2022 IEEE International Performance, Computing, and Communications Conference (IPCCC) | 978-1-6654-8018-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/IPCCC55026.2022.9894315

MARS: Malleable Actor-Critic Reinforcement Learning Scheduler

Betis Baheri*, Jacob Tronge*, Bo Fang†, Ang Li†, Vipin Chaudhary‡ and Qiang Guan*
*Kent State University, Kent, OH USA
{bbaheri, jtronge, qguan}@kent.edu
†Pacific Northwest National Laboratory, Richland, WA, USA
{bo.fang, ang.li}@pnnl.gov
‡Case Western Reserve University, Cleveland, OH, USA
vxc204@case.edu

***Abstract*—In this paper, we introduce MARS, a new scheduling system for HPC-cloud infrastructures based on a cost-aware, flexible reinforcement learning approach, which serves as an intermediate layer for next generation HPC-cloud resource manager. MARS ensembles the pre-trained models from heuristic workloads and decides on the most cost-effective strategy for optimization. A whole workflow application would be split into several optimizable dependent sub-tasks, then based on the predefined resource management plan, a reward will be generated after executing a scheduled task. Lastly, MARS updates the Deep Neural Network (DNN) model based on the reward. MARS is designed to optimize the existing models through reinforcement mechanisms. MARS adapts to the dynamics of workflow applications, selects the most cost-effective scheduling solution among pre-built scheduling strategies (backfilling, SJF, etc.) and self-learning deep neural network model at run-time. We evaluate MARS with different real-world workflow traces. MARS can achieve 5%-60% increased performance compared to the state-of-the-art approaches.**

***Index Terms*—HPC, Cloud System, Scheduling, Workflow Management, Reinforcement Learning, Deep Learning**

I. Introduction

As workflow applications grow in complexity, Scientific Workflow Management Systems (SWMS's) have become essential components in recent HPC-cloud infrastructure [1]. Active research in scientific workflow management has enabled systems used by scientists in practice, addressing many scientists' needs and improving system efficiency. Current workflow management systems, integrated with resource management systems, offer generic services to handle task management, distribution, monitoring and failure management on various types of platforms [2], [3]. Although workflow systems on cloud and HPC infrastructures have been studied with many services offering various capabilities, we still lack optimized and sophisticated scheduler systems, which allow for collaboration of scientists running tasks on HPC systems and those running tasks on cloud systems.

Disparate workflows require different optimizations depending on the condition of the execution environment, state of hardware, resource management and task scheduling. For instance, CPU-intensive tasks need to be optimized to enhance the instruction throughput. Memory-intensive tasks should be scheduled in such a way as to minimize the use of global memory and only write back the final results; this configuration can be achieved by setting the proper configuration in the existing HPC resource manager.

On the other hand, I/O intensive tasks should minimize the data transfers between different infrastructures. In some cases, workflows require different resources. For example, part of the workflow can be executed on CPU-based HPC, and the rest would be benefit from GPU-based HPC or cloud infrastructure. In this case, the cost of executing the tasks of such workflow on two or more different cloud systems should be considered.

Even though these challenges can be solved partially through meticulously designed heuristics, two or more of these factors should be considered for complex workflows. Pursuing recent research in HPC scheduling algorithms, the most common designs either apply an optimal solution for heuristic models or require changes at the system level that may need to replace the existing resource manager in the HPC system. This process must be repeated if the system workload changes or the metric of interest changes (e.g., more memory-intensive tasks than CPU-intensive tasks). The more appropriate solution is to use the existing resource manager by introducing an intermediate layer to create scheduling tasks. Following this architecture design, we can achieve a better optimal result without changing the system level resource manager.

In summary, we illustrate the major challenges within existing scheduling systems:

- The same scheduling strategy may not necessarily work for different infrastructures. For instance, in cluster scheduling, the execution time of a task varies with data locality, hardware health characteristics, interactions with other tasks, and interference on shared resources such as CPU caches, network bandwidth, etc.
- HPC system resources are usually managed by a resource manager, e.g., SLURM. However, these tools are not optimized for dynamic changes in workflow performance

characteristics. Accordingly, there are no optimizations for cost-effectiveness based on performance prediction.
- Practical instances have to make online decisions with noisy inputs and work well under diverse conditions. The decision between CPU, Memory, I/O and cost can have different meanings for individual workflow. Sub-optimization for an individual task may improve after running a couple of batches on the HPC system.
- Scheduling policy switching between different systems can be challenging, workflow requirements and system configuration can be different on an individual HPC system. In most cases switching the scheduling system means re-initiating the HPC system, which requires time and termination of other users' tasks.
- On basic workflows, optimization can be done using one dimension (CPU, Memory or I/O) optimization; there is no need for a multi-layer machine learning scheduler. The practical applications do not need further optimization. Simply using the existing resource manager would satisfy the users' needs.
- Lastly, workflows might benefit from various multi-level optimization and using machine learning scheduling techniques. In this case, each task has its own characteristics, which may be so sophisticated that we need to consider multi-dimension of the metrics of interest, such as IO, CPU and Memory.

To overcome these issues, we design a generic scheduling system, enabling self-learning, performance adaptation and naturally working with the existing resource manager on HPC systems.

In this paper, we introduce a malleable actor-critic reinforcement learning scheduler (MARS) to address the challenges within existing scheduling systems with the following features:

- MARS presents a malleable scheduling policy ensembling A3C reinforcement learning and heuristic policies.
- MARS optimizes scheduling performance through task parallelism and workflow classification through graph comparison and outperforms the state-of-the-art HPC schedulers by 5-60%.
- MARS requires none to minimal changes to the existing HPC resource manager such as Slurm and other cloud resource managers. From a design perspective, MARS is a good candidate for HPC-cloud heterogeneous environment.
- MARS design supports both simple and complex workflows, and the scheduling profile can be expanded from one to more dimensionality to bring more optimization on the desired characteristic.
- MARS requires none to minimal changes in users' workflows configuration, task optimization between user configuration and HPC system is done in the intermediate layer. Scheduling in such a way that we can benefit from simple optimization and complex machine learning schedulers.

