Orchestrating a DNN training job using an iScheduler Framework: a use case

SWATHI VALLABHAJOYULA, SANDEEP SATISH BUDHYA, AKANSHA JAIN, MAAZ BAIG, and RAJIV RAMNATH, The Ohio State University, USA

Orchestrating DNN training jobs efficiently on HPC centers such as Ohio Supercomputer Center (OSC), Texas Advanced Computing Center (TACC), and San Diego Supercomputer Center (SDSC) is crucial due to the prevalence of AI-driven workloads. However, managing these workloads effectively requires a deep understanding of available resources, allocation policies, and suitable execution configurations. Current approaches often lead to job interruptions, prolonged wait times, and inefficient resource utilization. To address these challenges, we propose the deployment of an iScheduler framework. This framework aims to automate workflow orchestration for DNN training by estimating resource needs (using existing state-of-art estimation models) and generating an infrastructure-aware execution plan. In this study, we demonstrate the practical application of the iScheduler framework in orchestrating a user-specific DNN training workflow, showcasing its capabilities in optimizing resource allocation and scheduling. This poster dives deeper into the user case and shows all user interactions with iScheduler and the responses.

CCS Concepts: • **Software and its engineering** → *Designing software*; • **Computing methodologies** → **Cost-sensitive learning**.

Additional Key Words and Phrases: AI4CI, AI4OPT, ML, estimation scalability, model, execution time estimation, workflow orchestration

ACM Reference Format:
Swathi Vallabhajoyula, Sandeep Satish Budhya, Akansha Jain, Maaz Baig, and Rajiv Ramnath. 2024. Orchestrating a DNN training job using an iScheduler Framework: a use case. 1, 1 (April 2024), 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 PROBLEM STATEMENT

HPC environments are vital for scientific workloads, but users often struggle to estimate resource needs accurately, leading to job interruptions and inefficiencies. The rise of AI-driven workloads exacerbates this challenge, with HPC centers facing increased demand for GPU-powered clusters. Despite available guidelines, optimizing resource allocation for DNN workloads remains complex. Existing AI-based resource estimation models and schedulers aim to address this, but accessing and integrating them into scheduling tasks is challenging for end-users. This gap motivates our work in introducing an **iScheduler** framework designed to seamlessly integrate AI models for automating scheduling tasks, thereby optimizing DNN workload efficiency and resource utilization.

2 METHODS

For the framework to simulate user interactions and manage the job execution workflow, it needs to address specific queries based on user-defined configurations. For example, if a user intends to train a ResNet50 model on WikiFaces Data with 20k images using the ADAM optimizer for 50 epochs, and desires to refine model accuracy by scaling batch sizes (e.g., 8, 16, 32, 64), the framework should provide solutions for the following questions:

Authors' Contact Information: Swathi Vallabhajoyula; Sandeep Satish Budhya; Akansha Jain; Maaz Baig; Rajiv Ramnath, The Ohio State University, Columbus, Ohio, USA.

Manuscript submitted to ACM

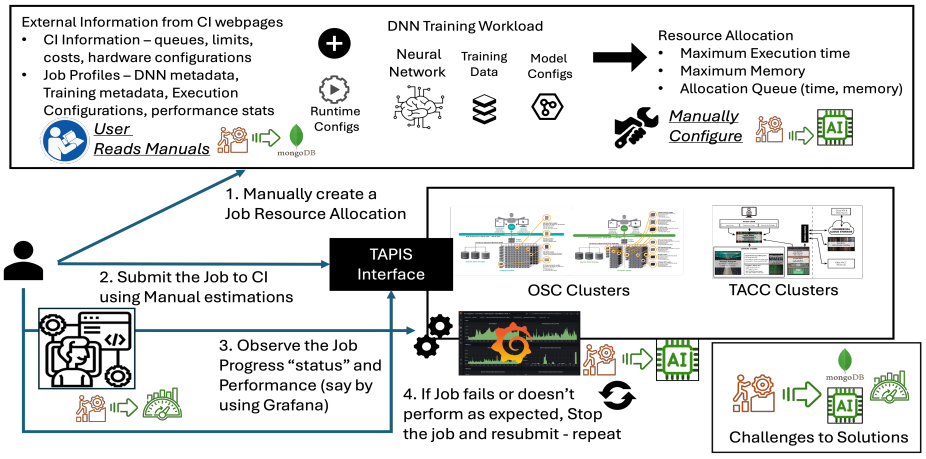

Fig. 1. A: The conventional user interaction workflow is depicted in the figure, highlighting the steps where users encounter challenges and illustrating how an AI-enabled software can address these challenges effectively.

