2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom) | 979-8-3503-3982-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/CloudCom59040.2023.00046

HCoop: A Cooperative and Hybrid Resource Scheduling for Heterogeneous Jobs in Clouds

Jinwei Liu*, Rui Gong†, Wei Dai‡, Wei Zheng§, Ying Mao¶, Wei Zhou‖, Feng Deng**
*Department of Computer and Information Sciences, Florida A&M University, Tallahassee, FL 32307, USA
†Department of Informatics and Mathematics, Mercer University, Macon, GA 31207, USA
‡Department of Computer Science, Purdue University Northwest, Hammond, IN 46323, USA
§Department of Civil and Environmental Engineering, Jackson State University, Jackson, MS 39217, USA
¶Department of Computer and Information Science, Fordham University, Bronx, NY 10458, USA
‖Center For Water Resources, Florida A&M University, Tallahassee, FL 32307, USA
**School of Automation, Beijing Information Science & Technology University, Beijing, China
*jinwei.liu@famu.edu, †gong_r@mercer.edu, ‡weidai@pnw.edu, §wei.zheng@jsums.edu,
¶ymao41@fordham.edu, ‖wei.zhou@famu.edu, **dengfeng@bistu.edu.cn

***Abstract*—Heterogeneous workloads composed of long batch jobs and short term jobs are increasingly common in modern datacenters, which poses a challenge to resource scheduling for improving throughput and resource utilization. Achieving high resource utilization and throughput is important to the cloud provider for achieving high revenue. To address this challenge, we propose HCoop, a cooperative and hybrid resource scheduling for high resource utilization and throughput in clouds. HCoop first uses the machine learning algorithm to classify jobs into two categories (long jobs and short jobs) based on the extracted features. Then, it allocates the regular virtual machines (VMs) to tasks of long jobs with higher SLO (Service Level Objective) availability guarantee, and selectively allocates the spot instances to tasks of short jobs for improving the overall resource utilization while ensuring the SLO availability for short jobs. Also, HCoop leverages the complementary of tasks' requirements on different resource types and the heterogeneity of jobs/tasks in job/task size, and it packs complementary tasks whose demands on multiple resource types are complementary to each other and allocates them to a VM to further increase the resource utilization. Extensive experimental results based on a real cluster and Amazon EC2 cloud service show that HCoop achieves high resource utilization and throughput compared to existing strategies.**

***Index Terms*—scheduling, resource utilization, heterogeneity, throughput, clouds**

I. Introduction

Cloud providers such as Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP) aim to reduce their datacenter costs and maximize their revenue. Low resource utilization is one of the key impediments for providers to achieve this goal because providers have to add more physical resources to meet increasing workload demands [1]. Although the elastic and on-demand nature of cloud computing supports users' dynamic and fluctuating demands with minimal management overhead, users' jobs are typically allocated enough resources to meet their expected peak resource demand which may rarely occur, resulting in resource wastage [1, 2]. Low resource utilization is common in existing cloud systems such as Amazon EC2 and production cluster at Twitter [3, 4].

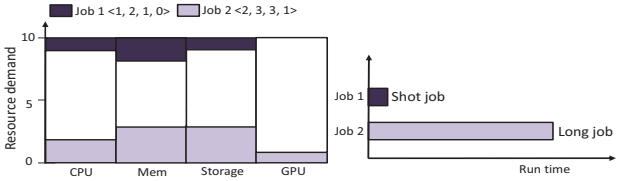


Fig. 1: An example of job heterogeneity in cloud datacenters.

The resources and workloads in production environment can be both heterogeneous and dynamic [5–7]. Production datacenters usually run vast number of applications with diverse characteristics [8], and they encounter increasingly heterogeneous workloads composed of long batch jobs (e.g., data analytics) and short jobs (e.g., operations of user-facing services) [6, 9–11]. In Microsoft production clusters, the durations of tasks can vary from a few milliseconds to tens of thousands of seconds [11]. Figure 1 shows an example of job heterogeneity in job size and resource requirements in cloud datacenters, and the resource wastage caused by resource fragmentation occurs if the a worker is allocated to Jobs 1 and 2. The heterogeneity and dynamicity of workloads not only challenges the scheduling efficiency (e.g., performance improvement on latency and throughput), but also poses a challenge for improving resource utilization [6, 12]. Although recent schedulers have improved scheduling efficiency, the datacenter resource utilization is still low, which is mainly caused by the resource under-utilization within containers such as virtual machines.

Many large cloud providers offer at least two types of virtual machines: regular (on-demand or reserved) instances and lower-priority *spot* instances (or *preemptible* instances) [13]. The regular instances allocated to users' jobs/tasks cannot be preempted, but the spot instances can be preempted/evicted at any time, and thus the price of spot instances is lower than regular instances. To satisfy regular VM customers, datacenters must be sized to handle peak VM demand. But this leads to idle datacenter capacity most of time, and the resources of spot instances that can increase the revenue for

2380-8004/23/$31.00 ©2023 IEEE
DOI 10.1109/CloudCom59040.2023.00046

cloud providers would be wasted.

To address this problem, in this paper, we aim to develop HCoop, a cooperative and hybrid resource scheduling for high resource utilization and throughput in clouds. Our goal is to simultaneously improve the resource utilization and throughput. HCoop outperforms previous schedulers in that it can well handle the scheduling of heterogeneous jobs, and simultaneously achieve high resource utilization and throughput by leveraging the idle spot instances and the complementary of tasks' requirements on different resource types for reducing resource fragmentation. We summarize the contribution of this work below.