The rest of the paper is organized as follows: in section II-A we discuss HPC workflow requirements and descriptions along with server parameters and our motivation. We explain how MARS integrates previous heuristic algorithms along with asynchronous actor-critic reinforcement learning, and we give a detailed explanation of the reinforcement learning approach and our decision on how to select the best suitable scheduling algorithm in section III. We discuss our implementation methods in section IV. In section V, we discuss our results and compare them to previous works and present our observations. We also explain why MARS is outperforming the other approaches. Lastly, in section VI we discuss prior work in this area and conclude in section VII.

II. Background and Motivation

A. Background

A workflow application is a set of tasks or instructions executed on arbitrary input by particular order as steps. Workflow can be chained computation in physics, chemistry, etc. To improve the performance of workflows and create more meaningful relations between tasks, steps and requirements, we can use a directed acyclic graph (DAG) based on each component. As Hongzi M. and et al showed in their approach, creating DAG from workflows can be done in two categories. First, it can be done considering pure output related and their dependency. Second, another DAG can be generated based on tasks' resource requirements.

Any workflow can have just one or multiple requirement DAGs based on the complexity of the workflow. Similarly, both DAGs can represent the target system resources and scheduling requirements. System resources are queries from existing resource manager which are explained in detail in section III-A1, scheduling requirements are the number of CPUs per node and/or entire workflow, the amount of memory, I/O and the cost based on desired parameters such as I/O throughput, CPU usage, GPU usage, etc. More details are provided in section III.

In order to better understand how to solve the complexity of both complicated and straightforward workflows scheduling, traditional optimization and machine learning techniques need to be studied. Mu'alem introduced optimization over First Come First Serve (FCFS) scheduling method, and AAhuva W. Mu'alem et al. [4], their Backfilling method over well known FCFS algorithm was to overcome the fragmentation problem. Backfilling uses dynamic partitioning to schedule tasks on distributed systems to maximize performance. There are two major implementations, conservative Backfilling and Easy Backfilling. However, even though both methods were introduced to reduce starvation in the case of large workflows, both versions can cause starvation. In practice, if the workflow contains many tasks, it might be more beneficial to use machine learning techniques. In contrast, the Backfilling scheduling method would achieve better optimization when the workflows do not contain enough tasks to train and test the machine learning model.

The volume of workflows tasks recently caused researchers to focus on machine learning techniques instead of traditional

methods. One of the most recent and popular methods is reinforcement learning. The reinforcement learning (RL) method uses vector-based image transition. In order to translate the HPC scheduling parameters into this method, each resource, CPU, Memory, and I/O would be shaped into images. One requirement of this method is that the workflows' sizes must be the same. Then the translated images would be used to train the RL model. The optimal solution can be achieved after training the system.

Knowing that both methods have limitations, we need to balance Backfilling and RL methods to support workflows more diversely. The backfilling method suffers from optimization for large workflows, and the reinforcement technique needs initialization to be optimal, depending on workflow requirement and the volume scheduling algorithm needs to switch between these methods.

B. Motivation

The regular Reinforcement Learning (RL) schedulers require replacing existing HPC resource management tools, and in most cases, users have to adapt and change their workflow to satisfy the new system's requirements. In a more specific explanation, HPC's existing resource manager would be replaced with an RL scheduler, and user configuration would have to change to use the new scheduler system.

One reinforcement learning limitation is that the entire training set must be specific for the HPC system. Otherwise, the training model would not be optimized. Tuning hyperparameters and optimizing in favour of all dimensions is one of the limitations of RL methods.

Existing HPC resource managers suffer from large numbers of tasks. For better optimization, we need to either replace them with more specific algorithms depending on workflow or use an intermediate layer to communicate with the resource manager. Replacing the resource manager is time-consuming and requires knowledge of workflows. Since replacing the resource manager is costly and compromises support for legacy workflows, in our approach, we do not require to change the subsystem, and we use the existing tools to increase the performance [5].

In our approach, we introduce a median layer to the existing HPC resource manager to avoid replacing the entire system and not depending on one solution for all possible cases. The user can specify how many parameter servers and nodes to use, including the amount of required resource (e.g. CPU, Memory, GPU, I/O, etc.), then submit the workflow to MARS. Our Scheduler MARS chooses between simple Backfilling or an advanced reinforcement learning (A3C) algorithm and then assigns tasks to nodes for execution by communicating with the existing resource manager on the HPC system.

III. MARS Design and Implementation

A. MARS System Overview

Figure 1 shows the overall system structure of MARS. We assume that the workflow description and generated DAG graph are provided to the scheduling system. One example

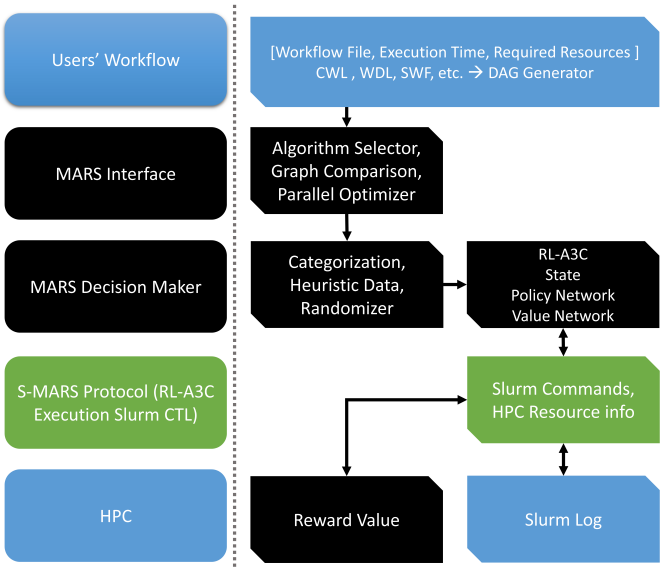

Fig. 1. MARS System Overview

of previous work done in BEEFlow [6], which proposed an in-situ analysis-enabled workflow management system that supports multiple platforms using HPC containers.

Our design consists of several parts as shown in the figure 1, MARS Interface is the API that provides an interface to HPC users to submit their workflows on an HPC system. MARS decision-maker is in charge of deciding between Backfilling and RL-A3C algorithm, where algorithm selection and different optimizer can be assigned based on workflows requirement DAG. As shown, we use heuristic data, users' configuration, along randomization for unknown workflows characteristics. As mentioned before, large workflows can be optimized based on more than one dimension. In order to achieve the optimal solution, the RL-A3C needs to train on data, and in our case, a randomizer helps speed up this process and tune the hyper-parameters faster. More sophisticated methods such as population-based (PBT) in ML can surely help but, we observe it is unnecessary to use more complicated methods. The rest of our design follows the RL-A3C principle with reward value read from the HPC resource manager, two policies, the model parameters from the DNN network, and the initial state of HPC resources. The Slurm commands and HPC resource information can be derived from existing resource managers such as Slurm, MARS uses the same commands to schedule tasks on an HPC system.

