Reference Implementation of Smart Scheduler: A CI-Aware, AI-Driven Scheduling Framework for HPC Workloads

Swathi Vallabhajosyula
vallabhajosyula.2@buckeyemail.osu.edu
The Ohio State University
Columbus, Ohio, USA

Sandeep Satish Budhya
sandeepsbudhya@protonmail.com
The Ohio State University
Columbus, Ohio, USA

Rajiv Ramnath
ramnath.6@osu.edu
The Ohio State University
Columbus, Ohio, USA

ABSTRACT

Many modern scientific workloads in HPC centers rely heavily on AI-driven tasks, particularly deep neural network (DNN) training workloads. Efficiently managing and scheduling these workloads via SLURM interfaces requires users to comprehensively understand available resources, allocation policies, and suitable execution configurations aligned with their models' estimated resource requirements and constraints. Typically, scheduling jobs involves using default configurations, adjusting them as needed, or requesting maximum available limits to ensure uninterrupted execution. However, this approach can lead to job interruptions due to underprovisioning, prolonged wait times, inefficient resource utilization, and increased costs from overprovisioning. These issues ultimately degrade cluster performance, emphasizing the need for a more efficient solution like an AI-enabled Scheduler framework that can profile the DNN workloads and estimate and provision resources dynamically. The existing resource estimation models are trained independently to predict various aspects of batch processing and scheduling, which do not work cohesively to orchestrate a job execution. In our work, we propose to introduce a framework that investigates the feasibility of implementing an iScheduler framework, which transforms the traditional SLURM resource provisioning workflow into an AI-enabled scheduler that plugs different estimators where needed to orchestrate workflow by generating a cyberinfrastructure-aware execution plan, schedules and monitors jobs till completion. We demonstrate the feasibility of our framework by orchestrating a user-specific DNN training workload.

CCS CONCEPTS

• **Computing methodologies** → *Planning under uncertainty*; Model verification and validation; • **Software and its engineering** → **Software design engineering**.

KEYWORDS

AI4CI, AI4OPT, ML, estimation scalability, model, execution time estimation, workflow orchestration, job scheduling

PEARC '24, July 21–25, 2024, Providence, RI, USA

ACM ISBN 979-8-4007-0419-2/24/07
https://doi.org/10.1145/3626203.3670555

ACM Reference Format:
Swathi Vallabhajosyula, Sandeep Satish Budhya, and Rajiv Ramnath. 2024. Reference Implementation of Smart Scheduler: A CI-Aware, AI-Driven Scheduling Framework for HPC Workloads. In *Practice and Experience in Advanced Research Computing (PEARC '24), July 21–25, 2024, Providence, RI, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3626203.3670555

1 INTRODUCTION

High-performance computing (HPC) environments have become indispensable for handling various scientific workloads, where resources are shared and allocated using batch scheduling based on user requests. Users typically rely on their experience or recommended defaults from Cyberinfrastructure (CI) to estimate resource needs. They aim to secure resources that enable job completion with minimal wait times, interruptions, and, optionally, adjustments to allocation costs.

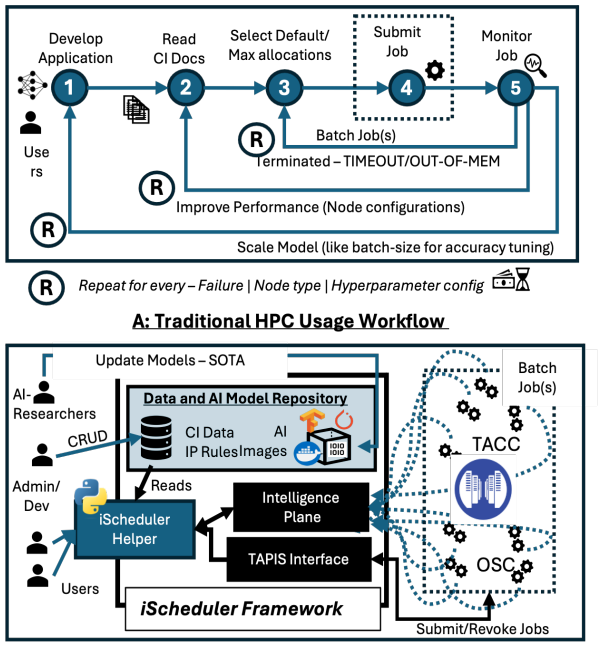

Figure 1: A: The Conventional user-HPC interaction for job scheduling; B: The Proposed iScheduler Framework

With the increasing popularity of Deep Neural Network (DNN) models and a huge shift towards Machine Learning (ML) and DNN-driven scientific research is observed and these workflows span from training novel models from scratch to fine-tuning state-of-the-art architectures. This shift has prompted HPC centers to deploy

more GPU-powered clusters to accommodate the influx of AI workloads.[1]. HPC centers have seen a splurge in resource demands as more researchers are initiating AI-driven approaches in their workflows, which are heavily computing-dependent and expensive.

