Kube-Knots: Resource Harvesting through Dynamic Container Orchestration in GPU-based Datacenters

Prashanth Thinakaran[1], Jashwant Raj Gunasekaran[1], Bikash Sharma[2],
Mahmut Taylan Kandemir[1] and Chita R. Das[1]
[1]*The Pennsylvania State University* [2]*Facebook, Inc.*

***Abstract*—Compute heterogeneity is increasingly gaining prominence in modern datacenters due to the addition of accelerators like GPUs and FPGAs. We observe that datacenter schedulers are agnostic of these emerging accelerators, especially their resource utilization footprints, and thus, not well equipped to dynamically provision them based on the application needs. We observe that the state-of-the-art datacenter schedulers fail to provide fine-grained resource guarantees for latency-sensitive tasks that are GPU-bound. Specifically for GPUs, this results in resource fragmentation and interference leading to poor utilization of allocated GPU resources. Furthermore, GPUs exhibit highly linear energy efficiency with respect to utilization and hence proactive management of these resources is essential to keep the operational costs low while ensuring the end-to-end Quality of Service (QoS) in case of user-facing queries.**

Towards addressing the GPU orchestration problem, we build *Knots*, a GPU-aware resource orchestration layer and integrate it with the *Kubernetes* container orchestrator to build *Kube-Knots*. *Kube-Knots* can dynamically harvest spare compute cycles through dynamic container orchestration enabling co-location of latency-critical and batch workloads together while improving the overall resource utilization. We design and evaluate two GPU-based scheduling techniques to schedule datacenter-scale workloads through *Kube-Knots* on a ten node GPU cluster. Our proposed Correlation Based Prediction (CBP) and Peak Prediction (PP) schemes together improves both average and 99^{th} percentile cluster-wide GPU utilization by up to 80% in case of HPC workloads. In addition, CBP+PP improves the average job completion times (JCT) of deep learning workloads by up to 36% when compared to state-of-the-art schedulers. This leads to 33% cluster-wide energy savings on an average for three different workloads compared to state-of-the-art GPU-agnostic schedulers. Further, the proposed PP scheduler guarantees the end-to-end QoS for latency-critical queries by reducing QoS violations by up to 53% when compared to state-of-the-art GPU schedulers.

I. Introduction

Modern data centers are being provisioned with compute accelerators such as GPUs and FPGAs to catch up with the workload performance demands and reduce the Total Cost of Ownership (TCO) [1]–[3]. By 2021, traffic within hyperscale datacenters is expected to quadruple with 94% of workloads moving into the cloud according to Cisco's global cloud index [4]. Based on the nature of the application, it can either benefit from the high throughput of GPUs or the low latency present in FPGAs, TPUs [5], [6], and custom ASICS [7]–[11]. This trend is also evident as public cloud service providers like Amazon [12] and Microsoft [13] have started offering GPU and FPGA-based infrastructure services.

In recent years, GPUs in particular have gained prominence due to the increasing computational demands of Deep Learning workloads. This includes both user-facing inference queries and batch-style model training of deep neural networks. With the increase in such workloads [4], [14], [15], public clouds have provisioned GPU resources at the scale of thousands of nodes in datacenters. Since GPUs are relatively new to the cloud stack, support for efficient GPU management lacks, as state-of-the-art cluster resource orchestrators [16], [17] treat GPUs only as a specific resource constraint while ignoring its unique characteristics and application properties.

In this paper, we address the issues at the cluster-level resource management layer. We notice that the cluster-level orchestration policies are heavily tuned for CPU-based clusters treating GPUs as a constraint for admission control. They do not harness the GPU's capability in full, leading to spare GPU cycles and poor utilization due to resource fragmentation. This is typically due to static provisioning of GPU resources. GPU containers earmark and hog the resources such as GPU memory to ensure crash-free executions of containers. In order to curb this, we propose *Kube-Knots* which dynamically harvests the spare compute cycles by resizing the containers. Further, we also show how GPU-specific management policies could guarantee the QoS of latency-sensitive queries while improving the cluster-wide GPU utilization leading to energy savings.

State-of-the-art resource orchestrators such as Kubernetes [16] and Mesos [17] perform GPU utilization-agnostic uniform scheduling, which statically assigns the GPU resources to the applications. The scheduled containers/pods[1] access the GPUs via PCIe pass-through, which lets the guest applications to have complete access to GPU resource bypassing the hypervisor. Hence, none of the hypervisor-level resource management policies such as fairness, utilization, etc., could be incorporated across GPUs where multiple tenants could share the same GPU resource in case of public datacenter. Further, Kubernetes supports dynamic orchestration by performing node affinity, pod affinity, and pod preemption in the case of CPUs. However, with respect to GPUs, pods have exclusive access to the device until completion and they cannot be preempted by the hypervisor. Based on this observation, we identify three specific problems pertaining to GPU-specific resource management.

[1]We use the terms (Google's) pod and container interchangeably.

978-1-7281-4734-5/19/$31.00

• **Context Switches in GPUs -** CPUs seamlessly context-switch across applications while ensuring SLOs (Service Level Objectives)/fairness. Modern CPU caches are virtually-indexed physically-tagged (VIPT), whereas GPUs are usually VIVT. Therefore, GPU caches are flushed after every preemption/context-switch. Moreover, GPU contexts are orders of magnitude larger than CPUs, leading to higher context-switch overheads. Although GPU containers can be made preemptible by time-sharing, preemption comes with high overheads which violates the SLO of latency-sensitive applications.
• **Utilization agnostic GPU scheduling -** Today's datacenter resource orchestrators treat GPU's as a hardware constraint and are agnostic of GPU utilization metrics. The QoS (Quality of Service) of latency sensitive queries are difficult to guarantee without knowing the system state due to performance interference [18]–[20], especially in the case of multi-tenancy.
• **Energy proportionality -** GPUs have linear performance per watt scaling, which implies that the maximum energy efficiency can be achieved only when GPUs are 100% utilized, unlike CPUs [21], as seen in Figure 1. It is crucial for a scheduler to fully pack and utilize the GPUs. While the CPUs peak efficiency is at 60-80% and pushing them beyond yield marginal returns due to hyper-threading issues.

To overcome these limitations, we propose *Kube-Knots*,[2] which enables GPU utilization-aware orchestration along with QoS-aware container scheduling policies. Towards this end, we design a runtime system *Knots* that aggregates real-time GPU cluster utilization metrics [22] which is leveraged by our proposed Peak Prediction (PP) scheduler. PP scheduler performs safe co-locations through GPU spare cycle harvesting by dynamically resizing the containers. This is in contrast to the state-of-the-art GPU scheduling techniques that focus on only one type of application such as Distributed Deep Neural Network training (DDNN) [14], [23]–[27]. In addition, *Kube-Knots* performs QoS-aware container co-locations on GPUs at the cluster-level without a priori knowledge of incoming applications. This is in contrast to prior node-level techniques [28]–[31] which require prior profiling of applications. We make the following contributions:

1) We build ***Knots***, a runtime system which aggregates the node-level GPU resource utilization metrics. This information is leveraged by the Kubernetes at the cluster-level to monitor the real-time GPU utilization at every node.
2) We design datacenter representative GPU cluster workloads to evaluate our proposed schedulers on a **real-system with multi-node GPU** cluster orchestrated by *Kube-Knots*.
3) We build two schedulers on top of Resource Agnostic **(Res-Ag)** orchestration namely, Correlation Based Prediction **(CBP)**, and Peak Prediction **(PP)** that leverages cluster-wide GPU utilization metrics aggregated from *Kube-Knots*.
4) We compare the **CBP+PP** based scheduler along with the state-of-the-art schedulers to schedule DNN workloads and show that utilization-aware application resizing and SLO-aware scheduling leads to improved JCTs of DNN tasks.

In summary, our proposed CBP and PP schedulers leverages the real-time GPU usage correlation metrics through spare cycle harvesting to perform safe pod co-locations ensuring crash-free **dynamic container resizing** on GPUs. CBP along with PP scheduler can guarantee the end-to-end QoS of latency critical queries by predicting the incoming load through ARIMA [32] based time-series forecasting for GPU-aware scheduling.

II. Background and Motivation

Traditional resource schedulers for CPU-based datacenters improve the overall utilization by virtualization. However, General Purpose GPUs (GPGPUs)[3] pose unique challenges in virtualization. Hence, public cloud offerings like AWS do not support virtualization of GPGPUs and FPGAs. But in the case of private datacenters, to improve the GPGPU utilization, time-sharing of the GPU compute cores (SMs) and space-sharing of the memory is enabled. In this section, we discuss the properties unique to GPUs and potential implications of sharing and virtualization leading to QoS violations.

A. Utilization vs Performance per Watt

The processing elements of CPUs are designed to operate for an average case load at peak energy efficiency [21] in terms of performance per watt (PPW). As seen in Figure 1 the energy efficiency varies at different load scenarios for CPUs and GPUs. The newer generation CPUs are much more energy proportional when compared to older architectures. For the CPU-bound queries, we observe that the peak energy efficiency at around 60% to 70% core utilization, where the point of peak efficiency is no longer at peak utilization for CPUs.