- What are the estimated resource requirements for training ResNet50 with different batch sizes?
- How will the execution time vary for each batch size configuration?
- What are the potential costs associated with training the model using different batch sizes?
- Can the model be trained within specified constraints such as available cluster resources and time limitations? (as shown in Table 1)
- Are there any optimal configurations that effectively balance resource utilization and job completion time?
- Is it feasible to schedule jobs on the immediately available node instead of waiting for a more efficient allocation? Additionally, can we preempt and resume execution on the preferred node once it becomes available? (as shown in Table 1)
- Is it viable to utilize development nodes that become idle after work hours to execute workloads? This could enable jobs waiting for allocations on regular nodes to be executed at least partially before receiving a full allocation)
- Is it feasible to execute DNN training on CPU nodes until GPUs become available? ((as seen from Table 1, CPU-only nodes are currently idle, even though the queue has pending jobs. However, many of these jobs are multiple requests from a few user IDs and may not be allocated due to their maximum CPU core limitations

2.1 Steps for interacting with the job scheduler - iScheduler Framework.

The User imports the iSchedulerHelper tool and creates an instance (TAPIS and DB authentications are set up here).

(1) The User writes his DNN code (create his model, say ResNet50) to generate model summary JSON and prepares the training data (processing Wikifaces data loop and getting the training meta-data).
(2) The User invokes the iSchdulerHelper to request resource estimations for executing his DNN loop against different hyper-parameters.
   (a) The estimator is invoked as shown in formula 1 - which in turn calls the DB [1] to fetch node

[1]DB SChema could be found here: https://lucid.app/lucidchart/bdad353a-cf83-4a3b-b087-0f1807e54c64/edit?viewport$_{loc}$ = −2230%2C − 1555%2C6169%2C2552%2C0$_0$*invitationId* = $inv_4a2655f8 - f613 - 49c3 - b3f8 - ddb5c38ee8aa$

Table 1. An overview of the available system state at two times (on and off work hours). ** A/I/T - Allocated/Idle/Total #-40-core nodes. PD-Pending

CI/ Cluster	Partition	Nodes min-max per Job	Alloc Type	Max Time DD-HH:MM:SS	At 11:00:00 CST		At 5:00:00 CST	
					Nodes ** A/I/T	Jobs PD	Nodes ** A/I/T	Jobs PD
TACC/ lonestar6	development	1-4	CPU	2:00:00	17/1/18	8	15/3/18	0
	gpu-a100	1-4	GPU	48:00:00	65/8/73	76	69/4/73	72
	gpu-a100-dev	1-4	GPU	2:00:00	4/0/0	7	0/4/4	0
	gpu-a100-small	1-1	GPU	48:00:00	24/0/24	6	22/2/24	0
OSC/ ascend	debug	1-4	GPU	1:00:00	23/1/24	0	24/0/24	0
	gpu	1-2	GPU	7-00:00:00	23/1/24	55	24/0/24	50
TACC/ frontera	development	1-40	CPU	2:00:00	371/23/394	5	225/167/394	0
	normal	3-512	CPU	48:00:00	6194/1586/8008	1162	7649/123/8008	1184
	rtx-dev	1-2	GPU	2:00:00	5/1/6	1	0/6/6	0
	small	1-2	GPU	48:00:00	208/8/216	135	208/7/216	113
OSC/ pitzer#	serial	1-1	CPU	7-00:00:00	99/55/164	2249	122/42/164	2246
	parallel	2-40	CPU	96:00:00	99/55/164	6	122/42/164	0
	gpuserial	1-1	GPU	7-00:00:00	11/3/14	0	13/1/14	2
	gpuparallel	2-10	GPU	96:00:00	11/3/14	1	13/1/14	1

(b) The node configuration, user model, and training configs are passed to the memory and walltime estimators.
(c) We invoke the Memory and wall-time algorithms—and submit one or more TAPIS apps (controllers registered as applications) based on how the resellers configure their modules.
(d) Wait for the estimators to execute, consolidate the results, and provide users results like in Table 2