- We propose HCoop, a cooperative and hybrid resource scheduling for high resource utilization and throughput in clouds. HCoop allocates the regular virtual machines to tasks of long jobs with higher SLO availability guarantee, and selectively allocates the spot instances to tasks of short jobs for improving the overall resource utilization while ensuring SLO availability for short jobs.
- HCoop leverages the complementary of tasks' requirements on different resource types and the heterogeneity of jobs/tasks in job/task size, and it packs complementary tasks whose demands on multiple resource types are complementary to each other and allocates them to a VM to further increase the resource utilization.
- HCoop extracts different types of features (i.e., job related features and system related features), and it classifies jobs into two categories (long jobs and short jobs) by predicting the execution time of jobs based on the extracted features.
- We conduct workload analysis based on the real trace data from a large production cluster, and the analysis results demonstrate that the production cluster has low resource utilization.

The remainder of this paper is organized as follows. Section II shows the workload analysis based on the production workloads. Section III describes the system model used in this paper. Section IV presents the system design of HCoop. Section V presents the performance evaluation for HCoop. Section VI reviews the related work. Section VII concludes this paper with remarks on our future work.

II. Workload Analysis

To verify that workloads in production environments are heterogeneous, we conducted workload analysis.

A. Job Heterogeneity

To understand job heterogeneity, we chose the production workloads Facebook 2009 (FB09) [14], Facebook 2010 (FB10) [14], Google cluster trace data [15], Yahoo cluster trace [16], and Cloudera Hadoop workload (Cloudera-b and Cloudera-c) [14] from Facebook, Google, Yahoo and Cloudera, and analyzed the distribution of jobs and the distribution of jobs' resource consumption based on job size. The Facebook workload comes from historical Hadoop traces on a 600-machine cluster at Facebook. The original

Fig. 2: Distribution of short jobs and long jobs in various commercial workloads.

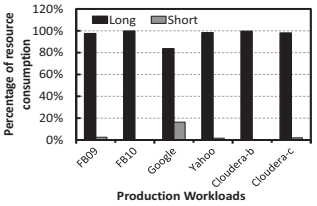


Fig. 3: Distribution of resource consumption of short jobs and long jobs in various commercial workloads.

trace spans 6 months from May 2009 to October 2009, and contains roughly 1 million jobs. The Yahoo cluster workload contains the HDFS audit logs that contain information about HDFS file access. Figure 2 and Figure 3 show the distribution of jobs and resource usage on job size in various commercial systems. The sizes of jobs vary significantly as shown in Figure 2, and long jobs consume a large portion of resource as presented in Figure 3.

III. System Model

In this section, we first introduce some concepts and assumptions, then we introduce our system model. Finally, we present our proposed algorithm for improving resource utilization.

A. Concepts and Assumptions

In a cloud system, a job is supposed to be split into m tasks, and the tasks are allocated to workers based on the their requirements on resources. We assume each worker has a buffer queue which is used for queueing tasks when a worker is allocated to more tasks than it can run concurrently. *Throughput* is the total number of jobs that complete their execution per time unit.

B. The Optimization of the Resource Utilization and Latency

Problem Statement: Given a certain amount of resources (e.g., CPU, MEM, etc.) in terms of VMs, resource demands of each job, and resource capacity constraints of VMs, how to allocate the VM resources to the heterogeneous jobs to achieve higher resource utilization and throughput?

Resource scheduling in cloud computing is an NP-hard problem and has a high computational complexity [17–21]. Thus, we propose a heuristic method called HCoop. HCoop first accurately estimates the run time of jobs [22], and it then classifies the jobs into long and short jobs based on the estimated run time of jobs. Next, HCoop splits the jobs into tasks and distributes the tasks to the master nodes. Then, HCoop packs tasks by leveraging the complementary of tasks' requirements on different resource types. Finally, the master nodes distribute tasks to workers in the system based on priority of tasks and workers and the resource demands of tasks and the available resources of workers.

C. Job Priority Determination

To classify jobs, HCoop extracts different types of features job related features and system related features (The system-level features reflect the environment in which the jobs will be

TABLE I: Features for predicting jobs' execution time.

Feature	Description
Job-related features	
Required CPU	Amount of CPU resource required by the job
Required MEM	Amount of MEM resource required by the job
Required storage	Amount of storage resource required by the job
Required GPU	Amount of GPU resource required by the job
# of tasks	Number of tasks which the job contains
System-related features	
CPU utilization	VM's CPU utilization
MEM utilization	VM's MEM utilization
Storage utilization	VM's storage utilization
GPU utilization	VM's GPU utilization

running, and we extracted the system-level features) [23–25]. Then, HCoop predicts if the job is a short job by predicting the execution time of the job based on the selected features shown in Table I. If the predicted execution time of the job is no longer than the threshold (T_{th}), then the job is a short job; otherwise, the job is a long job. In the job level, HCoop sets two priority levels: high priority and low priority. If the predicted job is a short job, the job is a high priority job; otherwise, the job is a low priority job. HCoop trains the SVM classifier using the sigmoid kernel function. HCoop repeatedly divides the dataset into training and testing sets using 10-fold cross-validation with percentage split and performs classification.

D. Improving Resource Utilization

Resource fragmentation results in unnecessary over-provisioning of resources and thus can lead to low resource utilization [26, 27]. The reduction of resource fragmentation is critical for improving resource utilization and increasing cost savings. To improve resource utilization in clouds, HCoop presents a task packing strategy. HCoop first identifies the complementary tasks, and packs complementary tasks. Then, it allocates resources to the packed tasks based on their resource demands. The task packing is used to reduce resource fragmentation for achieving high resource utilization.

