DeepVM: Integrating Spot and On-Demand VMs for Cost-Efficient Deep Learning Clusters in the Cloud

Yoochan Kim[1], Kihyun Kim[1], Yonghyeon Cho[1,†], Jinwoo Kim[1], Awais Khan[2], Ki-Dong Kang[3]
Baik-Song An[3], Myung-Hoon Cha[3], Hong-Yeon Kim[3], Youngjae Kim [1,‡]

[1]*Dept. of Computer Science and Engineering, Sogang University, Seoul, Republic of Korea*
[2]*Oak Ridge National Laboratory, TN, USA,* [3]*ETRI, Daejeon, Republic of Korea*

***Abstract*—Distributed Deep Learning (DDL), as a paradigm, dictates the use of GPU-based clusters as the optimal infrastructure for training large-scale Deep Neural Networks (DNNs). However, the high cost of such resources makes them inaccessible to many users. Public cloud services, particularly Spot Virtual Machines (VMs), offer a cost-effective alternative, but their unpredictable availability poses a significant challenge to the crucial checkpointing process in DDL. To address this, we introduce DeepVM, a novel solution that recommends cost-effective cluster configurations by intelligently balancing the use of Spot and On-Demand VMs. DeepVM leverages a four-stage process that analyzes instance performance using the FLOPP (FLoating-point Operations Per Price) metric, performs architecture-level analysis with linear programming, and identifies the optimal configuration for the user-specific needs. Extensive simulations and real-world deployments in the AWS environment demonstrate that DeepVM consistently outperforms other policies, reducing training costs and overall makespan. By enabling cost-effective checkpointing with Spot VMs, DeepVM opens up DDL to a wider range of users and facilitates a more efficient training of complex DNNs.**

***Index Terms*—Cloud Computing, Distributed Deep Learning, Checkpoint-Restart**

I. Introduction

Distributed Deep Learning is used to train Deep Neural Networks (DNNs) in environments that involve multiple devices, typically GPUs, or multiple nodes in a network. By leveraging data parallelism or model parallelism techniques [1, 2, 3, 4, 5, 6, 7], DDL addresses the high computational demands of training DNNs and significantly reduces the training. However, setting up a GPU-based cluster for large-scale training is challenging and costly, making it impractical for small organizations to run extensive DDL workloads. Fortunately, there is an alternative available in the form of public cloud services provided by major providers such as Amazon AWS, Microsoft Azure, and Google Cloud Platform.

Users can run DDL workloads at low costs without the need for an on-premise GPU cluster with public cloud services [8]. Cloud Service Providers (CSPs) offer various options that cater to several budgetary considerations. There are *On-Demand VM* instances available at regular prices as well as surplus resources known as *Spot VM* instances, which can be obtained at significantly discounted prices, up to 90% off (in case of AWS). Therefore, Spot VM instances have garnered attention among cost-conscious users because they allow for the use of a VM with the same specifications as an On-Demand instance at a much lower price. CSPs provide Spot VM instances as a specialized resource to maximize profits by utilizing idle resources. However, it is important to note that there is a trade-off in terms of availability. Depending on pricing fluctuations and demand [9], Spot VM instances may be terminated to make the resources available to other users. Due to the nature of Spot VM instances provided by the CSP, they are cost-effective but come with the inherent risk of sudden interruption or termination at any time, as noted in SpotLake [10]. To mitigate this risk, users should employ methods such as checkpointing that demonstrate tolerance towards interruptions when using Spot VM instances.

Checkpointing Challenges in Spot VMs: In DDL, Checkpoint-restart is a key technique, widely used for a range of purposes. It is instrumental in preserving the state of the best-performing model, safeguarding against unexpected interruptions, and facilitating debugging during training. However, the use of checkpoint-restart in Spot VM environment presents a significant challenge. The main issue arises when a Spot VM instance is preempted. In such cases, users lose access to the VM's storage, which contains the checkpoint data. This loss of access makes it impossible to use the Spot VM's storage as a reliable medium for checkpointing. Since checkpointing relies on persistently storing the state of the DNN model and training process, the inability to access the storage on a preempted Spot VM renders the checkpointing process ineffective. This inherent risk of Spot VMs being preempted and losing storage access severely limits their suitability for checkpoint-restart in DDL, where data integrity and continuity are crucial.

Thus, other options must be considered for checkpointing on Spot VMs. One can adopt external cloud-based storage solutions. However, these solutions come with several non-trivial challenges, e.g., significant costs associated with data transfer and storage, making them economically unfeasible for frequent checkpoints of large models. Due to these challenges, it might be necessary to use on-demand VM instances for checkpointing on Spot VMs.

† Y. Cho is currently affiliated with LG Electronics.
‡Y. Kim is the corresponding author.

979-8-3503-9566-2/24/$31.00 ©2024 IEEE
DOI 10.1109/CCGrid59990.2024.00034

Challenges in establishing an Economical VM Cluster including On-Demand VM: The primary challenge in establishing an economical VM cluster, including on-demand options, lies in the complex and variable pricing structure of these resources. This complexity necessitates careful evaluation to find the best cluster configuration that not only fits within budgetary constraints but also maximizes efficiency in various dimensions [11]. This is crucial because, while GPU-based on-demand VMs are typically more expensive and thus pose a financial challenge, CPU-based on-demand VMs can sometimes be comparably priced or even cheaper than Spot GPU-VMs. Therefore, users must consider not only the technical differences between GPU and CPU VMs but also the economic implications of each choice.

Limitations of Existing Approaches: There is a very small effort to construct cost-effective clusters that leverage both Spot and On-Demand VMs. Some studies [12, 13, 14, 15] are limited by their narrow applicability, as they are effective only in certain usecases and/or are tailored for specific architectures. Other approaches [16, 17, 18, 19] fail to propose methods for cluster configuration adequately, lack comprehensive analysis or experimentation in complex scenarios, and fail to simultaneously consider both price and performance.

