Spotnik: Designing Distributed Machine Learning for Transient Cloud Resources

Spotnik: Designing Distributed Machine Learning for Transient Cloud Resources

Marcel Wagenländer, Luo Mai, Guo Li, Peter Pietzuch
Imperial College London

Abstract

To achieve higher utilisation, cloud providers offer VMs with GPUs as lower-cost *transient* cloud resources. Transient VMs can be revoked at short notice and vary in their availability. This poses challenges to distributed machine learning (ML) jobs, which perform long-running stateful computation in which many workers maintain and synchronise model replicas. With transient VMs, existing systems either require a fixed number of reserved VMs or degrade performance when recovering from revoked transient VMs.

We believe that future distributed ML systems must be designed from the ground up for transient cloud resources. This paper describes SPOTNIK, a system for training ML models that features a more adaptive design to accommodate transient VMs: (i) SPOTNIK uses an adaptive implementation of the all-reduce collective communication operation. As workers on transient VMs are revoked, SPOTNIK updates its membership and uses the all-reduce ring to recover; and (ii) SPOTNIK supports the adaptation of the synchronisation strategy between workers. This allows a training job to switch between different strategies in response to the revocation of transient VMs. Our experiments show that, after VM revocation, SPOTNIK recovers training within 300 ms for ResNet/ImageNet.

1 Introduction

Cloud providers offer *transient cloud resources*, e.g., transient virtual machines (VMs) with GPUs, which are periodically auctioned off to the highest bidder and therefore can be revoked at short notice. Amazon AWS supports EC2 Spot Instances since 2009 [3], which are revoked with a two-minutes notice; Microsoft announced Azure Spot VMs in 2019 [5], which have a 30-second revocation warning. From an economic point-of-view, the pricing of transient VMs follows the law of free markets. A free market is supposed to find the optimal price between the cloud providers and its customers through supply and demand. Transient VMs therefore allow providers to increase the utilisation of expensive hardware.

The distributed training of machine learning (ML) models, in particular through deep learning, is resource-hungry. In a distributed ML job, *workers* in a cluster train independent model replicas in parallel and synchronise at the end of each training step. For synchronisation, workers either use a central *parameter server* [26] or decentralised *collective communication*, e.g., following an *all-reduce* pattern [43].

In many cloud settings, large-scale ML jobs thus have become a dominant workload [11]. Large models and datasets yield higher accuracies at the cost of using more cloud resources (e.g., GPUs or TPUs) and having longer training times. Therefore, there is an economic incentive for distributed training on transient VMs, which can yield cost reductions of up to 90% compared to regular reserved VMs [1].

We argue that it should be possible for distributed ML jobs to run *exclusively* on transient VMs. Systems, such as PyTorch [38] and TensorFlow [49], however, face challenges when running on a transient cloud cluster: (i) workers may be added and removed from the cluster when transient VMs become available and are revoked, respectively. The system must be designed to support such dynamic changes, and these changes must not prolong the training process; and (ii) as a cluster changes based on the available transient VMs, configuration parameters of the ML job are affected. For example, a synchronisation strategy that is effective with few workers, such as *synchronous stochastic gradient descent* (S-SGD) [23], may suffer from network bottlenecks with many workers. Therefore, it is hard to deploy a distributed ML job in current systems with *one* configuration that achieves the best performance irrespective of the available transient VMs.

Existing work that attempts to address the above problems through a *hybrid* approach, in which a system utilises both transient and reserved VMs [17, 36]. Since reserved VMs cannot be revoked, systems use them e.g., to run parameter servers. This, however, increases the number of reserved VMs proportionally with more transient VMs, reducing cost benefits. Alternative approaches rely on expensive recovery mechanisms when transient VMs are revoked. For example, workers may periodically checkpoint their state to external storage, pausing the training during checkpointing [25, 33]. When transient VMs are revoked, the state of the lost worker is

recovered from the checkpoint, which increases job runtime.

