones, and over time. Can jobs leverage this information to achieve *better* allocations, as defined by the user's desired objective, by moving between instances (on the same or different cloud) over the course of training? Is this practical, given the overheads of moving model checkpoints and the associated input dataset?

3.1 Instance Type Choice for Various Models

Cloud providers like AWS, GCP, and Azure offer instances with various GPU types. Models use a diverse set of operators, leading to vastly different performance behavior on these hardware architectures. Table 1 shows the observed throughput speedups for various models and GPU types compared to a NVIDIA K80 GPU. While one of NVIDIA's more recent GPU offerings, the V100, out-performs other

Model	Dataset Size (GB)	Model Size (GB)	Dataset Cost	Model Cost
ResNet-50	150	0.098	9.13%	0.006%
BERT-Base	17	0.408	0.98%	0.025%

Table 2: Dataset and model sizes for ResNet-50 and BERT-Base architectures, along with the compute cost and egress costs (as a fraction of compute cost) for a single dataset and model transfer. Each transfer is from a North American region to the Internet. Each model transfer is extremely cheap. Dataset transfers are more expensive, but need to be performed only once per (dataset, cloud provider) pair.

GPUs for every model type, the relative speedup compared to the older K80 GPU is *model-dependent*, and varies from 2.2× to 9.6×. However, instances with V100 GPUs also cost more than instances with K80 GPUs.

The cost-effectiveness of instances for a particular model can be compared using the model's *cost-normalized throughput*. When normalizing by the GCP on-demand price (we use GCP since AWS does not offer P100 GPUs), we see that the K80 and P100 GPUs are superior compared to the V100 GPU for certain models, like A3C [15] and Transformer [18]. The best GPU for a given model on a cost basis can also change over time if using spot instances, which have dynamic pricing.

Moreover, users might have more nuanced deployments, where they have both cost and time budgets; in such situations, we may want to switch between instance types partway through training. For example, an optimal schedule may have a job spend 60% of training time on a cheap K80 GPU and the remaining 40% on a faster V100 GPU to minimize cost while respecting the provided time budget.

3.2 Dynamic Pricing for Cost Reduction

We now consider the various costs incurred when dynamically moving training jobs between instances.

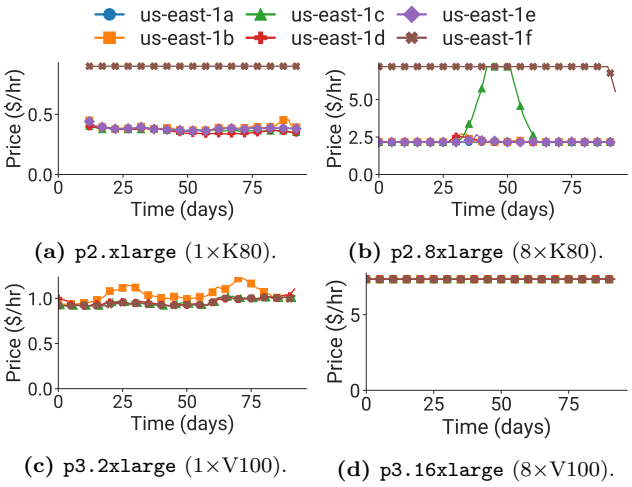
3.2.1 Cost of Data Movement between Clouds

Moving workloads between instances is only economical if the cost of the associated data transfer is less than the compute savings achieved by switching to the new instance. Table 2 lists the dataset and model sizes for two commonly benchmarked models (ResNet-50 [17] and BERT-Base [12]), as well as egress costs as a fraction of the cost of training these models for 160 hours on V100 spot instances. We use ImageNet [11] as the ResNet-50 dataset and English Wikipedia [5] as the BERT-Base dataset. The compute cost is measured as the cost of 160 V100-hours using spot instances. We use AWS prices for these measurements but find similar results across GCP and Azure. We approximate the cost of a single model transfer by computing the cost of 10,000 model transfers and dividing by 10,000. Ingress into each cloud is free, and does not need to be accounted for.

We observe that we can feasibly perform hundreds of transfers for each model before reaching even 10% of the compute cost, since the cost of transferring a single model checkpoint is cheap (on the order of cents). Furthermore, while a single dataset transfer is far more expensive than transferring a model checkpoint, the dataset need only be transferred once to each cloud during training and can be amortized over many jobs that use the same dataset. This transfer cost is zero if the user already has a copy of the input dataset available on all target clouds.

