M.S. THESIS

Cost-Efficient Machine Learning Training on Preemptible Cloud Clusters

선점가능형 클라우드 클러스터에서의 비용 효율적인 머신러닝 학습

2022 년 8 월

서울대학교 대학원

컴퓨터 공학부

구 윤 모

M.S. THESIS

Cost-Efficient Machine Learning Training on Preemptible Cloud Clusters

선점가능형 클라우드 클러스터에서의 비용 효율적인 머신러닝 학습

2022 년 8 월

서울대학교 대학원

컴퓨터 공학부

구 윤 모

Cost-Efficient Machine Learning Training on Preemptible Cloud Clusters

선점가능형 클라우드 클러스터에서의 비용 효율적인 머신러닝 학습

지도교수 전 병 곤

이 논문을 공학석사학위논문으로 제출함

2022 년 6 월

서울대학교 대학원

컴퓨터 공학부

구 윤 모

구윤모의 공학석사 학위논문을 인준함

2022 년 7 월

위 원 장	염 헌 영	(인)
부위원장	전 병 곤	(인)
위 원	장 병 탁	(인)

Abstract

Due to the high cost of building a physical GPU cluster infrastructure for AI model training, the demand for training on "pay-as-you-go" public cloud clusters has increased rapidly. In particular, training AI models using preemptible(*i.e.*, spot) VMs provided at steep price discounts has attracted the attention of many researchers. However, since cloud providers can unilaterally revoke preemptible VMs at any time, it may result in the loss of underway training states. Due to the trade-off between cost and reliability, researchers are disinclined to actively adopt preemptible VMs for their experiments. In this paper, we discuss the major challenges of AI model training on preemptible VMs and propose *Spotify*, an AI model training job orchestrator, which automatically deals with the challenges and enables reliable training on preemptible cloud clusters. Researchers can run training jobs on low-price preemptible clusters under the illusion of using reliable on-demand clusters. Our evaluations show that *Spotify* reduces the 62% of end-to-end training cost with only sacrificing 2.86% additional latency overhead compared to the training on on-demand clusters.

Keywords: Machine Learning, Cloud, Preemption Handling

Student Number: 2020-27792

Contents

Chapter 1

Introduction

As the size of deep learning models has increased rapidly, GPU becomes the necessary resource for the compute-intensive model training job. Since the speed of model training depends on the performance and the number of GPUs, AI researchers need to build well-configured GPU clusters for their model development. However, the cost of building and maintaining physical clusters is very high, and next-generation GPUs that outperform previous models come to the market every year.

For such reasons, rather than managing expensive physical GPU clusters, many researchers shift to public cloud services that provide unlimited VM resource capacity with various types of GPU and pay only the amount of money they spent during leasing the VMs. More interestingly, many cloud providers offer *preemptible* VMs (*e.g.*, Amazon spot instances, Azure spot VMs and Google Cloud spot VM) at a 60∼90% discount [1, 2, 3], but with the risk of resource revocation.

Despite the advantage of price, the unreliability due to unsuspected preemp-

tion makes it hard to use preemptible VM for AI model training. Simply using the preemptible VMs for model training without a proper interruption handling mechanism may result in a loss of training states. Checkpointing model parameters is a commonly-used fault tolerance technique to reduce the effect of preemption. However, too frequent checkpointing brings longer training time since checkpointing makes additional overhead to serialize and write the data to disk. Therefore, it is important to find a sweet spot for checkpointing intervals to minimize the overall training time, but it is difficult for developers to configure a suitable checkpointing interval for each training job by hand. Furthermore, manually reallocating VMs per every preemption time is also time-consuming and cumbersome.

To address the problems, three principal challenges need to be addressed: (1) automatizing the entire training process on a preemptible VM, (2) minimizing the end-to-end execution time of training jobs, and (3) supporting arbitrary training jobs in a non-invasive manner. In this paper, we propose *Spotify*, a system that solves these challenges to enable reliable AI model training on preemptible VMs.

First, when users submit training jobs, *Spotify* automatically allocates cloud VMs according to the request configuration. When the preemption occurs, *Spotify* detects the event via pre-delivered preemption notice and executes appropriate handling routines to minimize the impact of the interruption. At the same time, *Spotify* reallocates new VMs as quickly as possible to seamlessly migrate the ongoing training job.

Second, *Spotify* configures the optimum checkpointing policy to minimize the end-to-end training time. The system adaptively adjusts checkpointing intervals during the job runtime to reflect the inconsistent preemption probability. Exploiting the on-the-fly profiling results from the running training job and

system-widely collected preemption-related metrics, *Spotify* finds and applies the optimal checkpointing strategy to minimize the training time.

Third, users of *Spotify* can easily train their AI models on preemptible VMs by making a few lines of code addition. We design user APIs which are generally applicable for model training scripts with minimal code changes.

This paper makes the following primary contributions.

- We propose *Spotify*, an AI model training job orchestrator that automates cloud resource management and checkpoint migration to enable reliable training for arbitrary training workloads on low-price preemptible VMs.
- We propose an adaptive checkpointing policy that takes into account both real-time job metrics and pre-delivered preemption notice. By applying the auto-configured checkpointing policy, we enable the spot training at 38% of the price of the on-demand training with sacrificing only 2.86% of latency overhead.
- We analyze the trade-off between checkpointing interval and rework time caused by preemption and evaluate the efficiency of adaptive checkpointing policy using real-world applications.

Chapter 2

Background

2.1 Preemptible Virtual Machines

Cloud providers offer "on-demand" virtual machines, which are charged at a price per second. Customers have complete control over the on-demand VM lifecycle; they can start, stop, reboot, or terminate VMs when needed. In order to stably satisfy the demand for VMs, cloud providers typically maintain resource capacity on a peak request rate basis, so the VMs usually tend to be underutilized.

To leverage the spare capacity, cloud vendors offer preemptible VMs from a spare resource pool at a significantly lower price (60∼90%) than on-demand VMs of the same type. Unlike on-demand VMs, if cloud providers cannot meet on-demand VM requests from the spare resource pool, preemptable VMs can be interrupted and the capacity returns to the provider at any time. For example, the most popular cloud computing services such as Amazon EC2, Azure Virtual Machines, and Google Cloud Compute Engine commonly offer "spot" VMs.

Although each cloud provider has a different preemption policy in detail, preemption occurs based on the following conditions in general. First, VMs can be evicted when cloud providers need the capacity back to handle on-demand requests. The capacity is different by the type of VM and availability region, thus how frequently VMs get evicted depends on them. Second, when the market price for the VM exceeds the *max price* (the bid price the customer is willing to pay to acquire the VM), the VM gets evicted [4, 5]. Since the max price is an important factor not only for the cost but also for the probability of preemption and resource acquirement, efficient bidding strategies have been studied [6, 7, 8]. However, the concept of bidding exists only for certain cloud providers (*e.g.*, Amazon EC2 and Azure VM), and several cloud providers, such as Google Cloud, provide preemptible VMs without bidding. In addition, the high bidding price does not prevent the preemption caused by capacity issues.