Each task has a dominant resource, defined as the resource type on which the task has the highest demand. HCoop first packs the tasks with complementary dominant resources such that the summation of the deviation of the two tasks' resource demands on each resource type is the largest. The rationale behind this method is that larger summation of the deviation increases the chance of the VM's resources being fully utilized. Given a list of tasks, HCoop fetches each task T_i, and tries to find its complementary task from the list to pack with T_i. Note that it is possible that task T_i's complementary task cannot be found from the list. In this case, the task T_i solely constitutes an entity to be allocated with resources in a server. To find T_i's complementary task, HCoop uses Equ. (1) to calculate its deviation with every other task T_j with different dominant resource.

$$DV(j,i) = \sum_{k=1}^{l}((d_{jk} - \frac{d_{jk}+d_{ik}}{2})^2 + (d_{ik} - \frac{d_{jk}+d_{ik}}{2})^2), \quad (1)$$

where d_{ik} is T_i's resource demand on resource type k. Finally, the task with the highest deviation value is the complementary task of T_i. This method can also be applied to task packing for more tasks (e.g., three tasks).

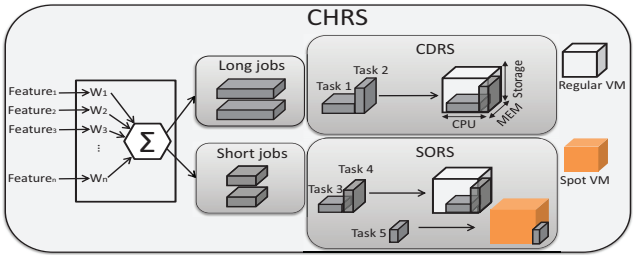


Fig. 4: A cooperative and hybrid resource scheduling (CHRS) for heterogeneous jobs.

E. Resource Allocation

After task packing, HCoop assigns each task entity (packed tasks or a task) to a server with the resources. To further improve resource utilization, HCoop chooses the server that has the least remaining resources (called most matched server) from the servers that meet the resource demand of the task entity.

IV. System Design

In this section, we introduce the design of HCoop.

A. Job Submission and Job Assignment

In HCoop, when users submit their jobs, the jobs are delivered to the schedulers near them. If the scheduler is heavily loaded, the job will be delivered to the lightly loaded neighbor of the heavily loaded scheduler to achieve load balance (see Algorithm 3). Each scheduler has a unique key, and each master has the resource information of the workers associated with master. The workers periodically report their resource information to the master, and the master periodically updates a table which records the resource information of each worker associated with the master.

B. Cooperative and Hybrid Resource Scheduling

HCoop first uses the machine learning algorithm Support Vector Machine (SVM) to classify the jobs into long jobs and short jobs by predicting the execution time of jobs based on the extracted features. If the predicted execution time of a job is no longer than the threshold (T_{th}), then the job is a short job; otherwise, the job is a long job. HCoop trains the SVM classifier using the sigmoid kernel function. HCoop repeatedly divides the dataset into training and testing sets using 10-fold cross-validation with percentage split and performs classification. To improve resource utilization and increase the revenue, HCoop presents a cooperative and hybrid resource scheduling (CHRS) for heterogeneous jobs (see Algorithm 4). Figure 4 shows the cooperative and hybrid resource scheduling. If the predicted jobs are long jobs, HCoop allocates the regular VMs to the tasks of those jobs based on Cooperative Demand-based Resource Scheduling (CDRS); if the predicted jobs are short jobs, HCoop selectively chooses tasks of short jobs and allocates spot VMs (spot instances) to those tasks based on Semi-opportunistic-based Resource Scheduling (SORS).

1) Cooperative Demand-based Resource Scheduling: In cooperative demand-based resource scheduling, the resources (e.g., regular VMs) allocated to tasks of jobs cannot be reallocated to other tasks. HCoop packs the complementary tasks belonging to the same job type, and then finds the VM that can best fit the resource demands of the packed tasks. Algorithm 1 shows the pseudocode for the cooperative demand-based resource scheduling in HCoop.

Algorithm 1: Pseudocode for cooperative demand-based resource scheduling

```
Input: A certain amount of resources (e.g., CPU, MEM, Storage, etc.) in terms of VMs, resource demands of each task, and resource capacity constraints of VMs
Output: The mapping between task entities and VMs
for each task ∈ 𝒯 do  //𝒯 is a set of tasks
    Compute the dominant resource  //Refer to Section III-D
    Compute its deviation with every other task in the task set 𝒯
Pack the tasks with complementary dominant resources such that the summation of the deviation of the two tasks' resource demands on each resource type is the largest
for each task entity ∈ 𝒯_E do  //𝒯_E is a set of task entities
    Assign a regular VM that meets the task entity's resource demand to the task entity
```

Algorithm 2: Pseudocode for semi-opportunistic-based resource scheduling

```
Input: A certain amount of resources (e.g., CPU, MEM, storage, etc.) in terms of VMs, resource demands of each task, and resource capacity constraints of VMs
Output: The mapping between task entities and VMs
for each task ∈ 𝒯 do  //𝒯 is a set of tasks
    Compute the dominant resource
    Compute its deviation with every other task in the task set 𝒯
Pack the tasks that have complementary dominant resources and can be allocated spot VMs such that the summation of the deviation of the two tasks' resource demands on each resource type is the largest
Pack the tasks that have complementary dominant resources and cannot be allocated spot VMs such that the summation of the deviation of the two tasks' resource demands on each resource type is the largest
for each task entity ∈ 𝒯_E do  //𝒯_E is a set of task entities
    if The task entity can be allocated spot VM then
        if ∃ spot VMs with the resources satisfying the resource demand of the task entity then
            Assign the most matched VM to the task entity
    else
        if ∃ regular VMs with the resources satisfying the resource demand of the task entity then
            Assign the most matched regular VM to the task entity
```