In general, we take the following steps shown in algorithm 1 to accomplish the optimization for each workflow.

The corresponding benefits to our design are:

- Each workflow can be executed independently from others
- HPC systems do not depend on a single algorithm
- Workflows can run simultaneously with other workflows. In respect to users' workflows are not restricted or limited by the algorithms used for scheduling

Result: Saved Model M
Input: Created DAG from workflows ζ
Input : Decision D
Input: Policy ν
Input: Available HPC Resources from Existing Resource Manager (SLURM) HPC_R
if *Task* ι *and* ι_{i+1} ! = *dependency* **then**
  Compare and Parallel Tasks $\iota + \iota_{i+1}$
else
  D = $MARS_Decision(\zeta)$
  ν = $MARS_Policy$(D)
  M = MARS(ζ,ν,HPC_R)
  return M
end

Algorithm 1: MARS Overall Algorithm

- Since the full optimization is done regardless of existing HPC systems. We can update saved models based on best suitable parameters

*1) **Algorithm Selection**:* In typical cases, resource management in the HPC system is based on CPU, memory, and I/O utilization. On the other point of interest, considering the cost of each task execution on other cloud infrastructures can help scientists minimize the overall cost.

Traditionally schedulers optimize tasks only on one dimension. A simple Backfilling scheduler can be an example. In Backfilling scheduling, the scheduler tries to optimize CPU usage. In the next step, more sophisticated schedulers use modern Machine Learning algorithms to optimize tasks based on CPU, Memory and I/O. However, in most methods, schedulers either sacrifice one feature for another or find the average solution. Recent ML schedulers use one specific reward function to update the trained model and learn from the previous execution.

In our proposal, MARS can adapt on different reward values read from the HPC resource manager and decide between a simple algorithm such as Backfilling to a more complicated online algorithm such as asynchronous actor-critic reinforcement learning to execute tasks. By creating a model based on the RL-A3C algorithm and updating that model with the similar technique that D. Zhang previously introduced and et al. [7] we can reuse a trained model with similar workflows. However, the training of the system is highly correlated to the size and number of tasks in one arbitrary workflow.

Based on our observation, small workflows such as a simple RNA search would be an inefficient model. On the other hand, complex and large workflows in RL, such as Blast, would cause an over-fitting of the network. This phenomenon would result in an inefficient reward value and model. In our approach, by combining time window and custom loss function, the reward value and model generated from the workflow would be more accurate compared to previous approaches.

B. Policy Model and Algorithm

Our policy model depends on the size of the workflow. In terms of small workflows that can be optimized with the simple FCFS algorithm, MARS bypasses the RL algorithm and creates a simple schedule for tasks ready to be executed on HPC. On the other hand, when workflows contain a large subsection of tasks and the running time requires hours to days, MARS selects an arbitrary RL-A3C algorithm based on previously saved models.

The reinforcement learning module in MARS contains a scheduler agent, environment, and neural network based on server parameters input and reward value from the HPC environment. At each time step *t* the agent observes the parameters on HPC state s_t, then chooses an action a_t. Following that action, the environment's state would proceed to s_{t+1} and the agent receives reward r_t. The state transitions and rewards are stochastic and are assumed to have the Markov property; i.e. the state transition probabilities and rewards depend only on the state of the environment s_t and the action taken by the agent a_t.

In most RL approaches, learning is done by performing *gradient-decent* on the policy parameters. The critical idea in policy gradient methods is to estimate the gradient by observing the trajectories of executions obtained by following the policy. Similar to *Monte Carlo Method* [8], samples are taken of multiple trajectories, and the empirically computed cumulative discounted reward is used. However, this approach is based on a naive algorithm and usually calculates a local maximum instead of the global maximum. In order to overcome this limitation, we use a similar method as other researchers , RL with Actor-Critic Algorithm (ACA) in MARS.

*1) **Reinforcement Learning Objects**:* Based on the definition for objective function for policy gradients, in our approach, parameters are read from the existing resource manager, and the action taken upon optimizing task execution is done by MARS. Using well-known machine learning techniques [9], [10], mapping between HPC server parameters and RL properties, we can redesign reinforcement learning to support HPC systems.

*2) **Reinforcement Learning Using Actor Critic** :* We can define the Actor-Critic method, where the Critic estimates the value function, which can be Q-Value or state value V-Value [11]. In our approach, we took the state value from an existing resource manager such as Slurm. MARS uses Slurm manager outputs to calculate the reward value. 2 MARS Policy RL-A3C Algorithm:

As we explained earlier, the computation of the reward value can have different meanings. The critic is a state-value function, MARS can be optimized based on Parameter Server values read from Slurm or any other resource manager, and final value results can be used to determine if there was an improvement or not.

Figure 2 shows the Policy Structure of MARS. User's workflow description can be in any standard format such as Common Workflow Language (CWL), The Workflow Description Language (WDL), Standard Workload Format (SWF), etc. As mentioned before in section II preferably, the DAG is generated from workflow description containing tasks (tasks) to execute and the dependency between them. In our example, one workflow can be as simple as one task or have multiple

Result: HPC Reward Estimation $\pi_\theta \approx \pi_*$
Input: HPC Scheduling Action based on State Parameters $\pi(a|s,\theta)$ Input: HPC CPU, Memory, I/O, Cost Values $\check{\mathrm{v}}(s,w)$
Algorithm parameters: step sizes $\alpha^\theta > 0, \alpha^w > 0$
Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$(e.g., to 0) Set weights to 0 at beginning,
Initializing C as the Cost Probability added to evaluation;
while *for each epochs* **do**
  Initialize S (first state of episode);
  $I \leftarrow 1$;
  while S *is not terminal (for each time step)* **do**
    $A \approx \pi(\cdot|S,\theta)$;
    Take action A, Observe S', R;
    $\delta \leftarrow R + \gamma\ \check{\mathrm{v}}(S',w) - \check{\mathrm{v}}(S,w)$(if S' is terminal, then $\check{\mathrm{v}}(S',w) = 0$);
    $w \leftarrow w + \alpha^w I \delta \nabla_w \check{\mathrm{v}}(S,w)$;
    $\theta \leftarrow \theta + \alpha^\theta I \delta \nabla_\theta ln\pi(A|S,\theta)$;
    $\theta \leftarrow \theta + \nabla C$;
    $I \leftarrow \gamma I$;
    $S \leftarrow S'$;
  end
end

Algorithm 2: MARS RL-A3C Policy

dependent parts, such as the Blast example, or similar to a linear search workflow. The generated data then would be fed to our categorizing module, which determines the depth of the workflow based on the description, graph comparison algorithm and heuristic generated models.

