High Performance Computing (HPC) Job Scheduler Using Deep Deterministic Policy Gradient (DDPG) Algorithm

Raman Singh

Abstract— Batch jobs continue to dominate today's high-performance computing (HPC) platforms. As a result, to achieve high system efficiency, proper batch job scheduling is critical. Currently existing HPCs use heuristic priority-based functions to prioritize and schedule jobs. However, once created and deployed by experts, such functions are unable to respond to changes in job loads, optimization targets, or system parameters, potentially resulting in system inefficiency. However, the complex HPC systems nowadays which are combined with varied loads make the manual process difficult, time-consuming, and prone to human error. This approach is to use the DDPG algorithm, an automated HPC batch work scheduler based on reinforcement learning is presented to address the fundamental issue. This scheduler approach has used multiple knowledge from different sources, yet it can try to learn quality scheduling policies through trial and error methods. To optimize and stabilize the learning process, we have used a neural network based DDPG algorithm. It can try to learn quality scheduling strategies for varied workloads and optimization goals with a bit less computation cost, according to thorough testing. Furthermore, I tried to demonstrate that the learned models perform consistently even when applied to unknown workloads, making them suitable for usage in production.

Introduction

It is very crucial to efficiently schedule jobs in a High-Performance Computing (HPC) as a single second of delay or improper allocation of the jobs will result in a significant decrease in the performance of the HPCs and will lead to the continued delay in future processes. System administrators have traditionally defined scheduling strategies based on their knowledge of individual systems and workloads. However, with increasingly complex HPC systems and very different application workloads, such a manual engineering and tuning process becomes increasingly impossible.

For scheduling the jobs earlier automated functions relied on a single job attribute, such as submission time (First Come First Server, FCFS) or job duration time (Shortest Job First, SJF), etc. Others used to compute priorities based on multiple job attributes which were part of the table in the input file. In the new era, the researchers proposed to use advanced algorithms, e.g., utility functions or machine learning techniques, to develop priority functions.

A new field of machine learning called reinforcement learning has recently been successfully applied to a variety of decision-making challenges, including games, self-driving automobiles, and autonomous robotics. It is a type of machine learning in which a computer learns to make decisions by interacting with its surroundings and it has been quite successful in achieving the same.

To use the above learning process, I have implemented an algorithm named Deep Deterministic Policy Gradient (DDPG) to find the optimum function which is in the form of policy with the help of actor-critic methodology.

Deep Deterministic Policy Gradient (DDPG) Algorithm

DDPG is an off-policy actor-critic algorithm for continuous control using Deep-RL that can be considered a successful variation of the DPG algorithm. The DDPG policy function is based on a neural network-based policy and value function with belongs to the actor and critic respectively. The basic architecture of the algorithm includes each interaction thread's created trajectories that are cached in its cache buffer. These buffers are thread-agnostic and aren't used for direct experience replay. For the real experience replay, we designed two separate memory buffers: memory. According to the storing rule, trajectories cached in the cache buffers will be placed in one of the two memory buffers, namely memory. According to the sampling rule, the trajectories stored in memory are sampled to be used for policy updating as shown in Fig.1. The DDPG network structure, which is divided into two parts: the actor-network and the

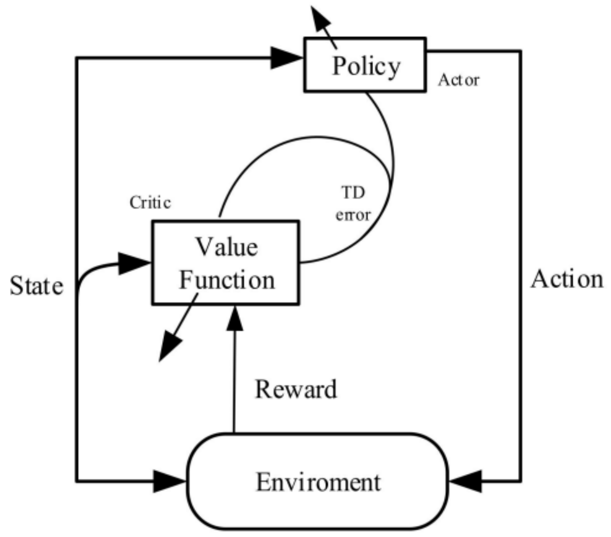

Fig. 1. Architecture of deep deterministic policy gradient (DDPG).

critic network. To approximate the policy function(s) and the state-action value function Q (s,a), DDPG employs the actor-network $\mu(s|\theta^A)$ and the critic network $Q(s,a|\theta^C)$. The actor-network and critic network weights are θ^A and θ^C, respectively. The primary idea is for the actor network to generate the action, and the critic network to evaluate the action using the state-action value function, then direct the update of its own network and actor-network weights through the evaluation.

