2021 IEEE 39th International Conference on Computer Design (ICCD) | 978-1-6654-3219-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCD53106.2021.00056

CHARM: Collaborative Host and Accelerator Resource Management for GPU Datacenters

Wei Zhang*, Kaihua Fu*, Ningxin Zheng†, Quan Chen*, Chao Li*, Wenli Zheng*, Minyi Guo*
*Department of Computer Science and Engineering, Shanghai Jiao Tong University Shanghai, China
†Microsoft research Asia Shanghai, China
{zhang-w,midway}@sjtu.edu.cn, Ningxin.Zheng@microsoft.com, {chen-quan,lichao,zheng-wl,guo-my}@cs.sjtu.edu.cn

***Abstract*—Emerging latency-critical (LC) services often have both CPU and GPU stages (e.g. DNN-assisted services) and require short response latency. Co-locating best-effort (BE) applications on the both CPU side and GPU side with the LC service improves resource utilization. However, resource contention often results in the QoS violation of LC services. We therefore present CHARM, a collaborative host-accelerator resource management system. CHARM ensures the required QoS target of DNN-assisted LC services, while maximizing the resource utilization of both the host and accelerator. CHARM is comprised of a BE-aware QoS target allocator, a unified heterogeneous resource manager, and a collaborative accelerator-side QoS compensator. The QoS target allocator determines the time limit of an LC service running on the host side and the accelerator side. The resource manager allocates the shared resources on both host side and accelerator side. The QoS compensator allocates more resources to the LC service to speed up its execution, if it runs slower than expected. Experimental results on an Nvidia GPU RTX 2080Ti show that CHARM improves the resource utilization by 43.2%, while ensuring the required QoS target compared with state-of-the-art solutions.**

I. Introduction

Intelligent Latency-Critical (LC) services running on datacenters require consistently high accuracy and low response time to attract and retain users [22]. Many LC services have widely adopted Deep Neural Networks (DNNs), as the recent advances have made DNNs achieve human-level accuracy on various tasks (e.g., digit/image recognition, and speech recognition). Modern DNN-assisted LC services have two stages: *data preprocess* and *inference*. Since DNNs are more and more compute-demanding, heterogeneous accelerators (such as GPUs) are often used for inference, and the host (CPU) is used for data preprocessing, including decoding and data resizing [13]. The *host-accelerator interaction stage* (Memcpy) is supported by the PCI-e bus. Cloud gamings, another fast-growing LC service, also have both *host-stage* (Game logic) and *accelerator stage* (Rendering graphics).

Because LC services often experience a diurnal pattern [2] (leaving the resources under-utilized for most of time except peak hours), it is cost-effective to co-locate the LC services with low-priority, best-effort (BE) applications that have no QoS requirements. However, the co-location may bring performance penalty for LC services and lead to QoS violation, as co-scheduled applications contend for shared resources.

Quan Chen and Minyi Guo are the corresponding authors.

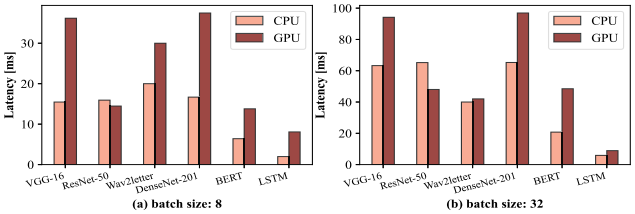


Fig. 1: The CPU/GPU execution time in one query of different LC services with different batchsize.

Prior work has recognized the problem and proposed techniques to guarantee QoS and improve resource utilization on both CPUs and accelerators [7], [8], [10], [21], [23], [26], [29], [31], [32], [34]. Existing efforts on CPU co-location generally use two approaches. The first approach disallows LC services from sharing resources with other applications to avoid interference or allows only certain co-locations based on profiling and instrumentation information [23], [32]. The second one is dynamically managing interference by throttling resources of BE applications to ensure the LC service's QoS [8], [26]. Managing performance interference on accelerators is also well-studied. On time-sharing accelerators, queuing-based methods (e.g., GrandSLAm [18] and Baymax [7]) reorder the GPU kernels. On spatial multitasking accelerators, profiling-based methods (e.g., Laius [30]) partition processing elements between LC service and BE application.

Unfortunately, prior researches are limited to the co-location on either CPU or the accelerator, respectively. They are not general enough to manage these new application scenarios where LC services have both host-stage and accelerator-stage. There are new challenges in managing heterogeneous co-location which needs to be solved urgently.

Firstly, it is hard to appropriately split the entire QoS target into two parts that maximize resource utilization. As shown in Figure 1, services from Table III spend different time on host-stage and accelerator-stage, and the same service spends different time on the two stages with different batch sizes. In this case, the appropriate QoS target quotas of the two stages vary when the load or the co-located BE applications change. A viable option is traversing all possible combinations, but this method is expensive and unsuitable for online situations.

Secondly, the independent management results in the QoS violation. For the host stage, the query may run longer than

978-1-6654-3219-1/21/$31.00 ©2021 IEEE
DOI 10.1109/ICCD53106.2021.00056

expected. If the profiling-based resource management method is adopted in the host stage, the LC service suffers from QoS violation when the latency prediction is inaccurate. Prior work [5] shows that the accuracy of profiling-based methods on CPU is around 80%. If the feedback-based resource management method is adopted, LC service suffers from QoS violation when the feedback mechanism is not timely enough. The delay caused by the feedback cannot be compensated even if sufficient resources are provided.

We therefore introduce **CHARM**, a heterogeneous runtime system that meets two objectives: satisfy QoS target for the LC service and maximize performance of all BE jobs on both CPUs and GPUs. CHARM is comprised of a *BE-aware QoS target allocator*, *a unified heterogeneous resource manager*, and a *collaborative accelerator-side QoS compensator*.