The algorithm selector module decides whether to use RL-A3C or basic FCFS, as mentioned before, for simple workflows which require only limited execution time. If no other workflows are running, and the description requires most system resources, running RL-A3C would cause overhead. However, in case MARS can combine multiple independent workflows and run RL-A3C, it would switch back to using the RL-A3C algorithm and build the best suitable model for that specific type. We kept the traditional algorithms such as FCFS, Backfilling, etc.

In order to support legacy workflows and save on training time and in case an HPC system is not equipped with a GPU, a small optimization based on the known graph combining algorithm [12] would run next to combine the parallel tasks. Compared to the standard Reinforcement Learning technique, we use this graph search algorithm to identify the best possible model to gain an optimal outcome and user input as a variable to differentiate between CPU, Memory, I/O, and Cost of each task. The generated model will train the system for optimization and feedback output.

Next, MARS queries the available resources from Slurm, knowing the current state of the system and workflow description. Next MARS creates a state description based on Job type, the number of time slots run, remaining epochs, allocated resources on HPC, the number of workers based on the workflow description, and the number of parameters. Based on the previous discussion, we build a policy and value network, calculate a baseline, and initiate action; then, using the Slurm interface on HPC, we initiate a batch of tasks on HPC (Action).

In addition, MARS needs to decide the best split between tasks and parallelism based on available resources, knowing that each workflow can be divided into sub-workflows based on searching paths, MARS categorizes tasks into groups. After this separation, it generates a deep neural network based on user input and CPU, Memory and I/O values.

Finally, using Slurm CTL MARS queries about remaining available resources, current executing tasks, previously executed times, and corrupted previous tasks. MARS then calculates the reward value and uses a baseline. It updates the neural network. In order to overcome training overhead and inefficient models, MARS creates an arbitrary base network based on heuristic workflow data. If the data is absent from the database, we generate a similar workflow with smaller tasks to train the network.

3) ***Graph Comparison and Parallel Optimizer****:* In most RL-based schedulers, the generated workflow graph and cost are not considered. The deep neural network is purely based on workflow input data or previous execution. However, if we consider the graph generated from the workflow and use search algorithms to find the similarities in individual tasks, we can predict and categorize each task based on their CPU, Memory, I/O intensity. In addition, we can also consider the cost of each execution. Based on a predefined table, we can calculate how much each task would cost to run on some arbitrary cloud infrastructure.

In practice, Directed Acyclic Graphs (DAGs) have tens or hundreds of stages with different requirements and execution times. Based on the dependencies and requirements, each task can be executed in parallel or wait for other tasks to be completed. This complexity can be challenging in terms of scheduling, and to solve this issue MARS needs to execute tasks in parallel as much as possible without wasting CPU, or Memory utilization [13].

As mentioned before, graph comparison is algorithmically hard, similar to C. Delimitrou and et al. [14] approach, we use a scale-up and scale-out method to achieve the categorization. Assuming that the individual parts of a workflow's DAG can be categorized and compared to each other based on size and resources, MARS tries to combine the independent tasks as a single parallel task.

4) ***Decision Making****:* MARS decision making is based on comparing the DAG and heuristic data, using a heuristic data model, DAG classification, or based on the size of the workflow MARS chooses the best suitable algorithm between basic back-filling and RL-A3C to execute an arbitrary workflow 2. In complementing combining CPU, Memory, I/O and creating a general neural network, we generate an individual network based on graph comparison and user input for RL-A3C candidate workflows. Complementary to the previous

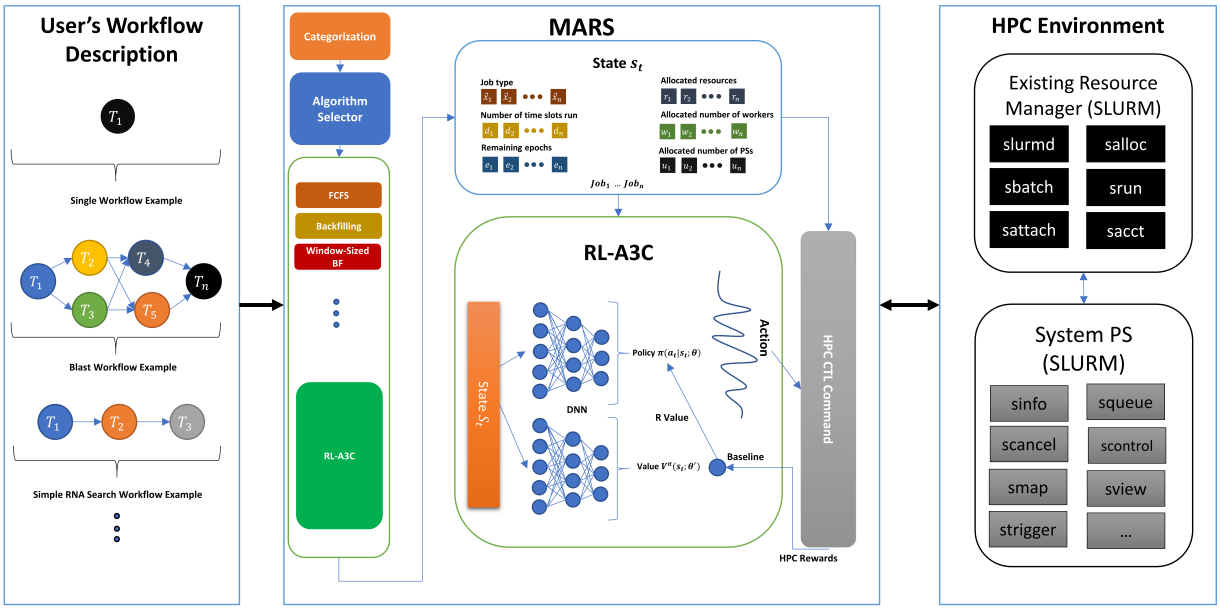

Fig. 2. MARS Policy Network

method, the users' variable is used to determine the intensity of requirements and also, in order to achieve a better result, the logs from the target HPC system will be used in the evaluation.