To alleviate the effect of preemption, cloud providers give prior notification for the scheduled preemption. For example, Azure Metadata Service offers an endpoint to poll scheduled events which comes 30-second before the preemption scheduling [9]. In the same manner, AWS and Google Cloud also give a pre-delivered preemption notice, which is delivered at least 1 hour and 2 minutes in advance respectively [10, 11]. For this grace period, user applications can prepare for the upcoming interruption by storing stateful data or gracefully shutting down running programs.

2.2 Model Training and Checkpointing

One of the widely-used fault tolerance techniques to mitigate the effect of resource interruption is *checkpoint-and-restart*, which writes the intermediate states of a process from memory to disk and restarts the process from the

saved states. There are many pieces of research modeling the optimal checkpoint strategy [12, 13, 14, 15, 16, 17] in the context of HPC workloads. In the context of ML model training, checkpointing is not only used for the purpose of fault-tolerance for resource failure like VM preemption, and it is also used for the purpose of transferring the learned model states to another training job such as fine-tuning, or inference.

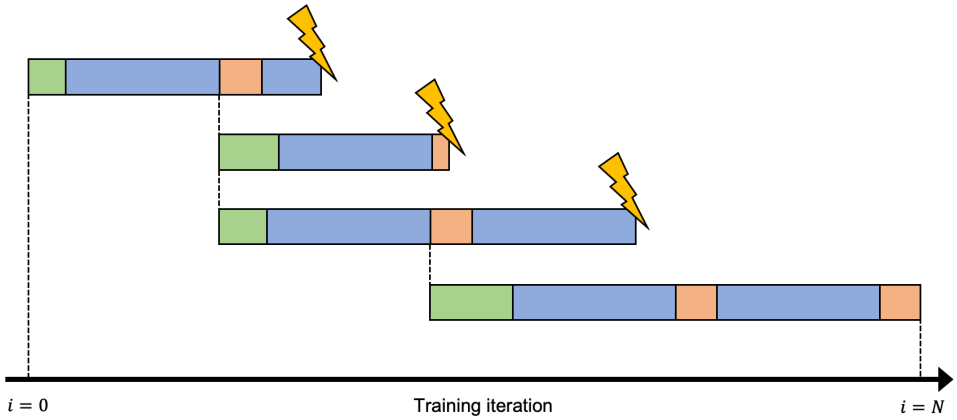

Figure 2.1 Training process on a preemptible VM. The green, blue, and orange color in the box represents preparation time, iteration time, and checkpoint time respectively. The yellow lightning symbols denote preemption events.

Figure 2.1 visualizes the normal AI model training process from iteration $i = 0$ to $i = N$ using preemptible VMs. The training process starts with preparation time (green bar), which includes the latency of VM allocation, initialization, and training dataset loading. After the preparation stage, training iterations (blue bar) start. Per certain iteration periods, intermediate model parameters residing in memory space are serialized and dumped to the disk (orange bar). In the middle of the training process, preemption (yellow lightning symbol)

can occur, and the training halts until a new VM is allocated to continue the training. After the VM reallocation, the training process can be resumed from the latest checkpoint.

Since the iterations after the last checkpointing are lost and the wasted iterations should be recomputed, model checkpoints should be created as close as possible to preemption events. The most naive and simple way to reduce the amount of wasted work is frequent checkpointing. However, as each synchronous checkpointing entails overhead of serialization and disk write, excessive checkpointing increases the entire training time. Therefore, it is important to figure out the optimal checkpointing point on a sweet spot that reduces the entire training time.

Chapter 3

Challenges

3.1 Unpredictability of Preemptions

Cloud providers unilaterally evict running preemptible VMs when they need the capacity back. Figure 3.1 shows the real-world preemption trace of the Azure Standard_NC24s_v3 spot instance with 4 V100 GPUs. The preemption rate is not consistent during the day and there are peak times when the demand for VM increases rapidly. A similar preemption pattern is also reported by many existing works studying cloud VM preemptions regardless of cloud providers [18, 19, 14]. According to those researches, the preemption probability is not only affected by the time of a day, but also affected by other various factors such as the geographical region, days of the week, and instance size. For some cases, even the running workload characteristics and VM lifetime are also related to the preemption frequency [14].

As explained in Section 2.2, it is important to consider the preemption pattern to checkpointing policy to reduce the latency overhead which comes from

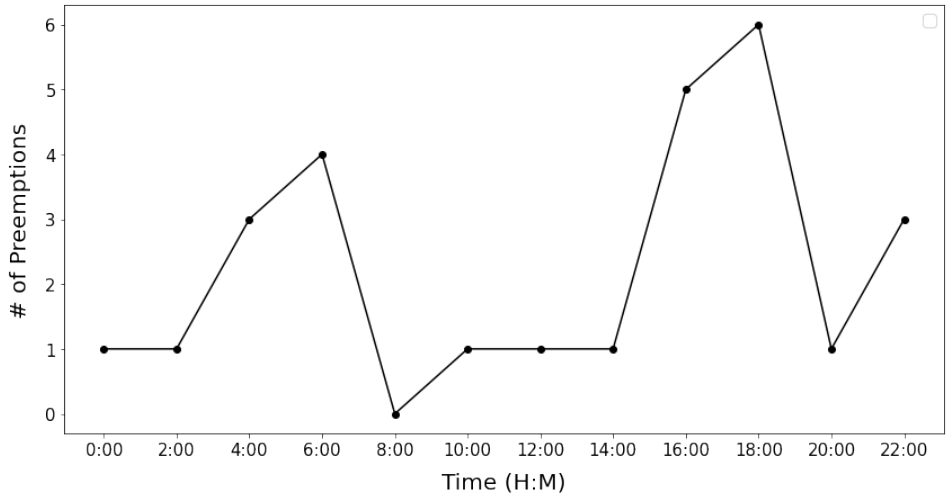

Figure 3.1 24-hour real-world preemption trace collected from the 6 Azure Standard_NC24s_v3 VM instances. The time zone is the East US region (Virginia).

both file-write and recomputing the wasted works—excessive checkpointing increases the entire job latency due to the high file write overhead and checkpointing too sparsely also slows down the progress of training due to the high recomputation overhead. Unfortunately, the pattern of preemption is affected by so many factors in the real world and changes from moment to moment, thus it is challenging to reflect the cloud VM availability when determining the checkpointing policy.

This challenge motivates our system to monitor and manage the metrics required to determine the optimum checkpointing strategy. *Spotify* collects the runtime training information and preemption-related metrics such as the VM allocation time and the average preemption interval and determines the checkpointing interval based on them.