The QoS target allocator builds a heuristic method to split the QoS target of an LC service for the host-stage and accelerator-stage. The QoS allocation is determined by the LC service itself and the current BE applications on both host-side and the accelerator-side. The unified resource manager provides theoretically-grounded resource partitioning of host and accelerator resources among the co-located applications. It makes the overall resource utilization more efficient and economical while ensuring QoS of the LC service. The manager does not build complex performance models that need extensive offline profiling or recompilation. The QoS compensator adjusts the resource allocation of the accelerator stage online according to the performance of the host stage. It avoids unpredictable QoS violations caused by host-stage contention. In this work, we rely on the bandwidth reservation technique proposed in Baymax [7] to ensure that an LC query can always transfer data at full speed, thus eliminates QoS violation which is resulted from PCIe bandwidth contention.

The main contributions of CHARM are as follows:

- **Comprehensive analysis of QoS interference on heterogeneous systems -** The analysis reveals new opportunities to ensure QoS while maximizing utilization through collaborative heterogeneous resource management.
- **The design of a unified resource management solution -** We use an improved SMAC (Sequential Model-Based Optimization for General Algorithm Configuration) method that requires a small number of samples to build low-cost performance model and navigate this search space intelligently to find near-optimal unified configurations.
- **The design of online process perceptron for identifying potential QoS violation -** Once the query is perceived to be slower than expected, accelerator-side compensator allocates more resources to compensate for the lag.

II. Related Work

There have been some researches that focus on improving the resource utilization of CPU datacenters. Bubble-up [23] predicted the interference caused by CPU colocation. SMiTe [32] further extended Bubble-up to predict performance interference between applications on SMT processors. Dirigent [34] proposed a lightweight technology to build accurate performance profiles for LC tasks and adjust resource allocation accordingly based on the predicted results. There are some works using feedback mechanisms to adjust the resource division to eliminate resource contention [8], [21], [25]. When the QoS violation of the LC task is identified, the feedback scheduler dynamically adjusts the resource division of the BE task to ensure the QoS. Twig and CLITE [24], [26] adopted a black-box optimization method to divide each service's resources in the co-located server to maximize the throughput of BE tasks. These works do not consider the accelerator-side as CHARM does.

There are also works devoted to optimizing GPU resource utilization. TimeGraph [19] and GPUSync [12] used priority-based scheduling to guarantee the performance of real-time kernels. Baymax and GrandSLAm [7], [18] predicted the kernel duration and reordered the kernel based on the QoS headroom of user-facing queries. Prophet [6] designed the interference model to accurately predict the performance loss caused by resource competition in GPUs. They all assumed that the accelerator is time-sharing and non-preemptive. HSM [33] predicted the slowdown of co-located applications on spatial multitasking GPUs. However, it relies on a broad spectrum of performance event statistics not available on real system GPUs. Laius [30] focuses on partitioning to fulfill QoS of the LC job and improving utilization by leveraging resource equivalence. These works do not consider the host-side contention, and cannot improve both host and accelerator utilization, as CHARM does.

Besides, there are some works that focus on efficient co-execution of applications on GPU-based clusters and cloud servers. Mystic [28] is a framework enabling interference-aware scheduling for GPU workloads. Li *et al.* introduced a priority-based PCIe scheduling policy and the semi-QoS application management on CPU-GPU communication to improve multi-GPU throughput [20]. These works do not consider the characteristics of multi-stage tasks, and cannot improve the overall cluster resource utilization, as CHARM does.

III. Motivation and Challenges

In this section, we first investigate the efficiency of state-of-the-art scheduling policies at both host and accelerator sides on maximizing utilization and ensuring QoS of LC services in terms of tail latency. Then, we analyze why previous work fails to handle these LC services, and discuss the opportunities to use collaborative resource scheduling.

A. Investigation Setup

We co-locate a typical DNN-assisted LC service (realized by Resnet), selected from Table III, with both host-side BE applications and accelerator-side BE applications. As shown in Table I, we use three benchmarks in Parsec [3] as host-side BE applications and eight benchmarks in Rodinia [4] as GPU-side BE applications. To enable the integration between host-side and accelerator-side solutions, we adopt Clite [26] on host-side and Laius [26] on accelerator-side to manage the

TABLE I: The benchmarks used as the BE applications.

Benchmark Suite	BE Workloads
Parsec [3]	SC (streamcluster), CA (canneal), BS (blackscholes)
Rodinia [4]	BFS, B+tree, PF (pathfinder), NW, LUD HS (hotspot), MD (lavaMD), MY (myocyte)

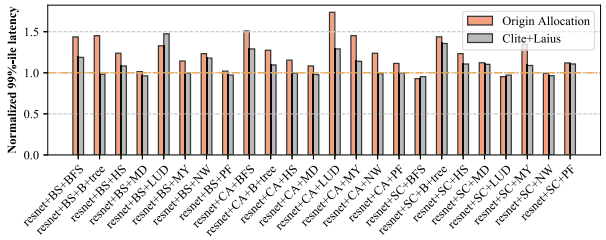


Fig. 2: The CPU and GPU execution time in one query for different DNN-assisted LC services at co-location.

shared resources between the co-located applications. Similar to prior work targeting traditional LC services, we use 150 ms as the QoS target here [7].

B. Problem of QoS Violation

When we combine Clite with Laius, the QoS target of an LC service should be divided into the QoS target on the host side for Clite, and the GPU side for Laius. As the LC services have different characteristics, there is no optimal division that fits all services. To find the best division for each co-location pair, we use the static optimal division in this experiment. For instance, if the QoS target of an LC service is 100ms, we try to use the division of (10ms, 90ms), (20ms, 80ms), (30ms, 70ms), etc. This experiment reports the latencies of the LC service with the best static QoS target division between CPU phase and GPU phase.

