MediatorDNN: Contention Mitigation for Co-located DNN Inference Jobs

Seyed Morteza Nabavinejad
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA
snabavinejad@wpi.edu

Sherief Reda
School of Engineering
Brown University
Providence, RI
sherief_reda@brown.edu

Tian Guo
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA
tian@wpi.edu

***Abstract*—With the increase in computing power of cutting-edge hardware platforms, it is a common practice to run multiple jobs on a single machine for improved resource utilization and throughput. However, this leads to inevitable resource contention among co-located jobs, impacting their performance. The resource contention can worsen due to fluctuations in resource utilization of jobs caused by variations in their input workload. To tackle the co-location contention for DNN inference jobs, we propose *MediatorDNN*, which considers contention and resource utilization variation when co-locating DNN inference jobs. It profiles each DNN, monitors microarchitectural metrics such as memory bandwidth and cache access pattern, and high-level resource utilization like CPU utilization. Based on profiling results and leveraging Modern Portfolio Theory (MPT), *MediatorDNN* decides on the co-location of jobs. Experimental results with various DNNs on two hardware platforms show that *MediatorDNN* improves throughput by up to 108% (21% on average) compared to an approach only considering contention and ignoring resource utilization variation.**

***Index Terms*—deep neural networks, contention, co-location, throughput**

I. Introduction

Employment of Deep Neural Networks (DNNs) in different domains is on the increase. To quench the thirst of DNNs for high computing resources and achieve high performance, various hardware platforms such as ASICs [1], FPGA-based accelerators [2], CPU-based servers [3] and GPU-based accelerators [4] are employed. Since the resource requirement of DNNs might be less than the hardware resource capacity, it is a common approach to co-locate several DNNs on a single machine to improve resource utilization [5]–[7].

When multiple DNNs are co-located, performance contention arises due to competition for shared resources like memory bandwidth or cache. This contention impacts overall performance and challenges achieving maximum throughput. Prior research has addressed this issue across different hardware platforms [8]–[11]. Some passive approaches aim to mitigate the impact of contention after co-locating the applications and during runtime [10]–[13]. Another category proactively profiles the applications and co-locates them accordingly to avoid or mitigate contention [14], [15].

While these approaches consider the effect of contention on performance, they overlook the resource utilization variation of DNN inference jobs originating from their input. Batching is a widely used method to achieve high throughput for DNN inference [3], [16]–[18]. It allows reusing the DNN model parameters for several inputs and reduces the overhead of copying input data to the memory of hardware accelerators such as GPUs [16], [19]. The size of batches and their inter-arrival time varies over time, affecting resource utilization. Ignoring this variation can lead to over/under-utilization, resulting in degraded performance.

To fill this gap and consider the contention and the resource utilization variation simultaneously, we introduce *MediatorDNN*. It aims to minimize the low-level contention while avoiding the over/under-utilization of computing resources due to resource utilization variation. To this end, *MediatorDNN* profiles the jobs for a short period to monitor their microarchitectural level metrics such as cache access pattern, as well as high-level resource utilization, e.g., CPU utilization. Based on the profiling results, *MediatorDNN* decides on the co-location of DNN inference jobs. We make the following contributions in this paper:

- Employing several DNNs with varying architectures and two hardware platforms, we study the impact of contention on the performance of DNN inference jobs, as well as the impact of input workload on the resource utilization variation. We show that the impact of contention on the latency of jobs significantly depends on the computational complexity of DNNs. Furthermore, we show that depending on input workload, jobs can experience significant resource utilization variation.
- We introduce two metrics to quantify the contention and resource utilization variation of DNN inference jobs: 1) *ConScore* that quantifies the low-level contention between every two jobs based on microarchitectural metrics profiling, using Euclidean Distance (ED). 2) *RUVScore* to quantify the Resource Utilization Variation (RUV) proximity of each pair of jobs by profiling the resource utilization of each DNN for a short period.
- We design and implement the *MediatorDNN* approach to mitigate the impact of contention on the performance of co-located DNN inference jobs. *MediatorDNN* uses the ConScore and RUVScore to calculate the *ColoScore* and co-locate the jobs based on that. It also leverages

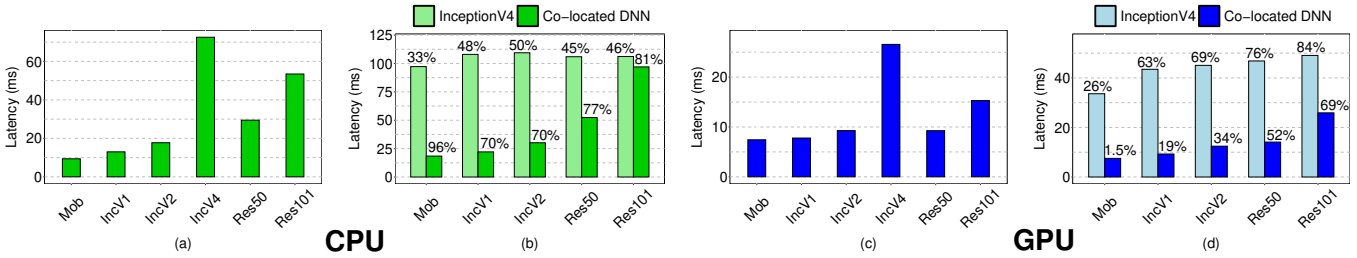


Fig. 1: The impact of co-location contention on inference latency of different DNNs on CPU and GPU. The numbers on top of the bars indicate the percentage of increase in latency of DNNs when co-located, compared with the case when they are running individually.

the Modern Portfolio Theory (MPT) to decrease the total RUV of co-located jobs, resulting in improved throughput.

We use several DNNs and datasets from different domains and conduct experiments on two hardware platforms (CPU and GPU) to evaluate *MediatorDNN*. Our experimental results show that *MediatorDNN* can improve the throughput by up to 108% (21% on average) compared to Horus [15] which considers the low-level contention but ignores the resource utilization variation.

The rest of the paper is organized as follows: In Section II, we motivate our approach by showing the impact of contention on the latency of different DNNs, as well as the impact of varying batch sizes and arrival time on resource utilization. Then, we introduce our proposed approach, *MediatorDNN*, in Section III and present the experimental results in Section IV. Related works are discussed in Section V, and the paper is concluded in Section VI.

II. Motivation

A. Co-location Contention

With the increase in computing power and memory capacity of hardware used in clusters and data centers, it is a common practice to co-locate several applications on a single machine to improve resource utilization and throughput. This section examines how co-location affects the performance of DNN inference jobs. We assess the performance contention resulting from various co-location combinations of six DNNs (MobileNetV1-1, Inception-V1, Inception-V2, Inception-V4, ResNetV2-50, ResNetV2-101) on two hardware platforms: a server with Intel Xeon CPUs and a Tesla P40 GPU. Initially, we run each DNN individually to measure their average inference latency (time needed to process one input) on both CPU and GPU (Fig 1 (a) and (c)). Next, we select Inception-V4 as a pivot and co-locate it with each of the other DNNs (e.g., Inception-V4 and MobileNetV1-1) to measure average latency again (Fig 1 (b) and (d)). This analysis aims to identify any significant differences in latency among different co-location combinations.