3.2 Resource Management

Since preemptible VMs can be revoked at any time, it is bothersome for AI model developers to reallocate new VMs by themselves. According to our real-world preemption trace from Azure Standard_NC12s_v3 instance equipped with 2 V100 GPUs, the preemption occurs 3.95 times per 12 hours on average of 20 samples. That is, without automatizing the preemption recovery process, users have to restart the stopped VMs by hand every 3 hours. In addition, as it is impossible to exactly predict the preemption time and preemptions usually occur in irregular intervals, users need to keep monitoring the VM status until the end of training.

To resolve this issue, *Spotify* automatizes the preemption detection and recovery process—it detects the preemption with the pre-delivered preemption notice and keepalive heartbeat signals, and reallocates a new VM to seamlessly continue the training job. The training can be resumed on any VM because the model checkpoint files generated before the preemption are uploaded to remote cloud storage. When reallocation is rejected by cloud providers due to the low availability, the system continuously retries the allocation.

Chapter 4

Modeling Checkpointing Policy

4.1 Approximating Optimal Checkpointing Interval

There are some studies trying to quantify the optimal checkpointing interval that minimizes the application wall time latency on a system with the possibility of failure [20, 21, 17, 14]. One of the most simple and general checkpointing model is proposed by Daly [21], which calculates the optimal chechpointing time interval as $\sqrt{2 \cdot \delta \cdot (MTTP + R)}$, where δ, $MTTP$, and R denotes checkpoint dump time, mean time to preemption, and restart time respectively. This checkpointing policy assumes that the occurrence of a preemption event follows a Poisson distribution. In this section, we explain the details of this approximation briefly and propose a way to apply the approximation systematically.

The entire job latency for AI model training on interruptable resources consists of process time, checkpoint time, restart time, and rework time as described in Figure 2.1, and it can be formulated as the following objective function $T(\tau)$.

$$T(\tau) = T_p + \frac{T_p}{\tau} \cdot \delta + \phi \cdot (\tau + \delta) \cdot p(\tau) + R \cdot p(\tau) \tag{4.1}$$

The entire process time that includes computation and communication time for model training is defined as T_p. Then, the total latency of checkpointing is $\frac{T_p}{\tau} \cdot \delta$, where τ is the time interval between each checkpointing, and δ is the time for model parameter serialization and writing the data to a disk. The total amount of rework time is $\phi \cdot (\tau + \delta) \cdot p(\tau)$, which is the product of $\tau + \delta$ (process time for a *segment*), ϕ (progress rate of the *segment*), and $p(\tau)$ (the number of preemptions). Note that a *segment* consists of the computation for several iterations between two checkpointing points and the following checkpointing for the computed iterations. The total amount of restart time after each preemption, which includes time for VM reallocation, process initialization, and data loading, is $R \cdot p(\tau)$, where R is the one-time restart latency.

When we assume that the preemptions occur according to a Poisson distribution whose probability density function is $f(t) = \frac{1}{\lambda} e^{-t/\lambda}$, then the expected number of preemption $p(\tau)$ is approximated as follows:

$$p(\tau) \approx \frac{T_p}{\tau} \cdot (\frac{\tau + \delta}{\lambda}) \tag{4.2}$$

According to Eq. 4.2, the objective function in Eq. 4.1 can be rewritten as:

$$T(\tau) = T_p + \frac{T_p}{\tau} \cdot \delta + [\phi \cdot (\tau + \delta) + R] \cdot \frac{T_p}{\tau} \cdot (\frac{\tau + \delta}{\lambda}) \tag{4.3}$$

If we assume that preemption normally occurs in the middle of a segment, then ϕ can be set to $\frac{1}{2}$. The scale parameter λ is parameterized with the value of $MTTP$.

$$T(\tau) = T_p + \frac{T_p}{\tau} \cdot \delta + \left[\frac{1}{2} \cdot (\tau + \delta) + R\right] \cdot \frac{T_p}{\tau} \cdot (\frac{\tau + \delta}{MTTP}) \tag{4.4}$$

Finally, by differentiating Eq. 4.4 for τ, we can find the value of τ that minimizes the end-to-end job latency $T(\tau)$ as:

$$\tau_{opt} = \sqrt{2 \cdot \delta \cdot (MTTP + R)} \tag{4.5}$$

Although the policy suggested by Daly [21] is simple and easy to compute, it is hard to set the value of $MTTP$ and R properly in real-world applications, and the value of δ can be known after running the job. Furthermore, since AI model training usually takes from a few days to months, the proper value of $MTTP$ and R can be changed during the training job. To solve this problem, *Spotify* collects the wall time latency of VM lifetime (*i.e.*, the period between VM start time and the preemption time) and VM allocation time and uses the information to determine the next checkpointing interval.

4.2 Emergency Save

Cloud providers issue VM interruption notice before they terminate running VMs. How early they deliver the notice is different by vendors. Amazon EC2, Azure VM, and Google Cloud Compute Engine give the preemption warning 2 minutes, 30 seconds, and 1 hour before the actual event scheduling respectively [10, 9, 11]. The preemption notice can be polled by sending an HTTP request to a REST endpoint from running preemption VMs managed by cloud VM services.

Spotify continuously polls the preemption notice and check whether there is enough time for a *emergency save*. The *emergency save* writes checkpoint files to the local file system and transfers the files to the remote cloud storage (*e.g.*, Amazon S3, Azure Blob Storage, and Google Cloud storage) as soon as the ongoing iteration is computed. By leveraging preemption notice for the

fault-tolerance mechanism, the rework time becomes zero since the job can be resumed from the latest step before the preemption. When the *emergency save* is available, the only wasted time due to preemption is VM reallocation time and training preparation time such as data loading.

4.3 Insurance Save

The best handling routine for preemption is the *emergency save* as it makes zero overhead for the rework time and intermediate checkpointing. However, since the *grace period* before the preemption is limited, it is not an always-available solution. If there is not enough time to store and back up checkpoints after receiving preemption notice, checkpoints should be saved in the middle of the training job to minimize the rework time after preemption. We call this intermediate checkpointing as an *insurance save* as it is the insurance for the catastrophic state loss. It is important to determine the appropriate checkpoint interval because too frequent checkpointing is rather disadvantageous.

Spotify uses the simple checkpointing policy described in Section 4.1, which is proposed by Daly [21], and the value of the next checkpointing iteration is parameterized based on the collected preemption-related metrics such as average VM lifespan, average checkpointing time, and average VM allocation latency. *Spotify* collects every preemption event time and allocation time for every supported cloud VM type from all training jobs submitted to the system. That is, the values of $MTTP$ and R are not fixed during the execution of a job and are changed by the cloud VM availability. The more jobs are submitted to *Spotify*, the more likely it is to determine better checkpoint iterations that minimize end-to-end job latency.