In contrast, we want to design a distributed ML system that runs *exclusively* and *efficiently* on dynamic transient VMs. We want the design to be future-proof: even when revocation times are reduced from minutes to seconds, a distributed ML job should achieve fast convergence. In general, the less time is spent on handling revocations, the more time can be used by training tasks. This becomes especially important in serverless cloud environments [2, 4, 19] in which functions are short-running and there is no concept of revocation notifications.

The paper introduces SPOTNIK, a distributed ML system for transient VMs that explores two ideas:

(1) Adaptive collective communication. SPOTNIK uses a novel adaptive collective communication layer that handles dynamic changes to membership of the cluster. The layer (i) detects worker failure due to revoked transient VMs, (ii) negotiates a new membership among all running workers; and (iii) notifies the workers to continue training with the new membership. To handle revocation, this layer embeds control messages as part of an all-reduce operation so that workers can mutually detect revocation and recover the system by exchanging local states. To reduce performance overheads, the control messages reuse the scalable all-reduce graph, and their number sent by each worker is bounded.

(2) Adaptive worker synchronisation. To synchronise workers efficiently irrespective of the cluster size, SPOTNIK adapts its synchronisation strategy at runtime with low overhead. It can shift synchronisation among *synchronous stochastic gradient descent* (S-SGD) [23], *synchronous model averaging* (S-MA) [22] and *asynchronous decentralised parallel SGD* (AD-PSGD) [28]. For this, SPOTNIK relies on high-level synchronisation operators that cover both collective operations, such as computing global averaged gradients, and point-to-point operations, such as pair-wise model averaging. Changing strategies has low overhead because it does not require the allocation of new system resources. SPOTNIK makes a decision to change the synchronisation strategy by monitoring workers.

2 ML training with transient cloud resources

2.1 Distributed ML training in cloud

Distributed ML systems [38, 49] use parallel workers to split the training workload for large ML models. Each worker samples a mini-batch of data from the training dataset and trains its model replica using the mini-batch *stochastic gradient descent* (SGD) algorithm [42]. As replicas are trained with different mini-batches, their parameters (i.e., weights) diverge after each training iteration. The system therefore must use a *synchronisation strategy* to control the divergence of model replicas. A number of such strategies exist, e.g., *synchronous SGD* (S-SGD) [23] updates model replicas using averaged gradients; *model averaging* [22] coordinates model replicas using a central model. A synchronisation strategy can be implemented in different ways, and there are two major approaches used today: *all-reduce* organises the workers in a ring topology, which is used to compute aggregated gradients [43] or a *parameter server* allows workers to send their local gradients and responds with aggregated gradients [26].

When users deploy a ML job in the cloud, they can scale it using a cluster of VMs, which typically includes accelerators such as GPUs for tensor computation. In practice, users face a high monetary costs when using dedicated VMs for training. For example, training a BERT-like model with Megatron-LM [45] on 512 NVIDIA V100 GPUs on Azure costs $0.56 per second.

2.2 Exploiting transient cloud resources

Transient cloud resources, such as Google Preemptible VMs [15], Azure low-priority/spot VMs [6], AWS Spot Instances [1] and IBM Transient VMs [20] have emerged as a means to reduce the cost of cloud computing. Transient VMs offer weak service level agreements (SLAs) and can be *revoked* when the provider needs the capacity back. For example, AWS and Azure set a dynamic market price for VMs and revoke VMs if the spot price goes above the price declared by users; Google and Azure also provide preemptible VMs, which are purchased at fixed discount prices but can be reclaimed after a 30-second notification.

Cloud providers offer transient VMs because they have benefits for both cloud providers and users. In data centres, cloud providers must maintain redundant resources for planned growth, which can be sold as transient VMs before they are reserved by future customers. Furthermore, data centres show a diurnal pattern [14]. In off-peak times, they may have substantial idle resources, which can be monetised as transient resources to improve utilisation.

Cloud users can significantly reduce their costs of using cloud GPUs by saving up to 90% compared to reserved instances [1]. Training with transient VMs offers users substantially lower GPU costs: with the same budget, a user can use 10 times more GPUs, reducing training times by 80% [27]. In the future, transient cloud resources are likely to grow in popularity and thus become the dominant resource type for distributed ML systems.