As expected, co-locating the Inception-V4 with another DNN results in increased latency in both CPU and GPU. However, the amount of latency increase varies from one DNN to another. The hardware also affects the amount of contention and latency increase. Co-location of Inception-V4 and MobileNetV1-1 (Mob) leads to 26% and 1.5% latency increase, respectively using GPU accelerator. However, the co-location of Inception-V4 with ResNetV2-101 (Res101) leads to a significant latency increase (84% for Inception-V4 and 69% for ResNetV2-101). The root of this difference is the varying performance contention raised in the presence of various combinations of DNNs co-located together. Finally, the amount of latency increase in the CPU differs from the GPU. We conclude that while all the co-located DNNs experience a level of contention, its intensity depends on which DNNs are co-located. Moreover, the hardware platform has a significant role in the contention.

TABLE I: Specification of the DNN models and their workloads (batch size and inter-arrival time distributions) used for RUV experiments.

DNN Name	Complexity (MFLOPs)	Batch Size Distribution		Inter-Arrival Time Distribution (ms)	
		Mean (μ)	SD (σ)	Mean (μ)	SD (σ)
Inception-V3	54.25	31.52	2.03	206.85	14.47
NASNet-Large	177.11	23.08	14.05	382.89	19.66
PNASNet-Mobile	10.06	13.23	14.82	195.02	13.93

B. Resource Utilization Variation

In DNN inference systems, requests usually arrive at different batch sizes and varying inter-arrival times [3], [20]. This varying input size and arrival time results in resource utilization fluctuation. This fluctuation directly affects the overall resource utilization of co-located jobs and can lead to over/under-utilization, and consequently, degraded performance or power efficiency. To study the impact of request fluctuation on resource utilization of DNN inference jobs, we deploy DNNs on the hardware platform one by one (no co-location) and feed them with an input workload with varying batch sizes and inter-arrival times. For each DNN, a different workload is generated using the workload generator from DeepRecSys [3]. The specifications of DNNs and their input workload are presented in Table I. The resource utilization results are shown in Fig. 2.

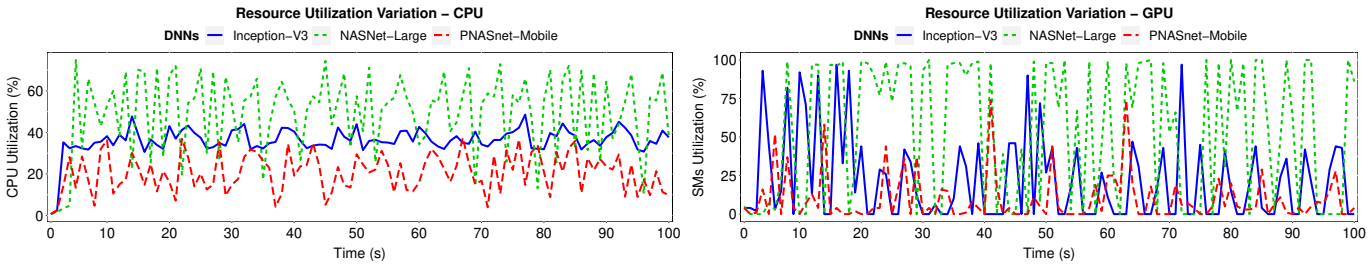


Fig. 2: Fluctuating workload leads to significant RUV for each DNN on both hardware platforms.

We see that the significance of fluctuation depends on the computational complexity of the DNN model, batch size and inter-arrival time distributions, and the hardware platform. For example, while the average batch size (31.52) in the Inception-V3 is larger than that of NASNet-Large (23.08), its maximum CPU utilization is lower due to its lower computational complexity. On the other hand, since the batch size in Inception-V3 has a low standard deviation (SD) of 2.03, the fluctuation in CPU utilization of Inception-V3 is lower than both NASNet-Large and PNASNet-Mobile with SD of 14.05 and 14.82, respectively. Furthermore, comparing the CPU and GPU results indicates that the significance of fluctuation is a function of the hardware platform.

The results reveal that the key to improving the performance of DNN co-location is considering the contention among co-located DNNs and RUV of each DNN, simultaneously. The contention can increase the latency of DNNs which results in degraded overall throughput. Moreover, RUV can lead to over-utilization which exacerbates the contention since it leads to competition for high-level resources (e.g., SMs), in addition to low-level ones (e.g., cache and memory bandwidth).

III. Methodology

In this section, we present the design and implementation of *MediatorDNN*, which aims to maximize the throughput of DNN co-location. We start with the problem statement and then describe the *MediatorDNN* in detail. The overall flow of our approach is depicted in Fig. 3.

A. Problem Statement

The problem we address in this work is as follows: We have a set of DNN inference jobs and we want to divide them into several groups and co-locate each group to maximize the overall throughput of jobs. DNN jobs belong to best-effort applications with no quality of service (QoS) requirement, i.e., no latency constraint [21]. The DNNs have various architectures and sizes, which means their computational complexity is different from one another. When co-locating jobs, they compete for hardware resources that affect their throughput. Moreover, the input workload of each job is in the form of a sequence of batches with different sizes and inter-arrival times. This fluctuating workload results in RUV in each job. When co-locating the jobs, the RUV can be another source of contention. In this problem, we can profile each job for a short period to collect microarchitectural level metrics, as well as monitor its RUV pattern.

B. Low-level Profiling: ConScore

To mitigate the impact of contention, the first step is to quantify its intensity between each two jobs. To this end, we propose gathering low-level metrics and estimating the amount of relative contention between each two jobs accordingly. The metrics that can be gathered depend on the hardware platform and the tool used for profiling. For the CPU, we use the Perf tool to instrument CPU performance counters and gather our desired microarchitectural metrics such as cache access. For the GPU, NVIDIA offers a profiling tool called nvprof [22]. The kernels launched by an application on the GPU are profiled by this tool to instrument various low-level metrics. After profiling each job for a short time, we have its profiling vector $\vec{pv}$. The size of this vector depends on the number of gathered metrics. Having the $\vec{pv}$ for all the jobs, we proceed to the next stage which is quantifying the level of contention between each pair of jobs. In this stage, we calculate the Euclidean Distance (ED) of each two profiling vectors. Denoting the set of DNN jobs with DS, and the number of low-level metrics profiled with n, we have:

$$ED_{i,j} = \sqrt{\sum_{k=1}^{n} (p\vec{v}_{i_k} - p\vec{v}_{j_k})^2} \quad , \quad \forall (i,j) \in DS \tag{1}$$

For each job, its ED with the rest of the jobs is standardized to obtain its ConScore with others. We use standardization as the ConScore and the RUVScore, which we discuss in the next section, have different scales and their sum is used for co-location. For standardization, the mean (μ) and standard deviation (σ) of the ED of a job with the rest of the jobs are calculated. Then, by subtracting the mean from the ED and dividing it by the standard deviation, the standardized value that is the ConScore is obtained as (2). For each job, its ConScore is calculated against all the other jobs.

$$\begin{aligned} \forall i \in DS, \forall j \neq i \in DS, \quad \mu_i &= \frac{\sum_j ED_{i,j}}{N-1}, \\ \sigma_i &= \sqrt{\frac{1}{N}\sum_j (ED_{i,j} - \mu_i)^2}, \\ ConScore_{i,j} &= \frac{ED_{i,j} - \mu_i}{\sigma_i} \end{aligned} \tag{2}$$