4.4 Adaptive Checkpointing

To reduce the job execution time, *Spotify* exploits both *emergency save* and *insurance save* together for model checkpointing. Algorithm 1 describes how the *adaptive checkpointing* of *Spotify* works. For each training iteration i among the entire iterations I, the step process time T_i is measured and the average step time T_{step} is updated (See (2) in Algorithm 1). When the current step i reaches one of the periodic checkpointing step I_{save}, which is given by a user who submits the job, the trained model states are dumped into a VM local disk. Then, the checkpoint files in the local file system are uploaded to remote cloud storage (*e.g.*, S3, Azure Blob, Google Cloud Storage) to make them persistent and accessible on any other VM. This backup routine is executed in the background, so it does not block the training process. When the checkpointing is done, the next *insurance save* step is scheduled to prepare in advance for the preemption that may occur at any time (See (3) in Algorithm 1). The optimal checkpointing time interval τ_{opt} is calculated by $\sqrt{2 \cdot \delta \cdot (MTTP + R)}$ as explained in Section 4.3. According to τ_{opt}, the next *insurance save* step number i_{ins} is determined as $i + \frac{\tau}{T_{step}}$, where i denotes the current step number (See (1) in Algorithm 1). Note that if the periodic checkpointing is scheduled in a shorter period of interval than i_{ins}, the *insurance save* does not occur.

When the preemption notice containing the preemption scheduling time arrives, the remaining time to preemption T_{grace} is determined. How early the notice arrives in advance of the actual preemption scheduling varies to cloud providers as described in Section 4.2. If there is enough time for *emergency save*, which computes the ongoing training step and save-then-backup checkpoint files, in the time range of T_{grace}, *Spotify* can capture the latest training states before the revocation and resume the training from the checkpoint without a

waste of time to recompute the lost states (See (4) in Algorithm 1). Otherwise, *Spotify* does *insurance save* at the i_{ins}-th step to minimize the rework time due to the preemption (See (5) in Algorithm 1).

Algorithm 1: Adaptive checkpointing policy

Data: $\delta, \beta, MTTP, R, I, I_{save}, T_{grace}$

function `calcInsuranceSaveStep`$(\delta, MTTP, R, T_{step}, i)$**:**

  $\tau_{opt} \longleftarrow \sqrt{2 \cdot \delta \cdot (MTTP + R)}$

  $i_{ins} \longleftarrow i + \frac{\tau_{opt}}{T_{step}}$ `# (1) Next insurance save step`

  return i_{ins}

end

foreach $i \in I$ **do**

  $T_{step} \longleftarrow \frac{\sum_{k=1}^{i} T_k}{i}$ `# (2) Average step time`

  if $i \in I_{save}$ **then**

    `# (3) Periodic save step`

    do **SaveAndBackup**

    $i_{ins} \longleftarrow$ `calcInsuranceSaveStep`$(\delta, MTTP, R, T_{step}, i)$

  end

  if $T_{step} + \delta + \beta < T_{grace}$ **then**

    `# (4) Enough time for` *emergency save*

    if *preemption notice arrives* **then**

      do **EmergencySave**

    end

  else

    `# (5) Otherwise, do` *insurance save*

    if $i = i_{ins}$ **then**

      do **InsuranceSave**

      $i_{ins} \longleftarrow$ `calcInsuranceSaveStep`$(\delta, MTTP, R, T_{step}, i)$

    end

  end

end

Chapter 5

System Design

5.1 System Architecture and Workflow

In this section, we describe details of *Spotify* including its system architecture and overall workflow to enable preemption-tolerant training. Figure 5.1 depicts the system architecture and workflow of *Spotify*. Users who want to train AI models with *Spotify* submit jobs to *Spotify Master*. The job is defined in YAML format, which includes configurations for model training source code, dataset, input checkpoint, package dependencies, and VM specifications. When *Spotify Master* receives a job, it tries to allocate VMs according to the job configuration. The wall time latency for VM allocation is recorded to a database and the collected data is used to calculate *insurance save* step numbers.

After a cloud VM is allocated, *Remote Launcher* launches the process for model training and *Preemption Manager* on the remote cloud VM. The training process is executed in an environment described in the job configuration. While training the model, the training process sends latency metrics to *Pre-*

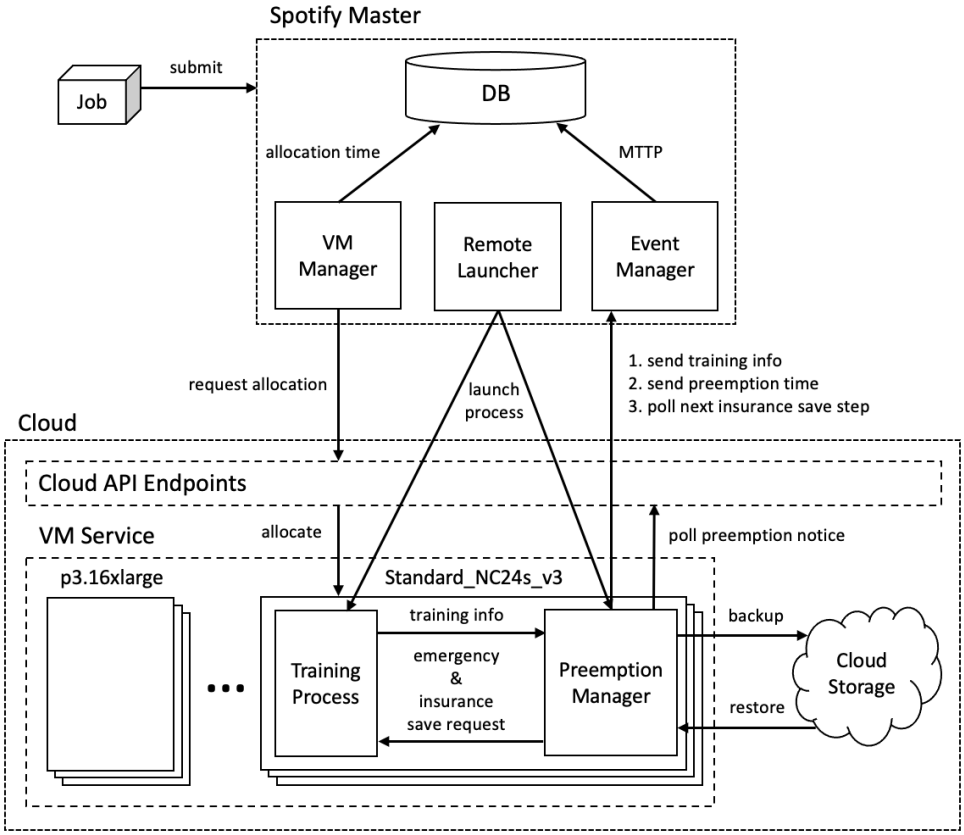

Figure 5.1 *Spotify* system architecture and workflow

emption Manager through named pipes between them. The metrics include latency of initialization, data loading, checkpointing, and iteration computation. When *Preemption Manager* receives the metrics, it delivers the metrics to *Event Manager* in *Spotify Master*. The metrics are used to determine when to generate checkpoints based on *adaptive checkpointing* policy described in Section 4.4. The next *insurance save* iteration number is calculated at *Event Manager* according to the training info, allocation time and $MTTP$ stored in

the database, and *Preemption Manager* polls the calculated iteration number from *Event Manager*. When the training process reaches the iteration that is equal to the scheduled *insurance save* iteration, *Preemption Manager* sends a *insurance save* request, which enforces immediate checkpointing, to the training process. If the profiled metrics satisfy the availability condition of *emergency save*, $T_{step} + \delta + \beta < T_{grace}$, then the *insurance save* is not executed.