Cloud Provider	Region	GPU Type K80	P100	V100
Amazon (AWS)	`us-east-1`	2.7×	N/A	3.3×
Google (GCP)	`us-west-1`	3.4×	3.4×	3.3×
Microsoft (Azure)	`us-east-1`	7.3×	8.0×	5.1×

Table 3: Best-case cost reduction moving from on-demand instances to spot instances with a single GPU on each cloud. The best-case cost reduction varies widely with cloud provider; however, as we show later in Figure 2, availability also varies with cloud provider and instance type.



(a) `p2.xlarge` (1×K80). **(b)** `p2.8xlarge` (8×K80). **(c)** `p3.2xlarge` (1×V100). **(d)** `p3.16xlarge` (8×V100).

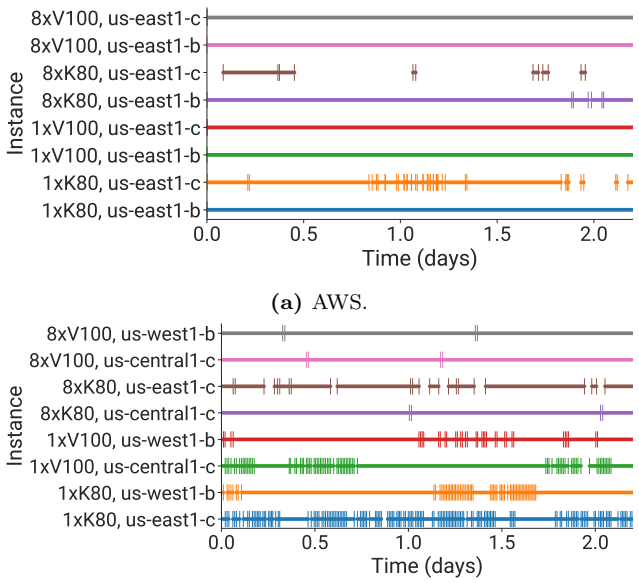
Figure 1: Per-hour price of AWS spot instances with various GPU accelerators in the `us-east-1` region. Prices can change with time and across availability zones, and are often capped at the on-demand price (`p2.xlarge`, `us-east-1f`). Some instances (`p3.16xlarge`) exhibit no price variation.

3.2.2 Volatility in Spot Instance Pricing for Compute

We collected spot instance prices for AWS and Azure over a month in February 2020; we were able to collect 3 months of backfilled data for AWS. We only include the most interesting graphs in this section; more graphs from our analysis are available at `https://github.com/stanford-futuredata/training_on_a_dime`.

Cost reduction from spot instances. Table 3 shows the best-case cost reduction observed when moving from an on-demand instance to a spot instance in the same region, for different clouds. Cost reductions vary from 2.7× to 8×.

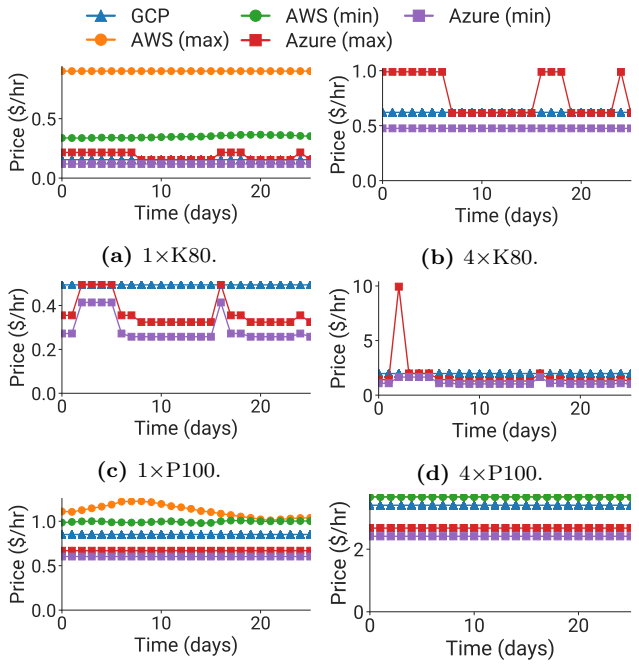
Variation of spot price with time. The price of spot instances can change with time as demand changes. Figure 1 shows the variation in spot prices for various instances with GPUs in the AWS `us-east-1` region. We observe that price changes across regions are not highly correlated with each other, with some regions capped at the on-demand price. The cheapest availability zone in a region can change with time. We also observe that some instances show extremely stable pricing (`p3.16xlarge`).