Therefore, this study introduces DeepVM, a novel approach toward cost-effective cluster configurations by intelligently balancing the use of Spot and On-Demand VMs. This operates through a four-stage process, (i) DeepVM begins by collecting the user pricing willingness; (ii) analyzing all available instances, using the 'FLOPP' (Floating-point Operations Per Price) metric to assess each instance's performance in terms of its cost; (iii) the performance of each instance is evaluated within predefined architectures. This stage employs linear programming to examine the most cost-effective combinations within each architecture, taking into account constraints and potential overheads; (iv) identifies the best combination, culminating the process by choosing the most efficient and economical configuration for user-specific needs. DeepVM achieves cost-efficiency, performance consideration, and flexibility through these stages.

We evaluated DeepVM using simulations and a real deployment in the AWS environment. For the simulation, we experimented with seventeen different types of virtual instances. The results show that the configurations recommended by DeepVM consistently outperformed the other policies across all price ranges. For real-world AWS environment, we performed experiments on ten available instance types available at that time. The configurations recommended by DeepVM exhibited lower total costs and shorter makespan compared to other policies.

II. Motivation

A. Spot VMs and the On-Demand Dilemma in Checkpointing

In distributed deep learning (DDL), users utilize checkpoints to periodically save training parameters such as DDL models and optimizer states. These checkpoints serve multiple purposes. Firstly, they assist in resuming training from the point of interruption due to system failures or unexpected terminations [1, 2, 20, 21, 22, 23, 24, 25]. Secondly, they contribute to comparing the performance of models at specific epochs, aiding in the selection of the optimal model [26, 27, 28]. Thirdly, they are beneficial in addressing various issues such as debugging, and regression analysis that may arise during the deep learning process [29, 30, 31]. All these purposes significantly contribute to enhancing the efficiency and effectiveness of DDL training.

Nevertheless, using checkpoints in Spot VM clusters presents several challenges. The primary issue is the inaccessibility of Spot VM instance storage upon preemption, rendering local storage unusable for checkpoint storage. Consequently, users must consider alternative storage solutions. One immediate alternative is the use of external storage; however, this too comes with its own set of challenges. For example, when using external storage, write speed issues can occur. In our empirical analysis, we evaluated cloud I/O performance compared to on-premises local setups using the FIO benchmark. Our findings revealed that the write speed on AWS's Elastic Block Store (EBS) is significantly slower than that on local SSDs. A detailed comparative analysis of our empirical analysis is provided in [32]. This difference in speed can adversely affect the learning process, highlighting the importance of rapid data storage, especially given the unpredictable preemption time of Spot VMs in a cloud environment.

To address these problems, users can opt for on-demand instances. On-demand instances are not subject to preemption, thereby ensuring the stability of checkpoint data and reducing the need for rapid checkpointing. However, on-demand instances are more expensive than Spot VMs, and their use involves considering several variables. Decisions such as whether to involve on-demand instances in training, use them solely as memory nodes for checkpointing, choose the type of instances, and determine the number of instances required, complicate the economic configuration of the cluster and present a significant challenge to users.

B. Existing Approaches and Their Limitations

We observe several limitations in existing resource allocation studies that make them unsuitable for constructing cost-effective clusters leveraging both Spot and On-Demand VMs.

Narrow Applicability: Several existing approaches often suffer from limited applicability, restricting their effectiveness in specific scenarios. Proteus [12] only considers distributed learning environments with a Parameter Server (PS) architecture, and as the number of Spot VMs increases, the proportion of On-Demand VM usage also increases, reducing cost-efficiency. Spotnik [13] is limited to situations where only part of the cluster faces preemption and is incapable of avoiding complete checkpoint approaches when the entire cluster is revoked. DeepSpotCloud [14] is focused only on single GPU instances and lacks the necessary expansion for distributed deep learning platforms. Andrzejak et al. [15] only consider Spot VMs and does not consider On-Demand VMs.

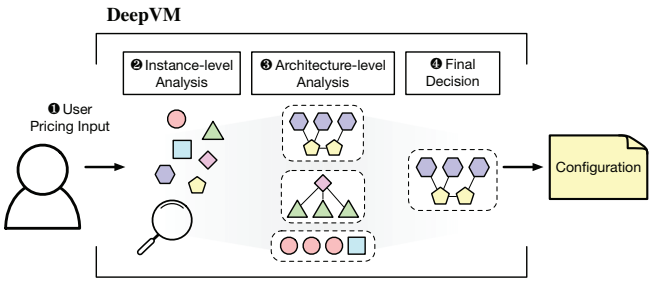

Fig. 1. An overview of DeepVM.

Inadequate Methods for Cluster Configuration: Some existing approaches fail to adequately propose methods for cluster configuration or have limitations in their proposed approaches. H. Wang et al. [16] analyze the relationship between cloud pricing and system configurations, but lacks a method for how best to configure clusters based on this analysis and does not experiment with it. S. Deochake [17] investigates various methods for optimizing cloud costs, acknowledging the potential for cost savings. However, it lacks analysis in complex scenarios, particularly in configuring clusters considering both price and performance simultaneously. P. Kokkinos et al. [18] perform resizing based on the utilization and cost of currently used instances. However, it fails to propose a cluster configuration prior to performing tasks, limited to configuring clusters only through monitoring. E. M. Malta et al. [19] identify cost-effective instances for deep learning applications based on the cost and performance of instances and presents a general methodology. However, this approach requires predicting performance and cost for each application, leading to the disadvantage that the most efficient instance may vary for each application.

III. Design of DeepVM

DeepVM operates through the following four stages: User Pricing Input, Instance-level Analysis, Architecture-level Analysis, and Final Decision.

- **Step#1:** User Pricing Input: DeepVM begins with collecting the user's Pricing Willingness. In this initial phase, the algorithm gathers data on the maximum price per hour that the user is willing to pay.
- **Step#2:** Instance-level Analysis: This stage analyzes all available instances for the user. Using the 'FLOPP' metric, it measures each instance's performance relative to its cost.
- **Step#3:** Architecture-level Analysis: Upon completing instance analysis, the performance of each instance in predefined architectures is analyzed. This stage explores the most cost-effective instance combinations within these architectures using linear programming, considering each architecture's constraints and potential overheads in parallel processing.
- **Step#4:** Final Decision: After identifying the best combinations for all architectures, this stage compares their performance to propose the most cost-effective cluster configuration under the given conditions.