Preemption Manager continuously polls a preemption notice once per second. When the preemption notice arrives, *Preemption Manager* first checks whether emergency save is available before the scheduled preemption occurs. If it is possible to save-then-backup checkpoint files during the grace period, *Preemption Manager* sends an *emergency save* request, which orders immediate checkpoint saving, to the training process through the connected named PIPE channel. At the same time, *Preemption Manager* also sends a message that notifies preemption scheduling to *Event Manager*. Then *Event Manager* tries to reallocate a new replacement VM in advance of actual preemption scheduling, and it updates the average VM lifespan, which corresponds to $MTTP$, to the database. Note that the database manages $MTTP$ per combination of VM type and region.

All checkpoint files, which are generated from *insurance save*, *emergency save*, and user-provided periodic save, are uploaded to cloud storage to make them downloadable from the other machines. *Preemption Manager* uploads the files to the closest cloud storage to the VM region to reduce upload latency. When the checkpoint files are successfully committed to the cloud storage, the files in the VM local disk are deleted to secure the free space on the disk. To save the storage cost, the checkpoint files created by *insurance save* and *emergency save* are deleted from the cloud storage when a new checkpoint is committed to the cloud storage.

Name	Description
`sp.enter_step()`	Mark the start of training step.
`sp.exit_step()`	Mark the end of training step.
`sp.step_context()`	Context manager wrapping a training step loop.
`sp.backup(path)`	Backup checkpoint to cloud storage in background.
`sp.check_save()`	Check insurance/emergency save request arrival.

Table 5.1 *Spotify* user APIs

Spotify Manager detects that the preemption actually occurs when *Event Manager* does not send a heartbeat signal anymore. At this point, *Remote Launcher* launches the process to the newly allocated VM again, and *Preemption Manager* restores the latest checkpoint from the cloud storage to continue training from the most recent states. After restoring the latest checkpoint, the model training is resumed, and the aforementioned workflow is repeated until the end of the job.

5.2 API Design

This section describes *Spotify* user APIs to enjoy the benefits of preemption-tolerant training. The APIs are designed under the principle to support an arbitrary training job with minimum code changes.

Table 5.1 enumerates a list of *Spotify* user APIs and their functionalities, and Listing 5.1 shows an example that applies the APIs to a simple training code. The highlighted lines in the Listing are the only required changes to use *Spotify*. First, there are APIs for marking the range of a training step; `sp.enter_step()` is called at the start of a training step, and `sp.exit_step()` is called at the end of the step. For ease of use, a context manager API, `sp.step_context()`, which wraps `sp.enter_step()` and `sp.exit_step()`, is also provided (line 5).

When checkpoint files are saved, users can upload the files to cloud storage by calling `sp.backup()` (line 13). To save and upload checkpoint files for either *emergency save* or *insurance save*, users can use `sp.check_save()` that returns true when the save request arrives (line 11). By adding only 4 lines of code, users can reliably train their AI models on preemptible VMs at a low cost.

```
1  import spotify as sp
2
3  for i, inputs in enumerate(data_loader):
4      ...
5      with sp.step_context():
6          output = model(inputs)
7          optimizer.zero_grad()
8          loss.backward()
9          optimizer.step()
10     ...
11     if i % save_interval == 0 or sp.check_save():
12         save_checkpoint()
13         sp.backup(checkpoint_path)
```

Listing 5.1 Example training script using *Spotify* APIs

Chapter 6

Evaluation

6.1 Environment

6.1.1 Cloud VM

For evaluation, we use Azure VM instances of Standard_NC12s_v3 type, which is $6.2/hour for an on-demand and $2.3/hour for a spot. The VM instance is equipped with two NVIDIA V100 GPUs with 16GB GPU memory, 12 vCPUs, and 240GB CPU memory. To simplify experiments, We only evaluate the latency and costs of single-node training jobs which use just one VM instance, and the VMs are allocated in the East US region. Although we limit the evaluation environment to a single Azure VM instance type, *Spotify* also supports other VM types from various cloud providers, and multi-node distributed training is available as well.

Spotify utilizes the pre-delivered preemption notice for *emergency save*. Azure delivers 30 seconds ahead preemption notice to eviction target VMs, and it is the shortest grace period length among the other popular cloud providers

such as AWS and Google Cloud. The longer the grace period, the more loaded jobs benefit from *emergency save*.

6.1.2 Job Specification

For all experiments, jobs submitted to *Spotify* pretrains a GPT [22] model with 1.2TB The Pile [23] dataset. To evaluate the efficiency of *adaptive checkpointing* by workload size, we use GPT-117M with 117 million parameters as a light workload and GPT-1.3B with 1.3 billion parameters as a heavy workload. We don't use many different models for experiments because the characteristics of the ML model training job are not different by model type. In general, ML model training starts with model initialization and data-loading, followed by multiple training loops and intermediate checkpointing.

6.2 Evaluation Tools

To evaluate the efficiency of *Spotify* in terms of cost and latency, we develop a preemption injector and offline job simulator. This section describes details of the evaluation tools and evaluates the accuracy of the simulator to ensure that the simulation results are trustworthy and close to real-world experiment results.