(a) AWS.

(b) GCP.

Figure 2: Availability of AWS and GCP preemptible instances. Vertical lines at the start of a horizontal line show the time at which the request was granted, and vertical lines at the end of a horizontal line show the time at which the instance was preempted. The frequency of preemption changes with both availability zone and instance type. GCP preempts instances at least every 24 hours.



(a) 1×K80.

(b) 4×K80.

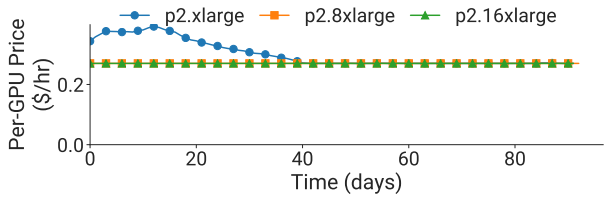
(c) 1×P100.

(d) 4×P100.

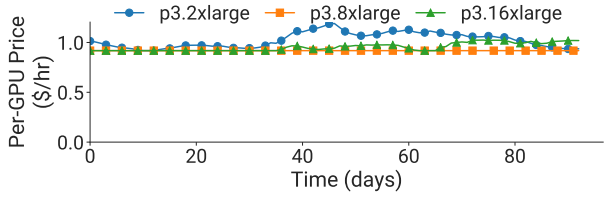
(e) 1×V100.

(f) 4×V100.

Figure 3: Minimum and maximum spot price over all availability zones and regions in the US for various cloud providers. GCP uses a static pricing model. Instance types have different relative orderings, and at any given time, the ordering can change (e.g., as in Figure 3d).



(a) K80.



(b) V100.

Figure 4: Normalized cost on a per-GPU basis for instances with K80 and V100 GPUs. Instances with K80 GPUs have 1, 8, and 16 GPUs, while instances with V100 GPUs have 1, 4, and 8 GPUs. We found that instances with a greater number of GPUs generally exhibit more stable pricing.

Availability. GCP adopts an alternate pricing model for preemptible instances: prices stay constant, but instances might be preempted when demand exceeds supply. Figure 2 shows timelines of availability for instances with GPUs on AWS and GCP. Instances on AWS are more reliably available for longer (not capped at 24 hours). Instances in some regions were preempted more often than others (greater frequency of vertical lines); 8×GPU instances were preempted less frequently on GCP. Preemption is preceded by a 2-minute warning which can be used to checkpoint the model. For most regions and instance types on AWS, preemption is relatively infrequent (order of hours instead of minutes).

Instance prices across clouds. Figure 3 shows the price of the cheapest and most expensive instances with different numbers of accelerators across clouds. The cheapest cloud provider changes with instance type. In some cases (not shown), GCP is the cheapest option, but jobs are preempted after at most 24 hours.

Per-GPU price for multi-GPU instances. We also studied the variation of price on a per-GPU basis across instances with different numbers of the same GPU type (e.g., AWS has 1×, 8×, and 16×K80 instances). As shown in Figure 4, we found that on a per-GPU basis, instances with a larger number of GPUs have more stable pricing. However, a user may need to pack multiple jobs onto the larger instance (or run a single multi-GPU job) to fully utilize it.

3.2.3 *End-to-end Cost Reduction*

We show the net reduction in *compute* cost of training a single ML model using all these sources of price variation in Figure 5. Each ML training job takes 4 days to complete, and we show price reductions for single-GPU jobs for simplicity. All strategies before *multi-cloud* use AWS instances with GPUs in the `us-east-1` region; *multi-cloud* and *dynamic* pick instances across AWS and Azure. *GPU type* chooses the GPU with best cost-normalized through-