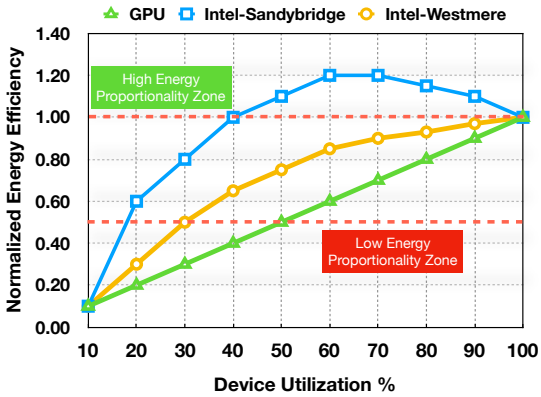
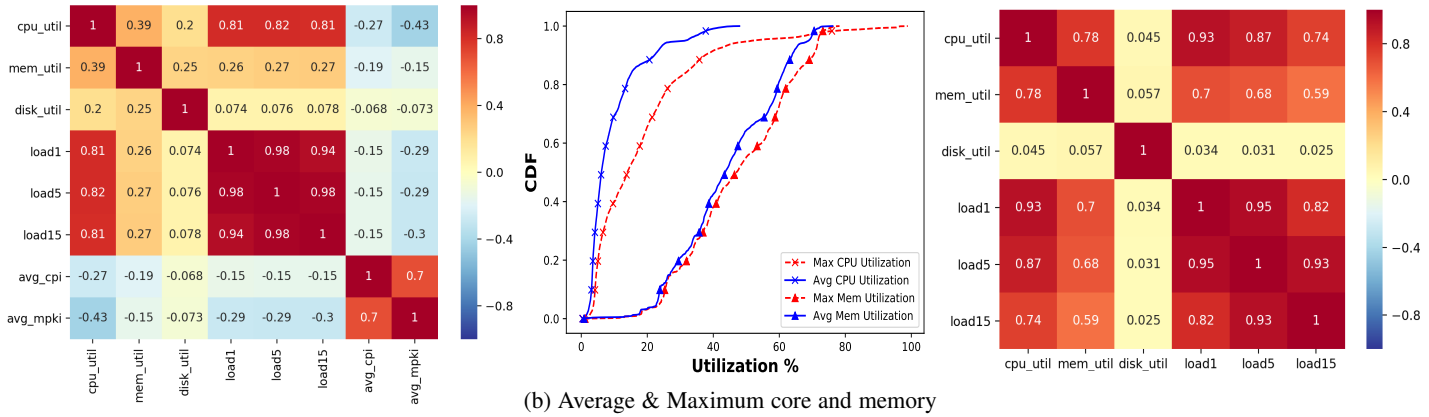


Fig. 1: Energy Efficiency (EE) of CPU and GPU at varying resource utilization %. EE normalized to EE at 100% utilization.

We observe that GPUs are fundamentally different from CPUs as their energy efficiency has a linear relationship with respect to their utilization. Therefore, a cluster-level resource manager needs to consolidate workloads to operate the GPUs at 100% utilization. However, it is not practical to operate all the GPUs in a cluster due to diurnal load interval trends. Hence,

[2]Google's *Kubernetes* means helmsman or captain of the ship managing multiple containers. In *Kubernetes*, *Knots* orchestrates the scheduling speed of the accelerated containers.

[3]We use GPUs and GPGPUs interchangeably. Although the graphic pipeline of GPUs can be virtualized while the GPGPUs do not have the support.



(a) Latency-critical task's usage metrics

(b) Average & Maximum core and memory utilization of containerized services.

(c) Long running batch task's usage metrics

Fig. 2: Alibaba trace analysis of resource utilization of 1300 machines across 12,951 batch jobs & 11,089 containers (12hr period).

the GPU cluster scheduler needs to actively harvest the spare cycles of GPUs to perform load consolidation while letting other GPU nodes be in deep sleep state. Therefore, scheduling policy in terms of utilization should be much more aggressive when compared to CPUs while ensuring the QoS [33]–[36].

Observation 1: *Keeping the GPU utilization high is essential for high energy efficiency when compared to CPUs.*

B. Cluster-level Trace Analysis

In order to obtain high utilization in GPUs, we need to know the user behavior in terms of resource request and resource usage trends. To this end, we analyze the open-sourced Alibaba production datacenter CPU-traces [37]. Cluster-level GPU traces are not publicly available, to robustly evaluate our scheduler, we use CPU traces to to gain insights into the datacenter task arrival trends. We also analyse the characteristics of both batch and latency-critical jobs along with their actual resource demands.

• **Resource overcommitment -** First, jobs tend to overstate their resource requirements. As observed from Figure 2b, average CPU and memory utilization is 47% and 76% respectively. Half of the scheduled pods consume less than 45% of the provisioned memory on an average. This is because over-commitment indirectly helps to ensure the QoS of such queries by provisioning for the worst (peak) case leading to overall underutilization of the cluster [38]. In case of CPUs, virtualization, preemption and fine-grain resource sharing mitigate the problem, while it becomes a critical design point factor as the GPU memory (typically 8-16GB) is limited compared to server-class CPU memory (64-256GB).

Observation 2: *Due to varying resource needs of an application, instead of provisioning resources for the application's worst-case utilization, it is efficient to provision for the average-case. It is ideal if the scheduler could dynamically provision and harvest the spare cycles based on the run-time growth of the application instead of static provisioning.*

• **Utilization metrics are tightly correlated -** Figure 2a plots the Spearman's correlation across eight container resource utilization metrics as a heat map. Equation 1 gives the correlation score ρ where, d_i is the difference between the ranks (ordered in descending i.e, highest value gets rank 1 and lowest value gets rank 8) of corresponding utilization metrics in the heatmap and n is the number of observations. Positive relationship between two metrics is represented by a score close to +1 and vice-versa.

$$\rho = 1 - \frac{6\Sigma d_i^2}{n(n^2-1)} \tag{1}$$

Figure 2c plots the correlation between six utilization metrics of batch workloads. It is seen that the memory utilization strongly correlates with the core utilization. Unlike latency-sensitive queries, the batch applications exhibit strong correlation (both +ve & -ve) between its utilization metrics. Mainly the CPU and memory utilization of batch tasks are positively correlated when compared to latency-sensitive tasks in Figure 2a. Therefore, its relatively easy to predict the utilization footprint of batch tasks.

It can also be seen from Figure 2c, there is a significant relationship between the CPU cores (core_util) and the average system load recorded every 1, 5 and 15 second(s) (load_1, load_5, load_15) in case of batch tasks. However from Figure 2a there are no clear correlation indicators to predict utilization since these tasks are short-lived (few seconds). Thus the load of datacenter for batch applications could be accurately predicted by up to 15 seconds ahead and meanwhile the spare compute and memory could be harvested by dynamically sizing their containers. This enables co-location of other latency-sensitive queries waiting in the pending queue which runs for less than few seconds.

Observation 3: *Alibaba traces show that the batch task's utilization footprint is fairly predictable and the correlation across different resource utilization metrics of batch tasks provides early markers for proactive resource harvesting.*

C. Node-level GPU Characterization

We identify the popular production workloads which subscribe to GPUs. The workloads include applications that use DNN frameworks like Tensorflow, which are hosted as micro-service deployments inside containers and are shipped via orchestrators like Kubernetes [16], Mesosphere [39], Docker

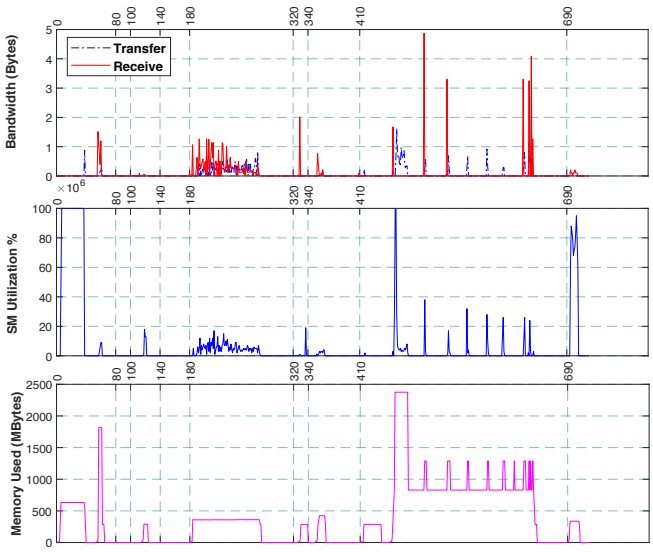


Fig. 3: GPU Resource consumption of entire Rodinia suite on Nvidia's P100 GPU. The grid-lines mark the individual benchmark's runtime. (where x-axis is time in milliseconds)

swarm [40], etc. Generally, these deployments are hosted as services such as object detection, feature extraction, etc. Queries to these containers arrive in short bursts, and the tasks are short-lived. For example, the image recognition DNN-based inference query takes 90*ms* on an average, on Nvidia P100 [41]. Whereas, other batch jobs may run for hours on a GPU, such as, HPC applications, DNN training, etc.