2) Semi-opportunistic-based Resource Scheduling: To improve resource utilization while ensuring SLO availability for short jobs, HCoop introduces a concept called Semi-opportunistic-based Resource Scheduling (SORS). Denote ε as the pre-specified prediction error tolerance and P_{th} is a pre-defined probability threshold. For the predicted execution time of a job with prediction error δ, if δ satisfies

$$Pr(0 \leq \delta < \varepsilon) \geq P_{th}, \tag{2}$$

then the tasks of this job can be allocated spot VMs; otherwise, regular VMs must be allocated to these tasks. This can fully utilize spot VMs and improve the resource utilization while ensuring SLO availability for short jobs. Algorithm 2 shows the pseudocode for the semi-opportunistic-based resource scheduling in HCoop.

Algorithm 3: Job_Processing()

```
Input: A submitted job
A Job is submitted
sort(s[])  // Sort schedulers by distances between schedulers and the user who submitted the job
Use the SVM algorithm to classify the job by predicting the execution time of the job
for i ← 1 to Len(s) do
    if s[i] is lightly loaded then
        Assign the job to s[i]
        s[i] split the job into m tasks
        s[i] distributes the tasks to r randomly selected masters that are not heavily loaded
        Use the cooperative and hybrid resource scheduling algorithm to allocate VMs to tasks  // Call Algorithm 4
```

Algorithm 4: Pseudocode for cooperative and hybrid resource scheduling

```
Input: A certain amount of resources (e.g., CPU, MEM, etc.) in terms of VMs, resource demands of each job, and resource capacity constraints of VMs
Output: The mapping between job entities and VMs
for each task ∈ 𝒯_L do  //𝒯_L is the set of tasks of long jobs
    Use the cooperative demand-based resource scheduling algorithm to allocate resource to the task  //See Algorithm 1
for each task ∈ 𝒯_S do  //𝒯_S is the set of tasks of short jobs
    Use the semi-opportunistic-based resource scheduling algorithm to allocate resource to the task //See Algorithm 2
```

C. Architecture of the Scheduler

Figure 5 shows the architecture of HCoop. In HCoop, schedulers are scattered in a distributed system. When a user submits a job to the cloud system, the system will deliver the job to the scheduler that is not heavily loaded and has the smallest geographic distance to the user. The scheduler first splits the job into tasks, then it distributes the tasks to r randomly selected masters that are not heavily loaded (Algorithm 3 shows the processing of jobs). The masters pack tasks that have complementary resource requirements based on the task packing strategy in Section III-D. Each master maintains two task queues: one queue stores tasks that can only be allocated regular VMs and another queue stores tasks that can be allocated spot VMs. Next, masters assign task entity to workers.

V. Performance Evaluation

In this section, we present our trace-driven experimental results. We first conducted experiments on a large-scale real cluster [28]. We tested multiple evaluation metrics and compared our method with three other methods. To further evaluate

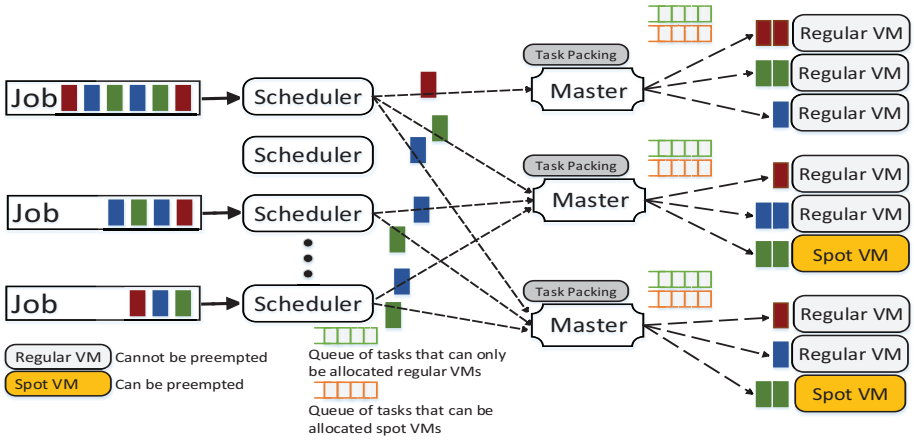


Fig. 5: Architecture of the scheduler in HCoop.

TABLE II: Parameter settings.

Parameter	Meaning	Setting
n	# of servers	30-50
h	# of jobs	100-2500
m	# of tasks of a job	10-2000
L_{th}^{s}	Threshold for heavily loaded scheduler	0.85
L_{th}^{m}	Threshold for heavily loaded master	0.85
P_{th}	Probability threshold	0.95

the performance of our method, we conducted experiments on the real-world Amazon EC2 [29]. In the following, we present the experimental results on a real cluster and the experimental results on Amazon EC2, respectively.

We deployed our testbed in a large-scale cluster. We implemented our method and other three methods in our testbed. We compared the results of our method and the other three methods Pigeon [6], Eagle [10] and Sparrow [30]. Pigeon [6], Eagle [10] and Sparrow [30] are the representatives of hierarchical schedulers, hybrid schedulers and distributed schedulers, respectively. We used up to 1,000 heterogeneous jobs that have different resource requirements.