Efficient allocation of computational resources is crucial for DNN training and inference on cloud/HPC. Researchers must comprehensively understand their current workloads to ensure optimal resource requests. This guarantees a fair distribution of resources amongst researchers and promotes efficient utilization of shared resources. A wealth of CI documents, articles, and blogs offer best practices for resource allocation for DNN workloads, considering factors such as batch and epoch scaling, choice of optimizers, and memory requirements. Furthermore, research initiatives have been undertaken to study and understand AI workloads, improve resource allocation, and develop new hardware architectures/chips for compute-intensive AI training, inferencing, and generative AI modeling. Datasets like The MIT Supercloud Dataset [2] have been created to facilitate research on workload profiling and estimating resource requirements, particularly for deep neural network (DNN) workloads.

To enhance the efficiency of DNN workloads against the available resources, significant advancements have been made in the development of AI-based resource estimation models and Reinforcement Learning (RL) based schedulers. These models estimate DNN memory requirements and training times, profile user workloads, and analyze and scale DNN jobs online. (See section 2 for related literature).

However, accessing these models remains challenging for end-users who request resources and schedule jobs due to the absence of reproducible code or ready-to-use components. To our knowledge, no framework is available to seamlessly integrate these models for automating end-to-end job orchestration (from development to execution). Our work aims to bridge this gap by introducing a smart-scheduler framework - **iScheduler**. This framework orchestrates job execution by leveraging various AI models designed to allocate resources and monitor jobs via helper functions and API calls (to interact with HPC centers).

The iScheduler Framework (as shown in Figure 1B) offers the following features to orchestrate the job execution[3]:

- Provides an end-user helper tool, a iScheduler Helper Python module, to interact with cloud-hosted estimators for fetching resource requirements and orchestrating job submissions.
- Utilizes a database (DB) to store system information and policies, aiding predictions and decision-making before job submission.[4]
- Returns list of feasible predictions (memory, walltime, queueing times), execution status, and cost feasibility against available systems to end users, enabling them to either submit jobs manually or delegate them (via Intelligence Plane).
- Includes an Intelligence Plane (IP) Service for job submission, monitoring, and tracking on behalf of the user (using TAPIS APIs [4])
- Supports plug-and-play upgrades to AI estimators (deployed as container images and executed as TAPIS apps).

We elaborate the framework components in Section 3 and illustrate the workflow for training a vision model in Section 4. We execute estimators and run the target training workload as TAPIS Jobs by registering them as applications. While certain components have broad applicability (like the Database, which is applicable for any HPC job), we focus on prototyping a workflow for training DNN models for three main reasons: a) the increasing adoption of neural networks in workloads, b) the deterministic nature of DNN architecture implementation, facilitating the development of AI-based estimators with wider applicability and reducing the need for individual workload profiling, and c) the simplicity of checkpointing and re-training models, enabling dynamic scaling and reallocation.

The code for all components can be found on GitHub [5]

2 BACKGROUND

Conventional User Job Execution Life-cycle: A user interacts with HPC environments (depicted in Figure 1, A) by developing their applications, consulting CI documentation for suitable resources, submitting batch allocation requests, and monitoring job progress. If a job fails, users diagnose issues and analyze runtime requirements before resubmitting. This iterative process, known as online analytics, aims to address submission errors, enhance performance, and experiment with parameters for scalability and accuracy improvement. ***However, it is both costly and time-consuming.***

2.1 Existing Research

Various AI-driven models have emerged for enhancing scheduling estimations, each addressing different job allocation and execution aspects. For instance, DNNMem[2] focuses on estimating memory requirements, while TPUGraphs[3] predict training times per epoch. Frameworks like HARP[7] conduct offline profiling of DNN training loops to understand resource needs. Conversely, models like Scavenger[5] perform online profiling, adjusting allocations based on consumption. Some models predict job start times, while others, like Mirage[1], use reinforcement learning for scheduling GPU jobs. Despite advancements, these models exhibit some estimation errors and still require user assistance in resource allocation when used independently. For example, our walltime estimator for DNNs has a 20% error rate, which can either lead to job failure or longer waittimes. While models like Scavenger offer better accuracy (4-20% errors), they can only profile jobs while executing them, so they cannot pre-compare executions across different architectures.