In order to create a datacenter representative workload mix for GPUs, which consists of both batch jobs and user-facing queries: (i) we use applications from Rodinia suite [42] for batch (compute-intensive) workloads and, (ii) we use a mix of DNN-based inference queries from Djinn and Tonic suite [2] for user facing queries. We execute the application-mix on a single GPU node to understand the batch-workload's resource demands. We design a load generator for *Kube-Knots* that mimics the real-world datacenter. We model the load generator after the Alibaba datacenter task inter-arrival times. Further details about the workload are explained in Section III.

1) GPU Batch Workload: Figure 3 plots the GPU resource consumption over time for the eight different applications, which were run sequentially. We characterize memory used, Streaming Multiprocessor (SM) utilization and PCIe bandwidth which are the three dominant resources for a batch application. It is observed that in general the resource consumption of Rodinia is relatively low with a few exceptions over time without increasing the problem size. From Figure 3, we also observe that the resource consumption of applications are highly varying in nature. Hence provisioning for worst-case utilization would lead to a lot of under-utilization. Further, we observe that these applications show a very deterministic pattern of phase changes. For example, typically if an application's input PCIe bandwidth activity is high, it is implied that the compute and memory would follow the same trend in the next few milliseconds. These subtle utilization cues are picked up through real-time feedback and leveraged for resource

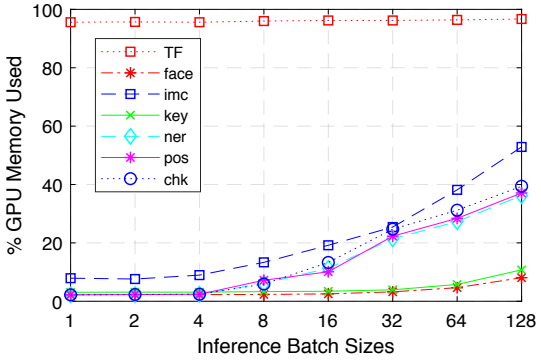


Fig. 4: Memory footprint of DNN inference queries of Djinn and Tonic workload suite. TF is the Tensorflow managed memory consumption.

harvesting and dynamic container resizing by *Kube-Knots*.

Observation 4: *A GPU batch application's utilization footprint is accurately predictable through correlation markers and forecasting. Thus, enabling opportunities for resource harvesting and dynamic container orchestration.*

2) GPU User-Facing Queries: The Djinn and Tonic benchmark uses Tensorflow (TF) as the DNN-framework for executing ML queries. TF also manages the execution flow of the queries that run inside the GPU. By default, TF earmarks the entire GPU memory despite actual workload demands. The TF runtime is designed to make this a default choice [43] because GPUs are assumed to run a single context at a given time and are not designed for multi-tenant/application scenarios. However, in case of public datacenters, to keep the costs low, it is ideal to share the GPUs. This is because the individual application/tenant utilization is generally low.

Figure 4 shows the maximum memory consumption of different machine learning inferences. For most of the single inference queries, the memory consumption is less than 10%. We batched these queries up to 128 queries per batch request. However, the majority of the inferences even with batching consume less than 50% of the device memory presenting an opportunity to be co-located along batch applications. However, the same queries are run using TF, they consume 99% of the GPU memory causing severe internal memory fragmentation. Although, this could have been avoided by exposing the TF APIs to the scheduler.

Observation 5: *To enforce a cluster-level scheduling policy, it is essential to expose the framework APIs to the cluster scheduler to avoid GPU memory fragmentation along with real-time utilization metrics.*

III. Workload Modeling

Accelerators like GPUs in production datacenters are relatively recent, we wanted to faithfully capture the dynamics of applications that would be representative of a production datacenter load. We capture the inter-arrival pattern of tasks in the Alibaba datacenter traces [37] along with the task resource constraints as discussed in Section II. Using the arrival rate of incoming jobs to the Alibaba datacenter, we schedule a mix of batch and interactive GPU jobs to our ten node cluster. This mix is determined by a fixed cut-off based on the Pareto

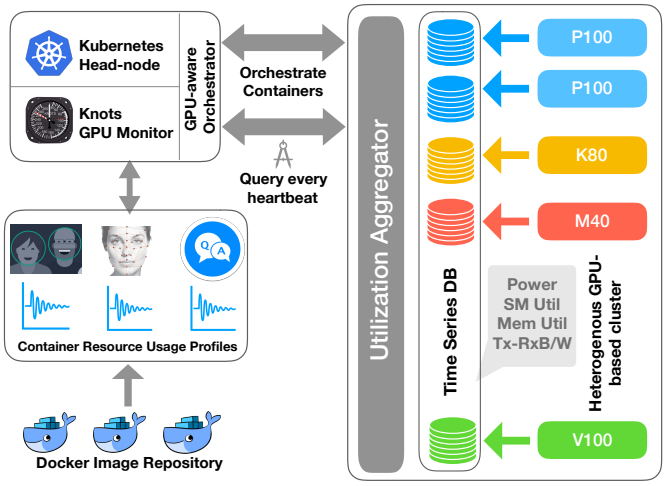


Fig. 5: *Kube-Knots* GPU orchestrator design.

Batch workloads from Rodinia suite					Latency critical	Load	COV
App-Mix-1	leukocyte	heartwall	particlefilter	mummergpu	face,key	HIGH	LOW
App-Mix-2	pathfinder	lud	kmeans	streamcluster	chl,ner,pos	MED	MED
App-Mix-3	particlefilter	streamcluster	lud	myocyte	imc,face	LOW	HIGH

TABLE I: GPU load and covariance for cluster workload suite mixed with batch jobs and latency critical ML inference queries.

principle [44] where 80% jobs consume only 20% of the datacenter resources as they are short-lived tasks and the rest are the long-running batch jobs.

A. Representative GPU workload

We choose eight scientific applications from the Rodinia benchmark suite [42] to represent the batch jobs. For short-lived tasks, we use a mix of DNN-based inference queries (speech, text, and image) from Djinn and Tonic workload suite [2] (From Table I[4]). Based on the resource usage trend in the Alibaba traces, we categorize the application-mix and associate it with three different bins based on the load and covariance as seen in the Table I.

B. Application-mix Analysis

To schedule these workloads onto the GPU nodes, we use the *Kubernetes*'s uniform scheduler to schedule containers. GPU sharing is disabled by default in *Kubernetes* as it complicates the scheduler-level performance/QoS guarantees and to ensure fairness across different co-scheduled applications. To set a fair baseline, we enable GPU sharing across multiple applications and we call this as GPU-agnostic scheduler and set it as our baseline scheduler for comparison. Note that the GPU compute is time-shared while the memory is space-shared. Therefore, scheduler can pack multiple containers on GPUs through periodic resource harvesting by resizing the containers.

We create an experimental setup as shown in Table I and evaluate our baseline GPU-agnostic (Res-Ag) scheduler. We schedule all three app-mixes and plot the 50^{th}, 90^{th}, 99^{th} percentile GPU utilization along with maximum utilization of individual GPUs as seen in Figure 6. We observe in app-mix-1 (in Figure 6a) the median is much closer to 90^{th}/99^{th} percentile utilization when compared to app-mix-3 (in Figure 6c). This implies that the sustained load in case app-mix-1 is *higher* than app-mix-3, as we classified in Table I. On the other hand, app-mix-2 in Figure 6b has the 50^{th} to 99^{th} percentile evenly spaced. This also correlates to the classification we make for app-mix-2 in Table I (*medium and steady* load).

It is evident that, the applications have varying resource needs throughout their lifetime. Also, the applications always overstate their resource requirements by provisioning for peak utilization (also seen in Figure 3). This leads to under-utilization and queuing delays resulting in poor energy efficiency and significant QoS violations. An efficient GPU scheduler would leverage this dynamic workload behavior resulting in improved utilization and energy efficiency.

C. Utilization Variance Across GPU Nodes

We look into the variability of GPU node utilization for these three different application mixes through a statistical metric called Coefficient Of Variation (COV). COV is measured by the ratio of standard deviation σ and mean μ. An application-mix with low COV$\leq$1 denotes consistent nature of its load in terms of GPU utilization, making it easier to guarantee the performance of the pods to be scheduled in future and vice-versa. Further, co-locating a pod in a heavy-tailed distribution (COV$>$1) case would lead to application interference(noisy-neighbor) and severe capacity violations.

We sort the COV for all GPU nodes within an application mix and show it in Figure 7. We observe that the COV is less than 1 for both app-mix-1 and app-mix-2. Therefore, it is easier to predict and guarantee while scheduling in those scenarios. In case of app-mix-3, the COV is greater than 1, and on top these applications have very low resource consumption. However, if the scheduler is agnostic to the real-time utilization of GPUs, it will lead to co-location that results in heavy-tail distribution (e.g, time quantum 410-690 in Figure 3) resulting in capacity failures. Hence it is important for a GPU orchestrator to provision for an application while considering the real-time GPU utilization metrics.