Pigeon [6]. Pigeon is a two-layer, hierarchical scheduler for heterogeneous jobs. Pigeon divides workers in a cluster into groups and delegates task scheduling in each group to a group master.

Eagle [10]. Eagle is a hybrid data center scheduler for data-parallel programs, and a centralized scheduler handles long jobs and distributed schedulers handle short jobs. Eagle dynamically divides the nodes of the data center in partitions for the execution of long and short jobs, thereby avoiding head-of-line blocking.

Sparrow [30]. Sparrow is a stateless decentralized scheduler that provides near optimal performance using two key techniques: batch sampling and late binding. Sparrow provides fine-grained task scheduling, which is complementary to the functionality provided by cluster resource managers.

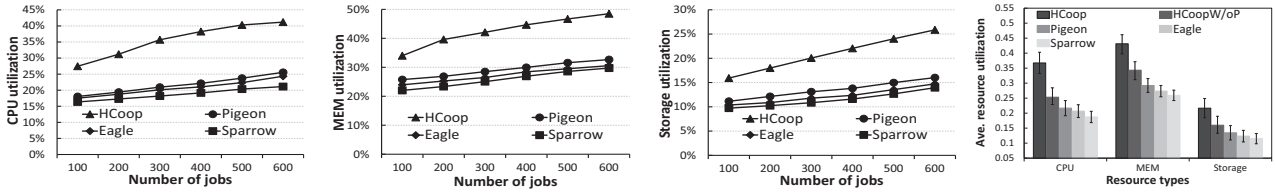
We first deployed our testbed on 50 servers in a real cluster. Users' job arrival rates follow a Poisson distribution [31]. The servers in the real cluster are from Sun X2200 servers (AMD Opteron 2356 CPU, 16GB memory). We then conducted experiments on 30 instances in the real-world Amazon EC2 and the instances in EC2 are from commercial product HP ProLiant ML110 G5 servers (2660 MIPS CPU, 4GB memory). We considered each instance as a server. Each server (instance) was set to have 1GB/s bandwidth and 720GB disk storage capacity in both real cluster and EC2 experiments.

In each experiment, we varied the number of jobs from 100 to 600 with step size of 100. We used the Google cluster trace data [15] to set the parameters. The Google cluster trace [15] records resource usage on a cluster of about 11,000 machines from May 2011 for 29 days. We randomly chose tasks from the jobs in the period between May 1 to May 7. The CPU and memory consumption, and execution time for each task were set based on the Google cluster trace, and the disk and bandwidth consumption for each task was set to 0.02MB [32] and 0.02MB/s [33, 34], respectively. Table II shows the parameter settings in our experiment unless otherwise specified.

A. Experimental Results on A Real Cluster

Figures 6(a), 6(b) and 6(c) show the relationship between the resource (CPU, Memory and Storage) utilization and the number of jobs on a real cluster. In these figures, we see that the resource utilization follows HCoop>Pigeon>Eagle>Sparrow. The resource utilization in HCoop is higher than that in Pigeon because HCoop utilizes the idle spot VMs to improve the resource utilization by selectively allocating the spot instances to tasks of short jobs. Also, HCoop leverages complementarity of tasks' demands on different resource types and uses a task packing strategy to reduce the resource fragmentation and improve the resource utilization. The resource utilization in Pigeon is higher than that in Eagle and Sparrow because all the workers in a group are shared among tasks from short jobs in a work-conserving manner, while all the low priority workers are shared by tasks from long jobs. The resource utilization in Eagle is higher than that in Sparrow because Eagle utilizes Succinct State Sharing (SSS) to improve resource utilization by dynamically allowing short jobs to run in the general partition.

We also measured the average resource utilization of different resource types in different methods. To test the effectiveness of task packing in increasing resource utilization,



(a) CPU (b) Memory (c) Storage (d) Average

Fig. 6: Utilizations of different resource types vs. number of jobs of different methods on a real cluster.

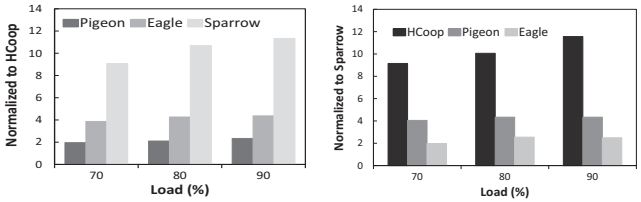


Fig. 7: Job completion times normalized to HCoop on a real cluster.

Fig. 8: Throughput normalized to Sparrow on a real cluster.

we also evaluated the performance of HCoopW/oP, a variant of HCoop in which job packing is not used. Figure 6(d) shows the average resource utilization of different resource types in different methods. We observe that the average resource utilization follows HCoop>HCoopW/oP>Pigeon>Eagle>Sparrow due to the same reasons.

We evaluated the overhead of different methods by measuring the job completion time of each method on the real cluster. We followed the work [6, 10] to measure the job completion time of Pigeon, Eagle and Sparrow normalized to HCoop under different cluster loads. Figure 7 shows different methods' job completion time normalized to HCoop under different cluster loads. In Figure 7, we see that the job completion time follows HCoop<Pigeon<Eagle<Sparrow. As Sparrow does not distinguish between short jobs and long jobs, it incurs up to 11.3, 4.4 and 2.3 times longer job completion time than HCoop, Pigeon and Eagle, respectively.