TABLE I
Declaration of Variables

Variable	Description
Input	
$\mathbf{V}$	Set of GPU-VM instances
v	GPU-VM instance
$\mathbf{W}$	Set of CPU-VM instances
w	CPU-VM instance
PW	Pricing willingness
f	Size of the checkpoint file
s	Buffer size of remote memory
$\mathbf{A}$	Set of VM cluster architectures and their constraints and performance functions.
Output	
a_0	Recommended architecture
v_0	Recommended GPU-VM instances
n_0	Number of v_0
w_0	Recommended CPU-VM instances
m_0	Number of w_0
P_0	Hourly price
Element of Instance Tuple	
$SPFP(x)$	Spot VM *FLOPP* of instance x
$ODFP(x)$	On-demand VM *FLOPP* of instance x
$SPPR(x)$	Spot VM price per hour of instance x
$ODPR(x)$	On-demand VM price per hour of instance x
$MEM(x)$	Memory capacity of instance x

Figure 1 describes these steps of DeepVM. Table I defines the mathematical notations used in the algorithm.

1) User Pricing Input: In the initial phase of the DeepVM algorithm, the user is prompted to specify their Pricing Willingness (PW). PW represents the maximum amount, on an hourly basis, that the user is willing to pay. The user defines their financial threshold for the cloud resources by setting their maximum affordable hourly rate. This rate is pivotal in guiding the subsequent stages of the algorithm. Based on the primary intention of forming an economical cluster using spot instances, it is assumed that users will typically propose a price lower than what a fully on-demand cluster would cost. This assumption aligns with the goal of achieving cost-efficiency in a cloud environment. The flexibility in PW allows users to tailor their experience. Those seeking higher performance might opt for a higher PW, while users prioritizing cost savings might set a lower PW. This variability ensures that the algorithm caters to a diverse range of user needs and preferences. Throughout the subsequent stages, the algorithm utilizes the PW as a key parameter. It strives to maximize performance relative to cost, aiming to recommend the most cost-effective cluster configuration that aligns with the user's set PW.

2) Instance-level Analysis: To analyze the cost-efficiency of individual instances, understanding how to simultaneously consider price and performance is essential. For performance, one might consider the theoretical FLOPS (Floating Point Operations Per Second) of GPUs. However, actual deep learning performance can differ due to factors like data transfer and memory access patterns. To address this, we introduced 'eFLOPS', a numerical metric representing performance in executing deep learning workloads. We specifically selected image processing workloads as representative tasks for deep learning, benchmarking them across various AWS instances

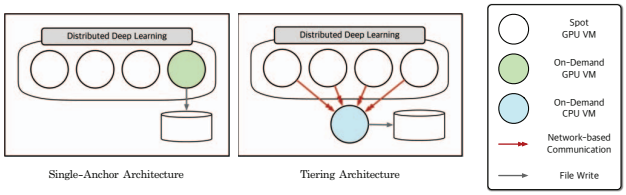

Fig. 2. Depiction of Single Anchor and Tiering Architectures.

to reflect real-world performance. Additionally, users or administrators interested in different workloads/tasks can benchmark performance using execution time. Parallel to this, we analyzed the hourly pricing of instances provided by Cloud Service Providers (CSPs) to consider the cost aspect. This led to the derivation of 'FLOPP' (Floating-point Operations Per Price), a metric defined in Equation 1.

$$\begin{aligned} \text{FLOPP} &:= \text{eFLOPS} \div (\text{price per time}) \\ &= \frac{operations}{time} \times \frac{time}{price} = \frac{operations}{price} \end{aligned} \tag{1}$$

FLOPP indicates the amount of deep learning operations that can be performed per unit price, which varies based on spot and on-demand options, and price fluctuations. DeepVM pre-calculates the eFLOPS value for each instance. Utilizing this, the algorithm computes the on-demand and spot FLOPP values for available instances to determine the most optimal instance selection.

3) Architecture-level Analysis: Following the instance evaluation, DeepVM considers a wide and expandable range of predefined architectures **A** to find the optimal combination of instances. For simplicity, we assume all GPU-VMs use the same instance. The performance of an architecture is represented by the product of the performance of the GPU-VMs involved in training and their quantity. We account for different FLOPP values between spot and on-demand instances and introduce a *ScalingFactor* to reflect the non-linear performance increase in parallel tasks. Thus, the performance function Z, as described in Equation 2, models this relationship.

$$Z = (N \times SPFP(v) + M \times ODFP(v)) \times ScalingFactor(v, n) \tag{2}$$

where N is the number of spot instances, and M is the number of on-demand instances.

The paper focuses on two architectures: 'Single Anchor', a combination of one on-demand GPU-VM and multiple spot GPU-VMs, and 'Tiering', a combination of on-demand CPU-VMs as memory nodes and spot GPU-VMs for training. Figure 2 illustrates the two architectural scenarios assumed in this paper. Further analysis of each architecture, including their constraints, and an in-depth discussion on overhead considerations, including the *ScalingFactor*, is presented in [32].

4) Final Decision: Upon completing the analysis of architectures, the maximum values of the performance function Z for each architecture are compared. This process selects the architecture that provides the best performance within the given pricing willingness, ensuring the highest cost-efficiency. The selected architecture is then presented to the user, who has the option to utilize it or return to the initial step with a modified PW value, considering different price or performance requirements.

TABLE II
Specifications of Simulated VMs, including 10 GPU-VMs and 7 CPU-VMs based on AWS specifications.