In the case of RL-A3C workflows, the first initiation and task execution would have to be on a more general deep neural network with a more straightforward reward function due to the lack of training data. However, after a couple of workflow executions, the first network can be replaced with a more complex network. After that process, MARS would get the output from the HPC system and calculate the universal reward means. As we know, returning a positive value from the reward function can identify the desired settings then and would cause MARS to continue optimizing on the same network for similar workflows. On the other hand, the cumulative negative reward value would cause a feature selection change in the network and update the loss function.

Algorithm 3 shows the basic decision making of the MARS scheduler. Our design uses workflow size and configuration to decide on the algorithm policy. In our experiment, we observe that workflows with a size less than 512 are not sufficient to run directly on RL-A3C. In order to improve this issue, we either combine the following workflow with the previous one or run the heuristic algorithm. In the algorithm's first part, we combine the following workflow with the current workflow. Next, if the compatibility of dimension fails or the existence of the following workflow is absent, then MARS chooses the heuristic algorithm. Next, for the large workflows, we split those into sub-workflows and execute the RL-A3C algorithm to avoid over-fitting the network. In each step, we save the RL-A3C model for future use.

IV. Implementation

The MARS algorithms are implemented using Tensorflow [15] and Gym OpenAI [16]. For the training process we used Proximal Policy Optimization (PPO) algorithm derived from OpenAI Spinning Up library [10], [17].

We used a randomly generated data set based on real workflows and actual real-world data from different sources to evaluate the proposed solution. The real-world workflows are based on SWF archive data as shown in Table I.

TABLE I
List of Workload Traces

Name	CPU	Month(s)	Date
SDSC IBM-SP2	128	24	1998
SDSC IBM-Blue	1152	32	2000
High Performance Computing Center	240	42	2002
Argonne National Laboratory Intrepid	163840	8	2009
Synthetic_G001	256	12	2019
Synthetic_G002	1024	6	2019

In our experiment, we aim to compare the previous works with MARS. We compare MARS with heuristic job scheduling algorithms, shown in Table II. The table II shows the heuristic scheduling policies infused with MARS, which can improve the performance of legacy and modern workflows. MARS is compared with two well-known policies: First Come First Served (FCFS), where the arrival order schedules tasks; and Shortest Job First (SJF), where tasks with shorter processing times are scheduled ahead of the other tasks. Some other comparative policies are WFP3, and UNICEF [18], which are based on the processing time, requested number of cores and waiting time of the tasks. WFP3 favours shorter and older tasks over large ones without starvation, and UNI favours small tasks by using a fast turnaround policy for performance

Result: Best Suitable Action α
Input: Workflow χ & Workflow size η
Initializing workflow task size, Queue, Task, Model:
$\eta \leftarrow \chi$, Q, ω, M
if η <*MEDIAN* **then**
  if $\chi_{i+1} == TRUE$ & χ_{i+1} *is compatible (RL-A3C vector dimensions) with* χ_i **then**
    $\chi = \chi_i + \chi_{i+1}$ >MEDIAN ;
    $Q \leftarrow \eta$;
    $M \leftarrow MARS - RL - A3C(Q)$;
  else
    if η <*MIN* **then**
      $Q \leftarrow \eta$;
      SJF(Q);
    else
      $Q \leftarrow \eta$;
      UNICEF(Q);
    end
  end
  $M \leftarrow MARS - RL - A3C(Q)$;
else
  while η >*MAX* **do**
    $\omega = \frac{\omega}{2}$
    $Q \leftarrow \omega$;
    $MARS - RL - A3C(\omega)$
    $M \leftarrow MARS - RL - A3C(Q)$;
  end
  $Q \leftarrow \omega$;
  $MARS - RL - A3C(\omega)$
  $M \leftarrow MARS - RL - A3C(Q)$;
end

Algorithm 3: MARS Decision Making Policy

enhancement. Policy F1, F2, F3, and F4 [19] represent the nonlinear machine learning-based scheduling algorithms for minimizing the average bounded slowdown of tasks. Based on our observation, switching to known heuristic algorithms and RL-A3C increases the performance and saves a noticeable amount of time in training for the basic legacy workflows.

TABLE II
HEURISTIC SCHEDULING POLICY USED

Name	Function
FCFS	$ABS(t) = s_t$
SJF	$ABS(t) = r_t$
WFP3	$ABS(t) = -(w_t/r_t)^3 * n_t$
UNICEP	$ABS(t) = -w_t/(log_2(n_t) * r_t)$
F1	$ABS(t) = log_{10}(r_t) * n_t + 8.70 * 10^2 log_{10}(s_t)$
F2	$ABS(t) = \sqrt{r_t} * n_t + 2.56 * 10^4 * log_{10}(s_t)$
F3	$ABS(t) = r_t * n_t + 6.86 * 10^6 log_{10}(s_t)$
F4	$ABS(t) = r_t * \sqrt{n_t} + 5.30 * 10^5 log_{10}(s_t)$

In an HPC system, workflow tasks may arrive continuously. In order to train the model using RL-A3C, we save the training results after a predefined window time, and then we let the actor-critic algorithm improve the model. After building a basic model based on the RL algorithm, the Actor-Critic part evaluates the network. This strategy would create a training batch for the workflow. If the batch size is too small, MARS' decision module gives two options if the remaining workflow size is sufficient enough MARS combine sub-workflows. On the other hand, in the absence of sufficient size, MARS would switch back to back-filling or FCFS algorithm.

In our experiment running basic workflows on RL-A3C takes a significant amount of time to train and causes inefficiency in HPC systems. In order to overcome this issue, a combination of legacy and RL-A3C algorithms would be more appropriate. Another issue in RL-A3C is over-fitting the model due to the large batch size and exponential growth of the number of possible tasks. In order to solve this issue, we introduce a median layer to create sub-workflows. Based on our observation, the best training sets are between 512 to 20000 running on 2000 to 4000 epochs for RL-A3C. Knowing that the smaller or larger batch sizes could introduce an issue, the MARS decision module would combine or split the sub-workflows.