We also measured the throughput of different methods on the real cluster. We measured the throughput of HCoop, Pigeon and Eagle normalized to Sparrow under different cluster loads. Figure 8 shows different methods' throughput normalized to Sparrow under different cluster loads. In Figure 8, we see that the throughput follows HCoop>Pigeon>Eagle>Sparrow. As Sparrow does not distinguish between short jobs and long jobs, it incurs up to 11.6, 4.3 and 2.5 times lower throughput than HCoop, HCoopW/oRL, Pigeon and Eagle, respectively.

B. Experimental Results on Amazon EC2

To fully test the performance of our method, we also conducted experiments on the real-world Amazon EC2. Figures 9(a), 9(b) and 9(c) show the relationship between the resource (CPU, Memory and Storage) utilization and the number of jobs on Amazon EC2. We observe that the resource utilization follows HCoop>Pigeon>Eagle>Sparrow. The resource utilization in HCoop is higher than that in Pigeon because HCoop utilizes the idle spot VMs to improve the resource utilization by selectively allocating the spot instances to tasks of short jobs. Also, HCoop leverages complementarity of tasks' demands on different resource types and uses a task packing strategy to reduce the resource fragmentation and improve the resource utilization. The resource utilizations in Pigeon is higher than that in Eagle and Sparrow because all the workers in a group are shared among tasks from short jobs in a work-conserving manner, while all the low priority workers are shared by tasks from long jobs. To further test the performance of resource utilization of different methods, we also evaluated the performance of HCoopW/oP and measured the average resource utilization of different resource types in different methods. Figure 9(d) shows the average resource utilization of different resource types in different methods. We observe that the average resource utilization follows HCoop>HCoopW/oP>Pigeon>Eagle>Sparrow due to the same reasons.

We also measured the job completion time of Pigeon, Eagle and Sparrow normalized to HCoop under different cluster loads on Amazon EC2. Figure 10 shows different methods' job completion time normalized to HCoop under different cluster loads. In Figure 10, we see that the job completion time follows HCoop<Pigeon<Eagle<Sparrow. As Sparrow does not distinguish between short jobs and long jobs, it incurs up to 11.1, 4.2 and 2.2 longer job completion time than HCoop and Pigeon, respectively. The result in Figure 10 is consistent with that in Figure 7, and the reasons are the same as that explained in Figure 7.

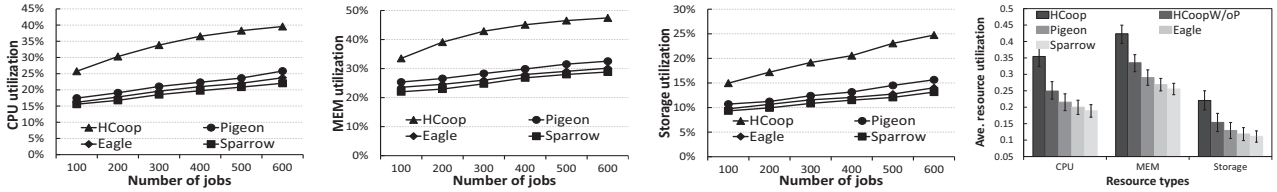
We also measured the throughput of HCoop, Pigeon and Eagle normalized to Sparrow under different cluster loads on Amazon EC2. Figure 11 shows different methods' throughput normalized to Sparrow under different cluster loads. In Figure 11, we see that the throughput follows HCoop>Pigeon>Eagle>Sparrow due to the same reasons explained in Figure 8. As Sparrow does not distinguish between short jobs and long jobs, it incurs up to 11.3, 4.3 and 2.5 times lower throughput than HCoop, Pigeon and Eagle, respectively.

VI. Related Work

First, we review the prior work on throughput optimization in scheduling. Then, we describe previous studies on improving resource utilization in scheduling. Finally, we indicate the advantages of our proposed resource scheduling HCoop compared with the previous studies.

A. Throughput Optimization

Many methods have been proposed for improving throughput in resource scheduling. Qiao *et al.* [35] proposed Pollux



(a) CPU (b) Memory (c) Storage (d) Average

Fig. 9: Utilizations of different resource types vs. number of jobs of different methods on Amazon EC2.

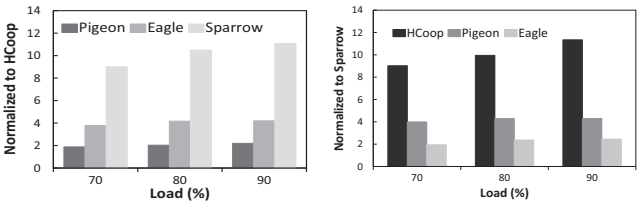


Fig. 10: Job completion times normalized to HCoop on Amazon EC2.

Fig. 11: Throughput normalized to Sparrow on Amazon EC2.

to dynamically adjust the batch size and learning rate for each job to improve the cluster-wide throughput. Specifically, by monitoring the status of each job during training, Pollux models how their goodput (a metric that combines system throughput with statistical efficiency) would change by adding or removing resources. Pollux dynamically (re-)assigns resources to improve cluster-wide goodput. Gu *et al.* [36] proposed Tiresias, a GPU cluster manager tailored for distributed DL training jobs. Tiresias measures the GPU time of the received jobs, and adopts the Least-Attained Service and Gittins Index algorithm to increase the job throughput and decrease the average job completion time. Karanasos *et al.* [37] proposed Mercury, a hybrid resource management framework that supports the full spectrum of scheduling, from centralized to distributed. Mercury exposes the trade-off between execution guarantees and scheduling efficiency to applications through a rich resource management API. Mercury harnesses this flexibility by opportunistically utilizing resources to improve task throughput. Ousterhout *et al.* [30] proposed Sparrow, a stateless distributed scheduler that adapts the power of two choices load balancing technique to the domain of parallel task scheduling. To provide higher scheduling throughput and lower latency, Sparrow makes approximations when scheduling and trades off many of the complex features supported by sophisticated, centralized schedulers. Amaro *et al.* [38] proposed faster swapping mechanisms and a far memory-aware cluster scheduler that make it possible to support far memory at rack scale.