Name	Type	On-Demand	Spot VM	Network (Gbps)	eFLOPS	Name	Type	On-Demand	Spot VM	Network (Gbps)
A	GPU	0.75	0.225	10	100	K	CPU	0.199	0.126	1.7
B	GPU	0.526	0.158	25	377	L	CPU	0.1664	0.103	5
C	GPU	1.006	0.302	10	696	M	CPU	0.17	0.1	10
D	GPU	0.55	0.165	15	700	N	CPU	0.192	0.12	12.5
E	GPU	0.368	0.11	1.7	100	O	CPU	0.499	0.158	15
F	GPU	1.236	0.371	25	800	P	CPU	0.216	0.127	25
G	GPU	0.973	0.292	30	200	Q	CPU	0.2268	0.127	30
H	GPU	0.252	0.076	5	150	-				
I	GPU	1.622	0.487	30	900	-				
J	GPU	0.22	0.066	5	50	-				

TABLE III
Various Types of instances provided by AWS. This information was available on October 1, 2023 in N.Virginia region.

Name	Type	On-Demand	Spot VM	vCPU (#)	Mem (GiB)	Network (Gbps)	Storage (Gbps)
g3s.lxarge	GPU	0.75	0.225	4	30.5	- 10	N/A
g4dn.xlarge	GPU	0.526	0.1941	4	16	- 25	- 3.5
g5.xlarge	GPU	1.006	0.3018	4	16	- 10	- 3.5
p2.xlarge	GPU	0.2704	0.225	4	61	High	0.75
p2.8xlarge	GPU	7.2	2.1165	32	488	10	- 5
p2.16xlarge	GPU	14.4	4.2262	64	732	25	- 10
c3.xlarge	CPU	0.21	0.1281	4	7.5	Moderate	N/A
c4.xlarge	CPU	0.199	0.1257	4	7.5	High	0.75
t3.xlarge	CPU	0.1664	0.1033	4	16	- 5	N/A
c5.xlarge	CPU	0.17	0.0999	4	8	- 10	- 4.75
m6i.xlarge	CPU	0.192	0.12	4	16	- 12.5	- 10
d3.xlarge	CPU	0.499	0.1584	4	32	- 15	0.85
c5n.xlarge	CPU	0.216	0.127	4	10.5	- 25	- 4.75
c6in.xlarge	CPU	0.2268	0.1272	4	8	- 30	- 20

IV. Evaluation

A. Experimental setup

To evaluate DeepVM, we implemented a two-pronged experimental approach that included both simulated and actual AWS environment setups. This approach allowed us to test the system within the constraints and variability of real-world cloud infrastructure. Additionally, it enabled us to extend our investigation into a controlled simulation, encompassing a wider array of instance types and configurations than those currently available on AWS. By employing this dual setup, we aimed to provide a robust validation of DeepVM's performance across a spectrum of scenarios. DeepVM source code along with the associated tools and algorithms, is publicly available at https://github.com/lass-lab/DeepVM. For detailed instructions on how to use these resources, please refer to the appendix A.

Simulation Setup: Given the limited availability of specific instance types on AWS and restrictions on the number of instances, we encountered constraints in the experimental setup. To overcome these limitations and validate a broader range of instances, we turned to simulations. Consequently,

we created additional virtual instances, simulating 10 GPU-VMs and 7 CPU-VMs based on AWS specifications as shown in Table II. Among these, instances D through J were specifically designed to meet our experimental objectives, offering a variety of performance, pricing, and parallel processing capabilities. This approach allowed us to explore more diverse configurations and their impact on performance. Detailed descriptions of these instances, including their characteristics, are listed in Table II. In this setup, we use the following metric to assess DeepVM.

- **Performance:** This represents a simulation of the capability to process deep learning workloads. The value is calculated by multiplying the eFLOPS of the GPU-VMs and their quantity, along with the *ScalingFactor* that indicates the efficiency of parallel processing. Through this, we can evaluate the performance of each configuration.

Real AWS Experiment Setup: The experiments on AWS were conducted in the Northern Virginia (us-east-1) region from 5 am to 9 pm KST, spanning from September to December 2023. To minimize the impact of cloud user traffic and maintain consistent experimental conditions, tests were carried out across various availability zones (from us-east-1a to us-east-1f). For the GPU-VMs, we utilized `g3.xlarge`(NVIDIA M60), `g4.xlarge`(NVIDIA T4), and `g5.xlarge`(NVIDIA A10G), and for the CPU-VMs, we used the CPU type instances specified in Table III. The instances' storage was selected as AWS's EBS(gp2). Due to AWS's vCPU allocation policy, our usage of GPU-VMs was limited to a maximum of 32 out of 128 vCPUs. To validate the efficiency of DeepVM, we employed major image processing models like ResNet-18 and EfficientNet-v2-m, using the CIFAR-10 dataset. The batch size for single GPU processing was fixed at 128, and each DDL training was performed up to 50 epochs utilizing the Adam optimizer with the `ReduceLROnPlateau` scheduler. We report an average of 5 runs for each experiment. To complement our workload, we also considered using ResNet-18, ResNet-34, ResNet-50, ResNet-152, as well as EfficientNet-V2L models.

We use two metrics to evaluate DeepVM performance.

- **Total cost:** The actual cost incurred in executing the workload and calculated based on the number of instances, the hourly rate, and the makespan. It is used to evaluate the cost-efficiency of a specific VM cluster configuration.
- **Makespan:** The total time required to complete the workload execution. The makespan duration impacts the overall cost, and is directly proportional to preemption probability of spotVM. It is used to assess how quickly training was conducted using a specific VM cluster configuration.

B. Effectiveness of DeepVM

1) Baseline: In this experiment, we evaluate DeepVM's configuration policy against various other policies to verify how effectively DeepVM can configure a range of instances. As mentioned in Section II-B, no existing studies were found that were conducted in a similar environment. Therefore, for baseline comparison, we introduce three distinct policies (algorithms). All algorithms take into account the user Pricing Willingness (PW). In the simulation, instances can be utilized without a limit on their number as long as their hourly cost does not exceed PW. In actual experimental environments, however, the maximum number of instances is determined by AWS policies.