2.3 Challenges in using transient resources

In practice, performing distributed ML training with transient cloud resources is challenging:

(1) Workers may lose resources at any time. The revocation of transient VMs adversely affects distributed training. Workers maintain large intermediate training state, and revoking a worker requires this state to be either discarded, compromising consistency and accuracy of the trained model [7], or migrated to another worker, adding I/O load and affecting training performance [26].

In addition, distributed ML systems require global computation barriers across workers for the synchronisation of

model replicas. When a worker is revoked due to the loss of a transient VM, it may cause the entire computation to block [32, 35]. In such a case, the system must restarted manually from a checkpoint, losing performance [25].

(2) Resource availability may change substantially. Depending on the magnitude of a resource bid and the ratio of the transient resource price to the on-demand one [44], the probability of revocation increases. VMs with GPUs in AWS are among the ones with the highest revocation probability [47]. As a result, the cluster size may vary substantially over time if a user's spending remains fixed.

Such dynamic changes to the cluster size shift the performance bottlenecks during the training process. This makes it hard for users to choose a single ML job configuration that achieves the best training performance. For example, the overhead of maintaining a global barrier is smaller with tens of workers compared to hundreds. In addition, larger clusters have a higher probability of stragglers [10], and thus require extra mechanisms for their mitigation.

2.4 Existing approaches and their limitations

We observe several reasons why existing distributed ML training systems are ill-suited for using transient cloud resources.

(1) Inefficient recovery with transient resources. When faced with the revocation of transient VMs, existing systems broadly follow two approaches: MPI-based distributed systems such as Horovod [43] treat revocations as failures. They must collect periodic checkpoints and recover from the last checkpoint when the state of a worker was lost, In practice, this results in high performance overheads (see §4).

Systems that use parameter servers [17] place them typically on expensive reserved VMs. In a hybrid model, a user's budget is split across transient and reserved VMs for workers and parameter servers, respectively. The split is workload-specific, making it hard to find a good compromise [52].

Systems that scale elastically assume that scaling is possible at any time [41]. When training with transient resources, however, the environment controls the available resources, and new resources may be unavailable. Prior work [41] searches for the optimal cluster configuration for a training job; we want to do the opposite, i.e., optimise the training job for a given cloud environment.

(2) No mechanisms to adapt to resource changes. Current distributed ML systems require users to select parameters of the training process, such as the synchronisation strategy, at job deployment time. The choice of S-SGD, the de-facto standard, is hard-coded as part of most training programs. This makes it hard for systems to scale when more VMs become available. Synchronisation strategies for large clusters, such as AD-PSGD [28], are poor general choices because they exhibit lower training performance than S-SGD on small clusters. This puts users in a dilemma: they prefer to use S-SGD for high training accuracy with few transient VMs, but then they sacrifice scalability when more VMs become available.

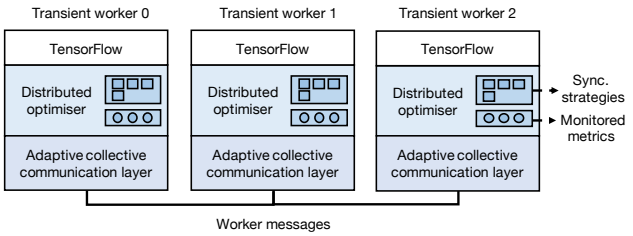

Figure 1: SPOTNIK design

3 SPOTNIK design

3.1 Architecture overview

The design of SPOTNIK is driven by several goals: (i) we want a distributed ML job that uses transient VMs to remain identical to one with reserved VMs. This makes it easy for developers to adopt SPOTNIK in practice; (ii) we want to handle the revocation of transient VMs during training with low overhead. This means that SPOTNIK must support fast revocation times; and (iii) we want to adapt the training cluster when the number of transient resource changes. In particular, SPOTNIK must avoid network performance bottlenecks.