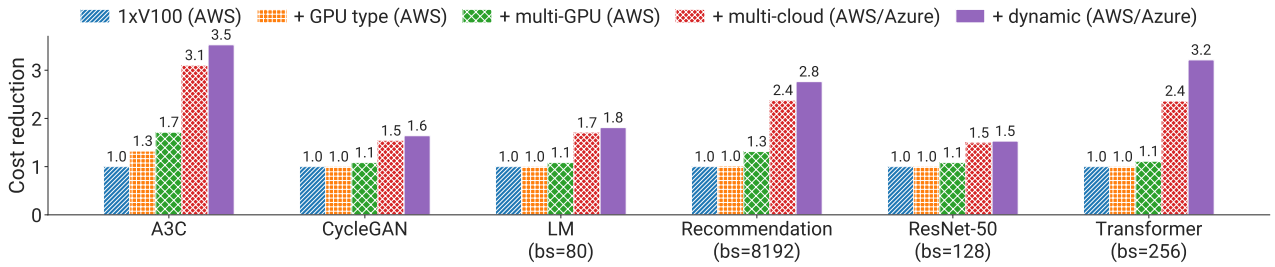


Figure 5: Average cost reduction to run the same number of training iterations (4 V100-days of computation), while cumulatively adding more sources of price variation. **1×V100** uses the cheapest 1×V100 instance within the `us-east-1` AWS region, **GPU type** chooses the GPU with highest cost-normalized throughput, **multi-GPU** picks instances with multiple GPUs if they are cheaper on a per-GPU basis; all these strategies use AWS instances only. The **multi-cloud** strategy picks the cheapest instance across AWS and Azure at the start of training, and then sticks with this choice throughout training. **Dynamic** continually picks the cheapest instance across AWS and Azure through training as prices change. Costs reduce as sources of price variation are added.

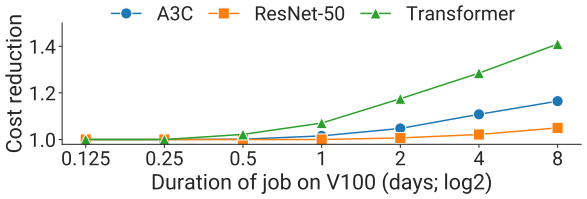


Figure 6: Average cost reduction from allowing *dynamic* switching of instance type, cloud, and availability zone during training, while varying job duration. Longer jobs are able to make use of greater variability in prices over longer horizons, consequently leading to larger cost reductions. The right two bars in Figure 5 shows the impact of dynamic switching for jobs with a duration of 4 V100-days.

put (instead of 1×V100 instances) when the job starts and then sticks with that choice throughout, *multi-GPU* picks instances with multiple accelerators if they are cheaper on a per-GPU basis, and *dynamic* adapts the choice of instance through training as prices change. All results assume that datasets are available on each cloud.

We can reduce costs by up to 3.5× compared to the baseline of using the cheapest 1×V100 instance. The effectiveness of each strategy depends on the GPU type where the model has the highest cost-normalized throughput (Table 1), which can change with time depending on the pricing behavior of these instance types across AWS and Azure. For example, ResNet-50 [17] is always cheapest on V100 instances, which show stable pricing; consequently, cost reductions are minimal. We note that the movement of checkpoints is extremely cheap (cents / transfer) and the number of transfers is small, since prices change only daily and not every price change leads to an instance switch.

Impact of job duration on effectiveness of dynamic scheduling. We further study the impact of job duration on cost savings when using dynamic scheduling, where jobs can be moved between instances as training proceeds and the initial instance choice is not locked in through the duration of training. In Figure 6, we show the cost reduction of switching instances across GPU types, availability zones, and clouds during training as job duration changes compared to using the best option across cloud providers at the start of training and sticking with this choice (red and purple bars in Figure 5). We see a cost reduction of up to 1.4× for long-duration jobs that can take advantage of pricing over longer horizons. Long-duration training jobs are common as models become larger. For example, the recently released GPT-3 model [9] requires about 100 V100-years of training computation.

Cost reductions vary across models since cost-normalized throughputs for different models can change with time, e.g., the Transformer model switches between the Azure K80 and P100 instances. Cost reductions are small for short-duration jobs since instance pricing is stable over the short term (≤ 2 days). The number of switches between instances needed for these cost savings is small (≤ 3). We note that even though we only looked at single-GPU jobs in this section, the cost savings are valid even for multi-GPU jobs. In particular, the durations of distributed jobs which use many GPUs is still often on the order of weeks to months [9].