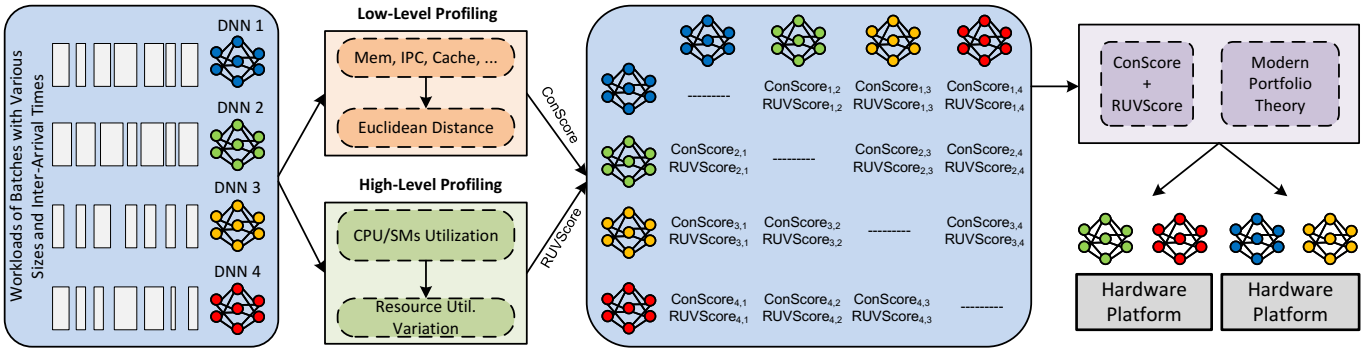


Fig. 3: Overall flow of *MediatorDNN*.

Algorithm 1 ConScore

Input: set of DNN jobs (DS), n (number of metrics to be profiled)

Profiling

for i in DS **do**
  $p\vec{v}_i$ = profile-low-level-metrics (i)
for $i \in DS$ **do**
  for $j \in DS$ **do**
    if $i \neq j$ **then**
      $ED_{i,j} = \sqrt{\sum_{k=1}^{n}(p\vec{v}_{i_k} - p\vec{v}_{j_k})^2}$
for $i \in DS$ **do**
  calculate μ_i, σ_i
for $i \in DS$ **do**
  for $j \in DS$ **do**
    if $i \neq j$ **then**
      $ConScore_{i,j} = \frac{ED_{i,j} - \mu_i}{\sigma_i}$

In this way of calculating the ConScore, two jobs with similar profiling vectors have low ConScore, while jobs with very diverse profiling vectors have high ConScore. It is preferred to co-locate the DNN jobs with high ConScore together, so the high access of one to low-level resources (e.g., cache) is mitigated by low access of the other one, and hence, the overall contention is reduced. The pseudo-code for calculating ConScore is shown in Algorithm 1.

C. High-level Profiling: RUVScore

As shown in Section II, due to fluctuation in the input workload of DNN inference jobs, variation in their resource utilization over time is inevitable. Fluctuation in resource usage of co-located jobs directly affects the total resource utilization of the system. Therefore, adopting a RUV-oblivious approach can result in either under-utilization and poor power efficiency, or over-utilization and poor performance. To address this challenge, we first profile a job with its workload for a short time and monitor its resource utilization (e.g., CPU utilization or SMs utilization). Then, we derive the normal distribution of its resource utilization to have the mean and SD. Having the resource utilization standard deviation (RUSD) of all the jobs, we are ready to calculate the RUVScore of each job to the rest of them, considering Modern Portfolio Theory. In the following, we introduce this theory and describe how to obtain RUVScore.

1) Mitigating RUV by Modern Portfolio Theory: Modern Portfolio Theory (MPT), a theory of finance introduced by Harry Markowitz [23], helps investors to select the assets of their portfolio considering the individual risk of each one such that the risk of the portfolio for a certain expected return is minimized. Using the normal distribution to model the return of assets, the risk of an asset is the standard deviation of the distribution. The portfolio effect used by MPT is as follows: the risk of the returns of a collection of assets in a portfolio is less than the sum of the risk of returns of each asset, separately. We use X_i to denote an asset, and μ_i and σ_i to indicate the mean and standard deviation (risk) of its return, respectively. Hence, for the portfolio Y, which is a collection of N assets, the portfolio effect can be formulated as follows [24]:

$$\mu_Y = \sum_{i=1}^{N} \mu_{X_i} \quad , \quad \sigma_Y = \sqrt{\sum_{i=1}^{N} \sigma_{X_i}^2} \leq \sum_{i=1}^{N} \sigma_{X_i} \tag{3}$$

In our study, we consider the DNN jobs submitted by users as assets and the computing capacity of the hardware platform as a portfolio. The return is the resource utilization mean and risk is the resource utilization standard deviation of each job. We aim to leverage the portfolio effect to improve the resource utilization of the hardware platform, as well as mitigate the RUV of jobs when co-locating them on it. The equation (3) assumes there is no correlation between resource utilization of jobs, which means their high-level resource utilization patterns should be independent. For instance, increasing the resource utilization of one job, should not increase the resource utilization of another. As the input workloads of the jobs are independent and uncorrelated, their resource utilization is uncorrelated as well, which means we can use (3).

The illustrative example in Fig. 4 shows the impact of portfolio effect on the total resource utilization of co-located DNNs. Assume that we have four DNN jobs that have the same mean resource utilization (30%), but different standard deviations (two of them 20%, two of them 2%), and we want to co-locate them in pairs. If we co-locate them as the left form, the sum of resource utilization of both hardware platforms would be more than when we co-locate them as the right form. In both co-locations, the sum of the mean of resource utilization is equal. However, the portfolio effect decreases the

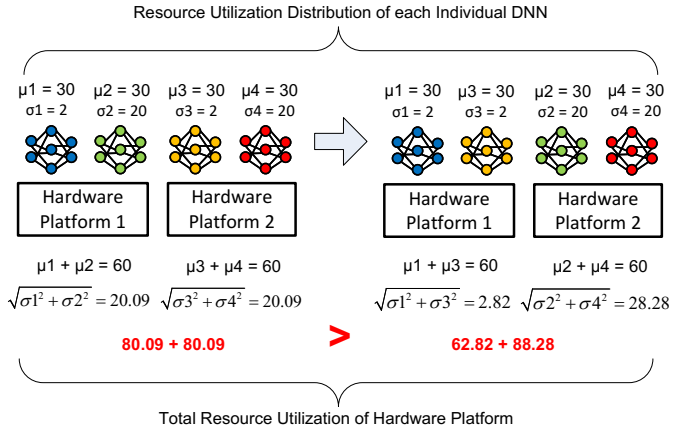


Fig. 4: Illustrative example that shows the impact of portfolio effect on the resource utilization of co-located DNNs. The numbers show the mean and standard deviation of each job. The left figure shows a co-location that does not consider the portfolio effect, while the right one shows a co-location that factors it in.

standard deviation of resource utilization from 40 (the sum of both platforms in left co-location) to 31 (the sum of both platforms in right co-location).