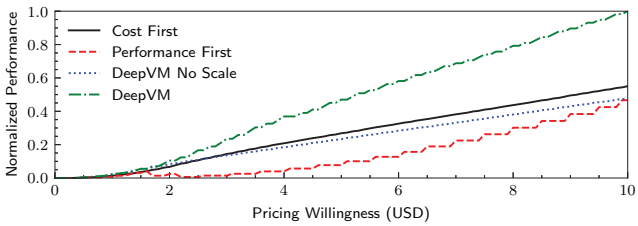

Fig. 3. This graph represents the normalized performance of VM cluster configurations recommended by Cost-First, Performance-First, DeepVM-NoScale, and DeepVM policies across different levels of Pricing Willingness (PW) ranging from $0 to $10. Performance is normalized to the highest performance observed for DeepVM at PW=$10.

- Cost-First: This policy utilizes the most cost-effective GPU-VM instances per hour as the single anchor. This setup aims to use as many instances as possible.
- Performance-First: This policy is oriented towards GPU-VM instances that have the highest FLOPS, often correlating with higher costs and potentially fewer instances used.
- DeepVM-Noscale: This policy employs the same algorithm as DeepVM, but does not consider the *ScalingFactor*. Consequently, it may select instances with high FLOPP but less suitable for parallel processing.

Cost-First and Performance-First focus primarily on the GPU-VM aspect, thus both algorithms essentially consider a Single-Anchor architecture in their approach.

2) Validation in Simulation: The simulation was conducted by varying the Pricing Willingness (PW) from $0 to $10 in increments of $0.1. For each PW value, the recommended configurations from four policies (Cost-First, Performance-First, DeepVM-Noscale, and DeepVM) were evaluated. Performance was assessed for each configuration based on the number of GPU-VMs, eFLOPS, and a performance function utilizing the *ScalingFactor*. The results of these experiments are depicted in Figure 3.

Our findings demonstrate that DeepVM consistently delivered the highest performance across all PW values. In contrast, DeepVM-Noscale showed poorer performance than Cost-First when PW exceeded $2.5, attributed to the presence of instances with high FLOPP but subpar parallel processing capabilities. The Cost-First policy, despite increasing PW values, did not show significant performance improvements compared to DeepVM. This is because it only considers cheaper instances, thus failing to account for instances that offer better performance relative to their cost. Performance-First experienced a sharp decrease in performance starting at a PW of $1.6. This decline occurred because the policy focuses solely on performance, missing the opportunity to utilize

multiple instances. However, as PW increases, Performance-First gradually improves in performance by considering multiple high-performing instances.

Interestingly, for all four policies - Cost-First, Performance-First, DeepVM-Noscale, and DeepVM - the performance improvement follows a stepwise pattern beyond certain PW thresholds. This phenomenon occurs because, for each policy, the optimal instance selection stabilizes beyond a specific PW value. As a result, every time the PW increases enough to include an additional instance of this optimal type, there is a noticeable incremental improvement in performance. This consistent pattern across different policies highlights how performance scales in a stair-step manner as PW allows for the inclusion of more instances of the most efficient type as per the respective policy's criteria. We observed that this trend consistently persists even beyond a PW of $10.

In conclusion, the three baseline policies fail to provide the optimal combination of instances. However, DeepVM effectively recommends the best possible combination by comprehensively considering various factors.

3) Validation in AWS: To validate the effectiveness of the VM cluster configuration proposed by DeepVM in a real AWS environment, we created instances on AWS and executed DDL workloads. Concurrently, VM cluster configurations suggested by three baseline policies were also included in the experiment, considering the top 3 configurations of DeepVM and one configuration according to each baseline policy. This comparative analysis was conducted to assess the cost efficiency of DeepVM. For a more realistic setup, the pricing willingness input for DeepVM was set to $3, based on the spot price of $2.5248 for 16 `g4dn.xlarge` instances. The experiments applied training workloads using the ResNet50 model.

The experimental results are depicted in Table IV and Figure 4. Table IV shows the VM cluster configurations as recommended by DeepVM, where the first recommendation is the least costly and the sequence follows an increasing order of total cost. This pattern demonstrates that DeepVM recommends configurations that effectively balance cost and performance. In contrast, the configurations suggested by the Cost-First and Performance-First policies led to higher total costs than those recommended by DeepVM. The DeepVM-NoScale policy recommended a configuration identical to the first configuration of DeepVM. This outcome is due to the absence of instances with high FLOPP but poor parallel processing capabilities among the provided instances.

Figure 4 compares the makespan, total cost, and hourly price of each configuration. Across all scenarios, DeepVM's recommendations resulted in the shortest makespans. The configurations suggested by the Cost-First and Performance-First policies, however, exhibited longer makespans, leading to higher overall costs. Particularly, the Performance-First policy's configuration nearly doubled the makespan and total cost compared to DeepVM's recommendations.

From these results, two key facts can be inferred: (i) DeepVM does not merely recommend configurations with low hourly prices. Instead, it provides recommendations based on a comprehensive consideration of various cloud VM variables. (ii) Recommendations by DeepVM do not compromise makespan delay just for a lower price.

TABLE IV
COST AND CONFIGURATION COMPARISON FOR 30-EPOCH RESNET50 TRAINING AT PW=$3: TOP THREE CONFIGURATIONS FROM DEEPVM AND BOTTOM THREE FROM BASELINE ALGORITHMS.

Configuration	Total cost (USD)	GPU	# of GPU	CPU	# of CPU
DeepVM 1st	0.1991	g4dn.xlarge	17	c5.xlarge	1
DeepVM 2nd	0.2013	g4dn.xlarge	17	m6i.xlarge	1
DeepVM 3rd	0.2022	g4dn.xlarge	17	c5n.xlarge	1
DeepVM-NoScale	0.1991	g4dn.xlarge	17	c5.xlarge	1
Cost-First	0.2385	g4dn.xlarge	16	-	-
Performance-First	0.3865	g5.xlarge	7	-	-

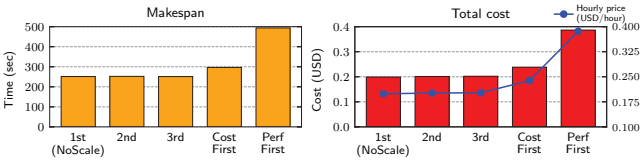

Fig. 4. Executing the same workload with PW=$3, Buffer size=2, and checkpoint file size=300MB, comparing DeepVM configuration (1st, 2nd, 3rd) and baseline policies (Cost-First, Perf-First, NoScale).