Figure 2 reports the normalized 99%-ile latencies of LC services (normalized to QoS targets) when they co-located with BE applications on both CPU and GPU side. We change the load every ten minutes during runtime. The load range is determined according to Figure 3 that shows the 95%-ile tail latency of Transform (CPU stage of Resnet) at different loads. And the maximum load is 80 queries per second.

In Figure 2, "Origin Allocation" means that the co-located applications contend for the shared resources with OS scheduling. The x-axis shows the co-location pairs. For instance, *resnet+BS+BFS* shows the case that the LC service *resnet* is co-located with benchmark *BS* on CPU and *BFS* on GPU.

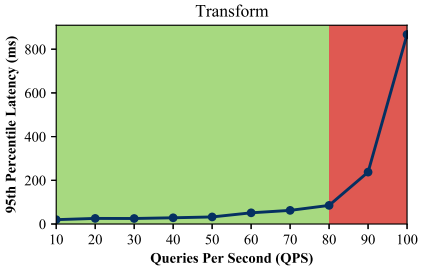


Fig. 3: The 95th percentile tail-latencies of transform (CPU stage of Resnet) at different loads.

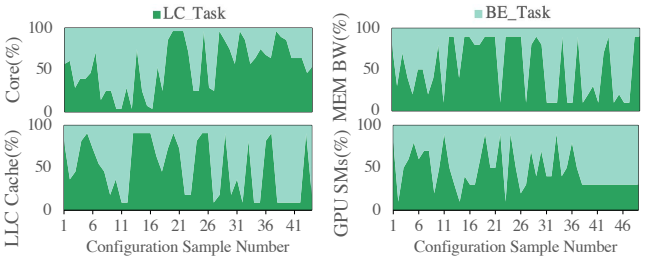


Fig. 4: The resource partitions of *Resnet* during sampling.

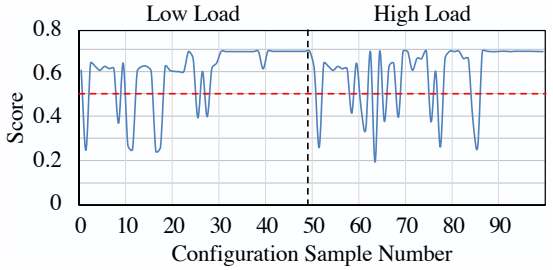


Fig. 5: The QoS scores of *Resnet* when its load changes sample 50.

As observed from Figure 2, LC services in 13 out of the 24 co-locations suffer from the QoS violation with Clite+Laius, while 20 QoS violations with OS-scheduling. To better understand the QoS violation with Clite+Laius, we take a particular mix of co-location *Resnet+BS+NW* as an example.

Figure 4 shows the amount of resources allocated to the LC service and the co-located BE applications at different sampling steps. Observed from Figure 4, even if the load of the LC service is stable, Clite+Laius needs 40 samples to find the near-optimal resource allocation in where 15 samples facing QoS violations. During the sampling and searching for the best configuration, last-level cache, memory bandwidth, and SMs in GPU vary significantly.

In addition, we also investigate the ability of Clite+Laius in adapting to the dynamic loads of the LC services. We increase the load of *Resnet* from a low load to a high load at sample 50 and report the QoS score of *Resnet* in Figure 5. Observed from this figure, Clite+Laius can tune the resource partition between the co-located applications and stabilize to a new optimal resource partition, with a relatively long tuning time.

Therefore, Clite+Laius is not applicable for heterogeneous co-location of LC services with load variation and BE jobs.

C. Opportunity and Challenges

The above investigation has shown state-of-the-art work is not applicable for co-locating LC services with varying load with BE applications. We therefore propose **CHARM**, a collaborative host-accelerator resource management system, to take advantage of this opportunity.

However, three challenges have to be resolved in CHARM.

- **It is challenging to identify QoS target divisions between CPU and GPU phases.** The decision should be based on the characteristics of the current running BE applications on the host side and the accelerator side.
- **It is challenging to identify the optimal resource allocation quickly enough for dynamic service loads.** When there are many types of to-be-allocated resources, the multi-dimensional search space is prohibitively large. CHARM explores less expensive scheduling methods.
- **It is challenging to eliminate the QoS violation when searching for the optimal resource allocation.** CHARM should identify the LC queries that spend longer time than expected quickly at runtime, and compensate the lag for ensuring the QoS.

IV. Overview of CHARM

In this section, we describe the methodology of **CHARM** that maximizes the throughput of BE jobs on both host and accelerator while guaranteeing the QoS of LC services.

Figure 6 demonstrates the design overview of CHARM. It is comprised of a *BE-aware QoS target allocator*, a *unified heterogeneous resource manager*, and a *collaborative accelerator-side QoS compensator*. As shown in Figure 6, CHARM handles LC queries and BE applications in different ways. Once an LC query is submitted, the QoS target allocator splits its QoS target to CPU side QoS target and GPU side QoS target heuristically. It then passes the divisions to the unified resource allocator as the initial sample points to search for the best resource allocation. The resource allocator identifies the optimal resource allocation that maximizes the economical throughput of BE jobs while guaranteeing the QoS of LC services. Moreover, the collaborative compensator monitors the progress of the query's CPU stage. It speeds up its accelerator-side execution if the query spends a long time on CPU than its CPU side QoS target.

Let Q represent the LC service. CHARM manages the resource allocation for Q in the following steps.

(1) The QoS target allocator builds a performance model for Q based on the characteristics of the currently running BE jobs on CPU and GPU. Based on the model, CHARM proposes a heuristic method to splits its QoS target to the CPU and GPU phases of Q. The QoS target division is passed to the unified heterogeneous resource manager as the initial sample point (Section V-A). This step significantly impacts the number of tries needed to identify the best resource allocation later.