As we described in Section III, in RL-A3C, the state is the input of the DNN agent, and the representation of state is a vector containing available resources and pending tasks. In HPC number of pending and arriving tasks can vary. However, in DNN, the vector to create the network should be fixed-sized. In order to overcome this issue, we took the same approach as previous works and added extra 0s to the end of the vector [20].

V. EVALUATION

In this section, we present our results obtained by running MARS scheduler on a simulated environment using data traces generated from HPC data centers. First, we describe the environment setup and workflow traces used in our experiment, then we evaluate different algorithms and compare them to our approach. We discuss the performance evaluation under different conditions and workloads of HPC environments. Our simulator was inspired by a similar method used by D. Zhang et al. [7]. However, to comply with our approach, we extended the simulator with Gym and OpenAI to return the proper reward values from the environment. Running the training set on an actual HPC environment requires an enormous number of iterations to learn, considering that most HPC environments are not capable of running the RL-A3C algorithm due to lacking GPU capability or available resources for non-HPC applications. The best approach is to either dedicate an arbitrary external server to train the model or run the simulation in a local environment.

A. Simulation Environment

We simulate a homogeneous HPC environment executing tasks based on moving forward the timestamp instead of running those tasks. These workflows were based on traces collected from real systems, but we use the CWL and SWF workflows formats to guarantee compatibility. When a workflow is generated, if the resources required to run an arbitrary task belonging to the generated workflow are not present, the simulator uses the back-filling method to run smaller tasks first.

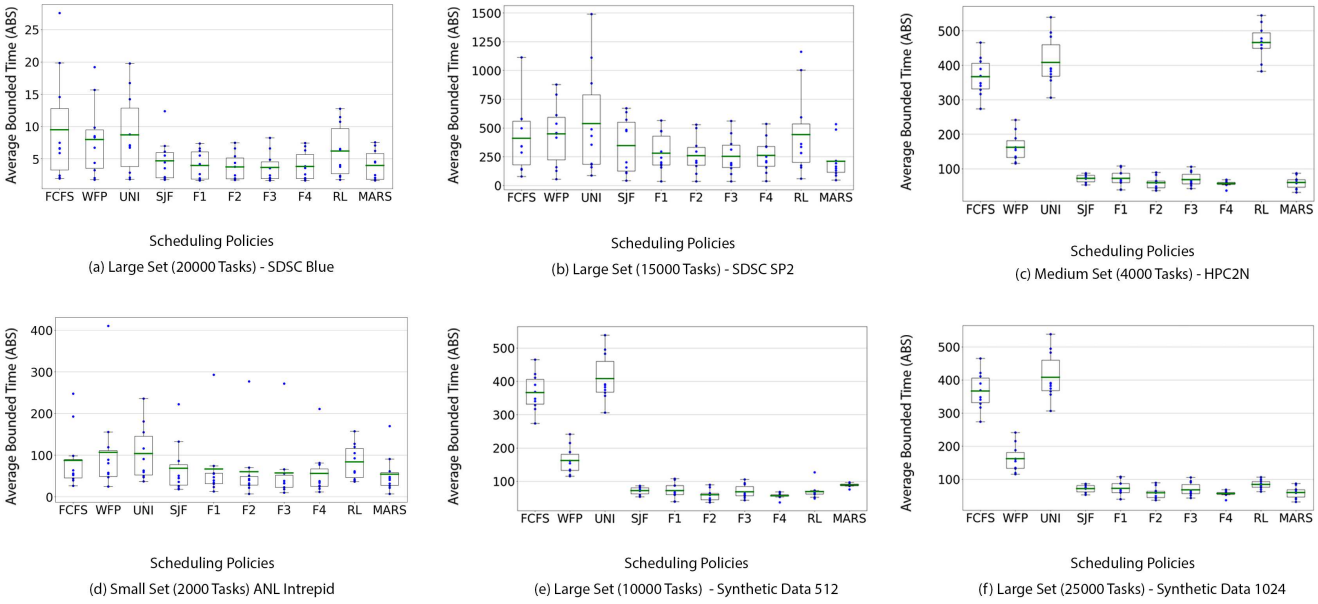

Fig. 3. Performance of scheduling policies with different workload traces.