6.2.1 Preemption Injector

Since we cannot control the occurrence of preemption, we build a preemption injector by using interruption simulating tools officially provided by cloud providers [24, 25]. The injector gets (1) the name of the victim VM and (2) a list of $MTTP$ values as inputs and forces to interrupt the corresponding VM according to the preemption intervals, which are sampled from an exponential

Model	MTTP	Adaptive		Static	
		Processed Steps (12h)	Simulated Latency	Processed Steps (12h)	Simulated Latency
117M	1500	7310	12:00:51 (0:07:15)	6910	12:06:22 (0:06:30)
	2700	8390	12:07:16 (0:04:22)	8090	12:10:58 (0:05:18)
	5400	8910	12:05:11 (0:02:44)	8420	11:54:10 (0:04:56)
1.3B	1500	1070	12:06:08 (0:15:25)	1000	11:56:20 (0:13:41)
	2400	1400	12:00:57 (0:11:43)	1380	11:54:44 (0:11:00)
	4500	1860	12:12:39 (0:09:57)	1770	12:12:32 (0:09:10)

Table 6.1 Accuracy of Simulator. The values in parentheses of the simulated latency column are standard deviations. The simulated latency close to 12:00:00 corresponds to the accurate estimation.

distribution whose probability density function is Eq. 6.1 and parameterized by the given $MTTP$.

$$f(x; \frac{1}{MTTP}) = \frac{1}{MTTP} e^{-\frac{x}{MTTP}} \tag{6.1}$$

When the preemption is injected, the target VM is evicted after getting a preemption notice like the real preemption.

6.2.2 Training Simulator

To efficiently evaluate *Spotify* and its checkpointing policy, we develop an offline training simulation framework. The simulator takes (1) a list of $MTTP$ values, (2) total training iterations, (3) per-step computation latency, (4) checkpointing latency, (5) backup latency, (6) VM allocation latency, (7) preparation latency, (8) grace period, (9) VM type and (10) checkpointing policy as inputs, and calculates the estimated job latency and cost. Note that all latency values are

provided with a lower and upper bound, and the simulator samples the latency from a uniform distribution over the half-open interval every time.

Before using the simulator to evaluate *Spotify*, we first need to verify the estimates by the simulator are sufficiently accurate. Table 6.1 shows the processed iterations for 12 hours when preemptions are injected by the $MTTP$ values and simulated latency to process the same iterations with the same $MTTP$ values. We submit multiple jobs training GPT-117M and GPT-1.3B to *Spotify*and inject preemptions using the simulator introduced in Section 6.2.1. For both *adaptive checkpointing* and static period checkpointing, which saves the checkpoint for a fixed interval, the simulator calculates the latency with negligible errors.

6.3 Training Performance and Cost

In this section, we evaluate the efficiency of *Spotify* in two aspects: (1) how much latency can be saved by *adaptive checkpointing* of *Spotify*, and (2) how much cost can be saved by *Spotify* compared to the on-demand training and the spot training without *adaptive checkpointing* All experiment results in this section are reported with an average of 100 simulations.

6.3.1 Efficiency of Emergency Save

Figure 6.1 shows the additional latency overhead of training GPT-117M on a spot VM relative to the on-demand training latency, where the red bar is a recomputation latency, the green bar is a checkpointing latency, the blue bar is a VM allocation latency, and purple bar is a preparation latency including the time for checkpoint downloading, model initialization, and data-loading. We evaluate the latency of *adaptive checkpointing* and static checkpointing with

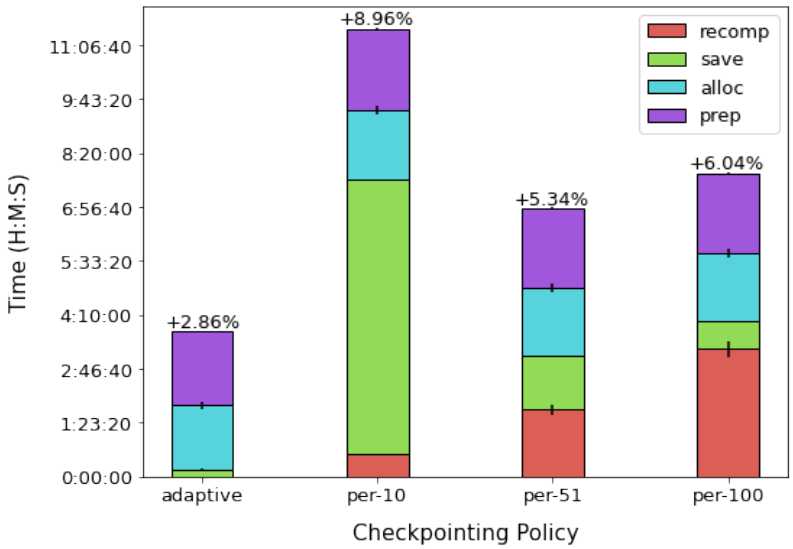

Figure 6.1 GPT-117M spot training latency overhead for 0.1M iterations. We use 10800 (3 hours) for the $MTTP$ value because the real preemption occurs 4 times per 12 hours on average according to the 20 real-world preemption traces.

the interval of 10, 51, and 100 steps. Note that 51 is the approximated optimal checkpointing interval calculated by Eq. 4.5.

Checkpointing per 51 steps makes 5.34% additional latency and is located at the sweet spot between frequent checkpointing (per-10 with 8.96% overhead) and sparse checkpointing (per-100 with 6.04% overhead). According to Table 6.2, which shows the latency breakdown for Figure 6.1, the inefficiency of the frequent checkpointing per 10 steps comes from the excessive file write overhead which occupies 5.08% of the entire running time. Although frequent checkpointing can save time for recomputation, the file-write overhead exceeds the advantage. In contrast, the sparse checkpointing per 100 steps makes a high recomputation overhead, which cannot be redeemed by the short checkpointing

	Adaptive	Per-10	Per-51	Per-100	On-demand (Per-10000)
Comp.	5d, 7:46:48 (97.15%)	5d, 8:21:00 (92.13%)	5d, 9:30:24 (96.15%)	5d, 15:35:35 (96.67%)	5d, 7:46:41 (99.93%)
Recomp.	0:00:00 (0.00%)	0:34:11 (0.41%)	1:43:37 (1.28%)	3:17:31 (2.43%)	0:00:00 (0.00%)
Save	0:11:47 (0.15%)	7:04:57 (5.08%)	1:23:18 (1.03%)	0:42:29 (0.52%)	0:00:25 (0.01%)
Alloc.	1:38:22 (1.25%)	1:47:29 (1.29%)	1:45:07 (1.30%)	1:46:02 (1.30%)	0:02:07 (0.03%)
Prep.	1:54:30 (1.45%)	2:05:31 (1.50%)	2:02:45 (1.52%)	2:02:43 (1.51%)	0:02:40 (0.03%)
Total	5d, 11:31:28	5d, 19:18:59	5d, 14:41:34	5d, 15:35:35	5d, 7:51:54
Cost ($)	**303.1**	**321.1**	**310.3**	**312.5**	**793.4**

Table 6.2 GPT-117M Spot training result breakdown.

time.

For GPT-117M, since *Spotify* can save a checkpoint and upload it to cloud storage in the 30-second grace period, *emergency save* is available. Therefore, *Spotify* can reduce much more time by *adaptive checkpointing* than the optimal static checkpointing per 51 steps. Table 6.2 shows that *Spotify* saves 62% of the VM cost with only sacrificing 2.86% latency relative to the on-demand training by leveraging the advantage of *adaptive checkpointing*. The total latency for VM allocation and preparation is almost constant under all checkpointing policies.