(2) Once the QoS target of Q is split, the resource manager allocates various resource configurations (cores, memory bandwidth, LLC, SMs) to Q and the co-located BE applications on the CPU and GPU side based on the optimized SMAC algorithm. When performing the allocation, CHARM maximizes the economical throughput of BE jobs while alleviating QoS violation of Q due to resource contention (Section V-B). The challenging part is to minimize the time needed to identify the optimal allocation to adapt to dynamic loads.

(3) The accelerator-side QoS compensator monitors the progress of Q in the CPU stage. If the queries of Q run slower than expected, the compensator speeds up Q's GPU stage by allocating it more computational resources (Section V-C). The hardpoints here are quickly identifying the new computational resource quota for Q on GPU without seriously degrading the throughput of BE applications on GPU.

It is worth noting that CHARM does not need offline profiling for LC services on GPU, while prior work like Laius requires excessive profiling.

V. Design and Implementation

This section introduces the technical details of CHARM, including a BE-aware QoS target allocator, a heterogeneous resource manager and an accelerator-side QoS compensator.

A. BE-aware QoS target allocator

We first propose a BE-aware QoS target allocator, which allocates the QoS target for CPU stage and GPU stage of the LC task according to the characteristics of BE tasks on heterogeneous devices. The difficulty lies in how to solve the QoS division within limited samplings accurately .

In our preliminary evaluation, increasing resource quotas of BE tasks can effectively improve the BE task's performance. However, different BE tasks have different sensitivity to shared resources (cores, LLC, memory bandwidth in CPU and SMs in GPU). To maximize the utilization of computation resources, the shared resource usage of LC tasks should have as little impact as possible on the BE tasks' performance.

Hence, we design a heuristics searching algorithm to perform QoS division for the LC service as shown in Figure 7(a). Specifically, we initially set each resource ((cores, LLC, memory bandwidth in CPU and SMs in GPU)) quota of the LC task to their minimum resource unit, while allocating the rest resources to the BE task. To ensure the QoS of the LC task, shared resource quotas allocated to LC task should be increased. Each time, we adjust the QoS allocation to CPU-GPU stage according to the performance surface of shared resources. We respectively record the increase in QoS of the LC task ΔQoS_d and BE task performance degradation $\Delta perf_d$ in the searching step. We use Equation 1 to select the best resource allocation d^*, then adjust the resources d^* from the BE task to the LC task and perform the next loop.

$$d^* = argmax_{d \in core, LLC, BW, SM} \frac{|\Delta QoS_d|}{|\Delta perf_d|} \quad (1)$$

Finally, we get the near-optimal results that not only satisfy QoS but also try to ensure the minimum performance degradation of BE tasks. The final time of CPU stage and GPU stage are scaled up to the QoS target and used as the QoS division. The QoS division is passed to the unified heterogeneous resource manager as the initial sample. This step significantly impacts the number of samples searching for the best resource allocation later.

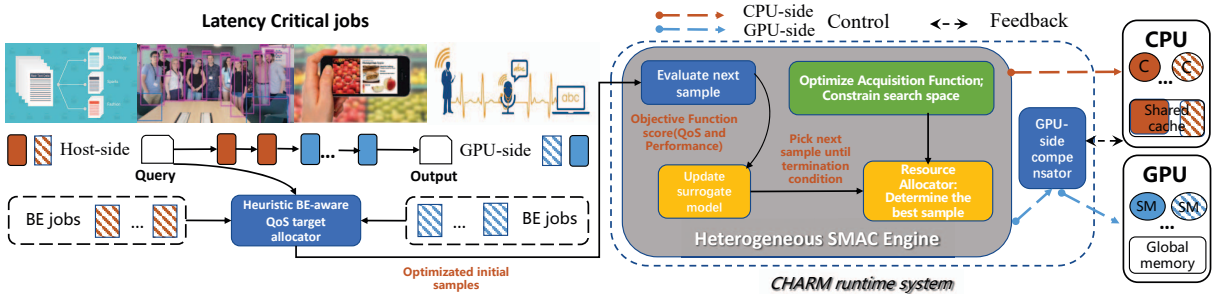


Fig. 6: Design overview of CHARM.

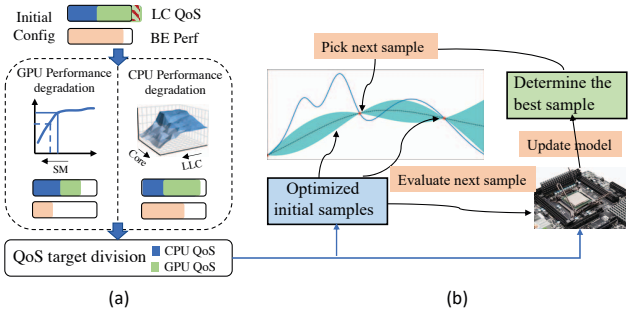


Fig. 7: (a) BE-aware QoS target allocator; (b) SMAC-based resource manager

B. SMAC-based unified heterogeneous resource manager

Once the QoS allocator obtains the QoS target division, CHARM needs to manage the resources for the co-location service on each device to improve the overall resource utilization while ensuring the QoS of the LC service. Given the large resource allocation space in heterogeneous co-locations, it is essential for our resource manager to quickly identify the best resource partition with the minimum sampling times.

CHARM uses Sequential Model-Based Optimization for General Algorithm Configuration (SMAC) to perform the resource partition as shown in Figure 7(b). However, two problems make it inappropriate to use the SMAC algorithm directly for co-located services. First of all, the SMAC algorithm selects random initial sampling points. While the approach work for simpler services, they are prone to covariant shift in heterogeneous situation, which causes frequent QoS violations in the process of sampling. Secondly, the traditional objective function in SMAC optimization returns a single value (e.g., the throughput of the system or execution time) to be maximized. CHARM cannot apply traditional SMAC since CHARM needs to satisfy multiple criteria (QoS of LC jobs and maximize performance of BE jobs).