In our previous research, HARP[6, 7], demonstrated offline profiling's viability for user-centric workflows and developed application-specific AI-based walltime estimators. We compared default job configurations with our estimators' predictions, assessing feasibility and cost-effectiveness. HARP involved: 1) pre-profiling user workloads against predefined configurations to generate training data; 2) training off-the-shelf estimators, dynamically selecting suitable models based on validation accuracies; and 3) configuring of CI

[1]Clusters like 'cardinal' at OSC are deployed, and chipsets like NVIDIA Blackwell are being released

[2]https://registry.opendata.aws/dcc/

[3]The entire iScheduler framework (from Figure 1B) can be configured on one server (or node), and the users need to deploy only the iScheduler Helper python module on their work environment

[4]We used the CI documentation and SLURM commands to create the DB

[5]https://github.com/manikyaswathi/iSchedulerFramework

Table 1: Components of iScheduler Framework with software requirements, implementation details, and functionalities

iScheduler Helper tool **S/W:** Python (Compatible with TensorFlow and PyTorch) **Actors:** - Developers (Create functions to add workflows) - HPC Users (Utilize functions to invoke inference models) **Functionality:** - It offers an interface for end users to interact with the iScheduler via API calls. - Users configure their access tokens (TAPIS/DB) to interact with DB, IP and to submit jobs.
CI Configurations (Local or Cloud Server) [Database] **S/W:** MongoDB (Chosen for its lightweight and NoSQL, nature ideal for prototyping, flexible to schema changes) **Actors:** - Developers/Admins (Configure and manage the DB), - Automated Script (Updates the DB with CI changes), - HPC Users (Interact with the DB using the iScheduler) **Functionality:** The DB stores CI details to estimate resource needs against node configs, validating allocations against batch limitations and execution costs against allocations.
Intelligence Plane (IP) **S/W:** - Kafka (Kafka is used to capture the profiling and progress status of the executing jobs for online analytics), - Flask (The consumers of the monitoring data that invoke IP rules and make appropriate decisions.) **Actors:** - Developers (Add more IP features), - HPC Users (Interact with IP using the iScheduler Helper.) **Functionality:** - Submits a TAPIS job on behalf of the user by creating a job hook. - Maintains independent daemons for tracking job progress and making ad-hoc decisions for rescheduling. - The current prototype supports rescheduling jobs based on progress and remaining allocation time
AI Models (Local or Cloud Server) **S/W:** Images (Docker/Apptainer) (facilitates on-the-fly inferencing deployment, seamless integration with TAPIS , and supports plug-and-play for developing new models) **Actors:** - Developers/Admins: (Create iScheduler helper functions to add new workflow estimation models), - HPC Users (Invoke these models via the helper functions). **Functionality:** Replaces human estimations and incorporates existing work (online and offline analytic models) that could be invoked based on need in the workflow at appropriate times
The Framework uses **TAPIS** to invoke AI models and automate the Job execution. It is a cloud-hosted API configuring systems on multiple CIs, and applications for job scheduling.

policies to estimate predictions' cost and feasibility. ***However, users needed to run all three steps once to create and store estimators locally, requiring local HARP instances and familiarity with its modules.***

2.2 Opportunities for HPC Job Execution Workflow Enhancement

Current Bottlenecks: 1: Manual fine-tuning is time-consuming, expensive, and resource-intensive, as depicted in Figure 1A.; 2: Independent use of current AI-models still needs more accurate resource requirement estimation, hindering end-to-end job automation.; 3: Utilizing available queues rather than waiting for ideal configurations demands diligent job monitoring and frequent rescheduling. For example, development queues offer similar computing capabilities to longer allocation queues but with limited allocation time (2 hours), requiring users to fragment their execution and submit multiple jobs [6].