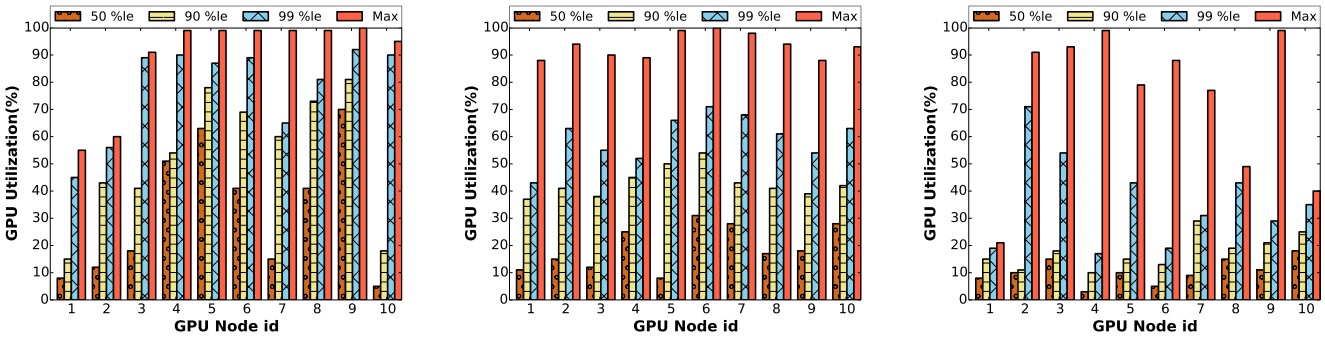
IV. Kube-Knots Design and Integration

In this Section, firstly, we discuss the design of *Knots* framework which serves as a real-time utilization metrics aggregator at the head-node. The data collected by *Knots* is critical to perform GPU-aware scheduling. Secondly, we the describe the design of GPU-aware resource harvesting schedulers we built using the *Kube-Knots* orchestration framework.

A. Knots Design

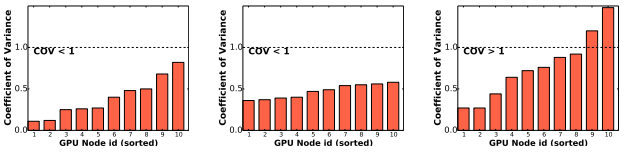
Knots is designed to collect and log the real-time GPU utilization metrics through pyNVML [45]. pyNVML is a python-based API library that exposes utilization metrics to the node-level aggregator. As shown in Figure 5, we log the following five GPU metrics in real-time: (i) streaming multiprocessor (SM) utilization, (ii) memory utilization, (iii) power consumption, (iv) transfer bandwidth and, (v) receive bandwidth. These metrics are pushed to a node-level time-series

[4]The abbreviations for application names can be found in [2]



(a) Application-Mix-1 (b) Application-Mix-2 (c) Application-Mix-3

Fig. 6: 50/90/99th percentile & maximum GPU utilization for different application mixes scheduled using GPU-agnostic scheduler.



(a) App-Mix-1 (b) App-Mix-2 (c) App-Mix-3

Fig. 7: Coefficient of Variance across three Application-Mix.

database IDB (Influx-DB) [46]. *Knots* from the head-node can query the GPU nodes for utilization data via the aggregator which is further explained in detail in Section IV.

Based on the desired prediction accuracy, the quality of scheduling is determined by the frequency at which *Knots* queries the IDB. We call this as heart-beat interval which is a crucial factor in determining the placement quality of the scheduler. From Section II-C2, we know that pods could be dynamically resized by predicting the memory consumption using the metrics like I/O bandwidth and SM_Util, which *Knots* leverages for compaction of pods.

B. Resource agnostic sharing

We enable scheduling of multiple containers (pods) on a single GPU by modifying the Nvidia's k8s-device-plugin [47] for Kubernetes. The GPU-compute is time-shared while the memory is space-shared. We used first fit decreasing order bin-packing algorithm to pack the pods on the GPU which greatly improves the average GPU utilization and job turnaround times. The three application mixes shown in Table I are scheduled using baseline GPU-agnostic (Res-Ag) scheduler.

Figure 6 shows the utilization of each of the ten GPU nodes. Res-Ag scheduler is actually better in cases where the utilization is already high (for instance application-mix-1 from Figure 6a). GPU-sharing satisfies the **Observations 1 & 3** which is maximizing the utilization by enabling resource sharing and keeping the utilization high for energy efficiency respectively. However, it fails to consider the GPU metrics such as free memory and queue length. This leads to severe QoS and capacity violations leading to pod crashes (discussed in **Observation 2**). When GPU sharing is enabled, it is important that the cluster-wide resource scheduler should consider the real-time GPU utilization to safely schedule and co-locate the containers (pods) which lead to our next scheme.

C. Correlation Based Provisioning

Following the observations made from Alibaba cluster traces, in order to make effective scheduling decisions, the correlation between the utilization metrics must be used for proactive application placement, especially when GPU sharing is enabled. Since *Knots* leverages the utilization aggregator to collect the real-time utilization statistics from each GPU nodes, the scheduler can leverage this information to predict whether the pods can be co-located. For example, two pods positively correlated for GPU memory is scheduled to different GPU nodes as the pods have a high probability of memory capacity violations when co-located.

Further, such capacity violations can also lead to container crashing and relaunching. Although the relaunch latency is typically in the order of few seconds, the tasks when relaunched cannot be prioritized over tasks of other pods that are already ahead on the queue. In this case, this particular heavy tailed task affects the overall job completion times.

As observed in Figure 3, all the representative GPU batch workloads on an average have stable resource usage for most of their execution except for the times when the resource demand surges. Specifically, SM utilization has a 90x difference between its median and peak. Similarly, bandwidth utilization differs by 400x between the median and peak. The application uses the whole allocated capacity only for 6% of the total execution time, but it is always provisioned for the peak utilization case. This leads to resource fragmentation and underutilization. This issue could be fixed by harvesting the memory back from the pods by resizing them to co-locate with another pending pod. For example, if two uncorrelated applications are containerized as pods and placed individually, they have an X% probability of failure. Whereas if they are co-located, they have a probability of failing together, which is $1-(1-X)^2$. Hence, provisioning based on an average usage and packing uncorrelated applications together can lead to potential savings over Res-Ag scheduling. We call this scheme correlation based prediction (CBP) which resizes the uncorrelated pods for efficient packing on GPUs.

However, not all over-subscribing pods are suitable candidates for resizing. As seen from Observation 4, we only resize the pods of batch applications with suitable correlation metrics. After harvesting the spare memory the latency-sensitive

tasks are co-located along with the batch-tasks. In doing so, CBP ensures crash-free co-location. CBP scheduler takes $\boldsymbol{O}(N^2d)$ to find the optimal placement where N is the number of pending pods and ***d*** being the time-series window size which is a crucial configuration parameter that determines the scheduling accuracy. CBP scheduler bin packs the uncorrelated applications together by resizing their respective pods for a common case (80^{th} percentile consumption) than for the maximum case. We provision for 80^{th} percentile based on the first principle observed from Figure 2b, since the maximum memory utilization for almost all the containers do not exceed more than 80% of the provisioned memory. Further, aggressive provisioning (Viz. 50^{th}, 60^{th}, etc.) lead to constant resizing which affects the docker performance at scale. CBP is based on our observations that the probability of all co-located pods reaching their peak resource consumption at the same time is very low. Thus, CBP is aware of both utilization and the pod's correlation metrics such as memory, SM load and bandwidth.

We notice that the GPUs are still being underutilized due to static provisioning of CBP. This is due to the inter-arrival pattern of pods, since there are not enough negatively correlating pods to co-locate which results in a skewed schedule order. This indirectly restricts the potential number of GPU nodes to schedule the pods and results in increased queue waiting times for the pending pods which are positively correlating. The average pod waiting time worsens especially in case of heavily loaded application-mix. Pod affinities would further restrict scheduling options for correlating pods. This leads us to our next scheme that attempts to co-locate two positively correlated pods in the same node by predicting the peak resource demand phases of the scheduled pods.