B. Improving Resource Utilization

There are many studies on job/task scheduling for high resource utilization. But many studies [39–43] have a common unrealistic assumption, i.e., the resources required by tasks are identical. Recently, Delgado*et al.* [10] proposed Eagle, a hybrid data center scheduler for data-parallel programs. Eagle dynamically divides the nodes of the data center in partitions for the execution of long and short jobs. Eagle achieves high resource utilization by dynamically allowing short jobs to run in the general partition, which is primarily dedicated to long jobs. Wang *et al.* [6] proposed Pigeon, a two-layer, hierarchical scheduler that employs a divide-and-conquer approach in task scheduling for achieving low latency for short jobs while maintaining high resource utilization without significantly sacrificing the performance of long jobs. Pigeon divides workers in a cluster into groups and delegates task scheduling in each group to a group master. Upon job submission, Pigeon assigns the tasks of an incoming job to the masters as evenly as possible. To improve resource utilization, some studies [26, 44] pack tasks/jobs to machines based on their requirements of all resource types for reducing resource fragmentation. Grandl *et al.* [26] proposed Tetris, a multi-resource cluster scheduler that packs tasks to machines based on their requirements along multiple resources. Grandl *et al.* [44] proposed an online altruistic multi resource DAG scheduler CARBYNE which packs tasks for improving resource utilization. However, these methods do not consider job heterogeneity (job heterogeneity in both execution time and resource demands for task packing. Zhao *et al.* [45] proposed Muri, a multi-resource cluster scheduler for DL workloads. Muri exploits multi-resource interleaving of DL training jobs to achieve high resource utilization and reduce job completion time). To maximize interleaving efficiency, Muri uses a scheduling algorithm based on Blossom algorithm for multi-resource multi-job packing.

Unlike the existing approaches, HCoop first classifies jobs into long and short jobs and allocates the regular VMs to the tasks of long jobs with higher SLO availability guarantee, and selectively allocates the spot instances to tasks of short jobs for improving the overall resource utilization while ensuring SLO availability for short jobs. Also, HCoop leverages the complementary of tasks' requirements on different resource types and tasks' heterogeneity in task size, and it packs complementary tasks whose demands on multiple resource types are complementary to each other and allocates them to a VM to further increase the resource utilization.

VII. Conclusions

This paper presents HCoop, a cooperative and hybrid resource scheduling that improves resource utilization and throughput while ensuring SLO availability in clouds. HCoop first extracts two types of features and uses the machine learning algorithm to classify jobs into two categories (long jobs and short jobs) based on the extracted features. Then, it allocates the regular virtual machines (VMs) to tasks of long

jobs with higher SLO availability guarantee, and selectively allocates the spot instances to tasks of short jobs for improving the overall resource utilization while ensuring the SLO availability for short jobs. Also, HCoop leverages the complementary of tasks' requirements on different resource types and the heterogeneity of jobs/tasks in job/task size, and it packs complementary tasks whose demands on multiple resource types are complementary to each other and allocates them to a VM to further increase the resource utilization. We compare our method with the existing methods under various scenarios using a real cluster and Amazon EC2 cloud service, and demonstrate that HCoop outperforms the exiting methods under both the real cluster and Amazon EC2 cloud service. In the future, we will use different cloud/cluster workloads (including GPU-based workloads) to fully verify the performance of our method. We will test the performance of cost savings. Also, we will consider fairness, dependency for further optimizing the performance of HCoop. In addition, we will consider fault tolerance in designing an efficient scheduling system so that the system can handle node failures/crashes or straggler.

ACKNOWLEDGMENT

This research was supported in part by the Faculty Research Awards Program at Florida A&M University.

REFERENCES

[1] N. Bashir, N. Deng, K. Rzadca, D. Irwin, S. Kodak, and R. Jnagal. Take it to the limit: Peak prediction-driven resource overcommitment in datacenters. In *Proc. of ACM EuroSys*, 2021.

[2] W. Shi, L. Zhang, C. Wu, Z. Li, and F. Lau. An online auction framework for dynamic resource provisioning in cloud computing. In *Proc. of SIGMETRICS*, Austin, 2014.

[3] Y. Zhu, S. D. Fu, J. Liu, and Y. Cui. Truthful online auction toward maximized instance utilization in the cloud. *IEEE/ACM Trans. Netw.*, 26(5):2132–2145, 2018.

[4] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *Proc. of ASPLOS*, pages 127–143, 2014.

[5] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proc. of SoCC*, San Jose, October 2012.

[6] Z. Wang, H. Li, Z. Li, X. Sun, J. Rao, H. Che, and H. Jiang. Pigeon: an effective distributed, hierarchical datacenter job scheduler. In *Proc. of ACM SoCC*, Santa Cruz, 2019.

[7] T. Jin, Z. Cai, B. Li, C. Zheng, G. Jiang, and J. Cheng. Improving resource utilization by timely fine-grained scheduling. In *EuroSys*, 2020.

[8] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das. Modeling and synthesizing task placement constraints in google compute clusters. In *Proc. of SoCC*, 2011.

[9] P. Delgado, D. Didona, F. Dinu, and W. Zwaenepoel. Kairos: Preemptive data center scheduling without runtime estimates. In *SoCC*, 2018.