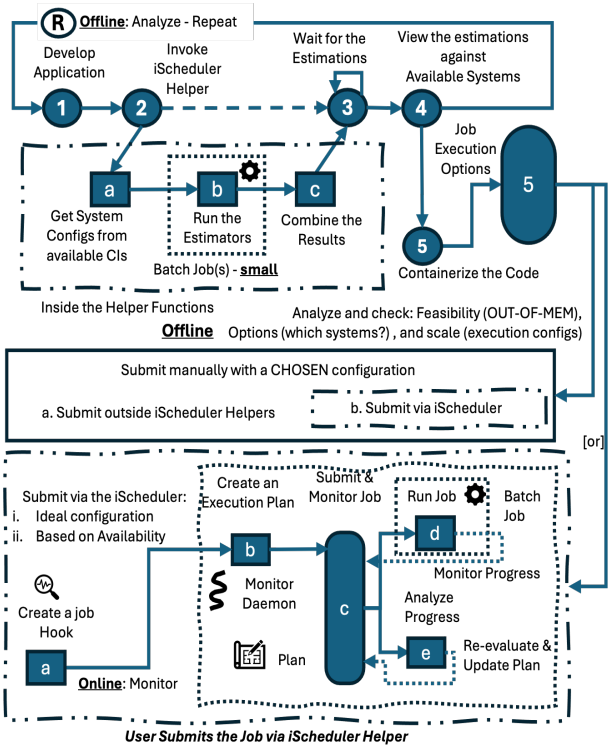

Figure 2: The user interaction flow with the iScheduler

The proposed framework (Figure 1B) seamlessly integrates plug-and-play resource estimators by leveraging CI awareness through a centralized database for end-to-end job execution. Key components of our architecture are (a) **CI database:** Stores all CI policies and aids in Intelligence Plane decisions. (b) **Model repository:** Hosts AI estimator models. (c) **TAPIS Interface:** Facilitates job scheduling across diverse CIs and clusters via APIs. (d) **Intelligence Plane (IP):** Manages job scheduling, monitors execution progress, and adjusts schedules as needed. Additionally, our iScheduler Helper module offers various functions for users to invoke suitable AI models for estimations based on their configurations. The subsequent sections discuss further details on these components and an example workflow.

3 THE ISCHEDULER COMPONENTS

Table 1 describes different iScheduler components as shown in Figure 1B, including their objectives, implementation details, and rationale behind the selected software stack. The current prototype is designed to facilitate DNN vision model training, and the software stack choice aims to complement other ICICLE[7] modules, such

[6]Refer to the cluster batch limitations outlined in the respective CI documentation.
[7]https://icicle.osu.edu/

as cloud-to-edge orchestration. Figure 2 shows the job execution orchestration workflow using iScheduler.

4 ISCHEDULER WORKFLOW WITH A DNN TRAINING JOB EXECUTION SCENARIO

Figure 2 illustrates the user interaction flow with the iScheduler Helper module to ascertain their job's resource requirements against available systems. It shows the orchestration of job execution with the Intelligence Plane.

Use Case: A user trains a ResNet50 vision model with 20k images and 50 epochs to improve accuracy by testing various batch sizes. They utilize the iScheduler helper to manage resource requirements efficiently. Workloads are single-node allocations managed through TAPIS, with in application checkpointing and callbacks ensuring job continuity and progress monitoring.

Workflow: The workflow between the user and the Smart Scheduler is as follows:

- After configuring the DNN architecture, the user invokes the appropriate iScheduler Helper function, providing model details and desired batch sizes (32, 64).
- The Helper Function retrieves node configurations (CPU and GPU note configs for Pitzer Cluster) from the CI database and runs estimators to provide resource predictions.
- The user reviews the estimations and selects an execution plan, either manually submitting the job or delegating it to the Intelligence Plane. Result: A Pitzer CPU node can execute ResNet50 for all batch sizes with maximum execution times of 9.2 and 9.7 hours and corresponding costs of $0.7728 and $0.7308 for batch sizes 32 and 64, respectively. Additionally, Pitzer GPU (with a maximum of 32 GB memory) can handle only batch size 32 with an estimated time of 2.4 hours and costs of $0.4176.
- The user has two options after reviewing the estimation results: a) Manual Submission, where they configure a SLURM script to submit the job themselves, and b) Delegating the job execution and monitoring to the Intelligence Plane. The prototype demonstrates two feasibilities - submitting with the best configurations (ideal system configurations for minimal interruptions) and based on system availability (CI queue state: number of idle nodes, number of waiting jobs).
- Containerize the code, publish it to a publicly accessible location, and create a TAPIS Application.
- The Intelligence Plane manages job execution, submitting, and monitoring progress through callbacks. Jobs may run as single tasks or cascading ones if exceeding maximum walltime. It reassesses plans after each sub-job, adjusting or rescheduling as needed, triggering AI-finetuning loops for memory and walltime estimations. For instance, submitting a job (batch size 32) with an ideal configuration (Pitzer GPU) has an estimated job completion of 28 hours (26 hours wait time + 2 hours execution) and $0.348 cost. Alternatively, executing based on availability yields an estimated job completion of 9 hours (no waiting time as resources are allocated immediately) with $0.756 cost.