2) RUVScore Calculation: From MPT, we conclude that it is best to co-locate the jobs with high RUSD with each other, to reduce the total resource utilization. Therefore, if we sort the jobs by their RUSD, we prefer to co-locate the ones that are next to each other, i.e., the ones with the lowest distance between their SD. Hence, we calculate the RUVScore as follows: for each job i in the set of DNN jobs, calculate the difference between its RUSD and the RUSD of all the other jobs as (4). Then, calculate the mean and SD of all those RUSD differences. Finally, subtract each RUSD from their mean and divide by SD to achieve the RUVScore as (5). Similar to ConScore, RUVScore is calculated for each pair of jobs. The higher RUVScore means that the RUSD of the two jobs is closer to each other, and it is preferred to co-locate them together. Note that $\text{RUVScore}_{i,j}$ is not equal to $\text{RUVScore}_{j,i}$ and they should be calculated separately. The pseudo-code for calculating RUVScore is shown in Algorithm 2.

$$RUSDD_{i,j} = RUSD_i - RUSD_j \quad , \quad \forall (i,j) \in DS \tag{4}$$

$$\begin{aligned} \forall i \in DS, \forall j \neq i \in DS, \quad \mu_i &= \frac{\sum_j RUSDD_{i,j}}{N-1}, \\ \sigma_i &= \sqrt{\frac{1}{N}\sum_j (RUSDD_{i,j} - \mu_i)^2}, \\ RUVScore_{i,j} &= \frac{RUSDD_{i,j} - \mu_i}{\sigma_i} \end{aligned} \tag{5}$$

D. MediatorDNN Design

MediatorDNN aims to optimize resource utilization and reduce contention when co-locating jobs. To find a trade-off between contention and resource utilization, it considers the sum of ConScore and RUVScore, which we call ColoScore. *MediatorDNN* works as follows (shown in Algorithm 3): Initially, it sorts the jobs based on their RUSD in descending order. Then, it selects the one with the highest RUSD and adds it to the co-location group. After that, the ColoScore of this job with all the other jobs is calculated, and they are sorted in descending order by the resulting ColoScores. *MediatorDNN* selects the job with the highest ColoScore and examines it with the currently selected job to see if its total resource utilization is less than or equal to the resource capacity of the hardware platform. If so, this job is selected and added to the group. Otherwise, the job is skipped, and *MediatorDNN* probes the next one with the highest ColoScore. When the co-location group size is greater than one, *MediatorDNN* calculates the ColoScore of a job by all the jobs in the group, and not only the first one (the sum of ColoScore of a job with all the members of the group is considered and jobs are sorted accordingly). This process continues until all the jobs are checked or the hardware reaches its maximum capacity. Then, the current group is finalized and ready for co-location, and *MediatorDNN* moves to the next group. It repeats these steps until there is no job left and all of them are assigned to a co-location group.

When *MediatorDNN* calculates the total resource utilization of a co-location group to ensure it is less than the computing capacity of hardware, it takes into account the RUV of each job through portfolio effect, in addition to its average resource utilization:

$$TotalResUtilGroup = \sum_{i \in group} \mu_i + \sqrt{\sum_{i \in group} \sigma_i^2} \tag{6}$$

Algorithm 2 RUVScore

Input: set of DNN jobs (DS), resource utilization profiling results of each job ($RUProfile$)

Profiling

for $i \in DS$ **do**
  $RUSD_i$ = SD ($RUProfile_i$)
for $i \in DS$ **do**
  for $j \in DS$ **do**
    if $i \neq j$ **then**
      $RUSDD_{i,j} = RUSD_i - RUSD_j$
for $i \in DS$ **do**
  calculate μ_i, σ_i for $RUSDD_{i,j}$
for $i \in DS$ **do**
  for $j \in DS$ **do**
    if $i \neq j$ **then**
      $RUVScore_{i,j} = \frac{RUSDD_{i,j} - \mu_i}{\sigma_i}$

IV. Evaluation

A. Experimental Setup

Platform. We run experiments on two platforms to show the generality of our approach: 1) A dual-socket Xeon server that has two E5-2680 v4 Xeon chips, each with 28 cores running at 2.4 GHz, and 128 GB of DDR4 memory. 2) A Nvidia Tesla P40 GPU Accelerator that leverages Nvidia Pascal architecture

Algorithm 3 *MediatorDNN*

Input: set of DNN jobs (DS), RUVScore, ConScore
RD: Resource Demand (based on profiling)
HRC: Hardware Resource Capacity

1: **for** $i, j \in DS$ **do**
2: $ColoScore_{i,j} = RUVScore_{i,j} + ConScore_{i,j}$
3: groupID = 1
4: **while** DS is not empty **do**
5: DS.Sort (based on RUV in descending order)
6: Group(groupID) = []
7: Group.add(DS[0]) // job with highest RUV
8: DS.remove(0)
9: **while** TRUE **do**
10: **for** $i \in DS$ **do**
11: **for** $j \in Group$ **do**
12: $TotalColoScore_i += ColoScore_{i,j}$
13: DS.Sort(based on TotalColoScore in descending order)s
14: **for** $i \in DS$ **do**
15: **if** $Group.RD + RD_i \leq HRC$ **then**
16: Group.add(DS[i]), DS.remove(i), Go to line 9
17: No more job can be added to group
18: groupID = groupdID + 1
19: create new group
20: Break while loop in line 8 and go to line 3

and has 3840 CUDA cores. The total memory capacity of GPU is 24 GB GDDR5 memory.

Networks and Dataset. We use DNNs from different domains with varying sizes and computational complexity. From image classification, we employ 16 models with two datasets in our experiments. One is the widely used ImageNet dataset [25], and the other one is Caltech-256 [26]. For these DNNs, throughput is defined as the number of images processed per second (image/second). From the natural language processing (NLP), we employ a DNN for text classification [27], which we call TextClassif. For the input data of this DNN, we use Sentiment140 [28] and IMDB Reviews [29] datasets. For this DNN, the throughput is defined as the number of sentences processed per second. DeepVS [30] is another DNN we use in our experiments that targets video saliency prediction and the throughput is defined as the number of frames processed per second. The list of networks and datasets is presented in Table II. [1]

System Comparison. To evaluate the performance of *MediatorDNN*, we use two approaches for comparison:

Bin Packing (BP) co-locates the jobs based on available information on their average resource utilization and ignores both contention and RUV. It only aims at maximizing the resource utilization of hardware by selecting the jobs such that the sum of their average utilization would be as close as possible to the hardware capacity.

Horus [15] aims to improve the performance of DNNs through co-location and considers both contention and resource utilization. It considers the effect of low-level contention when co-locating the DNNs. It uses resource utilization prediction for co-location as well; however, it only relies on average resource utilization and does not take into account the variation in resource utilization of DNNs over time in its co-location scheme.

[1]Note that for TextClassif and DeepVS DNNs we could not obtain computational complexity as the tool we used does not support them.