4. HIGHER-LEVEL OBJECTIVES

When training a collection of ML models, users might want to allocate resources while optimizing for higher-level objectives. For example, users might want to minimize cost alone, or minimize cost subject to performance SLOs (e.g., complete training in the next 12 hours), or minimize the time needed to complete a collection of training jobs with a given cost budget.

Representing allocations and throughputs. As we noted earlier, optimizing more complex objectives might result in allocations where jobs move dynamically between instance types. As a result, allocations need to specify the fraction of wall clock time a training job should spend on each instance type. We represent these allocations as allocation matrices X. X_{mj} is the fraction of time model m should spend on instance type j. As we shall show, scheduling policies can be expressed as optimization problems involving X that try to maximize or minimize an appropriate objective function. Objective functions can be written in terms of *effective throughput* [22], the time-weighted average throughput across instance types; given the relative performance of each job on each instance type (T), the effective throughput of a model m, throughput$_T(m, X)$, is simply $\sum_j T_{mj} \cdot X_{mj}$. Since ML training is iterative with stable performance across iterations, T can be estimated from short timing runs.

4.1 Baseline: Maximizing Total Throughput

Maximizing the total effective throughput achieved by a collection of jobs can be achieved by solving the following optimization problem.

$$\text{Maximize}_X \sum_m \text{throughput}_T(m, X)$$

We add the following constraints to ensure that each job is not over-allocated, and worker quotas are not exceeded.

$$\begin{aligned} \textstyle\sum_j X_{mj} &\leq 1 \qquad \forall m \\ \textstyle\sum_m X_{mj} &\leq \text{quota}_j \quad \forall j \end{aligned}$$

4.2 Minimizing Total Cost

The above policy can be extended to incorporate cost. To minimize total cost of training, one can optimize,

$$\text{Maximize}_X \sum_m \frac{\text{throughput}_T(m, X)}{\text{cost}(m, X)}$$

Here, $\text{cost}(m, X)$ is effective cost, computed as $\sum_j c_j \cdot X_{mj}$, where c_j is the per-hour cost of instance type j.

The numerator in each objective term represents the effective throughput in samples per unit time, the denominator represents the effective cost in dollars per unit time, and the resulting fraction is the effective normalized throughput in samples per dollar.

4.3 Objectives with Both Throughput and Cost

Jobs can have time SLOs as well, e.g., certain high-priority jobs might need to complete every 12 hours. We can add additional constraints: given SLO_m for each model m (models without SLOs can have SLO_m set to ∞),

$$\text{throughput}_T(m, X) \geq \text{num\_iterations}_m / \text{SLO}_m$$

Similarly, one could also formulate policies with a minimize makespan (time taken to complete all jobs in a collection) objective, while keeping the cost within a prescribed cost budget B. The objective here would be,

$$\text{Minimize}_X M$$

M is the makespan. In addition to the constraints above that ensure that each job is not-allocated and worker quotas are not exceeded, we need constraints that ensure that every job completes within this makespan M, while also staying within the cost budget B,

$$\begin{aligned} \frac{\text{num\_iterations}_m}{M} &\leq \text{throughput}_T(m, X) \quad \forall m \\ M \cdot (\textstyle\sum_m \text{cost}_T(m, X)) &\leq B \end{aligned}$$

This can be solved by binary searching for the smallest M with a feasible solution satisfying the above constraints.

5. SYSTEM DESIGN CONSIDERATIONS & DISCUSSION

In this section, we discuss important design considerations that real systems need to address to be able to deliver these cost reductions in a transparent way. We also highlight some open questions that we think are worth reflecting on.

Scheduling of applications on physical instances. Given a theoretical allocation computed from a policy, how should resources be allocated to applications, considering quotas on instances and applications that span multiple accelerators? In multi-cloud settings, how should datasets be streamed between clouds when not already available? How should instance preemptions be handled?

API between the scheduler and applications. An application can be moved either when the scheduler decides to take advantage of a pricing change, or when a spot instance is preempted by the cloud provider. How can we enable the movement of applications between clouds, regions, and availability zones seamlessly without user involvement?