D. Peak Prediction Scheduler

Peak Prediction scheduler is built on top of CBP to further exploit the temporal nature of peak resource consumption within an application. This allows PP to schedule two positively correlating pods to the same GPU as they may not contend for the resource at the same time. This is due to the fact that a GPU application goes through several phase changes during its course of execution. For instance, a DNN instance query first loads the inference weights from the host which results in peak input bandwidth consumption, this is followed by the next phase of the application that is the compute/memory intensive phase. This trend is also very evident in batch workloads as seen in Figure 3. Thus, the peak bandwidth consumption of an application is an early indicator of subsequent compute and memory peaks. Likewise, within an application, the consecutive peak resource demand trends can be predicted using the auto-correlation function given in Equation 2, r_k being the auto-correlation value where Y_i is the utilization measurement at a given time, $\bar{Y}$ is the moving average of Y, and n is the total number of events record at a time window T.

$$r_k = \frac{\Sigma_{i=1}^{n-k}(Y_i - \bar{Y})(Y_{i+k} - \bar{Y})}{\Sigma_{i=1}^{n}(Y_i - \bar{Y})^2} \tag{2}$$

Algorithm 1 CBP + Peak Prediction Scheduler

```
for gpu_node in ∀all_gpu_nodes do
    if isActive(QUERY(gpu_node)) then
        All_Active_GPUs.append(gpu_node)
Node_List ← Sort_by_Free_Memory(All_Active_GPUs)
Pending_Apps ← Sort_Apps_by_Memory_Size(Apps)
Harvest_Resource ← Docker_Resize(Node_List,Pend_Apps)
Selected_Node ← Head(Node_list)
for App in Pend_Apps do
    SCHEDULE(app,Selected_Node)

procedure QUERY(GPU_node)
    node_stats_mem ← query_db(gpu_node.memory)
    node_stats_sm ← query_db(gpu_node.sm)
    node_stats_tx ← query_db(gpu_node.tx_band)
    node_stats_rx ← query_db(gpu_node.rx_band)
    node_stats ← node_stats_mem + node_stats_sm + node_stats_bw
    return node_stats

procedure SCHEDULE(App, Selected_Node)
    if Can_Co-locate(COV,App,Selected_Node) then
        Ship_Container(App,Selected_Node)
    else
        if AutoCorrelation(node.memory) then    ▷ Eqn 2
            pred_free_mem ← ARIMA(node.mem)
            if pred_mem ≥ App.memory then
                Schedule(app)
            else
                Schedule(App,Head → Next(Node_list)
```

The current utilization metrics can be further forecast to predict the future GPU utilization using non-seasonal Auto-Regressive Integrated Moving Average(ARIMA). We quantitatively analyzed the mean-squared-error and profiling overheads of different regression models such as linear-regression, random forest, SGD, automatic relevance determination, Theil-Sen, and multi-layer perceptron. Due to space constraints, we could not discuss the accuracy/runtime costs of these models. Our sliding-window (five-seconds) consists of few data points, and therefore, a statistical model such as ARIMA works with good accuracy. Other complex models do not improve much due to limited real-time training data.

The interval between two consecutive peak resource consumption of a particular resource could be determined by the auto-correlation factor. If the auto-correlation value of a series (i.e., memory utilization) is zero or negative then it shows that (i) the input time-series data is limited or (ii) the trend is not strong enough to predict a positive correlation. In case of correlation value being greater than 0, we use first-order ARIMA to forecast the utilization of the GPU for the next one second as given in Equation 3 below which is a moving window based linear regression where $\hat{Y}_{pred}$ is a predicted utilization value from previous utilization time-series Y_{t-1}, where ϕ is the slope of the line and μ is the intercept.

$$\hat{Y}_{pred} = \mu + \phi_1 Y_{t-1} \quad (3)$$

Note that, the correlation between consecutive peak resource consumption is a better indicator than the correlation across complete runtime of the application. Also, forecasting the GPU utilization enables the system scheduler to predict the performance better and guarantee the QoS of the application which would be scheduled to that particular GPU node. The peak prediction scheme identifies the set of pods that attains peak consumption of a resource at the same time and schedules them to different GPU nodes.

Algorithm 1 captures the workflow of the scheduler, where for all the active GPU nodes in the datacenter excluding the GPUs which are in deep sleep power state, the utilization for the past five seconds is queried from the respective Influx-DB databases. This time-series interval window determines the prediction accuracy. The utilization aggregator in the head-node sorts the nodes by free memory available for the most recent timestamp. The schedule order of pending pods is sorted based first fit decreasing order of pod's requested memory and resized for 80^{th} percentile memory consumption. The `Select_Node` function analyzes the correlation in case of CBP and schedules to the `Selected_Node` in case of the correlation value is less than 0.5. In the case of PP, auto-correlation function is used on the utilization time-series data for a particular metric of the selected node. This is used to look for a trend, which predicts a possibility of an impending peak resource consumption to occur through ARIMA. Subsequently leading to schedule on the next available node in the list by repeating the same admission checks. Once the node is selected, the pod is shipped to the node using *Kubernetes*'s python-based client API call.

V. Evaluation Methodology

A. Hardware

We use Dell PowerEdge R730 eleven node cluster where ten worker nodes have P100 GPUs with Intel Xeon CPU host along with a CPU-only head-node. The details of the single node hardware configuration are listed in the Table II. We use *Kubernetes* as the resource orchestrator and its default uniform scheduler as a baseline in our experiments.

- **Worker node -** At every GPU worker node, there is a node-level resource monitor that uses python based API library, pyNVML [45] to query the GPU for every heartbeat interval. The query returns all the five utilization metrics namely, SM, memory, power, transfer and receive bandwidth. These metrics are logged as time-series data into the Influx-DB in their nodes.
- **Head node -** The configuration is similar to that in Table II, with an exception that it does not have a GPU. Next, the utilization aggregator on the head node queries the real-time GPU utilization from the Influx-DB. The frequency of querying interval is set to 1ms (justified in Section VI-D). The frequency of data logging (heartbeat) can be varied at the discretion of the utilization aggregator as seen in Figure 5.

Hardware	Configuration
CPU	Xeon E5-2670
Cores	12x2(threads)
Clock	2.3 Ghz
DRAM	192 GB
GPU	P100(16GB)

TABLE II: Hardware config.

Software	Version
Kubernetes	1.9.3
NvidiaDocker	2.0
pyNVML	7.352.0
InFluxDB	1.4.2
CUDA	8.0.61
Tensorflow	1.8

TABLE III: Software config.

B. Container Configuration

The software setup and configuration is given in Table III. Both the batch and user-facing applications are containerized as pods in *Kubernetes*. When the query is scheduled for the first time to a GPU node, the dependent docker layers such as Tensorflow is downloaded from a repository. The subsequent queries using the same image do not incur this cold-start latency. However, data transfer latency from storage nodes to CPU memory and subsequently to GPU is accounted for in our experiments. PP scheduler leverages the Nvidia-docker system commands to resize the GPU containers dynamically.

Recall from Section II that, for batch applications we used Rodinia [42], HPC workload suite. Similarly, for latency critical workloads, we used Djinn & Tonic workload suite [2] which has several DNN inference queries using TF on GPUs through Keras libraries for execution. These models are containerized and made publicly available at DockerHub [48]. We configured TF's GPU configuration knobs to allow incremental memory growth for the DNN inference queries. We capture the inter-arrival pattern of the Alibaba trace which is representative of the datacenter workload. We incorporate this inter-arrival pattern in our launch sequence of jobs. The job sequence comprises of a bin of application-mixes (refer Table I).

C. Simulator setup for DNN Tasks

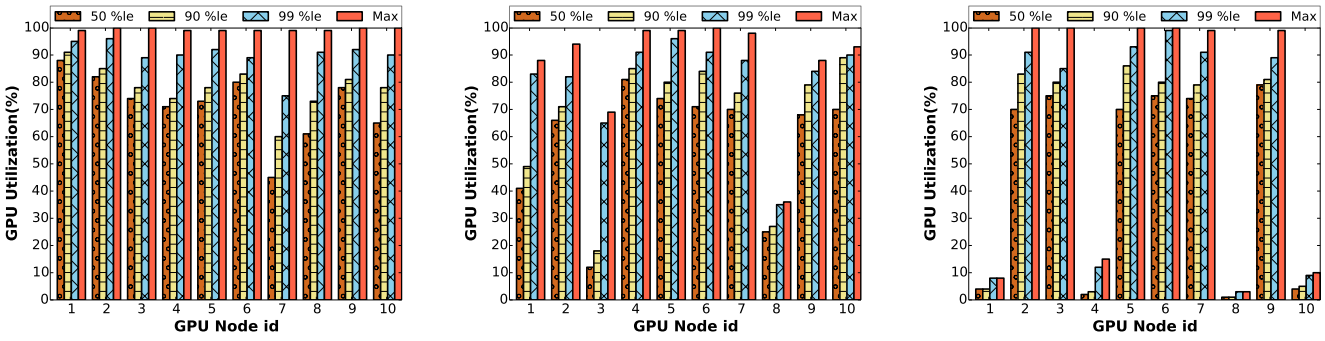
We build CBP+PP on top of the discrete-time simulator [49] to compare *Kube-Knots* with other state-of-the-art schedulers such as Tiresias, Gandiva, etc,. We generate an experimental workload of 520 DL Training (DLT) and 1400 DL Inference (DLI) tasks. The job requirements of DLT of various DNN metrics is modeled after Tiresias [50], while the distribution of DLT and DLI tasks is based on the app-mixes given in Table I. Execution time of these DLT tasks ranges from few minutes to few hours depending on the model and training rounds, while the DLI tasks are in the order of few milliseconds to seconds. The input to the simulator is modeled using the inter-arrival times in the Alibaba job arrival traces (12hr period). The simulated cluster configuration includes 32 nodes with each node having 8 GPUs, 40 CPU cores, and 256GB memory.