TABLE II: List of DNNs and Datasets Used in the Experiments

DNN (Abbreviation)	Domain	Dataset (Size)	Computationl Complexity (Mega FLOPs)
Inception-V1 (Inc-V1) [31]	Image Classification	ImgaeNet & Caltech-256 (20000 Images)	13.22
Inception-V2 (Inc-V2) [32]			22.34
Inception-V3 (Inc-V3 [33]			54.25
Inception-V4 (Inc-V4) [34]			91.95
MobilenetV1-1 (MobV1-1) [35]			8.42
MobilenetV1-05 (MobV1-05) [35]			2.64
MobilenetV1-025 (MobV1-025) [35]			0.93
MobilenetV2-1 (MobV2-1) [36]			6.94
MobilenetV2-14 (MobV2-14) [36]			12.12
NASNET-A-Large (NAS-Large) [37]			177.12
NASNET-Mobile (NAS-Mob) [37]			10.51
PNASNET-Large (PNAS-Large) [38]			171.77
PNASNET-Mobile (PNAS-Mob) [38]			10.06
ResNet-V2-50 (ResV2-50) [39]			51.01
ResNet-V2-101 (ResV2-101) [39]			88.89
ResNet-V2-152 (ResV2-152) [39]			120.08
TextClassif (-) [27]	NLP	Sentiment140 IMDB	-
DeepVS (-) [30]	Video Saliency	LEDOV [30] DHF1K [40], [41]	-

TABLE III: Specification of Sample Jobs Used in the Experiments

			Input Workload	
Job #	DNN	Dataset	Avg. Arrival Time (ms)	Avg. Batch Size
1	Inc-V1	ImageNet	168	17
4	Inc-V4	ImageNet	299	39
8	MobV2-1	ImageNet	295	28
12	PNAS-Large	ImageNet	111	36
19	Inc-V3	CalTech	459	6
20	Inc-V4	CalTech	439	11
24	MobV2-1	CalTech	357	1
30	ResV2-50	CalTech	202	32
36	TextClassif	IMDB	64	512
40	DeePVS	LEDOV	299	4

Jobs and Workloads. There are 40 jobs in the experiments: 32 image classification jobs (the combination of 16 image classification DNNs with two respective datasets), plus 4 NLP jobs (combination of TextClassif with the two datasets, each of which repeated two times) and 4 video saliency jobs (combination of DeepVS DNN with the two datasets, each of which repeated two times). For each of these jobs, a workload is generated using the workload generator of DeepRecySys [3], which consists of a series of batches with different sizes and inter-arrival times. The specifications of sample jobs are shown in Table III. Also, detailed workload distribution is shown in Fig. 5 for five representative jobs that cover different batch sizes and inter-arrival times. We see significant batch size and inter-arrival time variation within each job and across different jobs.

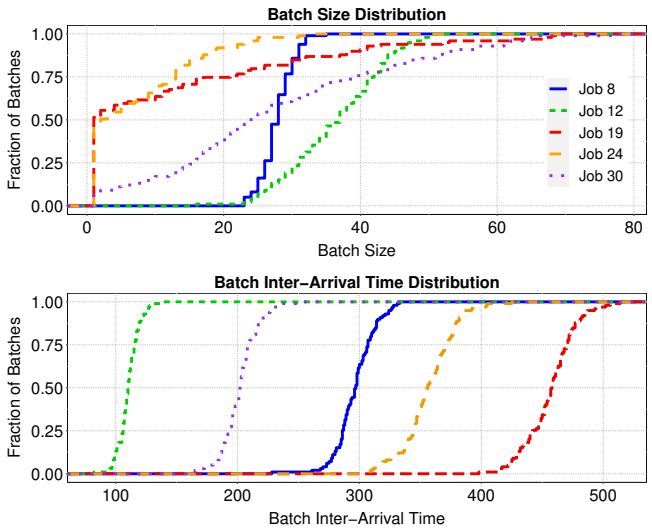


Fig. 5: Distribution of batch size and inter-arrival time of input workloads for representative jobs.

B. Profiling Results

As mentioned in Section III, first we profile jobs to identify the resource utilization patterns, as well as the access pattern of microarchitectural metrics. The resource utilization results are shown in Fig. 6a for the CPU server and Fig. 6b for the GPU accelerator. We have profiled each job with its input workload (presented in Table III for sample jobs) for 200 seconds. This period is enough to cover the input workload variation. We can see the RUV in the form of SD for each job, in addition to its mean resource utilization. The mean and SD of resource utilization vary from job to job, depending on the DNN and the workload distribution. While some jobs have very close mean resource utilization, there is a huge gap between their resource utilization SD. It clearly emphasizes the importance of RUV in the co-location process, as solely considering mean value for resource utilization can be misleading and result in poor performance. Moreover, the resource utilization pattern of the jobs is different on CPU and GPU (for the same workloads) which emphasizes the dependency of RUV on the hardware platform. In general, we see higher RUV in GPU than CPU, due to the copying time of data from host to GPU, which results in zero SMs utilization during that time.

Moreover, the detailed resource utilization of profiling results (the cumulative distribution) for sample jobs on CPU is depicted in Fig. 7. It can clearly show how the RUV of the jobs varies over time, as well as the difference between the amount of the resource utilization of various jobs. Furthermore, the low-level profiling results for some representative jobs are shown in Table IV. We see a significant difference between the access pattern of jobs to low-level metrics (up to four orders of magnitude), which depends on DNN and workload. These results further emphasize the importance of considering low-level metrics for mitigating contention.

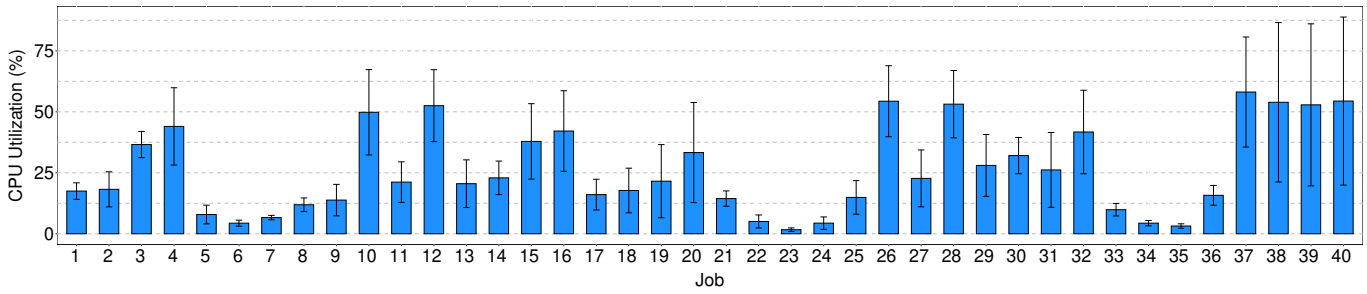
C. Throughput Comparison

In this section, we study the throughput improvement of *MediatorDNN* compared with other approaches. Fig. 8a and Fig. 8b show the throughput improvement of *MediatorDNN* compared with Horus and BP for each job on CPU and GPU, respectively. On CPU, *MediatorDNN* improves the throughput by up to 30% and 55% (10% and 22% on average) compared with Horus and BP, respectively. For GPU, the average improvement is 21% and 31% and the maximum is 108% and 79%. Since Horus considers the contention among the jobs, its performance is better than BP and can achieve higher throughput than that. However, since it ignores the RUV of jobs, its throughput is lower than *MediatorDNN*. Among the approaches, BP has the poorest performance as it ignores both contention and RUV and only considers the resource capacity of the hardware and average resource utilization of jobs.