At a high-level, SPOTNIK provides a distributed optimiser abstraction inspired by the optimiser interface of popular distributed ML libraries such as Horovod [43], TensorFlow Distributed Strategy [50] and BytePS [37]. To use SPOTNIK, users embed a *Spotnik distributed optimiser* (shown in Fig. 1) into their TensorFlow programs. This makes the use of transient VMs transparent: the optimiser allocates/deallocates resources, and replicates the training programs on workers.

Workers in SPOTNIK use an *adaptive collective communication* layer (see Fig. 1) to synchronise model replicas. A unique feature of this layer is that it can transparently recover from VM revocations, avoiding the need to restart the training job as in existing collective communication libraries, such as OpenMPI [35] and NCCL [32]. The layer detects revocations and recovers training state, as part of the regular synchronisation communication between workers. This removes the need for the user to handle revocations explicitly.

In addition, SPOTNIK supports *adaptive synchronisation strategies*. It shifts to a different synchronisation strategy in response to changes in transient VM availability. Instead of hard-coding a synchronisation strategy in a training job, the SPOTNIK distributed optimiser contains a set of *candidate* synchronisation strategies (see Fig. 1). It then *dynamically* selects one according to monitored cluster metrics, such as the current number of transient VMs.

3.2 Adaptive collective communication

We explore a design for SPOTNIK that allows it to recover efficiently from the revocation of transient VMs. SPOTNIK must coordinate potentially hundreds of workers. Each worker computes model gradients and launches parallel collective communication operations such as ring all-reduce to aggregate gradients. If a worker is revoked in the middle of this

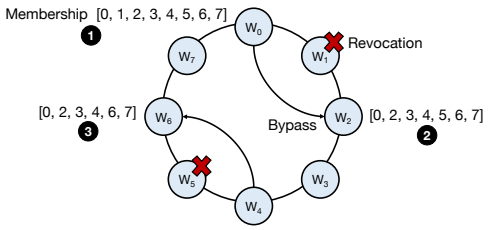

Figure 2: Example of repairing a broken all-reduce ring

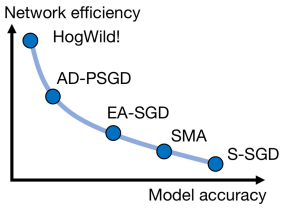

Figure 3: Landscape of synchronisation strategies

operation, its downstream workers in the all-reduce topology cannot proceed, potentially blocking the training job.

To address this issue, SPOTNIK exploits the following idea: since workers have already constructed a collective communication topology (all-reduce) and have replicated training states, they can detect revocation and coordinate among themselves to continue training by reusing existing state. Note that the design of SPOTNIK does not rely on workers receiving revocation notifications, making this approach applicable irrespective of the notification interval.

SPOTNIK realises this approach as follows: workers exchange control messages to handle revocation in a collective communication operation. Here, each worker must ensure that (i) control messages do not incur a performance overhead; and (ii) worker states, which are asynchronously updated by collective communication operations, can be reused safely. SPOTNIK achieves this through an *adaptive collective communication* layer, which has two parts:

(1) Revocation recovery algorithm. We design an algorithm that can handle revocations within a ring topology for collective communication topology, such as all-reduce. It reuses the existing data connections in the topology to pass control messages. The number of messages sent by each worker is bound by $O(K)$, where K is the number of concurrent revocations, which allows it to scale to many transient VMs.

We explain the algorithm in Fig. 2. This algorithm has three phases: *reconcile*, *accept* and *restart*. We assume that workers exchange messages on a ring, a representative collective communication topology, to synchronise model replicas. On the ring, each worker monitors the health of its neighbours. Assuming worker W_0 detects that W_1 has been revoked, W_0 removes W_1 (step ❶) from the membership, and propagates the new membership through the ring (step ❷). During this process, W_4 finds that W_5 has been revoked (i.e., there are multiple revocations). It removes W_5 from the membership and continue the propagation process (step ❸). The process terminates at a worker (i.e., W_6 in this example), which receives a membership list that is equal to the local one, indicating that all workers have reconciled the new membership. This worker initiates another propagation to have workers accept the new membership (accept phase). Finally, it propagates a message to restart training on all workers (restart phase).