Given the wait time for GPU node allocation, the current application (ResNet50) could be completed before the GPU allocation, potentially saving researchers time.

5 CONCLUSION AND FUTURE WORK

Offloading the job execution-monitoring loop to the IP enables the utilization of various estimation models, including offline profilers, estimated allocations based on cluster queue information, and online profiles with predefined rules for job execution. Opting for availability allows users to execute their jobs partially or entirely on less preferred nodes until a desired allocation becomes available, reducing overall execution time, including wait times. Development queues, with identical node configurations as full queues, are typically idle at night, making them ideal for utilization with the scheduler framework without requiring manual monitoring of short cascading jobs. Employing an intelligent scheduler can enhance the productivity of such compute-intensive workflows and diminish the human-in-the-loop effort.

Our future work will refine the current prototype to support DNN workflow orchestration, collaborating closely with domain experts. Estimating wait times is challenging, and currently, we're using basic SLURM commands to identify queues with idle nodes to determine immediate availability. We aim to implement approaches from existing papers for a more refined and accurate approach. This involves validating and designing each component, such as ensuring the CI database mirrors the SLURM DB, requiring collaboration with CI Admins for effective development. We want to conduct a load test on the scheduler server by submitting at least 10,000 jobs to assess scalability and service viability.

Funding and Collaboration: *This work was partially supported by the National Science Foundation and the NSF AI Institute for Intelligent Cyberinfrastructure with Computational Learning in the Environment(ICICLE) under grant agreements OAC-1945347 and OAC-2112606. We acknowledge Dr. Joe Stubbs and Dr. Richard Cardone from the Texas Advanced Computing Center (TACC) for their invaluable guidance and feedback in establishing the framework.*

REFERENCES

[1] Qiyang Ding, Pengfei Zheng, Shreyas Kudari, Shivaram Venkataraman, and Zhao Zhang. 2023. Mirage: Towards Low-interruption Services on Batch GPU Clusters with Reinforcement Learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13.

[2] Yanjie Gao, Yu Liu, Hongyu Zhang, Zhengxian Li, Yonghao Zhu, Haoxiang Lin, and Mao Yang. 2020. Estimating gpu memory consumption of deep learning models. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1342–1352.

[3] Mangpo Phothilimthana, Sami Abu-El-Haija, Kaidi Cao, Bahare Fatemi, Michael Burrows, Charith Mendis, and Bryan Perozzi. 2024. TpuGraphs: A Performance Prediction Dataset on Large Tensor Computational Graphs. *Advances in Neural Information Processing Systems* 36 (2024).

[4] Joe Stubbs, Richard Cardone, Mike Packard, Anagha Jamthe, Smruti Padhy, Steve Terry, Julia Looney, Joseph Meiring, Steve Black, Maytal Dahan, et al. 2021. Tapis: An API platform for reproducible, distributed computational research. In *Advances in Information and Communication: Proceedings of the 2021 Future of Information and Communication Conference (FICC), Volume 1*. Springer, 878–900.

[5] Sahil Tyagi and Prateek Sharma. 2023. Scavenger: A Cloud Service for Optimizing Cost and Performance of ML Training. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 403–413.

[6] Manikya Swathi Vallabhajosyula and Rajiv Ramnath. 2022. Towards Practical, Generalizable Machine-Learning Training Pipelines to Build Regression Models for Predicting Application Resource Needs on HPC Systems. In *Practice and Experience in Advanced Research Computing* (Boston, MA, USA) *(PEARC '22)*. Association for Computing Machinery, New York, NY, USA, Article 43, 5 pages. https://doi.org/10.1145/3491418.3535172

[7] Manikya Swathi Vallabhajosyula and Rajiv Ramnath. 2023. Insights from the HARP Framework: Using an AI-Driven Approach for Efficient Resource Allocation in HPC Scientific Workflows. In *Practice and Experience in Advanced Research Computing*. 341–344.