AlloX: Allocation across Computing Resources for Hybrid CPU/GPU clusters

Tan N. Le[1,2], Xiao Sun[1], Mosharaf Chowdhury[3], and Zhenhua Liu[1]
[1]Stony Brook University, [2]SUNY Korea, [3] University of Michigan

1 MOTIVATION

GPUs are considered as the accelerators for CPUs. We call these applications GPU applications. Some machine learning frameworks like Tensorflow support their machine learning (ML) jobs running either on CPUs or GPUs. Nvidia claims that Titan GPU K80 12GB can speed up 5-10x on average. Although GPUs offer the advantages on performance, they are very expensive. For example, a GPU K80 roughly costs $4000 while an Intel Xeon E5 Quad cores costs $350.

The coexist of traditional CPU and GPU applications urges cloud computing operators to build hybrid CPU/GPU clusters. While the traditional applications are executed on CPUs, the GPU applications can run on either CPUs or GPUs. In the CPU/GPU clusters, how to provision the hybrid CPU/GPU clusters for CPU and GPU applications and how to allocate the resources across CPUs and GPUs?

Interchangeable resources like CPUs and GPUs are not rare in large clusters. Some network I/O cards like wireless, ethernet, infinityband with different bandwidths can also be interchangeable.

In this paper, we focus on CPU/GPU systems. We develop a tool that estimates the performance and resource for an ML job in an online manner (§2). We implement AlloX system that supports resource allocation and places applications on right resources (CPU or GPU) to maximize the use of computational resource (§3). The proposed AlloX policy achieves up to 35% progress improvement compared to default DRF [2]. We build a model that minimizes the total cost of ownership for CPU/GPU data centers (§4).

2 PERFORMANCE & RESOURCE ESTIMATION

Job performance is often unknown before it actually finishes. The performance depends on the job itself, the runtime environment, and the allocated resources. Fortunately, ML jobs are iterative and predictable. To pick the best resource configurations for an ML job, the tool uses the shorter versions of the job to estimate the completion times and necessary resource usage. Unlike Ernest [4] and CherryPick [1], we do not estimate the number of nodes for a job. Instead, we estimate the job completion times and the resource usage.

Figure 1a shows that the completion times of ML jobs linearly increase when the number of batches goes up. This gives us the confidence to create a shorter version (a small number of batches or small datasets) to estimate the performance of the full job. We run 2 short versions (5% and 10% of the number of batches) of each job and use a linear model to estimate the completion times of the full job. Figure 1b shows that the simple approach works well as the errors are below 7%.

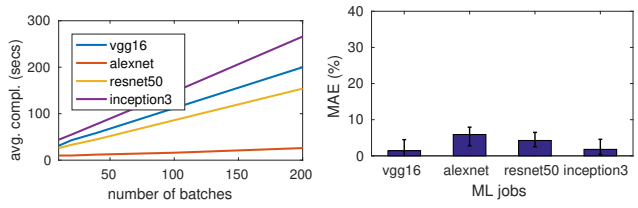

Figure 1: (a) The completion times of jobs linearly increase with respect to number of batches. (b) Running 2 short versions of a job can predict the job completion times under 7% error.

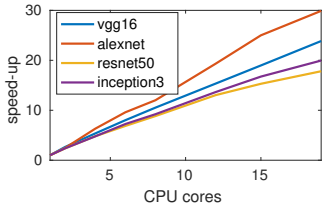

Figure 2: Using more CPU cores can improve the performance of ML jobs significantly .

Figure 2 shows the number of CPU cores plays an important role in ML jobs. The speed-up is computed based on how much faster a job running on multiple CPU cores v.s. running the job on a single CPU core is. As the performance significantly increases with larger CPU cores, we should pick the largest number of CPU cores possible to save other resources.

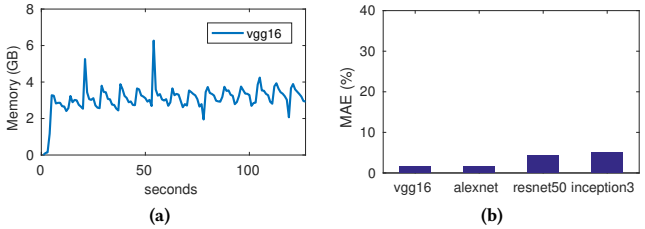

Figure 3: (a) The memory usage of an ML job has a similar pattern overtime (b) Averaging the memory usage of the short versions can estimate the memory usage average of the job at high accuracy.

Figure 3a shows memory usage of VGG16. As ML jobs are iterative, the memory usage maintains the similar pattern over time. So, we can use the short versions of ML jobs to estimate the average memory usage. Figure 3b shows that we can predict the memory usage well (below 5.5% error) using the short versions (25x shorter).

We do not contribute on picking the best number of GPUs for a job. A GPU is often powerful enough to carry on a single ML job and multiple GPUs are not efficient as the communication can be

SIGMETRICS'18 WIP, Irvine, California, USA
2018.