To this end, we have made two adaptive corrections to the SMAC algorithm. For initial sampling points, CHARM carefully select them based on different strategies: (1) the even-priority strategy (all CPU tasks get equal computation resources); (2) the resource partitions achieved from the Section V-A; (3) the QoS guaranteed strategy (minimum resource quota for BE job while the remaining resources for LC job). In general, the above three configurations can better discover potential resource partitions and speed up the sampling process. As to deciding which configurations to evaluate next, we carefully design the objective function so that SMAC optimization can be applied to heterogeneous co-location.

$$Score = \begin{cases} \frac{1}{2} \times \frac{QoS_{CPU}^{target}}{QoS_{CPU}^{eval}} \times \frac{QoS_{GPU}^{target}}{QoS_{GPU}^{eval}}, \text{If QoS not meet} \\ \frac{1}{2} + \alpha \times \frac{PerfCPU_r}{PerfCPU_s} + \beta \times \frac{PerfGPU_r}{PerfGPU_s}, \text{Otherwise} \end{cases} \quad (2)$$

We design a *score function* for CHARM that assigns scores to the objection function (i.e., the objective score is assigned at the end of the period when the system is run under the given resource partition configuration). This score function guides CHARM to search in the right direction in the large configuration space. We construct a segmented objective function which considers both the QoS of the LC job and economical throughput of BE jobs. The function value is shown in equation 2, which is between 0 (worst case scenario, no LC job meets its QoS) to 1 (ideal scenario, all LC jobs meet QoS and BE jobs achieve the same performance as if they run in isolation). The first objective is to meet the QoS target on both CPU and GPU. QoS^{target} is the QoS of LC job, QoS^{eval} is the latency of the LC job under current resource configuration. The score will attain a value less than 0.5 if all LC jobs suffer from QoS violations on any device regardless of the performance of BE jobs. Only when the score is more than 0.5, the second objective function is considered.

The second objective in equation 2 is to maximize the overall system throughput of BE jobs, where $perf_r$ is the throughput of BE jobs during sampling, $perf_s$ is the throughput of solo-run BE jobs. Considering the price for renting CPUs and GPUs varies greatly, we perform a weighted summation of throughput on both CPU and GPU, where $\alpha = 0.05$ and $\beta = 6$ are correlated with the rental prices of CPU and GPU.

$$\begin{aligned} &Object = MAX(a(Score(R))) \\ &\text{Constraint-1:} \quad 0 \le r_{ij} \le R_j \qquad 0<i \le n, 0<j \le m \\ &\text{Constraint-2:} \quad \sum_{i=1}^{n} r_{ij} \le R_j \qquad 0<j \le m \end{aligned} \quad (3)$$

To speed up the search process of SMAC optimization, CHARM uses a pruning strategy based on optimization problem shown in Equation 3 to constrain the search space, which can remove most "undesirable" resource allocations. Suppose that n tasks need to be deployed with m kind of resources. We

formulate the searching processing as shown in equation 3. Let R is a matrix with n rows and m columns, where r_{ij} represents the share of the j-th resource owned by the i-th task. R_j represents the total amount of resource j. The optimization problem contains two kinds of constraints. First, for each task, the maximum quota of each task does not exceed the total amount of the resource. Secondly, for each resource, the sum of the quota of all tasks cannot be larger than the total amount. Meanwhile, in order to find the global optimal resources partition, we calculate the final score of the resources partition using the acquisition function [17].

Algorithm 1: SMAC-based resources manager

Input: Initialize configuration samples set $D_k = ((x_1, Score(x_1)),\cdots,(x_k, Score(x_k)))$, Maximum interations N, Perfermance target(Pt)

1 Run the system with the inital configuration D_k.
2 **for** *i in k to N* **do**
3 $Score \leftarrow Fitmodel(D_i)$
4 Caculate the acquisition function $a(Score(D_i))$.
5 $x^* = argmax_{x \in D_i} a(Score(D_i))$
6 $Score(x^*) \leftarrow eval(x^*)$
7 $D_{i+1} \leftarrow (D_i, (x^*, Score(x^*)))$
8 **if** Pt *is met* **then**
9 break
10 **Return** the best resources configuration x^*

In summary, CHARM schedules the resources of the CPU-GPU colocation services according to Algorithm 1. CHARM creates an initial sampling configuration set and performs a looped sample. CHARM makes the following steps in each sample: 1) Construct a random forest model($Fitmodel$) based on the sampling configurations set to predict the performance contention of LC and BE tasks (line 3). 2) Compute the target optimization to solve the next optimal sampling configuration x^* in the target space (line 4-5). 3) Use the resource partitions corresponding to x^* to evaluate the contention services and return the performance score (line 6). 4) Add the resource partitions and the corresponding performance score into the sampling configuration set for the next loop (line 7).

C. Accelerator-side QoS compensator

During the execution of a LC query Q, the accelerator-side QoS compensator monitors the progress of Q in the CPU stage. If Q run slower than expected (i.e. sudden spikes in workloads or other contention which cannot be explicitly managed), the compensator speeds up Q's GPU stage by allocating it more computational resources. The hard points in this step are quickly identifying the new computational resource quotas for Q on GPU without seriously degrading the throughput of BE applications on GPU.