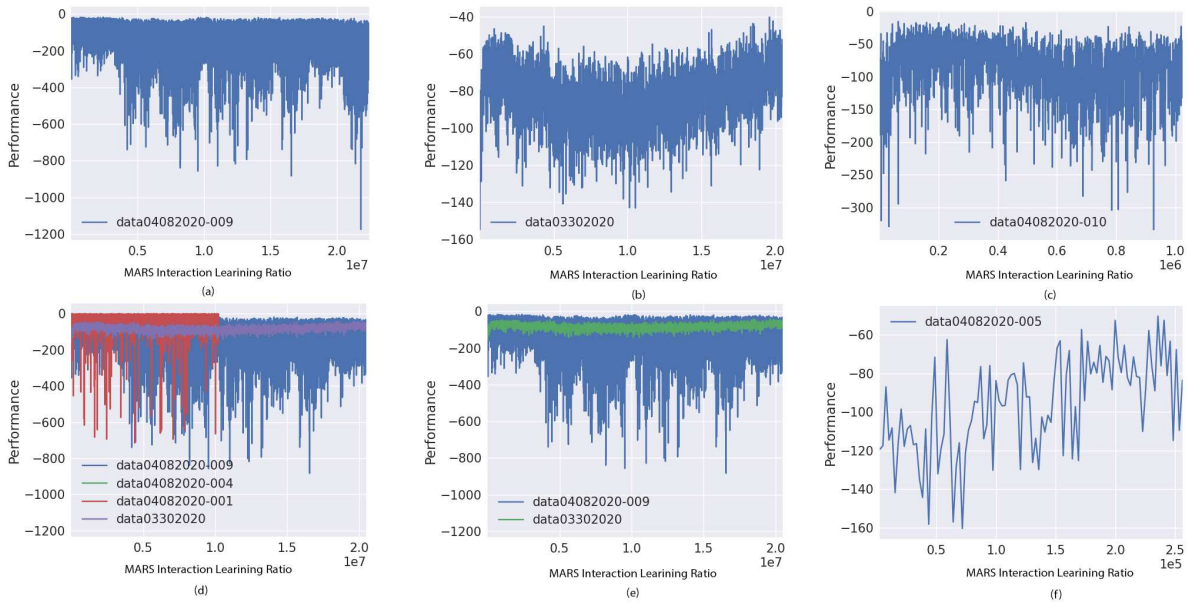

Fig. 4. Performance of MARS under different learning ratios

B. HPC Reward and Metrics

HPC scheduling metrics are mostly based on response time, and it is defined as the total wall-clock time from the instant at which the task is submitted to the system until it finishes its run. The most basic method to calculate the running time and wait time for tasks is slowdown, $slowdown = \frac{T_w + T_r}{T_r}$. A more sophisticated method is to take the average slowdown to minimize the wait time [21]. Table III shows different evaluation metrics. The problem with the slowdown metric is that it overemphasizes the importance of short jobs; to overcome this issue, Feitelson et al. [22] have suggested Bounded-slowdown. The behaviour of this metric depends on the choice of τ, which is the threshold value. Zotkin and et al. [23] have introduced a new problem where tasks that do the same amount of work with the same response time may lead to different slowdowns results due to their shape, which is the ratio of processors to time. This introduces another metric known as a per-processor slowdown. We used average bounded slowdown instead of per-processor because, in our workflow examples, the shape of our test systems are identical to each other.

In our approach we set the goal as minimizing the average

TABLE III
SCHEDULING METRICS T_r IS THE EXECUTION TIME OF THE JOB, T_w IS THE TIME SPENT IN TURNAROUND [24]

Metric	Formula
Slowdown	$\frac{T_w+T_r}{T_r}$
Bounded-slowdown	$max\{\frac{T_w+T_r}{max\{T_r,\tau\}}, 1\}$
pp-slowdown	$max\{\frac{T_w+T_r}{P*max\{T_r,\tau\}}, 1\}$

bounded slowdown $= -max\{\frac{T_w+T_r}{max\{T_r,\tau\}}, 1\}(-ABS)$. At the start of the algorithm, calculating the average is not possible, instead we return 0 as a reward. After finishing the entire task sequence then the RL-A3C agent gets the average as $-ABS$.

C. Results

In this section, we show that MARS, by using a combination of heuristic and the RL-A3C algorithms, can improve the performance, time and avoid over-fitting the network for scheduling tasks on HPC systems. Most reinforcement learning algorithms need to be configured with proper parameters from HPC. Figure 3 shows the different policies based on different configurations, where the y-axis is the average bounded slowdown, and the x-axis is the different scheduling policies.

Our scheduler ratio of training and testing was 70% to 30%, similar to most other RL algorithms. We categorized three different configurations and sizes for our testbed: the small data-set contained between 512 to 2000 tasks, the medium size data-set was from 2000 to 9000 tasks, and lastly, the large data-set was between 10000 to 25000 tasks. We randomly selected tasks from different data sets and performed experiments with different configurations. We considered the number of iterations per task in DNN and the delay between task arrival. By experimenting with different configurations, we showed that the proper configuration causes a significant difference in result in reinforcement learning and heuristic algorithms. Lastly, we added the cost-aware probabilities after creating the RL-A3C model.

In figure 3 part (a), we choose a large data-set from IBM SDSC Blue with 20000 tasks to train and 6000 tasks to test. However, since the data configuration was chosen randomly, the reinforcement learning algorithm reacts worst than MARS. Similarly, in part (b), we selected 15000 random tasks and observed the same result; however, if the workflow size is large enough and the data is consistent with the configuration of DNN, the RL-A3C algorithm will improve. Figure 3 part (c) was HPC2N data-set with 4000 selected tasks, and Figure 3 part (d) contains small selected tasks from ANL Intrepid data-set. All three experiment configurations were chosen randomly.

As discussed, the MARS scheduler tries to solve this issue in two ways: it either combines the tasks to generate a proper size for training and testing in RL-A3C or switches back to a heuristic algorithm. In our experiment, we showed that in the case of a proper and ideal configuration 3 (e), RL-A3C performs better compared to MARS. However, since in HPC, achieving the ideal configuration is rather difficult, in other cases, such as Figure 3 part (f), using the suggested method derives a better performance. Our experiment shows MARS on average can achieve between 5% to 60% better performance compared to other policies.

Another issue in reinforcement learning to consider is over-fitting the network. In figure 4 we observe that based on data-set configuration and learning ratio, we can achieve different performances. Figure 4 part (a) is a large data-set with 50000 iterations per task, which causes RL-A3C learning to interact frequently with the HPC system.

Figure 4 part (b) is the optimal configuration with the proper size data-set; however, in part (c), the configuration and HPC parameters change randomly, and that causes the RL-A3C agent to interact with HPC more often. Figure 4 part (d) and part (e) shows the comparison of different experiments together, and lastly, part (f) shows an insufficient data-set size to train. To resolve these issues, MARS tries to update the reward values from HPC after each iteration, and by selecting a heuristic algorithm for small data-set sizes, we bypass the inefficient training model.

In our test experiment, the cost of each task was randomly generated, and after RL-A3C soft-max values, we incorporate costs as another probability function as a probability between 0 and 1. We used Gaussian distribution to add the cost factor to the final step of the DNN soft-max calculation. As discussed before, adding the cost to the training model would result in a unique data model. As a consequence of keeping the model's generality, the cost would be incorporated after creating the DNN network. A more specific reward value can be derived from the HPC system by calculating the cost with each action taken by the agent. As shown in figure 3, with random configuration for RL-A3C, the performance decreases between 5% to 60%. However, by using MARS policy and combining heuristic and RL-A3C with cost-awareness, the performance improves back to an optimal solution.

VI. RELATED WORK

HPC task scheduling has been a long-time research topic. Countless studies have been done, including heuristic algorithms such as First Come First Serve (FCFS), Shortest Job First (SJF) and more sophisticated policies like WEP3, UNICEF and even machine learning approaches. MARS is clearly different from the existing studies as it takes advantage of existing resource management on HPC systems and it combines the best suitable algorithm to maximize the performance and reduce the training time [18], [19].