These questions are especially pertinent with distributed training where state, such as IP addresses of participating workers, needs to be reset when preemptions occur. Fortunately, both forced and voluntary preemptions are relatively infrequent (as can be seen in Figure 2 and §3.2.3), meaning the cost of reconfiguration can be easily amortized away without using sophisticated failover mechanisms like those proposed in Spotnik [29]. Recent work [24] has demonstrated how state in the Horovod communication library [25] can be reset with minimal user intervention when using elastic resources; similar techniques can be used for other communication libraries as well.

Instance preemption. Spot instances are preempted at different rates (Figure 2). How should one model the preemptions of instances? This is important since users might be willing to pay more for a more reliable instance. Can we estimate the mean time to failure to decide which instance types to use?

Spot instance pricing. Our measurements raise the following questions about how spot instances are priced: Why do availability zones in the same region show different pricing? Why do instance preemptions happen even when the instantaneous spot price is lower than the on-demand price?

Market movement. What happens if all cloud users exploit the cost inefficiencies described in this paper, and use regions and availability zones with cheaper and / or more stable pricing? Can this help with price smoothing, with each of the different AZs showing more similar pricing as demand equalizes? In other words, will drastic changes in demand based on the movement of applications to cheaper regions and availability zones cause prices to shift?

Incentivizing easier and more efficient multi-cloud deployments. In times of high demand, cloud providers can preempt spot instances. In such cases, it might make sense for a user to take their computation to a different cloud provider – this not only could give the user a better experience, but can also improve the experience of all other users by reducing demand and consequently the likelihood of preemption. An auction system where cloud providers can bid for a small fraction of another cloud provider's jobs could solve this problem – the original cloud can receive a small commission for forwarding the job to another cloud while also partially alleviating demand, the bidding cloud receives additional business that it might not have otherwise received, and users receives better service.

ML inference. Even though we only considered ML training as a target application in this paper, we believe ML inference is an interesting target application as well. ML in-

ference, however, introduces different challenges: in particular, instances need to be provisioned keeping system load in mind, since system load has downstream ramifications on other metrics of interest like application latency. Unlike training, where users mostly care about just throughput and consequently total time needed to train a model end-to-end, inference applications have a number of performance-related metrics of interest, such as average latency, tail latency, throughput, and throughput subject to latency constraints. Each of these performance metrics can be combined with cost. How does one optimize for these different objectives? Additionally, serverless offerings such as AWS Lambda and Google Cloud Functions [2, 6] can be used in the inference context; however, these do not come with accelerators attached. Can inference on cheap CPU cores for short durations compete with more expensive but faster accelerators?

Packing multiple applications onto a single accelerator. Concurrently executing multiple models on the same GPU using NVIDIA's Multi Process Service (MPS), CUDA streams, or new features like Multi-Instance GPU (MIG) on the just released A100 GPU can help improve utilization [19, 1, 23, 7]. Can this be used to further reduce cost and improve resource utilization for end users?

Performance modeling of applications. Instead of relying on timing runs for each application on each instance type, can we learn a performance model that predicts runtimes of applications? Can we use this in settings where multiple applications are packed onto a single instance?

Other applications. What other applications are long-lived and amenable to such optimizations? For example, are physical simulations a good fit? How can one get around the fact that performance in other applications might be less predictable, making optimization more challenging?

6. RELATED WORK

Existing work has looked at two ways to minimize cloud costs: performance modeling for instance sizing, and leveraging the spot market. However, no prior work considers *both*; prior work also does not specify how objectives over multiple jobs can be specified and acted upon in this setting.

Minimizing costs in the cloud. Existing systems, such as LLOOVIA [13, 14] and other resource provisioning systems [27], have taken advantage of multi-cloud to minimize costs, but have focused on on-demand and reserved cloud markets. AWS offers EC2 Fleet [4], a service that can launch multiple on-demand and spot instances within a maximum budget. Other systems have proposed using spot instances for DNN training. DeepSpotCloud [21] takes advantage of price differences within availability zones and regions. HotSpot [26] and Stratus [10] are cost-aware schedulers that move CPU jobs between spot instances to take advantage of dynamic pricing. However, all of these systems use instances of *pre-specified* types, do not account for the heterogeneity of application performance across instance types, and cannot determine the optimal instance type for a given job and objective.

Selecting instance types. Existing work has looked at picking the right instance type for different classes of applications. Ernest [28] and CherryPick [8] try to predict the runtime performance of various applications on instance types available in the cloud, but do not consider spot pricing of instances, and do not specify how these performance models can be used downstream to optimize for various higher-level objectives.