(3) The User invokes an iSchduler function to fetch the feasible execution systems for the desired memory and walltime requirements.
(a) The estimator is invoked as shown in formula 2 - which in turn calls the DBto fetch system queues
(b) The systems are validated against the feasibility of executions and respect execution costs. The availability (idle nodes available on the queue) is computed, and the list is returned to the User.
(c) The User selects a configuration that suits their requirements or offloads the execution to the iScheduler (IP).
(d) If the job is offloaded to IP with a scheduling option such as "immediate availability," it is scheduled to be executed on that queue even if better (high compute) queues are recommended but are unavailable at the moment.
(e) As the job executes, it pushes the progress to the Kafka broker, and a FLASK consumer checks (a) the progress is as planned and (b) If the "Ideal" queue is now available.
(f) The IP terminates the job if a better node becomes available and continues training from the last checkpoint location.
(g) The IP monitors the job and compares progress left with allocation left to either reevaluate resources and reset the job or continue moving the jobs. Table 2 shows the two examples of job orchestration based on resource requirements and node availability.

The User invokes an iSchduler function to fetch the feasible execution systems for the desired memory and walltime requirements.

$$Memeory, Walltime = Estimator(Features[DNN\_Arch, Training\_Data, DNN\_HypParams, Systems] \quad (1)$$

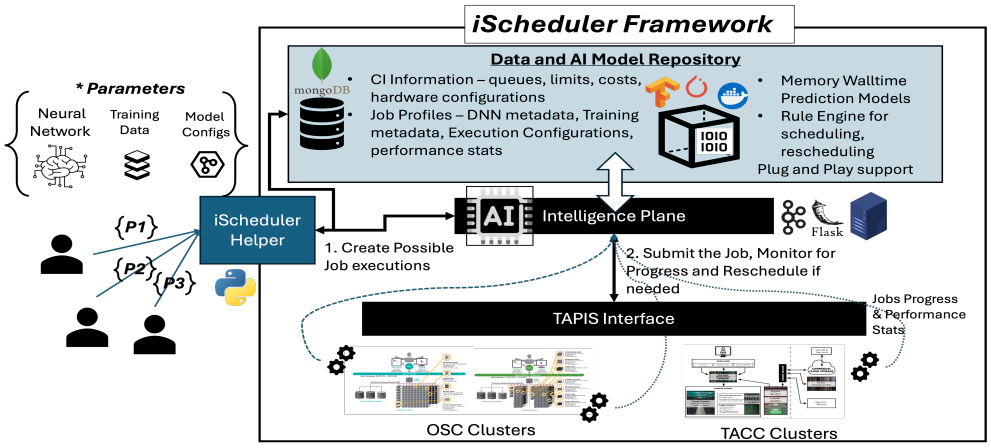

Fig. 2. The proposed iScheduler Framework with a use case of estimate resource requirements for executing a DNN application

Table 2. The Estimation Results from executing the iScheduler Functions - Equation 1 and 2

Training	Batch	Mem (GB)	System	Posible	Est. (Hrs)	Feasible Q	Idle?	Cost ($)
20000	8	7.02	Intel Xeon	yes	11	serial	yes	0.924
20000	16	14.05	Intel Xeon	yes	10	serial	yes	0.84
20000	32	28.11	Intel Xeon	yes	9.2	serial	yes	0.7728
20000	64	56.23	Intel Xeon	yes	8.7	serial	yes	0.7308
20000	8	7.02	NVIDIA V100	yes	3	gpuserial	no	0.522
20000	16	14.05	NVIDIA V100	yes	3	gpuserial	no	0.522
20000	32	28.11	NVIDIA V100	yes	2.4	gpuserial	no	0.4176
20000	64	56.23	NVIDIA V100	no	N/A	N/A	no	N/A

		Allocation Config		Execution Configutation:				
	Queue Name	Training Samples		Waitime (hrs)		Cost ($)	Total(hrs)	
Without Preemption:	serial	20000	32	0	8.4	1.764	8.4	
	gpuserial	20000	32	26	2	0.42	28	
With Preemption	serial + gpuserial	20000	32	0 + 4 <X>	4 + 1.4	0.84 + 0.294 = 1.134	6.4	

$$Waitime, Cost = Estimator(Memory, Walltime, System\_Queues) \quad (2)$$

3 RESULTS

Table 2 displays the outcomes of the user workflow. It's observed that although a slower node (serial) was allocated, the cost of execution in non-preemption mode is higher due to the longer allocation time and the node not being ideal for DNN training. However, the user's job was executed even before an Ideal Node became available, thereby reducing the waiting time. Similarly, in preemptive mode, as soon as the ideal nodes become available, the running job is terminated, and the model continues training from the last checkpoint in the new allocation.