(2) Atomic worker state update. SPOTNIK must ensure atomicity when modifying worker state. Distributed ML systems such as TensorFlow launch asynchronous all-reduce operations that overlap training and synchronisation. SPOTNIK must protect the integrity of worker state while it is modified concurrently, otherwise workers would observe a dirty state that has been *partially* updated.

Instead of directly updating parameters, SPOTNIK waits for *all* averaged gradients to be ready and applies them in an atomic fashion. The caching process has non-negligible performance overheads: SPOTNIK coordinates all-reduce operations in the background without blocking training. This is achieved through an all-reduce scheduler embedded in the dataflow graph that asynchronously receives notifications regarding events in the communication layer. If any all-reduce operation fails due to a worker revocation, the scheduler aborts the remaining all-reduce operations and discards cached results. The remaining workers then continue with the new membership state and re-try the failed training iteration.

3.3 Adapting synchronisation strategies

Past work has proposed a wide spectrum of synchronisation strategies for clusters of different sizes, which vary in terms of network efficiency and model accuracy. Fig. 3 compares several synchronisation strategies: on one end, synchronous strategies such as S-SGD [23] and SMA [22] incorporate model updates from all workers for improved model accuracy; while HogWild! [40], AD-PSGD [28] and EA-SGD [54] remove expensive barriers to avoid network bottlenecks, assuming that training can reach a targeted accuracy sooner.

Since the number of transient VMs, and thus the network load, can change substantially during training, SPOTNIK adapts the choice of synchronisation strategy. This is challenging to achieve in existing synchronisation libraries such as Horovod [43], which hide low-level communication primitives and expose only an S-SGD implementation.

To address this, SPOTNIK provides high-level synchronisation primitives implemented on top of the SPOTNIK communication layer. Users can combine these primitives to implement various synchronisation strategies. Since these strategies are implemented as part of a unified communication layer, we can thus efficiently switch between strategies during training.

SPOTNIK supports a range of synchronisation primitives, which can be broadly classified into two groups: (i) collective synchronisation operators (e.g., global gradient sum and model averaging), and (ii) point-to-point synchronisation operators (e.g., pair-wise model averaging and model caching). These operators are general enough to implement many synchronisation strategies, including S-SGD, SMA and

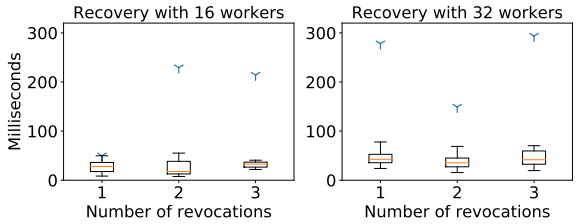

Figure 4: Recovery latency with concurrent revocations

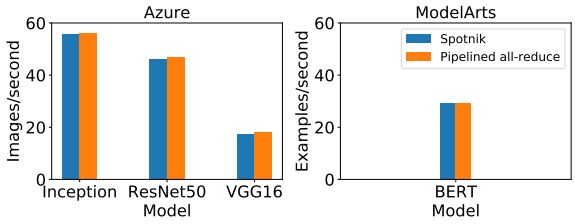

Figure 5: Revocation overhead during all-reduce

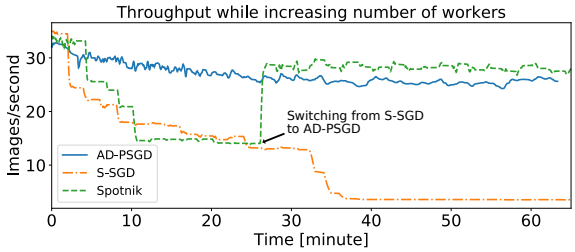

Figure 6: Effect of adaptive synchronisation

AD-PSGD. We are currently implementing more sophisticated strategies, such as the recent Lookahead strategy [53].

4 Evaluation

Our evaluation asks: (i) what is the recovery latency after a VM revocation? (ii) what is the overhead of supporting VM revocations during all-reduce? (iii) what are the benefits of adaptive synchronisation in a real-world training scenario?