[10] P. Delgado, D. Didona, F. Dinu, and W. Zwaenepoel. Job-aware scheduling in eagle: Divide and stick to your probes. In *SoCC*, 2016.

[11] J. Rasley, K. Karanasosy, S. Kandulay, R. Fonseca, M. Vojnovic, and S. Rao. Efficient queue management for cluster scheduling. In *Porc. of EuroSys*, London, 2016.

[12] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding. Mlaas in the wild: Workload analysis and scheduling in large-scale heterogeneous gpu clusters. In *NSDI*, 2022.

[13] S. G. Domanal and G. R. M. Reddy. An efficient cost optimized scheduling for spot instances in heterogeneous cloud environment. *Future Gener. Comput. Syst.*, 84:11–21, 2018.

[14] Y. Chen, S. Alspaugh, and R. Katz. Interactive analytical processing in big data systems: a cross-industry study of mapreduce workloads. In *Proc. of VLDB Endowment*, volume 5(12), pages 1802–1813, 2012.

[15] Google trace. https://code.google.com/p/googleclusterdata/ [accessed in July 2022].

[16] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The case for evaluating mapreduce performance using workload suites. In *MASCOTS*, 2011.

[17] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[18] A. Gorbenko and V. Popov. Task-resource scheduling problem. *International Journal of Automation and Computing*, 9(4):429–441, 2012.

[19] J. Zhang, N. Xie, X. Zhang, K. Yue, W. Li, and D. Kumar. Machine learning based resource allocation of cloud computing in auction. *Comput. Mater. Continua*, 56(1):123–135, 2018.

[20] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. L. Rountree, M. Schulz, and B. R. de Supinski. Practical resource management in power-constrained, high performance computing. In *HPDC*, 2015.

[21] J. Liu, H. Shen, and H. Narman. CCRP: Customized cooperative resource provisioning for high resource utilization in clouds. In *Proc. of IEEE Big Data*, pages 243–252, Washington D.C., 2016.

[22] P. Delgado, F. Dinu, A. Kermarrec, and W. Zwaenepoel. Hawk: Hybrid datacenter scheduling. In *Proc. of ATC*, 2015.

[23] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. Hacigümüs. Activesla: A profit-oriented admission control framework for database-as-a-service providers. In *Proc. of SoCC*, 2011.

[24] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz. Wrangler: Predictable and faster jobs using fewer resources. In *Proc. of SoCC*, 2014.

[25] A. Ganapathi, Y. Chen, A. Fox, R. H. Katz, and D. A. Patterson. Statistics-driven workload modeling for the cloud. In *ICDEW*, 2010.

[26] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *Proc. of SIGCOMM*, 2014.

[27] J. Liu and L. Cheng. Swifts: A dependency-aware and resource efficient scheduling for high throughput in clouds. In *INFOCOM WKSHPS*, 2021.

[28] Palmetto cluster. https://ccit.clemson.edu/research/about-palmetto/ [accessed in Mar. 2023].

[29] Amazon ec2. http://aws.amazon.com/ec2 [accessed in April 2023].

[30] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Distributed, low latency scheduling. In *Proc. of SOSP*, 2013.

[31] L. Zheng, C. Joe-Wong, C. G. Brinton, C. W. Tan, S. Ha, and M. Chiang. On the viability of a cloud virtual service provider. In *Proc. of SIGMETRICS*, Antibes Juan-Les-Pins, 2016.

[32] H. Kim, N. Agrawal, and C. Ungureanu. Revisiting storage for smartphones. In *Proc. of FAST*, 2012.

[33] A. L. Shimpi. The SSD anthology: Understanding SSDs and new drives from OCZ, 2014.

[34] J. Liu, H. Shen, and L. Chen. CORP: Cooperative opportunistic resource provisioning for short-lived jobs in cloud systems. In *Proc. of IEEE CLUSTER*, 2016.

[35] A. Qiao, S. K. Choe, J. S. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *Proc. of OSDI*, 2021.

[36] J. Gu, M. Chowdhury, K. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo. Tiresias: A gpu cluster manager for distributed deep learning. In *Proc. of NSDI*, 2019.

[37] K. Karanasos, S. Rao, C. Douglas, K. Chaliparambil, G. Fumarola, S. Heddaya, R. Ramakrishnan, and S. Sakalanaga. Mercury: Hybrid centralized and distributed scheduling in large shared clusters. In *Proc. of ATC*, 2015.

[38] E. Amaro, C. Branner-Augmon, Z. Luo, A. Ousterhout, M. K. Aguilera, A. Panda, S. Ratnasamy, and S. Shenker. Can far memory improve job throughput? In *Proc. of EuroSys*, 2020.

[39] B. Avi-Itzhak and H. Levy. On measuring fairness in queues. *Advanced in Applied Probability*, 36:919–936, 2004.

[40] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, second edition edition, 1992.

[41] D. Raz, H. Levy, and B. Avi-Itzhak. A resourceallocation queueing fairness measure. *ACM SIGMETRICS Performance Evaluation Review*, 32(1):130–141, 2004.

[42] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: flexible proportional-share resource management. In *Proc. of OSDI*, 1994.

[43] A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to unfairness in an m/gi/1. In *Proc. of SIGMETRICS*, 2003.

[44] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan. Altruistic scheduling in multi-resource clusters. In *Proc. of OSDI*, 2016.

[45] Y. Zhao, Y. Liu, Y. Peng, Y. Zhu, X. Liu, and X. Jin. Multi-resource interleaving for deep learning training. In *Proc. of SIGCOMM*, 2022.