VI. Results and Analysis

In this section, we evaluate the different GPU-based scheduling schemes for four major criteria: (i) Cluster-wide utilization, (ii) QoS violations, (iii) Power, (iv) Accuracy of prediction, and (v) Job Completion Times (JCT) and QoS comparison for Deep Learning Training (DLT) and Inference (DLI) Tasks.

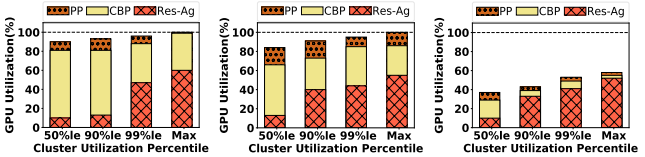
A. Cluster-wide GPU Utilization

Building upon the GPU-agnostic scheduling (shown in Figure 6), we propose two scheduling schemes: (i) CBP and



(a) Application-Mix-1 (b) Application-Mix-2 (c) Application-Mix-3

Fig. 8: 50/90/99th percentile & maximum utilization of GPUs for App-Mixes scheduled using Peak Prediction Scheme.



(a) App-Mix-1 (b) App-Mix-2 (c) App-Mix-3

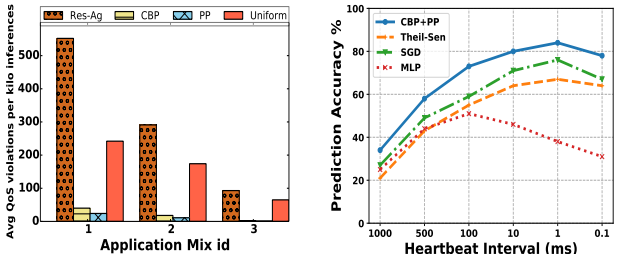
Fig. 9: Cluster-wide GPU utilization improvements.

(ii) PP. In Figure 8, we plot the utilization improvements of each of the ten GPU nodes for all the three app-mixes using the PP scheduler. The PP scheduler improves the utilization in all the three app-mixes by an average of 62%. Especially in app-mix 2 and app-mix 3, where the utilization is medium and low respectively, the PP scheme does an effective consolidation of workloads using a minimal number of GPUs as possible. As seen in Figure 8c, GPU nodes 1, 4, 8, and 10 are minimally used as the scheduler effectively consolidated all the low demand workloads to a minimum number of active GPUs as possible. This consolidation shows that, PP leverages the real-time utilization along with utilization forecasting before making any scheduling decisions.

Figure 9 plots the overall GPU utilization improvements for all three app-mixes for median and tail percentiles. It can be noted that the PP scheme consistently improved the overall GPU utilization in all low, medium and high application-mix cases. For example, PP in app-mix-1 improves by 80% for both median and tail percentiles when compared with Res-Ag scheduler. The improvements are also consistent in the other two app mixes when compared to Res-Ag scheduler improving median and peak utilization by up to 60% and 45% respectively.

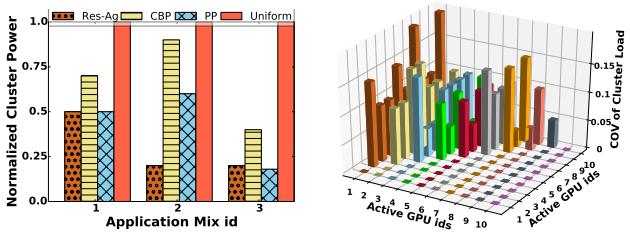
B. QoS of latency critical workloads

QoS violation threshold for latency-sensitive applications is typically set around 150 milliseconds [51]. We show in Figure 10a that the baseline scheduler violated QoS by 18% on average even though the applications are not co-located together. This is because of head-of-the-line (HOL) blocking of batch tasks resulting in queuing delays, when the query is sent to a busy GPU node. The Res-Ag performs worse than the baseline and violating by up to 10% (app-mix-3) and 53% (app-mix-1). On the contrary, CBP and PP have almost no violations in all three app-mixes (<1%). This is because they are aware of real-time GPU utilization and provision the container memory for 80



(a) Average QoS violations per 1000 (kilo) inference queries. (b) Prediction accuracy for varying heartbeat intervals in ms.

Fig. 10: QoS guarantees and Accuracy of PP Scheduler.



(a) Normalized power comparisons across four schedulers. (b) COV of SM-Load using CBP+PP for App-Mix-1.

Fig. 11: Cluster-wide Power savings and Load balancing.

percentile utilization while guaranteeing the QoS via ARIMA forecasting. Note that, we also co-locate the latency sensitive jobs along with the batch jobs without affecting their QoS. PP achieves this by predicting the inter and intra-application correlation for utilization and, at the same time by leveraging *Knots* real-time data, it can forecast the future GPU utilization.

Figure 11b plots the pair-wise COV of GPU cluster loads. The lower diagonal values are omitted for the sake of clarity. It can be observed that the COV of loads across different GPU nodes ranges within 0 to 0.2 when compared to the COV in Figure 7a which ranges from 0.1 to 0.7. Hence, PP performs efficient load balancing even in the case of high-load scenarios such as app-mix-1 along with consolidation in the case of highly sporadic low-load scenarios such as app-mix-3. The results for other app-mixes are also similar to this due to harvesting and consolidation. Thus, CBP along with PP ensures the load balancing in the case of peak load scenarios at the same time minimizing the total number of active GPUs for energy efficiency in the case of low load scenarios.

C. Energy efficiency

Figure 11a plots the overall power consumed by the different schedulers, where Res-Ag consumes the least power on an average (33%) while PP scheme consumes the second least power (43%). Note that, while Res-Ag optimizes for power it also violates QoS for almost 53% of the requests as discussed in section VI-B. On the other hand, the PP scheduler only consumes 10% more power than Res-Ag while guaranteeing minimal QoS violations due to that fact that PP scheduler attempts to pack only active GPUs but it ensures that two pods do not peak for resource consumption at the same time. The CBP policy consumes more power than both Res-Ag and PP schemes by 25% and 15% respectively, although being similar to PP scheme in terms of QoS violations. This emphasizes the importance of predicting the peak resource consumption of pods on top of the correlation metrics. The energy savings of CBP+PP schemes are also due to consolidating the queries in active GPUs in case of low load while letting the other GPUs be in minimum idle power consumption (`p_state 12`).

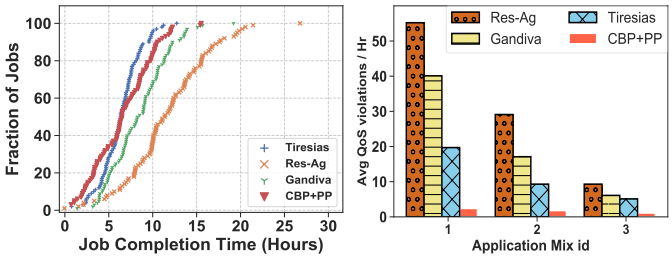
D. Accuracy of the Peak Predictor

In the case of CBP+PP scheduler, ***d*** is the frequency at which the utilization aggregator at head-node is querying the worker nodes for GPU utilization that forms our time-series data. We vary the frequency at which we query the GPUs to predict the peak resource usage. As seen in Figure 10b, CBP+PP improves its accuracy from 36% to 84% when the sampling interval for querying utilization is varied from 1000ms to 1ms, beyond which the prediction accuracy drops. Hence for the maximum prediction accuracy, we set the utilization aggregator to query the GPU nodes every 1ms. The overhead for querying the GPU at every 1ms interval does not affect the pod's performance.

We also have analyzed the accuracy of other complex ML models (SGD, Theil-Sen, and multi-layer perceptron (MLP)) for utilization estimation as shown in Figure 10b. They all are in the accuracy range that is similar or worse despite their high run-time complexity when compared to ARIMA based CBP+PP. This is mainly due to the limited training set size (5s) during the run-time. Further, increasing the heartbeat frequency beyond 1ms lead to poor utilization estimations due to over-fitting of the model from the training data.

E. Job Completion Time (JCT) for Deep Learning Workload

We compare the JCT of CBP+PP scheduler along with the other state-of-the-art DLT task schedulers (Gandiva [23], Tiresias [50]) using the experimental setup discussed in Section V-C. As observed from the Figure 12a, CBP+PP can effectively schedule by up to 60% of the jobs in the workload suite by significantly avoiding delays due to queuing, preemption, and migration of tasks. These jobs are predominately composed of Inference tasks. In case of DLT tasks, the application-aware schedulers (Gandiva and Tiresias) can effectively optimize



(a) Job completion time comparison shown in hours for App-Mix-1. (b) Average QoS violations of DL inference queries per hour.