We run experiments on Azure with 32 NC-6 VMs. Each VM has 6 vCPUs, 56 GB of memory and a NVIDIA K80 GPU. Additional experiments use the Huawei ModelArts Cloud [31] with two nodes that have 8 V100 GPUs each, linked via 100 Gbps InfiniBand. We implement SPOTNIK on top of KungFu, a distributed training library [24, 30], which scales out TensorFlow programs and achieves performance similar to Horovod. We use KungFu because of its ability to add and remove workers during training. We run SPOTNIK with KungFu 0.2.1 and TensorFlow 1.15.0. As models, we use ResNet-50 [18], VGG-16 [46], and Inception-V3 [48], designed for the ImageNet task, and BERT-base [12], designed for the SQuAD question-and-answering task [39].

Recovery latency. First we explore the latency after revocation. There are two factors that affect latency: the total number of workers, which determines the number of control messages, and the number of concurrently revoked VMs, which impacts the cost of the recovery operation. We therefore consider different numbers of workers (16 and 32) training a ResNet-50 model for ImageNet. During training, we revoke 1–3 workers concurrently and measure the recovery latency.

Fig. 4 shows the recovery latency of workers, as we increase the number of concurrent revocations. The whisker plot indicates the 25th, median and 75th latency percentiles, respectively; the markers show maxima. For both 16 and 32 workers, the maximum latency remains below 300 ms due to SPOTNIK's fast adaptation. This allows SPOTNIK to work effectively even in highly dynamic transient clusters.

Performance overhead. Next we measure the performance overhead when enabling revocation recovery during all-reduce operations. We compare four representative deep learning models (VGG-16, ResNet-50, Inception-V3, BERT-base), which cover a large spectrum of model sizes used in practice and have different all-reduce workloads. BERT-base runs on ModelArts; the others run on Azure.

Fig. 5 shows the difference between the atomic (SPOTNIK) and non-atomic (pipelined all-reduce operations) updates of gradients. The atomic updates have only a small impact on throughput. This reflects the benefits of using control messages for handling revocation inside all-reduce operations. The difference becomes negligible for the BERT-base model due to the higher bandwidth network on ModelArts.

Adaptive synchronisation. Now we investigate how SPOTNIK handles dynamic cluster changes. We activate the (accuracy-friendly) S-SGD and (network-friendly) AD-PSGD strategies in SPOTNIK's optimiser. We begin with one worker training ResNet-50, adding a new worker every 2 mins.

Fig. 6 shows how the per-worker throughput changes with more workers. For S-SGD, it decreases because the communication overhead of an all-reduce grows substantially with the number of workers; for AD-PSGD, it roughly stays constant because communication is asynchronous and it occurs only between pairs of workers. SPOTNIK switches from S-SGD to AD-PSGD after 26 mins of training time when the throughput falls below 40%. It therefore removes the communication bottleneck and maintains high throughput. We speculate that SPOTNIK is faster than AD-PSGD due to the performance variability of cloud resources.

5 Conclusions

SPOTNIK enables efficient distributed machine learning on transient cloud resources through adaptive collective communication and synchronisation. Its adaptive collective communication recovers from VM revocations by reusing the existing communication as part of the all-reduce ring. It also updates worker state atomically to ensure that gradient updates remain consistent during recovery. To account for the dynamic nature of transient clusters, SPOTNIK switches among different synchronisation strategies during training depending on network bottlenecks due to different cluster sizes.

6 Discussion

We consider SPOTNIK an initial design for a distributed ML system specifically created to work effectively in cloud environments with transient VMs. Beyond adaptive collective communication and synchronisation, there are, however, other system aspects that are affected by transient resources. For example, hyper-parameter tuning [29], which is important for current ML models to achieve high accuracy and performance, must be rethought when the cloud execution environment itself is highly dynamic. Here, the systems community would benefit from feedback by ML practitioners on other aspects of ML systems that are affected by transient resources.