C. Extended Evaluation of Modeling Overhead

We explored DeepVM for its effectiveness in environments with multiple GPU-VMs and the challenge of network bandwidth saturation in distributed learning. Our extensive experiments, analyzing scalability and network efficiency under various configurations, substantiate these aspects. Although this brief paper cannot encompass all the experimental details and findings, a more comprehensive examination is available in the full version of the paper [32]. This extended analysis offers a deeper understanding of DeepVM's capabilities, particularly in handling the complexities of multiple GPUs and network bandwidth limitations. For a comprehensive analysis and additional details, the full version is available at [32].

V. CONCLUDING REMARKS

In this paper, we introduced DeepVM, a strategy for efficiently and economically utilizing Spot and On-Demand VMs in distributed deep learning, with a focus on overcoming the checkpointing challenges in Spot VM environments. Our simulations and AWS deployments showed that DeepVM significantly reduces cost with high performance, and outperforming existing baseline models. DeepVM is designed for flexibility and adaptability to various AWS instances and cloud services, requiring only updates to instance-specific data. Though focused on an image processing deep learning model in this paper, the method is broadly applicable to different deep learning workloads. This is achieved by customizing benchmarking metrics like 'eFLOPS' and overhead functions to suit specific deep learning tasks and instances, ensuring its effectiveness across diverse scenarios.

Acknowledgement

This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2022-0-00498, Development of high-efficiency AI computing SW core technology for high-speed processing of large learning models). This work was also supported by, and used the resources of, the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at ORNL, which is managed by UT Battelle, LLC for the U.S. DOE (under the contract No. DE-AC05-00OR22725).

References

[1] Q. Anthony and D. Dai, "Evaluating multi-level checkpointing for distributed deep neural network training," in *Proceedings of the 2021 SC Workshops Supplementary (SCWS)*, pp. 60–67, 2021.

[2] A. Eisenman, K. K. Matam, S. Ingram, D. Mudigere, R. Krishnamoorthi, K. Nair, M. Smelyanskiy, and M. Annavaram, "Check-N-Run: A checkpointing system for training deep learning recommendation models," in *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, NSDI '22, pp. 929–943, 2022.

[3] A. Khan, A. K. Paul, C. Zimmer, S. Oral, S. Dash, S. Atchley, and F. Wang, "Hvac: Removing i/o bottleneck for large-scale deep learning applications," in *Proceedings of the 2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 324–335, 2022.

[4] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, *et al.*, "Pytorch distributed: Experiences on accelerating data parallel training," *arXiv preprint arXiv:2006.15704*, 2020.

[5] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[6] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Proceedings of the Neural Information Processing Systems*, vol. 32 of *NIPS '19*, Curran Associates, Inc., 2019.

[7] C. Kim, H. Lee, M. Jeong, W. Baek, B. Yoon, I. Kim, S. Lim, and S. Kim, "torchgpipe: On-the-fly pipeline parallelism for training giant models," *arXiv preprint arXiv:2004.09910*, 2020.

[8] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Analysis and exploitation of dynamic pricing in the public cloud for ml training," in *Proceedings of the Workshop on Distributed Infrastructure, Systems, Programming, and AI*, August 2020.

[9] Z. Cai, X. Li, R. Ruiz, and Q. Li, "Price forecasting for spot instances in cloud computing," *Future Generation Computer Systems*, vol. 79, pp. 38–53, 2018.

[10] S. Lee, J. Hwang, and K. Lee, "Spotlake: Diverse spot instance dataset archive service," in *Proceedings of the 2022 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 242–255, 2022.

[11] G. Fragiadakis, V. Liagkou, E. Filiopoulou, D. Fragkakis, C. Michalakelis, and M. Nikolaidou, "Cloud services cost comparison: a clustering analysis framework," *Computing*, vol. 105, pp. 1–28, 03 2023.

[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, p. 589–604, 2017.

[13] M. Wagenländer, L. Mai, G. Li, and P. Pietzuch, "Spotnik: Designing distributed machine learning for transient cloud resources," in *Proceedings of the 12th USENIX Workshop on Hot Topics in Cloud Computing*, HotCloud '20, USENIX Association, July 2020.

[14] K. Lee and M. Son, "Deepspotcloud: Leveraging cross-region gpu spot instances for deep learning," in *Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing*, CLOUD '17, pp. 98–105, 2017.

[15] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under sla constraints," in *Proceedings of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '10, pp. 257–266, IEEE, 2010.

[16] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed systems meet economics: pricing in the cloud," in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing*, HotCloud '10, 2010.

[17] S. Deochake, "Cloud cost optimization: A comprehensive review of strategies and case studies," 2023.

[18] P. Kokkinos, T. A. Varvarigou, A. Kretsis, P. Soumplis, and E. A. Varvarigos, "Cost and utilization optimization of amazon ec2 instances," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, pp. 518–525, IEEE, 2013.

[19] E. M. Malta, S. Avila, and E. Borin, "Exploring the cost-benefit of aws ec2 gpu instances for deep learning applications," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, pp. 21–29, 2019.

[20] J. Mohan, A. Phanishayee, and V. Chidambaram, "CheckFreq: Frequent, Fine-Grained DNN checkpointing," in *Proceedings of the 19th USENIX Conference on File and Storage Technologies (FAST 21)*, pp. 203–216, Feb. 2021.

[21] B. Nicolae, J. Li, J. M. Wozniak, G. Bosilca, M. Dorier, and F. Cappello, "Deepfreeze: Towards scalable asynchronous checkpointing of deep learning models," in *Proceedings of the 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*, CCGRID '20, pp. 172–181, 2020.