7. CONCLUSION

In this paper, we analyzed the impact of the dynamic pricing market in public clouds on the cost of performing ML training. We found that moving jobs between instances is cheap, that jobs can be preempted fairly rarely (once a day) to leverage the benefits from price variations, that jobs themselves are preempted fairly rarely by the cloud provider, and that the cost of end-to-end training for a given model can be reduced by up to 3.5× by exploiting the different sources of price variation. We also showed how one can write policies that optimize combinations of speed and cost for collections of jobs. We believe this is is an exciting area of future work, with applications to many other domains besides ML training.

8. ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their feedback. This research was supported in part by affiliate members and other supporters of the Stanford DAWN project (Ant Financial, Facebook, Google, Infosys, NEC, VMware), as well as Toyota Research Institute, Cisco, SAP, NSF CAREER grant CNS-1651570, and NSFGRF grant DGE-1656518. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Toyota Research Institute (TRI) provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

9. REFERENCES

[1] NVIDIA Multi-Process Service. `https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf`, 2019.

[2] AWS Lambda. `https://aws.amazon.com/lambda/`, 2020.

[3] AWS Spot Pricing Model. `https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing/`, 2020.

[4] EC2 Fleet. `https://docs.amazonaws.cn/en_us/AWSEC2/latest/UserGuide/ec2-fleet.html`, 2020.

[5] English Wikipedia. `https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2`, 2020.

[6] Google Cloud Functions. `https://cloud.google.com/functions`, 2020.

[7] NVIDIA Ampere Architecture. `https://devblogs.nvidia.com/nvidia-ampere-architecture-in-depth/`, 2020.

[8] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 469–482, 2017.

[9] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165*, 2020.

[10] A. Chung, J. W. Park, and G. R. Ganger. Stratus: Cost-Aware Container Scheduling in the Public Cloud. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 121–134, 2018.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[13] J. L. Díaz, J. Entrialgo, M. García, J. García, and D. F. García. Optimal Allocation of Virtual Machines in Multi-Cloud Environments with Reserved and On-demand Pricing. *Future Generation Computer Systems*, 71:129–144, 2017.

[14] J. Entrialgo, J. L. Díaz, J. García, M. García, and D. F. García. Cost Minimization of Virtual Machine Allocation in Public Clouds Considering Multiple Applications. In *International Conference on the Economics of Grids, Clouds, Systems, and Services*, pages 147–161, 2017.

[15] D. Griffis. RL A3C PyTorch. `https://github.com/dgriff777/rl_a3c_pytorch`, 2018.

[16] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo. Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 485–500, 2019.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[18] Y.-H. Huang. Attention is All You Need: A PyTorch Implementation. `https://github.com/jadore801120/attention-is-all-you-need-pytorch`, 2018.

[19] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, pages 947–960, 2019.

[20] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12, 2017.

[21] K. Lee and M. Son. DeepSpotCloud: Leveraging Cross-Region GPU Spot Instances for Deep Learning. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 98–105, 2017.

[22] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. *arXiv preprint arXiv:2008.09213*, 2020.

[23] D. Narayanan, K. Santhanam, A. Phanishayee, and M. Zaharia. Accelerating Deep Learning Workloads through Efficient Multi-Model Execution. In *NeurIPS Workshop on Systems for Machine Learning*, 2018.

[24] A. Or, H. Zhang, and M. Freedman. Resource Elasticity in Distributed Deep Learning. In *Proceedings of Machine Learning and Systems 2020*, pages 400–411. 2020.

[25] A. Sergeev and M. Del Balso. Horovod: Fast and Easy Distributed Deep Learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

[26] S. Shastri and D. Irwin. HotSpot: Automated Server Hopping in Cloud Spot Markets. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 493–505, 2017.

[27] S. N. Srirama and A. Ostovar. Optimal Resource Provisioning for Scaling Enterprise Applications on the Cloud. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 262–271, 2014.

[28] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 363–378, 2016.

[29] M. Wagenländer, L. Mai, G. Li, and P. Pietzuch. Spotnik: Designing Distributed Machine Learning for Transient Cloud Resources. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, 2020.

[30] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 595–610, 2018.