Fig. 12: Comparison of CBP+PP across different DLT Schedulers.

for model convergence through suspend-and-resume based opportunistic scheduling policy and Least Attained Service (LAS) seem to perform well. Tiresias, especially, is able to achieve better JCT at 99% which are predominantly of DLT jobs. However, this comes at a cost of multiple SLO violations of DLI jobs due to task migrations, preemptions, and HOL blocking of these latency-sensitive tasks.

The quantitative breakdown of JCT is given in Table IV, where CBP+PP performs $1.3\times$ and $1.11\times$ better than Gandiva and Tiresias respectively in terms of median JCT. This is due to SLO-aware scheduling of latency-sensitive DLI tasks. Further, during peak resource demands the PP scheduler ensures crash-free resizing of the DLT workloads by predicting the peak-utilization (mini-batch training phases) to accommodate DLI tasks. In the case of average JCT, CBP+PP performs $1.36\times$ better than Gandiva as the latter performs trial-and-error task placement leading to severe HOL blocking of small tasks. However, in the case of Tiresias, the average JCT improvements are marginal as LAS prioritizes for small DLT jobs by reducing the queuing delays. One of the main advantages of CBP+PP comes from the real-time utilization of the cluster, as it can perform colocations without preemption or HOL blocking.

Scheduler	Average	Median	99%
Resource-Agnostic	$1.63\times$	$1.67\times$	$1.47\times$
Gandiva	$1.36\times$	$1.30\times$	$1.11\times$
Tiresias	$1.07\times$	$1.11\times$	$0.91\times$

TABLE IV: JCT improvements across different Schedulers for DL-workloads normalized by CBP+PP scheduler.

Figure 12b shows the average QoS violations per hour (12hr trace). Depending on the DNN model, inference queries running on GPUs can be in the order of 10 to 50 ms. However, during peak GPU resource demands, these tasks incur severe queuing delay due to HOL blocking in the case of Gandiva. To mitigate this, it performs job migrations that incur latency up to few seconds affecting the QoS. However, Tiresias mitigates this delay by performing job-preemptions to prioritize other short jobs ensuring QoS during resource demand surges. As seen from the Figure 12b, in case of app-mix-1, Gandiva and Tiresias incur on an average up to 33% and 17% SLO violations when compared to CBP+PP, which schedules them on FCFS basis without any queuing delays or preemptions. Therefore, CBP+PP along with *Kube-Knots* can perform application-agnostic placement and scheduling of DNN workloads by improving the overall JCT while guaranteeing the QoS.

VII. Related Work

GPU-first resource management in datacenters is an emerging research problem. We have done a comprehensive analysis of relevant scheduling frameworks, shown in Table V and broadly classified in to following four categories.

GPU-aware runtime systems: There have been number of works including Bubble-up [29] and Bubble-flux [52] in the past addressing CPU utilization aware scheduling. Quite recently, researchers have proposed runtime changes to the GPU to enable better scheduling of the GPU tasks either by predicting task behavior or reordering queued tasks. Ukidave et al. [31] optimized for workloads which under-utilize the device memory and bandwidth. Chen et al. [53] proposed Baymax, a runtime mechanism which does workload batching and kernel reordering to improve the GPU utilization. Other techniques such as interference driven resource management proposed by Phull et al. [54], predicts the interference on a GPU to do safe co-location of GPU tasks. They do not consider memory bandwidth contention nor over-commitment challenges as they assume sequential execution of the GPU tasks while performing static profiling of applications to ensure safe co-locations. Most of these approaches aim to increase utilization of a individual GPU node and do not scale at the cluster level as they depend on offline training models for node-level run time prediction [28], [31], [53].

Node-level GPU-aware scheduling: Recent works [55], [56] have proposed docker-level container sharing solutions. In this approach, multiple containers can be made to fit in the same GPU as long as the active working set size of all the containers are within the GPU physical memory capacity. These scheduling techniques over-commit resources for individual containers to ensure crash free execution. This may lead to severe internal memory fragmentation as we show in Figure 4. We address this by predicting the resource usage and dynamically resizing the containers based on the future resource consumption projections while guaranteeing the QoS.

Distributed GPU scheduling for DNN applications: Distributed DNN training based applications have started taking advantage of multiple GPUs in a cluster. There are emerging schedulers such as Gandiva [23], Optimus [24], and several other works [25], [26], [57], [58] that focus on prioritizing the GPU tasks that are critical for the DNN model accuracy in case of parameter server-based architecture. These schedulers are designed to cut down on the DNN model exploration time and do not scale well to other application domains as it needs domain-specific knowledge and the application progress metrics (mini-batch completion time). Domain-agnostic scheduling and consolidation is important in case of a public datacenters with multiple-tenants sharing the GPUs, they might run a mix of batch and latency jobs of various application domains and not just the DLT (Deep Learning Training) jobs. *Kube-Knots* is designed to cater multi-faceted GPU applications and does not require any application-domain knowledge or a priori profiling.

Hardware support for virtualizing GPUs: There has been extensive work on providing hardware support for GPU virtualization [59], [60] and preemption [61], [62]. Gupta et al. [59] implemented a task queue in the hypervisor to allow virtualization and preemption of GPU tasks. Tanasic et al. [63] proposed a technique that improves the performance of high priority processes by enabling preemptive scheduling on GPUs. Aguilera et al. [64] proposed a technique to guarantee QoS of high priority tasks by spatially allocating them on more SMs in a GPU. All these techniques require vendors to add extra hardware extensions. Our proposed scheduler does not need any hardware level changes and can be readily deployed in commercial off-the-shelf datacenters using *Kubernetes*.

Features	Baymax [53]	Gandiva [23]	Optimus [24]	Mystic [31]	Tiresias [50]	KubeKnots
Workload consolidation	✓	✓	✓	✓	✓	✓
Application Agnostic	✓	✗	✗	✓	✗	✓
SLO Guarantees	✓	✗	✗	✓	✗	✓
Needs a priori profiling	✓	✗	✗	✓	✗	✗
Application memory resizing	✗	✗	✗	✗	✗	✓
Accelerator power-aware	✗	✗	✗	✗	✗	✓
Application memory resizing	✗	✗	✗	✗	✗	✓
Resource peak prediction	✗	✗	✗	✗	✗	✓

TABLE V: GPU-based scheduling features comparison.

VIII. Conclusion

In this paper, we identify several challenges in existing resource orchestrators like *Kubernetes* towards dynamic resource orchestration for GPU-based datacenters. Motivated by our observations, we propose *Knots*, a GPU aware orchestration layer. *Knots* along with *Kubernetes* performs GPU-aware orchestration. We further evaluate two GPU-based container scheduling techniques on a ten node GPU-cluster, which leverages *Kube-Knots* to harvest the spare GPU resources. Our proposed Peak Prediction (PP) scheduler, when compared against the GPU-agnostic scheduler, improves the cluster-wide GPU utilization by up to 80% for both average and 99^{th} percentile, through QoS aware workload consolidation leading to 33% cluster-wide energy savings. Further, our trace-driven simulations to schedule DNN workloads on a cluster of 256 GPUs showed that the CBP+PP together can improve the average job completion times by up to $1.36\times$ in the case of latency-sensitive inference tasks when compared to other state-of-the-art DNN schedulers. *Kube-Knots* also reduced the overall QoS violations of latency sensitive queries by up to 53% when compared to the GPU-agnostic scheduling with GPU utilization prediction accuracy as high as 84%.

IX. Acknowledgment

We thank our shepherd and the reviewers for their suggestions and valuable comments. We are also indebted to Ram Srivatsa Kannan, Ashutosh Pattnaik, Cyan Mishra, Vivek Bhasi, and Prasanna Rengasamy for their insightful comments on several drafts of this paper. Also, this research is generously supported by NSF grants #1931531, #1912495, #1908793, #1763681, #1629129, #1629915 and we thank NSF Chameleon Cloud project CH-819640 for their generous compute grant.

REFERENCES

[1] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim *et al.*, "A cloud-scale acceleration architecture," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–13.

[2] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 27–40.

[3] J. Gunasekaran, P. Thinakaran, M. T. Kandemir, B. Urgaonkar, G. Kesidis, and C. R. Das, "Spock: Exploiting serverless functions for slo and cost aware resource procurement in public cloud," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019.

[4] "Cisco's Global Cloud Index," https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html.

[5] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 1–12.

[6] J. Jeffers and J. Reinders, *High performance parallelism pearls volume two: multicore and many-core programming approaches*. Morgan Kaufmann, 2015.

[7] V. Nagarajan, K. Lakshminarasimhan, A. Sridhar, P. Thinakaran, R. Hariharan, V. Srinivasan, R. S. Kannan, and A. Sridharan, "Performance and energy efficient cache system design: Simultaneous execution of multiple applications on heterogeneous cores," in *2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2013, pp. 200–205.

[8] V. Nagarajan, R. Hariharan, V. Srinivasan, R. S. Kannan, P. Thinakaran, V. Sankaran, B. Vasudevan, R. Mukundrajan, N. C. Nachiappan, A. Sridharan *et al.*, "Scoc ip cores for custom built supercomputing nodes," in *2012 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2012, pp. 255–260.