In addition, there are interesting design questions for cloud applications when finer-grained resources, beyond individual VMs, become transient. In the future, cloud providers may decide to auction off memory and network bandwidth as transient resources. In such fully disaggregated cloud environments [13], we could imagine that the mix of available transient resources changes over time. Future distributed ML systems must therefore be designed to work in such dynamic disaggregated cloud environments, e.g., by being able to shift quickly between different resource types and accounting for the associated trade-offs. As part of the discussion, we would ask for input by cloud providers whether the above is how they believe future cloud environments will evolve.

With the highly-disaggregated dynamic clouds of the future, the execution model may resemble today's *serverless* offerings, such as AWS Lambda [2] and Azure Cloud Functions [4]. Early work exists that explores how distributed ML jobs can be supported effectively in serverless clouds [8]. The techniques to split the ML training computation into many short-running, potentially stateless, functions is applicable to transient resources. Especially when transient resources are allocated and revoked at a fine granularity, with short revocations times on the order of seconds, the models of transient cloud resources and serverless computing begin to unify.

All of this will require cloud pricing models that are intuitive and predictable for users. Users are often charged for resources in fixed-sized allocation blocks. We believe that future clouds will support more fine-grained resource accounting, e.g., at the level of CPU cycles [9]. With the rise of ML accelerators such as GPUs [34], TPUs [51], Intel NNP [21] and IPUs [16], simple pricing models that clearly capture trade-offs between cost and performance of technologies will be necessary for adoption by the ML community. We are interested in hearing about innovative cloud pricing models that will impact the design of next-generation ML systems.

Nore that SPOTNIK can only handle revocations of a part of the cluster. Since no node is left when the whole cluster is revoked, SPOTNIK cannot entirely avoid checkpointing. We want to consider options for supporting this scenario better. To decrease the probability of losing all transient VMs, users may adopt bidding schemes such as heterogenous bids [55].

References

[1] Amazon EC2 Spot Instances. https://aws.amazon.com/ec2/spot/, 2020.

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

[3] Announcing Amazon EC2 Spot Instances. https://aws.amazon.com/about-aws/whats-new/2009/12/14/announcing-amazon-ec2-spot-instances/, 2009.

[4] Azure Functions. https://azure.microsoft.com/en-us/services/functions/, 2020.

[5] Announcing the preview of Azure Spot Virtual Machines. https://azure.microsoft.com/en-gb/blog/announcing-the-preview-of-azure-spot-virtual-machines/, 2019.

[6] Azure Spot Virtual Machines. https://azure.microsoft.com/en-us/pricing/spot/, 2020.

[7] Peva Blanchard, Rachid Guerraoui, Julien Stainer, et al. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[8] Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. Cirrus: a Serverless Framework for End-to-end ML Workflows. In *Proceedings of Symposium on Cloud Computing (SOCC)*, 2019.

[9] Chen Chen, Petros Maniatis, Adrian Perrig, Amit Vasudevan, and Vyas Sekar. Towards verifiable resource accounting for outsourced computation. In *Conference on Virtual Execution Environments (VEE)*, 2013.

[10] James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Gregory R Ganger, Garth Gibson, Kimberly Keeton, and Eric Xing. Solving the straggler problem with bounded staleness. In *Workshop on Hot Topics in Operating Systems (HotOS)*, 2013.

[11] Jeff Dean, David Patterson, and Cliff Young. A new golden age in computer architecture: Empowering the machine-learning revolution. *IEEE Micro*, 2018.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[13] Peter X Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Network requirements for resource disaggregation. In *Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[14] Gareth George, Rich Wolski, Chandra Krintz, and John Brevik. Analyzing AWS spot instance pricing. In *International Conference on Cloud Engineering (IC2E)*. IEEE, 2019.

[15] Google Preemptible Virtual Machines. https://cloud.google.com/preemptible-vms/, 2020.

[16] GraphCore IPU. https://www.graphcore.ai/products/ipu, 2020.