Environment Setup

To train and test the algorithm I have used the standard workload format (SWF) which is created to decrease the use of workload logs and models. The logs are downloaded from the website, https://www.cs.huji.ac.il/labs/parallel/workload/logs.html. With the help of these files, we can analyze the workloads or simulate system scheduling using programs that can use only a single format and can be applied at multiple places. The input file consists of 18 parameters which are mainly job_id, submit time(sec), wait time(sec), run time, processor allocation, CPU used, memory used, processor requirement, user estimation, memory requirement, status, user id,

group id, execution number, queue number, partition number, previous job and think time. So, in order to perform the scheduling of the jobs we have used 2 parameters from the log file that are wait time and run time. But for the whole process, the algorithm uses wait time, run time, a predefined reward function, and machine states. For each process, the cost function is defined concerning the time which is explained in detail later.
The reward for scheduling the task when it is in the queue is defined as shown below:

reward = ∑ (-Normalized (run time) * max ((1, scheduled time - submit time + run time) / max (run time,10)))/ (no. of batch jobs)

The machine states try to refer the machine which will be free, or which must be allocated since the scheduler will depend on the number of machines available, and if it is busy the scheduler must wait till the next is free.
I have used backfilling which is a simple procedure when a job is picked in a batch queue, the system looks for resources to provision. If the project is a success, resources can be allocated, and work will begin. Otherwise, the work will hang around until its request is met. In the meantime, backfilling can be used to find jobs with resource allocations that can be satisfied immediately without compromising the planned execution of the waiting work, hence increasing system efficiency.

Procedure

The workflow for the procedure used in the algorithm is shown as below:

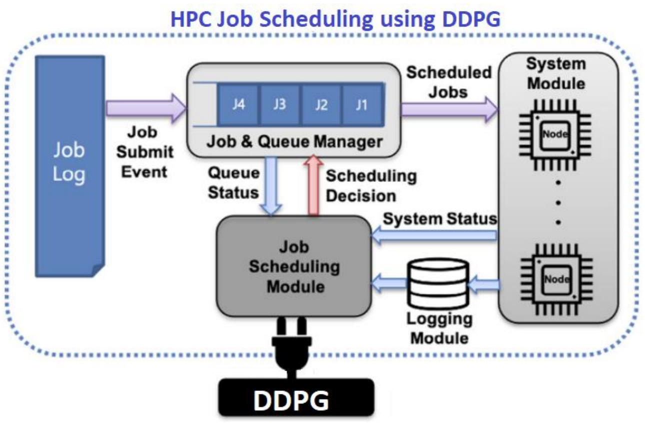

Fig.2. HPC Scheduling using DDPG Algorithm

In the process firstly the log file, an average log file consists of 100,000 job records on an average. Then the job log is submitted to the job and queue manager which is linked to the job scheduling module. There are various ways in which the jobs are scheduled in the job scheduling module like first come first serve, shortest job first, etc. Here we have used DDPG to plan and schedule the jobs according to the new updated policy. The scheduling decision is sent back to the manner and scheduled jobs are sent to the system for processing. The system status and logging are also sent back to the scheduling module which is DDPG here and learns from the system status.
In the DDPG algorithm, only 3 attributes are used from the log file for each job that is run time, scheduled time, and submit time. The values of actors and critics optimize the policy and help with the results. The environment has been changed to the gym for convenience as it's easy to iterate the action and sample space.

Results and Conclusion

The results obtained from the experiment is quite promising which can be seen from the graph below:

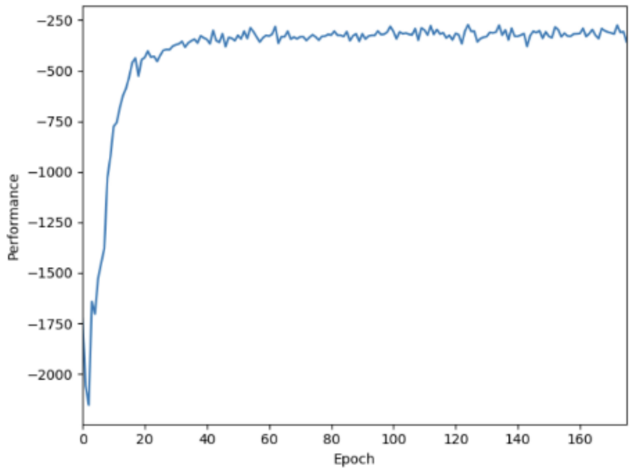

Fig.3. Performance result obtained from DDPG

After multiple trials and tuning the hyperparameters for the actor and critic, finally I obtained the probable optimised result as can be observed from the performance. The reward function also had been optimised seeing the results from the graphs repeatedly. As observed from the graph the epoch stabilizes near 20 epochs.

The other results from the experiment at a random step are as follows:

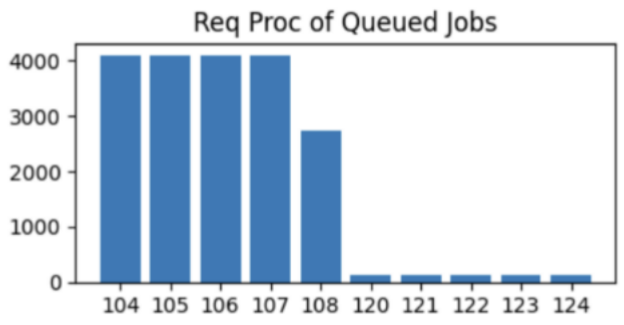

Fig.4. Required Processors of Queued Jobs at random iteration

The required processor for the queued jobs is high in the start as since the number of jobs scheduled to run are more as it decreases the number of required processors also decrease as shown in Fig.4.