6.3.2 Efficiency of Insurance Save

We evaluate the checkpointing policies for the heavier-loaded job which trains GPT-1.3B for 10K steps. Unlike the GPT-117M training job, *emergency save* is

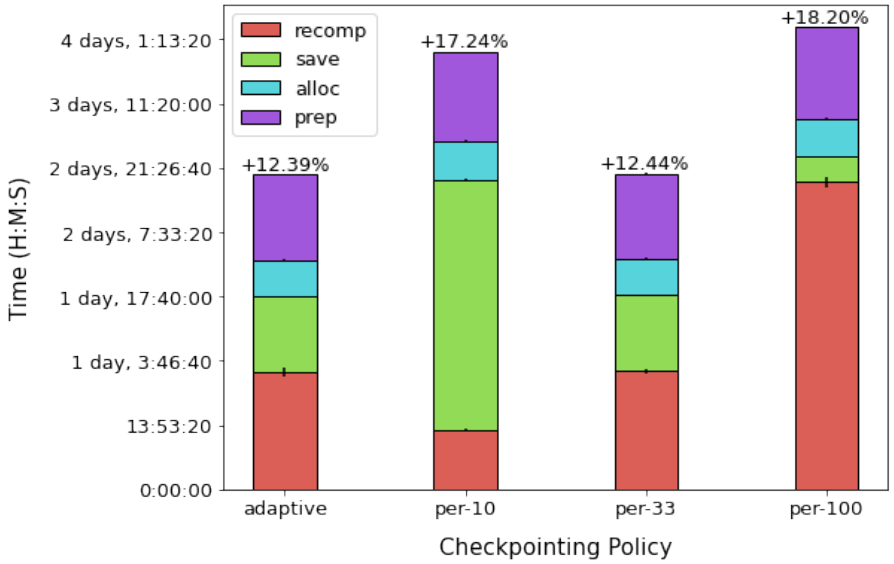

Figure 6.2 GPT-117M spot training latency overhead for 0.1M iterations. The $MTTP$ value is 10800 (3 hours).

unavailable since the sum of computation time, checkpointing time, and backup time is larger than the 30-second grace period. According to the *adaptive checkpointing*, *Spotify* automatically configures the optimal checkpointing interval at every checkpointing time based on the profiled job runtime information and system-widely collected $MTTP$ and VM allocation latency.

Figure 6.2 shows the additional latency overhead of training GPT-1.3B on a spot VM relative to the on-demand training latency. Checkpointing per 33 steps is the optimal checkpointing interval calculated by hand, and it is at the optimum sweet spot that minimizes the entire job latency with 12.44% of overhead. *Spotify* systematically configures the optimum according to the profiling information, and users do not need to struggle to find out the proper

	Adaptive	Per-10	Per-33	Per-100	On-demand (Per-10000)
Comp.	23d, 21:52:08 (93.07%)	23d, 9:25:01 (87.29%)	23d, 22:07:45 (93.07%)	25d, 15:00:42 (94.84%)	22d, 20:37:09 (99.88%)
Recomp.	1d, 1:14:57 (4.09%)	12:47:49 (1.99%)	1d, 1:31:20 (4.14%)	2d, 18:23:41 (10.24%)	0:00:00 (0.00%)
Save	16:22:29 (2.66%)	2d, 6:10:03 (8.42%)	16:24:42 (2.66%)	5:24:58 (0.84%)	0:32:30 (0.10%)
Alloc.	7:49:17 (1.27%)	8:12:37 (1.28%)	7:49:07 (1.27%)	8:19:56 (1.28%)	0:02:16 (0.01%)
Prep.	18:31:32 (3.00%)	19:23:12 (3.01%)	18:30:19 (3.00%)	19:41:20 (3.04%)	0:05:25 (0.02%)
Total	25d, 16:35:27	26d, 19:10:54	25d, 16:51:55	27d, 0:26:57	22d, 21:17:22
Cost ($)	**1421.1**	**1482.4**	**1421.7**	**1494.5**	**3408.3**

Table 6.3 GPT-1.3B Spot training result breakdown.

checkpointing interval. Moreover, *Spotify* can flexibly adjust the checkpointing interval during the job runtime. Because the probability of preemption can be changed during the job runtime according to the cloud VM availability, without *Spotify*, users have to stop-and-restart the job with a newly configured checkpointing interval when the preemption rate is drastically changed.

Table 6.3 shows the cost and latency breakdown of GPT-1.3B training jobs with different checkpointing policies. The *adaptive checkpointing* and the per-31-step checkpointing, which are optimum, enable the spot training for a 58% cheaper price than the on-demand training with about 12% latency overhead. Similar to the GPT-117M training job, per-10-step checkpointing and per-100-step checkpointing increase the latency and cost due to the high checkpointing and recomputation overhead respectively.

6.4 Effect of Preemption Frequency

In this section, we evaluate how the preemption frequency affects the spot training latency. Figure 6.3 shows the latency overhead of GPT-1.3B training job for different preemption rates, where $MTTP$ values are from 3,600 (1 hour) to 43,200 (12 hours). When preemption occurs very frequently like 1 or 2 hours, the latency overhead is not negligible due to the high recovery cost. However, as the preemption rate gets lower, the overhead also decreases exponentially. That is, when the preemption frequency is at a moderate level, *Spotify* enables the spot training at a reasonable latency overhead while getting a huge price discount.

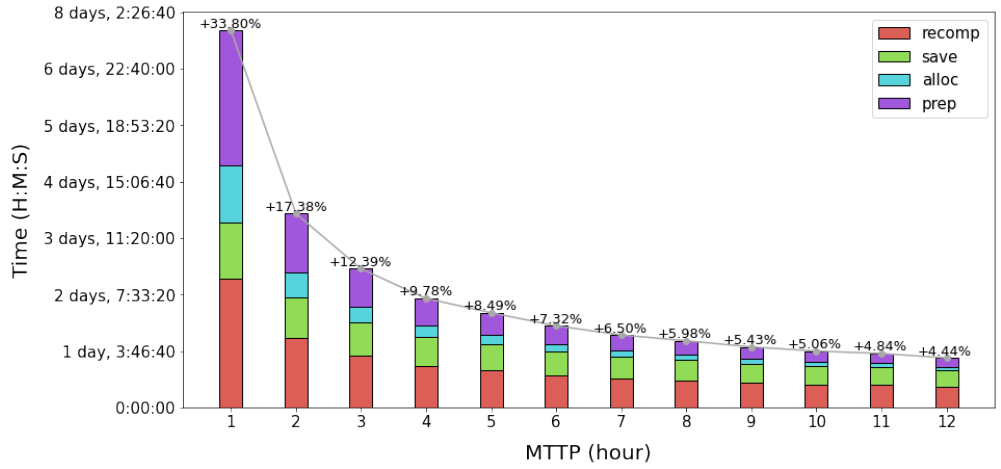

Figure 6.3 GPT-1.3M spot training latency overhead for 0.1M iterations for different MTTP values when the *adaptive checkpointing* is used.