Mirhoseini et al. [25], [26] use DRL to optimize placement of computation graph, Xu et al. [27] use the same method to select routing paths between network nodes for traffic, and Mao et al. [28] used the same principle to select video stream rates dynamically.

Recently, several studies also started to leverage deep reinforcement learning in resource allocation and job scheduling in a distributed environment, such as DeepRM [29], and Decima

[20]. However, none of these uses existing HPC resource management and combines the heuristic algorithm with deep reinforcement learning.

Although they used similar DRL methods as MARS, these studies are not designed for scheduling HPC tasks, which are fixed, rigid, and non-preemptable.

These differences led to different designs and optimizations in MARS, detailed in Section III-A. The most recent HPC tasks scheduling [19] uses brute force simulations to generate a large number of data samples, each of which shows the best scheduling decision given a random job sequence. Then, applying machine learning methods on these data samples to build scheduling functions that can best fit these samples.

VII. Conclusion

In this study, we proposed a new cost-aware reinforcement learning policy for task scheduling on HPC systems using the existing resource manager, allowing the system administrators and users to optimize the scheduling of tasks based on any preferred algorithm and cost-effectiveness. We showed that using MARS, which combines heuristic and deep reinforcement learning actor-critic algorithm, HPC systems can be optimized for both legacy and complex workflows. We performed better by choosing different configurations and switching between heuristic and RL-A3C. MARS can improve the modularity and support for both legacy and complex workflows, and it can optimize task execution based on the most appropriate approach.

References

[1] S. Olabarriaga, G. Pierantoni, G. Taffoni, E. Sciacca, M. Jaghoori, V. Korkhov, G. Castelli, C. Vuerli, U. Becciani, E. Carley, and B. Bentley, "Scientific workflow management – for whom?" in *2014 IEEE 10th International Conference on e-Science*, 2014, pp. 298–305.

[2] S. Shumilov, Y. Leng, M. El-Gayyar, and A. B. Cremers, "Distributed scientific workflow management for data-intensive applications," in *2008 12th IEEE International Workshop on Future Trends of Distributed Computing Systems*, Oct 2008, pp. 65–73.

[3] J. Zhang, P. Votava, T. J. Lee, O. Chu, C. Li, D. Liu, K. Liu, N. Xin, and R. Nemani, "Bridging vistrails scientific workflow management system to high performance computing," in *2013 IEEE Ninth World Congress on Services*, June 2013, pp. 29–36.

[4] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, June 2001.

[5] G. Y. and H. M., "Evaluating scalability and efficiency of the resource and job management system on large hpc clusters," in *Job Scheduling Strategies for Parallel Processing*. Berlin, Heidelberg, Germany: Springer, Berlin, Heidelberg, 2013, pp. 134–156. [Online]. Available: https://doi.org/10.1145/3005745.3005750

[6] J. Chen, Q. Guan, Z. Zhang, X. Liang, L. Vernon, A. McPherson, L. Lo, P. Grubel, T. Randles, Z. Chen, and J. Ahrens, "Beeflow: A workflow management system for in situ processing across hpc and cloud systems," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1029–1038.

[7] D. Zhang, D. Dai, Y. He, and F. S. Bao, "RLScheduler: Learn to Schedule HPC Batch Jobs Using Deep Reinforcement Learning," *arXiv e-prints*, p. arXiv:1910.08925, Oct. 2019.

[8] S. Raychaudhuri, "Introduction to monte carlo simulation," in *2008 Winter Simulation Conference*, 2008, pp. 91–100.

[9] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," *arXiv e-prints*, p. arXiv:1502.05477, Feb. 2015.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv e-prints*, p. arXiv:1707.06347, Jul. 2017.

[11] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM J. Control Optim.*, vol. 42, no. 4, p. 1143–1166, Apr. 2003. [Online]. Available: https://doi.org/10.1137/S0363012901385691

[12] J. Li, X. Li, and D. He, "A directed acyclic graph network combined with cnn and lstm for remaining useful life prediction," *IEEE Access*, vol. 7, pp. 75 464–75 475, 2019.

[13] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "Graphene: Packing and dependency-aware scheduling for data-parallel clusters," in *n Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*., 2016, p. 81–97.

[14] R. Kramer, R. Gupta, and M. L. Soffa, "The combining dag: a technique for parallel data flow analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 8, pp. 805–813, 1994.

[15] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[16] "Gym openai," in *https://gym.openai.com/*, 2020.

[17] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

[18] W. Tang, Z. Lan, N. Desai, and D. Buettner, "Fault-aware, utility-based job scheduling on blue, gene/p systems," in *2009 IEEE International Conference on Cluster Computing and Workshops*, 2009, pp. 1–10.

[19] D. Carastan-Santos and R. Y. de Camargo, "Obtaining dynamic scheduling policies with simulation and machine learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126955

[20] H. Mao, M. Schwarzkopf, S. Bojja Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning Scheduling Algorithms for Data Processing Clusters," *arXiv e-prints*, p. arXiv:1810.01963, Oct. 2018.

[21] D. G. Feitelson, "Metrics for parallel job scheduling and their convergence," in *Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, ser. JSSPP '01. Berlin, Heidelberg: Springer-Verlag, 2001, p. 188–206.

[22] D. Feitelson, L. Rudolph, U. Schwiegelshohn, K. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," vol. 1291, 09 1997.

[23] D. Zotkin and P. J. Keleher, "Job-length estimation and performance in backfilling schedulers," in *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, ser. HPDC '99. USA: IEEE Computer Society, 1999, p. 39.

[24] D. Zotkin and P. Keleher, "Job-length estimation and performance in backfilling schedulers," in *Proceedings. The Eighth International Symposium on High Performance Distributed Computing (Cat. No.99TH8469)*, 1999, pp. 236–243.

[25] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V. Le, and J. Dean, "Hierarchical planning for device placement," 2018. [Online]. Available: https://openreview.net/pdf?id=Hkc-TeZ0W

[26] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device Placement Optimization with Reinforcement Learning," *arXiv e-prints*, p. arXiv:1706.04972, Jun. 2017.

[27] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven Networking: A Deep Reinforcement Learning based Approach," *arXiv e-prints*, p. arXiv:1801.05757, Jan. 2018.

[28] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 197–210. [Online]. Available: https://doi.org/10.1145/3098822.3098843

[29] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 50–56. [Online]. Available: https://doi.org/10.1145/3005745.3005750