[22] A. Wood, M. Hershcovitch, I. Ennmouri, W. Zong, S. Chennuri, S. Cohen, S. Sundararaman, D. Waddington, and P. Chin, "Towards fast crash-consistent cluster checkpointing," in *Proceedings of the IEEE High Performance Extreme Computing Conference*, HPEC '22, pp. 1–8, 2022.

[23] Z. Wang, Z. Jia, S. Zheng, Z. Zhang, X. Fu, T. S. E. Ng, and Y. Wang, "Gemini: Fast failure recovery in distributed training with in-memory checkpoints," in *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, p. 364–381, 2023.

[24] T. Dey, K. Sato, B. Nicolae, J. Guo, J. Domke, W. Yu, F. Cappello, and K. Mohror, "Optimizing asynchronous multi-level checkpoint/restart configurations with machine learning," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops*, IPDPSW '20, pp. 1036–1043, 2020.

[25] A. Das, F. Mueller, C. Siegel, and A. Vishnu, "Desh: Deep learning for system health prediction of lead times to failure in hpc," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '18, p. 40–51, 2018.

[26] F. Wang, G. Wei, Q. Liu, J. Ou, x. wei, and H. Lv, "Boost neural networks by checkpoints," in *Advances in Neural Information Processing Systems* (M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), vol. 34, pp. 19719–19729, Curran Associates, Inc., 2021.

[27] E. M. B. Nagoudi, M. Abdul-Mageed, and H. Cavusoglu, "Growing together: Modeling human language learning with n-best multi-checkpoint machine translation," in *Proceedings of the Fourth Workshop on Neural Generation and Translation* (A. Birch, A. Finch, H. Hayashi, K. Heafield, M. Junczys-Dowmunt, I. Konstas, X. Li, G. Neubig, and Y. Oda, eds.), (Online), pp. 169–177, July 2020.

[28] H. Chen, S. M. Lundberg, and S.-I. Lee, "Checkpoint ensembles: Ensemble methods from a single training process," *ArXiv*, vol. abs/1710.03282, 2017.

[29] X. Xu, H. Liu, G. Tao, Z. Xuan, and X. Zhang, "Checkpointing and deterministic training for deep learning," in *Proceedings of the IEEE/ACM 1st International Conference on AI Engineering – Software Engineering for AI*, CAIN '22, pp. 65–76, 2022.

[30] E. Rojas, D. Pérez, J. C. Calhoun, L. B. Gomez, T. Jones, and E. Meneses, "Understanding soft error sensitivity of deep learning models and frameworks through checkpoint alteration," in *Proceedings of the 2021 IEEE International Conference on Cluster Computing*, CLUSTRE '21, pp. 492–503, 2021.

[31] E. Rojas, A. N. Kahira, E. Meneses, L. B. Gomez, and R. M. Badia, "A study of checkpointing in large scale training of deep neural networks," *ArXiv*, vol. abs/2012.00825, 2020.

[32] Y. Kim, K. Kim, Y. Cho, J. Kim, A. Khan, K. Kang, B. An, M. Cha, H. Kim, and Y. Kim, "Deepvm: Integrating spot and on-demand vms for cost-efficient deep learning clusters in the cloud," *ArXiv*, vol. abs/2403.05861, 2024.

Appendix A
Artifact Description

A. Abstract

In this document, we provide detailed instructions and steps that enable readers to: (i) install all programs, source codes, and dependencies related to the LP-based DeepVM; (ii) reproduce experiments in the same environment to extract data; (iii) generate figures from the data; (iv) create scenarios for customized experiments.

B. Description

1) Check-list (artifact meta information):

- **Program:**
  1) DeepVM (LP modeling) and its simulation;
  2) Software and example codes for each architecture for effectiveness experiments;
  3) Training example codes for overhead modeling validation experiments;
- **Dataset:** Information of available instances as of the experiment time, virtual instance information based on Amazon environment, workload (CIFAR-10);
- **Hardware:** Various x86 or x64 CPUs (for DeepVM and simulation), AWS instances (for training)
- **Experiment workflow:**
  1) DeepVM
     a) Run the software after installation.;
     b) If verification is desired, launch instances in the cloud as per recommended configuration;
     c) Measure the execution time and cloud cost to generate tables and figures;
  2) Simulation: Run the simulation after installation. Generate figures with the results;
  3) Overhead modeling validation: Run the training code after installation. Generate figures with the results;
- **Output:** Top n recommended configurations (DeepVM), Data and graphs showing the performance of the policy and the PW value (Simulation), Data and graphs representing the validation results
- **Experiment customization:** See below;
- **Publicly available?:** Yes.

2) How the artifact can be obtained: The Persistent ID of the artifact is:

https://doi.org/10.5281/zenodo.10672948

Furthermore, to ensure that readers can use the same version of the software for experiment reproduction, the code and necessary files along with instructions have been archived in Software Heritage:

https://archive.softwareheritage.org/swh:1:dir:561f421fa40bdc4fdb1f9f0c63b8f93e73886717;origin=https://github.com/lass-lab/DeepVM

Finally, all source materials can also be downloaded from the git repository associated with the Persistent ID:

https://github.com/lass-lab/DeepVM.git

While the content is identical across all platforms, be aware that the git repository may be updated over time.

The artifact consists of four main directories:

- `solution` : Includes the LP modeling and simulation programs, as well as the instance dataset and dataset generation program used.
- `arch_sw` : Contains example source codes for distributed learning per architecture to reproduce the effectiveness experiments of DeepVM in the paper.
- `validation` : Includes training source codes and usage instructions to reproduce the overhead modeling experiments in the paper.
- `results` : Contains all source codes to generate the graphs presented in the paper using the result files.

3) Hardware requirements: DeepVM can be run on most x86 or x64 CPUs. However, using too many instance data may increase execution time depending on CPU performance. For validation or conducting experiments based on DeepVM results, it can be performed in a cloud environment. It is necessary to be able to launch instances with GPUs as both spot and ondemand types. Each instance must be able to communicate with each other, and be launched without incurring additional network communication costs depending on the region. This paper conducted experiments in an AWS environment, a representative CSP, and readers wishing to replicate the experiment in the same environment can secure sufficient quotas by consulting AWS.