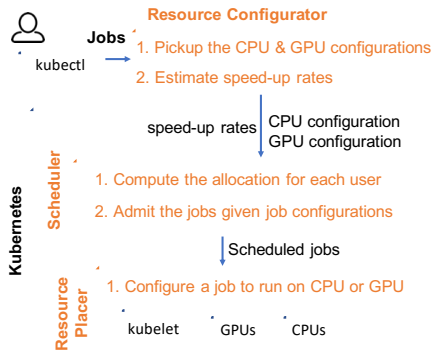

Figure 4: The AlloX system picks the resource configurations for each job, allocates resources to each users, and finally places the jobs on CPU or GPU for execution.

the bottleneck. If picking the number of GPUs for a job is necessary, similar approaches like Ernest [4] or CherryPick [1] can be used.

3 ALLOX: RESOURCE ALLOCATION SYSTEM

To the best of our knowledge, AlloX is the first interchangeable resource allocation system. We build AlloX based on Kubernetes (Figure 4). There are three main components: *Resource Configurator*, *Scheduler*, and *Resource Placer*. Resource Configurator picks the best CPU and GPU configurations for GPU jobs (§2). It also estimates the speed-up rate of a job when it runs on GPU v.s. CPU. Scheduler computes the resource allocation for each user based on their resource demand and speed-up rates. Resource Placer executes the scheduled jobs on the right resource (CPU or GPU).

AlloX policy. We implement AlloX policy that maximizes the computational power while maintains fairness among users. Each user is given a budget to purchase CPU and GPU to maximize their progresses. The policy keeps varying the price on GPU and fixes the price on CPU at 1. Some small speed-up rate users prefer to purchase CPUs while others prefer GPUs. The optimal price of GPU is decided when the policy maximizes the use of both CPU and GPU, i.e. their normalized usages are equal.

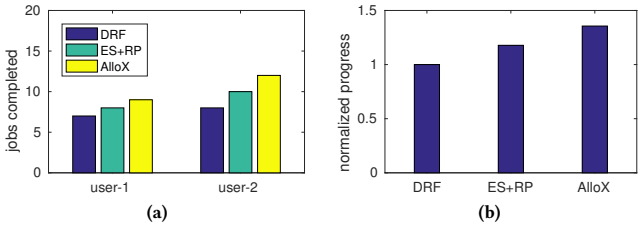

Figure 5: AlloX provides the most progress with 35% more than DRF and 15% more than ES+RP.

Evaluation. We evaluate AlloX on a small cluster with 88 CPU cores, 240 GB RAM, and 4 Nvidia k80 GPUs on Chameleon. AlloX is compared with DRF without Resource Placer and Equal Share with Resource Placer (ES+RP). There are 2 users: user-1 submits ML jobs with the speed-up rate 2.588 and user-2 submits ML jobs with the speed-up rate 1.258. Jobs are queued up at the beginning, and we count the number of jobs completed at 10 minutes. The total progress is the make-span of all completed jobs if they run on CPUs sequentially. In DRF, all ML jobs are submitted to GPUs and CPUs are not used, so it has the worst performance. In ES+SP, resources are equally shared and jobs are placed in CPUs or GPUs in the best-effort manner. Figure 5a shows AlloX is better than ES+SP on each user, and still provides fairness as users are given the same budget to purchase their own resources. Figure 5b shows AlloX provides the most progress with 35% more than DRF and 15% more than ES+RP.

4 PROVISIONING CPU/GPU CLUSTERS

We extend the model in our previous work [3] to have GPU workload and servers. The model has both power demand and supply for a data center. Power demand includes workload demand and the cooling system. At power supply side, there are renewable generations, an electricity grid, and non-renewable generations. The goal is to maximize the total cost of operating and capacity planning for a data center over several years.

In addition to the traditional workload [3], we model GPU interactive jobs and GPU batch jobs. The key modification is that GPU jobs can be executed on either CPU or GPU nodes. There are the transfer rates for GPU interactive job k and GPU batch job l to run on CPU nodes. To process the GPU workloads, we add GPU servers to the model.

Evaluation. We set up the simulation based on the setting of an HP EcoPOD data center. The detailed setting can be found in [3]. The simulation uses HP trace for CPU workload and the Microsoft Azure workload as GPU workload. GPU servers are 5x more expensive than CPU servers. The transfer rate from GPU to CPU is 32. We compare 2 approaches: CPU and GPU can be interchanged (proposed) and CPU and GPU cannot be interchanged (traditional). The ratios of CPU to GPU workload are varied from 1 to 10 times as CPU workload is often dominant. Figure 6 shows that the proposed tool saves the costs up to 8% for GPU nodes .

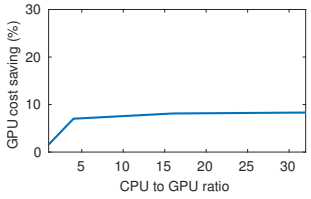

Figure 6: Using CPU nodes reduces the costs for GPU nodes up to 8%.

5 ACKNOWLEDGMENTS

This research is supported by NSF grants 1617698, 1717588, 1730128, 1563095, 1617773, 1629397, and was partially funded by MSIP, Korea, IITP-2017-R0346-16-1007.

REFERENCES

[1] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *NSDI*, volume 2, pages 4–2, 2017.

[2] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, 2011.

[3] T. N. Le, Z. Liu, Y. Chen, and C. Bash. Joint capacity planning and operational management for sustainable data centers and demand response. In *Proceedings of the Seventh International Conference on Future Energy Systems*, page 16. ACM, 2016.

[4] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *NSDI*, pages 363–378, 2016.