The compensator periodically checks whether it runs slower than expected in the CPU stage. Let T_{cpu} and $T^{'}_{gpu}$ represent the actual CPU duration of Q with the current resource quotas and the GPU duration of Q with new identified computational resource quotas on GPU, respectively. $T_{save} = T_{cpu} - QoS^{target}_{CPU}$ calculates the increased duration of executing the CPU stage. If T_{save} is larger than the reduced GPU duration of Q with the new resource quota, which means Equation 4 is satisfied, Q is able to meet the QoS target.

$$T_{cpu} - QoS^{target}_{CPU} \geq (QoS^{target}_{GPU} - T^{'}_{gpu}) \quad (4)$$

Based on Equation 4, the compensator is able to identify the new "just-enough" computational resource quota on GPU-side for Q. In the equation, $T^{'}_{gpu}$ can be achieved from the performance model, QoS^{target}_{CPU} and QoS^{target}_{GPU} are from the Section V-A, T_{cpu} is measured at runtime directly. Once the new quota is identified, the computational resource quotas allocated to BE jobs are also updated simultaneously. If the CPU progress of Q satisfies QoS with the new quota, the resource quota allocated to Q rolls back to its original quota. In this way, CHARM ensures Q completes before the QoS target, and minimizes the resource used by it. In this work, we rely on the process pool technique proposed in Laius [30] to enable the resource reallocation on GPUs.

TABLE II: Hardware and software specifications.

	Specification
Hardware	Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz GeForce RTX 2080Ti
	Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz NVIDIA DGX-2 with 16 Tesla V100s-SXM3
Software	Ubuntu 16.04.5 LTS with kernel 4.15.0-43-generic PyTorch 1.8.1 CUDA SDK 10.0 CUDNN 7.4.2

VI. Evaluation

A. Experiment Setup

In our experiments, we choose four representative LC services with both CPU stage and GPU stage from NLP and CV scenarios to Gaming in Table III, three BE jobs on CPU from Parsec [3] and four BE jobs on GPU from Rodinia [4], in which we categorize the first type as compute-intensive workloads (HS) and the second type as memory-intensive workloads (B+tree, MD, BFS). To represent a production environment, we provide an open-loop asynchronous workload generator to simulate users' requests. The arrival time of user requests follows an exponential distribution.

The experiments are carried out on a 28-core server equipped with one Nvidia GPU RTX 2080Ti. The detailed setups are summarized in Table II. Note that CHARM does not rely on any special hardware features of 2080Ti and is easy to be set up on other spatial multitasking accelerators.

Throughout our experiments, the QoS target is defined as the 99%-ile latency. The overall QoS target for the DNN-assisted LC job is 150ms which is widely accepted by users, while the QoS of Gaming is 60 FPS(frame per second). The throughput is represented by IPS (instructions per second). Besides, the cost-oriented throughput of BE jobs is calculated as $\alpha * perf_{cpu} + \beta * perf_{gpu}$, which explained in Section V-B. We use taskset interface in OS and Nvidia Volta Multi-Process Service (MPS) to enable the resource allocation. Besides, we use the process pool proposed in Laius [30] to enable the resource reallocation.

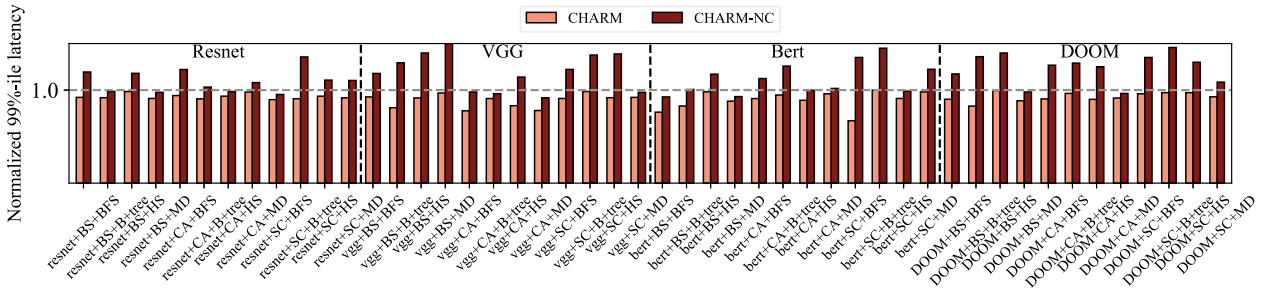


Fig. 8: The 99%-ile latency of LC job normalized to the QoS target with CHARM and CHARM-NC.

TABLE III: The DNN models used as the LC services.

Neural model	Scenario	Dataset
VGG-16 [27]	Style transfer	COCO& WikiArt
ResNet-50 [15]	CV	ImageNet
Wav2letter [9]	Speech Recognition	LibriSpeech
DenseNet-201 [16]	CV	ImageNet
BERT [11]	NLP	GLUE
LSTM [14]	Translation	Real and Fake News
DOOM [1]	Open-sourced Game	&

B. Ensuring the QoS with CHARM

We evaluate the effectiveness of CHARM in ensuring the QoS target of LC services. Figure 8 presents the 99%-ile latency of LC services normalized to their QoS target when they are co-located with BE jobs on CPU and GPU. There are overall $4 \times 3 \times 4 = 48$ co-location pairs (4 LC jobs, 3 BE jobs on CPU and 4 BE jobs on GPU). Observed from this figure, CHARM ensures the QoS of LC jobs. On the contrary, the original allocation and Clite+Laius allocation in Figure 2 as mentioned in Section 3 results in QoS violation of LC jobs.

CHARM monitors the progress LC queries at CPU-side and allocates more GPU resources to a slow query to compensate for the delay. To evaluate this design choice, we also verify the effectiveness of the collaborative accelerator-side compensator. We disable the collaborative compensation part of CHARM and test the system. Figure 8 also presents the 99%-ile latency of LC services at co-location in CHARM without the compensator as CHARM-NC, a system that disables the accelerator-side compensator.

Figure 8 also shows the existence of QoS violation in CHARM without the compensator as CHARM-NC. Observed from Figure 8, LC services in 33 out of the 48 co-locations suffer from QoS violation in CHARM-NC. For instance, *vgg* suffers from up to 1.5X QoS violation when it is colocated with *BS+MD*. The QoS violation is due to the cost of optimization algorithm and the fluctuation of sampling points during load changes. It implies that the compensator is able to efficiently reduce the QoS violation due to the configuration search at CPU side.