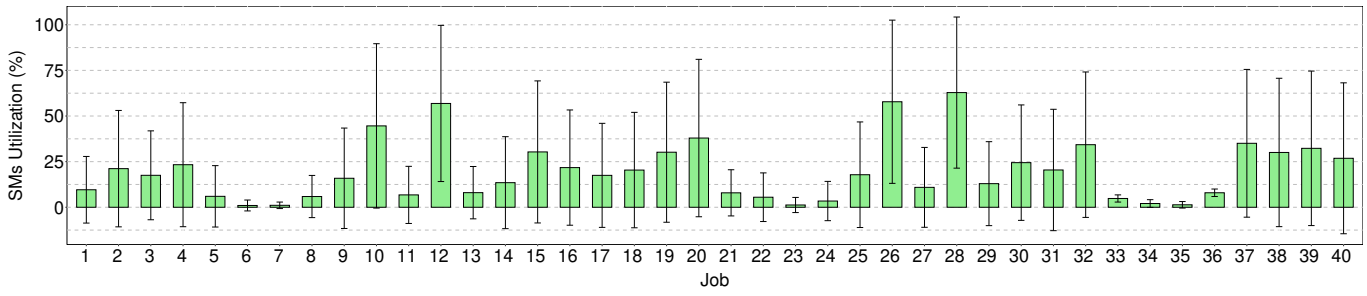
The amount of throughput improvement of *MediatorDNN* varies from job to job. The jobs have different request arrival patterns and computing requirements, so their throughput improvement is different. Moreover, the way the jobs are grouped differs between approaches. One job in Horus or BP might be grouped by other jobs with low computational complexity, while the same job might be grouped by jobs with high computational complexity in *MediatorDNN*. Therefore, the ultimate goal of *MediatorDNN* is to increase the total throughput of entire jobs instead of focusing on each job, separately.

We can also observe that the amount of throughput improvement depends on the hardware platform. It originates from different resource utilization patterns of jobs on other platforms, the computing power of the platforms, and the co-location of jobs for each platform. For the CPU server, as the memory capacity is very high, the total CPU utilization determines the number of jobs that can be co-located (their total CPU utilization should be less than the capacity of the server). For the GPU on the other hand, in addition to SMs utilization, the memory capacity should be considered as well. To estimate the memory requirement of different DNNs, the approach introduced in Horus [15] can be used. When co-locating the jobs on the GPU, both their total SMs utilization and memory requirement should be less than the total capacity of the hardware.

To show the impact of considering RUV on the performance of different approaches, we depict the resource demand pattern of the jobs for one of the co-location groups for each approach. Fig. 9 shows the resource demand pattern for each job in the group that is obtained in the profiling step. The green line shows the sum of the resource demand of all the jobs over time, and the red line shows the maximum resource capacity of the server. The green line crossing the red line means that the resource demand of the jobs in the group is more than the total capacity of hardware, and hence, the jobs would compete for the resources. As mentioned before, both Horus and BP ignore RUV when co-locating the DNNs and only consider the average resource demand of jobs from the



(a) CPU server



(b) GPU accelerator

Fig. 6: Mean and SD of resource utilization of jobs profiled by their input workloads for each platform.

TABLE IV: Low-level Profiling Result of DNNs on CPU server for Selected Jobs

			Microarchitectural Metrics							
Job #	DNN	Dataset	cache-references	L1-dcache-loads	L1-dcache-stores	LLC-load	LLC-stores	dTLB-loads	dTLB-stores	iTLB-loads
1	Inc-V1	ImageNet	7.1509e+6	1.9167e+8	1.7818e+7	1.5885e+6	7.3889e+5	3.8333e+7	7.1291e+6	1.7095e+4
12	PNAS-Large	ImageNet	1.0503e+8	3.1079e+9	8.5082e+8	2.6382e+7	8.9077e+6	9.1602e+8	4.5014e+8	1.4249e+4
20	Inc-V4	CalTech	3.1681e+7	1.1991e+9	7.4299e+7	8.1900e+6	2.1976e+6	3.5313e+8	3.9290e+7	6.3411e+3
30	ResV2-50	CalTech	1.2147e+7	4.0935e+8	2.7103e+7	3.2720e+6	1.0126e+6	1.0635e+8	1.2432e+7	3.7221e+3
36	TextClassif	IMDB	5.4490e+4	1.6658e+6	3.4124e+5	1.2256e+4	6.3303e+3	6.2752e+5	1.5717e+5	2.6653e+2
40	DeepVS	LEDOV	9.7234e+7	3.0751e+9	2.3128e+8	3.2953e+7	6.0958e+6	6.1472e+8	9.2679e+7	3.6901e+4

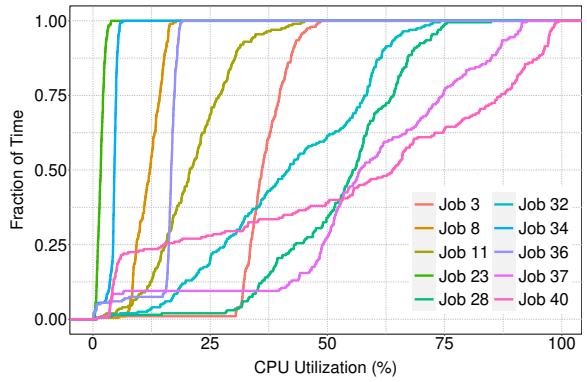
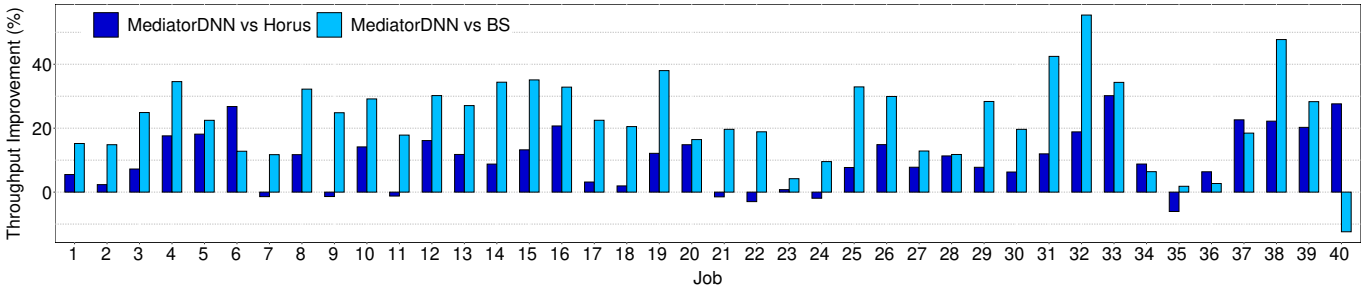


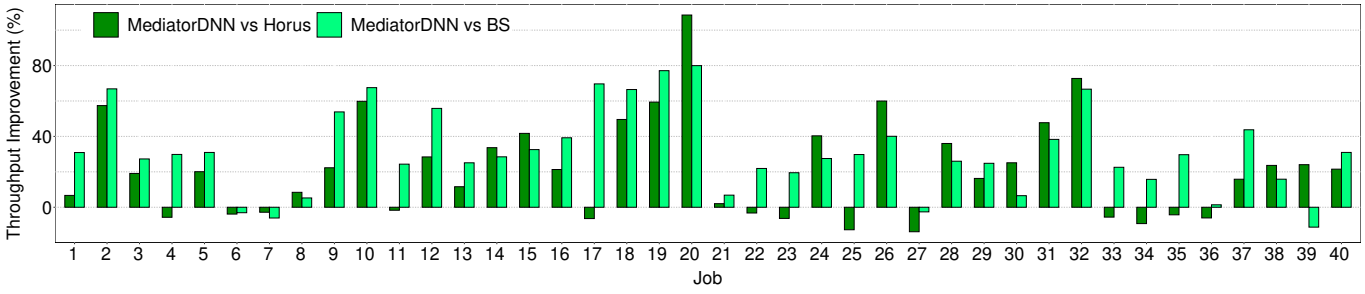
Fig. 7: Distribution of CPU utilization profiling results of selected jobs.