[9] V. Nagarajan, V. Srinivasan, R. Kannan, P. Thinakaran, R. Hariharan, B. Vasudevan, N. C. Nachiappan, K. P. Saravanan, A. Sridharan, V. Sankaran *et al.*, "Compilation accelerator on silicon," in *2012 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2012, pp. 267–272.

[10] P. V. Rengasamy, H. Zhang, N. Nachiappan, S. Zhao, A. Sivasubramaniam, M. T. Kandemir, and C. R. Das, "Characterizing diverse handheld apps for customized hardware acceleration," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2017, pp. 187–196.

[11] P. V. Rengasamy, H. Zhang, S. Zhao, N. C. Nachiappan, A. Sivasubramaniam, M. T. Kandemir, and C. R. Das, "CritICs Critiquing Criticality in Mobile Apps," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2018, pp. 867–880.

[12] "Amazon EC2 Elastic GPUs," https://aws.amazon.com/ec2/elastic-gpus/.

[13] "Microsoft VM with GPU," https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-gpu.

[14] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, "Grandslam: Guaranteeing slas for jobs in microservices execution frameworks," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. ACM, 2019, pp. 34:1–34:16.

[15] S. Zhao, P. V. Rengasamy, H. Zhang, S. Bhuyan, N. C. Nachiappan, A. Sivasubramaniam, M. Kandemir, and C. R. Das, "Understanding energy efficiency in iot app executions," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. ICDCS, 2019.

[16] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 50–57, Apr. 2016. [Online]. Available: http://doi.acm.org/10.1145/2890784

[17] "Mesos Scheduler GPU support," http://mesos.apache.org/documentation/latest/gpu-support/.

[18] R. S. Kannan, A. Jain, M. A. Laurenzano, L. Tang, and J. Mars, "Proctor: Detecting and investigating interference in shared datacenters," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2018, pp. 76–86.

[19] R. S. Kannan, "Enabling fairness in cloud computing infrastructures," Ph.D. dissertation, University of Michigan, 2019.

[20] R. S. Kannan, M. Laurenzano, J. Ahn, J. Mars, and L. Tang, "Caliper: Interference estimator for multi-tenant environments sharing architectural resources," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 16, no. 3, p. 22, 2019.

[21] D. Wong, "Peak efficiency aware scheduling for highly energy proportional servers," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016, pp. 481–492.

[22] P. Thinakaran, J. Raj, B. Sharma, M. T. Kandemir, and C. R. Das, "The curious case of container orchestration and scheduling in gpu-based datacenters," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '18. ACM, 2018, pp. 524–524.

[23] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang *et al.*, "Gandiva: introspective cluster scheduling for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 595–610.

[24] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 3.

[25] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on gpu clusters," *arXiv preprint*, 2017.

[26] S. R. Seelam and Y. Li, "Orchestrating deep learning workloads on distributed infrastructure," in *Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning*. ACM, 2017, pp. 9–10.

[27] M. Jeon, S. Venkataraman, J. Qian, A. Phanishayee, W. Xiao, and F. Yang, "Multi-tenant gpu clusters for deep learning workloads: Analysis and implications," MSR-TR-2018, Tech. Rep., 2018.

[28] Q. Chen, H. Yang, M. Guo, R. S. Kannan, J. Mars, and L. Tang, "Prophet: Precise qos prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2017, pp. 17–32.

[29] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 248–259. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155650

[30] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "Timegraph: Gpu scheduling for real-time multi-tasking environments," in *Proc. USENIX ATC*, 2011, pp. 17–30.

[31] Y. Ukidave, X. Li, and D. Kaeli, "Mystic: Predictive scheduling for gpu based cloud servers using machine learning," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 2016, pp. 353–362.

[32] " Auto-Regressive Integrated Moving Average," http://people.duke.edu/~rnau/411arim.htm#arima010.

[33] A. Pattnaik, X. Tang, O. Kayiran, A. Jog, A. Mishra, M. T. Kandemir, A. Sivasubramaniam, and C. R. Das, "Opportunistic computing in gpu architectures," in *Proceedings of the 46th International Symposium on Computer Architecture*. ACM, 2019, pp. 210–223.

[34] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, "Scheduling techniques for gpu architectures with processing-in-memory capabilities," in *Parallel Architecture and Compilation Techniques (PACT), 2016 International Conference on*, 2016, pp. 31–44.

[35] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, ser. USENIX'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 28–28. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855807.1855835

[36] A. Jog, O. Kayiran, T. Kesten, A. Pattnaik, E. Bolotin, N. Chatterjee, S. W. Keckler, M. T. Kandemir, and C. R. Das, "Anatomy of gpu memory system for multi-application execution," in *Proceedings of the 2015 International Symposium on Memory Systems*, ser. MEMSYS '15. New York, NY, USA: ACM, 2015, pp. 223–234. [Online]. Available: http://doi.acm.org/10.1145/2818950.2818979

[37] "Alibaba Cluster Trace Data description," https://github.com/alibaba/clusterdata.

[38] P. Thinakaran, J. R. Gunasekaran, B. Sharma, M. T. Kandemir, and C. R. Das, "Phoenix: A constraint-aware scheduler for heterogeneous datacenters," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 977–987.

[39] "Mesosphere description," https://mesosphere.com/738085.html.

[40] "Docker Swarm description," https://docs.docker.com/engine/swarm/swarm-tutorial/.

[41] "GPU-Based Deep Learning Inference," https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf.

[42] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. Ieee, 2009, pp. 44–54.

[43] "Memory Growth in TensorFlow," https://www.tensorflow.org/guide/using_gpu#allowing_gpu_memory_growth.

[44] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012, p. 7.

[45] "Python Bindings for the NVIDIA Management Library," https://pypi.python.org/pypi/nvidia-ml-py/4.304.04.

[46] "InfluxDB time series," https://github.com/influxdata/influxdb.

[47] "Nvidia k8s-device-plugin for Kubernetes," https://github.com/NVIDIA/k8s-device-plugin.

[48] "Docker TensorFlow / HPC experiments used in evaluation of kube-knots," https://hub.docker.com/r/prashanth5192/gpu.

[49] "Tiresias discrete-time trace-driven simulator," https://github.com/SymbioticLab/Tiresias/tree/master/simulator.

[50] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A gpu cluster manager for distributed deep learning," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI'19)*, 2019, pp. 485–500.

[51] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2408776.2408794

[52] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 607–618. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485974

[53] Q. Chen, H. Yang, J. Mars, and L. Tang, "Baymax: Qos awareness and increased utilization for non-preemptive accelerators in warehouse scale computers," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2, pp. 681–696, 2016.

[54] R. Phull, C.-H. Li, K. Rao, H. Cadambi, and S. Chakradhar, "Interference-driven resource management for gpu-based heterogeneous clusters," in *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '12. New York, NY, USA: ACM, 2012, pp. 109–120. [Online]. Available: http://doi.acm.org/10.1145/2287076.2287091

[55] J. Gleeson and E. De Lara, "Heterogeneous gpu reallocation," in *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*, 2017.

[56] D. Kang, T. J. Jun, D. Kim, J. Kim, and D. Kim, "Convgpu: Gpu management middleware in container based virtualized environment," in *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 2017, pp. 301–309.

[57] M. Amaral, J. Polo, D. Carrera, S. Seelam, and M. Steinder, "Topology-aware gpu scheduling for learning workloads in cloud environments," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 17.

[58] H. Zhang, L. Stafman, A. Or, and M. J. Freedman, "Slaq: quality-driven scheduling for distributed machine learning," in *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 2017, pp. 390–404.

[59] V. Gupta, K. Schwan, N. Tolia, V. Talwar, and P. Ranganathan, "Pegasus: Coordinated scheduling for virtualized accelerator-based systems," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 3–3.

[60] C.-H. Hong, I. Spence, and D. S. Nikolopoulos, "Gpu virtualization and scheduling methods: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, p. 35, 2017.

[61] J. J. K. Park, Y. Park, and S. Mahlke, "Chimera: Collaborative preemption for multitasking on a shared gpu," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: ACM, 2015, pp. 593–606. [Online]. Available: http://doi.acm.org/10.1145/2694344.2694346

[62] K. Sajjapongse, X. Wang, and M. Becchi, "A preemption-based runtime to efficiently schedule multi-process applications on heterogeneous clusters with gpus," in *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '13. New York, NY, USA: ACM, 2013, pp. 179–190. [Online]. Available: http://doi.acm.org/10.1145/2462902.2462911

[63] I. Tanasic, I. Gelado, J. Cabezas, A. Ramirez, N. Navarro, and M. Valero, "Enabling preemptive multiprogramming on gpus," in *Proceeding of the 41st Annual International Symposium on Computer Architecture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 193–204.

[64] P. Aguilera, K. Morrow, and N. S. Kim, "QoS-aware dynamic resource allocation for spatial-multitasking GPUs," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014, pp. 726–731.