We further compare CHARM with Clite+Laius, a straightforward resource management solution using the method in Clite on CPU co-colocation for the host stage and the method

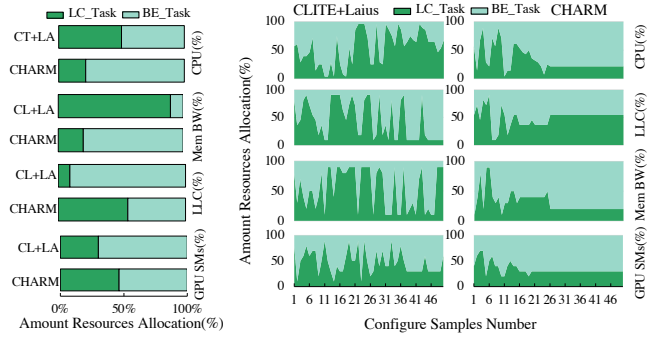


Fig. 9: The snapshot of resource allocation for a particular mix of co-located jobs (Left). The resource allocation over time of a particular load setting where Clite+Laius does not meet the QoS while CHARM does (Right).

in Laius on accelerator co-colocation for the accelerator stage. Adopting Clite+Laius, the end-to-end QoS target of an LC service is first divided into the QoS for the host stage, and the QoS for the accelerator stage. After that, the resources on the host and accelerator are managed independently to fulfill the host QoS target and the accelerator QoS target. And then we choose the static optimal pair as the baseline solution.

The left subfigure in Figure 9 provides the snapshot of the resource allocation for a particular mix of co-located jobs (resnet with BS and MD as the BG jobs). For this mix, both CHARM and Clite attain the QoS targets for the LC job as shown in Figure 8 and Figure 2, but the resource allocations are different for all jobs.

The right subfigure in Figure 9 shows the resource allocation over time of a particular load setting where Clite+Laius does not meet the QoS, while CHARM does. The co-location corresponds to the co-location of resnet, blackscholes and NW. These results provide a deeper view into why Clite+Laius approach cannot meet QoS targets even after 50 configuration samples, while CHARM meets the QoS for the LC job in less than 26 configuration samples and stabilizes. This is because the Gaussian processing-based Bayesian optimization needs much more tries than the SMAC method used in CHARM to find the appropriate resource allocation. LC queries suffer

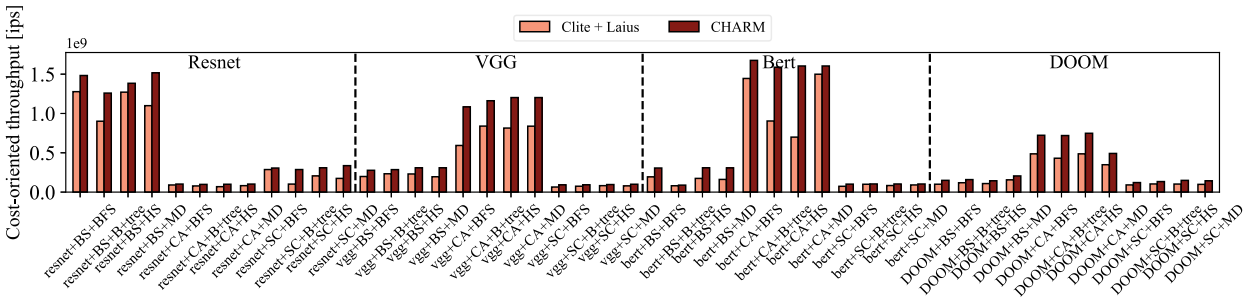
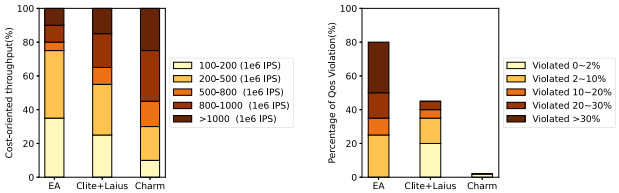


Fig. 10: The cost-oriented throughput of BE jobs at co-location with CHARM and Clite+Laius.



(a) Cost-oriented throughput at different level

(b) Percentage of QoS violation

Fig. 11: The scale-out results on DGX2 servers.

from QoS violation during the search with Clite+Laius. The potential QoS violation during the search with CHARM is eliminated by the accelerator-side compensator.

C. Cost-oriented throughput

In this subsection, we evaluate the effectiveness of CHARM in maximizing the cost-oriented throughput. Figure 10 shows the cost-oriented throughput of BE applications at co-location with CHARM and Clite+Laius. Observed from this figure, BE applications in the $3 \times 3 \times 8 = 72$ co-locations (including Resnet50, Vgg16 and Bert) achieve higher throughput with CHARM than Clite+Laius. Specifically, CHARM improves the cost-oriented throughput of BE applications by 43.2% on average compared with Clite+Laius.

CHARM's evaluation demonstrates its effectiveness, robustness, and practical feasibility across a range of scenarios and workloads. CHARM's LC job performance is better than the previously proposed solutions such as Clite and Laius, and BE jobs' throughput is more than 40% in many cases. CHARM can co-locate a set of resource-hungry and the latency-critical job while meeting their QoS targets, and can still provide high performance to BE jobs, in comparison the competing techniques (including Clite and genetic algorithm approach).

D. Scale-out Study

To further evaluate the effectiveness of CHARM in GPU datacenters, we evaluate CHARM with four DGX2 servers which equipped with V100 GPUs (Table II). We evenly use one GPU server for each type of LC services (Resnet, vgg, Bert, DOOM) with 3 BE jobs on CPU (Parsec) and 8 BE jobs on GPU (Rodinia) in Table I. As shown in Figure 11, *CHARM* achieves higher cost-oriented throughput and lower QoS violation compared with *Even Allocation(EA) policy* and *Clite+Laius*. On average, CHARM improves the throughput by 39.2% compared with Clite+Laius. As shown in Figure 11(b), 40.6% of LC jobs suffer from severe QoS violations(20% degradation) with EA while 24.7% of LC jobs exceed QoS (more than 2% degradation) with Clite+Laius. On the contrary, CHARM can maintain the QoS target, and less than 2.3% of LC jobs suffer from insignificant QoS violations (less than 2% degradation).