profiling step. Therefore, the total resource demand of the jobs they have selected for co-location is frequently higher than the computing capacity of the hardware (Fig 9(b)(c)). Therefore, the jobs suffer from resource shortage and yield lower throughput. On the other hand, considering RUV and leveraging MPT to mitigate its effect helps *MediatorDNN* to co-locate the jobs more efficiently compared with Horus and BP. We see in Fig. 9(a) that the number of times the sum of the resource demand of jobs exceeds the maximum capacity is lower than Horus and BP. Consequently, the jobs would achieve higher throughput.

As *MediatorDNN* utilizes more resources than Horus and BP to serve the jobs, it increases the power consumption. Therefore, to have a fair comparison, we compare the power efficiency of *MediatorDNN* with other approaches. We define power efficiency as the amount of throughput per power for each job. Since it is not possible to measure the power consumption of jobs in each co-location group separately, we divide the power consumption by the number of jobs in each group to estimate the power consumption per job in that group. Having the power consumption and throughput, we calculate the power efficiency for each job. The average and maximum power efficiency improvement of *MediatorDNN* is shown in Table V. These results indicate that while *MediatorDNN* resource usage is higher than other approaches, its



(a) CPU server



(b) GPU accelerator

Fig. 8: Throughput improvement of *MediatorDNN* compared with other approaches. The negative value indicates lower throughput in *MediatorDNN* compared with other approaches. For most of the jobs, *MediatorDNN* improves the throughput, and the amount of improvement is more significant in BP compared with Horus.

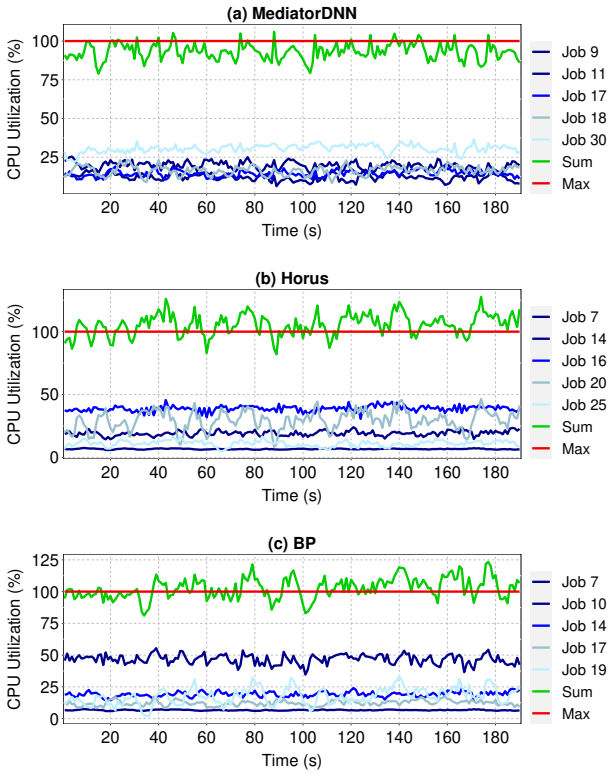


Fig. 9: Resource utilization pattern of jobs in a group for each approach.

TABLE V: Comparing the Power Efficiency Improvement of *MediatorDNN* with Horus and BP

	CPU		GPU	
	Average(%)	Max (%)	Average(%)	Max (%)
MediatorDNN vs Horus	4.74	105.68	8.05	276.98
MediatorDNN vs BP	17.06	244.09	7.42	100.80

power consumption increase is compensated with significant throughput improvement.

V. Related Work

A large body of research has focused on challenges and opportunities of co-location of DNNs [14], [42], [43]. PERSEUS [44] and Jain *et. al.* [45] studied the impact of co-location on the performance, cost, and latency of co-located DNN models. They showed that while co-location can help to improve the throughput and resource utilization, it degrades latency. VELTAIR [7] and Xu et al. [46] develop contention models for co-located DNN jobs on CPUs and GPUs using linear and random forest regression.

Baymax [12] and C-Laius [47] try to mitigate the impact of co-location on the latency of interactive jobs that share the resources with throughput-oriented jobs. They aim to maximize the latter's throughput while meeting the former's latency by reallocating time slots [12] or computing resources [47]. COLTI [48] targets the high-bandwidth memory (HBM) co-location in GPU accelerators to improve the throughput and resource utilization of DNN inference and training jobs.

SODA [5] targets clusters that host GPU accelerators for deep neural network (DNN) tasks alongside traditional CPU jobs. it uses a passive approach to eliminate the contention among jobs in real-time. DeepRecSys [3], Clockwork [49], and Nexus [50] consider the fluctuating batch sizes, along with different inter-arrival time for batches. However, they assume the models are already co-located. Therefore, they focus on resource allocation techniques to improve the performance of co-located jobs. unlike *MediatorDNN*, these approaches do not offer any mechanism for methodically selecting the DNNs for co-location in the first place. They can only be applied after the co-location has happened to reduce the contention. *MediatorDNN* approach is complementary to them and they can be used together.

iGniter [6] is a proactive GPU resource provisioning framework for DNNs in the cloud. It uses a DNN contention performance model to capture performance contention and a cost-efficient GPU resource provisioning strategy. This strategy optimizes GPU resource allocation and uses batching based on the contention model. iGniter ensures predictable performance for DNN workloads while minimizing expenses. It relies on its adaptive batching and resource allocation to mitigate the contention and does not consider the resource utilization fluctuation originating from pre-determined batch sizes in the input workload. Horus [15] and Themis [14] are two approaches that co-locate the DNNs considering the slowdown they will experience from co-location, and try to mitigate the impact of contention on performance. However, similar to iGniter, none of them takes into account the fluctuating workloads and resulting RUV of DNN jobs as our approach does.

VI. Conclusion

In this paper, we introduced the *MediatorDNN* to enhance the throughput of co-located DNN jobs. Our observations underscore that the contention between co-located jobs directly affects their throughput. Moreover, the fluctuating input workload of jobs contributes to resource utilization variation. Therefore, only relying on average resource utilization is insufficient for co-location decisions. *MediatorDNN* considers this variation, along with the access pattern of jobs to low-level resources of the hardware platform, when co-locating them. Leveraging MPT to mitigate the impact of resource utilization variation on total resource utilization helps *MediatorDNN* achieve higher throughput and power efficiency than previous approaches. The experimental results show that *MediatorDNN* can improve the throughput by 31% on average compared with the Bin Packing approach that does not consider contention and RUV, and 21% on average compared to the Horus approach, which considers the average resource demand of DNN jobs and contention among them but ignores RUV.