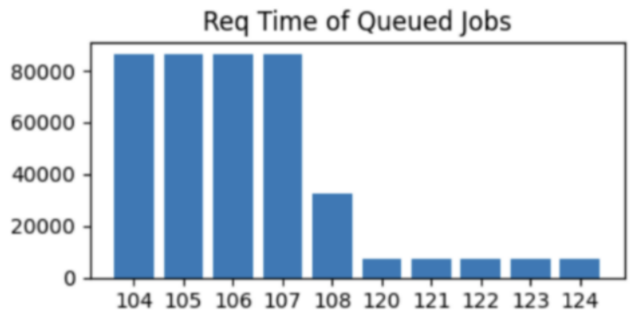

Fig.5. Required time of Queued Jobs at random iteration

The same pattern can be observed from the above plot in Fig.5. The required time for the queued jobs is higher in the start and decreases gradually because the processes will be free and will execute the rest of the tasks.

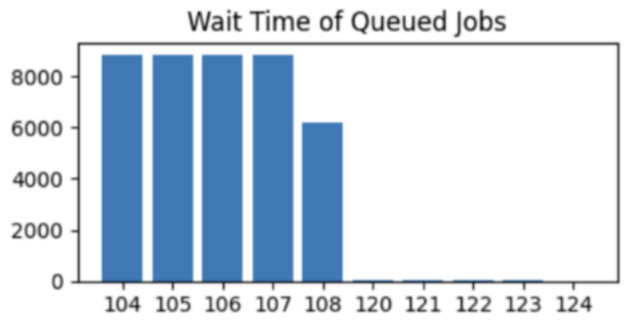

Fig.6. Wait time of Queued Jobs at random iteration

The wait time decrease drastically after a certain number of iterations because the jobs consuming more time will be executed first since they will computationally heavy. The jobs in the last will be of less time consuming and will be less computationally expensive.

FUTURE WORK

We can future try to optimize the reward function and the hyperparameters to optimize the results. The performance of DDPG can be compared to shortest job first (SJF), first come first serve (FCFS), DDQN, WFP3, UNICEP, longest job first (LJF), dynP. Furthermore, we can analyse the without the backfilling feature in the code and compare the results for research purposes. One more iteration of DDPG as DDPG LSTM can be tried.
We can also try with the multi-agent scheduler working in parallel and try multi-agent reinforcement learning algorithms.

ACKNOWLEDGEMENT

I would like to thank Professor Roni Khardon for providing me the opportunity to perform this project under his sincere guidance. He provided us with invaluable advice and helped us in difficult periods. His motivation and help contributed tremendously to the successful completion of the project.

At last, but not in least, we would like to thank everyone who helped and motivated us to work on this project.

REFERENCES

[1] Julita and Marco. Modular Workload Format: extending SWF for modular systems, 15:2513–2519, August 1980.
[2] Yuping Fan, Zhiling Lan, DRAS-CQSim: A reinforcement learning based framework for HPC cluster scheduling, Volume 8, 2021
[3] Abel Souza, Kristiaan Pelckmans, and Johan Tordsson. 2021. A HPC Co-scheduler with Reinforcement Learning. In Job Scheduling Strategies for Parallel Processing: 24th International Workshop, JSSPP 2021, Virtual Event, May 21, 2021, Revised Selected Papers. Springer-Verlag, Berlin, Heidelberg, 126–148. https://doi.org/10.1007/978-3-030-88224-2_7
[4] D. Zhang, D. Dai, Y. He, F. S. Bao and B. Xie, "RLScheduler: An Automated HPC Batch Job Scheduler Using Reinforcement Learning," SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1-15, doi: 10.1109/SC41405.2020.00035.
[5] Learning 2021, imating images with drawings. In Andrew Glassner, editor, Proceedings of SIGGRAPH '94 https://github.com/anita-hu/TF2-RL/blob/master/DDPG/(Orlando, Florida, July 24–29, 1994),Computer Graphics Proceedings, Annual Conference Series, pages 409–412. ACM SIGGRAPH, ACM Press, July 1994.
[6] Luqin Fan, Jing Zhang, Yu He, Ying Liu, Tao Hu and Heng Zhang, https://github.com/betisb/MARS/Optimal Scheduling of Microgrid Based on Deep Deterministic Policy Gradient and Transfer Learning
[7] PRASHANT GIRIDHAR SHAMBHARKAR, SIDDHANT BHAMBRI, VAIBHAV KUMAR, Multiple Resource Management and Burst Time Prediction using Deep Reinforcement Learning, 2019
[8] Pierre Tassel, Martin Gebser, Konstantin Schekotihin, A Reinforcement Learning Environment for Job-Shop Scheduling, 2021
[9] https://www.cs.huji.ac.il/labs/parallel/workload/logs.html - Sample_Log_Download