Chapter 7

Conclusion

By leveraging the runtime profiling information, *Spotify* enables reliable and cost-efficient AI model training on preemptible cloud clusters. Our system adaptively schedules checkpointing intervals during the job runtime based on the real-time profiling metrics. The evaluation conducted with our offline simulation framework shows that the *adaptive checkpointing* policy can find out the optimal checkpointing strategy for both small and large-size deep learning models, and it reduces the end-to-end training cost up to 62% with only 2.86% of latency overhead.

Bibliography

[1] "Amazon EC2 Spot Instances." https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html.

[2] "Azure Spot Virtual Machines." https://docs.microsoft.com/en-us/azure/virtual-machines/spot-vms.

[3] "Google Cloud Spot VM." https://cloud.google.com/compute/docs/instances/spot.

[4] "Amazon EC2 Spot Instances: Iterruption reasons." https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/interruption-reasons.html.

[5] "Azure Spot Virtual Machines: Eviction policy." https://docs.microsoft.com/en-us/azure/virtual-machines/spot-vms#eviction-policy.

[6] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang, "How to bid the cloud," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, (New York, NY, USA), p. 71–84, Association for Computing Machinery, 2015.

[7] S. Tang, J. Yuan, and X.-Y. Li, "Towards optimal bidding strategy for amazon ec2 cloud spot instance," in *2012 IEEE Fifth International Conference on Cloud Computing*, pp. 91–98, 2012.

[8] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing amazon ec2 spot instance pricing," *ACM Trans. Econ. Comput.*, vol. 1, sep 2013.

[9] "Azure Metadata Service: Scheduled Events for Linux VMs." `https://docs.microsoft.com/en-us/azure/virtual-machines/windows/scheduled-events`.

[10] "EC2 instance rebalance recommendations." `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/rebalance-recommendations.html`.

[11] "Google Cloud Compute Engine: Handle GPU host events." `https://cloud.google.com/compute/docs/gpus/gpu-host-maintenance`.

[12] A. Harlap, A. Tumanov, A. Chung, G. R. Ganger, and P. B. Gibbons, "Proteus: Agile ml elasticity through tiered reliability in dynamic resource markets," in *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, (New York, NY, USA), p. 589–604, Association for Computing Machinery, 2017.

[13] P. Sharma, T. Guo, X. He, D. Irwin, and P. Shenoy, "Flint: Batch-interactive data-intensive processing on transient servers," in *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, (New York, NY, USA), Association for Computing Machinery, 2016.

[14] J. Kadupitige, V. Jadhao, and P. Sharma, "Modeling the temporally constrained preemptions of transient cloud vms," in *Proceedings of the 29th*

International Symposium on High-Performance Parallel and Distributed Computing, HPDC '20, (New York, NY, USA), p. 41–52, Association for Computing Machinery, 2020.

[15] B. Ghit and D. Epema, "Better safe than sorry: Grappling with failures of in-memory data analytics frameworks," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '17, (New York, NY, USA), p. 105–116, Association for Computing Machinery, 2017.

[16] A. Marathe, R. Harris, D. Lowenthal, B. R. de Supinski, B. Rountree, and M. Schulz, "Exploiting redundancy for cost-effective, time-constrained execution of hpc applications on amazon ec2," in *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '14, (New York, NY, USA), p. 279–290, Association for Computing Machinery, 2014.

[17] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Gener. Comput. Syst.*, vol. 22, p. 303–312, feb 2006.

[18] S. Athlur, N. Saran, M. Sivathanu, R. Ramjee, and N. Kwatra, "Varuna: Scalable, low-cost training of massive deep learning models," in *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, (New York, NY, USA), p. 472–487, Association for Computing Machinery, 2022.

[19] J. Thorpe, P. Zhao, J. Eyolfson, Y. Qiao, Z. Jia, M. Zhang, R. Netravali, and G. H. Xu, "Bamboo: Making preemptible instances resilient for affordable training of large dnns," 2022.

[20] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Commun. ACM*, vol. 17, p. 530–531, sep 1974.

[21] J. Daly, "A model for predicting the optimum checkpoint interval for restart dumps," in *Proceedings of the 2003 International Conference on Computational Science*, ICCS'03, (Berlin, Heidelberg), p. 3–12, Springer-Verlag, 2003.

[22] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[23] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, "The pile: An 800gb dataset of diverse text for language modeling," 2021.

[24] "Azure Virtual Machines - Simulate Eviction." `https://docs.microsoft.com/en-us/rest/api/compute/virtual-machines/simulate-eviction`.

[25] "AWS Fault Injection Simulator Documentation." `https://docs.aws.amazon.com/fis/index.html`.

초록

인공지능 모델 학습을 위해 물리적으로 GPU 클러스터를 구축 및 관리하는 데에는 많은 비용이 투자되어야 한다. 이에 따라 인공지능 모델 개발자들 사이에서는 사용한 만큼의 비용만을 지불하여 사용이 가능한 클라우드 클러스터를 사용하여 모델 학습을 하려는 수요가 점차 증가하고 있다. 특히 큰 폭의 할인된 가격으로 제공되는 선점가능형 가상머신을 사용하여 모델 학습을 하는 방식이 큰 주목을 받고 있다. 하지만 선점가능형 가상머신은 클라우드 제공사에 의해 언제든지 일방적으로 선점을 당할 수 있기 때문에 진행 중이던 학습 상태의 손실이 야기될 수 있다. 비용과 안전성 면에서 교환이 발생하기 때문에 개발자들은 선점가능형 가상머신을 모델 학습 및 실험에 적극적으로 사용하는 데 어려움을 겪고 있다. 본 연구에서는 선점가능형 가상머신에서 인공지능 모델 학습을 진행하는 데 있어 존재하는 주요한 어려움들에 대해 논의하고, 자동화된 방식을 통해 그러한 어려움을 해결함으로써 선점가능형 클라우드 클러스터에서 안정적인 학습을 가능하게 하는 인공지능 모델 학습 작업 관리 시스템인 *Spotify*를 제안한다. 우리의 실험 결과는 *Spotify*가 선점가능형 클라우드 클러스터에서 학습을 수행할 때 온디맨드 클라우드 클러스터에서 학습을 진행하는 것 대비 2.86%의 지연시간 오버헤드만을 희생하여 최대 62%에 달하는 비용을 절약할 수 있음을 보인다.

주요어: 기계학습, 클라우드, 선점 핸들링

학번: 2020-27792