[17] Aaron Harlap, Alexey Tumanov, Andrew Chung, Gregory R Ganger, and Phillip B Gibbons. Proteus: agile ML elasticity through tiered reliability in dynamic resource markets. In *Proceedings of European Conference on Computer Systems (EuroSys)*, 2017.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of Computer Vision and Pattern Recognition (CVRP)*, 2016.

[19] IBM Cloud Functions. https://cloud.ibm.com/functions/, 2020.

[20] IBM Transient Virtual Servers. https://www.ibm.com/cloud/blog/transient-virtual-servers?mhsrc=ibmsearch_a&mhq=transient, 2020.

[21] Intel Neural Network Processors. https://www.intel.ai/nervana-nnp, 2020.

[22] Alexandros Koliousis, Pijika Watcharapichat, Matthias Weidlich, Luo Mai, Paolo Costa, and Peter Pietzuch. CROSSBOW: scaling deep learning with small batch sizes on multi-GPU servers. *Proceedings of Very Large Data Bases (VLDB)*, 2019.

[23] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[24] KungFu. https://github.com/lsds/KungFu, 2020.

[25] Kyungyong Lee and Myungjun Son. Deepspotcloud: leveraging cross-region GPU spot instances for deep learning. In *International Conference on Cloud Computing (CLOUD)*. IEEE, 2017.

[26] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[27] Shijian Li, Robert J Walls, Lijie Xu, and Tian Guo. Speeding up deep learning with transient servers. In *IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2019.

[28] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. *arXiv preprint arXiv:1710.06952*, 2017.

[29] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint arXiv:1807.05118*, 2018.

[30] Luo Mai, Alexandros Koliousis, Guo Li, Andrei-Octavian Brabete, and Peter Pietzuch. Taming Hyperparameters in Deep Learning Systems. *SIGOPS Operating Systems Review*, 2019.

[31] ModelArts. https://www.huaweicloud.com/en-us/product/modelarts.html, 2020.

[32] NVIDIA Collective Communications Library. https://developer.nvidia.com/nccl, 2020.

[33] Jose Pergentino A Neto, Donald M Pianto, and Célia Ghedini Ralha. A prediction approach to define checkpoint intervals in spot instances. In *International Conference on Cloud Computing (CLOUD)*. Springer, 2018.

[34] Nvidia Tesla. https://www.nvidia.com/en-gb/data-center/tesla/, 2020.

[35] High Performance Message Passing Library. https://www.open-mpi.org, 2020.

[36] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of European Conference on Computer Systems (EuroSys)*, 2018.

[37] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. A generic communication scheduler for distributed DNN training acceleration. In *Proceedings of Symposium on Operating Systems Principles (SOSP)*, 2019.

[38] PyTorch. https://pytorch.org, 2020.

[39] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[40] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[41] Resource elasticity in distributed deep learning. In *Proceedings of Machine Learning and Systems (MLSys)*. 2020.

[42] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, 1951.

[43] Aleaxander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

[44] Supreeth Shastri and David Irwin. HotSpot: automated server hopping in cloud spot markets. In *Proceedings of Symposium on Cloud Computing (SOCC)*, 2017.

[45] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[46] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[47] Spot Instance Advisor. https://aws.amazon.com/ec2/spot/instance-advisor/, 2020.

[48] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of Computer Vision and Pattern Recognition (CVRP)*, 2016.

[49] TensorFlow. https://www.tensorflow.org, 2020.

[50] Distributed training with TensorFlow. https://www.tensorflow.org/guide/distributed_training, 2020.

[51] Google Cloud TPU. https://cloud.google.com/tpu/, 2020.

[52] Pijika Watcharapichat, Victoria Lopez Morales, Raul Castro Fernandez, and Peter Pietzuch. Ako: Decentralised deep learning with partial gradient exchange. In *Proceedings of Symposium on Cloud Computing (SOCC)*, 2016.

[53] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead Optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[54] Sixin Zhang, Anna E Choromanska, and Yann LeCun. Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[55] Xiaoxi Zhang, Jianyu Wang, Gauri Joshi, and Carlee Joe-Wong. Machine learning on volatile instances. *arXiv preprint arXiv:2003.05649*, 2020.