Acknowledgment

This work was supported in part by NSF Grants #2105564 and #2236987, and a VMWare grant.

References

[1] S. D. Manasi, S. Banerjee, A. Davare, A. A. Sorokin, S. M. Burns, D. A. Kirkpatrick, and S. S. Sapatnekar, "Reusing gemm hardware for efficient execution of depthwise separable convolution on asic-based dnn accelerators," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023, pp. 475–482.

[2] W. Zhao, Q. Dang, T. Xia, J. Zhang, N. Zheng, and P. Ren, "Optimizing fpga-based dnn accelerator with shared exponential floating-point format," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.

[3] U. Gupta, S. Hsia, V. Saraph, X. Wang, B. Reagen, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 982–995.

[4] D. Luo, T. Yu, Y. Wu, H. Wu, T. Wang, and W. Zhang, "Split: Qos-aware dnn inference on shared gpu via evenly-sized model splitting," in *Proceedings of the 52nd International Conference on Parallel Processing*, 2023, pp. 605–614.

[5] H. Zhao, W. Cui, Q. Chen, J. Leng, D. Zeng, and M. Guo, "Improving cluster utilization through adaptive resource management for dnn and cpu jobs co-location," *IEEE Transactions on Computers*, 2023.

[6] F. Xu, J. Xu, J. Chen, L. Chen, R. Shang, Z. Zhou, and F. Liu, "igniter: Interference-aware gpu resource provisioning for predictable dnn inference in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 3, pp. 812–827, 2022.

[7] Z. Liu, J. Leng, Z. Zhang, Q. Chen, C. Li, and M. Guo, "Veltair: towards high-performance multi-tenant deep learning services via adaptive compilation and scheduling," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 388–401.

[8] U. Gupta, C.-J. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, M. Hempstead, B. Jia, H.-H. S. Lee, A. Malevich, D. Mudigere, M. Smelyanskiy, L. Xiong, and X. Zhang, "The architectural implications of facebook's dnn-based personalized recommendation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 488–501.

[9] T. Patel and D. Tiwari, "Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 193–206.

[10] G. Chen, Y. Zhao, X. Shen, and H. Zhou, "Effisha: A software framework for enabling effficient preemptive scheduling of gpu," in *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2017, pp. 3–16.

[11] W. Zhang, Q. Chen, N. Zheng, W. Cui, K. Fu, and M. Guo, "Towards qos-awareness and improved utilization of spatial multitasking gpus," *IEEE Transactions on Computers*, 2021.

[12] Q. Chen, H. Yang, J. Mars, and L. Tang, "Baymax: Qos awareness and increased utilization for non-preemptive accelerators in warehouse scale computers," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 681–696, 2016.

[13] F. Guo, Y. Li, J. C. Lui, and Y. Xu, "Dcuda: Dynamic gpu schedulingith live migration support," in *Proceedings of the ACM Symposium on Cloud Computing*, 2019, pp. 114–125.

[14] M. Wei, W. Zhao, Q. Chen, H. Dai, J. Leng, C. Li, W. Zheng, and M. Guo, "Predicting and reining in application-level slowdown on spatial multitasking gpus," *Journal of Parallel and Distributed Computing*, vol. 141, pp. 99–114, 2020.

[15] G. Yeung, D. Borowiec, R. Yang, A. Friday, R. Harper, and P. Garraghan, "Horus: Interference-aware and prediction-based scheduling in deep learning systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 88–100, 2022.

[16] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 613–627.

[17] S. Zhang, W. Li, C. Wang, Z. Tari, and A. Y. Zomaya, "Dybatch: Efficient batching and fair scheduling for deep learning inference on time-sharing devices," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 609–618.

[18] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "Batchsizer: Power-performance trade-off for dnn inference," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 819–824.

[19] Y. Shen, M. Ferdman, and P. Milder, "Escher: A cnn accelerator with flexible buffering to minimize off-chip transfer," in *IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 93–100.

[20] J. Zhang, S. Elnikety, S. Zarar, A. Gupta, and S. Garg, "Model-switching: Dealing with fluctuating workloads in machine-learning-as-a-service systems," in *12th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, 2020.

[21] P. Pang, Q. Chen, D. Zeng, C. Li, J. Leng, W. Zheng, and M. Guo, "Sturgeon: Preference-aware co-location for improving utilization of power constrained computers," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 718–727.

[22] Nvidia, "Cuda profiler," https://docs.nvidia.com/cuda/profiler-users-guide/index.html, 2022, accessed: April 25, 2022.

[23] H. M. Markowitz, *Portfolio selection: efficient diversification of investments*. Yale university press, 1968, vol. 16.

[24] I. Hwang and M. Pedram, "Portfolio theory-based resource assignment in a cloud computing system," in *IEEE International Conference on Cloud Computing (CLOUD)*, 2012, pp. 582–589.

[25] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[26] G. Griffin, A. Holub, and P. Perona, "The caltech-256: Caltech technical report," *vol*, vol. 7694, p. 3, 2007.

[27] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014. [Online]. Available: http://arxiv.org/abs/1408.5882

[28] "Sentiment140," http://help.sentiment140.com/, 2022, accessed: April 25, 2022.

[29] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015

[30] L. Jiang, M. Xu, T. Liu, M. Qiao, and Z. Wang, "Deepvs: A deep learning based video saliency prediction approach," in *Proceedings of the european conference on computer vision (ECCV)*, 2018, pp. 602–617.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[33] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[34] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[35] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[36] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

[37] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8697–8710.

[38] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19–34.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision (ECCV)*. Springer, 2016, pp. 630–645.

[40] W. Wang, J. Shen, F. Guo, M.-M. Cheng, and A. Borji, "Revisiting video saliency: A large-scale benchmark and a new model," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[41] W. Wang, J. Shen, J. Xie, M. Cheng, H. Ling, and A. Borji, "Revisiting video saliency prediction in the deep learning era," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[42] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant {GPU} clusters for {DNN} training workloads," in *USENIX Annual Technical Conference (ATC'19)*, 2019, pp. 947–960.

[43] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 220–233.

[44] M. LeMay, S. Li, and T. Guo, "Perseus: Characterizing performance and cost of multi-tenant serving for cnn models," in *2020 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2020, pp. 66–72.

[45] P. Jain, X. Mo, A. Jain, H. Subbaraj, R. S. Durrani, A. Tumanov, J. Gonzalez, and I. Stoica, "Dynamic space-time scheduling for gpu inference," *arXiv preprint arXiv:1901.00041*, 2018.

[46] X. Xu, N. Zhang, M. Cui, M. He, and R. Surana, "Characterization and prediction of performance interference on mediated passthrough {GPUs} for interference-aware scheduler," in *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.

[47] W. Zhang, Q. Chen, N. Zheng, W. Cui, K. Fu, and M. Guo, "Toward qos-awareness and improved utilization of spatial multitasking gpus," *IEEE Transactions on Computers*, vol. 71, no. 4, pp. 866–879, 2022.

[48] J. Mobin, A. Maurya, and M. M. Rafique, "Colti: Towards concurrent and co-located dnn training and inference," in *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, 2023, pp. 309–310.

[49] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving {DNNs} like clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 443–462.

[50] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: A gpu cluster engine for accelerating dnn-based video analysis," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 322–337.