4) Software requirements:

- Git: A version control system storing all materials for simulation codes, execution codes, datasets, and extracting result files.
- Python 3 (version 3.10.6): Used for LP modeling and execution, data extraction, and graph generation. Also used in architecture-specific software for deep learning. Required libraries and their versions include: numpy(1.25.0), pandas(2.0.2), torch(2.0.1), torchvision(0.15.2), matplotlib(3.7.1), jupyter nodebook(6.5.4)
- pip (version 22.0.2): Used to download and manage libraries needed for running Python 3.
- OpenMPI (version 4.1.4): Used for data transmission in multi-level checkpointing when using software for Tiering architecture.
- GCC (version 11.4.0): Used to compile modules implemented in C++ in the Tiering architecture.

Readers who do not wish to use Git can download the artifact from Software Heritage as mentioned earlier. The required Python Packages are stored in `requirements.txt` in each directory.

5) Datasets: There are two types of datasets for modeling: virtual instance data and real instance data. Virtual instance data simulates AWS environment instances, and real instance data consists of information about instances available at the time of the experiment. Network saturation point data is embedded in `.py` files, and the scaling factor of actual instances is built into the DeepVM source code. All other data is stored in `.json` format. Detailed content of the

dataset is described in the experiment customization section. For training, the CIFAR-10 dataset was used. This dataset is automatically downloaded to the local computer at the start of training by PyTorch, and can also be downloaded using the provided shell file.

6) Installation: First, download the artifact (using Software Heritage or Git). If using Git, the following command can be used to copy the Git repository to the local computer: `git clone https://github.com/lass-lab/DeepVM.git`. If necessary, GCC, Python 3, pip, OpenMPI should be installed additionally. For packages mentioned in the software requirement section, they can be installed directly or using the `pip install -r requirements.txt` command.

7) Experiment Workflow:

- **DeepVM:** To run DeepVM, navigate to the `./solution` directory and execute the command: `./run.py`. Parameters including PW can be entered along with the command. If run without entering any parameters, default parameters are automatically entered: `pw=3, bf=2, ckpsize=0.5, MAX_LIMIT=256`. For detailed usage including parameter input, refer to `DeepVM/solution/README.md`.
- **Simulation:** To perform the simulation, navigate to the `./solution` directory and execute the command: `./simulation.py`. The usage of parameters is similar to DeepVM; refer to the same README file for guidance. When executed, a `.pkl` file containing the simulation results will be generated.
- **Validation (Effectiveness):** Readers who want to validate a specific configuration can do so by creating the same cluster on AWS. Distributed learning is performed using the software suitable for the recommended architecture. To prepare for distributed learning, copy all files in the respective directory to the user's home directory, then prepare with the following command: `bash ./initialization.sh`. After preparation, distributed learning can be started with the command: `bash ./run_experiments.sh`. For detailed execution methods, refer to the architecture-specific README in the `arch_sw` subdirectory.
- **Validation (Scaling Factor):** After creating the maximum number of identical instances, copy all files under `validation/scaling_factor` to the master's home directory. Then, after preparing similarly to the single anchor architecture software, perform distributed tasks with the command: `bash ./run_experiments`. This shell automatically records the time while increasing the number of training nodes. For detailed execution methods and examples, refer to the README in the `validation/scaling_factor` subdirectory.
- **Validation (Network Saturation):** First, create the desired cluster on AWS. While maintaining the number of GPU spot instances in the cluster, repeat the experiment by increasing the number of CPU spot instances. The method of preparation and execution for distributed learning is similar to the previously described tiering architecture software. For detailed execution methods and examples, refer to the README in the `validation/network_saturation` subdirectory.

8) Evaluation and Expected Result:

- **DeepVM:** The four policies mentioned in the experiment section of the paper will output their respective best configurations. However, for DeepVM policy, only the top 3 are outputted. Readers can launch clusters with these configurations to validate their effectiveness.
- **Simulation:** A `simul_results.pkl` file is created. This file contains the performance of configurations proposed by the four policies for each PW value in dictionary form. To generate graphs using this file, use the following Jupyter notebook file: `results/simulation/graph.ipynb`
- **Validation (Effectiveness):** A `.result` file is generated per experiment. This file can be opened with a text editor and contains the `makespan` time consumed in training. Readers can i) collect the makespan from various `.result` files, ii) collect the hourly price of each configuration, iii) enter these into the `result/real_world` directory's `fig8_data.xlsx` file. To generate graphs using this file, use the following notebook file: `results/real_world/graph.ipynb`
- **Validation (Scaling Factor):** Once training is completed, a file with all recorded data is created in the master node's home directory. This data includes the number of training nodes and the training time logged for each. Record this in the appropriate column of `results/fig5_data.xlsx`. To generate graphs using this file, use the notebook file mentioned in effectiveness.
- **Validation (Network saturation):** Each checkpointing yields `[serialization]` and `[sending]` logs. The number of logs per checkpointing can vary depending on the number of shards and remote nodes. The average time difference between serialization and sending pairs with the same sharding-checkpointing number is recorded as the time for that checkpointing. Average several checkpointing times to record the transmission time of that cluster in `results/fig6_data.xlsx`. To generate graphs using this file, use the notebook file mentioned in effectiveness.

9) Experiment customization: This section describes how to run user-defined scenarios using this artifact. Examples of user-defined scenarios include changing the type of virtual or real instances, changing prices, modifying performance figures, and altering the number of bottlenecks per network bandwidth. To change the type, price, or metrics of instances, modify the `.json` files in the directory. Then, DeepVM or simulation can be performed using the modified file. The values in the files consist of instance names, types, prices, parameters of regression equations, network performance, and availability, and detailed information can be found in the artifact's README. The method of creating figures using the result data is the same as in the original scenarios, but some scenarios may require additional code changes.