E. Overhead of CHARM

As described in Section 5, CHARM dynamically allocates computation resources to LC and BE tasks based on the current system resource usage at runtime. Among them, the sampling process of running the SMAC algorithm once consumes less than 10ms. In addition, the compensator needs to reallocate GPU computation resources according to the runtime conditions of the CPU side, and it takes less than 0.1ms for to perform a search. In general, the overall overhead introduced by CHARM does not exceed 7% of the LC job's QoS.

VII. Conclusion

We propose CHARM, a collaborative host-accelerator resource management runtime system that considers the interference of both the host and the accelerator side comprehensively. CHARM uses the QoS target allocator to assign the QoS target of the host side and accelerator side respectively, and uses improved SMAC to allocate appropriate resources (cores, SMs, Cache, Memory bandwidth, etc) for the LC job and BE job. In addition, CHARM also uses a compensator to dynamically adjust the resources allocated to the LC jobs. The experimental results show that CHARM improves the utilization by 43.2% without incurring QoS violations.

Acknowledgment

This work is partially sponsored by the National Natural Science Foundation of China (62022057, 61832006, 61632017, 61872240).

REFERENCES

[1] "Doom," https://github.com/leereilly/games.

[2] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.

[3] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chip-multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, vol. 2011, 2009, p. 37.

[4] 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 *IISWC*. Ieee, 2009, pp. 44–54.

[5] Q. Chen, Z. Wang, J. Leng, C. Li, W. Zheng, and M. Guo, "Avalon: towards qos awareness and improved utilization through multi-resource management in datacenters," in *ICS*, 2019, pp. 272–283.

[6] 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 *ASPLOS*, 2017, pp. 17–32.

[7] Q. Chen, H. Yang, J. Mars, and L. Tang, "Baymax: Qos awareness and increased utilization for non-preemptive accelerators in warehouse scale computers," in *ASPLOS*. ACM, 2016, pp. 681–696.

[8] S. Chen, C. Delimitrou, and J. F. Martínez, "Parties: Qos-aware resource partitioning for multiple interactive services," in *ASPLOS*, 2019, pp. 107–120.

[9] R. Collobert, C. Puhrsch, and G. Synnaeve, "Wav2letter: an end-to-end convnet-based speech recognition system," *arXiv preprint arXiv:1609.03193*, 2016.

[10] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *ACM SIGPLAN Notices*, vol. 48, no. 4. ACM, 2013, pp. 77–88.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[12] G. A. Elliott, B. C. Ward, and J. H. Anderson, "Gpusync: A framework for real-time gpu management," in *RTSS*. IEEE, 2013, pp. 33–44.

[13] S. García, J. Luengo, and F. Herrera, *Data preprocessing in data mining*. Springer, 2015, vol. 72.

[14] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[16] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "Densenet: Implementing efficient convnet descriptor pyramids," *arXiv preprint arXiv:1404.1869*, 2014.

[17] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[18] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, "Grandslam: Guaranteeing slas for jobs in microservices execution frameworks," in *Eurosys*, 2019, pp. 1–16.

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

[20] C. Li, Y. Sun, L. Jin, L. Xu, Z. Cao, P. Fan, D. Kaeli, S. Ma, Y. Guo, and J. Yang, "Priority-based pcie scheduling for multi-tenant multi-gpu systems," *IEEE Computer Architecture Letters*, vol. 18, no. 2, pp. 157–160, 2019.

[21] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *ISCA*, 2015, pp. 450–462.

[22] J. Mars and L. Tang, "Whare-map: Heterogeneity in " homogeneous" warehouse-scale computers," in *ISCA*, 2013, pp. 619–630.

[23] 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 *Micro*, 2011, pp. 248–259.

[24] R. Nishtala, V. Petrucci, P. Carpenter, and M. Sjalander, "Twig: Multi-agent task management for colocated latency-critical cloud services," in *HPCA*. IEEE, 2020, pp. 167–179.

[25] 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 *IPDPS*. IEEE, 2020, pp. 718–727.

[26] T. Patel and D. Tiwari, "Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers," in *HPCA*. IEEE, 2020, pp. 193–206.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

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

[29] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 607–618.

[30] W. Zhang, W. Cui, K. Fu, Q. Chen, D. E. Mawhirter, B. Wu, C. Li, and M. Guo, "Laius: Towards latency awareness and improved utilization of spatial multitasking accelerators in datacenters," in *ICS*, 2019, pp. 58–68.

[31] W. Zhang, N. Zheng, Q. Chen, Y. Yang, Z. Song, T. Ma, J. Leng, and M. Guo, "Ursa: Precise capacity planning and fair scheduling based on low-level statistics for public clouds," in *49th International Conference on Parallel Processing-ICPP*, 2020, pp. 1–11.

[32] Y. Zhang, M. A. Laurenzano, J. Mars, and L. Tang, "Smite: Precise qos prediction on real-system smt processors to improve utilization in warehouse scale computers," in *Micro*. IEEE, 2014, pp. 406–418.

[33] X. Zhao, M. Jahre, and L. Eeckhout, "Hsm: A hybrid slowdown model for multitasking gpus," in *ASPLOS*, 2020, pp. 1371–1385.

[34] H. Zhu and M. Erez, "Dirigent: Enforcing qos for latency-critical tasks on shared multicore systems," in *ASPLOS*, 2016, pp. 33–47.